Facultad de Ciencias

Técnicas de "deep learning" para el análisis de cubos de datos astrofísicos del SKA Science Data Challenge 2

Deep learning techniques applied to the analysis of astrophysical datacubes from the SKA Science Data Challenge 2

Trabajo de Fin de Grado
para acceder al

GRADO EN FÍSICA

Autor: David Girón Ceballos

Director: Diego Herranz Muñoz

Co-Director: Patricio Vielva

Junio-2022

Abstract

Radioastronomy has always been limited by the small resolution of its instruments, mainly due to unavoidable physical reasons. Nevertheless, one of the SKA project's aims is to overcome this difficulty by the construction of the biggest radiotelescope ever created, making use of the technique of interferometry to obtain its images. These images produced by the telescope will have a resolution never seen in any instrument, including the ones located in the outer space. In order to train the community in the use of the large datasets, the project's developers have created a data challenge, consisting in the detection of radio sources in a continuum emission datacube. To solve the challenge, a convolutional neural network has been implemented, based in the U-Net architecture. Finally, the architecture was tested against other preprogrammed architectures, from the keras-unet-collection python package, resulting in none of them achieving a precision bigger than a 10%.

Resumen

Las innumerables ventajas de la observación en ondas de radio han estado siempre limitadas por su gran desventaja, la resolución. Sin embargo, el proyecto SKA pretende deshacerse de este lastre, a través de la construcción del mayor radiotelescopio del mundo, haciendo uso del fenómeno de interferencia aplicado a ondas de radio. Las imágenes producidas por este aparato tendrán una resolución mucho mayor que la de otros telescopios de su tipo, permitiendo la exploración de lugares nunca antes vistos. Sin embargo, el manejo por parte de la comunidad del enorme volumen de datos que tiene previsto generar, requiere de un entrenamiento previo. Dentro de ese entrenamiento, se ha desarrollado un algoritmo capaz de detectar galaxias, utilizando una imagen con su emisión en el continuo de frecuencias. Para este fin, se ha construido una modelo de red neuronal basado en la arquitectura *U-Net*, que finalmente se ha comparado con otros modelos ya programados. A pesar del esfuerzo, ninguna de las arquitecturas ha conseguido una precisión superior al 10 %.

Keywords: redes neuronales convolucionales, segmentación, radioastronomía, SKA, python, catálogos, línea HI, convolutional neural networks, image segmentation, radioastronomy, HI emission line.

Índice

1 Introducción **2**
- 1.1 Radioastronomía: contexto histórico . . . 2
  - 1.1.1 Origen . . . 2
  - 1.1.2 Comparación de la observación en radio frente al visible . . . 2
  - 1.1.3 Línea de 21 centímetros . . . 3
- 1.2 Observatorio Square Kilometer Array . . . 4
- 1.3 Fuentes de radio: núcleos de galaxia activos y galaxias formadoras de estrellas . . . 5
- 1.4 *Data challenges* . . . 7
  - 1.4.1 Formato de los datos del desafío . . . 8
  - 1.4.2 Generación de los cubos de datos . . . 9
  - 1.4.3 Estado actual del desafío . . . 10
- 1.5 Redes neuronales . . . 10
  - 1.5.1 ¿Qué son? Aplicaciones en diversos campos . . . 10
  - 1.5.2 Aplicación de redes neuronales al proyecto SKA. Segmentación de imágenes . . . 12
  - 1.5.3 Redes neuronales convolucionales: Arquitecturas, funciones de activación e hiperparámetros . . . 13
- 1.6 Motivación y objetivos . . . 17

2 Método experimental **19**
- 2.1 *Hardware*: especificaciones del equipo . . . 19
- 2.2 *Software*: *python* y el paquete Anaconda . . . 19
  - 2.2.1 Tensorflow y las GPUs . . . 20
- 2.3 Cubo de datos y catálogo . . . 22
- 2.4 Estructura del programa . . . 25

3 Resultados: código y *outputs* **28**
- 3.1 Código pre-red neuronal . . . 28
  - 3.1.1 Modificaciones previas: división del cubo de datos y solapamiento . . . 29
  - 3.1.2 Creación de las máscaras. Condición de borde de galaxia . . . 32
- 3.2 Código post-red neuronal . . . 33
  - 3.2.1 Arquitectura, hiperparámetros y entrenamiento de la red neuronal . . . 34
  - 3.2.2 Detección post-entrenamiento . . . 35
- 3.3 Resumen del código realizado . . . 37
- 3.4 *SExtractor* para identificar fuentes en la máscara . . . 37
- 3.5 Comparación de diferentes arquitecturas . . . 38
- 3.6 Rendimiento. Comparación de tiempos entre GPU y CPU . . . 43

4 Conclusiones **43**
- 4.1 Valoración de los resultados . . . 43
- 4.2 Opciones de mejora . . . 45
- 4.3 Resultados del aprendizaje . . . 46

5 Apéndice **51**
- 5.1 Página web con documentación: *Read the docs* . . . 51
- 5.2 Instrucciones de uso del código . . . 51

1 Introducción

1.1 Radioastronomía: contexto histórico

1.1.1 Origen

Aunque pueda parecer contraintuitivo, una de las mejores herramientas que tenemos para estudiar los cuerpos celestes se basa en un región del espectro electromagnético que no podemos "ver". Toda la astronomía desarrollada antes del siglo XX se basaba en la pequeña región del espectro que supone el visible, desde los mapeos y catálogos de estrellas de griegos y árabes, hasta las leyes de Galileo y las mediciones del observatorio de Thyco Brahe [1]. Por cuestiones evolutivas (el máximo de emisión del Sol se encuentra en esas longitudes de onda, a lo que se suma la poca opacidad de la atmósfera frente a radiación en el visible), los seres humanos solo pueden percibir radiación (mediante la vista) en un pequeño intervalo de longitudes de onda, entre 350 y 750 nanómetros aproximadamente. Sin embargo, una gran parte de los cuerpos celestes emiten fuera de esas longitudes, o solo emiten una pequeña parte en ellas. Esa fue la razón por la que empezó a desarrollarse la astronomía en la región del ultravioleta, de los rayos X y, la que se va a tratar en esta memoria, ondas de radio.

La radioastronomía nació en la década de 1930, de la mano de Karl Jansky, que fue la primera persona en detectar ondas de radio procedentes del espacio, concretamente del interior de la Vía Láctea. Para ello construyó su propio radiotelescopio, fotografiado en la Fig.1.1. A partir de ahí, la tecnología de detección fue mejorando, facilitando el desarrollo de esta rama de la astrofísica. En 1967, la astrofísica norirlandesa Jocelyn Bell observó un púlsar por primera vez utilizando un radiotelescopio. De hecho, este descubrimiento supuso un premio nobel siete años después, pero ella no figuraba entre los premiados [2]. En la misma década se construyó también el icónico telescopio de Arecibo, en Puerto Rico (Fig.1.2), famoso por su plato de más de 300 metros de diámetro.

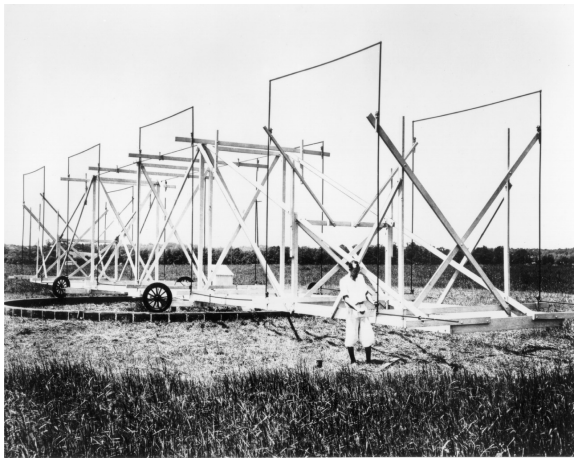

Figura 1.1: Fotografía del primer radiotelescopio, construido por el físico estadounidense Karl Jansky en 1933. [3]

Figura 1.2: Fotografía del telescopio Arecibo, situado en la localidad puertorriqueña del mismo nombre, en 2012, previa al derrumbe del disco del 2020. [4]

1.1.2 Comparación de la observación en radio frente al visible

El propósito de construir telescopios tan grandes era contrarrestar la mayor desventaja de la observación en el espectro de las ondas de radio, su mala resolución angular θ, i.e., la capacidad que tienen distintos telescopios de separar imágenes muy próximas (en radianes) [5]:

$$\theta = 1.22 \cdot \frac{\lambda}{D} \tag{1}$$

donde D es el diámetro del telescopio y λ la longitud de onda de la radiación. Las ondas de radio tienen una longitud de onda del orden de 10^9 veces la longitud de onda del visible. Por lo tanto, es necesario un telescopio con un diámetro mayor para obtener una resolución similar. Sin embargo, construir instrumentos enormes no parece la mejor de las ideas, puesto que supondría un desperdicio enorme de

recursos. Afortunadamente, este problema se solucionó gracias a la interferometría. Los primeros usos del fenómeno óptico de interferencia, en el visible, se atribuyen a Albert Michelson (junto a Edward Morley). Su famoso experimento confirmó, aunque sin saberlo, la teoría de la relatividad especial, que sería formulada unos años después por Albert Einstein. No fue hasta mediados del siglo pasado cuando se desarrolló la interferometría de ondas de radio, orientada a la observación espacial. La interferometría permitió sustituir un gran telescopio por una matriz de pequeños telescopios, separados entre sí. Dicha matriz tenía un diámetro efectivo equivalente a un único telescopio con sus dimensiones. Así, fue posible conseguir resoluciones más potentes que incluso en el visible, gracias al desarrollo de esta tecnología.

Solucionado el problema, fue posible aprovechar todas las ventajas que suponía la observación en ondas de radio. Las observaciones, además de no depender de las condiciones meteorológicas ($\lambda >>$ diámetro gotas de lluvia), permitieron observar objetos que emitían poco o nada en el visible. Además, fue posible estudiar objetos más lejanos, puesto que, por el mismo motivo que las observaciones no se ven afectadas por las nubes, las ondas de radio no son absorbidas ni modificadas por el polvo interestelar [6].

1.1.3 Línea de 21 centímetros

En un principio, la astronomía de ondas de radio se orientó a la observación del continuo de emisión. Como la energía de las transiciones atómicas era del orden de unidades de electrón-voltio, no se esperaba observar ninguna transición que emitiera o absorbiera fotones de una millonésima de electrón-voltio (energía de los fotones de ondas de radio), ni siquiera las transiciones relacionadas con la estructura fina (interacción entre electrón, a través del espín, y el momento angular orbital del propio electrón).

Sin embargo, en 1945 un joven estudiante holandés, Henk Van de Hulst, fue el primero en considerar la posibilidad de que el hidrógeno emitiera radiación de 21 centímetros de longitud de onda. Según describió en su artículo, el hidrógeno neutro emite radiación debido a la transición entre los dos posibles estados de energía de espín de su electrón. A pesar de lo interesante de su propuesta, después de dedicarle apenas un par de párrafos, concluyó que esta línea no se podría observar debido al efecto Stark [7]. Sin embargo, esta desafortunada afirmación se debió a un error de cálculo. La primera observación de este suceso tuvo lugar 6 años después, a cargo del físico estadounidense Harold Ewen.

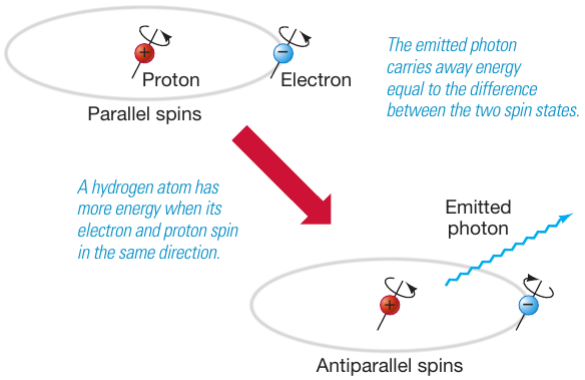

Figura 1.3: Esquema con el proceso de emisión de fotones de 21 centímetros de longitud de onda, para un átomo de hidrógeno. [8]

Actualmente, con todo el desarrollo de la mecánica cuántica realizado durante el siglo pasado, se conoce perfectamente su origen y su magnitud. El origen de esta mínima diferencia de energías reside en la interacción entre núcleo y electrón, concretamente entre sus espines. Este tipo de interacción provoca que el nivel con los dos espines paralelos tenga una energía ligeramente superior al estado con los dos espines antiparalelos, como se muestra en la Fig.1.3. A esta rotura de degeneración se la conoce como estructura hiperfina.

Como la línea de emisión se encuentra en el régimen de ondas de radio, prácticamente no sufre atenuación al propagarse por el espacio. Así, ha sido posible estudiar cualquier región del universo con hidrógeno, por ejemplo estudiando el corrimiento al rojo (o al azul) para estimar la distancia o la velocidad de rotación, entre otras cosas.

A diferencia de las líneas de absorción, que requieren de una fuente de radiación colocada justo detrás de la zona que se quiere estudiar, las líneas de emisión permiten estudiar cualquier zona del universo que emita fotones con la suficiente intensidad.

Pero, ¿por qué no se encuentran todos los átomos en su estado fundamental casi catorce mil millones de años después del origen del universo? La energía de los fotones emitidos viene dada por la relación de Planck:

$$E_{em} = h \cdot \frac{c}{\lambda} \tag{2}$$

donde h es la constante de Planck y c la velocidad de la luz. Por tanto, los fotones emitidos en la transición entre los dos estados de espín tienen una energía de $E_{em} \sim 6 \cdot 10^{-6}$ eV. Por otro lado, la temperatura típica de las nubes de gas hidrógeno es del orden de 100 Kelvin [9]. La temperatura T y la energía por partícula E, se relacionan a través de la constante de Boltzmann k [10]:

$$E \sim kT \tag{3}$$

Según esta ecuación, las partículas con temperatura $T = 100$ K tienen una energía del orden de milielectrón-voltios. Por tanto, la energía disponible en una nube de hidrógeno a 100 Kelvin es mucho mayor que la diferencia de energía entre los dos estados de espín. Así, los electrones son capaces de excitarse espontáneamente, para desexcitarse posteriormente, emitiendo los famosos fotones de 21 centímetros.

1.2 Observatorio Square Kilometer Array

El Square Kilometer Array (SKA)[1] es un proyecto que tiene como objetivo construir el mayor radiotelescopio del mundo, compuesto por una matriz de antenas que cubran un área de 1 kilómetro cuadrado. Esta matriz de telescopios se está construyendo en dos emplazamientos distintos, en la región de Karoo, en Sudáfrica y en la región australiana de Muchinson Shire. Se espera que el telescopio esté completamente operativo a mediados de esta década. El proyecto es una colaboración internacional entre 14 países, que aglutinan al 40 % de la población mundial, entre los que se encuentra España. Además se les suman 8 países africanos en condición de socios, que colaborarán con la expansión del proyecto en África [11].

La enorme superficie efectiva del telescopio va a permitir observar las estrellas y galaxias que se formaron pocos años después (relativamente) del Big Bang, hecho que no se ha podido conseguir ni siquiera con telescopios fuera de la influencia de la atmósfera. El estudio de estas galaxias permitirá aumentar el conocimiento que se tiene del proceso de formación del universo, así como de los años posteriores. Además de estudiar estrellas y galaxias primitivas, el telescopio permitirá entender mejor los procesos de formación y evolución de galaxias, especialmente de las creadas en los primeros años de vida del universo. En la evolución galáctica está incluida una de las mayores incógnitas de la astrofísica actual, la materia oscura. El telescopio SKA, al detectar un número tan elevado de objetos, permitirá entender mejor las características de esta materia no emisora de radiación. Una de las mejores herramientas para estudiar la materia oscura son las curvas de rotación de las galaxias, que representan la velocidad de rotación de las estrellas que la componen, en función de su distancia al centro[2]. El SKA permitirá medir este tipo de curvas detalladamente, gracias a la medida del corrimiento al rojo de cada sección de galaxia, utilizando la línea de 21 centímetros. [12]

Probablemente el objetivo más curioso de todo el proyecto SKA es la búsqueda de vida fuera de nuestro planeta. Para ello, será necesario comprender cómo se forman los planetas similares a la Tierra, de forma que sea posible buscar planetas candidatos a contener vida. La posibilidad de generar imágenes del interior de los discos, donde se forman este tipo de planetas, es una de las características de SKA [13]. Conviene recordar que las ondas de radio apenas sufren atenuación en su viaje hacia la Tierra, por lo que permiten observar este tipo de lugares, ocultos en el espectro visible. Aquí entra en juego la potentísima resolución angular de SKA, ya que aumentará el número de regiones observadas.

[1] Toda la información acerca de SKA se puede consultar en su página web: `https://www.skatelescope.org/`

[2] Contrariamente al comportamiento descrito en la tercera ley de Kepler, las observaciones demuestran que la velocidad de rotación se mantiene más o menos constante, independientemente de la distancia al centro (salvo para estrellas situadas cerca de este). Este comportamiento se explica con la existencia de una masa, que rodea la galaxia, pero no emite radiación, la materia oscura.

Otro de los objetivos del proyecto SKA es probar la teoría de la relatividad general de Einstein. A pesar de que ya ha sido sometida a todo tipo de pruebas, superándolas con éxito, es conveniente probarla en otro tipo de campos [14]. Se prevé que el telescopio sea capaz de encontrar una combinación de cuerpos celestes muy poco probable, un agujero negro con un púlsar orbitando a su alrededor. Los púlsares se originan cuando estrellas muy masivas colapsan, formando un tipo de estrella de neutrones. Una característica única de este tipo de objetos es que, debido al enorme campo magnético al que están sometidos, la intensidad de radiación que se detecta en la Tierra oscila periódicamente con el tiempo. Esta oscilación es muy precisa, por lo que con un telescopio con suficiente precisión, sería posible estudiar el efecto gravitatorio de un agujero negro sobre su emisión. Los púlsares no emiten solo en frecuencia de ondas de radio, pero por los motivos que se han mencionado al final de la Sec.1.1.2, es mucho más factible su estudio a través de su radiación en la parte menos energética del espectro.

Otro de los *test* que el SKA tiene preparado para la relatividad general está relacionado con la que es, quizás, la predicción más asombrosa de las ecuaciones de Einstein. Las ondas gravitacionales, cuya primera observación directa se produjo hace apenas siete años por los detectores LIGO [15], han servido como uno de los test más exigentes a los que se han sometido las ecuaciones de Einstein. El radiotelescopio SKA pretende profundizar todavía más en su estudio, gracias a la detección de ondas gravitacionales con una frecuencia mucho mayor, provenientes de colisiones de agujeros negros supermasivos o, incluso, de la creación del universo. Esto se pretende conseguir utilizando una matriz de púlsares, concretamente mediante el estudio de las perturbaciones en su intensidad de emisión.

Finalmente, el SKA tiene como objetivos comprender el origen de los campos magnéticos que pueblan el universo. Actualmente, dos conocidos efectos provocados por campos magnéticos, el efecto Zeeman y la rotación de Faraday, proporcionan información de la intensidad de campo en nubes de gas frías y de la componente de campo en la dirección de observación, respectivamente [16]. Sin embargo, el efecto más prometedor, al menos para este proyecto, provocado por el magnetismo es la radiación de sincrotrón. Cuando un electrón es acelerado en un campo magnético, de forma que este cambia su trayectoria siguiendo las líneas de campo, emite radiación en forma de fotones en la región de las ondas de radio. Gracias a la elevada sensibilidad de este telescopio, en comparación con otros ya existentes, será posible estudiar regiones nunca antes observadas.

1.3 Fuentes de radio: núcleos de galaxia activos y galaxias formadoras de estrellas

En la siguiente sección, en la que se explicará el desafío que se pretende resolver, se hace referencia continuamente a fuentes, tanto del continuo de radio como de HI. Pero, ¿qué son estas fuentes? En esta sección se va a desarrollar la astrofísica que hay detrás de los archivos de datos del desafío.

Lo primero es distinguir entre fuentes del continuo y de HI (o línea de 21 cm). Como se ha comentado en la Sec.1.1.3, las nubes de hidrógeno atómico emiten por su cuenta la famosa línea de 21 centímetros, en el espectro de las ondas de radio. Por lo tanto, cualquier objeto que tenga una nube de hidrógeno atómico a su alrededor emitirá en esta longitud de onda. Sin embargo, fuera de esa longitud de onda no emiten radiación significativa. Debido a la temperatura típica de este tipo de nubes, apenas unos pocos grados por encima del cero absoluto, su emisión en el continuo de frecuencias es muy reducida. Según la ley de Stefan-Boltzmann, la potencia emitida es directamente proporcional a la cuarta potencia de la temperatura. Por lo tanto, para estructuras como las nubes de hidrógeno, la emisión en el continuo es menos relevante (en cuanto a intensidad) que las líneas de emisión, provocadas por transiciones electrónicas.

No es este el caso de las grandes fuentes de radiación del universo, que se caracterizan por un espectro continuo, con emisión significativa en todas las frecuencias. El ejemplo más característico de este tipo de emisión son los objetos que se comportan como un cuerpo negro. Un cuerpo negro es aquel que absorbe toda la radiación que recibe, independientemente de la longitud de onda o del ángulo de incidencia, y la reemite con la forma de la famosa curva de cuerpo negro.

Habiendo discutido las diferencias entre la emisión en continuo y en discreto (línea HI), solo queda concretar cuáles son los objetos que generan estas emisiones, es decir, cuáles son las fuentes de ondas de radio que se pretenden estudiar con SKA.

Uno de las grandes objetivos del proyecto SKA es arrojar luz al proceso de formación de estrellas, tanto primitivas como más actuales, a través de la observación de las llamadas *Star Forming Galaxies* o galaxias formadoras de estrellas (SFGs). Este tipo de objetos son una fuente muy importante de ondas

de radio, principalmente debido a dos procesos muy relacionados entre sí: la acreción de materia y los *jets* de partículas.

Las estrellas no se crean desde la nada, requieren de un proceso de formación de varios millones de años, en el que una pequeña masa crece hasta convertirse en la estrella definitiva, entrando en la secuencia principal. Este proceso de formación comienza cuando una pequeña masa comienza a atraer materia, principalmente polvo (o *dust*)[17]. Sin embargo, la materia no es absorbida directamente, sino que empieza a orbitar alrededor de la protoestrella, formando un disco a su alrededor. El proceso por el que la materia de este disco es "absorbida"por la protoestrella se conoce como acreción. Este fenómeno es particularmente interesante puesto que consiste en partículas cargadas siendo aceleradas, en presencia del campo magnético generado por la estrella, esto es, radiación sincrotrón. Esta radiación, en la sección del espectro de ondas de radio, domina la emisión en el continuo de este tipo de galaxias. Esto se debe a que la aceleración que experimentan las partículas cargadas y la intensidad del campo magnético dependen de la distancia a la protoestrella, de forma que el proceso emite radiación en todas las frecuencias (continuo).

El otro proceso emisor de radiación en ondas de radio son los *jets* de partículas. Estos se producen cuando una fracción de la materia acretada por la protoestrella es expulsada violentamente a través de sus polos magnéticos. Este fenómeno vuelve a consistir en partículas cargadas, aceleradas a velocidades casi relativistas, en presencia de un fuerte campo magnético. Por lo tanto, emite en el continuo de las ondas de radio a través del mecanismo de radiación de sincrotrón.

Estos dos mecanismos de emisión en el continuo (de ondas de radio) son los dominantes en cualquier galaxia en la que se estén formando estrellas. Uno de los objetivos del desafío es identificar las regiones del espacio en las que se está produciendo este tipo de emisión, para posteriormente analizarlas. Si bien la principal vía de estudio de este tipo de galaxias será el continuo de emisión, la emisión discreta también proporcionará información relevante sobre estos cuerpos, por ejemplo sobre la interacción entre el disco y la protoestrella [18]. Será muy interesante catalogar galaxias de estos tipos con corrimientos al rojo muy diferentes (calculables a partir del desplazamiento de la línea HI), de forma que sea posible estudiar la formación de estrellas primitivas, comparándolas con las actuales. También es especialmente interesante la comparación entre la formación de estrellas muy masivas y otras más normales. Pero todo esto requiere de una gran muestra de galaxias, con variedad de orientaciones tamaños y distancias. Afortunadamente, el telescopio SKA proporcionará imágenes con un número enorme de galaxias. Es por esto que el desafío es tan importante, puesto que semejante cantidad de galaxias no puede ser procesada de forma manual. Desarrollar algoritmos de detección es, probablemente, la tarea más importante que la comunidad astrofísica está realizando antes de la finalización del telescopio.

Las galaxias generadoras de estrellas no son los únicos *target* del desafío y del futuro telescopio. Algunas galaxias, entre las que se pueden incluir SFGs, contienen una fuente muy potente de radiación, capaz de eclipsar la radiación de todas las estrellas de la galaxia; por varios órdenes de magnitud, de hecho. Lo curioso es que esta región se concentra en una pequeña parte, en el núcleo, y solo de algunas galaxias. Sin embargo, su curva de emisión supera por mucho a la de todas las galaxias, independientemente de la frecuencia (Fig.1.4). Las galaxias con este tipo de núcleo se conocen como galaxia con núcleo activo (AGN).

Toda la física que rodea a los núcleos de galaxia activos tiene ciertas limitaciones, principalmente debido a lo difícil que es su observación. Como ya se ha mencionado en la sección de ventajas de la observación en radio, la parte del espectro electromagnético que comprende desde los infrarrojos hasta los rayos X es muy difícil de observar, puesto que la radiación interacciona con la materia que se interpone entre la fuente y nuestro telescopio. Actualmente, se cree que los AGN están formados por agujeros negros supermasivos, que acretan enormes cantidades de materia de las galaxias, polvo o nubes de hidrógeno que tengan cerca, debido a su inmensa masa (entre 10^6 y 10^9 masas solares). La emisión en el continuo, en el espectro de las ondas de radio, de estos objetos es otro de los *targets* del proyecto SKA, debido a la gran cantidad de información nueva que puede proporcionar. Debido al grosor óptico del disco de acreción, apenas llegan a la Tierra unos pocos fotones procedentes del núcleo y del interior del disco. La excepción a esta regla son las ondas de radio que, debido a su gran longitud de onda (en comparación con el tamaño de las partículas), puede atravesar el disco de acreción.

Al igual que en las SFGs, los núcleos de galaxia activos tienen dos mecanismos de emisión en el continuo de las ondas de radio. Son la acreción de materia cargada, en presencia del enorme campo magnético generado por el agujero negro supermasivo, y los *jets* de partículas cargadas, que emiten

ondas de radio por a través de la radiación de sincrotrón (Fig.1.5).

