Universidad de Valladolid

Máster universitario

Ingeniería Informática

Trabajo Fin de Máster

Técnicas de Deep Learning para la segmentación de colas de marea en imágenes astronómicas

Realizado por **Darío de la Torre Guinaldo**

✠✠✠

Universidad de Valladolid

29 de junio de 2023

Tutores:

Benjamín Sahelices Fernández

Fernando Buitrago Alonso

Resumen

La formación y evolución de las galaxias es una de las áreas de investigación más activas de la Astrofísica, y uno de los fenómenos astronómicos que aún no se comprenden del todo es la fusión de galaxias, cuyo resultado son las colas de marea. Su estudio se lleva a cabo mediante los llamados telescopios sinópticos, aparatos capaces de cartografiar grandes regiones del cielo. El más moderno de todos ellos es Euclid, un telescopio espacial lanzado por la Agencia Espacial Europea, cuya principal misión será estudiar la aceleración del Universo y la energía oscura. Debido a la masiva cantidad de datos que este telescopio producirá, se evidencia la necesidad de crear herramientas que automaticen las labores tediosas y repetitivas que tradicionalmente han realizado los astrónomos a la hora de analizar estas imágenes. Para ayudar con dicha tarea, en este proyecto se han desarrollado modelos de aprendizaje profundo capaces de clasificar y segmentar colas de marea en imágenes astronómicas. Estos modelos han sido entrenados íntegramente mediante imágenes generadas de forma sintética que simulan galaxias y colas de marea como si fueran observadas por Euclid, aleatorizando todos sus parámetros con el objetivo de disponer de un conjunto ilimitado de galaxias y colas de marea. Es la primera vez, con respecto a trabajos relacionados en este ámbito, que se realiza un flujo por etapas que comprende tanto clasificación de imágenes con y sin cola de marea, como segmentación de los píxeles que conforman la cola de marea detectada. El resultado final es, en primera instancia, un modelo de clasificación con un AUC de 0.987 y, en segundo lugar, un modelo de segmentación semántica alcanzando un índice Dice de 0.83, ambos evaluados sobre un conjunto de datos de prueba de 2000 imágenes simuladas de manera realista.

Descriptores

aprendizaje profundo, redes neuronales convolucionales, visión artificial, segmentación semántica, clasificación, Astrofísica, Euclid, colas de marea, galaxias, simulaciones.

Abstract

The formation and evolution of galaxies is one of the most active areas of research in Astrophysics, and one of the astronomical phenomena that is still not fully understood is the merging of galaxies, which results in tidal tails. Its study is carried out by means of the so-called synoptic telescopes, devices capable of mapping large regions of the sky. The most modern of them all is Euclid, a space telescope launched by the European Space Agency, whose main mission will be to study the acceleration of the Universe and dark energy. Due to the massive amount of data that this telescope will produce, there is a clear need to create tools to automate the tedious and repetitive tasks that astronomers have traditionally performed when analyzing these images. To help with this task, this project has developed deep learning models capable of classifying and segmenting tidal tails in astronomical images. These models have been fully trained using synthetically generated images that simulate galaxies and tidal tails as if they were observed by Euclid, randomizing all their parameters with the aim of having an unlimited set of galaxies and tidal tails. This is the first time, with respect to related work in this field, that a stepwise flow is performed comprising both classification of images with and without tidal tails, and segmentation of the pixels that make up the detected tidal tail. The final result is, in the first instance, a classification model with an AUC of 0.987 and, secondly, a semantic segmentation model achieving a Dice index of 0.83, both evaluated on a test dataset of 2000 realistically simulated images.

Keywords

deep learning, convolutional neural networks, computer vision, semantic segmentation, classification, Astrophysics, Euclid, tidal tails, galaxies, simulations.

Agradecimientos

A todos los que me han apoyado en la realización de este trabajo, y en especial a Benjamín y Fernando, por orientarme con su experiencia y sus conocimientos, y por brindarme en cada momento los recursos que he necesitado.

Índice general

Índice de figuras

Índice de tablas

1: Introducción

En este primer capítulo del trabajo se pretende introducir el contexto y la motivación del problema a tratar, y describir la estructura y organización del resto de este documento. Este Trabajo de Fin de Máster es la continuación del proyecto realizado durante la asignatura de I+d+i del Máster en Ingeniería Informática en modalidad no presencial en el grupo de investigación GCME, de la Escuela de Ingeniería Informática de la Universidad de Valladolid, en colaboración con el grupo de investigación GEELSBE de la Facultad de Ciencias de la Universidad de Valladolid. Los tutores de este trabajo son Benjamín Sahelices Fernández, profesor de la Escuela de Ingeniería Informática y Fernando Buitrago Alonso, profesor de la Facultad de Ciencias.

1.1. Contextualización del problema

Uno de los retos actuales de la Astronomía es averiguar más detalles sobre la evolución y formación de los objetos del espacio profundo, como galaxias. Un fenómeno muy interesante de estudiar es el resultado de la fusión entre dos galaxias, donde la más masiva conserva en cierto modo su forma, y la menos masiva se distorsiona hasta formar lo que se conoce como "cola de marea". El estudio de las colas de marea proporciona información muy útil sobre la evolución y formación del Universo y sus elementos. Por ello, en este proyecto de investigación se trabajará en la creación y entrenamiento de modelos de aprendizaje profundo que sean capaces de detectar estas colas de marea en imágenes astronómicas.

1.2. Estructura del documento

La estructura de esta memoria consta de las siguientes partes. En el capítulo 2 se presenta el plan de proyecto establecido para completar los objetivos planteados. En el capítulo 3 se desarrolla el contexto científico del problema que estamos tratando, es decir, la segmentación de colas de marea en imágenes astronómicas y el contexto tecnológico mediante el cual estamos intentando resolver el problema, que es mediante redes neuronales convolucionales. En el capítulo 5 se explicará en detalle los distintos

aspectos de la metodología de experimentación utilizada para llevar a cabo el proyecto. En el capítulo 6 se mostrarán los resultados obtenidos para poder compararlos y discutir sobre ellos. En el capítulo 7 se finalizará con las conclusiones obtenidas tras realizar el trabajo y con las líneas de trabajo futuras del proyecto.

2: Objetivos y planificación

En este capítulo se detalla la planificación establecida para llevar a cabo este Trabajo de Fin de Máster. Este plan de proyecto se ha diseñado siguiendo el PMBOK[24], un estándar desarrollado por el PMI, que define una serie de buenas prácticas para la gestión de proyectos de cualquier ámbito. No es una metodología cerrada que haya que seguir en su totalidad al pie de la letra, sino más bien una serie de pautas que deben aplicarse según las necesidades y características del proyecto a desarrollar.

Primero se expondrán los objetivos que se esperan cumplir en este proyecto. A continuación, se detalla el alcance de este, que incluye los requisitos y el desglose de las actividades a realizar, junto con su cronograma. Posteriormente, se procederá a la explicación del plan de gestión de riesgos. Por último, se explicará la metodología utilizada para desarrollar el proyecto.

2.1. Objetivos

El objetivo global de este trabajo es desarrollar modelos de aprendizaje profundo capaces de segmentar y distinguir las colas de marea de cualquier otro elemento en imágenes astronómicas con una precisión razonable. La principal dificultad es que no disponemos de suficientes imágenes reales de colas de marea, por lo que tendremos que utilizar distintas herramientas y técnicas para crear datos artificiales lo más realistas posibles. En relación a este objetivo global, se han establecido los siguientes objetivos específicos:

- Estudiar conceptos astronómicos y astrofísicos sobre las colas de marea y la formación de galaxias.
- Aprender a utilizar herramientas astroinformáticas para el tratamiento y la generación de conjuntos de datos de imágenes astronómicas.

- Reforzar los conocimientos sobre redes neuronales convolucionales que se adquirieron en el Trabajo de Fin de Grado y profundizar en conceptos de aprendizaje profundo más avanzados, como las redes en U[26].
- Adquirir destreza con el *framework* PyTorch[22] para trabajar de forma más eficiente en tareas de investigación utilizando aprendizaje profundo.
- Diseñar 3 niveles distintos de experimentación de dificultad incremental en función de la complejidad de las imágenes creadas sintéticamente mediante código. Y para cada uno de estos 3 niveles se realizarán las siguientes tareas:
  - Entrenar modelos de segmentación semántica y medir su rendimiento mediante el coeficiente de Dice.
  - Entrenar modelos de clasificación binaria y medir su rendimiento mediante el área bajo la curva ROC.

Por lo tanto, al finalizar este trabajo, debemos disponer de un modelo de segmentación semántica y de un modelo de clasificación binaria para el tercer nivel de experimentación, es decir, con el conjunto de datos más complejo, con unos desempeños lo suficientemente altos como para satisfacer los requisitos y objetivos establecidos. En el caso de la segmentación semántica debemos alcanzar valores de Dice por encima de 0.75, y en clasificación obtener valores de área bajo la curva ROC de al menos 0.9.

2.2. Alcance

El alcance incluye el trabajo que debe realizarse para completar el proyecto de forma satisfactoria y se define en términos de los objetivos definidos. Para detallar el alcance hay que especificar los requisitos y las tareas necesarias para llevarlos a cabo, que se agrupan de forma jerárquica mediante la estructura de desglose del trabajo (EDT).

2.2.1. Requisitos

Para cumplir satisfactoriamente los objetivos del proyecto se han identificado una serie de requisitos. En las tablas 2.1, 2.2, 2.3, 2.4, 2.5, 2.6 y 2.7 se muestran detallados cada uno de los requisitos necesarios para completar satisfactoriamente este proyecto, especificando los siguientes campos:

- **ID**: Identificador único para cada requisito.
- **Nombre**: Nombre descriptivo para el requisito.
- **Alcance o tiempo**: Indica a cuál de estas magnitudes afecta el cumplimiento o el no cumplimiento del requisito.

ID	**RQ-01**
Nombre	Inicio del proyecto
Descripción	El proyecto deberá comenzar el día 12 de junio de 2023.
Alcance o tiempo	Tiempo
Prioridad	Alta
Partes interesadas	Alumno, tutores

Tabla 2.1: Requisito RQ-01.

ID	**RQ-02**
Nombre	Finalización del proyecto
Descripción	El proyecto deberá finalizar antes del día 2 de julio de 2023.
Alcance o tiempo	Tiempo
Prioridad	Alta
Partes interesadas	Alumno, tutores

Tabla 2.2: Requisito RQ-02.

ID	**RQ-03**
Nombre	Generación del conjunto de datos
Descripción	Se utilizará el software GNUAstro[2] para generar los conjuntos de datos de imágenes astronómicas artificiales que servirán como *input* a los modelos de *deep learning*.
Alcance o tiempo	Alcance
Prioridad	Alta
Partes interesadas	Alumno, tutores

Tabla 2.3: Requisito RQ-03.

- **Prioridad**: Importancia que tiene el requisito para el cumplimiento de los objetivos. Puede ser baja, media o alta.
- **Partes interesadas**: Son las personas o grupos de personas(*stakeholders*) interesadas en el cumplimiento del requisito.

ID	**RQ-04**
Nombre	*Framework* de *deep learning*
Descripción	Se utilizará el *framework* PyTorch[22] en lenguaje Python para implementar todo el código relacionado con el entrenamiento de los modelos de aprendizaje profundo.
Alcance o tiempo	Alcance
Prioridad	Alta
Partes interesadas	Alumno

Tabla 2.4: Requisito RQ-04.

ID	**RQ-05**
Nombre	Registro de la experimentación
Descripción	Se utilizará la plataforma CometML para el registro y seguimiento de todos los resultados de los experimentos realizados.
Alcance o tiempo	Alcance
Prioridad	Baja
Partes interesadas	Alumno

Tabla 2.5: Requisito RQ-05.

ID	**RQ-06**
Nombre	Métrica de comparación de segmentación
Descripción	Para comparar la bondad de los distintos modelos de segmentación entrenados se utilizará principalmente el coeficiente de Sorensen-Dice.
Alcance o tiempo	Alcance
Prioridad	Media
Partes interesadas	Alumno, tutores

Tabla 2.6: Requisito RQ-06.

ID	**RQ-07**
Nombre	Métrica de comparación de clasificación
Descripción	Para comparar la bondad de los distintos modelos de clasificación entrenados se utilizará principalmente el área bajo la curva ROC.
Alcance o tiempo	Alcance
Prioridad	Media
Partes interesadas	Alumno, tutores

Tabla 2.7: Requisito RQ-07.

2.2.2. Estructura de desglose del trabajo

En este apartado se detallan las tareas necesarias para alcanzar los objetivos del proyecto. Se muestran de forma jerárquica mediante la EDT, donde cada tarea principal se desglosa en subtareas. Algunas de estas tareas ya han sido realizadas parcialmente en la estancia en el grupo de investigación de la asignatura de I+d+i, pero en este trabajo se ha iterado de nuevo por algunas de ellas para optimizarlas, mejorarlas y/o corregirlas, como es el caso de estudiar documentación de astronomía y *deep learning* o implementar el código de entrenamiento de modelos mediante PyTorch. En la figura 2.1 se muestra gráficamente la estructura de desglose del trabajo a realizar.