Históricamente, la pobre resolución de los radiotelescopios ha decantado la balanza hacia otras longitudes de onda para el estudio de estos cuerpos. Sin embargo, el SKA promete solventar este problema, proporcionando imágenes de altísima resolución, que permitan observar multitud de estos objetos, en diversas etapas de su vida. Es por ello que es clave realizar una detección y clasificación a través de un algoritmo lo más eficiente posible. Cuantos más objetos se puedan extraer de una imagen, más física se podrá realizar.

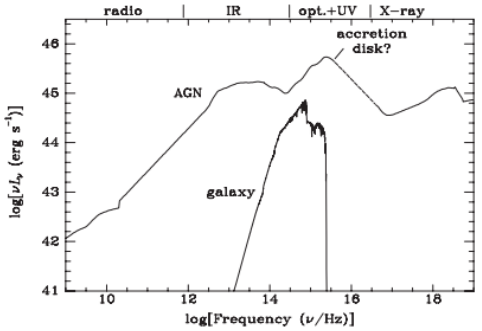

Figura 1.4: Distribución espectral de energía típica para un núcleo de galaxia activo, en comparación con una galaxia normal. El eje vertical de la figura representa la intensidad.[19]

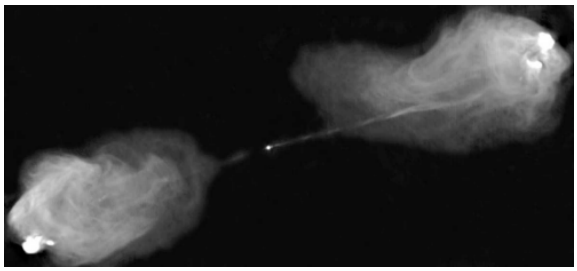

Figura 1.5: Imagen de la galaxia Cygnus-A, a una longitud de onda de seis centímetros. Se pueden distinguir perfectamente los dos *jets* emitidos por su núcleo (donde se intuye el disco de acreción), con una longitud aproximada de 100 kpc. [19]

Respecto a las fuentes de HI (entre las que pueden estar incluidas fuentes del continuo), comentar brevemente su origen y utilidad. La radiación de 21 centímetros tiene su origen en nubes de hidrógeno atómico, y se genera a través de una transición de estructura hiperfina, explicada en la Sec.1.1.3. La principal utilidad de esta radiación es mapear la distribución de hidrógeno en las galaxias que pueblan el universo. Este mapeo puede ayudar a explicar los procesos de evolución que experimentan las galaxias, simplemente observando cómo se distribuye el hidrógeno en estas. Además del estudio astrofísico de la evolución galáctica, la línea de 21 centímetros permite estimar la cantidad del galaxias por unidad de luminosidad y de distancia. Aunque "contar" galaxias parezca una tarea sin ninguna utilidad, es extremadamente útil a la hora de estimar parámetros cosmológicos (o, al menos, acotarlos), como son la curvatura del universo k, los parámetros de densidad Ω o la edad del universo. Esto es posible gracias a la comparación de modelos teóricos sobre la distribución esperada de galaxias, frente a los datos obtenidos a partir del mapeo de la línea de HI. Además, esta comparación puede proporcionar ideas o ayudar en el estudio de la formación y evolución de las galaxias, a lo largo de la historia del universo.

Sin embargo, es inútil ajustar modelos a los datos si el número de galaxias detectadas no es lo suficientemente grande. Por ejemplo, si los catálogos incluyen una proporción de galaxias más luminosas que la real (puesto que son más fáciles de detectar), falsearán los resultados de los parámetros cosmológicos, puesto que la distribución no se ajustará a la realidad. Es por esto que son necesarios cartografiados enormes, con miles de millones de galaxias, de forma que reflejen de la forma más fiel posible la realidad. Con la puesta en marcha de SKA dentro de unos pocos años, todos los registros previos que se tenían de cartografiados de galaxias parecerán ridículos en comparación con los que proporcionará SKA. Sin embargo, obtener semejante cantidad de galaxias tiene un coste. Se necesitan algoritmos de detección extremadamente potentes y eficientes, que de forma completamente autónoma puedan generar un catálogo de galaxias, partiendo de una imagen generada por el telescopio. Iniciar el camino del *software* de detección de galaxias es el principal objetivo del trabajo que se expone en las siguientes páginas. En un futuro, la idea es que este código sea ampliado para obtener una estimación de los parámetros cosmológicos, a partir del catálogo que genere la red neuronal.

1.4 *Data challenges*

Con el objetivo de familiarizar a la comunidad científica con los archivos de datos que se generarán en el observatorio, así como su posterior manipulación e interpretación, SKA ha creado 2 "desafíos"

hasta la fecha (con un tercero en proceso). Debido a la enorme cantidad de datos que se prevén generar, resulta imperativo, no solo entrenar a las personas que se encargarán de su procesado, sino también diseñar y probar algoritmos de manejo de datos nuevos. A esto se le añade la necesidad de hacer uso de supercomputadores, una actividad que también requiere de práctica por parte de la comunidad. Finalmente, estos desafíos tienen como objetivo promover los principios de reproducibilidad y de *Open Science*. De hecho, los desafíos incorporan una valoración del grado de consecución de estos principios, permitiendo a la comunidad mejorar en este aspecto de cara al futuro, cuando el telescopio esté operativo. Es por ello que el *software* desarrollado en este trabajo será completamente libre, para cualquiera que quiera utilizarlo o, incluso, modificarlo.

1.4.1 Formato de los datos del desafío

Estos desafíos consistieron en detectar el mayor número posible de fuentes[3], a partir de una imagen que simula una observación. Conseguir un *software* de detección eficiente será clave a la hora de trabajar con datos reales, ya que permitirá estudiar la formación y evolución de las grandes estructuras de materia a lo largo de la historia del universo. Pero, ¿en qué consiste detectar? La respuesta es obtener una serie de magnitudes físicas de las galaxias que pueblen un cubo de datos. En este desafío se trabajarán las siguientes magnitudes:

- Coordenadas del centro del objeto (Ascensión Recta, Declinación), en grados decimales.
- Semieje mayor, en segundos de arco.
- Los dos ángulos que describen la orientación espacial del objeto, inclinación y "ángulo de posición"(*Position Angle*). La inclinación (i) describe el ángulo entre un vector normal a la superficie de la galaxia y la línea de visión desde la Tierra (Fig.1.7). En cambio, el ángulo de posición (PA) describe cómo de rotada está la galaxia respecto al polo norte celeste, en sentido antihorario (Fig.1.6).
- Anchura de la línea de HI al 20 % de su máximo, en km/s.
- Flujo integrado sobre todas las frecuencias, en JyHz.

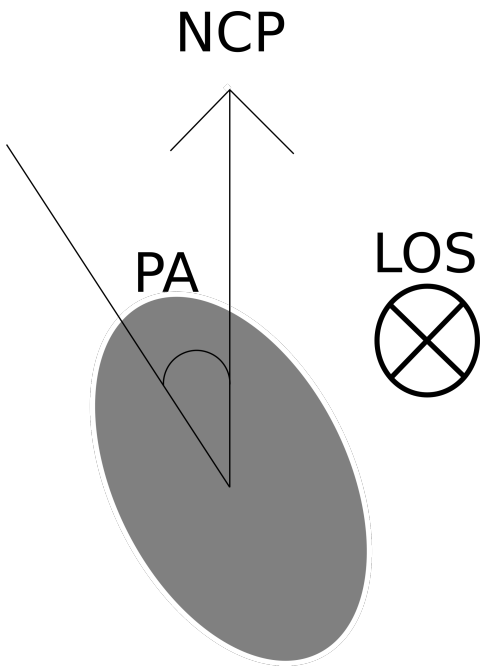

Figura 1.6: Esquema con la definición del ángulo de posición. LOS (*line of sight*) indica la dirección en la que se está observando la galaxia.

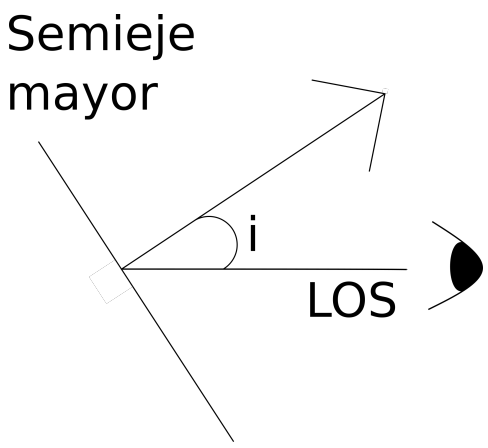

Figura 1.7: Esquema con la definición del ángulo de inclinación. LOS (*line of sight*) indica la dirección en la que se está observando la galaxia.

[3]Los tipos de fuentes que se incluyen en este desafío se explican en la Sec.1.4.2

En el *Data Challenge 2*, además de los parámetros anteriores agrupados en un catálogo, se proporcionan dos cubos de datos. El primero es el *HI datacube*, y contiene información de las fuentes de la línea de 21 centímetros, dentro de un ancho de banda, esto es, entre dos *redshifts* determinados. El segundo representa el continuo de emisión, para un ancho de banda mayor. Los parámetros concretos de los dos cubos de datos se proporcionan en la sección 2.3.

1.4.2 Generación de los cubos de datos

El proceso mediante el cual se han generado los cubos de datos se encuentra descrito en detalle en [20]. En esta sección se proporciona solo un resumen del *pipeline*, redactado a partir del documento original, puesto que el cubo de datos, junto al catálogo, han sido generados en su totalidad por los organizadores del desafío.

El primer paso es producir un catálogo de fuentes, con propiedades tanto en el continuo como en la línea de 21 centímetros. Utilizando esta relación [21], es posible obtener una distribución de masa de hidrógeno en el universo, en función del *redshift* y de varios parámetros ajustados en el artículo. Finalmente esta distribución de masa se traduce en flujo integrado (intensidad) y tamaño de las fuentes, usando otras relaciones.

En el apartado del continuo, existen dos fuentes de radiación principales. Las galaxias formadoras de estrellas (SFGs o *Star Forming Galaxies* en inglés) y los núcleos de galaxia activos (AGNs o *Active Galaxy Nuclei* en inglés). La distribución de estos objetos se realizó con el *software* T-RECS [22], para a continuación asignarle una masa de hidrógeno.

Una vez los dos catálogos están completos, se procede a juntar las fuentes. A cada fuente del continuo se le asigna una del catálogo de fuentes de emisión de la línea de 21 centímetros, con las mismas características. Finalmente, las fuentes de HI que quedan sin asignar se reservan como fuentes únicamente de HI.

Una vez el catálogo está conformado, el siguiente paso es crear los cubos de datos, que simulen las intensidades medidas durante una observación. Las fuentes del catálogo de HI se incorporan al cubo de datos utilizando muestras de otras fuentes de HI reales. Después de una serie de modificaciones, las fuentes del cubo de datos sufren una transformación de escala en sus principales propiedades (semieje mayor, semieje menor y grosor de la línea de emisión). Particularmente interesante es la relación entre la inclinación i y el semieje menor b, a través del semieje mayor D:

$$\cos^2 i = \frac{(b/D)^2 - \alpha^2}{1 - \alpha^2} \tag{4}$$

con $\alpha = 0.2$. Respecto al cubo de datos del continuo, se lleva a cabo una clasificación de las fuentes, dependiente de su tamaño. Las fuentes con un tamaño menor a 3 píxeles se describen como no-resueltas (*unresolved* en inglés), añadiéndose al cubo como Gaussianas. El resto de fuentes, con tamaño mayor a 3 píxeles, se generan utilizando un perfil de Sersic, en el caso de las SFGs. Los AGNs se simulan de forma diferente dependiendo si son de "espectro inclinado"(*steep-spectrum*) o "espectro plano"(*flat-spectrum*). Finalmente, se añaden posibles fuentes que atenúen la señal de la línea HI, proveniente de las fuentes previamente creadas.

El último paso consiste en simular cómo afectaría al cubo de datos la observación a través de un telescopio. Entrenar las redes neuronales o practicar con *software* ya creado con el cubo de datos original convertiría el desafío en algo poco realista. Para que la comunidad científica pueda enfrentarse a los datos que proporcionará el telescopio del SKA, estos tienen que ser lo más fieles posibles a la realidad.

En el caso del cubo del continuo, se produce un cubo de ruido, del mismo tamaño que el original, que represente el "error de calibración de ganancia"(*gain calibration error*). Para cada canal (imagen 2-dimensional a una frecuencia determinada), se produce una matriz de errores no correlacionados entre sí y con una r.m.s de $\sigma = 10^{-3}$. Una vez se ha simulado el cubo de ruido que generarán los equipos de detección, se juntan todas las contribuciones (telescopio más absorción, principalmente) conformando el valor final de la intensidad en cada píxel.

Por supuesto, a pesar de todos los esfuerzos realizados, el fichero de datos no es una representación exacta de la realidad. Hay varias limitaciones, principalmente en los modelos de emisión y absorción (tanto del continuo, en SFGs o AGNs, como para la línea HI), que pueden distorsionar ligeramente la

realidad. Sin embargo, cuando el telescopio entre en funcionamiento, el entrenamiento realizado con estos datos será sin duda de gran utilidad.

1.4.3 Estado actual del desafío

Durante los seis meses en los que el desafío estuvo activo, un total de 12 equipos presentaron su propuesta de *software* de detección de galaxias. Los metodología empleada por cada uno de los equipos participantes se puede consultar en el artículo que resume el desafío [20]. Una rápida lectura permite distinguir dos corrientes claras a la hora de atajar el problema: mediante programas "tradicionales"de detección, siendo SOFIA-2 [23] el más utilizado, y mediante inteligencia artificial, principalmente arquitecturas de redes neuronales convolucionales.

A pesar de que la efectividad de los programas de detección tradicionales está ampliamente probada, en los últimos años se está extendiendo el uso de redes neuronales para la detección de cuerpos espaciales. De hecho, los equipos que han quedado en primer y segundo lugar, con bastante ventaja sobre el resto, han utilizado redes neuronales, al menos, en algún punto de su solución. Como se va a comentar en la Sec.1.6, una de las principales motivaciones del trabajo es familiarizarse con los algoritmos que se utilizan actualmente para detectar galaxias. Por lo tanto, se ha elegido desarrollar una red neuronal desde cero, aunque su efectividad sea previsiblemente peor que la de otros programas más evolucionados.

1.5 Redes neuronales

1.5.1 ¿Qué son? Aplicaciones en diversos campos

Las redes neuronales están de moda, a nadie le cabe la menor duda. Pero una gran parte de la población desconoce cómo funcionan o, incluso, no saben qué son y para qué se utilizan. Una red neuronal es una estructura artificial que pretende simular el comportamiento de las neuronas humanas a la hora de reconocer patrones, principalmente. Ahora bien, para entender cómo funcionan las neuronas artificiales es útil comprender el funcionamiento de sus parientes humanas.

Las neuronas humanas se pueden dividir en tres partes principales (Fig.1.8): las dendritas, el núcleo y el axón. Los componentes que forman cada una de estas partes no es relevante para obtener una visión general. Los impulsos o estímulos llegan a la neurona a través de las dendritas, elementos que están conectados a otras neuronas o a otras partes del cuerpo como tejidos, por ejemplo. Una vez el estímulo entra en la neurona, su núcleo procesa la información, para finalmente terminar siendo transmitida a otras neuronas o a alguna parte del cuerpo, como podría ser una articulación, a través de su axón.

El detalle de lo que sucede en el núcleo de la neurona se puede ver en la Fig.1.9. La diferencia de potencial en su interior se mantiene constante en -70 mV cuando la neurona se encuentra en reposo. Si el estímulo que llega de otra neurona es lo suficientemente intenso, esto es, consigue que la diferencia de potencial supere un umbral determinado (-55 mV), la neurona se activará. Producirá un impulso eléctrico, de 35 mV de valor máximo, cuya forma dependerá de la magnitud del impulso inicial. Finalmente, el impulso generado se propagará a otras neuronas, repitiéndose el proceso hasta llegar al músculo, tejido o órgano receptor. La neurona inicial retornará a su estado de reposo, a la espera de un nuevo impulso.

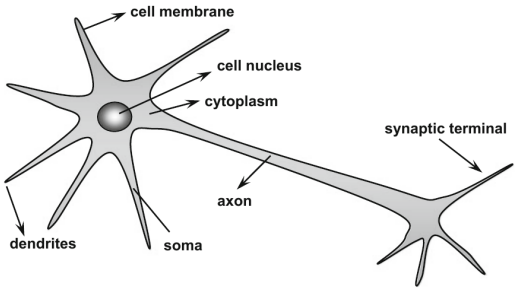

Figura 1.8: Esquema de una neurona humana, con los nombres de todas las partes importantes que la componen. [24]

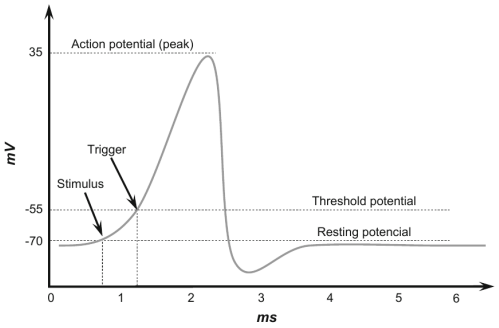

Figura 1.9: Evolución de la diferencia de potencial en el interior de una neurona, al verse estimulada por un impulso eléctrico. [24]

En resumen, la información es recogida por las dendritas (*input*), evaluada por el núcleo (produciendo un *output*) y transmitida por el axón a otras neuronas (información que servirá de *input* para las neuronas conectadas a la original). La idea detrás de las neuronas artificiales es imitar este comportamiento, pero dentro de un ordenador.

Los primeros en plantear algo parecido a una neurona artificial fueron McCulloch y Pitts en 1943, creadores del primer modelo matemático que simulaba una neurona biológica. En los 15 años posteriores se fueron desarrollando e incluso se creó el primer método para **entrenar** (concepto clave que se desarrollará posteriormente) redes de esas primitivas neuronas artificiales [25]. Sin embargo, uno de los mayores avances en la historia de este campo de la informática se produjo en 1958. El "Perceptron" es considerado como el primer computador capaz de tomar decisiones por cuenta propia [26]. Su creador Frank Rosenblatt, aunque desgraciadamente no ha podido ver cómo el mundo ha cambiado gracias a su modelo primitivo, sentó las bases de la revolución de la inteligencia artificial. Su computadora apenas era capaz de reconocer patrones ultrasencillos, como un papel pintado en la derecha o en la izquierda. Los avances tecnológicos en el campo de la computación, especialmente en cantidad de memoria, han permitido complicar mucho el diseño original de Rosenblatt, pero la base de su modelo se ha mantenido en el tiempo.

El funcionamiento base de una neurona artificial es similar al de las neuronas biológicas: en función de unos parámetros, el núcleo genera una respuesta que se transmite a otras neuronas (Fig.1.10). En este caso, las señales con la información que recibe la neurona son los parámetros $(x_1, x_2, ..., x_n)$, caracterizados por los pesos $(w_1, w_2, ..., w_n)$. Estos pesos definen cómo de importante es cada parámetro, es decir, cual va a tener más relevancia a la hora de decidir si la neurona se activa o no lo hace.

El valor de estos parámetros es de vital importancia para que la red funcione adecuadamente. El proceso mediante el cual se ajustan se conoce como entrenamiento (*training*, en inglés). En el caso de las neuronas humanas, el equivalente al entrenamiento tiene lugar durante la vida del individuo. Por ejemplo, los niños y las niñas aprenden de pequeños a diferenciar entre un coche y un árbol, es decir, su red neuronal natural se ajusta para esos *inputs* determinados (en este caso, impulsos generados en el ojo o en la nariz). La red neuronal humana es capaz de generalizar, esto es, a partir de un conjunto de datos pequeño, poder identificar objetos concretos que nunca había visto. Cuando la red se expone a estímulos similares a los que fue entrenada, es capaz de dar la misma respuesta que con los objetos originales. Al fin y al cabo, nadie ha visto todos los árboles del planeta, pero si se encuentra con uno que no haya visto, va a saber que es un árbol.

El entrenamiento de las redes neuronales artificiales funciona de forma muy similar, pero en un periodo ínfimo de tiempo, si se compara con el equivalente humano. Para entrenar una red neuronal, es necesario proporcionarle una serie de *inputs* y decirle cuáles son los *outputs* finales que esta tiene que producir. Así, minimizando el error, consigue asignarle un valor a cada uno de estos parámetros. A partir de ese momento, la red está preparada para generalizar y ser capaz de proporcionar una predicción para los datos principales.

Cuando la neurona recibe un *input*, los valores de los parámetros principales y sus respectivos pesos terminan en el núcleo de la neurona (Σ en la Fig.1.10). Después de sumarse, se comparan con el *threshold* o umbral de activación, para decidir la respuesta que tendrá la neurona. Si el valor del sumatorio es mayor que el umbral, la neurona generará una respuesta, cuyo valor dependerá del valor del sumatorio. En función de este valor, la función de activación (en la Fig.1.11 se muestra un ejemplo de un tipo de función de activación) devolverá un valor, nunca superior al máximo y al mínimo de la función de activación. Finalmente, el resultado y servirá como parámetro x_i para otra neurona. Las neuronas individuales se agrupan en capas o *layers*, que están conectadas a otras capas, formando una red. Esta red funciona como una gran única neurona, puesto que recibe unos *inputs* (una lista con imágenes) y es capaz de producir unos *outputs*, siempre y cuando haya sido previamente entrenada.

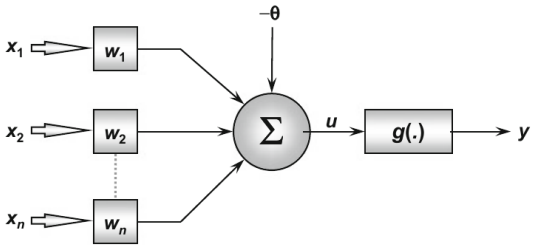

Figura 1.10: Esquema de una neurona artificial. [24]

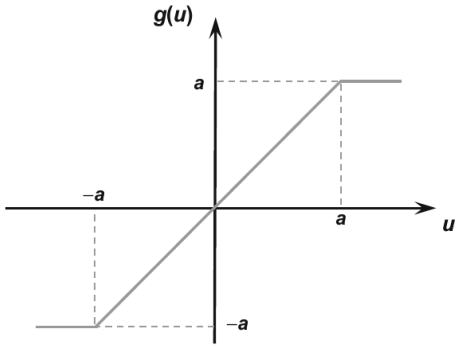

Figura 1.11: Función de activación lineal de una neurona artificial. [24].

En este trabajo se van a utilizar arquitecturas de redes neuronales, con el objetivo de detectar galaxias en una imagen de un telescopio. Sin embargo, su uso se ha extendido a campos tan diferentes como pueden ser:

- Medicina: identificación de tumores, fracturas y otras dolencias, a partir de imágenes (radiografías, ecografías...). Estos avances podrían reducir el tiempo necesario para el diagnóstico de una enfermedad, así como aumentar la precisión del diagnóstico proporcionando una segunda opinión.
- Reconocimiento de voz/facial/texto: una de las aplicaciones más interesantes tiene que ver con el lenguaje. Con redes neuronales (propiamente entrenadas) es posible identificar las palabras que un sujeto está diciendo, independientemente del idioma que esté utilizando, transcribiéndolas o incluso traduciéndolas a otro idioma. Otro ejemplo interesante tiene que ver con la seguridad vial. La nueva normativa aprobada por la dirección general de tráfico obliga a los fabricantes de coches a incorporar una serie de ayudas a la conducción, entre la que destaca el aviso de cansancio. Este sistema es capaz de "aprender" cómo se comporta una persona cansada (expresiones faciales, movimientos...), pudiendo avisar al conductor para que haga una pausa en su viaje.
- Biología y geología: identificación de diversos tipos de plantas o rocas a partir de una imagen. También es interesante la detección de partes de células a partir de imágenes de microscopio.
- Programación: con el fin de ahorrar tiempo a los programadores durante la escritura de código, se han desarrollado recientemente programas capaces de predecir (o por lo menos, sugerir la opción más probable) las siguientes palabras que se van a escribir.
- Detección de objetos: otra de las innumerables aplicaciones de las redes neuronales es la detección de determinados objetos en imágenes con más tipos de objetos.

1.5.2 Aplicación de redes neuronales al proyecto SKA. Segmentación de imágenes

Volviendo a la tarea inicial, la detección de fuentes en una imagen simulada del futuro telescopio SKA, las redes neuronales permitirán un aumento en la eficiencia de detección. En los orígenes del cartografiado de galaxias era posible llevar a cabo una clasificación casi manual de estas, algo actualmente imposible debido al aumento de las detecciones en varios órdenes de magnitud. El SKA pretende aumentar esos órdenes de magnitud todavía más, por lo que se hace imprescindible utilizar algún tipo de *software* de detección y clasificación de galaxias. Gracias al empleo de redes neuronales, será posible detectar más galaxias y de una forma más eficiente, llegando incluso a clasificarlas según su tipo. Sin embargo, conseguir desarrollar una red neuronal que detecte un gran número de las fuentes reales, con un reducido número de falsos positivos, no es labor sencilla.

Mantener un número bajo de falsos positivos es una de las prioridades a la hora de elegir la arquitectura de la red. Tratar como una fuente algo que no lo es puede confundir a programas de análisis posterior o, incluso, afectar a las teorías que se obtengan de estos datos.

Se ha optado por aplicar la técnica de segmentación de imágenes para el *software* de detección de galaxias, pero de forma inversa. En su forma original, este proceso consiste en extraer una máscara (*mask*) por cada una de las imágenes de una lista. Esta máscara no es más que una matriz en la que a cada píxel de la imagen se le asigna un valor concreto (etiqueta o *label*). Habitualmente se utilizan números naturales, siendo la elección más común (0, 1, 2) para (fondo, interior, borde). Por ejemplo, una red neuronal diseñada para detectar perros asignaría 1 a los píxeles que formen el interior del perro, 2 a los bordes (los píxeles que lo delimitan) y 0 al resto de píxeles que no componen la imagen del perro (Fig.1.12).

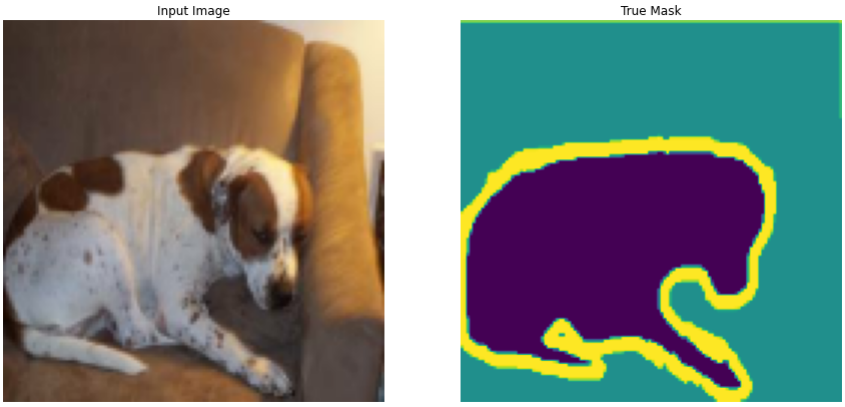

Figura 1.12: Ejemplo de imagen con su máscara respectiva. En este caso, la red está diseñada para encontrar perros. La red distinguiría entre borde (región de transición entre el perro y el fondo), perro y fondo (todo lo que no es perro). [27]

La idea inicial es una adaptación de este problema, utilizando los datos del catálogo de fuentes para crear las máscaras. Una vez se hayan generado las máscaras para cada uno de los subcubos en los que se divida el cubo grande, las imágenes junto con estas máscaras servirán para entrenar a una red neuronal. Cuando la red esté entrenada, será capaz de devolver una máscara para cada imagen que se le proporcione como parámetro (Fig.1.13).

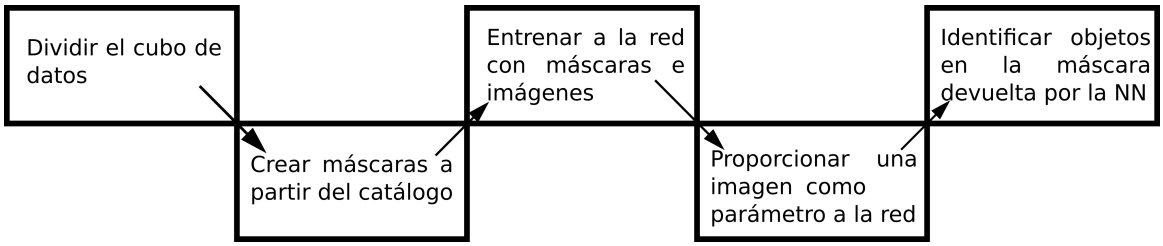

Figura 1.13: *Pipeline* del proyecto.

1.5.3 Redes neuronales convolucionales: Arquitecturas, funciones de activación e hiperparámetros

La arquitectura más sencilla de red neuronal consiste en crear capas de neuronas, como las de la Fig.1.10. Estas capas se unen entre sí hacia adelante sin que haya interacción entre capas, a parte de con las contiguas (la capa previa proporciona los parámetros y la siguiente recibe el resultado). Programar una arquitectura como esta no tiene mucha complicación, pero el resultado es poco probable que sea satisfactorio. Son necesarias arquitecturas más complejas, con una mayor conexión entre capas de neuronas.

La complejidad del problema inicial implica la entrada de lleno en el mundo del *Deep Learning*. El *Deep Learning* es un conjunto de arquitecturas de redes neuronales, cuyo objetivo es imitar cómo el cerebro humano es capaz de extraer características y patrones de objetos, clasificándolos posteriormente. Dentro de estos algoritmos, el más útil para segmentación de imágenes son las redes neuronales convolucionales (*Convolutional Neural Networks* o CNNs).

Lo primero a la hora de crear una red neuronal es escoger el número y tipo de capas que la formarán. Desgraciadamente, no existe una norma general o un modelo universal que tenga asegurada su eficacia. Diseñar una red neuronal requiere mucho tiempo de ensayo y error, modificando la arquitectura de la red (además de los hiperparámetros, de los que se hablará un poco más adelante). Teniendo esto en cuenta, existen tres tipos principales de capas con las que construir una CNN [28]:

- Capa Densa: estas capas reciben como parámetros todos los *outputs* de la capa previa (los de todas las neuronas). Habitualmente son las encargadas de llevar a cabo la parte de clasificación en la red. Conformarán por tanto, las últimas capas de la red.
- Capa Convolucional: son las encargadas de extraer la información de la imagen original. En estas capas, un núcleo (o *kernel*) de un tamaño determinado (parámetro *kernel size*) recorre la imagen principal, devolviendo un valor para cada matriz de datos accedida por este. De esta forma, la dimensión de la imagen principal se reduce. Es común evitarlo rellenando el borde de la imagen con ceros, hasta igualar las dimensiones originales, utilizando el parámetro de relleno (*padding*). Finalmente, la capa siguiente toma como *inputs* los *outputs* la capa anterior.
- Capa *Max Pooling*: estas capas se sitúan habitualmente después de una o dos capas convolucionales. Funcionan con un *kernel* recorriendo la imagen de la misma manera que las convolucionales, devolviendo el valor máximo de esa matriz de números. Sin embargo, su función principal es disminuir la dimensión de la imagen que devuelve la capa convolucional previa, habitualmente a la mitad.

Aunque existan más tipos de capas, la arquitectura base se va a construir utilizando estas tres únicamente. El *pipeline* de la arquitectura es el siguiente:

1. Leer la imagen inicial.
2. Extraer características mediante capas convolucionales.
3. Reducir la dimensión a la mitad, mediante capas *Max Pooling*.
4. Repetir los dos pasos anteriores hasta que se consiga la dimensión final deseada.
5. Construir la máscara utilizando capas densas o convolucionales traspuestas, cuyo *output* se junta con el de capas de la parte de reducción de dimensión.

A este tipo de arquitectura se la conoce comúnmente como *U-Net*. Fue utilizada por primera vez en 2015, por personal del departamento de *Computer Science* de la universidad de Friburgo, en Alemania. El *paper* original de Olaf Ronneberger, Philipp Fischer y Thomas Brox utilizaba esta arquitectura para "segmentación de imágenes biomédicas"[29]. Sin embargo, ha demostrado ser una arquitectura muy versátil, llegando a ser utilizada en campos muy distantes a la biomedicina. Una de sus ventajas más importantes respecto a otras arquitecturas aparece a la hora de entrenar la red con un *dataset* reducido. Mientras que otro tipo de modelos pierden eficiencia de detección si la muestra no es lo suficientemente grande (algo que no siempre se puede conseguir), *U-Net* es capaz de extraer las características que necesita de un conjunto de datos muy reducido. Además, el tiempo de entrenamiento de la red es mucho menor que para otras arquitecturas, hecho que permite dedicar más tiempo al ajuste de hiperparámetros.

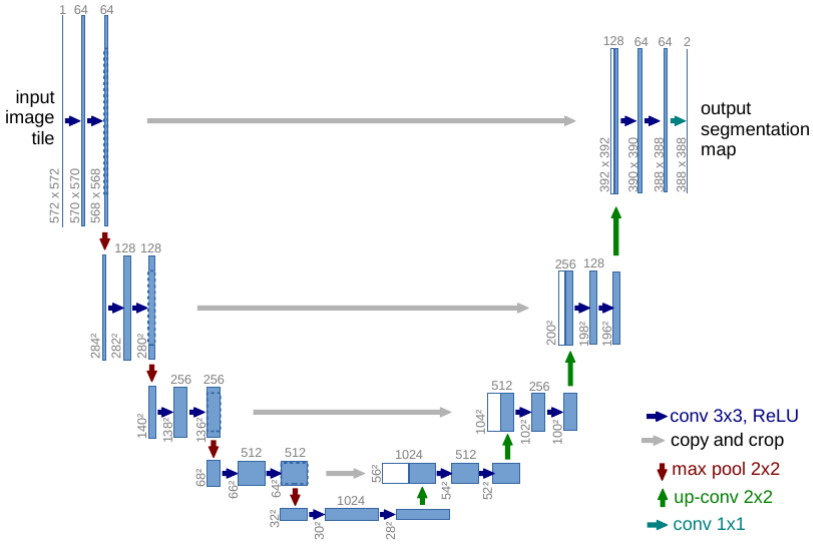

Figura 1.14: Esquema de la arquitectura de la CNN *U-Net* original. Las flechas que aparecen en la leyenda indican las operaciones realizadas entre *feature maps* (rectángulos azules). Los rectángulos mitad blancos y mitad azules hacen referencia al proceso de concatenación entre *outputs* de la fase de reducción de dimensión (mitad izquierda) y la de aumento (mitad derecha). [29]

A pesar de que la arquitectura, representada en la Fig.1.14, tiene una estructura bastante cerrada, hay una serie de parámetros que se deben ajustar y que dependen de los datos. Estos son, principalmente, el tipo de función de activación, así como otro tipo de hiperparámetros, como puede ser el tamaño de los núcleos (*kernel size*) o el número de filtros de cada capa.

La función de activación define la respuesta de la neurona, en función del resultado de la comparación con el *threshold*. En las CNNs es típico utilizar la función ReLU (*REctified Linear Unit*), cuya forma se puede representar por la siguiente expresión matemática:

$$f(x) = max(0, x) \tag{5}$$

Si el valor del *output* es negativo, la neurona no se activa, mientras que si este es positivo, la neurona devolverá el mismo valor. Esta función de activación es particularmente eficiente, puesto que evita que todas las neuronas se activen al mismo tiempo. Sin embargo, su uso se limita a las capas intermedias de la red, junto con la que procesa los *inputs*. La capa final de la arquitectura suele tener una función de activación diferente. Para problemas de segmentación, donde el *output* es un mapa de "probabilidad", se utiliza habitualmente una del dúo *softmax* y *sigmoid*.

La función *sigmoid* (ec.6) tiene sus valores acotados entre 0 y 1, de forma que, al situarla en la última capa, devolverá un mapa de "probabilidad". Realmente no se trata de una probabilidad propiamente dicha, únicamente sirve para comparar los dos tipos de *output*. Es labor del programador seleccionar qué valor servirá de umbral para clasificar cada píxel en una de las dos etiquetas posibles.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

Sin embargo, a pesar de que la función *sigmoid* parezca ideal para afrontar el problema, presenta ciertos problemas. El más importante aparece a la hora de clasificar píxeles en más de dos clases, como en el problema del perro (Fig.1.12), donde las tres clases eran (fondo: 0, perro: 1, borde: 2). Este tipo de problemas requieren de una adaptación de la función *sigmoid*, adaptada para devolver tres mapas de probabilidades, la función *softmax*, definida a continuación:

$$f(x_i) = \frac{e^{x_i}}{\sum_j x_j} \tag{7}$$

Existen innumerables funciones de activación para utilizar en las capas de las redes neuronales. Las más características se pueden consultar en [30], donde aparece su expresión matemática. Desgraciadamente (o afortunadamente, según se vea), una red neuronal convolucional con más de dos clases no

admite gran variedad de funciones. La recomendación general es utilizar *softmax* para la última capa, la que proporciona el *output* final y ReLU para el resto de capas. Sin embargo, aunque este caso es una excepción, no se pueden establecer parámetros generales, que funcionen para cualquier *dataset*. Si se demuestra que otra función de activación diferente proporciona un mejor resultado, no habría ningún problema en utilizarla.

Es importante distinguir entre dos conceptos que, a menudo se utilizan como sinónimos, aunque no sea del todo correcto. La arquitectura de la red hace referencia a la estructura de la red neuronal, mientras que el modelo consiste en una arquitectura adaptada al problema que se tiene. El matiz importante es que la arquitectura sí que es común a todos los problemas, mientras que el modelo es una adaptación de esta al problema (y depende del tipo de datos de los que se disponga). En resumen, la diferencia entre varios modelos de una misma arquitectura viene dada por unos parámetros externos ajustables, los hiperparámetros.

En la siguiente lista se definen algunos de los más relevantes, especialmente en el diseño de modelos de CNNs, con los que se jugará para encontrar el mejor modelo (los que no están incluidos en esta lista no se modificarán) [31]:

- Tamaño de la muestra o *Batch size*: principalmente por limitaciones de memoria (y de tiempo), no es posible entrenar la red neuronal con toda la lista de imágenes a la vez. Por ello, este se divide en pequeños *datasets*, que recorren la red de la misma manera que lo haría el original. A pesar de que hay ciertos valores con más probabilidad de funcionar (32, 64 o 128, por ejemplo) no se puede establecer una regla general. De hecho, es necesario encontrar un equilibrio entre valores muy pequeños, que pueden dar lugar a pérdidas de precisión; y muy grandes, que saturen la memoria o consuman mucho tiempo.

- Número de épocas o *Epochs*: con el objetivo de minimizar el error, el conjunto de datos recorre más de una vez la red. El número total de veces que la recorre se denomina épocas. Es un parámetro delicado, puesto que puede suponer la diferencia entre un modelo con *underfitting* y otro con *overfitting*. Idealmente, parece razonable que si a un modelo se le expone durante mucho tiempo a un conjunto de datos, su aprendizaje será mejor. Pero, ¿qué pasaría si, por ejemplo, a una persona que no ha visto un árbol en toda su vida, se le enseñan fotos de solo unos pocos tipos de árboles? Pues que seguramente le costaría poco identificar árboles que fuesen muy parecidos a los de su entrenamiento, pero tendría problemas a la hora de generalizar. Lo mismo sucede con las redes neuronales, un número elevado de épocas puede provocar que el modelo se ajuste en exceso al *dataset* empleado en su entrenamiento, fenómeno conocido como *overfitting*. Por otra parte, si el modelo no tiene tiempo suficiente para extraer características de los datos, no podrá "aprender" y sus predicciones serán igual de malas. Este fenómeno, opuesto al anterior, se conoce como *underfitting*. Por lo tanto, el número de épocas se convierte en uno de los parámetros críticos a la hora de conseguir un buen modelo.

- Número de capas ocultas (o *hidden*): un aumento del número de capas suele estar ligado a un mejor aprendizaje, ya que cuantas más filas de neuronas haya, más información podrán extraer (al menos en teoría). Sin embargo, cuando se usan arquitecturas creadas por otras personas, este parámetro suele venir fijado.

- Número de filtros en cada capa: explicado de forma sencilla, los filtros son los encargados de modificar el *input* inicial, o el que se recibe de otra neurona, creando un mapa de características (*feature map*). Para ajustar el número de filtros por capa tampoco existe ninguna regla, la experiencia es el único factor que puede ser útil. Es posible ajustar también el movimiento de los filtros por la imagen inicial, a través del paso o *stride*. Este parámetro determina cuántos píxeles se desplaza el filtro al recorrer la imagen.

- Tasa de aprendizaje o *Learning Rate*: la velocidad de actualización de los parámetros de la red, o *Learning Rate* determina cómo de rápido "aprende" el modelo. Si la velocidad de aprendizaje es muy elevada, el tiempo de entrenamiento se reducirá drásticamente, pero es posible que el modelo no converja. Por otra parte, si es excesivamente baja el modelo empleará mucho tiempo en converger. La tasa de aprendizaje que se va a emplear no va a ser constante en el tiempo, sino

que va a decaer siguiendo una función exponencial a medida que pasen las épocas. Es común que, en lugar de programar la función que debe seguir la tasa de aprendizaje, se opte por utilizar un optimizador. Este algoritmo se encargará de ajustar la tasa de aprendizaje a medida que avance el entrenamiento de la red.

- Fracción de datos para entrenar a la red: este parámetro define la cantidad de datos que se van a emplear para el entrenamiento y los que se van a usar para la fase de validación. Este parámetro es bastante más flexible, aunque lo común suele ser utilizar en torno a 3/4 partes de los datos para entrenar a la red.

- Dimensión de las imágenes o *input size*: el tamaño de las imágenes, que compondrán el vector que se le proporcionará a la red, también puede jugar un papel importante (aunque no se considere un hiperparámetro propiamente dicho), puesto que imágenes muy pequeñas pueden complicar la extracción de características (sería el equivalente a hacer muchísimo zoom en una foto). Sin embargo, imágenes demasiado grandes pueden agotar la memoria rápidamente.

- Función de pérdida o *loss function*: el entrenamiento de una red neuronal se puede entender como un problema de optimización, en el que el objetivo es ajustar los pesos para minimizar el error en la predicción. La función que hay que minimizar en el problema se la conoce por el nombre de función de pérdida. De entre las diferentes posibilidades disponibles, se va a utilizar la "sparse_categorical_crossentropy", utilizada ampliamente en problemas de segmentación.

Conviene definir también las posibles métricas disponibles para, tanto el ajuste de los pesos de la red, como para la evaluación de los resultados que esta proporcione. Las dos más empleadas, debido principalmente a su relativa sencillez, son la exactitud y la precisión, o por como se las conoce habitualmente *accuracy* y *precision*. Para entender su significado es necesario definir unas pocas magnitudes más. Aplicados a este problema, los positivos verdaderos (*true positives* o TP) son los píxeles que la red clasifica correctamente o, de cara al análisis, el número de galaxias detectadas que aparecen en el catálogo original. Por otra parte, los falsos positivos (*false positives* o FP) son los píxeles asignados a una etiqueta incorrecta o las detecciones que no aparecen en el catálogo original. Continuando con el mismo ejemplo, los negativos correctos (*true negatives* o TN) representan los píxeles del fondo que han sido calificados correctamente. Finalmente, los falsos negativos (*false negatives* o FN) representan los píxeles en los que han sido considerados como fondo, cuando en realidad, deberían corresponder a una detección.

A partir de estas cuatro magnitudes, se puede definir la exactitud:

$$acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{8}$$

y la precisión:

$$prec = \frac{TP}{TP + FP} \tag{9}$$

Estas dos magnitudes se emplearán en los entrenamientos de las redes neuronales, principalmente para evaluar su rendimiento, es decir, cómo de bien se están ajustando. Más adelante, la precisión será la métrica escogida, debido a su sensibilidad a falsos positivos. Es por esto que se usa en este tipo de problemas de segmentación, donde hay muchos píxeles que pertenecen a la misma etiqueta.

Ajustar los parámetros es posiblemente una de las tareas más complicadas a la hora de desarrollar una red neuronal, sea cual sea su tipo. Son necesarias muchas pruebas para conseguir el resultado más óptimo, lo que lo convierte en una de las tareas que más tiempo consumen.

1.6 Motivación y objetivos

Este trabajo comparte la motivación que el SDC2 (*Science Data Challenge* 2) tuvo en su momento, ayudar a la comunidad a familiarizarse con los datos que genere el telescopio SKA cuando esté operativo. A pesar de que el desafío terminó hace prácticamente un año, enfrentarse a él sigue teniendo sentido, incluso con acceso al artículo con las soluciones implementadas por otros equipos.

Lógicamente, desarrollar un *software* de detección igual de eficiente que los presentados en el artículo no es viable, debido a la escasez de tiempo y recursos. Sin embargo, que el programa detecte más o menos cuerpos no es lo importante. Es mucho más valiosa la experiencia que se espera adquirir en el manejo de este tipo de conjuntos de datos, así como en el diseño y manejo de redes neuronales. Además, se pueden incorporar ideas sueltas de otros equipos en el proyecto, pero sin llegar a copiar su *pipeline* completamente. De hecho, aprender a proponer soluciones para problemas nuevos es una de las principales motivaciones de este trabajo. Es común que durante todos los años de la carrera se resuelvan problemas con una solución clara, conocida de antemano por el profesor o la profesora. Sin embargo, fuera de la carrera no existen soluciones únicas a los problemas, ya que varios grupos de personas pueden proponer soluciones igualmente válidas (de hecho, es lo que ha pasado en este desafío). Este es el mayor cambio respecto al resto de trabajos que se realizan en la carrera.

Con respecto al campo de la astrofísica, el objetivo principal es interpretar las imágenes que simulan las que proporcionará el futuro telescopio. Gracias al análisis de los datos, será posible entender cómo emiten los cuerpos en el espectro de las ondas de radio y poder identificarlos en futuras imágenes. Además, el trabajo servirá para entender las magnitudes que se utilizan para describir las galaxias, especialmente en el campo de ondas de radio.

En el campo de la inteligencia artificial, y concretamente en las redes neuronales, se pretende desarrollar una red neuronal que sea capaz de identificar alguna de las fuentes presentes en un cubo de datos. Para ello, será necesario profundizar en el campo de la segmentación de imágenes, identificando las arquitecturas más apropiadas para ello. También, a diferencia de la mayoría de los problemas de segmentación de imágenes, este problema requiere construir las máscaras desde cero, a partir de un catálogo con datos. Para ello, será necesario indagar en el significado físico de cada magnitud del catálogo, como se menciona en el párrafo anterior.

Durante las secciones anteriores se ha remarcado la dificultad de ajustar un modelo con los mejores parámetros posibles. Es un hecho que la experiencia es un añadido que facilita su ajuste óptimo (o al menos reduce el tiempo que se emplea en conseguirlo). Otro de los objetivos del trabajo es mejorar el manejo de estos hiperparámetros, de forma que en futuros problemas se emplee una menor cantidad de tiempo.

Pero, la parte de programación no se limita únicamente al desarrollo de la red neuronal. Mejorar en el manejo de archivos de datos muy pesados es otro de los objetivos importantes. A la hora de manejar este tipo de archivos hay que tener mucho cuidado con las funciones y operaciones que se utilizan, pues estas pueden alargar enormemente el tiempo de ejecución. Por ello, se pretende que este trabajo sirva como iniciación en el manejo de este tipo de archivos, intentando conseguir el mejor tiempo de ejecución posible. Este aspecto no se tiene habitualmente en cuenta en trabajos realizados en la universidad, pero el rendimiento del programa es igual de importante que la exactitud de sus resultados. Otro de los aspectos que se pasan por alto en trabajos de programación universitarios (aunque también fuera de la universidad) es la documentación del código. Aprender a documentar correctamente todos los módulos de un paquete es otro de los propósitos de este trabajo. Cuando se desarrolla un trabajo de estas dimensiones es de extrema importancia llevar la documentación al día, no solo para que otras personas puedan entender el código, sino para ser capaz de entender lo que uno ha hecho hace varios meses.

Finalmente, también es un objetivo importante comprender cómo se trabaja en los centros de investigación, fuera del ambiente controlado de la universidad. A esto se le suma el aprender a llevar un trabajo al día.

2 Método experimental

Este trabajo se ha desarrollado íntegramente en un ordenador, por lo que no hay un montaje experimental propiamente dicho. Sin embargo, es interesante comentar brevemente el *setup* que se ha utilizado, concretamente el *hardware* y *software*, de forma que el *pipeline* seguido pueda ser reproducido fácilmente por otras personas.

2.1 *Hardware*: especificaciones del equipo

Los equipos que compitieron en el SDC2 utilizaron los recursos proporcionados por SKA, en la forma de ocho supercomputadores, localizados en ocho países diferentes. En total, 15 millones de horas de CPU y 15 TB fueron facilitadas para el desarrollo del desafío [20]. Sin embargo, como este finalizó hace casi un año, estos recursos ya no se encuentran disponibles. Aún sin estos recursos computacionales, existía la posibilidad de usar el supercomputador Altamira (en el IFCA), pero su uso excedía por mucho los objetivos del trabajo (no se utilizó principalmente, por falta de tiempo).

Debido a las limitaciones computacionales, se ha simplificado el problema lo más posible, aún comprometiendo la eficacia de detección. Sin embargo, a pesar de estar lejos de las especificaciones de un supercomputador, se ha dispuesto de un ordenador con 16 GB de memoria RAM, junto con un procesador Intel-I7. Pero el componente más importante ha sido la tarjeta gráfica, que en vez de estar incorporada en la CPU, es una unidad independiente. En efecto, la tarjeta NVIDIA RTX2060, con 6 GB de memoria dedicada, ha sido de gran ayuda, gracias a la posibilidad de utilizarla para entrenar a la red neuronal, como se comentará en la siguiente sección.

2.2 *Software*: *python* y el paquete Anaconda

El objetivo del trabajo es desarrollar una red neuronal que detecte galaxias, partiendo de una imagen que simule la observación a través de un telescopio. Para ello, se pretende utilizar únicamente el lenguaje de programación *python*, ampliamente utilizado en muchas áreas de la ciencia, entre las que se incluye el campo de la astrofísica. Esto es debido a la multitud de paquetes, una gran parte creados por la comunidad, que permiten manejar este tipo de datos. Además su simplicidad y eficiencia (esta última gracias a la utilización de paquetes programados en lenguajes más rápidos, como el caso de *numpy*, programado en C) hacen especialmente atractivo su uso.

Como es habitual, toda la instalación de *python* y sus librerías se ha realizado a través de Anaconda [32]. Esta instalación permite instalar de una forma sencilla el lenguaje y un interprete (en este caso, *Spyder*), además de las librerías más comunes (*numpy*, para manejo de vectores y matrices, *matplotlib*, para manejo de gráficas...)

Una vez instalado, es común no utilizar la instalación base de anaconda para desarrollar programas. Por ejemplo, si un determinado paquete no ha recibido actualizaciones recientes, es posible que necesite una versión anterior de otros paquetes para funcionar. Para evitar tener que modificar la instalación principal cada vez que se desarrolla un proyecto, anaconda permite crear ambientes o *environments*, que no son más que instalaciones independientes, donde es posible modificar las librerías sin romper la instalación principal. Como el programa se va a desarrollar en *Linux*, concretamente en Fedora 35, toda la instalación se realiza a través de la línea de comandos. Para crear y activar el *environment* se ha utilizado el *script* de línea de comandos Cód.1.

Código 1: Creación y activación de un *environment*, con nombre tf_gpu y versión de *python* 3.10

```
#!/bin/bash
conda create --name tf_gpu python=3.10
conda activate tf_gpu
```

Una vez creado el ambiente, solo falta instalar las librerías que se vayan a utilizar (solo las más generales se instalan al crear el ambiente). Para ello, después de activar el ambiente, se ejecuta el *script* Cód.2

Código 2: Instalación de dos paquetes en el nuevo ambiente, en este caso *astropy* y *pandas*.

```
#!/bin/bash
```

```
conda install astropy
conda install pandas
```

A parte de los paquetes *numpy* y *matplotlib*, obligatorios en casi cualquier proyecto, los dos paquetes en los que se basará el código son *pandas* y *astropy*. El primero tiene un uso muy extendido en cualquier problema de *Data Science* que involucre grandes cantidades de datos. Permite manejar de forma muy sencilla (y eficiente) las matrices de datos, pudiendo modificarlas sin dificultad. Por otra parte, *astropy* es especialmente útil en el manejo de archivos FITS [33], además de para realizar transformaciones de unidades, como se comentará más adelante.

Estos dos paquetes se utilizarán para la preparación y manejo de los datos, pero es necesaria una librería más para crear las redes neuronales convolucionales. El paquete por excelencia para realizar esta tarea en *python* es *tensorflow*. Este paquete reduce enormemente la complejidad y el tiempo empleado en programar redes neuronales, además de en su entrenamiento, gracias a todos los métodos y funciones ya implementados que proporciona. Merece también una mención especial la librería *keras*, incluida dentro de *tensorflow* que facilita la creación de las capas de la red neuronal. Programado usando las funciones de esta librería está *keras_unet_collection* [34], un paquete creado por la comunidad con una serie de arquitecturas de redes, orientadas a problemas de segmentación, ya preprogramadas. De esta forma, solo aportando una serie de hiperparámetros es posible probar diferentes modelos. Estas redes se utilizarán, junto con una red creada desde cero, para encontrar el mejor modelo posible.