2.3. Plazo

La duración total del proyecto es aproximadamente 150 horas, cuya fecha de inicio es el día 12 de junio de 2023 y cuya fecha de fin es el día 2 de julio de 2023. Es decir,

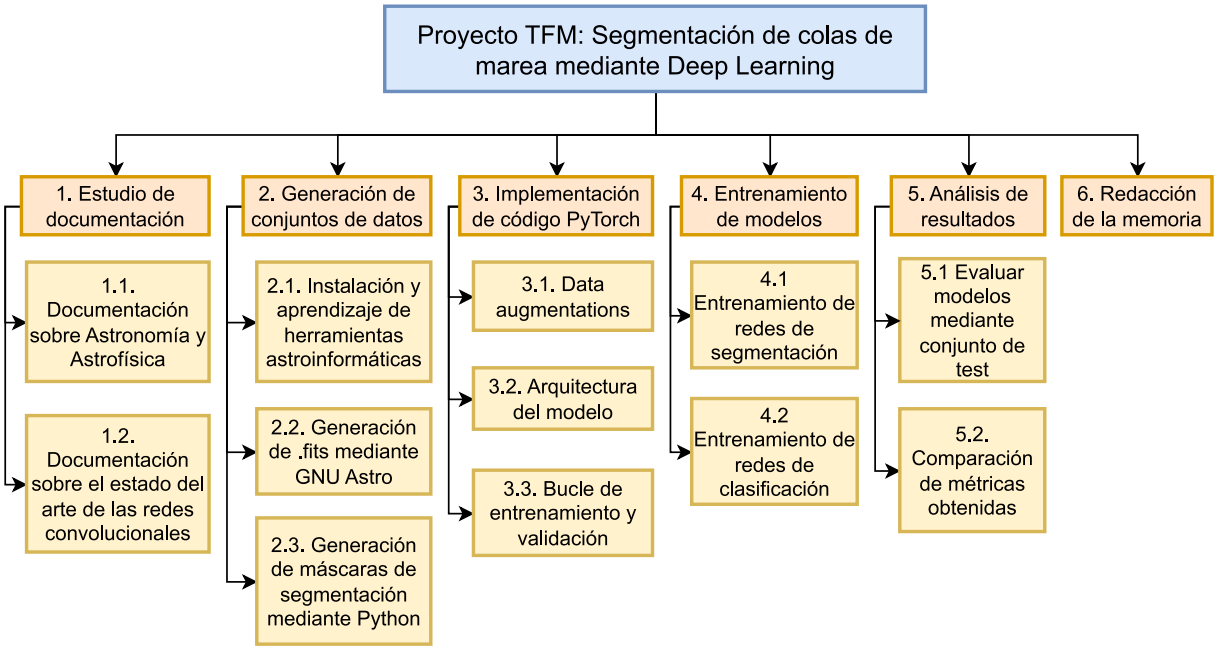


Figura 2.1: Estructura de desglose del trabajo.

	Tareas de la EDT	Duración: 150 horas
1	Estudio de documentación	10 horas
1.1	Documentación sobre astronomía	5 horas
1.2	Documentación sobre redes convolucionales	5 horas
2	Generación de conjuntos de datos	20 horas
2.1	Instalación y aprendizaje herramientas astroinformáticas	10 horas
2.2	Generación de .fits mediante GNU Astro	5 horas
2.3	Generación de máscaras de segmentación con Python	5 horas
3	Implementación de código PyTorch	30 horas
3.1	Data augmentations	10 horas
3.2	Arquitectura del modelo de segmentación	10 horas
3.3	Bucle de entrenamiento y validación	10 horas
4	Entrenamiento de modelos	20 horas
4.1	Entrenamiento de redes de segmentación	10 horas
4.2	Entrenamiento de redes de clasificación	10 horas
5	Análisis de resultados	30 horas
5.1	Evaluar modelos mediante conjunto de test	15 horas
5.2	Comparación de métricas obtenidas	15 horas
6	Redacción de la memoria	40 horas

Figura 2.2: Cronograma de las tareas a realizar.

las tareas a realizar se reparten a lo largo de 3 semanas. El cronograma detallado de las tareas a realizar se muestra en la figura 2.2. Algunas de las tareas son iterativas debido a la naturaleza del trabajo, en concreto, la generación de los conjuntos de datos y el entrenamiento de los modelos a partir de estos se repite 3 veces a lo largo del trabajo porque se han definido 3 niveles de experimentación, como se ha enunciado previamente en la sección 2.1 y como se desarrollará más adelante. La redacción de esta memoria se ha ido elaborando progresivamente desde el primer día de trabajo.

ID	RI-01
Descripción	Enfermedad o lesión grave que impida la realización del trabajo.
Probabilidad	1
Impacto	10
Importancia	10
Estrategia de prevención	Seguir las recomendaciones de las autoridades sanitarias.
Acciones de contingencia	Retrasar la entrega y defensa del trabajo en proporción al tiempo perdido.

Tabla 2.8: Riesgo RI-01

2.4. Gestión de los riesgos

Durante el desarrollo de cualquier proyecto pueden ocurrir situaciones o condiciones que impacten negativamente sobre los objetivos de este. Por ello, es necesario identificar los riesgos conocidos y desarrollar acciones de prevención para minimizar la probabilidad de que ocurran, y acciones de contingencia para minimizar el impacto en caso de que lleguen a ocurrir. En las tablas 2.8, 2.9, 2.10, 2.11, 2.12 y 2.13 se describen los riesgos identificados, especificando los siguientes campos:

1. **ID**: Identificador único para cada riesgo.
2. **Descripción**: Descripción detallada del riesgo.
3. **Probabilidad**: Probabilidad estimada entre 1 y 10 de que se materialice el riesgo.
4. **Impacto**: Valor estimado entre 1 y 10 de las consecuencias que tiene el riesgo sobre el desarrollo del proyecto cuando se materializa.
5. **Importancia**: Importancia del riesgo identificado. Es el producto de la probabilidad por el impacto.
6. **Estrategia de prevención**: Acciones tomadas para evitar que ocurra el riesgo.
7. **Acciones de contingencia**: Acciones tomadas para minimizar el impacto del riesgo cuando ya se ha materializado.

2.5. Metodología de trabajo

La metodología seguida para desarrollar este trabajo ha sido *SCrum Of REsearch* (SCORE)[15], una metodología ágil inspirada en Scrum orientada especialmente a los proyectos de investigación, donde la incertidumbre es muy alta y es necesario adaptarse y evolucionar constantemente. Siguiendo la metodología, se han definido una serie de *sprints* en función de las necesidades del trabajo a realizar. En concreto, se han definido 3 *sprints*, con una reunión con los tutores al final de cada uno de ellos para poner en común el trabajo realizado y confirmar los siguientes pasos a realizar. Cada uno de estos *sprints* está

ID	**RI-02**
Descripción	Avería, malfuncionamiento o cualquier hecho que provoque la falta de disponibilidad de los recursos *hardware* necesarios para realizar el entrenamiento de las redes neuronales.
Probabilidad	1
Impacto	10
Importancia	20
Estrategia de prevención	Plantear la adquisición de nuevos recursos *hardware*, posiblemente servicios de suscripción en la nube.
Acciones de contingencia	Contratar los recursos *hardware* previamente planteados.

Tabla 2.9: Riesgo RI-02

ID	**RI-03**
Descripción	Avería, malfuncionamiento o cualquier hecho que provoque la pérdida del progreso del trabajo realizado en el ordenador personal del alumno, ya sean *datasets*, modelos entrenados o la propia memoria.
Probabilidad	1
Impacto	10
Importancia	20
Estrategia de prevención	Realizar *backups* periódicos de los *datasets*, modelos y memoria del trabajo. Pueden ser en un almacenamiento en la nube o en un disco duro externo.
Acciones de contingencia	Recuperar estos *backups*.

Tabla 2.10: Riesgo RI-03

ID	**RI-04**
Descripción	Incapacidad para conseguir un *dataset* artificial lo suficientemente realista como para que los resultados sean totalmente válidos.
Probabilidad	9
Impacto	8
Importancia	72
Estrategia de prevención	Revisar y estudiar en detalle el código de generación de las imágenes astronómicas sintéticas para que sean suficientemente realistas.
Acciones de contingencia	Rehacer *datasets* y experimentos en la medida de lo posible.

Tabla 2.11: Riesgo RI-04

ID	**RI-05**
Descripción	Incapacidad para obtener modelos entrenados que tengan un desempeño lo suficientemente bueno, de acuerdo a los objetivos planteados.
Probabilidad	7
Impacto	9
Importancia	63
Estrategia de prevención	Utilizar modelos basados en arquitecturas ya probadas en este tipo de problemas a resolver.
Acciones de contingencia	Probar más modelos y rehacer experimentos en la medida de lo posible.

Tabla 2.12: Riesgo RI-05

ID	RI-06
Descripción	Estimación optimista del tiempo y recursos requerido para completar los objetivos del proyecto
Probabilidad	4
Impacto	9
Importancia	36
Estrategia de prevención	Realizar una planificación realista teniendo en cuenta los posibles riesgos que se pueden materializar y que provocarían un retraso en el desarrollo.
Acciones de contingencia	Replanificar en la medida de lo posible y adaptar el alcance al tiempo restante.

Tabla 2.13: Riesgo RI-06

marcado por la creación de un *dataset* y el entrenamiento de los modelos de segmentación y clasificación alimentados con los datos generados. Por lo tanto, los 3 sprints definidos con los objetivos de cada uno son los siguientes:

- *Sprint* 1: Implementación de todo el código en PyTorch necesario para entrenar. Generación del *dataset* 1 y entrenamiento de los modelos correspondientes.
- *Sprint* 2: Generación del *dataset* 2 y entrenamiento de los modelos correspondientes.
- *Sprint* 3: Generación del *dataset* 3 y entrenamiento de los modelos correspondientes.

3: Contexto

En este capítulo se van a introducir algunos conceptos que es necesario comprender para entender el dominio del problema que estamos intentando resolver, tanto a nivel científico, como a nivel tecnológico.

3.1. Contexto científico

A continuación, se van a explicar varios conceptos necesarios para entender la naturaleza científica del problema de la detección y segmentación de colas de marea.

3.1.1. Brillo superficial

El brillo superficial es una medida de luminosidad aparente de un objeto celeste visto desde la Tierra por unidad de área subtendida en el cielo. Sus unidades de medida son magnitudes por arcosegundo al cuadrado: $mag/arcsec^2$. Un arcosegundo es una unidad de medida angular utilizada en Astronomía para medir el tamaño aparente de un objeto en el cielo. Una circunferencia completa son 360^o, cada grado son 60 arcominutos y cada arcominuto son 60 arcosegundos.

Una magnitud (mag) es una medida de luminosidad aparente de un objeto, y cuanto menor es el número de magnitudes, más brillante es el objeto. Este concepto fue introducido por Hiparco de Nicea, un destacado astrónomo y matemático griego del siglo II a.C. Es considerado uno de los fundadores de la Astronomía científica, y se le atribuyen numerosas contribuciones a este campo. Una de ellas fue crear un catálogo de estrellas y clasificarlas según su brillo aparente, desde magnitud 1 (las más brillantes), hasta magnitud 6 (las menos brillantes). Entre una magnitud y otra hay un salto de un factor de aproximadamente 2.5, es decir, una estrella de magnitud 1 es aproximadamente 2.5 veces más brillante que una de magnitud 2, y así sucesivamente. En la figura 3.3 se explica este concepto de forma gráfica.

El brillo superficial tiene en cuenta el área de superficie del objeto, lo que implica que dos objetos pueden tener la misma luminosidad total, pero uno puede tener un valor de

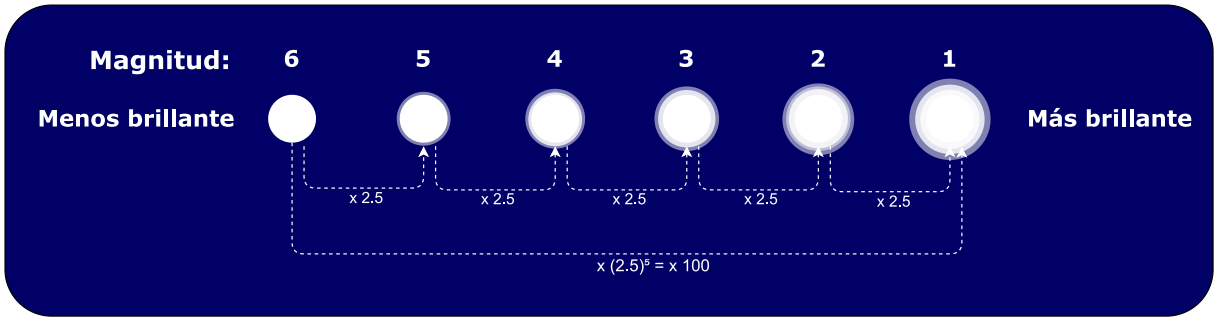


Figura 3.3: Relación entre las unidades de magnitud estelar de Hiparco de Nicea.

brillo superficial más alto si su luz está concentrada en una superficie más pequeña. Por lo tanto, el brillo superficial es una medida utilizada para comparar objetos celestes en función de su tamaño angular aparente y su brillo aparente. La fórmula para su cálculo es la siguiente:

$$\mu = -2.5log\frac{F}{A} + zp = -2.5log\overline{F} + zp + Slogp_x$$

donde F es el flujo que recibimos del objeto en un área A, $\overline{F}$ es el flujo medio, p_x la escala del píxel y zp es el *zeropoint*(la función respuesta del telescopio).