Estos y el resto de principales paquetes instalados en el ambiente se presentan en la Tab.2.1

Tabla 2.1: Resumen con los paquetes más relevantes instalados en el ambiente de conda "tf_gpu", con sus respectivas versiones. La información completa se encuentra disponible en el *git* del proyecto [35], en el fichero *version_paquetes_env.txt*

Librería	Versión
astropy	5.0.2
cudnn	8.2.1.32
keras	2.7.0
keras-preprocessing	1.1.2
keras-unet-collection	0.1.13
matplotlib	3.5.1
numpy	1.22.3
pandas	1.4.1
python	3.10.2
sphinx	5.0.0
sphinx_rtd_theme	1.0.0
spyder	5.2.2
tensorflow_gpu	2.7.0

2.2.1 Tensorflow y las GPUs

Es habitual que los *scripts* de *python* se ejecuten utilizando el procesador del ordenador. Sin embargo, entrenar una red neuronal es una actividad que consume enormes cantidades de recursos computacionales. De esta forma, se convierte en tarea imposible entrenar una red neuronal convolucional con un mínimo de complejidad. Es por ello que en *tensorflow* desarrollaron una versión alternativa de sus librerías, capaz de ejecutarse en la tarjeta gráfica. Para ello, se sirvieron de la herramienta CUDA, disponible únicamente para las tarjetas gráficas de NVIDIA. Como dice en su propia página web [36]: "With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs".

Afortunadamente, el ordenador portátil utilizado para la programación del código dispone de una tarjeta gráfica dedicada, concretamente una tarjeta NVIDIA RTX2060, con 6 GB de memoria dedicada. Por lo tanto, a pesar del sufrimiento que puede generar instalar este tipo de cosas (principalmente debido

a problemas de compatibilidades), merece la pena por el futuro tiempo ahorrado en el entrenamiento de las redes, además del incremento de memoria disponible (que hace posible utilizar arquitecturas más complejas). A todo esto hay que añadirle el conocimiento que se adquirirá, que será posible utilizar en futuros proyectos. A continuación se describen los pasos seguidos para instalar este *software*, en un ordenador con el sistema operativo Fedora 35.

Es conveniente recordar que en los ordenadores con *Linux* es necesario instalar los *drivers* propios de NVIDIA, ya que los que vienen instalados son de código libre. Afortunadamente, este proceso se ha simplificado enormemente en los últimos años, con la inclusión de estos en el repositorio RPM. Los pasos seguidos para su instalación [37] se encuentran descritos en Cód.3. Además de realizar estos pasos, es imprescindible, si el ordenador cuenta con una tarjeta gráfica integrada, a parte de la dedicada, desactivar el *secure-boot* en las opciones de la BIOS.

Código 3: *Script* con los pasos seguidos para descargar e instalar los drivers de NVIDIA en Fedora 35.

```
#!/bin/bash
sudo dnf install dnf-plugins-core -y
sudo dnf install https://download1.rpmfusion.org/free/fedora/rpmfusion-free-release-$(rpm -E
    ↪ %fedora).noarch.rpm
sudo dnf install
    ↪ https://download1.rpmfusion.org/nonfree/fedora/rpmfusion-nonfree-release-$(rpm -E
    ↪ %fedora).noarch.rpm
sudo dnf update --refresh
sudo dnf install akmod-nvidia
sudo dnf install xorg-x11-drv-nvidia-cuda
reboot
```

Una vez se ha comprobado la instalación de los *drivers* de NVIDIA, el siguiente paso es instalar la versión de *tensorflow* orientada a GPUs, junto con la herramienta CUDA, propiedad de NVIDIA. Este proceso resultó ser terriblemente arduo, principalmente debido a problemas de compatibilidades entre librerías, el propio CUDA, el sistema, etc. Sin embargo, después de intentar multitud de métodos de instalación (incluso los recomendados en la documentación de *tensorflow*), gracias a este artículo [38] fue posible hacer una instalación exitosa. De hecho, no solo fue posible, sino que fue extremadamente sencillo. El truco está en hacer la instalación vía conda. La instalación se puede realizar con exclusivamente una línea de código (Cód.4)

Código 4: Comando para la instalación de la versión adaptada a GPUs de *tensorflow*. Este comando se debe ejecutar una vez se haya activado el ambiente de conda.

```
#!/bin/bash
conda install tensorflow-gpu
```

Este comando instalará todas las dependencias que "tensorflow-gpu" necesite, con todas las versiones compatibles. Sencillamente maravilloso.

Por último, solo queda revisar la instalación, es decir, comprobar que al importar la librería *tensorflow* en *python* detecta y asigna la GPU como medio de ejecución. Para ello, después de activar el ambiente y entrar en *spyder*, se ejecuta el *script* Cód.5. Como el *output* se corresponde con lo esperado (mismo formato que el del artículo anterior [38]), concluimos que la instalación ha sido correcta.

Código 5: Código de *python* para verificar la instalación de *tensorflow_gpu* (dos primeras líneas). El *output* del *script* debe contener un texto con el mismo formato que las dos últimas líneas.

```
import tensorflow as tf
sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(log_device_placement=True))

* output:
Created device /job:localhost/replica:0/task:0/device:GPU:0 with 4191 MB memory: -> device:
    ↪ 0, name: NVIDIA GeForce RTX 2060, pci bus id: 0000:01:00.0, compute capability: 7.5
```

Después de varias horas de búsqueda de información, el uso de la GPU para entrenar redes neuronales fue posible. Aunque para este proyecto probablemente no salga a cuenta, debido a su relativa simplicidad,

es una herramienta que seguro tiene mucha utilidad en futuros proyectos. Las diferencias en el rendimiento del programa entre GPU y CPU se presentan al final de la sección de resultados, concretamente en la Sec.3.6.

2.3 Cubo de datos y catálogo

Los datos que se van a utilizar, tanto para entrenar y validar la red como para hacer predicciones, están disponibles en la página web de SKA [39]. En esta página web hay un archivo de datos de casi 1 TB, el mismo que los equipos que participaron en el desafío utilizaron para desarrollar sus métodos. Idealmente, utilizar este cubo de datos completo sería la mejor opción. Sin embargo, una serie de limitaciones computacionales complicaron enormemente su uso (memoria y almacenamiento, principalmente), ya que sería necesario disponer de un supercomputador. Como se decidió reducir la complejidad del problema para adaptarlo a un trabajo de final de grado, fue necesario conformarse con los dos cubos de menor tamaño que se proporcionan en la web. Estos son el *development dataset*, de 40 GB, y el *development dataset* versión todavía más reducida, que se utilizará para probar la red.

Los cubos de datos que proporciona SKA son ficheros FITS[33], un formato ampliamente utilizado en el campo de la astrofísica. Los archivos de este tipo consisten en una serie de *keywords* con su respectivo valor, en formato ASCII (legible para personas), seguido de toda la información de la imagen encriptada. Dentro de cada *dataset* hay un par de estos archivos, ambos representando la intensidad, en unidades de JY/BEAM para un total de 2000 horas de observación (incluyendo ruido, interferencias...). Uno de ellos representa el continuo de emisión (*cont_ldev.fits*) y el otro la línea de emisión HI (*sky_ldev_v2.fits*). Los parámetros más representativos del archivo con el continuo de emisión se presentan en la Tab.2.2

Tabla 2.2: Parámetros principales del archivo FITS con las intensidades del continuo, con una descripción de lo que representa cada uno [40], junto con la *keyword* y el valor que aparece en el archivo original. El resto de parámetros se pueden consultar en el archivo original, que se encuentra en la web de SKA[39].

Descripción	***Keyword***	**Valor**
Número de bits que representan un valor	BITPIX	-32
Número de ejes de los datos	NAXIS	3
Tamaño del eje 1	NAXIS1	643
Tamaño del eje 2	NAXIS2	643
Tamaño del eje 3	NAXIS3	21
¿Contiene extensiones?	EXTEND	T(Sí)
Valor para los elementos no definidos de la matriz	BLANK	-1
Unidades de la matriz de datos	BUNIT	“JY/BEAM”
Píxel de referencia del sist. coord. 1	CRPIX1	322
Anchura de un píxel en el eje 1	CDELT1	$7.\overline{7} \cdot 10^{-4}$
Nombre del eje 1	CTYPE1	“RA—SIN”
Píxel de referencia sist. coord. 2	CRPIX2	322
Anchura de un píxel en el eje 2	CDELT2	$7.\overline{7} \cdot 10^{-4}$
Nombre del eje 2	CTYPE2	“DEC—SIN”
Píxel de referencia del sist. coord. 3	CRPIX3	1
Anchura de un píxel en el eje 3	CDELT3	$1 \cdot 10^{7}$
Nombre del eje 3	CTYPE3	“FREQ”
Longitud del polo norte del sistema	LONPOLE	180
Latitud del polo norte del sistema	LATPOLE	-30
Valor máximo de la intensidad	DATAMAX	$8.4426 \cdot 10^{-3}$
Valor mínimo de la intensidad	DATAMIN	$-1.2484 \cdot 10^{-5}$
Unidades del eje 1	HISTORY CUNIT1	’deg’
Unidades del eje 2	HISTORY CUNIT2	’deg’
Unidades del eje 3	HISTORY CUNIT3	’Hz’

El archivo del continuo contiene la información de las fuentes que emiten en todas las frecuencias de un ancho de banda determinado, concretamente entre 950 y 1400 MHz, en incrementos de 50 MHz. Por otra parte, el cubo de datos con las fuentes de HI es ligeramente diferente, operando en la banda 950–1150 MHz, con un incremento de 30 kHz entre imágenes [20]. Sin embargo, la red neuronal se va a entrenar únicamente con el cubo de datos del continuo, por cuestiones de simplificación del problema. Por lo tanto, no será capaz de detectar fuentes únicamente emisoras en 21 cm. Como se ha comentado en la sección 1.4.2, los dos catálogos se crean de forma independiente, para a continuación identificar una fuente del continuo con una de HI. Pero no todas las fuentes de HI se pueden asociar con una del continuo, por lo que en el cubo de datos aparecen fuentes solo de HI. De esta forma, de ahora en adelante se va a considerar el cubo del continuo como único cubo de datos.

Durante las secciones previas se ha asociado el *dataset* a un cubo, aunque la intuición diga que las imágenes de telescopios son en dos dimensiones (Ascensión Recta y Declinación, en este caso). La novedad de este *Data Challenge* respecto al primero es la inclusión de una tercera dimensión en los ficheros de datos. Aunque esto pueda parecer contraintuitivo, no lo es tanto. La banda de frecuencias, de la que se hacia referencia en el párrafo anterior, funcionará como tercer eje en el cubo. El valor de este eje (equivalente a profundidad o a distancia a la fuente) es especialmente relevante en el cubo con fuentes de HI, donde se puede identificar fácilmente con el *redshift* z de la parte concreta de la fuente que representa un píxel en la imagen. Si un vector normal al plano galáctico tiene un ángulo de inclinación con respecto a nuestra línea de visión, el corrimiento al rojo será diferente en los dos extremos de la galaxia. Por lo tanto, la línea de HI estará desplazada a una frecuencia diferente. Este corrimiento al rojo puede deberse a la expansión del universo o al movimiento intrínseco de la galaxia (que puede desplazar al azul la línea si la galaxia no está muy alejada del telescopio, de forma que la expansión del universo no influya apenas).

Sin embargo, no es tan relevante en el cubo del continuo, puesto que apenas hay diferencia entre las imágenes a distintas frecuencias. Como su propio nombre indica, en este cubo se representa el continuo de emisión de varios tipos de cuerpos (SFGs y AGNs, principalmente). La función que modela este espectro de emisión no tiene grandes variaciones, sino que es más bien suave. Todo lo contrario sucede con la línea HI, que se modela por una delta de Dirac, al menos idealmente. Otra forma de visualizar la tercera dimensión es a través del espectro, que no es más que la intensidad de un píxel para el intervalo de frecuencias, es decir, equivalente al cubo visto de lado.

Como se ha planteado el problema sin tener en cuenta el cubo de datos de HI, la primera simplificación que se va a realizar es reducir las dimensiones del problema. Como apenas hay diferencia entre secciones del cubo de datos del continuo (2.1), es razonable reducir la dimensionalidad del problema proyectando todas las imágenes, de distintas frecuencias, sobre un único plano. La intensidad de cada píxel se correspondería con la media de todas las intensidades para diferentes frecuencias. El eje y (vertical) de la figura tiene valores negativos debido a que el telescopio SKA estará situado en el hemisferio sur, hecho que explica la declinación negativa.

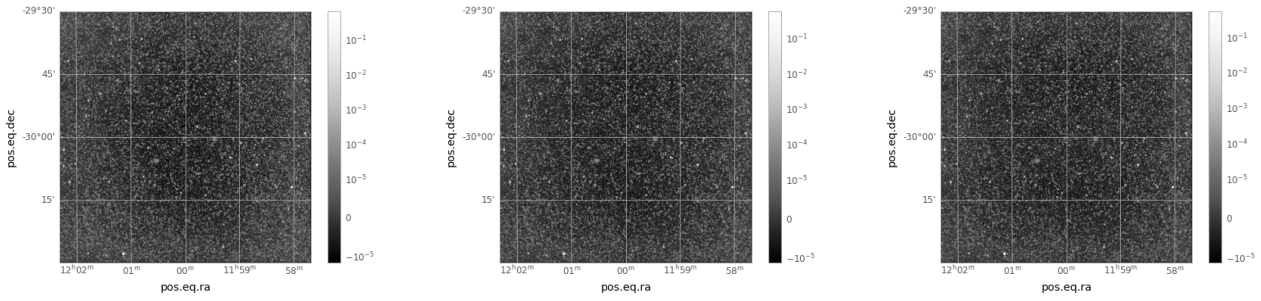

Figura 2.1: Sección del cubo de datos del continuo en tres frecuencias diferentes: 950 Hz, 1200 Hz y 1400 Hz, respectivamente.

La validez de la simplificación se confirma con la Fig.2.2, que no presenta diferencias significativas con las imágenes individuales, como las representadas en la Fig.2.1. Por lo tanto, en las siguientes secciones se convertirá en la única fuente de datos.

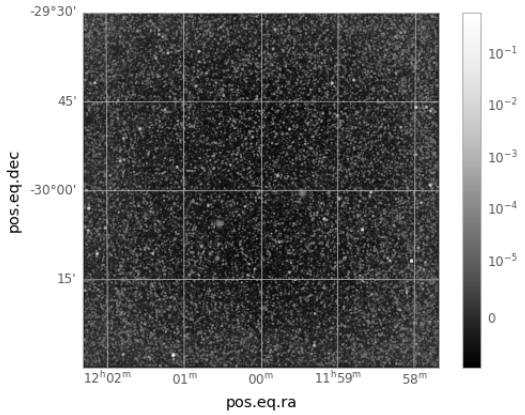

Figura 2.2: Representación de los datos en dos dimensiones, después de transformar la intensidad en cada píxel en la media de las intensidades para todas las frecuencias.

Además del archivo con las fuentes, SKA proporciona un catálogo con información sobre estas (fichero *sky_ldev_truthcat_v2.txt*). Se trata de un simple fichero de texto, con los valores y etiquetas separados por espacios, como se muestra en la Tab.2.3. Las columnas de esta tabla representan cada una magnitud física, que se debe interpretar a la hora de generar las máscaras. Las dos primeras columnas, sin contar "id" que simplemente es el número de la galaxia, representan las dos coordenadas espaciales básicas (del centro de la galaxia), "altura" y "anchura", en el sistema de coordenadas ecuatorial. La ascensión recta, "ra" representa el ángulo entre un punto especial, tomado como referencia (el *Vernal Equinox*), y el meridiano que atraviesa el punto a observar. Por otra parte, la declinación, "dec", representa la apertura angular entre el ecuador celeste y el punto. Por otra parte, la magnitud "hi_size" representa el tamaño del semieje mayor de la galaxia, medido en el telescopio. El parámetro "central_frec" completa la dupla de coordenadas espaciales, proporcionando la distancia o "profundidad". Sería equivalente a utilizar el corrimiento al rojo como magnitud de distancia, en este caso, al centro de la galaxia. El parámetro "w20" consiste en la anchura de la línea de HI, a un 20 % de la altura de su pico. Este parámetro guarda relación con la distribución de velocidades de los cuerpos que forman las galaxias. Según el efecto Doppler, cuanto más rápido gire un cuerpo emisor de radiación (con respecto a nuestra posición), más diferencia habrá entre la frecuencia emitida y la recibida. Finalmente, "i" y "pa" simbolizan la inclinación y el ángulo de posición de todos los cuerpos. La definición de estos dos ángulos se presenta en la Sec.1.4.1 de la introducción.

Tabla 2.3: Formato del catálogo de fuentes. Las unidades de los parámetros son: grados decimales (ra y dec), segundos de arco (hi_size), JyHz (line_flux_integral), Hz (central_freq), grados (i y pa) y km/s (w20).

id	ra	dec	hi_size	line_flux_integral	central_freq	pa	i	w20
0	180.1913	-29.7410	4.4519	3.6691	$1.1208 \cdot 10^9$	268.1051	53.3826	145.9504
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

La información que se extraiga de este archivo servirá para entrenar a la red neuronal, como se explicará en el apartado de resultados. El catálogo original tiene 11090 fuentes, por lo que es imposible, y bastante poco interesante, representarlo en la tabla anterior. Por ello, se ha sobreimpresionado sobre la imagen del telescopio, en la Fig.2.3, de forma que sirva para hacerse una idea de la cantidad de fuentes que hay en la imagen.

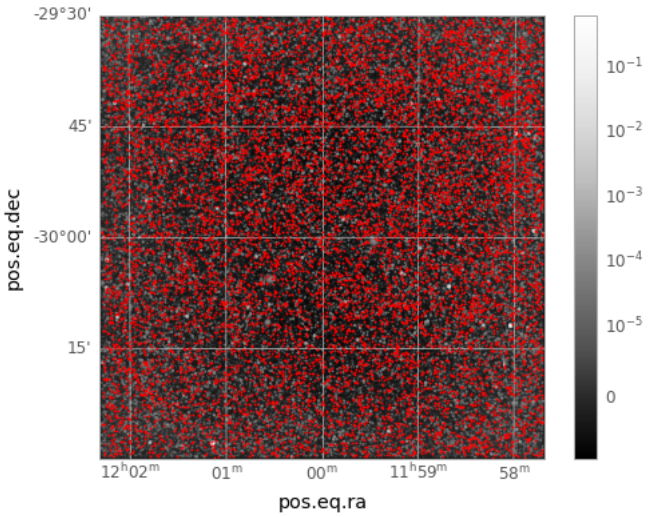

Figura 2.3: Imagen medida por el telescopio, después de la reducción de dimensión, con las fuentes marcadas como un punto rojo. El tamaño de las fuentes no se corresponde con el del punto, ya que con tan poco zoom se solaparían si el punto tuviese un diámetro igual a hi_size.

2.4 Estructura del programa

La Fig.1.13 servía como breve introducción a la estructura del programa. En esta sección se pretende profundizar todavía más en esta, principalmente a través del "diagrama de secuencia"(el nombre original en inglés es *sequence chart*) de la Fig.2.4. Las cajitas amarillas que aparecen en este se corresponden con los módulos o clases que se generarán para manipular los datos. Para más información sobre estos módulos y clases, consultar la Fig.3.14 y la sección de resultados, donde se comentará esta figura.

Como muestra la Fig.2.4, el centro del paquete será la clase *GalaxyDetector*. Esta clase será la encargada de interaccionar con el usuario, recibiendo sus órdenes y ejecutándolas, llamando al resto de clases y módulos. Cuando el usuario o la usuaria cree un objeto de esta clase, es decir, una observación; esta leerá el archivo FITS que se encuentre en la ruta proporcionada como parámetro (así como el catálogo). A continuación, reducirá la dimensionalidad del problema a únicamente dos dimensiones (ascensión recta y declinación).

El siguiente paso es especialmente interesante, puesto que está orientado a mejorar el rendimiento de la red, así como el tiempo que se dedica a su entrenamiento. Consiste en reducir el tamaño de las matrices de datos que se utilizarán para este fin. Es evidente que una red neuronal no se puede entrenar con una sola imagen, por muy grande que sea. Por lo tanto, se procede a generar una serie de matrices cuadradas pequeñas, a partir de la imagen original, con un tamaño (en píxeles) que se pase como parámetro. De esta forma, es posible aumentar el tamaño de la muestra hasta números suficientes para entrenar una red neuronal. Sin embargo, no es oro todo lo que reluce. Este planteamiento genera una duda, ¿qué pasaría si una galaxia cae justo en el píxel donde se separan los minicubos?

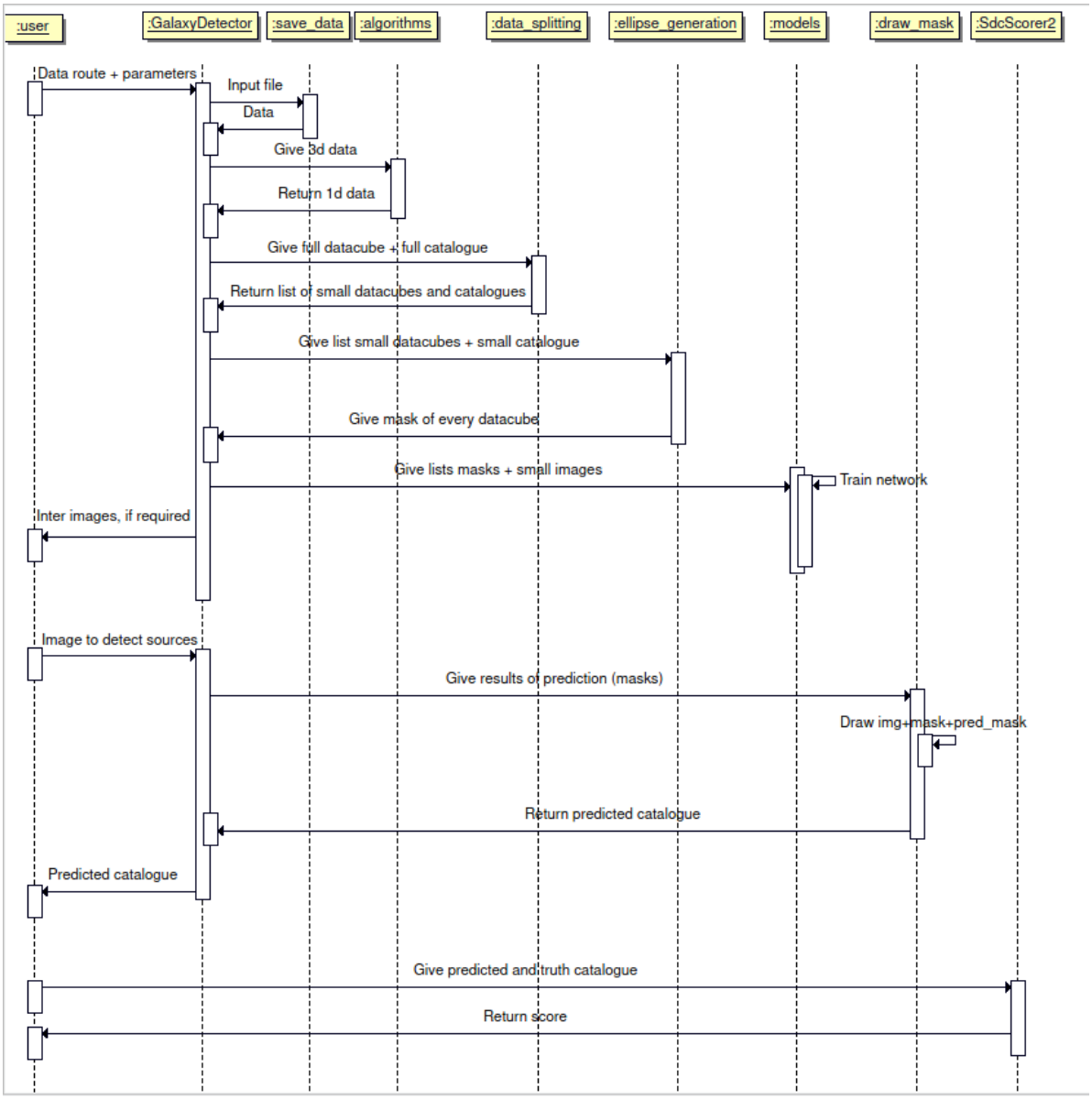

Figura 2.4: *Sequence chart* representando el *pipeline* del proyecto.

Es exactamente lo que se pretende mostrar en la Fig.2.6, donde son dos las galaxias que son cortadas al dividir el cubo. Este hecho puede provocar confusión a la red neuronal, ya que se le está diciendo que media elipse es una galaxia. Si esto sucede un número importante de veces, la red no aprenderá bien y no será posible detectar galaxias eficientemente.

Una solución posible sería cortar el cubo solo por donde no corte ninguna galaxia. Sin embargo, la gran cantidad de fuentes que hay en la imagen (Fig.2.3) complicarían mucho conseguir cubos del mismo tamaño. Mucho mejor es la opción del solapamiento. De hecho, prácticamente todos los equipos participantes del SDC2 lo utilizaron. Consiste en incorporar a cada minicubo un trozo de cada uno de los cubos que le rodean (la magnitud de ese trozo de cubo, el solapamiento, se proporcionará como parámetro al crear el objeto de la clase GalaxyDetector), evitando así que aparezcan galaxias cortadas (Fig.2.7). En realidad esta solución no evita que se corten galaxias, ya que eso sería imposible. Lo que asegura es que las galaxias aparezcan completas en una imagen como mínimo, es decir, que la red neuronal vea como mínimo una vez a todas las galaxias con su forma completa, sin que se pierda nada de información. Para evitar confundir a la red neuronal, las galaxias situadas en la parte solapada no figurarán en el catálogo de cada imagen pequeña.

Solucionado el problema de los bordes, el siguiente paso es generar una máscara para cada minicubo de datos. Inicialmente la idea era asignar 0 a todo lo que no fuese galaxia y 1 al resto. Sin embargo, para facilitar el entrenamiento de la red se ha añadido una etiqueta más, los bordes de las galaxias, con un valor 2. La distinción entre borde e interior se hará justo después de generar la máscara de la galaxia.

Para que un píxel sea considerado como borde se deben cumplir las siguientes condiciones:

1. El píxel que se quiere evaluar tiene valor 1, es decir, no forma parte del fondo.
2. La suma de los valores de los 8 píxeles que le rodean es inferior a 8 (Fig2.5).

Esta modificación, además de facilitar la labor de la red, facilita también la identificación de la forma de galaxias que se solapen entre sí, como se comentará más adelante.

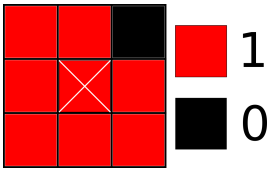

Figura 2.5: Ejemplo de píxel que se considera como borde, puesto que la suma de los píxeles que le rodean es inferior a 8.

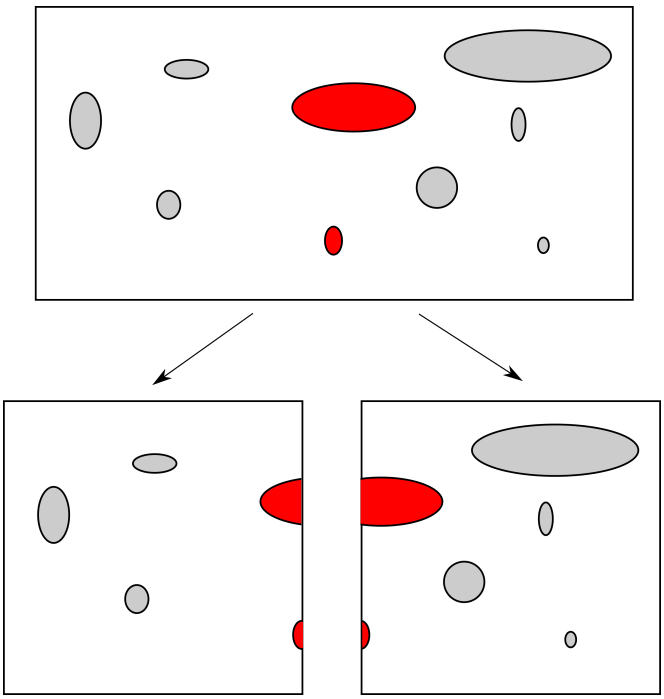

Figura 2.6: Representación gráfica del problema de los bordes. Cualquier galaxia que se sitúe cerca de donde se va a dividir el cubo quedará partida en dos, como le sucede a las galaxias rojas de la imagen.

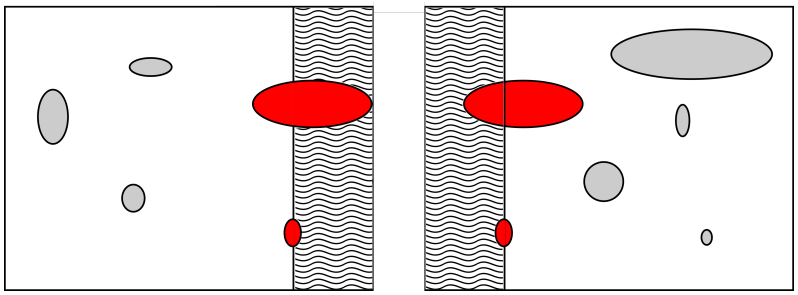

Figura 2.7: Solución al problema de los bordes. Consiste en aumentar el tamaño del minicubo con parte del siguiente cubo (parte ondulada). De esta forma, las galaxias que originalmente estaban cortadas, podrán ser vistas completas al menos una vez por la red neuronal. Es importante considerar este solapamiento en las cuatro direcciones, puesto que el *input* de la red neuronal debe ser un vector de imágenes con la misma dimensión.

Una vez que se han generado las máscaras, el siguiente paso es entrenar a la red neuronal. Para ello se destinará una fracción del vector con las imágenes pequeñas, además de otro vector, de la misma dimensión, con las máscaras generadas. Teniendo el modelo entrenado, ya solo falta hacer predicciones. Así, se llamará a un método de la clase principal, a través del objeto creado previamente, pasándole como parámetro la imagen sobre la que se quiere aplicar la red. Entonces, desde la clase principal, se llamará a un módulo que pinte una gráfica, comparando la imagen, la máscara real (si la hubiere) y la predicha por la red. Este método le devolverá al usuario el catálogo con las fuentes que haya detectado. Finalmente, la clase *GalaxyDetector* contará con un método estático que permita iniciar el proceso de *scoring*. Para ello, tomará como parámetros los dos catálogos, el predicho y el original.

Los algoritmos utilizados en cada uno de los módulos, así como los resultados intermedios que produzcan, se desarrollarán en detalle en la siguiente sección.

3 Resultados: código y *outputs*

Una vez la estructura del código está clara, como muestra la Fig.2.4, el siguiente paso es construir el código propiamente dicho, así como comprobar que está produciendo los resultados esperados. El último paso será probar varias arquitecturas de red neuronal, comparando los resultados que generen. El código completo del proyecto está disponible en el repositorio de *Github*[35]. Las instrucciones para su instalación se resumen en la última sección del apéndice. Todo el código presentado en esta sección ha sido creado durante el trabajo, salvo alguna librería en la que se indique lo contrario.

Para facilitar su seguimiento, esta sección se dividirá en dos partes: código pre-red neuronal y código post-red neuronal. Todas las imágenes presentadas en esta sección como ejemplo o comprobación del código, han sido obtenidas con los parámetros de la Tab.3.1, salvo en casos puntuales que se indique lo contrario. [4]

Tabla 3.1: Algunos de los parámetros utilizados para la generación del resto de imágenes de esta sección. Para el resto de parámetros se usan los establecidos por defecto.

Parámetro	Valor
dim	64
flujo_min	12.5
reduction_function	"mean_frec"
overlapping	"auto"
fraction_for_training	0.8
threshold	0.035
batch_size	32
epochs	200
filts	"auto"
architecture	"first_model"

3.1 Código pre-red neuronal

Como ya se ha comentado en la sección anterior sobre el método experimental, el *input* de la red neuronal no puede ser la imagen captada por el telescopio directamente, requiere de un tratamiento previo. Además, la red neuronal necesita de una serie de máscaras para el proceso de entrenamiento y validación. Por lo tanto, antes de empezar a buscar arquitecturas de redes convolucionales, es conveniente preparar el *dataset*.

[4] Para información acerca de los parámetros se recomienda consultar la documentación. En el apéndice se explica cómo visualizarla.

3.1.1 Modificaciones previas: división del cubo de datos y solapamiento

Una vez se ha conseguido el cubo (en realidad cuadrado) de datos definitivo, después de llevar a cabo la reducción de dimensión, el primer paso del *pipeline* es dividirlo en una lista con cubos más pequeños. Esta maniobra permitirá aumentar mucho el tamaño de la muestra, haciendo más eficiente el entrenamiento de la red.

Lo primero es transformar el cubo de datos (y el catálogo) a un formato que permita un manejo sencillo de estos. Para ello, se ha utilizado la función *DataFrame* del paquete *pandas* [41], donde los índices de la tabla son las coordenadas verticales (declinación) y los nombres de las columnas serán las coordenadas horizontales (ascensión recta). Sin embargo, para facilitar el acceso a la información de la tabla de *pandas* se ha prescindido de las coordenadas en formato grados decimales. La explicación de esta modificación es muy sencilla, pues es mucho más cómodo acceder a los datos cuando el nombre de la columna es el número del píxel (0, 1, ..., n) que cuando es un *string* extraño (ya que *pandas* no admite números con decimales en los nombres de filas y columnas, a no ser que se traten como *string*). Por lo tanto, para acceder a cualquier fila o columna basta con ejecutar un comando como el de Cód.6

Código 6: Comando para acceder a los datos de un *DataFrame* (data), previamente creado, de *pandas*. En este caso, selecciona todas las columnas de las filas 1 y 2 (en *python* no se cuenta el extremo derecho del *slice*)

```
import pandas as pd
row_slice = data.iloc[1:3, :]
```

Respecto al catálogo, se ha procedido de una forma similar, convirtiendo el fichero de texto en un *DataFrame*, donde los nombres de las columnas son los nombres de los parámetros (la primera fila del fichero de texto: RA, Dec...) y los índices se corresponden con la primera columna del fichero de datos, esto es, con el número de identificación del objeto.

El siguiente paso representa otra de las simplificaciones que se han llevado a cabo, con el objetivo de reducir la complejidad del problema, hasta un nivel asequible para un trabajo de final de grado. Consiste en eliminar del catálogo las fuentes con un flujo integrado a todas las frecuencias (parámetro del catálogo original, medido en JyHz) menor que una cierta cantidad, proporcionada como parámetro al crear el objeto. La justificación de esta aproximación está en la sec.1.4.2, donde se hace mención a un tipo de fuentes de HI que no tienen emisión en el continuo, y que por lo tanto, no tienen valores significativos de intensidad en el cubo de datos del continuo. El problema reside en que, las máscaras generadas le van a decir a la red neuronal que ahí hay una fuente, aunque en el cubo de datos no se pueda distinguir. Esto puede provocar confusión en la red neuronal, por lo que se ha optado por eliminarlas. Al fin y al cabo, es mejor una red que detecte pocas fuentes pero que lo haga con una mayor seguridad. La pregunta clave es ¿a partir de que flujo aparecen únicamente las fuentes con representación en el continuo?

Pues no hay una respuesta segura, ya que el catálogo original no proporciona la suficiente información, por lo que depende del *dataset*. En la Fig.3.1 se muestra un histograma con el número de galaxias y otros objetos, en función de su flujo integrado a todas las frecuencias. Destaca la gran cantidad de objetos con flujo en el intervalo 0-20 JyHz, que habitualmente corresponden a cuerpos relativamente pequeños, sin emisión en el continuo. Un valor que se ha demostrado razonable para este flujo mínimo es 12.5 JyHz. Este flujo, además de eliminar la mayor parte de las galaxias emisoras de HI únicamente, permite que haya un número suficiente para poder entrenar la red neuronal (Fig.3.2). Aun así, el número de galaxias decrece en un orden de magnitud (escala logarítmica), por lo que habrá una cierta pérdida de efectividad de la red, que no quedará otra que asumir.

Antes de dividir el cubo de datos es conveniente llevar a cabo una última modificación. Trabajar utilizando píxeles para nombrar filas y columnas, pero que las coordenadas de los cuerpos del catálogo estén en grados o segundos de arco, puede resultar contraintuitivo. Por lo tanto, se ha optado por uniformizar todos los sistemas de unidades, convirtiendo los valores de las magnitudes (RA, Dec y hi_size) a píxeles. Se ha optado también por convertir los ángulos (inclinación y el ángulo de posición) a unidades del sistema internacional, esto es, a radianes. Esto se ha conseguido gracias a la clase WCS (*World Coordinate System*), del paquete *astropy*, que permite crear un objeto que contenga toda la información de la imagen (resumida en el encabezamiento del fichero FITS). A través de este objeto es posible realizar transformaciones entre píxeles y coordenadas, y viceversa, de una forma muy sencilla.

Para más información se puede consultar la documentación de esta clase [42].

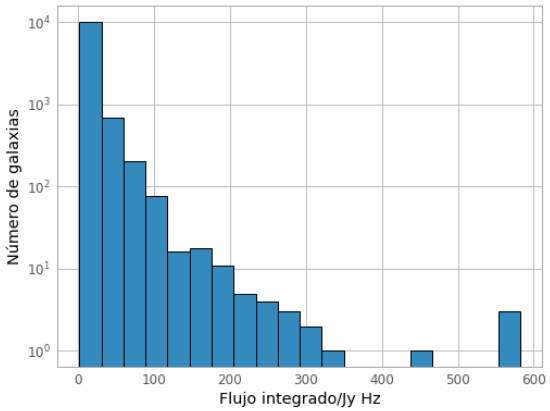

Figura 3.1: Histograma del número de galaxias en función de su flujo integrado, en JyHz, para el catálogo original, en escala logarítmica.

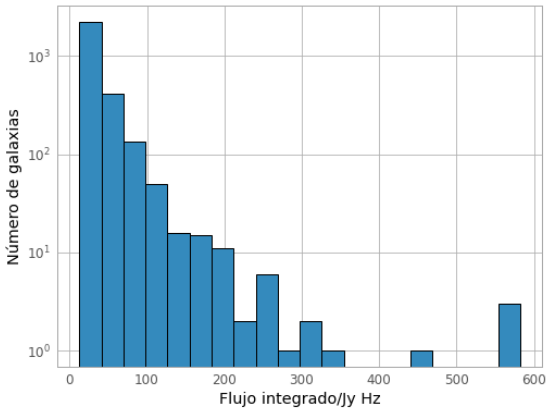

Figura 3.2: Histograma del número de galaxias en función de su flujo integrado, en JyHz, para el catálogo sin las galaxias de menos de 12.5 JyHz, en escala logarítmica.

Una vez el catálogo y el cubo de datos están listos, ya se puede dividir en cubos más pequeños. Para ello, se ha programado el módulo *data_splitting.py*, que interacciona con la clase *GalaxyDetector* a través de la función *split_into_small_boxes*. Esta función toma como parámetros el cubo de datos en dos dimensiones, el tamaño deseado de las mini-imágenes, el catálogo y el *overlapping* (o solapamiento). Su funcionamiento se describe en el diagrama de actividad (*activity diagram*), en la Fig.3.5.

El primer problema surge a la hora de dividir el cubo en "X" figuras iguales, cuyo tamaño se proporcione a través de un parámetro. Como resulta evidente, las probabilidades de que el número de píxeles de la imagen original sea divisible por el tamaño deseado de imágenes son ínfimas. De hecho, curiosamente la imagen pequeña tiene 643 píxeles, por lo que solo se podría dividir en 1 o 643 filas (643 es un número primo). Por tanto, es necesario incrementar el tamaño de la imagen hasta que sea divisible por el tamaño de las imágenes pequeñas. Una forma muy extendida de solucionar este problema es "reflejando" la imagen hasta conseguir un resto cero en la división. La parte reflejada se añadirá al final de cada imagen, como continuación de la última fila o columna de píxeles. De esta forma, se evita estropear el entrenamiento de la red neuronal, rompiendo la continuidad de la imagen.

Todo bien hasta aquí, cuando el bucle llegue a la última fila (o columna, si es el segundo bucle) incorporará a la lista de cubos de datos divididos las imágenes con la parte reflejada. Sin embargo, surge un problema cuando el solapamiento es mayor que el número de píxeles que sobran, es decir, que el solapamiento de la penúltima fila involucra parte de la imagen reflejada. Por este motivo, para tratar de forma diferente los dos casos, se evalúa en el primer *if* de la Fig.3.5 qué función se debe utilizar. Además, existe una tercera opción, que la imagen no necesite espejado porque su longitud es divisible por el tamaño de las imágenes pequeñas.

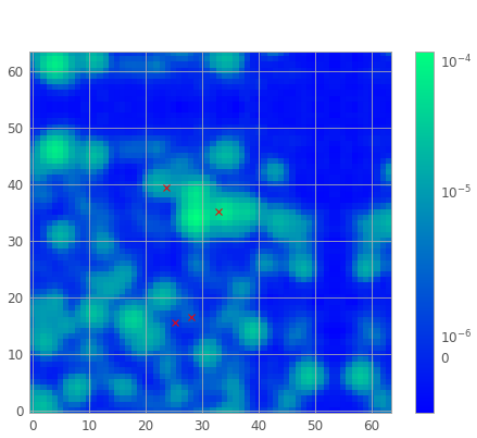

Figura 3.3: Ejemplo con uno de los mini-cubos de datos que requieren de una ampliación de tamaño, realizada a través del reflejo de la propia imagen. En este caso hay una doble reflexión, puesto que se trata de la esquina superior derecha. El tamaño del cubo original era de aproximadamente 50x50, por lo que se han añadido en torno a 15 píxeles de imagen reflejada.

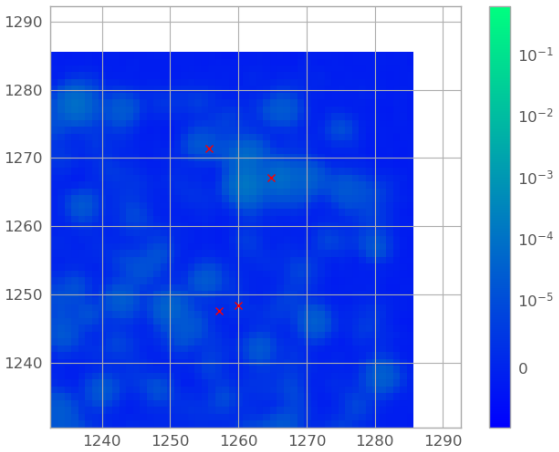

Figura 3.4: Zoom de la imagen original, representando la misma región que la Fig.3.5. Esta imagen se presenta a modo de verificación del algoritmo. La parte blanca (sin datos) representa la zona reflejada en la imagen de al lado.

A continuación se va a explicar el algoritmo seguido para el caso en el que el número de píxeles que sobren sea mayor que el solapamiento, puesto que es lo que sucede para los parámetros de la Tab.3.1. En primer lugar, un *loop* recorre las filas, seleccionando en cada iteración una fila de (tamaño imagen pequeña - 2 · solapamiento) píxeles. En cada una de estas iteraciones se actualiza el valor de la variable "multi_row". Cuando se está trabajando con una fila situada en el borde de la imagen, el solapamiento difiere respecto al resto de filas. En lugar de considerar el mismo solapamiento en la parte de arriba y abajo de la fila, como sería lo habitual, se considera un doble solapamiento en la parte que no da al borde de la imagen. De esta forma, es posible mantener un mismo tamaño para todas las imágenes, aunque estas estén situadas en un borde. El parámetro "multi_row"no es mas que el número por el que hay que multiplicar al solapamiento. Lo mismo sucede, pero aplicado a las columnas (derecha e izquierda, en lugar de arriba y abajo), con el parámetro "multi_col". Una vez se ha actualizado este valor, el siguiente paso es seleccionar el *slice* de filas, es decir, todas las columnas de un número de filas igual al tamaño de la imagen.

A continuación, un bucle, con longitud igual al número de imágenes en cada dirección (divisor de longitud total entre tamaño imagen pequeña), recorre las columnas de este *slice*. Dependiendo de la localización del futuro *slice*, se actualiza el valor de "multi_col" y se selecciona el *slice* vertical del *slice* horizontal. El resultado de cada iteración del segundo bucle es una imagen del tamaño deseado. A partir de los índices de esta imagen (recordar que los índices se corresponden con las coordenadas en píxeles), se seleccionan del catálogo las fuentes que se incluyan dentro de la imagen creada (salvo las que se localicen en la parte solapada, como ya se ha comentado). Cuando la lista con los catálogos y las fuentes esté actualizada con todos los que forman la fila, se repite el proceso para la siguiente tanda de filas, hasta terminar en el fondo de la imagen.

Si se diera el caso en el que el solapamiento fuese mayor que el número de píxeles sobrantes, únicamente cambiaría el tratamiento de la penúltima fila y de todas las penúltimas columnas, ya que en su solapamiento estaría incluida una parte de la imagen reflejada. Finalmente, en el improbable caso de que no quede ningún píxel desperdigado, se elimina toda la parte del código relacionada con la reflexión de la imagen. El resto del algoritmo es esencialmente el mismo.

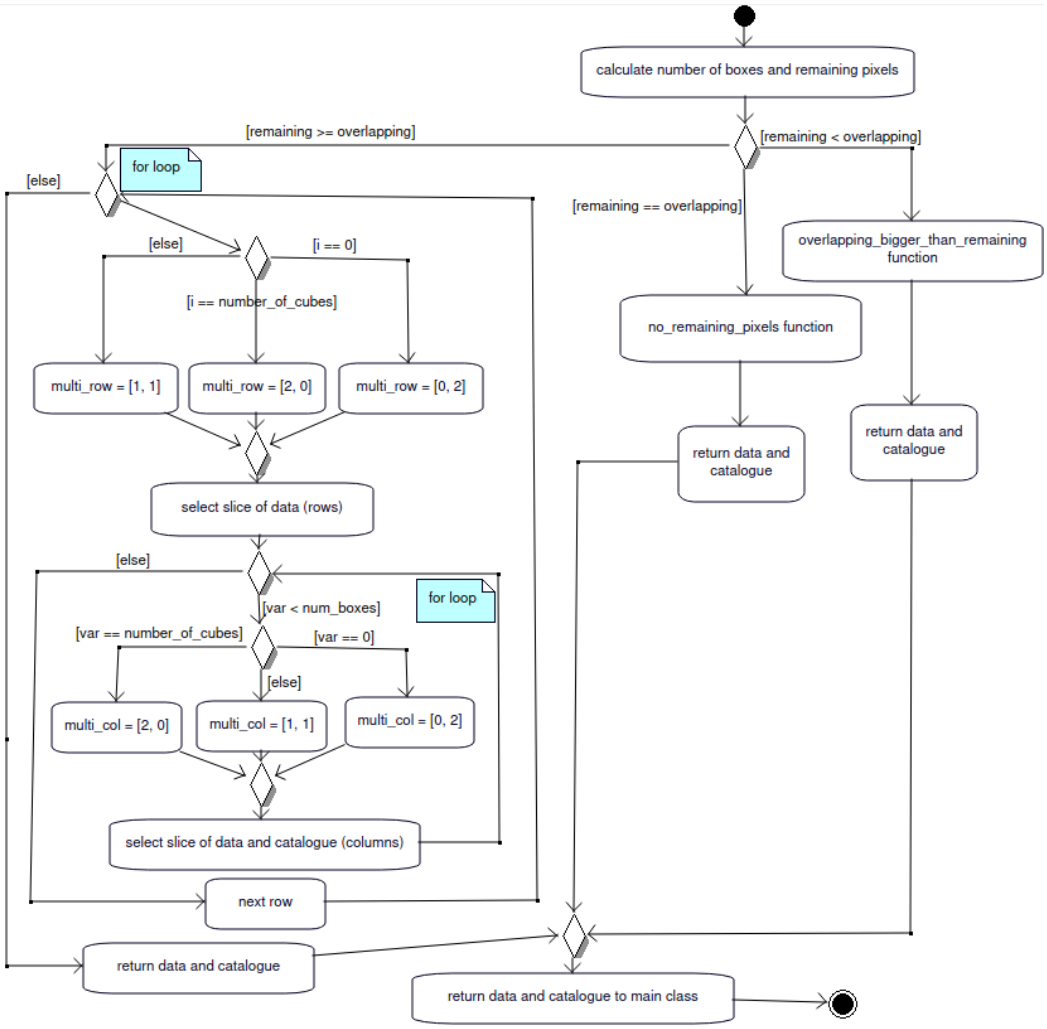

Figura 3.5: *Activity diagram* representando la estructura de la función split_into_small_datacubes, perteneciente al módulo data_splitting. En este tipo de diagramas, los rombos blancos representan nodos donde hay una decisión. El camino que sigue el programa depende de las condiciones que se presentan entre dos corchetes. Algunos de estos rombos pueden representar un bucle *for*, como los dos señalados con las notas azules. Al fin y al cabo el bucle *for* se puede entender como un condicional. Solo se presenta con detalle para una sola condición para asegurar su legibilidad. El algoritmo seguido en las otras dos funciones difiere ligeramente, pero la base es similar.

3.1.2 Creación de las máscaras. Condición de borde de galaxia

Una vez la función *split_into_small_boxes* ha devuelto una lista con las imágenes pequeñas y otra con sus respectivos catálogos (en el mismo orden, la imagen 0 tiene asociado el catálogo 0, la 1 el catálogo 1...), el siguiente paso consiste en generar una máscara para cada una de estas imágenes, a partir de los datos de su catálogo.

Para ello, se ha diseñado la función *generate_galaxies*, perteneciente al módulo *ellipse_generation*. En ella, un bucle recorre todas las fuentes del catálogo reducido (el que se corresponde al minicubo de datos), generando una elipse con valor 1 en su interior y 0 en el resto.

Sin embargo, ha sido necesario llevar a cabo una serie de pequeñas transformaciones a los datos, antes de generar la elipse que simule la galaxia:

1. Ajustar la orientación (ángulo de posición) de las fuentes en la parte reflejada. Para ello, se incrementa en $\pi/2$ el ángulo, si la galaxia ha sido reflejada una vez, y π si lo ha sido dos veces. Esto se consigue comparando sus coordenadas ((RA, Dec), en píxeles) con la longitud del cubo de datos original.

2. Sobrescribir el valor de "hi_size" con el valor del semieje mayor. Esto es necesario puesto que

"hi_size" representa el **diámetro** de la galaxia, no su semieje, por lo que se divide entre dos.

3. Calcular el semieje menor, a través de la ec.4, utilizando el recién calculado semieje mayor, además de la inclinación i.

Corregidos los datos, el siguiente paso es generar una elipse que simule cada una de las galaxias. Con el fin de ahorrar tiempo de programación, se ha optado por utilizar una función ya creada y probada, como es *Ellipse2D*[5], de la librería *astropy* [43].

El último paso, antes de dar por finalizada la generación de la máscara, es distinguir el borde de las galaxias, es decir, los píxeles que la delimitan, como ya se adelantaba en la Sec.1.5.2. La condición que se ha utilizado es la misma que se escenificaba en la Fig.2.5, esto es, para que un píxel sea considerado como borde, los ocho píxeles que lo rodean deben sumar menos de ocho (además de tener el píxel en cuestión valor uno[6]). Actualizadas las etiquetas de la máscara, solo queda actualizar el valor de una variable que guarde la máscara completa con la máscara recién creada.

Aunque pueda parecer que este añadido no tenga mucho sentido en galaxias pequeñas, especialmente en aquellas formadas por menos de 5 píxeles (ya que todos sus píxeles son considerados borde) se ha probado útil en varias situaciones. Una de ellas se dá cuando dos galaxias se solapan, es decir, que sus elipses se sobreponen. En este caso, se le ha dado prioridad al borde sobre el interior, esto es, se puede distinguir el contorno de una galaxia que pase por encima de otra.