3.1.2. Más allá del Universo local

Las galaxias, los ladrillos que forman nuestro Universo, son objetos con una masa superior a $10^6 M_\odot$(Masas solares, o sea $2 * 10^{30} kg$) compuestos por estrellas, gas en distintas formas y polvo. Este conjunto está vinculado por la gravedad y se mantiene estacionario debido a la rotación y/o a la dispersión de velocidades. Atendiendo a su tamaño, pueden ser enanas, normales o gigantes, y atendiendo a su forma, pueden ser elípticas, lenticulares, espirales, irregulares o peculiares. La formación y evolución de las galaxias es una de las áreas de investigación más activas de la Astrofísica, y busca encontrar respuesta a las siguientes preguntas:

- ¿Cómo se formaron las galaxias?
- ¿Cómo cambian las galaxias con el tiempo?
- ¿Cómo se ha generado un universo tan heterogéneo a partir de un universo homogéneo?

Después del Big Bang, el Universo fue muy homogéneo durante un periodo, pero fueron surgiendo fluctuaciones primigenias que causaron que los gases fueran atraídos hacia áreas de material más denso, originando una jerarquía de agrupaciones de galaxias, que de menor a mayor tamaño serían:

- **Grupo**: Conjunto de hasta 50 galaxias en un diámetro de hasta 2 *Megapársec* (Mpc) y una masa total del orden de $10^{13} M_\odot$.

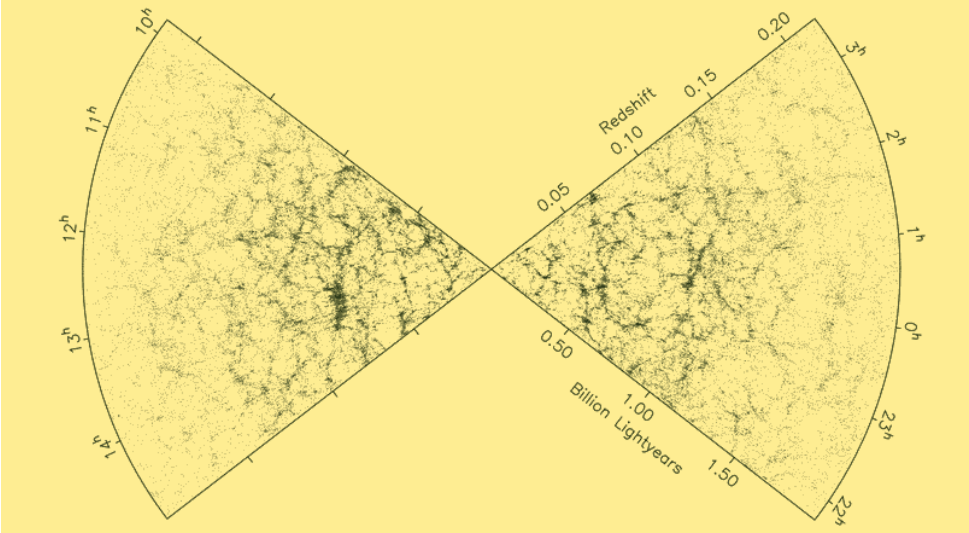


Figura 3.4: Cartografiado realizado con el instrumento 2dF en el Observatorio Anglo-Australiano[6].

- **Cúmulo**: Contienen desde 50 hasta miles de galaxias. Su masa total es del orden de $10^{14} - 10^{15} M_{\odot}$ y su diámetro típico es del orden de 8 Mpc.
- **Supercúmulo**: Decenas de miles de galaxias agrupadas a su vez en cúmulos y grupos. Pueden tener un diámetro de entre 50 y 100 Mpc. El número total de supercúmulos del Universo se estima en 10 millones.
- **Filamento**: son las mayores estructuras del Universo, y se componen de alineaciones de galaxias y supercúmulos. Pueden tener hasta 150 Mpc de diámetro.

En la figura 3.4 se muestra el cartografiado realizado con el instrumento *2 degree Field* (2dF) en el observatorio Anglo-Australiano. Se muestran las posiciones de 62.559 galaxias. La disminución del número de galaxias con la distancia se debe a que, a medida que aumenta la distancia, solo se pueden detectar las galaxias más brillantes. A esta escala se puede observar que el Universo es homogéneo y cómo los cúmulos y supercúmulos se agrupan en filamentos.

Nuestra galaxia, la Vía Láctea, se encuentra dentro del grupo llamado Grupo Local, que a su vez se incluye en el Supercúmulo de Virgo, y este, se engloba en la llamada Laniakea. Por lo tanto, podemos definir el espacio profundo, es decir, todo lo que está más allá del Universo Local, como todo aquello que se encuentra a más de 150 Mpc de distancia, lugar a partir del cual la gravedad de la sobredensidad de galaxias en la que vivimos (Laniakea) desaparece y la expansión del Universo puede tener lugar. Es decir, es el punto a partir del cual las galaxias se ven como objetos pequeños (de uno o varios arcosegundos) y difusos.

Sin embargo, incluso hoy en día, hay muchos componentes y fenómenos del Universo de los que sabemos poco o nada y necesitamos comprenderlos mejor. Uno de esos componentes

es la materia oscura, una forma de materia invisible que interactúa con la materia bariónica(materia ordinaria). Las estrellas se mantienen vinculadas entre sí formando galaxias gracias a la atracción gravitatoria que existe entre ellas, la cual está determinada por las masas de los cuerpos involucrados. Sin embargo, según los cálculos realizados por los astrofísicos, es imposible que se mantengan atraídas entre sí con tanta fuerza gravitatoria solo con sus propias masas estelares (que se calculan a partir de su brillo estelar). Hace falta mucha más masa de la que se observa en el cielo para que las estrellas se mantengan en su posición, por lo que se cree que la materia oscura es la que aporta esa masa que falta para que encajen los cálculos, aunque todavía sigue siendo una incógnita, y se necesita seguir realizando estudios sobre ella.

Usamos las imágenes más profundas del Universo para averiguar más información sobre cómo evoluciona este, ya que, en Astronomía, lejano en el espacio significa, al mismo tiempo, lejano en el tiempo. Es decir, al observar objetos muy distantes, estamos viéndolos con la apariencia que tenían hace mucho tiempo, por lo que nos proporcionan información sobre la formación y evolución del Universo. Por lo tanto, si queremos observar objetos muy profundos y/o poco brillantes, necesitamos utilizar telescopios que sean capaces de capturar estos objetos. Un ejemplo es la figura 3.5, donde se muestra el Hubble Ultra Deep Field, la imagen más profunda (con mayor tiempo de exposición) tomada por el telescopio espacial Hubble. Estas imágenes profundas, además de permitirnos observar objetos en los momentos más tempranos del Universo, nos permiten distinguir con increíble precisión las partes más tenues (las más externas) de galaxias cercanas.

3.1.3. Telescopios sinópticos

Los telescopios son las herramientas utilizadas para poder ver y fotografiar objetos del espacio profundo, ya que así tenemos una superficie colectora de luz mucho mayor que nuestros ojos. Además, permiten ajustar el tiempo de exposición en función de lo que se quiera observar. Este tiempo de exposición es el tiempo durante el cual el sensor óptico de una cámara o un telescopio está expuesto a la luz del objeto observado. Cuanto mayor sea el telescopio, menor tiempo de exposición se requiere para llegar a una cierta señal-ruido para un objeto dado.

Hoy en día, los telescopios más avanzados son los llamados telescopios sinópticos, que están especialmente diseñados para observar grandes áreas del cielo de manera eficiente, gracias a que tienen un campo de visión muy amplio. Un ejemplo de telescopio sinóptico es el Simonyi Survey Telescope, del Observatorio Vera C. Rubin (figura 3.6), ubicado en Chile, que contiene la cámara fotográfica más grande del mundo, de 3.2 GigaPixels. Este telescopio entró en funcionamiento en 2022 con el objetivo de llevar a cabo la *Legacy Survey of Space and Time* (LSST), un estudio que durará 10 años y que pretende responder preguntas sobre la estructura y la evolución del Universo y sus objetos.

Sin embargo, lo ideal es siempre disponer telescopios en el espacio para eliminar el emborronamiento de las imágenes creado por las turbulencias atmosféricas (el llamado *seeing*). Por ello, otro telescopio sinóptico es el telescopio espacial Euclid(figura 3.7) que

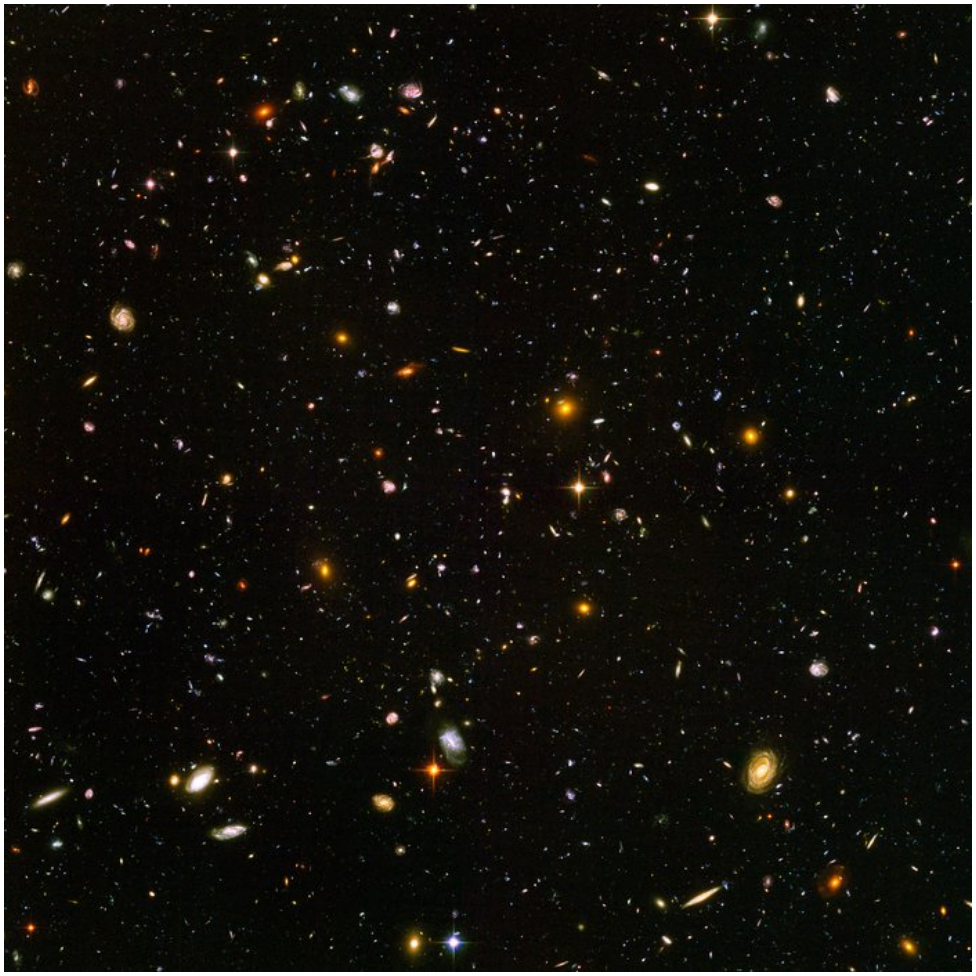


Figura 3.5: Hubble Ultra Deep Field, la imagen más profunda (con mayor tiempo de exposición) tomada por el telescopio espacial Hubble[3].

será lanzado al espacio por la Agencia Espacial Europea en julio de 2023, y cuya principal misión es comprender por qué se acelera la expansión del Universo y cuál es la naturaleza de la fuente responsable de esta aceleración, a la que los físicos denominan energía oscura. Este telescopio estudiará de forma exhaustiva más de un tercio del cielo desde el punto de Lagrange L2, a 1.5 millones de km de la Tierra, observando aproximadamente 1.500 millones de galaxias, por lo que tomará una cantidad masiva de fotografías del espacio profundo, donde se encontrarán muchas colas de marea. Uno de los objetivos posteriores a la realización de este TFM es poder disponer de imágenes reales tomadas por este telescopio para evaluar la calidad de los modelos de aprendizaje profundo que desarrollemos.

3.1.4. Colas de marea

Las colas de marea son las "huellas" que dejan las interacciones entre galaxias, y son estructuras que contiene pistas e información muy importante para entender la formación de las partes más externas de estas galaxias. Para que se forme una cola de marea, en una

Figura 3.6: Montaje del Simonyi Survey Telescope dentro del Observatorio Vera C. Rubin [27]. Es un telescopio sinóptico que, al contrario que el Euclid, se encuentra en tierra.