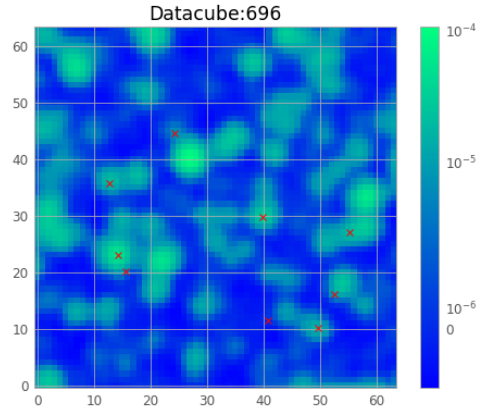

Figura 3.6: Mini-cubo de datos 696, con el catálogo sobreimpresionado (únicamente las fuentes con flujo superior a 12.5 JyHz). Se observa como prácticamente todas las fuentes tienen emisión en el continuo, salvo la situada en las coordenadas (40, 10), que es claramente fuente de HI únicamente

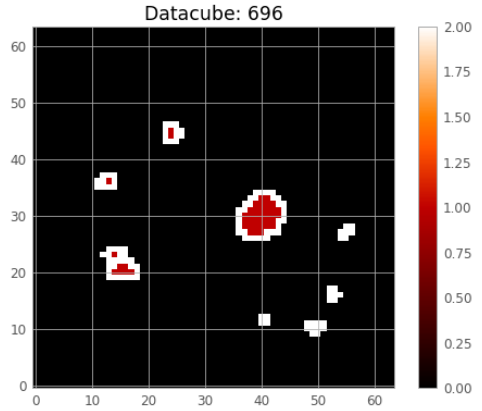

Figura 3.7: Máscara final del mini-cubo de datos de la Fig.3.6. La parte más interesante de esta máscara son los objetos situados en torno a (20, 15). En la máscara con los bordes como etiqueta es posible diferenciar claramente las dos galaxias, mientras que si se tratase de una máscara con únicamente dos etiquetas, 0 y 1, confundiría a la red neuronal.

3.2 Código post-red neuronal

Conseguidas las imágenes y sus etiquetas, ya solo falta crear un modelo de red neuronal y entrenarlo. Aunque parezca que la parte más complicada ya ha finalizado, en realidad no ha hecho nada más que empezar. El problema de las redes neuronales es que no hay una regla mágica que te diga qué red y con qué hiperparámetros entrenarla, por lo tanto es necesaria una intensa labor de documentación sobre posibles arquitecturas a implementar, identificando las típicas para problemas de segmentación.

[5]Nota: el ángulo de posición que esta función toma como parámetro empieza se mide en sentido antihorario, empezando por el eje x positivo. Sin embargo, el polo norte celeste de la imagen (la dirección a partir de la cual se empieza a contar el ángulo de posición) se sitúa en la dirección positiva del eje y. Por todo ello, el ángulo que recibe como parámetro la función se ha modificado, incrementando su valor $\pi/2$ radianes.

[6]Remarcar que el programa solo busca bordes en la región donde está la galaxia, de forma que el tiempo de ejecución del programa se reduce drásticamente.

Hay un paso previo al entrenamiento de la red que conviene resaltar. Es la normalización de los datos, esto es, cambiar el intervalo original de valores (sea el que sea), a un intervalo [0, 1]. Está ampliamente demostrado que las redes neuronales funcionan mejor con valores entre 0 y 1, pudiendo extraer de forma más eficiente sus características. De hecho, en este propio problema fue imposible encontrar una arquitectura capaz de detectar, no ya unas pocas galaxias, sino algo directamente, si no se normalizaban los datos. Curiosamente, la red era incapaz de diferenciar las imágenes, todas las imágenes generaban el mismo *output*. Por lo tanto, se han probado diversas opciones de normalización del *dataset*, desde dividir por el número más grande a incorporar una capa de normalización en la arquitectura de la red. Los resultados se presentan más adelante, en la Sec.3.5.

3.2.1 Arquitectura, hiperparámetros y entrenamiento de la red neuronal

Creado ya el objeto de la clase *GalaxyDetector*, el método *solve_NN* permite entrenar una arquitectura de red neuronal, cuyo nombre se facilita como parámetro (además del resto de hiperparámetros de la red). Después de dividir el *dataset*, dedicando una fracción de este (proporcionada como parámetro) al entrenamiento y otra a la validación de la red, utiliza alguna de las funciones disponibles del módulo *models*. Estas funciones no son más que arquitecturas ya programadas, que se convierten en modelos una vez reciben sus parámetros de la clase principal. Con el fin de comprender el proceso de construcción se ha diseñado una arquitectura desde cero, basada en la idea original de Olaf Ronneberger, pero adaptada a una imagen en escala de grises (sin tercera dimensión, como sería el caso de las imágenes en RGB) y con tres etiquetas. Para más detalles sobre el funcionamiento o el diseño de esta arquitectura se puede consultar el artículo original [29].

A grandes rasgos, la red está dividida en dos mitades. La primera, capaz de extraer las características (o *features*) de cada una de las imágenes, está formada por varios niveles (concretamente cinco en este caso). Cada uno de estos niveles está compuesto por dos capas convolucionales, para extraer las *features*, y una *max-pooling* para reducir la dimensión de la imagen a la mitad. Otra de las modificaciones de la arquitectura original aparece a la hora de fijar el tamaño del *kernel* de las capas convolucionales. Según el artículo [28], es recomendable aumentar el tamaño del *kernel* en las primeras capas de modelos que se enfrenten a imágenes con mucho ruido. Como en este caso no se ha aplicado ningún algoritmo para reducir el ruido de la imagen, se ha optado por ajustar el tamaño del *kernel* a (11x11), pero solo para las dos primeras capas (para todo el resto, se mantiene en (3x3)). Finalmente, el *output* que proporciona cada nivel sirve como *input* para el siguiente, hasta que las dimensiones de la imagen original se han reducido en 1/8. Esto es debido a que, por diseño, la última de las 5 capas no reduce la dimensión del mapa.

La parte más interesante de la arquitectura es la otra mitad de la "U", donde se construye la máscara final. En cada nivel de esta parte hay dos capas convolucionales, además de una "convolucional traspuesta" y una de concatenación. Es la capa traspuesta la que primero actúa, duplicando la dimensión de la imagen original. A continuación, el *feature map* de la parte opuesta de la "U" entra en juego, concatenándose a la imagen recién duplicada. De esta forma, se produce una interacción entre las dos partes de la "U". Finalmente, dos capas convolucionales completan cada uno de los cuatro niveles del camino de expansión.

Sin embargo, falta transformar este *output* en los tres mapas de probabilidades que uno esperaría obtener. Para ello, se necesita una capa que devuelva un mapa con dimensiones (pix, pix, 3), donde pix es el número de píxeles de la imagen inicial, en este caso 64 píxeles. La solución pasa por generar un último nivel con una capa convolucional normal (como la del resto de la arquitectura), seguida de una capa de activación. La función de activación de esta capa es la clave para conseguir un resultado satisfactorio, pues es la que va a decidir qué etiqueta tiene cada píxel. Como se comentó en la Sec.1.5.3, la función apropiada para problemas de segmentación con más de dos etiquetas es la *softmax*, por delante de la *sigmoid*, ideada para problemas con dos únicas etiquetas.

Los detalles de esta arquitectura, con los filtros, las dimensiones de los *input* y *output* de cada capa y las relaciones entre niveles, se encuentran resumidos en la imagen *model_scheme.png*, en la carpeta *imgs* del proyecto (debido a su tamaño, no ha sido posible incluirla en este documento).

Además de esta opción, se ha habilitado la posibilidad de utilizar las arquitecturas del módulo *keras-unet-collection* [34], creado por la comunidad. En él están disponibles hasta 10 evoluciones de la arquitectura original propuesta por O. Ronneberger por primera vez. El método *solve_NN* permite

utilizar estas arquitecturas con los hiperparámetros facilitados al llamarlo a través del objeto, de forma que es posible probar diferentes modelos de estas arquitecturas, comparando sus resultados.

Una vez construido el modelo, el siguiente paso es compilarlo, a través del método *compile* de *tensorflow*. El método permite elegir el optimizador, la función de pérdida y la métrica que se va a utilizar durante el ajuste del modelo a los datos. Después de compilarlo, solo queda proporcionarle al método *fit* de *tensorflow* las imágenes y sus correspondientes etiquetas, para que inicie el entrenamiento de la red neuronal, es decir, que ajuste los pesos $(w_1...w_n)$ para que la función de pérdida se minimice. Para este paso, que se ejecuta inmediatamente después de compilar la red, se ha diseñado un "parámetro"que permite finalizar el ajuste cuando la métrica deje de mejorar, ahorrando tiempo de computación. Para ello, se ha utilizado la clase *EarlyStopping*, perteneciente a la librería *tensorflow*.

Finalmente, el método pinta en una gráfica la evolución de la pérdida en el entrenamiento, así como en la validación (*training* y *validation loss*) durante las épocas en las que se desarrollan. Esta gráfica permite identificar, o al menos hacerse una idea, del rendimiento del proceso de ajuste a los datos. Normalmente, es una de las primeras herramientas que se utilizan para identificar un posible *overfitting* de la red, hecho que ocurre cuando la curva de validación está muy por encima de la de entrenamiento. En cambio, si ambas curvas tienen valores elevados, se está produciendo *underfitting*. Es vital conseguir que las dos curvas de esta gráfica mantengan valores reducidos, y lo más parecidos posibles entre ambas. En la Fig.3.8 se muestra un ejemplo de modelo bien ajustado, obtenido con los parámetros de la Tab.3.1.

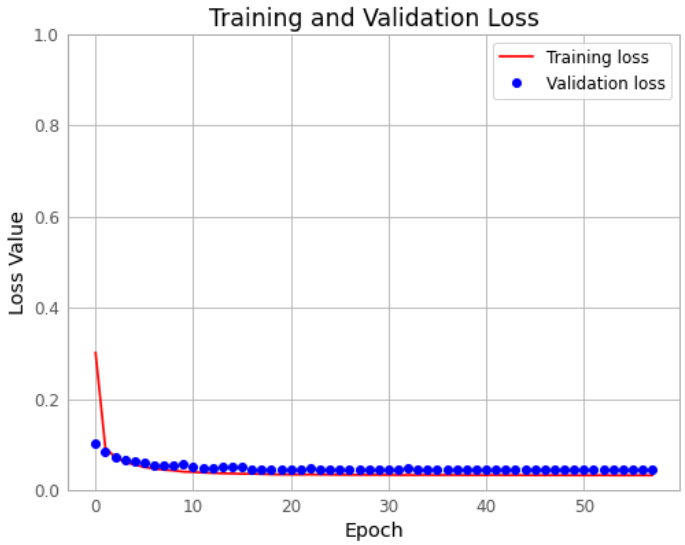

Figura 3.8: Gráfica con la evolución de la pérdida en el entrenamiento y en la validación del modelo descrito en la Tab.3.1

3.2.2 Detección post-entrenamiento

En esta parte del código creado se pone a prueba a la red neuronal, proporcionándole una imagen con el objetivo de generar una máscara, a partir de la cual se pueda extraer un catálogo de fuentes. Esta es la función del método *prediction* de la clase *GalaxyDetector*.

Una vez recibe la imagen como parámetro, preferiblemente del mismo tamaño ya que así se evita tener que reescalarla, llama al método *predict* del paquete *tensorflow*, que devuelve una lista con tres "mapas"(una por cada una de las tres etiquetas)[7]. Estos tres mapas asignan una probabilidad a cada píxel en cada uno de los tres mapas.

En las Fig.3.11 y la Fig.3.12 se presenta el resultado de aplicar el método *prediction* a la imagen representada en la Fig.3.9. Dependerá del valor de la variable *threshold*, que se proporciona como parámetro, determinar qué píxeles se consideran galaxias y cuáles fondo.

[7]Nota: es necesario ejecutar el método de la sección anterior, el encargado de entrenar a la red neuronal, para poder hacer predicciones.

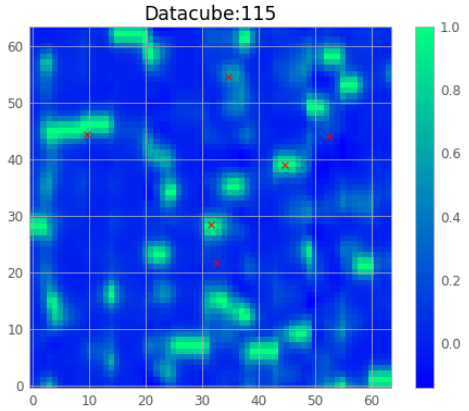

Figura 3.9: Imagen del cubo de datos número 115, del archivo de datos reducido. Se ha normalizado a la escala [0, 1]

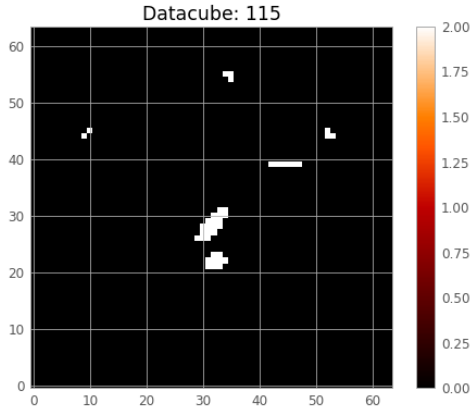

Figura 3.10: Máscara del cubo de datos de la Fig.3.9

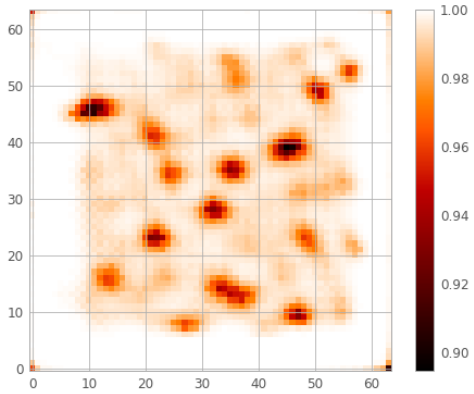

Figura 3.11: Mapa con las "probabilidades" de que cada píxel **no** forme parte de ninguna galaxia, es decir, con etiqueta 0, para la imagen Fig.3.9. Los píxeles más oscuros son los que tienen una mayor probabilidad de pertenecer a una galaxia.

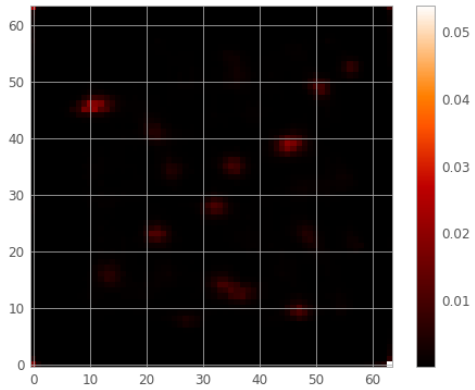

Figura 3.12: Mapa con las "probabilidades" de que cada píxel forme parte de alguna galaxia, es decir, con etiqueta 1, para la imagen Fig.3.9. Los píxeles más claros tienen una mayor probabilidad de pertenecer a una galaxia.

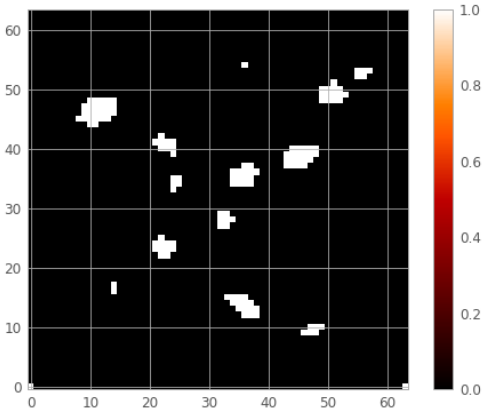

Figura 3.13: Máscara obtenida a partir del mapa de la Fig.3.11, aplicando un umbral igual a 0.035.

Por ejemplo, un valor de *threshold* de 0.035 genera la máscara de la Fig.3.13. Si bien es cierto que el modelo es capaz de detectar varias galaxias, las situadas en las coordenadas (10, 45), (30, 30), (45, 40) y (35, 40), es necesario ajustar más los parámetros y probar con más arquitecturas. En la Sec.3.5 se presentan los resultados de varias pruebas con diferentes parámetros y arquitecturas.

3.3 Resumen del código realizado

En la Fig.3.14 se presenta el diagrama de clases, con todas las clases/funciones creadas desde cero para resolver el problema, además de sus atributos/parámetros más característicos. Los parámetros más específicos de las funciones están explicados con detalle en la documentación general del proyecto, generada a través del paquete *sphinx*[44].

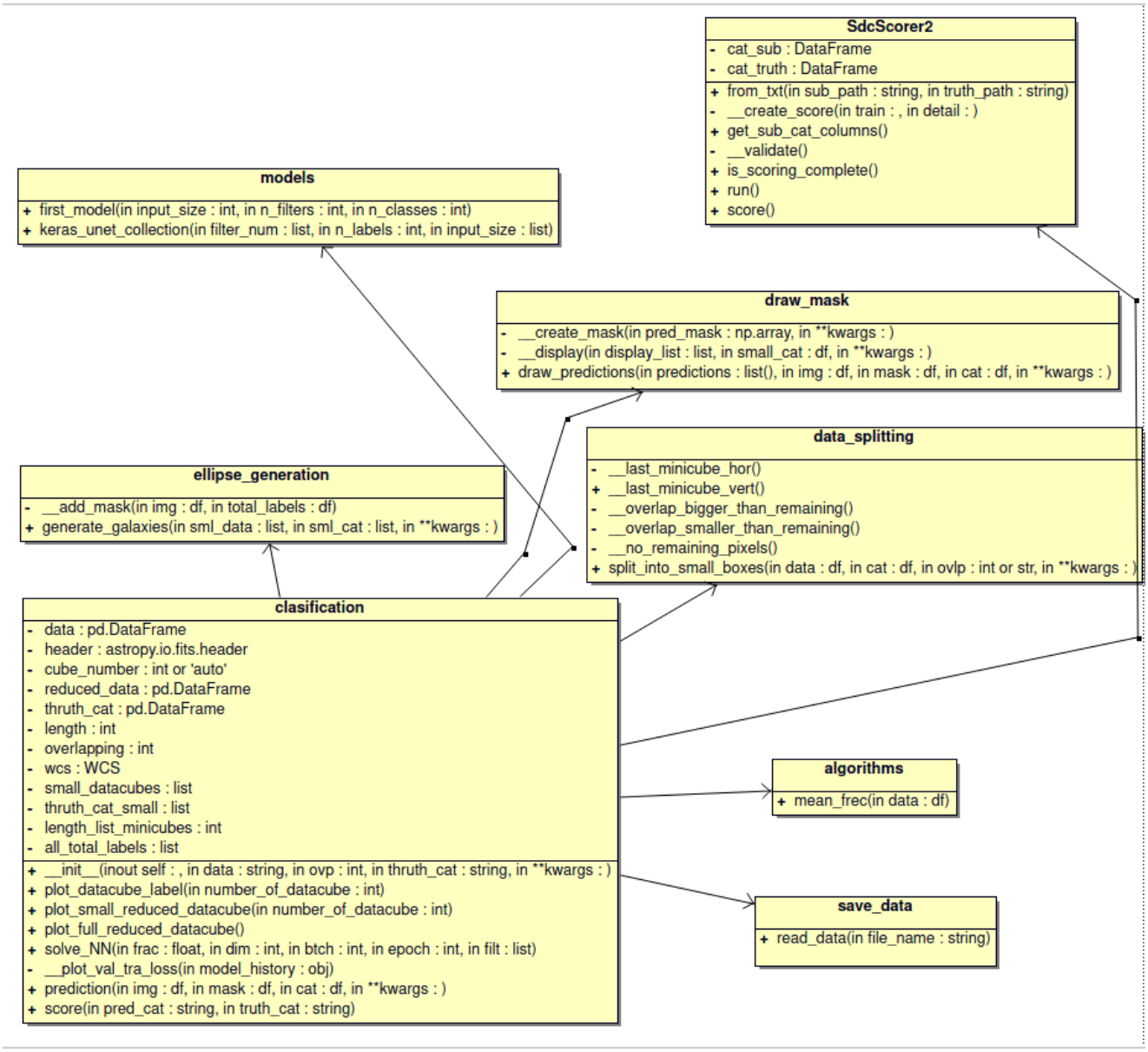

Figura 3.14: *Class diagram* representando la estructura del código generado. Dentro de cada cuadrado (clase o módulo) se sitúan los atributos, que aparecen precedidos por un signo menos, y las operaciones (o funciones), que lo hacen por un signo más (salvo que sean privadas). Las flechas indican la relación de dependencia de la clase principal con el resto de módulos creados.

3.4 *SExtractor* para identificar fuentes en la máscara

Es fácil para cualquier persona distinguir las galaxias que aparecen en la máscara de la Fig.3.13, pero no lo es tanto para un ordenador. Para realizar está labor, se ha utilizado un *software*, especializado en

detección de galaxias a partir de imágenes o máscaras, llamado *SExtractor* [45]. Este programa permite traducir la máscara a un catálogo de fuentes, devolviendo su posición (ascensión recta y declinación, en píxeles), sus dimensiones (semieje mayor, en píxeles) y su orientación (ángulo de posición, en grados). El único ligero inconveniente es el formato de los *input* de este programa, puesto que no tiene un paquete o librería adaptado a *python*. La única forma de ejecutar este programa es a través de un fichero FITS, por lo que se han convertido cada una de las tablas (en formato *DataFrame*) a través de la clase *PrimaryHDU* del paquete *astropy*. De esta forma, todas las imágenes se han traducido a un archivo FITS, con la misma estructura que la imagen inicial.

El programa se ha ejecutado directamente desde *python*, a través de la librería *os*, que permite correr *scripts* de *bash* directamente desde el código. El comando (o más bien la línea de código) para este fin se presenta en Cód.7.

Código 7: Comando para correr el *software* SExtractor en *python*. La segunda parte del comando, después de la *keyword* para lanzar el programa (sex), representa la ruta a la imagen en formato FITS. Por otra parte, la última ruta representa la ruta al archivo con la configuración, donde se define el comportamiento del *software*

```
import os
os.system('sex src/output/mask'+str(img_num)+'.fits -c src/output/daofind.sex')
```

Para conocer el formato de archivo de configuración se puede consultar el ejemplo, salvado en la ruta "src/output/daofind.sex", dentro del proyecto.

3.5 Comparación de diferentes arquitecturas

En esta sección se presentan los resultados obtenidos en la detección de galaxias, en la totalidad del cubo de datos más pequeño, utilizando el mediano para el entrenamiento de la red. Antes de empezar, conviene recordar que el cubo de datos utilizado para el entrenamiento apenas es capaz de generar unas 400-500 imágenes pequeñas, con sus correspondientes máscaras. De esta forma, todos estos modelos están probados con un *dataset* relativamente reducido. A modo de comparación, el *dataset* "MNIST"[46], famoso por sus imágenes de números del cero al nueve (ampliamente utilizado para aprender a programar redes neuronales), está formado por 60.000 imágenes para el entrenamiento y 10.000 para la validación. Para probar correctamente el algoritmo, sería ideal contar con la capacidad de ejecutar el programa con el *dataset* de 1 TB, pero como ya se ha comentado, escapa a los objetivos del trabajo. De hecho, el objetivo de esta sección es hacer una comparación relativa entre arquitecturas, sin prestar especial atención al valor absoluto de detecciones.

Respecto al sistema de puntuación, a pesar de estar comprobado su funcionamiento, no se ha podido utilizar el *software* de SKA para calificar las arquitecturas. Desgraciadamente, para calificar un algoritmo necesita **todos** los parámetros del catálogo inicial (la red neuronal únicamente proporciona coordenadas bidimensionales, orientación y tamaño). Las comparaciones se van a realizar de una forma algo menos sofisticada, que consiste en utilizar la clase *SkyCoord* [47], del paquete *astropy*. Esta clase permite convertir un simple par de vectores de *numpy* (o de columnas de *pandas*, en este caso) en una lista de objetos con la forma (RA, Dec), con las unidades indicadas (grados decimales, igual que el catálogo original). Los objetos de esta clase disponen de un método, llamado *match_to_catalog_sky*, que recibe como parámetro al catálogo original. El resultado de este método es una lista con los objetos cuyas coordenadas coinciden, con un margen de error que se proporciona también como parámetro. El número de elementos de esta lista es directamente el número de "positivos verdaderos" o *true positives*. Por otra parte, la diferencia entre la longitud del catálogo original y los *true positives* es el número de fuentes que no han sido detectadas, denominadas *not detected*. Finalmente, la diferencia entre la longitud del catálogo generado por la red y el número de *true positives* es el número de fuentes falsas que se han detectado (o *false positives*).

Debido a la poca definición de las galaxias en la máscara generada (Fig.3.15), *SExtractor* no centra perfectamente la elipse de ajuste. Por este motivo, se ha elegido un círculo de tolerancia con un radio de 20 segundos de arco, dibujado alrededor de las coordenadas del catálogo original, equivalente a unos seis o siete píxeles aproximadamente.

El primer modelo que se va a evaluar contiene la arquitectura diseñada desde cero, basada en la

U-Net de O. Ronneberger [29]. Los hiperparámetros elegidos se presentan en la Tab.3.2. A modo de ejemplo se presenta la máscara del minicubo número 115 (Fig.3.15), predicha por la red, junto con el catálogo de fuentes que devuelve.

id	ra	dec	hi_size	i	pa
0	48.3333	10.8889	0.971	45.106114	20.30
1	64.0000	1.0000	0.289	0.000000	45.00
2	1.0000	1.0000	0.289	0.000000	45.00
3	23.4000	41.6000	1.229	74.062786	-36.32
4	1.0000	64.0000	0.289	0.000000	45.00
5	46.2778	39.6111	1.423	47.499842	12.24
6	36.5000	36.2000	1.025	44.251252	-0.00
7	26.0000	35.0000	0.926	56.412502	45.00
8	33.0000	28.5000	0.816	53.762808	0.00
9	23.1818	24.1818	0.953	15.905882	45.00
10	49.6000	23.6000	1.085	77.294107	-69.01
11	15.1250	17.1250	0.866	38.641198	45.00
12	34.9000	15.8000	1.188	57.875801	-20.62
13	38.6000	12.8000	0.775	56.485775	71.57
14	51.0909	50.1818	1.068	37.890347	-36.87
15	11.8462	46.8462	1.572	68.918982	17.48

Tabla 3.2: Catálogo de las fuentes detectadas en la Fig.3.15 por *SExtractor*.

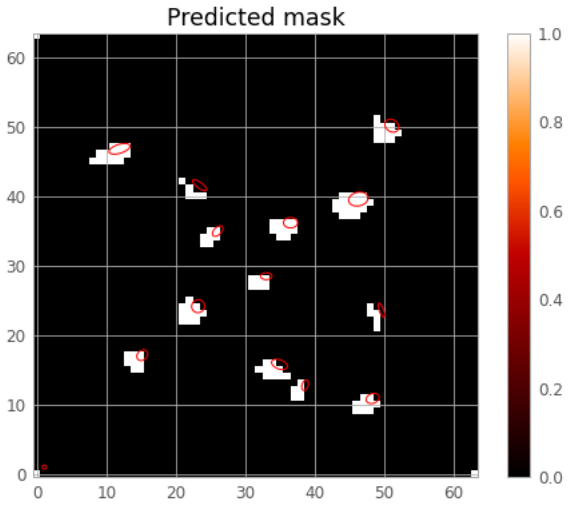

Figura 3.15: Máscara predicha por la red, entrenada con los parámetros de la Tab.3.2. Por simplicidad, se ha considerado como 1 todos los píxeles que pertenecen a una galaxia. Los círculos rojos representan las galaxias detectadas por *SExtractor*, con su tamaño y orientación reales.