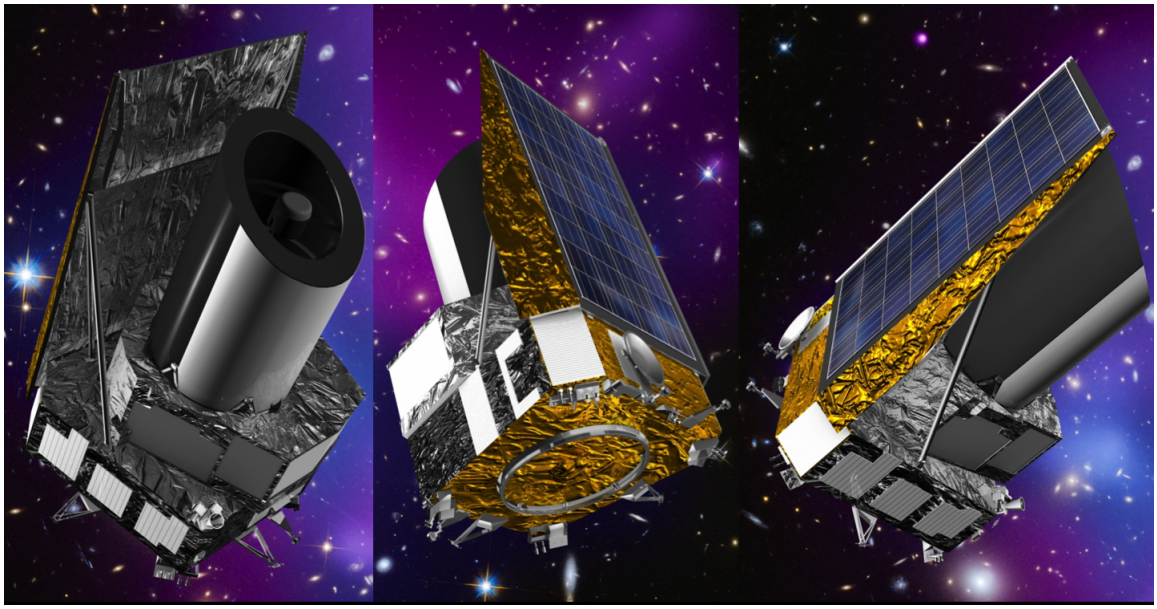


Figura 3.7: Modelo del telescopio Euclid [7].

Figura 3.8: Cola de marea de la galaxia del Renacuajo[25].

fusión de galaxias deben intervenir una galaxia más masiva y una galaxia menos masiva, ya que hay fusiones en las que ambas galaxias tienen masas parecidas, en cuyo caso no se formarían colas de marea, sino que se mezclarían caóticamente las estrellas de cada galaxia. Sin embargo, en el caso de que haya una gran diferencia de masa, la galaxia más masiva sigue manteniendo la morfología aproximadamente que tenía antes de la fusión, pero la galaxia menos masiva cambia completamente (se "deshace") y se convierte en una cola de marea(*tidal tail*). En la figura 3.8 se muestra un ejemplo de una cola de marea tomada por el telescopio Hubble en 2002.

3.1.5. Novedad con respecto a trabajos relacionados

En la actualidad existen varios artículos y trabajos realizados en el ámbito de la detección, clasificación y/o segmentación de objetos de bajo brillo superficial en fotometría del espacio profundo, con algunos elementos en común con el proyecto que se describe en esta memoria. A continuación, se describen las principales características de estos trabajos y se defiende la unicidad de este proyecto y las novedades que implica con respecto a los demás.

En este proyecto se busca realizar tanto detección (mediante clasificación binaria) como segmentación de colas de marea, y además, con un conjunto de imágenes totalmente sintéticas y simuladas por nosotros como aquellas que nos llegarían de telescopios espaciales, es decir, con mejor señal-ruido, y con resolución espacial más pequeña (y por ende mejor),

hablando de escalas espaciales en comparación con aquellas imágenes tomadas desde telescopios en tierra.

Las ventajas de simular nosotros las imágenes es que no dependemos de ninguna fuente externa y podemos generar el número de imágenes que queramos. La desventaja de este aspecto reside en la dificultad para conseguir crear imágenes realistas, y en la incertidumbre del comportamiento de los modelos entrenados cuando se aplique a imágenes reales. Por ello, varios estudios optan por etiquetar a mano imágenes reales, por ejemplo, en el artículo "Characterization of low surface brightness structures in annotated deep images"[34], los autores muestran una nueva herramienta desarrollada para etiquetar "a mano" estructuras de bajo brillo superficial en imágenes astronómicas y guardarlas en una base de datos. Estas máscaras de segmentación realizadas por astrónomos son utilizadas más tarde para entrenar mediante aprendizaje supervisado modelos de *machine learning*.

La utilidad de emplear una herramienta automatizada que hace uso de modelos de inteligencia artificial para realizar tareas repetitivas, como la clasificación, es que el tiempo dedicado por el ser humano a la inspección visual se reduce drásticamente. Esta es una gran ventaja con respecto al procedimiento que utilizan en el artículo "Hidden depths in the local Universe: The Stellar Stream Legacy Survey" [20], donde los investigadores han revisado manualmente miles de imágenes astronómicas tomadas en un cartografiado de galaxias desde tierra, y han separado y estudiado aquellas que poseían cola de marea.

La ventaja de utilizar imágenes de telescopios espaciales (o simulaciones de ellas) en vez de telescopios terrestres, es que nos libramos del emborronamiento producido por las turbulencias atmosféricas, y por tanto, obtenemos imágenes más nítidas. No obstante, muchos estudios se han realizado teniendo en cuenta ese aspecto, por ejemplo, en el artículo "Preparing for low surface brightness science with the Vera C. Rubin Observatory: Characterization of tidal features from mock images"[19]", los autores describen los procedimientos para crear simulaciones (*mock images*) de galaxias y colas de marea en imágenes como si fueran tomadas por el telescopio terrestre del observatorio Vera Rubin y desarrollar técnicas automatizadas para clasificar la morfología de estas galaxias.

Por último, estamos desarrollando técnicas tanto de clasificación como de segmentación porque queremos un obtener un *pipeline* completo en el que, por un lado, se detecte si hay cola de marea o no, y si la hay, se segmente, es decir, se obtengan los píxeles que conforman la cola de marea. Queremos saber los píxeles exactos porque, a partir de ellos, obtenemos sus valores de brillo, y a su vez, a partir de este, podemos conocer tanto el brillo total de un objeto, como su distribución espacial de brillo, lo que nos proporciona información sobre las poblaciones estelares que componen estas estructuras. Ningún estudio sobre colas de marea o estructuras de bajo brillo superficial realiza estas dos etapas conjuntas. Un artículo muy reciente en este ámbito y que sí utiliza *deep learning* es *Identification of tidal features in deep optical galaxy images with Convolutional Neural Networks*[30], donde los autores han utilizado las simulaciones creadas en [19] para entrenar modelos de redes neuronales convolucionales que realizan únicamente clasificación de distintas *tidal features*, pero no segmentación.

En definitiva, consideramos que el trabajo de investigación realizado en este Trabajo de Fin de Máster aporta un enfoque novedoso e innovador al estudio de las colas de marea debido al conjunto de características que reúne y que se han expuesto en esta sección.

3.2. Contexto tecnológico

A continuación, se van a explicar los conceptos de aprendizaje profundo más relevantes para entender la metodología aplicada y las arquitecturas de modelos utilizadas en el capítulo 3.2.6. En esta memoria no se van a exponer los fundamentos del aprendizaje profundo y/o de las redes neuronales en general, ya que, al tratarse este de un Trabajo de Fin de Máster, se parte de la suposición de que el lector está familiarizado con estos conceptos.

3.2.1. Inteligencia artificial

La inteligencia artificial es un área de las ciencias de la computación que abarca una gran cantidad de campos. Distintos autores relevantes en el campo han dado sus propias definiciones de IA. Por ejemplo, John McCarthy, matemático estadounidense doctorado en la Universidad de Princeton, quien es considerado uno de los padres de la inteligencia artifical, y el que acuñó el término en la Conferencia de Dartmouth en 1956, la define con la siguiente frase: "Es la ciencia e ingeniería de la fabricación de máquinas inteligentes". Una definición más moderna es la que dan Stuart Russell y Peter Norvig en su libro *Inteligencia artificial: Un enfoque moderno*[28], publicado por primera vez en 1995 y que se ha convertido en uno de los libros de referencia en las universidades para enseñar los fundamentos de la inteligencia artificial. Ellos proporcionan dos definiciones complementarias: "Es el estudio de cómo hacer que las computadoras realicen tareas que, hasta ahora, requerían inteligencia humana para llevarlas a cabo" y "Es la disciplina que se ocupa de la automatización de la conducta inteligente, es decir, la capacidad de los sistemas para percibir su entorno y actuar en él para lograr objetivos".

La inteligencia artificial tiene muchas aplicaciones y muchas ramas. Una de ellas es el aprendizaje automático o *machine learning*, que se centra en el uso de datos y algoritmos para imitar la forma en que aprenden los humanos, mejorando gradualmente su precisión. Uno de sus pioneros fue Arthur Samuel, científico de Bell Laboratories y más tarde de IBM que, en su artículo de 1959 titulado "Some studies in machine learning using the game of checkers"[29], define el aprendizaje automático de la siguiente forma:"Es el campo de estudio que confiere a los ordenadores la capacidad de aprender sin ser programados explícitamente". Se divide en 2 ramas principales: aprendizaje supervisado y no supervisado, aunque existen otras ramas como, por ejemplo, el aprendizaje semi-supervisado.

El aprendizaje profundo o *deep learning* es, a su vez, un subconjunto del aprendizaje automático (figura 3.9) que tiene una diferencia fundamental con respecto al resto de técnicas. En *machine learning* el humano guía a la inteligencia artificial sobre qué características debe buscar, mientras que en el aprendizaje profundo, la extracción de

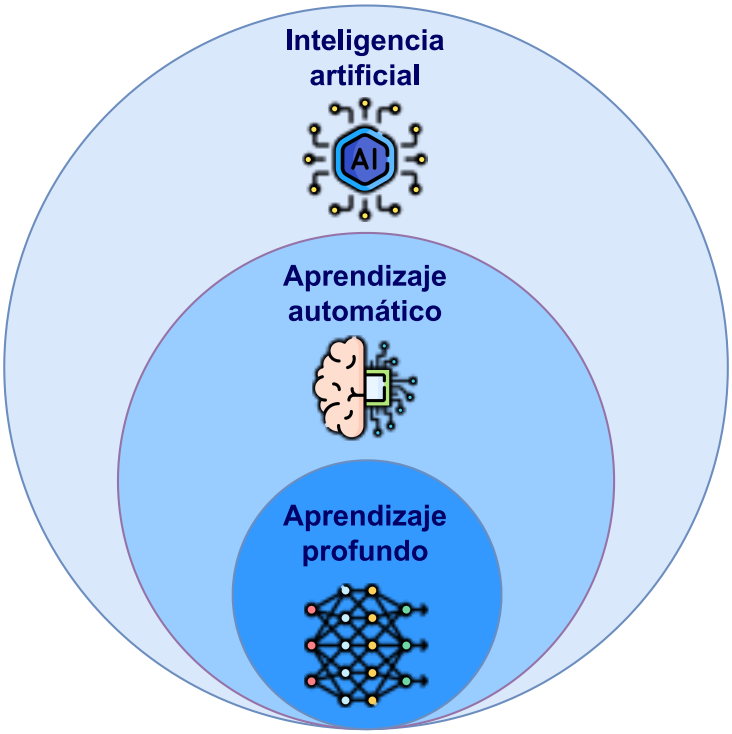


Figura 3.9: Relación entre IA, aprendizaje automático y aprendizaje profundo.

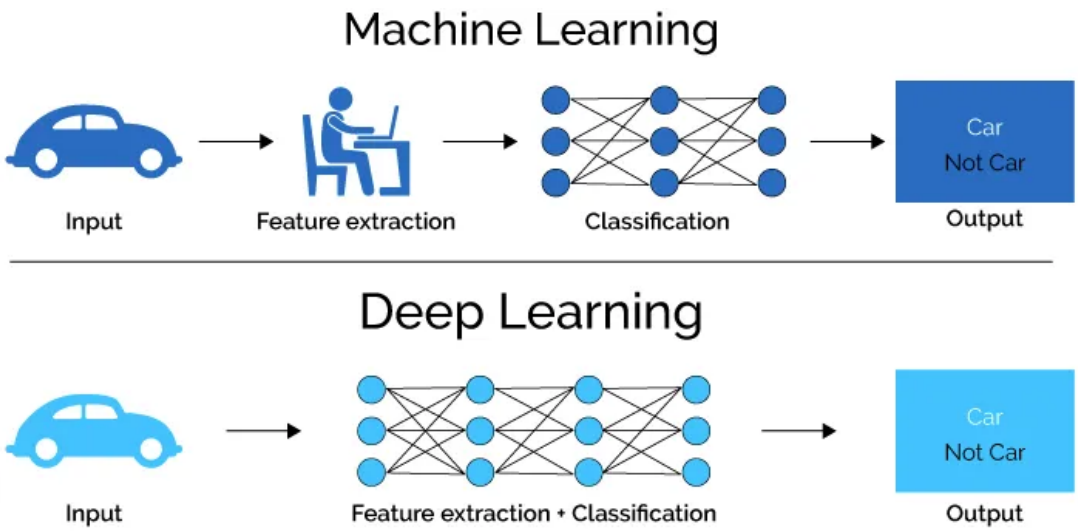


Figura 3.10: Diferencia entre *machine learning* y *deep learning*[18].

características(*feature extraction*) la realiza el propio algoritmo de aprendizaje de forma automática, por lo que ahorra muchísimo tiempo al desarrollador y además, los resultados son más robustos y precisos. En la figura 3.10 se muestra de forma gráfica la diferencia principal entre *machine learning* y *deep learning*.

3.2.2. Redes neuronales convolucionales

Una red neuronal es un modelo que emula el modo en que el cerebro humano procesa la información. Suele constar de un número elevado de unidades de procesamiento interconectadas que representan las neuronas del sistema nervioso humano y se organizan en capas. Una red neuronal es un algoritmo de aprendizaje automático que recibe información (*input*) a través de la capa de entrada, la procesa en la o las capas intermedias u ocultas, y genera el resultado(*output*) por la capa de salida. Por lo tanto, en su forma más simple, una red neuronal puede llegar a estar formada únicamente por 3 capas. Sin embargo, cuando una red neuronal está formado por multitud de capas ocultas que realizan operaciones complejas sobre cantidades masivas de datos estructurados y no estructurados, estamos ya hablando de aprendizaje profundo. Hoy en día se utilizan en una gran cantidad de aplicaciones con un amplio impacto en la sociedad: visión artificial, reconocimiento del lenguaje natural, robótica, diagnosis médica, conducción autónoma, traducción de idiomas... Pero también se pueden utilizar para ciertas tareas repetitivas más concretas que pueden ahorrar una gran cantidad de tiempo al ser humano como coloreado de fotos en blanco y negro o subtitulado de imágenes, o incluso pueden sustituir al ser humano en ciertos trabajos creativos que hasta estos últimos años no se pensaba que fuera posible, como la composición de música o la pintura de cuadros.

Las redes neuronales convolucionales son uno de los principales tipos de redes neuronales profundas, y se utilizan comúnmente en el procesamiento de imágenes y el reconocimiento de patrones. La arquitectura de una *Convolutional Neural Network* (CNN) está diseñada para procesar imágenes en bruto, es decir, la entrada de la red son los píxeles de una imagen, y la salida depende del problema que estemos intentando resolver. Mediante redes convolucionales se pueden resolver distintos tipos de cometidos:

- **Clasificación**: Es una tarea que consiste en asignar a una imagen de entrada una etiqueta que representa la clase o categoría a la que pertenece. Una CNN realiza muy bien esta tarea porque es capaz de reconocer patrones complejos en los píxeles y asignar las etiquetas adecuadas. Algunas de las aplicaciones donde se emplea esta tarea es en la clasificación de imágenes y en la detección de fraude en transacciones financieras.

- **Regresión**: Esta tarea consiste en predecir, en función de la entrada, valores numéricos continuos en lugar de etiquetas discretas como en clasificación. Algunas de sus aplicaciones son la predicción del precio de acciones y bienes raíces, el análisis de la calidad del aire o el diagnóstico médico.

- **Detección de objetos**: Consiste en localizar y clasificar objetos en una imagen o vídeo. Es decir, la red neuronal es capaz de detectar la presencia de objetos en una imagen, determinar su ubicación precisa y asignar una etiqueta o clase a cada objeto detectado. Típicamente, la salida de las redes de detección son las coordenadas de 2 esquinas opuestas de las *bounding boxes* que contienen los objetos junto con sus etiquetas. Las aplicaciones más comunes de esta tarea son la seguridad y vigilancia

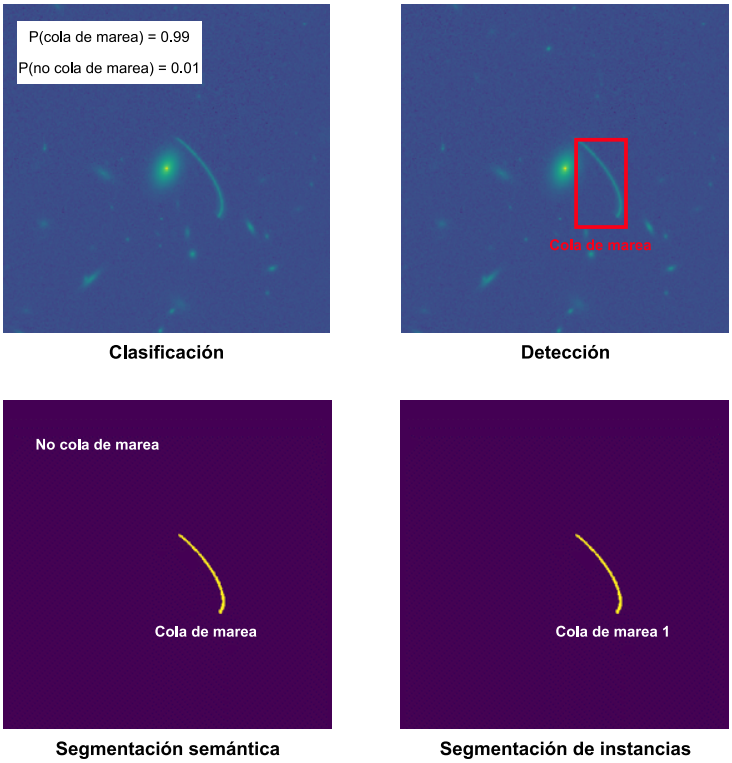


Figura 3.11: Distintas tareas de aprendizaje profundo aplicadas al supuesto del problema.

por vídeo o la automatización industrial en el contexto del *belt picking* y del *bin picking*.

- **Segmentación semántica**: Es una tarea en la que se clasifica cada píxel de una imagen en diferentes clases semánticas. Por lo tanto, la entrada y la salida de la red tienen el mismo número de píxeles, ya que la salida contiene lo que se conoce como máscara de segmentación, que es una imagen donde el valor de cada píxel indica la categoría a la que pertenece. Tiene diversas aplicaciones, que van desde la localización de elementos en imágenes médicas hasta la conducción autónoma.

- **Segmentación de instancias**: Esta tarea es ligeramente más compleja que la segmentación semántica que acabamos de explicar, ya que implica identificar y segmentar cada objeto individual en una imagen, asignándole una etiqueta única y permitiendo distinguir objetos superpuestos o que se parecen entre sí. Por lo tanto, esta técnica ofrece la posibilidad de seguir la pista a objetos individuales a medida que se mueven y cambian de forma. Las áreas de aplicación son similares a las de la segmentación semántica: sistemas de vigilancia, conducción autónoma de vehículos y robótica industrial.

En la figura 3.11 se muestra, a modo de ejemplo, cómo sería aplicar clasificación, detección, segmentación semántica y segmentación de instancias respectivamente al contexto del problema que hemos planteado. No obstante, en este proyecto trabajaremos únicamente

las tareas de clasificación, para distinguir entre imágenes con y sin cola de marea, y de segmentación semántica, para obtener los píxeles que pertenecen a la cola de marea detectada. Se utilizarán redes neuronales convolucionales profundas de naturaleza distinta para ambas tareas. Por ello, a continuación, se explican algunos conceptos fundamentales de este tipo de redes que justifican su idoneidad para alcanzar los objetivos de este proyecto.

Convolución

Una convolución es una operación matemática que se utiliza en diversas áreas, como la ingeniería, la física, la estadística, el procesamiento de señales y la visión artificial. En visión artificial o *computer vision* la convolución se refiere a la operación matemática de aplicar un filtro o *kernel*(una matriz de valores numéricos) a una imagen con el fin de resaltar ciertas características o patrones de la imagen. En las arquitecturas de las CNN se utilizan las capas convolucionales para aplicar filtros que extraen características de la imagen. Las características extraídas son las que utiliza la red para clasificar o detectar objetos en una imagen. Una operación de convolución está definida por los siguientes parámetros:

- **Filtro o *kernel***: Es una matriz numérica que se utiliza para aplicar una transformación a la imagen de entrada. Consiste en una ventana deslizante de números que se desplaza sobre la entrada. En una operación de convolución, la entrada puede tener cualquier número de dimensiones y, en consecuencia, el *kernel* que se aplica tendrá el mismo número de dimensiones. Si la entrada es una imagen en 2D, el *kernel* será una matriz bidimensional, pero si la entrada es un tensor en 3D, el *kernel* será una matriz tridimensional. En este trabajo nos centraremos solo en visión artificial en imágenes planas, por lo que las convoluciones que aplicaremos serán siempre bidimensionales. En estos casos, en cada posición del desplazamiento, los valores del *kernel* se multiplican por los píxeles correspondientes de la imagen de entrada y se suman para producir un valor de salida.
- **Paso o *stride***: Es el número de píxeles que el *kernel* se desplaza en cada paso durante la operación de convolución. El uso del *stride* permite controlar el tamaño de la salida de la convolución.
- **Relleno o *padding***: Es el número de píxeles que se añaden alrededor de la imagen de entrada antes de aplicar la convolución. El objetivo del *padding* es permitir que el *kernel* se desplace más allá de los bordes de la imagen de entrada y produzca una salida del mismo tamaño que la entrada.
- **Dilatación o *dilation***: Este parámetro se refiere a la adición de ceros entre los elementos del *kernel*, de forma que se aumenta la distancia entre los píxeles considerados durante la convolución. Cuando el valor de la dilatación es mayor que cero, se dice que estamos realizando una convolución dilatada.

En el artículo "*A guide to convolution arithmetic for deep learning*"[10], los autores Vincent Dumoulin y Francesco Visin explican gráficamente y de forma muy clara las

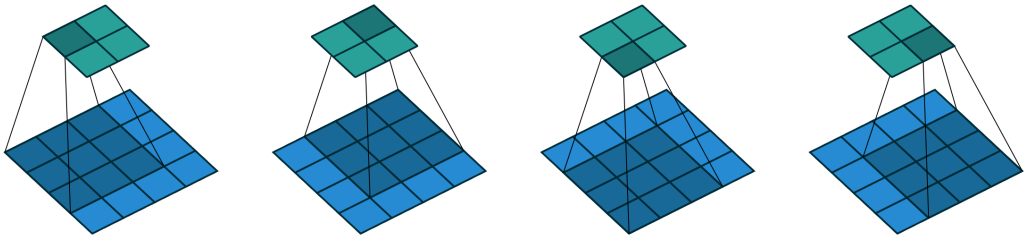


Figura 3.12: Convolución de un *kernel* de 3x3 sobre un *input* de 4x4 usando *stride* de 1 y sin *padding*[10].

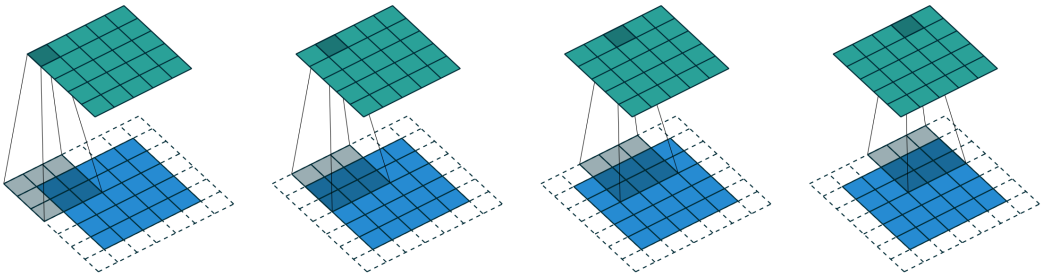


Figura 3.13: Convolución de un *kernel* de 3x3 sobre un *input* de 5x5 usando *stride* de 1 y un *padding* de 1x1[10].

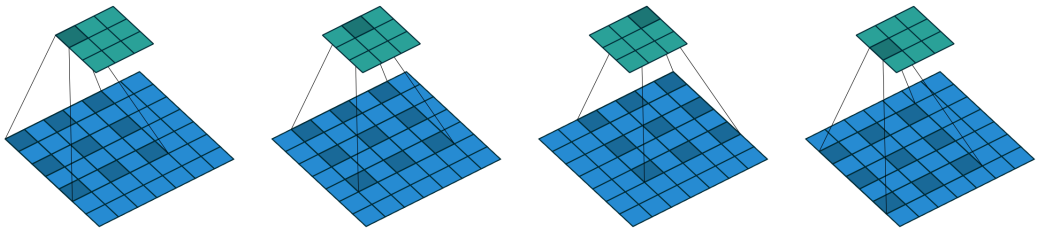


Figura 3.14: Convolución dilatada de un *kernel* de 3x3 sobre un *input* de 7x7 con dilatación de factor 2 usando *stride* de 1 y sin *padding*[10].

convoluciones. En las figuras 3.12, 3.13 y 3.14 se muestran algunos casos concretos para clarificar los conceptos recién explicados. En estas figuras, el bloque azul claro es el *input*, el bloque azul turquesa es el *output* y la proyección sombreada es el *kernel* aplicado. En la 3.12 se muestra cómo serían los primeros pasos del proceso para realizar una convolución de un *kernel* de 3x3 sobre un *input* de 4x4 usando un *stride* de 1 y sin ningún *padding*. En el caso de la figura 3.13 se muestra una operación de convolución con un *kernel* de 3x3 sobre un *input* de 5x5 usando *stride* de 1 y un *padding* de 1x1. Por último, en la figura 3.14 se muestra una convolución dilatada aplicando un *kernel* de 3x3 sobre un *input* de 7x7 con un factor de dilatación de 2 usando, además, un *stride* de 1 y un *padding* de 0.

Durante la construcción de las arquitecturas de los modelos de redes neuronales convolucionales uno de los factores más importantes a tener en cuenta es cuáles deben ser las dimensiones de entrada de un tensor a una capa convolucional y cuáles van a ser las dimensiones de salida. Los valores de altura(*Height*, H), y anchura(*Width*, W) del *output* de una capa convolucional 2D vienen dados por las siguientes fórmulas:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 * padding[0] - dilation[0] * (kernel\_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 * padding[1] - dilation[1] * (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor$$

En estas fórmulas H_{out} (o W_{out}) es la altura (o anchura) del tensor de salida, H_{in} (o W_{in}) es la altura (o anchura) del tensor de entrada y el resto de variables son las que hemos explicado con anterioridad. Mediante estas fórmulas somos capaces de calcular los tamaños de entrada y de salida de los tensores cuando van pasando a través de las diferentes capas convolucionales de los modelos que queramos construir.