A simple vista se puede ver como el modelo dista de ser perfecto. A pesar de detectar un total de 3 fuentes (la mitad del total), tiene una alta tasa de falsos positivos. Un suceso muy llamativo es la caracterización como galaxia de varias de las esquinas de la imagen. Este suceso se repite para la gran mayoría del resto de modelos estudiados. Aunque el origen de este efecto sea desconocido (probablemente debido a la escasez de imágenes para el entrenamiento de la red), se ha corregido de forma manual, eliminando las galaxias del catálogo. Sin embargo, es un síntoma de que la red no se está "aprendiendo" como debe, por lo que se corregirá en futuras versiones.

Como una imagen puede haber sido elegida mediante *cherry picking*, es decir, que no resuma fielmente el rendimiento de la red; se ha optado por evaluar el total de detecciones (*true positives*) frente a las no detecciones (*not detected*) y las falsas detecciones (*false positives*). El resultado para el modelo "first_model" se presenta en la Fig.3.16. El resultado es un porcentaje bastante razonable de detecciones, aproximadamente una cuarta parte del total. Sin embargo, el coste de este número de detecciones más elevado que el resto de modelos es muy elevado. Por cada galaxia del catálogo real, el modelo detecta tres. Dicho de otra manera, por cada *true positive* aparecen doce falsas detecciones, aproximadamente.

Parámetro	Valor
dim	64
flujo_min	12.5
reduction_function	“mean_frec”
overlapping	“auto”
fraction_for_training	64
threshold	0.035
batch_size	32
epochs	200
architecture	“first_model”
filt	16
Normalización	Div. por máximo

Tabla 3.3: Parámetros del modelo “first_model”, utilizados para obtener la Fig.3.16.

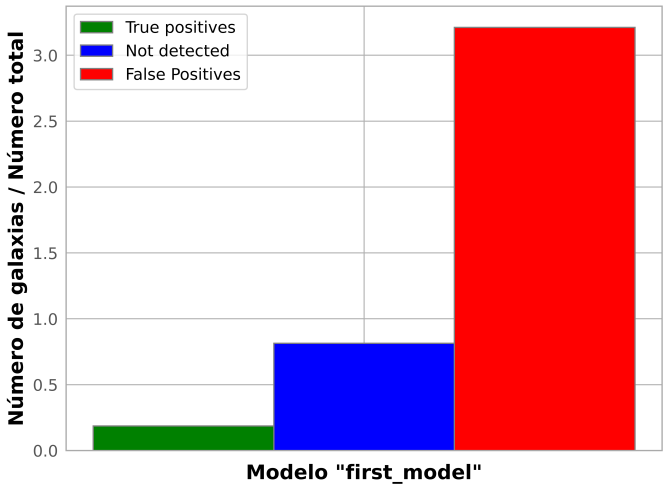

Figura 3.16: Diagrama de barras con el número de positivos verdaderos, falsos positivos y galaxias no detectadas, para el modelo descrito en la Tab.3.3.

El modelo “first_model” permitía crear una red convolucional de cinco capas, en forma de U, cuyo número de filtros se duplicaba en cada paso. Sin embargo, este diseño proporciona muy poca flexibilidad, pues para cambiar cualquiera de los dos parámetros es necesario reprogramar la arquitectura. Afortunadamente, el módulo *keras-unet-collection* incorpora el diseño de la U-Net original, pero más evolucionado. El siguiente paso será probar el efecto que tiene aumentar el número de capas (lo que me permite la memoria) a seis, mientras que el número de filtros comienza en 32. El resto de parámetros se mantienen respecto a la anterior prueba.

Parámetro	Valor
dim	64
flujo_min	12.5
reduction_function	“mean_frec”
overlapping	“auto”
fraction_for_training	0.8
threshold	0.035
batch_size	32
epochs	200
architecture	“unet_2d”
filt	[32, 64, 128, 256, 512, 1024]
Normalización	Div. por máximo

Tabla 3.4: Parámetros del modelo “unet_2d”, utilizados para obtener la Fig.3.17.

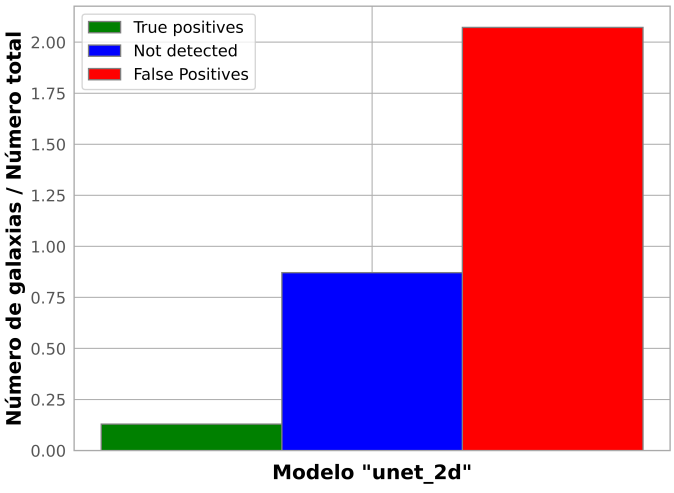

Figura 3.17: Diagrama de barras con el número de positivos verdaderos, falsos positivos y galaxias no detectadas, para el modelo descrito en la Tab.3.4.

En la Fig.3.17 se puede apreciar una ligera mejoría con respecto al modelo previo. El número de falsos positivos se ha reducido considerablemente, aunque se mantiene bastante elevado. Por otra parte, ha descendido ligeramente el número de detecciones, hasta aproximadamente una de cada cinco fuentes del catálogo original. A pesar de este inconveniente, el modelo es más preciso que el anterior, por lo que la modificación ha sido positiva.

Una de las operaciones que más influencia ha demostrado tener en el resultado final es la normalización. Ni una sola de estas arquitecturas fue capaz de hacer predicciones después de haber sido entrenada con datos sin normalizar, ni siquiera modificando los hiperparámetros. Por lo tanto, es casi obligatorio probar con otro tipo de normalización ligeramente más elaborada. Para esta prueba, se ha implementado el algoritmo descrito en este artículo [48], por Andrew Connolly. Manteniendo el resto de parámetros, de forma que quede reflejado el gran impacto que tiene la normalización, se ha entrenado la arquitectura descrita en la primera prueba.

Parámetro	Valor
dim	64
flujo_min	12.5
reduction_function	"mean_frec"
overlapping	"auto"
fraction_for_training	0.8
threshold	0.035
batch_size	32
epochs	200
architecture	"first_model"
filt	16
Normalización	Mét. A. Connolly

Tabla 3.5: Parámetros del modelo "first_model_n2", utilizados para obtener la Fig.3.18

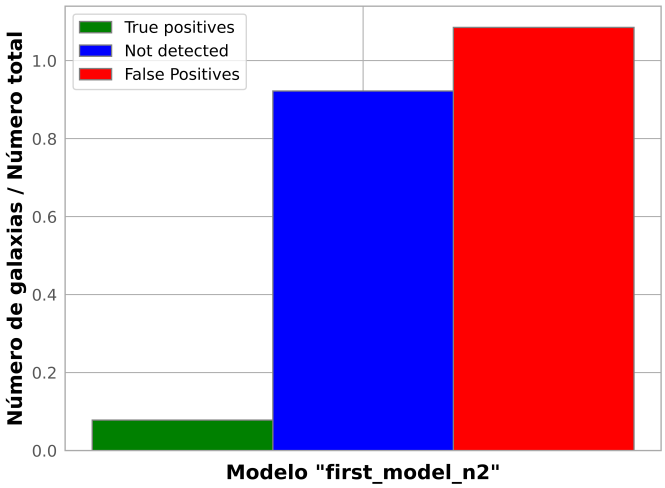

Figura 3.18: Máscara predicha por la red, entrenada con los parámetros de la Tab.3.5. Los círculos rojos representan las galaxias detectadas por *SExtractor*, con su tamaño y orientación reales.

El resultado de esta modificación, la Fig.3.18, no hace más que confirmar que la normalización es uno de los parámetros más importantes (si no el más importante), puesto que ha sido el que más ha modificado el resultado. El número de galaxias detectadas se ha reducido a apenas un 10 % del total. Aunque pueda dar la impresión de que este modelo es mucho peor, nada más lejos de la realidad. El número de falsos positivos se ha reducido hasta prácticamente uno por cada galaxia del catálogo original. Esto equivale a un *ratio* TP/FP del orden de un 10 %, el mismo resultado que obtuvieron otros equipos en su primeras versiones del algoritmo. De cara a futuras versiones del algoritmo, la normalización es sin duda es uno de los aspectos a mejorar.

Una de las opciones para aumentar el número de detecciones es disminuir el *threshold*, es decir, considerar como galaxia a píxeles con una menor probabilidad de serlo. Sin embargo, hay que tener mucho cuidado con este parámetro, pues puede disparar el número de falsos positivos (algo muy poco deseable), además de alterar el tamaño de todas las galaxias. El modelo descrito en la Tab.3.6 pretende demostrar el efecto que tiene elegir un *threshold* particularmente pequeño (una tercera parte del valor utilizado en el resto de modelos). Además, se ha reducido el tamaño de la muestra (*batch_size*) a la mitad. Una de las consecuencias más visibles es el incremento en el tiempo de ejecución, debido a la disminución del tamaño de muestra (son necesarias más iteraciones).

En la Fig.3.19 se puede ver el resultado de esta modificación, una mayor cantidad de objetos calificados erróneamente como galaxias, con respecto al modelo con la misma arquitectura y umbral menos permisivo (Fig.3.17). Como se esperaba, el número de galaxias calificadas correctamente como detecciones ha aumentado, al incluirse un mayor número de objetos de la máscara, generada por la red neuronal, en el catálogo. Aun así, este incremento de fuentes detectadas correctamente se ve eclipsado por el mayor número de falsas detecciones, empeorando la precisión de la red.

Parámetro	Valor
dim	64
flujo_min	12.5
reduction_function	"mean_frec"
overlapping	"auto"
fraction_for_training	0.8
threshold	0.010
batch_size	16
epochs	200
architecture	"unet2d"
filt	[32, 64, 128, 256, 512, 1024]
Normalización	Mét. A. Connolly

Tabla 3.6: Parámetros del modelo "unet_lwr_thr" utilizados para obtener la Fig.3.19.

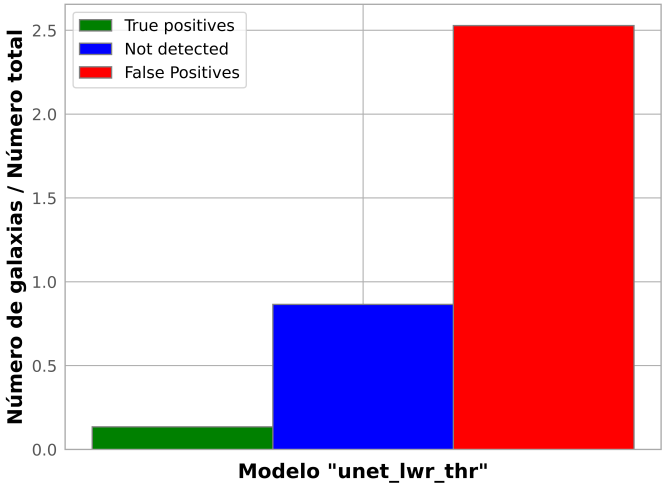

Figura 3.19: Diagrama de barras con el número de positivos verdaderos, falsos positivos y galaxias no detectadas, para el modelo descrito en la Tab.3.6.

Es posible que esta pérdida de precisión se deba al solapamiento entre galaxias. Al considerar más píxeles como válidos es posible que algunas galaxias se apilen, propiciando el error del programa de detección *SExtraxtor*, que las considerará como una unidad. Además, aunque no formaba parte de los objetivos principales del algoritmo de detección, las dimensiones y orientaciones de las galaxias, que ya de por sí eran poco fiables debido a la poca resolución, con este modelo lo son todavía menos, ya que se aumenta artificialmente su tamaño.

Finalmente, a pesar de que el resto de modelos hacían predicciones, más o menos mejores, no es siempre el caso. No todos los modelos creados a partir de las arquitecturas del módulo *keras-unet-collection* han proporcionado resultados satisfactorios. El modelo descrito en la Tab.3.7 demuestra que más complejidad no es necesariamente mejor. El tiempo de entrenamiento de este modelo era aproximadamente dos veces superior al resto y, sin embargo, no ha sido capaz de detectar ni una sola fuente en la imagen (aun con un *threshold* de 0.01). Aunque solo se presente esta imagen (Fig.3.20), ninguno del resto de mini-cubos de datos generó una respuesta distinta a cero.

Parámetro	Valor
dim	64
flujo_min	12.5
reduction_function	"mean_frec"
overlapping	"auto"
fraction_for_training	0.8
threshold	0.01
batch_size	16
epochs	200
architecture	"u2unet"
filt	[32, 64, 128, 256]
Normalización	Mét. A. Connolly

Tabla 3.7: Parámetros del modelo "u2unet" utilizados para obtener la Fig.3.20.

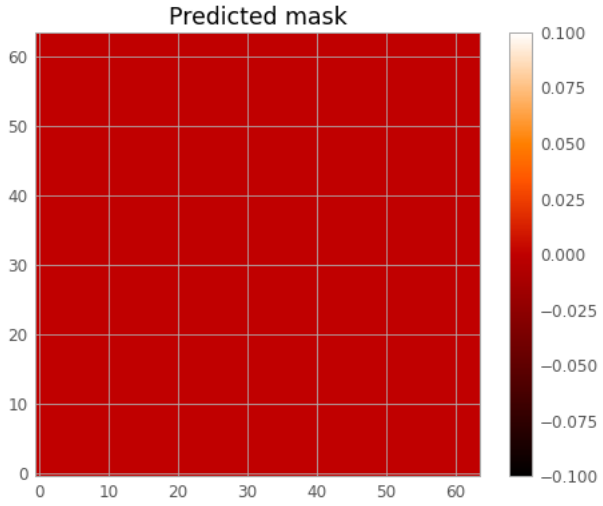

Figura 3.20: Máscara de un cubo aleatorio, predicha por la red con los parámetros de la Tab.3.7.

3.6 Rendimiento. Comparación de tiempos entre GPU y CPU

Esta sección pretende comparar los tiempos de entrenamiento de varias arquitecturas, utilizando las dos opciones disponibles: *tensorflow* con procesador (opción normal) y *tensorflow* con tarjeta gráfica (supuestamente mejor). La Fig.3.21 confirma las sospechas iniciales, *tensorflow* ejecutado en la tarjeta gráfica es mucho más eficiente que la ejecución por defecto en el procesador. Sin embargo, la diferencia ha resultado ser abismal. Por ejemplo, una época del entrenamiento de la arquitectura "u2net_2d" ha empleado más de un minuto utilizando el procesador, mientras que la tarjeta gráfica apenas empleaba dos o tres segundos. Aunque no parezca una diferencia tan grande, este es el tiempo de ejecución de una única época. Los modelos expuestos en la sección anterior han tomado un máximo de 200 épocas, por lo que el tiempo invertido en la instalación de *tensorflow* GPU ha resultado muy rentable.

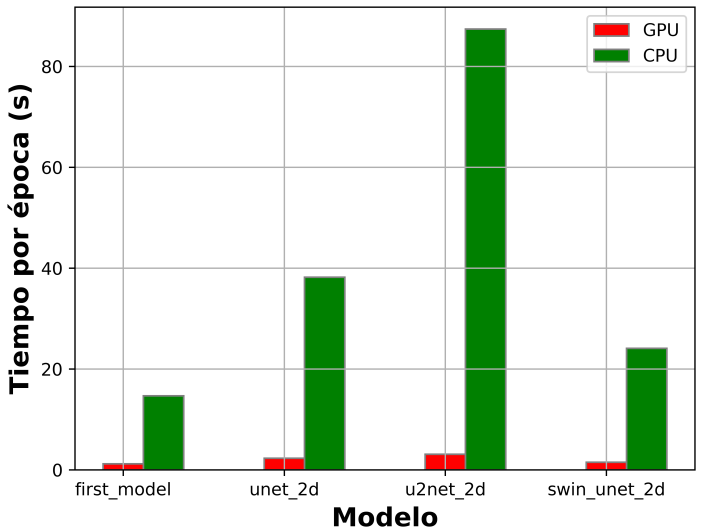

Figura 3.21: Tiempo empleado por varios modelo para finalizar el entrenamiento de una sola época, utilizando la CPU y la GPU. Todas las arquitecturas se han probado con los mismos hiperparámetros, para resaltar el papel de la GPU.

4 Conclusiones

4.1 Valoración de los resultados

Es conveniente recordar que el objetivo del trabajo **nunca** fue desarrollar un *software* capaz de detectar grandes cantidades de objetos celestes, puesto que esa labor la realizaron grupos (todos de más de cinco personas) en aproximadamente medio año. Si el objetivo hubiese sido obtener una buena calificación del algoritmo, no se habría diseñado un *pipeline* desde cero. El objetivo real era familiarizarse con el manejo de este tipo de datos, preparándolos para servir como entrenamiento para una red neuronal. De nada sirve utilizar *software* de detección por primera vez si no se conocen los fundamentos de su funcionamiento.

Dicho esto, se puede hacer una valoración positiva (relativamente) de los resultados. Aunque los modelos de redes neuronales no han sido ninguna maravilla, todas las modificaciones previas se han mostrado muy útiles. La primera, imprescindible, dividir el cubo de datos (una vez se han reducido sus dimensiones de 3 a 2) era posiblemente la más crítica, no tanto por su complejidad sino por su rendimiento. El problema de trabajar con archivos de datos pesados es que absolutamente todo el código tiene que estar lo más optimizado posible, para reducir los tiempos de ejecución. Después de muchas modificaciones del diseño original, el tiempo de ejecución de esta función se redujo de más de 4 minutos, a poco menos de uno en la versión final (para el fichero de datos grande). Además de esta mejora en su rendimiento, la Fig.3.5 y la Fig.3.4 demuestran que este algoritmo, encargado de dividir el cubo, está funcionando correctamente. La inclusión de los conceptos de "solapamiento" y de "espejado" en este algoritmo ha sido, sin duda, un acierto que ha contribuido a mejorar la calidad de los resultados. El

primero, evita que la red vea galaxias cortadas una única vez, mientras que el segundo, evita que la imagen tenga una discontinuidad en dos de sus bordes.

Por otra parte, el módulo de generación de máscaras ha funcionado como estaba previsto. Si bien es cierto que en un principio, debido a la inclusión de fuentes de HI únicamente en el catálogo, no todas las máscaras no coincidían con una fuente en las imágenes; la eliminación de las fuentes de menor flujo integrado ha supuesto una mejora sustancial. De haberse mantenido en el catálogo, habrían empeorado el entrenamiento de la red neuronal, consiguiendo unos resultados mucho peores en la parte de predicción.

La red neuronal, a pesar de todas las limitaciones a las que se enfrentaba (número de imágenes limitado, imágenes en 2 dimensiones, eliminación de fuentes con poco flujo o la no utilización del cubo de datos de HI, entre otros) ha sido capaz de predecir, de forma aceptable, la posición de una parte del catálogo de fuentes. Aunque el resultado final de un 10 % de precisión sea malo, el objetivo no era ese. Es necesario mucho más tiempo y experiencia para desarrollar un modelo de estas características. Precisamente esa experiencia es el resultado más importante de este trabajo que, con toda seguridad, contribuirá a mejorar en este campo.

Lamentablemente, pese al tiempo invertido en habilitar el *software* de *scoring* de SKA, no ha sido posible utilizarlo para evaluar el conjunto de la red, puesto que la red neuronal no estaba pensada para proporcionar todos los parámetros del catálogo. Es por este motivo que se ha optado por un método más primitivo de evaluación, como es la evaluación a través del método *match_to_catalog_sky* de la clase *SkyCoord* de *astropy*. El resumen de toda la comparación entre modelos se presenta en la Fig.4.1, utilizando la precisión, definida en la ec.9, como métrica para evaluar el rendimiento de los modelos.

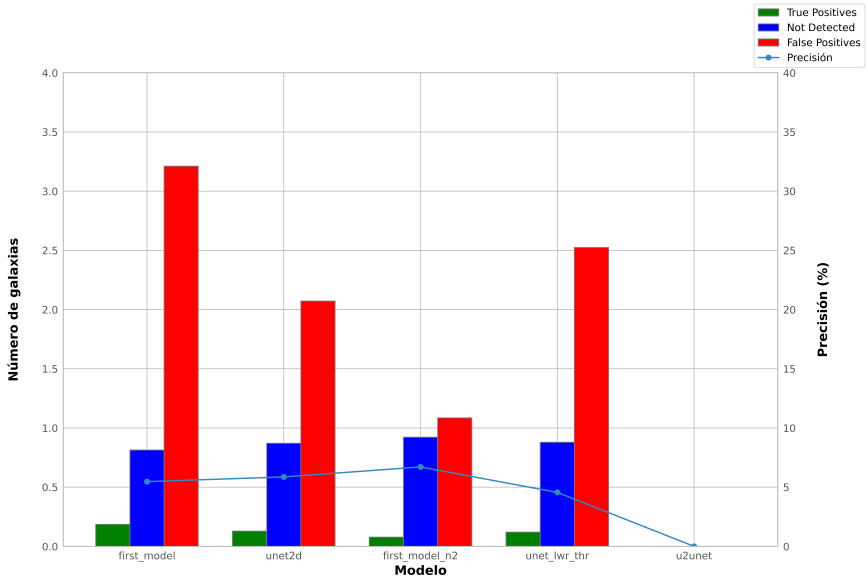

Figura 4.1: Histograma comparando los cinco modelos probados en la Sec.3.5, donde cada una de las barras representa los *True Positives*, los *False Positives* y el número de galaxias no detectadas (*Not Detected*), respectivamente. La "curva" de color azul representa la precisión (positivos reales entre la suma de positivos reales y falsos positivos) para cada uno de los modelos.

Atendiendo a la precisión, el mejor modelo es, curiosamente el que menos galaxias verdaderas ha detectado. La fortaleza del modelo "first_model_n2" está en su (relativamente) bajo número de falsos positivos, lo que le proporciona una ventaja frente al resto. Como ya se ha comentado, de nada sirve que un algoritmo detecte todas las galaxias del cubo si incluye en el catálogo 30 cuerpos que no son galaxias reales. La clave de este modelo ha sido el cambio en la normalización, utilizando un proceso más elaborado que en los menos precisos "first_model" y "unet_2d", donde simplemente se dividía por el valor máximo. Finalmente, merece la pena comentar el papel del umbral. El valor de este parámetro ha demostrado ser vital para el desenvolvimiento de la red. Un valor conservador de este parámetro ha generado una red con una mejor precisión.

En conclusión, el objetivo de familiarizarse con todo el proceso se ha superado por mucho, consiguiendo incluso un modelo capaz de realizar predicciones, aunque con ciertas limitaciones. Dedicando

más tiempo a encontrar los hiperparámetros y la arquitectura adecuada, sería posible conseguir precisiones superiores al 10 % con relativa facilidad. Sin embargo, por mucho que mejore el modelo, es poco probable que consiga una buena precisión. Para conseguir resultados más positivos es clave implementar un proceso de eliminación de ruido, previo al entrenamiento de la red. De esta forma, se conseguirá que esta se ajuste más a la realidad y mejorará la precisión. Un análisis que puede ser interesante es estudiar qué tipo de fuentes está detectando la red. Una de las conclusiones a las que llegaron organizadores y participantes en el desafío fue que no existe un único algoritmo de detección universal, cada uno tenía sus fortalezas y sus debilidades. Por lo tanto, podría darse el caso de que los modelos implementados en esta memoria estén detectando un tipo determinado de AGN, por ejemplo, que otros algoritmos no detecten tan fácilmente. Como la unión hace la fuerza, es posible que en combinación con otros algoritmos de detección mejorase la precisión de los modelos implementados.

4.2 Opciones de mejora

El margen de mejora del programa es enorme, y además comprende todas las fases del *pipeline*. Sin lugar a dudas, la adaptación del código para utilizar el archivo de datos de HI es la primera evolución que se implementará en futuras versiones. Aunque el tiempo disponible para realizar el trabajo ha imposibilitado su uso, es de vital importancia introducirlo en el entrenamiento de la red neuronal, como han realizado todos los equipos participantes en el desafío. Su uso, además de permitir detectar las fuentes emisoras de HI únicamente, que en este trabajo se han eliminado, permitiría incrementar la precisión en la detección de fuentes con espectro continuo. De hecho, uno de los equipos que ha implementado una solución basada en la segmentación, ha generado sus máscaras a partir del fichero con los datos de HI. Otra de las mejoras obligatorias para futuras implementaciones reside en el manejo del cubo de datos, concretamente en la parte en la que se simplifica su dimensionalidad. A pesar de que reducir el número de dimensiones permite simplificar el problema, también conlleva la pérdida de una parte importante de la información, la tercera dimensión (frecuencia). Esto supone un problema, no tanto a la hora de detectar más o menos galaxias, que también; sino para la utilización del *software* de evaluación del algoritmo. Este *software* requiere de un catálogo con exactamente el mismo formato que el original, es decir, además de las coordenadas, la dimensión y la orientación del objeto, la red debe predecir la integral del flujo para todas las frecuencias, la frecuencia central (tercera coordenada de posición) y la anchura de la línea de 21 cm. Este es otro de los motivos por los que urge adaptar el código al cubo de datos de HI, que contiene información imprescindible para obtener estos parámetros.

Sin embargo, estas modificaciones tienen un coste. El fichero de datos de HI es mucho más grande que el del continuo, ya que la diferencia de frecuencias entre las imágenes es mucho menor. Aun mejorando la eficiencia del código, los tiempos de ejecución se dispararán y, posiblemente, los recursos computacionales (memoria principalmente), no sean suficientes para manejar una red de esta complejidad.

A pesar de la "gran" revolución en el código que suponen las modificaciones anteriores, la siguiente parte del *pipeline*, la división del cubo de datos y del catálogo, no necesitaría de grandes cambios, pues se ha probado bastante eficiente. La única modificación obligatoria sería adaptar la función a la tercera dimensión, aumentando el número de bucles de dos a tres. El algoritmo para seleccionar *slices* en la tercera dimensión se podría extrapolar de las otras dos, sin apenas dificultad. El principal problema que tendría esta sección es, como se menciona en el párrafo anterior, el coste computacional. Los bucles *for* son poco eficientes en *python*, por lo que el tiempo de ejecución de esta función se dispararía. Para solucionarlo, o al menos mitigarlo, sería interesante la sustitución de bucles por *list comprehensions*, mucho más eficientes.