Convolución traspuesta

Al contrario que las convoluciones normales que hemos explicado en el apartado anterior, que permitían reducir la resolución de la imagen, las convoluciones traspuestas son operaciones que se utilizan para aumentar la resolución de la imagen de salida con respecto a la imagen de entrada. Esta operación requiere argumentos comunes a una convolución normal, es decir, *kernel*, *stride*, *padding* y *dilation*, sin embargo, puede tomar un argumento más:

- **Relleno de salida o *output padding***: Es el número de píxeles que se añaden a cada lado de cada dimensión de la imagen de salida después de aplicar la convolución traspuesta. El objetivo de este parámetro es permitir que la salida de una convolución traspuesta tenga el mismo tamaño que la imagen de entrada original.

En la figura 3.15 se muestra un ejemplo de convolución traspuesta donde se usa un *kernel* de 3x3 sobre un *input* de 2x2 con un *stride* de 1 y un *padding* de 2. De esta forma, la imagen de salida tiene más resolución que la imagen de entrada.

En las capas de convoluciones traspuestas también es importante calcular correctamente las dimensiones de entrada y de salida, y para ello, se utilizan las siguientes fórmulas, donde el signficado de cada variable es el mismo que en las fórmulas del apartado anterior:

$$H_{out} = (H_{in}-1)*stride[0]-2*padding[0]+dilation[0]*(kernel\_size[0]-1)+output\_padding[0]+1$$

$$W_{out} = (W_{in}-1)*stride[1]-2*padding[1]+dilation[1]*(kernel\_size[1]-1)+output\_padding[1]+1$$

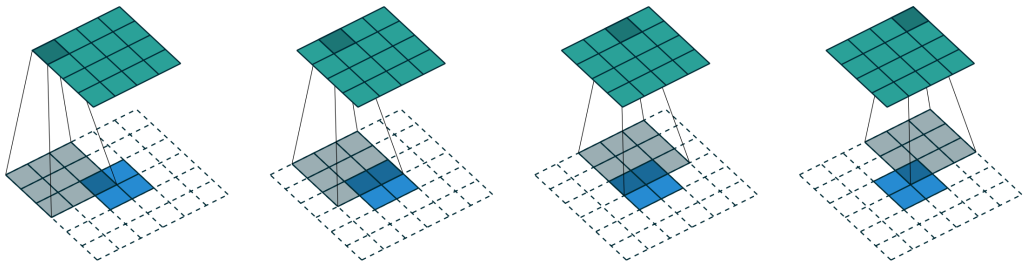


Figura 3.15: Convolución traspuesta usando un *kernel* de 3x3 sobre un *input* de 2x2 con un *stride* de 1 y un *padding* de 2[10].

ReLU

En las redes neuronales, una función de activación es una función matemática que se aplica a la salida de cada neurona en una capa de la red neuronal. Permiten que la red pueda modelar relaciones no lineales entre las entradas y las salidas. Además, la función de activación calcula la salida de una neurona a partir de su entrada ponderada y su sesgo, que son parámetros entrenables que le dan a la neurona la capacidad de aprender. Hay distintas funciones de activación que se utilizan comúnmente en *deep learning* según la naturaleza del problema que estemos resolviendo, como la función *Rectified Linear Unit* (ReLU), la función sigmoide, la función *Softmax* o la función tangente hiperbólica.

En las arquitecturas de segmentación que vamos a construir vamos a utilizar en múltiples ocasiones capas con la función ReLU por su simplicidad y eficiencia en el cálculo y, sobre todo, porque puede ayudar a prevenir el problema de desvanecimiento del gradiente (*vanishing gradient*) durante el entrenamiento de las redes neuronales. Su fórmula es la siguiente:

$$ReLU(x) = (x)^+ = ma(0, x)$$

Es decir, si la entrada es negativa la salida toma el valor 0, y si la entrada es positiva, la salida toma el valor de la entrada. En la figura 3.16 se muestra la gráfica de los valores que toma esta función.

3.2.3. ResNet

Una de las arquitecturas de redes neuronales convolucionales más relevantes en los últimos tiempos es la ResNet(*Residual Network*), introducida en 2015 en el artículo "*Deep Residual Learning for Image Recognition*"[13]. Estas redes son muy utilizadas en tareas de visión por computadora, especialmente en clasificación de imágenes, y por ello las vamos a utilizar más tarde en nuestra experimentación.

La característica principal de estas redes es que, al contrario que las redes convolucionales clásicas, que aprenden directamente la representación de una función, asimilan la diferencia entre la representación esperada y la representación real, es decir, el residuo.

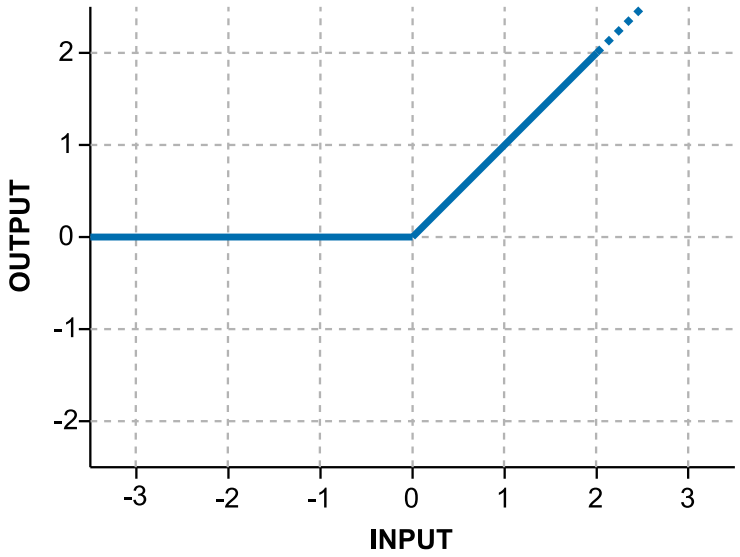


Figura 3.16: Gráfica de la función de activación *Rectified Linear Unit* (ReLU).

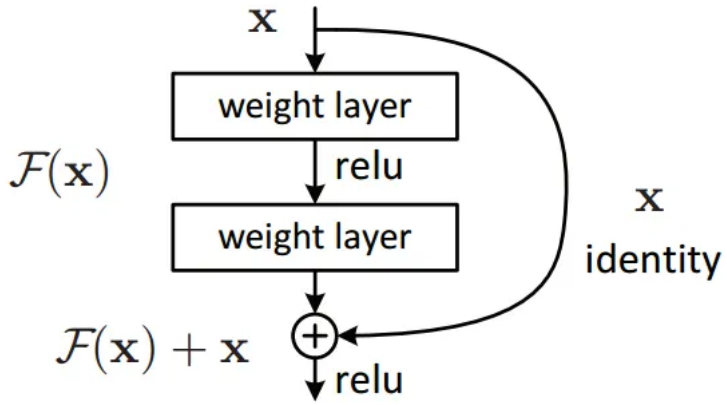


Figura 3.17: Bloque residual de los que se compone una ResNet[13].

Esto se logra mediante los bloques residuales(figura 3.17), de los que se compone una ResNet.

3.2.4. Autoencoders

Un *autoencoder* es un modelo de red neuronal que aprende a comprimir (codificar) los datos hasta una forma muy reducida y a descomprimirlos (decodificar) para reconstruir los originales. La arquitectura básica de un *autoencoder* tiene 2 partes bien diferenciadas, como se puede observar en la figura 3.18:

1. ***Encoder***: Se encarga de reducir los datos de entrada hasta un espacio latente de baja dimensionalidad.

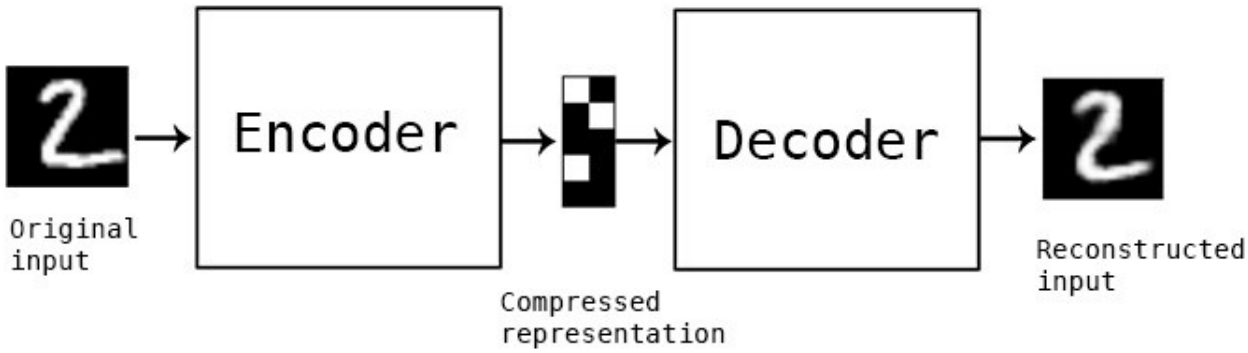


Figura 3.18: Arquitectura de un *autoencoder*[5].

2. ***Cuello de botella***: Es la capa donde los datos están representados mediante su espacio latente. Es decir, están comprimidos lo máximo posibles sin perder sus características principales y su información relevante.

3. ***Decoder***: Se encarga de reconstruir los datos originales a partir de su espacio latente para devolver la salida de la red.

La salida de la red se compara con la entrada para calcular las diferencias entre ambas mediante una función de pérdida o *loss function*. Después se aplica el algoritmo de retropropagación o *backpropagation* para minimizar la pérdida. El objetivo es que la red aprenda a reconstruir la imagen lo más parecido posible a la imagen de entrada. Algunas de las aplicaciones de este tipo de redes son las siguientes:

- **Detección de anomalías**: La red ha aprendido a reconstruir un cierto tipo de imágenes, por lo que si le pasamos una imagen que sí sabe reconstruir, el *loss* será muy bajo, pero si le pasamos una imagen distinta a lo que sabe, es decir, una anomalía, el *loss* será muy alto y podremos detectar estos casos.

- **Eliminación de ruido**: También podemos enseñar a la red a eliminar el ruido de imágenes, siendo la entrada la imagen con ruido, y la salida, la imagen sin ruido. Para ello, en el *input* debemos pasar la imagen sin ruido, y la imagen corrompida con ruido, de forma que el *output* de la red sea la imagen corrompida del *input* con el ruido quitado por la red, y el *loss* se optimiza en función de la comparación de esta salida con la imagen limpia original.

3.2.5. Redes en U

El objetivo principal de este trabajo es diferenciar la cola de marea de la galaxia y del fondo de la imagen, y para ello, se utiliza la técnica de la segmentación semántica, que asocia una etiqueta a cada píxel presente en una imagen. Para realizar la segmentación, se requiere utilizar una arquitectura cuya entrada sea una imagen y cuya salida sea

una imagen del mismo tamaño con un valor en cada píxel que indique a qué categoría pertenece, es decir, se necesita un tipo especial de *autoencoder*. Uno de los modelos de *deep learning* más reconocidos para esta tarea es la **U-Net**, que se utilizó por primera vez para la segmentación de imágenes biomédicas[26]. La U-Net está formada por 2 partes bien diferenciadas similares a las de un *auotencoder*, como se puede ver en la figura 3.19:

- ***Encoder*** o camino de contracción: Es un conjunto de capas convolucionales y de *max pooling* que van reduciendo el tamaño de la imagen a la vez que aumentan el número de canales para crear un mapa de características hasta llegar al espacio latente.
- ***Decoder*** o camino de expansión: Es un conjunto de convoluciones traspuestas que van aumentando el tamaño de la imagen y reduciendo el número de canales hasta devolverla a su forma original.

La principal diferencia con un *autoencoder* convencional es que la U-Net contiene *skip connections* entre la capa del *encoder* y del *decoder* del mismo nivel, es decir, en una capa del *decoder* se concatena la salida de la capa anterior del *decoder* con la capa del mismo nivel del *encoder*, y de esta forma, la U-Net es capaz de aprender detalles de grano fino de la parte del *encoder* para reconstruir la imagen segmentada.

3.2.6. Astroinformática

La **Astroinformática** es un campo interdisciplinar en el que se combinan Astronomía, Informática, Ciencia de datos y tecnologías de la información y la comunicación. Tiene como principal objetivo desarrollar métodos, aplicaciones y herramientas de ciencia de datos y estadística para aplicarlos al estudio de la astronomía [4].

Es un área de investigación muy amplia que tiene herramientas y conceptos propios que son necesarios conocer y manejar para entender el contexto del trabajo. En este trabajo de investigación se han utilizado algunas herramientas astroinformáticas para la generación y el tratamiento de los datos astronómicos y herramientas de programación para la construcción y el entrenamiento de los modelos de aprendizaje profundo.

- **GNUAstro** [2]: *GNU Astronomy Utilities* es un paquete oficial de GNU compuesto por un conjunto de librerías para manipular y analizar datos astronómicos. Se utiliza mediante una terminal en línea de comandos de Unix.
- **SAOImage DS9**[33]: Es una aplicación para visualizar datos e imágenes astronómicos. Uno de los formatos de fichero que soporta este software y que más vamos a utilizar es el `.fits`(*Flexible Image Transport System*), que contiene una cabecera con una serie de metadatos y un contenido que son los valores de los píxeles de la imagen astronómica almacenados con una precisión de punto flotante de 64 bits.