Por otra parte, el módulo de creación de máscaras apenas necesita modificaciones en su código. Sin embargo, al igual que en la sección de *data_splitting*, debe ser adaptado a las tres dimensiones. Sustituir a la función *Ellipse2D* de *astropy* por una adaptada a tres dimensiones es donde reside toda la dificultad de esta adaptación. Una vez creada la máscara, el resto del código no presenta mayores problemas.

Respecto a la red neuronal, al contrario de lo que pueda parecer, no necesita grandes cambios (más allá de adaptar el *input size*), ya que las arquitecturas utilizadas han sido diseñadas para problemas de esta complejidad. La forma de mejorar esta sección depende directamente de la parte de *scoring*, puesto que si esta estuviera operativa permitiría una rápida evaluación del modelo. Esta es la única mejora posible del modelo de red neuronal, probar el mayor número de arquitecturas más hiperparámetros posibles. Es por ello que una de las prioridades de futuras versiones es que la red (ya sea a través de una

única red o de varias) predigan el resto de magnitudes no proporcionadas por la actual, de forma que se pueda utilizar la función *score*, ya implementada en la clase *GalaxyDetector*.

4.3 Resultados del aprendizaje

Para terminar, en esta sección se resume toda la experiencia y aprendizaje adquiridos durante la realización del trabajo.

Durante el último verano llevé a cabo una práctica en el IFCA, concretamente en el campo de los Rayos-X, cuyo objetivo era traducir un manual con código en lenguaje *bash* a varios *jupyter-notebook*[8], para facilitar su interpretación. Así, aunque la práctica no implicaba por ningún lado al *machine learning*, sí que involucraba el manejo de archivos FITS y el uso de programas como SAOImageDS9 [49]. Esta práctica sirvió como iniciación, de forma que en este trabajo se ha profundizado en el manejo de estas herramientas (DS9, FITS, etc). Uno de los resultados del aprendizaje ha sido la soltura que se ha adquirido en el manejo de este tipo de ficheros, así como la utilización de *software* externo para validar los resultados.

Sin embargo, a pesar de que la práctica también consistía en manejar archivos que simulan observaciones (de hecho, en la práctica, su simulación era parte del código), guarda ciertas diferencias con respecto a este trabajo. Por ejemplo, ha sido necesario comprender el funcionamiento de una parte importante de las funciones disponibles en la librería *astropy*. La soltura en el manejo de este *software* es, sin duda, uno de los resultados más importantes que se han conseguido gracias a la realización del trabajo. Este *software* es ampliamente utilizado en todo el mundo, principalmente para el análisis y la manipulación de imágenes procedentes de telescopios. Por otra parte, los archivos de datos generados por estos telescopios suelen ser de un tamaño importante, aumentando la relevancia que tiene la optimización del código. Aquí es donde aparece la librería *pandas* que, aunque utilizada principalmente en los programas relacionados con *data science* en general (redes neuronales, big data...), también es de gran utilidad en el campo de la astrofísica, debido a la eficiencia de sus métodos y funciones. Saber manejar correctamente esta librería requiere bastante práctica, sobre todo a la hora de elegir la forma más eficiente de llevar a cabo una operación. Otro de los resultados del aprendizaje es por tanto, la destreza adquirida en la utilización de *pandas* como herramienta de manejo de datos. Además, para que un programa funcione razonablemente bien con grandes *datasets*, es clave la optimización de sus operaciones, más allá de las relacionadas con *pandas*. Al resultado de aprendizaje anterior se le añade la utilización de funciones u operaciones pensadas para reducir el consumo de memoria (y por tanto, el tiempo de ejecución), como puede ser la compresión de listas (sustituyendo a los bucles *for* tradicionales).

Pero, no todo lo que se ha aprendido está relacionado con programación. Si en la práctica de Rayos-X fue posible entender parte de la emisión en esta sección del espectro, este trabajo no podía ser menos. Así, también se ha ampliado el conocimiento sobre fuentes emisoras de ondas de radio, como pueden ser los núcleos de galaxia activos o las galaxias formadoras de estrellas. Especialmente interesante ha sido la documentación acerca de los procesos mediante los cuales estos cuerpos emiten un continuo de radiación en ondas de radio, como son la acreción de materia y los *jets* de partículas, ambos relacionados a través la radiación de sincrotrón.

La emisión de este tipo de objetos representa un espectro continuo, que es el que se ha utilizado en el algoritmo, pero también se ha ampliado el conocimiento acerca de la emisión discreta, concretamente la línea de 21 cm, así como su importancia en el estudio del cosmos, gracias a las ventajas que se describían previamente en la introducción. Concretamente, el mapeo de la distribución de hidrógeno en el universo, utilizando la línea HI, ha demostrado ser una herramienta muy interesante para estudiar la evolución de las galaxias, así como del universo en su conjunto. Además, la relación entre el mapeo de galaxias a través de la línea HI y la cosmología ha abierto un universo, y nunca mejor dicho, de oportunidades. En el futuro se estudiará, por tanto, implementar los modelos de distribución espacial de galaxias, con el fin de estudiar su evolución y de estimar los parámetros cosmológicos.

A esto se le suma la novedad que ha supuesto la frecuencia como tercera dimensión (profundidad), resultando particularmente interesante comprender cómo variaba la intensidad a medida que lo hacía la frecuencia, pues puede resultar ligeramente más contraintuitivo que simplemente desplazarse por las

[8] El *jupyter-notebook* o cuaderno de "jupyter" es una herramienta diseñada para facilitar el aprendizaje del código, en *python*, que se presenta. Permite la ejecución de celdas (trozos de código) de forma independiente, pudiendo construir el equivalente a un "manual de instrucciones".

coordenadas habituales (ascensión recta y declinación). Además, se ha entendido el papel que va a tener SKA en el desarrollo de la astrofísica en las ondas de radio, tanto probando la teoría de la relatividad general como catalogando un mayor número de galaxias, incluyendo algunas de las formadas pocos años después del Big Bang y que, con suerte, permitirán comprender mejor este proceso. En definitiva, este trabajo ha servido como iniciación en el campo de la astrofísica en ondas de radio, sirviendo como punto de partida para generar un código del que "extraer física" en el futuro.

Respecto a la sección de *machine learning*, uno de los resultados del aprendizaje más importante ha sido la creación de redes neuronales desde cero, así como la utilización de arquitecturas previamente programadas en otros módulos. Además ha sido necesaria la búsqueda de las arquitecturas más habituales para este tipo de problemas. También el verano pasado asistí a un curso (breve, apenas una semana) sobre el *machine learning* en *python*, donde se explicaba cómo construir redes neuronales y demás formas de inteligencia artificial (arquitecturas mucho más sencillas que esta, y con *datasets* adaptados a un nivel principiante) desde cero. La principal motivación del trabajo ha sido profundizar en este campo, ampliando el conocimiento adquirido en el curso de verano. Las máscaras generadas en la Sec.3.5 demuestran que este trabajo ha servido como ampliación de los conocimientos sobre las arquitecturas de redes neuronales convolucionales, así como de los problemas que involucran segmentación de imágenes.

Para el correcto y óptimo funcionamiento de los modelos de redes ha sido clave el manejo de la herramienta *tensorflow GPU*, desconocida antes de comenzar el trabajo. Durante la labor de documentación de las primeras semanas, surgió, gracias a un artículo que ya no recuerdo, la opción de utilizar la tarjeta gráfica dedicada en el entrenamiento de la futura red neuronal (por aquel entonces, la red neuronal no había pasado la etapa de boceto). A pesar del arduo proceso de instalación, que luego resultó ser absurdamente sencillo, ha merecido la pena comprender el funcionamiento de esta herramienta, que seguramente permitirá ahorrar multitud de horas de computación en futuros proyectos de *deep learning*.

Aunque la gran mayoría de las veces no reciba la atención que merece, la documentación es probablemente la parte más importante en el desarrollo de un programa. Por ello, se ha considerado imprescindible conocer y dominar la librería *sphinx* (desconocida hasta las últimas semanas del trabajo), ya que permite generar documentaciones interactivas (y bonitas), que pueden ser publicadas posteriormente en páginas web. Cualquier futuro proyecto destinado a ser publicado requerirá de una página web con documentación, tanto para fomentar su uso (generando una buena primera impresión) como para facilitar su comprensión y posibles modificaciones.

Otra de las partes que suelen pasar inadvertidas en un proyecto de estas características son los modelos y diagramas que describen el código. Es especialmente interesante describir el funcionamiento de funciones u operaciones complejas a través de diagramas, preferiblemente en lenguaje UML, ya que es universal y no depende del lenguaje de programación escogido. Esto supone una ventaja para personas que manejen un lenguaje diferente, pero estén interesados en la estructura del código. Por ello, los diagramas incluidos en este proyecto (además de los que no se han incluido por falta de espacio) han servido para profundizar y mejorar en el manejo del lenguaje UML, así como del intérprete utilizado BoUML [50], ambos idioma e intérprete introducidos en el curso *Advanced Computation*, en cuarto de carrera.

También, aunque no haya resultado tan llamativo como otras partes del código, el uso de *SExtractor* ha permitido ahorrar una gran cantidad de tiempo, que se habría invertido en la programación de una función capaz de detectar galaxias a partir de una máscara. Por lo tanto, otro de los resultados de aprendizaje, aunque algo más secundario, ha sido manejar las funciones más básicas de *SExtractor*, a través de la modificación de su archivo de configuración (default.sex) y de la selección de las magnitudes deseadas en el archivo de parámetros (default.param).

Finalmente, como resultado más global está la experiencia adquirida en la resolución de problemas complejos, para los que no existe una única solución válida. Esta ha sido la mayor diferencia respecto al resto de trabajos realizados durante la carrera, para los cuales se obtenía siempre un resultado más o menos esperado, gracias a la documentación a través de la bibliografía. En resumen, antes de comenzar la experimentación ya se conocía la forma que los resultados debían de tener. No ha sido así en ningún momento del desarrollo del trabajo, ya que ha sido necesario plantear propuestas creativas para intentar conseguir el mejor resultado posible. Esta experiencia adquirida será, con total certeza, clave para futuros proyectos de investigación fuera del ambiente controlado de la carrera.

Referencias

[1] ESA. *A history of astrometry - Part I Mapping the sky from ancient to pre-modern times*. https://sci.esa.int/web/gaia/-/53196-the-oldest-sky-maps. 2019 (vid. pág. 2).

[2] The Nobel Prize. *All Nobel Prizes in Physics*. https://www.nobelprize.org/prizes/lists/all-nobel-prizes-in-physics/. 2022 (vid. pág. 2).

[3] NRAO/AUI. https://commons.wikimedia.org/w/index.php?curid=58041073. 2022 (vid. pág. 2).

[4] H. Schweiker/WIYN y NOAO/AURA/NSF. https://commons.wikimedia.org/w/index.php?curid=113226349. 2012 (vid. pág. 2).

[5] Las Cumbres Observatory. *Radio Telescopes*. https://lco.global/spacebook/telescopes/radio-telescopes/. 2022 (vid. pág. 2).

[6] Eric Chaisson y Stephen McMillan. *Astronomy today*. English. Eight;Global; Harlow: Pearson, 2015, págs. 148-149. ISBN: 1292072660 (vid. pág. 3).

[7] Kenneth I. Kellermann, Ellen N. Bouton y Sierra S. Brandt. *Open Skies: The National Radio Astronomy Observatory and Its Impact on US Radio Astronomy*. English. Cham: Springer International Publishing AG, 2020, pág. 48. ISBN: 3030323447 (vid. pág. 3).

[8] Eric Chaisson y Stephen McMillan. *Astronomy today*. English. Eight;Global; Harlow: Pearson, 2015, pág. 458. ISBN: 1292072660 (vid. pág. 3).

[9] Steven N. Shore. «Magnetic Fields in Astrophysics». En: *Encyclopedia of Physical Science and Technology (Third Edition)*. Ed. por Robert A. Meyers. Third Edition. New York: Academic Press, 2003, págs. 903-918. ISBN: 978-0-12-227410-7. DOI: https://doi.org/10.1016/B0-12-227410-5/00392-6. URL: https://www.sciencedirect.com/science/article/pii/B0122274105003926 (vid. pág. 4).

[10] R.K. Pathria y Paul D. Beale. «1 - The statistical basis of thermodynamics». En: *Statistical Mechanics (Fourth Edition)*. Ed. por R.K. Pathria y Paul D. Beale. Fourth Edition. Academic Press, 2022, pág. 14. ISBN: 978-0-08-102692-2. DOI: https://doi.org/10.1016/B978-0-08-102692-2.00010-7. URL: https://www.sciencedirect.com/science/article/pii/B9780081026922000107 (vid. pág. 4).

[11] SKAO. *Participating Countries*. https://www.skatelescope.org/participating-countries/. 2022 (vid. pág. 4).

[12] SKAO. *Cosmic Magnetism*. https://www.skatelescope.org/galaxyevolution/. 2022 (vid. pág. 4).

[13] SKAO. *Cosmic Magnetism*. https://www.skatelescope.org/cradle-life/. 2022 (vid. pág. 4).

[14] SKAO. *Challenging Einstein*. https://www.skatelescope.org/challenging-einstein/. 2022 (vid. pág. 5).

[15] LIGO. *Timeline*. https://www.ligo.caltech.edu/page/timeline. 2022 (vid. pág. 5).

[16] SKAO. *Cosmic Magnetism*. https://www.skatelescope.org/magnetism/. 2022 (vid. pág. 5).

[17] NASA. *Star Formation*. https://science.nasa.gov/astrophysics/focus-areas/how-do-stars-form-and-evolve. 2022 (vid. pág. 6).

[18] Miguel Pérez-Torres et al. *Libro blanco del SKA*. English. Sociedad Española de Astronomi, 2015, págs. 170-171. ISBN: 978-84-606-8955-3 (vid. pág. 6).

[19] Dan Maoz. *Astrophysics in a Nutshell*. PRINCETON UNIVERSITY PRESS, 2016, págs. 200-210 (vid. pág. 7).

[20] A. Bonaldi P. Hartley y R. Braun et al. «SKA Science Data Challenge 2: analysis and results». En: *SKA* (2022) (vid. págs. 9, 10, 19, 23).

[21] Michael G Jones y col. «The ALFALFA Hi mass function: a dichotomy in the low-mass slope and a locally suppressed 'knee' mass». En: *Monthly Notices of the Royal Astronomical Society* 477.1 (feb. de 2018), págs. 2-17. ISSN: 0035-8711. DOI: 10.1093/mnras/sty521. eprint: https://academic.oup.com/mnras/article-pdf/477/1/2/24615118/sty521.pdf. URL: https://doi.org/10.1093/mnras/sty521 (vid. pág. 9).

[22] A Bonaldi y col. «Square Kilometre Array Science Data Challenge 1: analysis and results». En: *Monthly Notices of the Royal Astronomical Society* 500.3 (oct. de 2020), págs. 3821-3837. ISSN: 0035-8711. DOI: 10.1093/mnras/staa3023. eprint: https://academic.oup.com/mnras/article-pdf/500/3/3821/34673828/staa3023.pdf. URL: https://doi.org/10.1093/mnras/staa3023 (vid. pág. 9).

[23] T Westmeier y col. «sofia 2 – an automated, parallel Hi source finding pipeline for the WALLABY survey». En: *Monthly Notices of the Royal Astronomical Society* 506.3 (jul. de 2021), págs. 3962-3976. ISSN: 0035-8711. DOI: 10.1093/mnras/stab1881. eprint: https://academic.oup.com/mnras/article-pdf/506/3/3962/39458911/stab1881.pdf. URL: https://doi.org/10.1093/mnras/stab1881 (vid. pág. 10).

[24] Ivan N. da Silva y col. *Artificial Neural Networks: A Practical Course*. English. Cham: Springer International Publishing AG, 2016. Cap. 1.2, 1.3. ISBN: 3319431617 (vid. págs. 10, 12).

[25] Ivan N. da Silva y col. *Artificial Neural Networks: A Practical Course*. English. Cham: Springer International Publishing AG, 2016, pág. 6. ISBN: 3319431617 (vid. pág. 11).

[26] Frank Rosenblatt. «The Design of an Intelligent Automaton». En: *Research Trends 6, no. 2* (1958) (vid. pág. 11).

[27] Tensorflow. *Image segmentation*. https://www.tensorflow.org/tutorials/images/segmentation. 2022 (vid. pág. 13).

[28] Burger Becker y col. «CNN architecture comparison for radio galaxy classification». En: *Monthly Notices of the Royal Astronomical Society* 503.2 (feb. de 2021), págs. 1828-1846. DOI: 10.1093/mnras/stab325. URL: https://doi.org/10.1093/2Fmnras/2Fstab325 (vid. págs. 14, 34).

[29] Olaf Ronneberger, Philipp Fischer y Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04597. URL: https://arxiv.org/abs/1505.04597 (vid. págs. 14, 15, 34, 39).

[30] Pragati Baheti. *Activation Functions in Neural Networks [12 Types & Use Cases]*. https://www.v7labs.com/blog/neural-networks-activation-functions. 2022 (vid. pág. 15).

[31] Pranoy Radhakrishnan. «What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?» En: *Towards Data Science* (2017) (vid. pág. 16).

[32] Anaconda. https://www.anaconda.com/. 2022 (vid. pág. 19).

[33] NASA. *FITS Documentation*. https://fits.gsfc.nasa.gov/fits_documentation.html. 2021 (vid. págs. 20, 22).

[34] Yingkai Sha. *Keras-unet-collection*. https://github.com/yingkaisha/keras-unet-collection. 2021. DOI: 10.5281/zenodo.5449801 (vid. págs. 20, 34).

[35] David Girón Ceballos. *galaxy-detector*. https://github.com/duvidG/tfg.git. 2022 (vid. págs. 20, 28).

[36] NVIDIA. *CUDA Zone*. https://developer.nvidia.com/cuda-zone. 2022 (vid. pág. 20).

[37] Joshua James. «Install NVIDIA Drivers on Fedora Linux 34/35». En: *linuxcapable.com* (2022) (vid. pág. 21).

[38] Harveen Singh Chadha. «Tensorflow GPU Installation Made Easy: Use conda instead of pip». En: *Towards Data Science* (2018) (vid. pág. 21).

[39] SKA. *Data SDC2*. https://sdc2.astronomers.skatelescope.org/sdc2-challenge/data. 2021 (vid. pág. 22).

[40] NASA. *FITS Keyword Dictionaries*. https://heasarc.gsfc.nasa.gov/docs/fcg/standard_dict.html. 2021 (vid. pág. 22).

[41] Pandas. *Dataframe Documentation*. "https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html". 2022 (vid. pág. 29).

[42] Astropy. *WCS Documentation*. "https://docs.astropy.org/en/stable/wcs/index.html". 2022 (vid. pág. 30).

[43] Astropy. *Ellipse2D Documentation*. "https://docs.astropy.org/en/stable/api/astropy.modeling.functional_models.Ellipse2D.html". 2022 (vid. pág. 33).

[44] Sphinx. *Python Documentation Generator*. https://www.sphinx-doc.org/en/master/index.html. 2022 (vid. pág. 37).

[45] E. Bertin y S. Arnouts. ≪SExtractor: Software for source extraction.≫ En: *aaps* 117 (jun. de 1996), págs. 393-404. DOI: 10.1051/aas:1996164 (vid. pág. 38).

[46] Y. Bengio Y. LeCun L. Bottou y P. Haffner. *Gradient-Based Learning Applied to Document Recognition*. http://yann.lecun.com/exdb/mnist/. 1998 (vid. pág. 38).

[47] Astropy. *Matching Catalogues*. https://docs.astropy.org/en/stable/coordinates/matchsep.html. 2022 (vid. pág. 38).

[48] Andrew Connolly. ≪Deep Learning: Classifying Astronomical Images≫. En: *AstroML* (2022) (vid. pág. 41).

[49] W. A. Joye y E. Mandel. ≪New Features of SAOImage DS9≫. En: *Astronomical Data Analysis Software and Systems XII*. Ed. por H. E. Payne, R. I. Jedrzejewski y R. N. Hook. Vol. 295. Astronomical Society of the Pacific Conference Series. Ene. de 2003, pág. 489 (vid. pág. 46).

[50] Bruno Pagès. *BoUML*. https://www.bouml.fr/index.html. 2021 (vid. pág. 47).

5 Apéndice

5.1 Página web con documentación: *Read the docs*

La parte quizás más importante en la programación, aunque no lo parezca, es la documentación. De nada sirve un código capaz de resolver problemas supercomplejos si nadie lo sabe utilizar. También puede darse el caso de que el propio programador o programadora no sea capaz de recordar cómo funciona parte de su código (especialmente si ha sido programado hace mucho tiempo). Por ello, se ha dedicado una parte importante del tiempo disponible en documentar todas las funciones, además de comprender el funcionamiento del paquete *sphinx*. Gracias a este *software* se ha podido generar una página web con toda la documentación perfectamente organizada. A modo de ejemplo, se presenta en la Fig.5.1 la paǵina principal de esta. Para acceder a ella, debido a problemas de compatibilidad de *readthedocs.org*, se proporciona el archivo *html*. El siguiente comando permite visualizar la documentación en el navegador:

Código 8: Comando para lanzar la documentación en firefox, después de haber descargado el repositorio de *Github*. Para que funcione se debe ejecutar desde la base de este repositorio.

```
firefox galaxy_detector/docs/_build/html/index.html
```

5.2 Instrucciones de uso del código

En esta sección se incluyen unas breves instrucciones sobre la instalación del código creado:

1. Descargar el proyecto, alojado en la plataforma *Github*. Es posible descargarlo como .zip o también a través del siguiente comando en línea de comandos:

   Código 9: Comando para descargar el proyecto desde *Github*.

   ```
   git clone https://github.com/duvidG/tfg.git
   ```

2. Una vez instalado, el siguiente paso es instalar las dependencias necesarias para su correcta ejecución. Para ello se recomienda crear un ambiente de *conda* utilizando el fichero de texto "versión_paquetes_env.txt":

   Código 10: Comando para crear el ambiente de *conda*.

   ```
   conda env create --file version\_paquetes\_env.txt
   ```

3. Después de crear, y de activar, el ambiente ya solo queda utilizar el código. Para que el código funcione, el fichero o *script* desde donde se estén llamando a las funciones del paquete debe estar situado dentro de la carpeta "GalaxyDetector". En el archivo "main.py"se muestra un ejemplo de utilización, concretamente el que se ha utilizado para generar todas las figuras de este documento. La información sobre las funciones disponibles está disponible en la documentación.

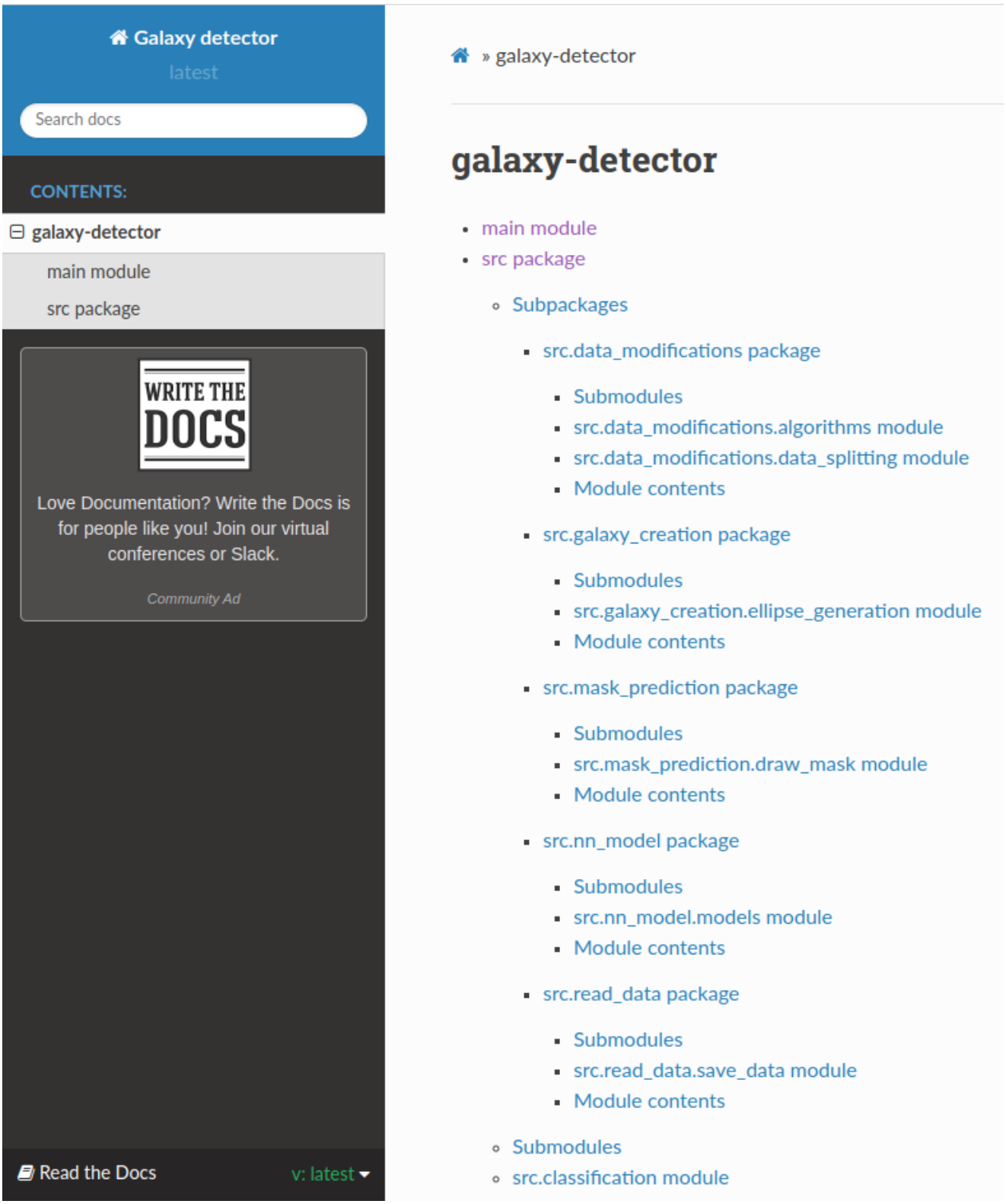

Figura 5.1: Captura con la página inicial de la documentación, publicada en la plataforma *readthedocs.org*