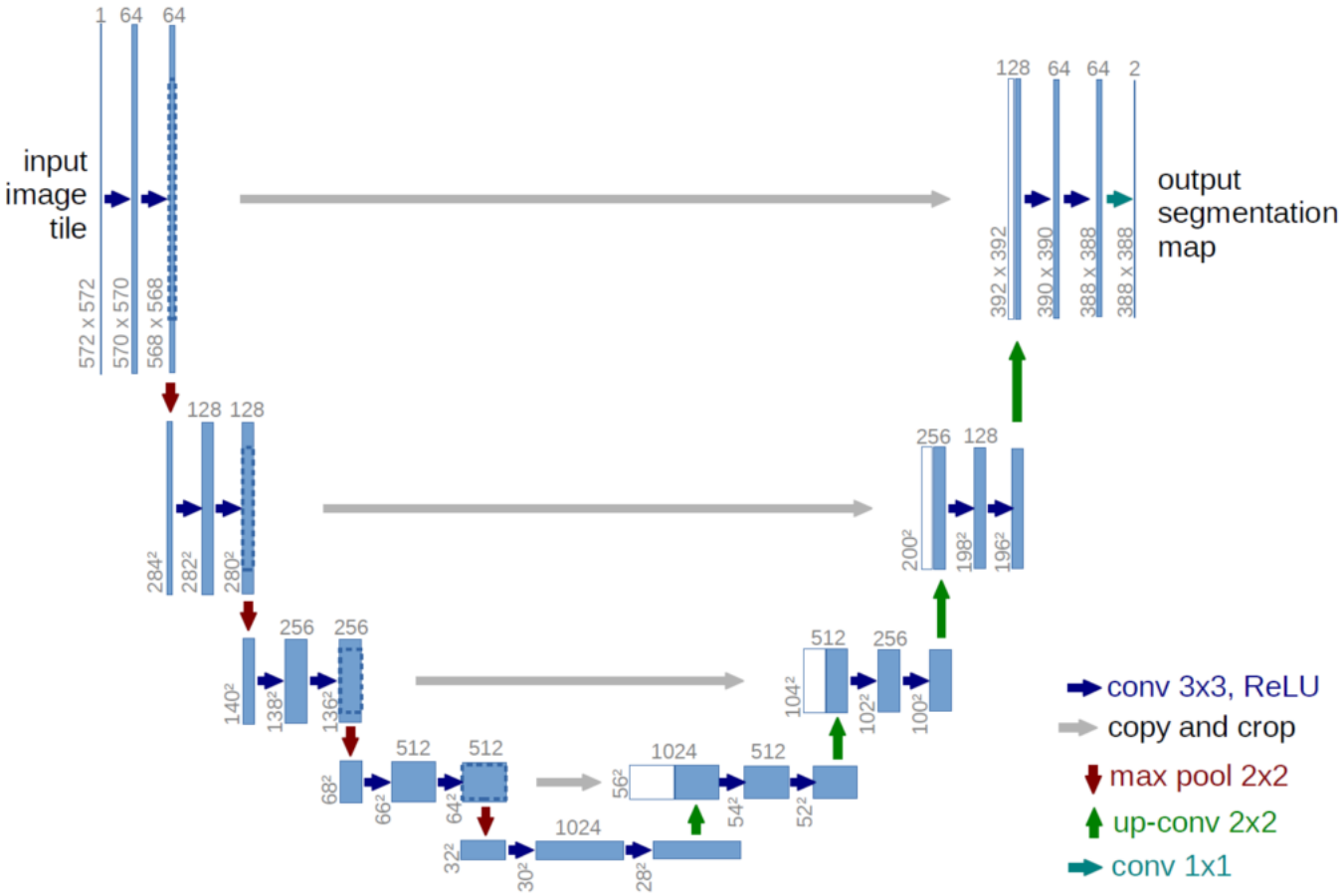


Figura 3.19: Arquitectura de la U-Net[26].

- **Astropy**[23]: Paquete de Python que contiene utilidades para el tratamiento de datos astronómicos. En este trabajo se usará principalmente para la manipulación de ficheros `.fits`.
- **MontagePy**[16]: Paquete que permite utilizar el software Montage en Python. Este software fue creado por el Instituto Tecnológico de California en 2005, y contiene un conjunto de herramientas para reproyectar, crear mosaicos y obtener imágenes RGB a partir de imágenes astronómicas en formato `.fits` al igual que `Astropy`.

Aparte de estas herramientas específicas de la Astroinformática, para implementar este trabajo se han utilizado recursos hardware y utilidades software de ámbito más generalista que se especifican a continuación.

Hardware

El entrenamiento de redes neuronales profundas requiere de GPU potentes con una gran cantidad de RAM para poder realizarlo en un tiempo razonable, ya que el entrenamiento en

CPU es mucho más lento. Por ello, los tutores han proporcionado varios equipos diferentes del GIR en los que se han ido ejecutando los códigos de entrenamiento creados:

- Máquina 1(Facsimile): GPU: Nvidia GeForce RTX3060 de 12 Gb de RAM. Procesador: Intel Xeon de 6 núcleos Memoria RAM: 128 Gb.
- Máquina 2(Hal): GPU: Nvidia RTXA5000 de 24 Gb de RAM. Procesador: AMD Ryzen 5900X de 12 núcleos. Memoria RAM: 64 Gb.
- Máquina 3(Nemesys): GPU: 2 Nvidia GeForce RTX3060 de 12 Gb de RAM cada una. Procesador: Intel i5 11700. Memoria RAM: 64 Gb.

La implementación y preparación del código y sus correspondientes pruebas previas se han realizado en una máquina local del estudiante.

Software

El principal lenguaje de programación utilizado ha sido Python, aunque también se han utilizado *scripts* en *bash*. El entorno de programación usado ha sido Jupyter Notebook[17], una herramienta pensada para el prototipado de experimentos, ya que permite ejecutar trozos de código por separado para ver los resultados paso a paso. Los paquetes de Python se han instalado en un entorno creado mediante Anaconda, que es una distribución orientada al uso científico de Python, ya que viene con una gran cantidad de paquetes instalados por defecto y permite gestionar distintos entornos con distintas versiones de los paquetes cómodamente. Los principales paquetes de Python utilizados durante este trabajo de investigación han sido los siguientes:

- **PyTorch**[22]: Es una librería de tensores optimizados para realizar *deep learning* usando tanto GPU como CPU. Es el *framework* principal que se usará en este trabajo.
- **Numpy**[12]: Es la librería de Python fundamental para computación científica, ya que provee todo tipo de operaciones optimizadas para *arrays* multidimensionales.
- **CometML**: Herramienta para el registro en la nube de datos de experimentos ejecutados en Jupyter Notebook. El procedimiento básico consiste en declarar las métricas que se calculan en cada época del entrenamiento y que se podrán visualizar en la interfaz web para su posterior análisis.

4: Metodología de experimentación

En este capítulo se explican en detalle los aspectos relevantes del desarrollo del proyecto. Al ser este un proyecto de aprendizaje automático, es vital empezar hablando del conjunto de datos utilizado, incluyendo su obtención y/o creación, limpieza y *data augmentation* utilizado en los entrenamientos de los modelos. Después se detallará la arquitectura del modelo utilizado y, por último, se justificará la función de pérdida y la métrica empleadas para evaluar la calidad del modelo.

4.1. Conjunto de datos

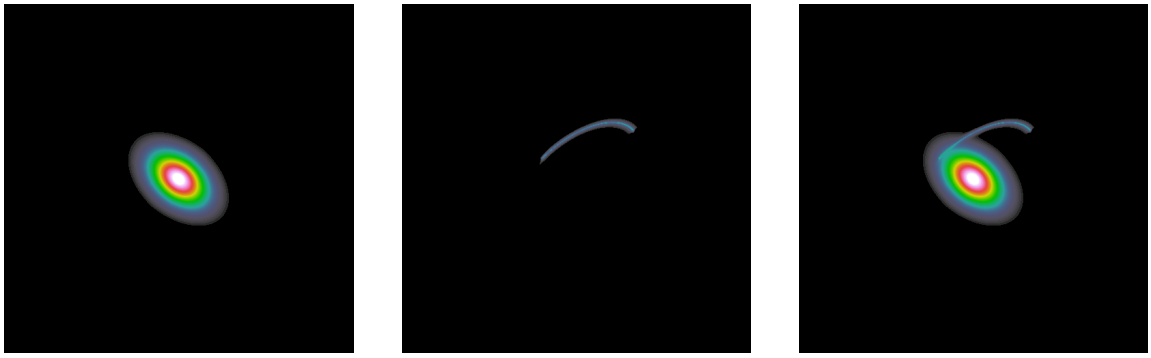
Para crear el conjunto de datos se ha usado la herramienta `gnuastro` y un *script* en lenguaje de comandos de Shell proporcionado por uno de los tutores del trabajo, Fernando Buitrago, cuyo procedimiento detallado es el siguiente[32]:

Modelizamos nuestras galaxias como dos funciones de Sérsic[31] bidimensionales (una para el bulbo y otra para el disco). Una función de Sérsic es una expresión matemática que describe la forma del perfil de brillo de una galaxia utilizando la ley de Sérsic. El perfil de brillo es una descripción de cómo se distribuye la luz emitida por una galaxia a diferentes distancias desde su centro. Esta ley se utiliza para ajustar los datos observados de intensidad de luz en función del radio y así obtener los parámetros que caracterizan el perfil de brillo de una galaxia. La función de Sérsic se define mediante la siguiente fórmula:

$$I(r) = I_0 exp\Big[- \left(\frac{r}{r_0}\right)^{\frac{1}{n}} \Big]$$

donde $I(r)$ es la intensidad de luz en un radio r, I_0 es la intensidad de luz en el radio r_0, que es igual a la longitud de escala del disco, o sea, donde decae el brillo un factor e, y n es el denominado índice de Sérsic. Dicho índice adquiere valores cercanos a 1 para discos y valores cercanos a 4 para esferoides [8] [11].

Por otro lado, las colas de marea se modelan como funciones bidimensionales cortadas tanto radial como acimutalmente. Para cada galaxia, hacemos aleatorio los tamaños del



(a) Galaxia (b) Cola de marea (c) Galaxia y cola de marea

Figura 4.20: Galaxia, cola de marea y combinación de ambas respectivamente.

bulbo y del disco, así como para este último su cociente de semiejes y ángulo de posición. El resto de parámetros observacionales, se asumirán como aquellos que el telescopio espacial Euclid tendrá. Por ejemplo, la profundidad que alcanza el telescopio es de 29.5 mag, que será el valor de brillo del *background* de las imágenes, y el ruido se modela como fluctuaciones de $3 * \sigma$ en cajas de 10x10 $arcsec^2$ sobre este *background*. Así mismo, la distorsión óptica que se asume en las imágenes (la llamada Función de Dispersión de Punto o PSF) viene dada por las simulaciones del citado telescopio.

Por lo tanto, este *script* permite generar cualquier número de imágenes de galaxias y colas de marea, en formato `.fits`, con variaciones aleatorias para obtener un conjunto de datos artificial del tamaño que se quiera. En la figura 4.20 se muestran por separado las partes generadas por el *script*. Como se generan por separado los distintos elementos, galaxia y cola de marea, este conjunto de datos que generamos nos sirve para entrenar tanto modelos de segmentación semántica, donde la variable independiente es la imagen de cola de marea y galaxia y la variable dependiente es la máscara de la cola de marea, como modelos de clasificación, donde la variable independiente es la imagen, que puede tener cola de marea o no, y la variable dependiente es la categoría que indica si tiene o no cola de marea.

Después, estas imágenes se "fusionan" con fragmentos de una imagen realista simulada como si fuera observada por Euclid que contiene una gran cantidad de objetos astronómicos cuyas dimensiones son 40350x40958 píxeles. Esta operación se realiza mediante funciones del paquete `MontagePy`. El motivo de hacer esto es situar la cola sobre un fondo realista, en vez de sobre un fondo uniforme o con ruido gaussiano sin más. Esta imagen corresponde a una imagen de simulación de la Agencia Espacial Europea para la misión del Euclid, que se almacena en el repositorio privado *Euclid Science Archive System*[1]. En la figura 4.21 se muestra un fragmento de esta imagen visualizada con el software SAO Image DS9. Para magnificar el contraste en esta visualización se ha usado un mapa de colores HSV y se ha aplicado una escala logarítmica con unos parámetros de escalado concretos, donde el mínimo es 0.0001 y el máximo es 5. De esta forma, en el caso de las estrellas, como la que se observa en la zona inferior izquierda de la figura 4.21, cuyo valor de brillo es mucho

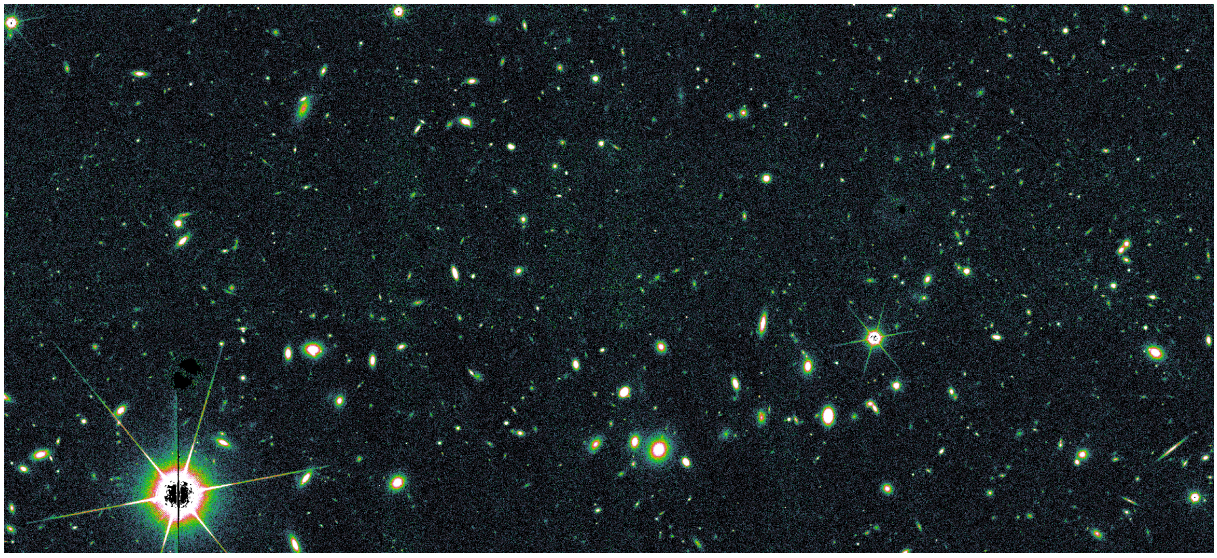


Figura 4.21: Fragmento de la imagen de simulación de la misión Euclid[1]. La imagen es en blanco y negro pero la coloreamos para que los objetos astronómicos aparezcan más claros.

mayor al del resto de objetos, su valor de brillo se satura, y el rango de valores queda más ajustado.

Por lo tanto, el *pipeline* para obtener la variable independiente en el problema de segmentación, es decir, el *input* del modelo, es el que se muestra en la figura 4.22. Para el problema de clasificación el *pipeline* es equivalente, salvo que a un porcentaje de las imágenes no se les añadirá cola de marea.

Llegados a este punto, se nos abre una cuestión esencial que debe ser resuelta, y es la etiqueta que se usará como máscara de segmentación. Tenemos dos opciones:

1. **Opción 1**: Utilizar como máscara de segmentación todos los píxeles donde el valor del brillo superficial de la cola de marea es mayor que 0 (figura 4.23a). Esta es la opción que se usó durante las actividades de la asignatura de I+d+i, es decir, es la que se planteó inicialmente y la que habíamos usado hasta ahora.

2. **Opción 2**: Utilizar como máscara de segmentación solamente los píxeles donde el valor del brillo superficial de la cola de marea es, al menos, 3 veces más grande que la profundidad que el Euclid puede alcanzar (figura 4.23b).

Finalmente se optó por la segunda opción, ya que se consideró que era más realista y, por tanto, más correcto teniendo en cuenta nuestro objetivo final, que es que los modelos entrenados sirvan, en la medida de lo posible, con imágenes reales. En la figura 4.23 se muestra la comparativa entre ambas opciones.

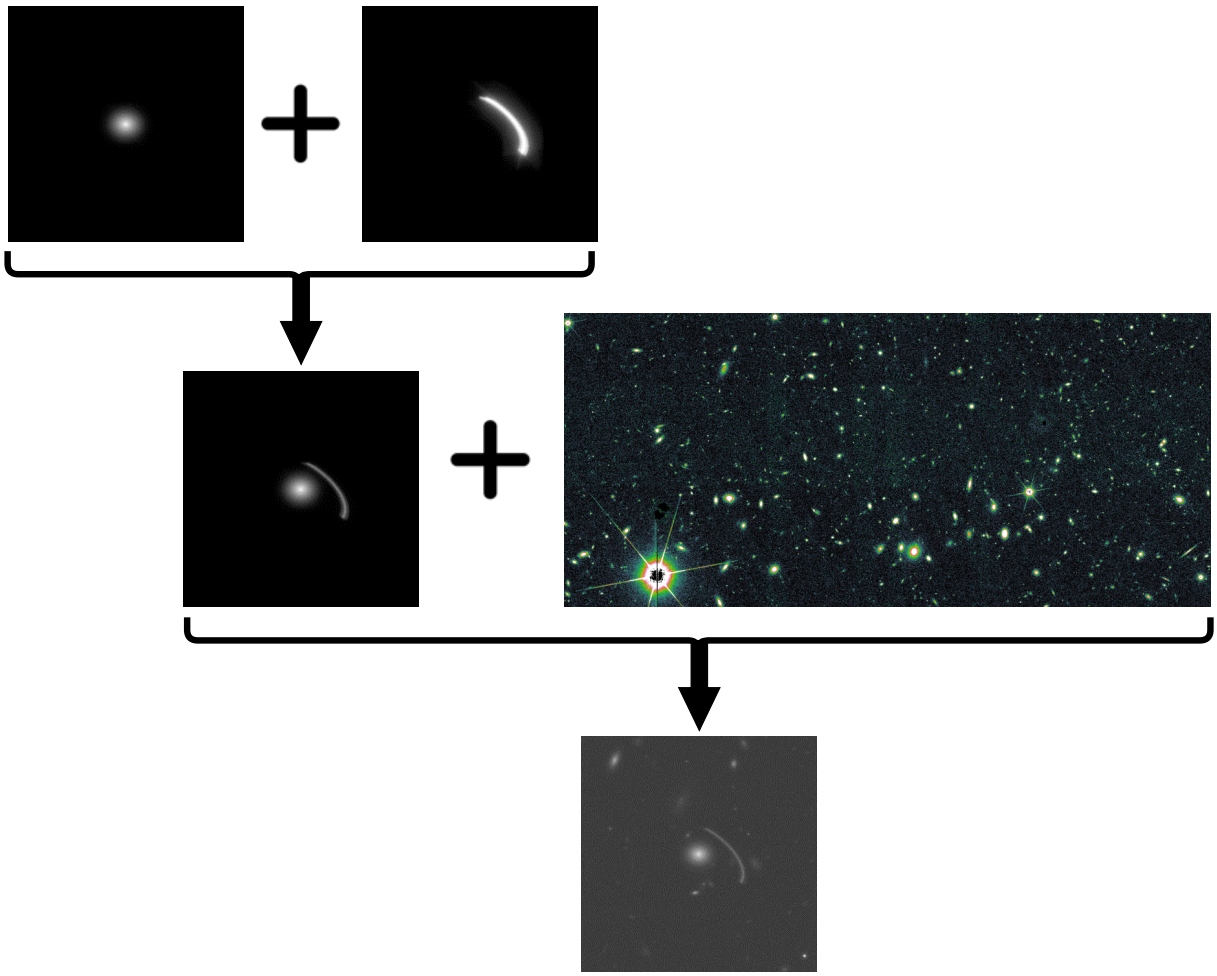
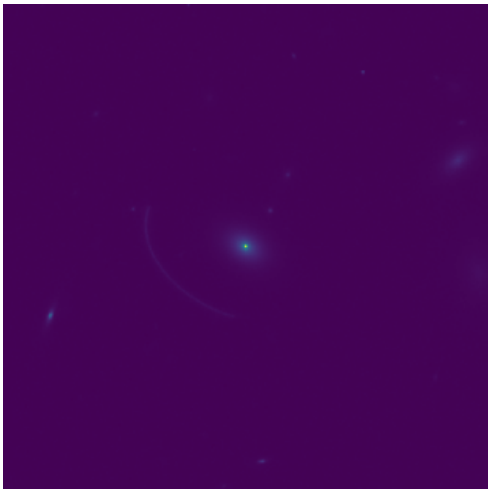


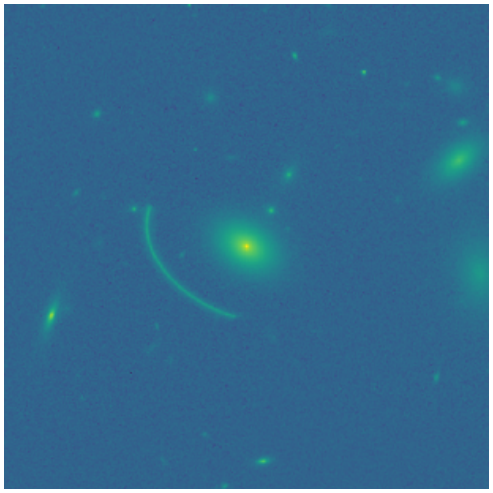
Figura 4.22: *Pipeline* para obtener el *input* del modelo.

(a) Opción 1 de máscara de segmentación (b) Opción 2 de máscara de segmentación

Figura 4.23: Las dos opciones planteadas como máscaras de segmentación.



(a) Imagen en escala lineal



(b) Imagen en escala logarítmica

Figura 4.24: Diferencia entre la imagen en escala lineal y la imagen en escala logarítmica

4.1.1. Preprocesado de los datos

Una vez elegidas las máscaras de segmentación se nos plantea otra cuestión a resolver. Los valores de brillo superficial del centro de una galaxia son varios órdenes de magnitud mayor que los valores de brillo superficial de una cola de marea, lo cual provoca que, en una misma imagen en escala lineal donde se muestran ambas a la vez, la cola de marea es indistinguible del fondo para el ojo humano. El ojo humano es un detector logarítmico[21], y no un detector lineal, porque la sensibilidad de los fotorreceptores en la retina del ojo humano varía con la cantidad de luz que incide en ellos. Por norma general, si al ojo humano le cuesta distinguir algo, también le costará a la red neuronal. Por ello, se ha decidido transformar cada imagen generada de escala lineal (figura 4.24a) a **escala logarítmica** (figura 4.24b), donde los valores de brillo superficial de la galaxia y la cola de marea se igualan. Esta es una decisión común dentro del mundo de la Astronomía.

4.1.2. Data augmentation

Para incrementar la variedad del conjunto de datos y **prevenir el sobreajuste**, se han preparado distintos tipos de *data augmentation* aleatorios que se utilizarán sobre cada imagen y máscara del conjunto de entrenamiento en cada época del bucle de entrenamiento. En concreto, se han usado las siguientes *augmentations*:

- Rotaciones aleatorias entre 180º y -180º.
- Traslaciones aleatorias en el eje X y/o en el eje Y de hasta un 10 % del tamaño total de la imagen. Se ha usado este valor concreto para que nunca se salgan partes de la galaxia o cola de marea fuera de la imagen, y siempre se mantengan en la imagen en su totalidad.

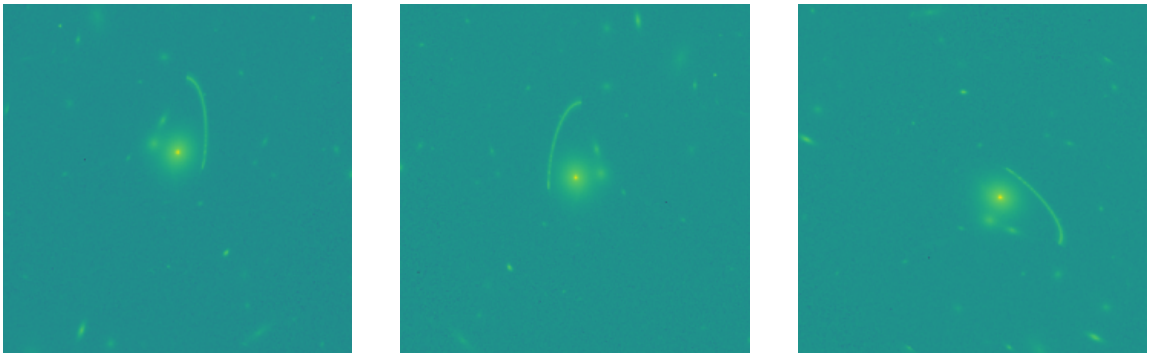


Figura 4.25: Distinto *data augmentation* aplicado a la misma galaxia.

- Inversiones en los ejes X y/o Y.

En la figura 4.25 se muestran estas operaciones de *data augmentation* aplicadas a una imagen de entrada de ejemplo del conjunto de datos generado.

4.2. Arquitectura del modelo de segmentación

El modelo utilizado es una red de segmentación semántica multiclase implementada desde cero tomando como guía las instrucciones proporcionadas en el repositorio https://github.com/hamdaan19/UNet-Multiclass, que a su vez, está basada en la U-Net, que ya explicamos en la sección 3.2.5, por lo tanto su estructura y funcionamiento se basa en el de esta. En la figura 4.26 se muestra el esquema de la arquitectura de esta red, donde se distinguen claramente las dos partes de una U-Net: el camino de contracción y el camino de expansión.

Esta arquitectura está construida a partir de bloques de doble convolución que se repiten con una estructura común formada por las siguientes capas:

1. `Conv2d(num_canales_entrada, num_canales_salida, kernel_size=(3,3), stride=(1,1), padding(1,1), bias=False`: La primera capa es una convolución 2D que, a partir de un tensor de entrada con el número de canales indicado en el primer parámetro, genera un tensor de salida con el número de canales indicado en el segundo parámetro. El tamaño de kernel usado para realizar la convolución es siempre de 3x3, el *stride* es de 1 y se añade un *padding* de 1 pixel. El parámetro `bias` lo ponemos a `False` porque cualquier sesgo de canal añadido solo afecta a la media del canal. Dado que hay una capa de `BatchNorm2d` que se aplica justo a continuación, y que eliminará la media del canal, no tiene sentido añadir sesgo a esta capa.

2. `BatchNorm2d(num_canales_salida)`: Normaliza los valores de activación de cada canal de la capa de entrada, con respecto a la media y la desviación estándar del *mini-batch* actual.

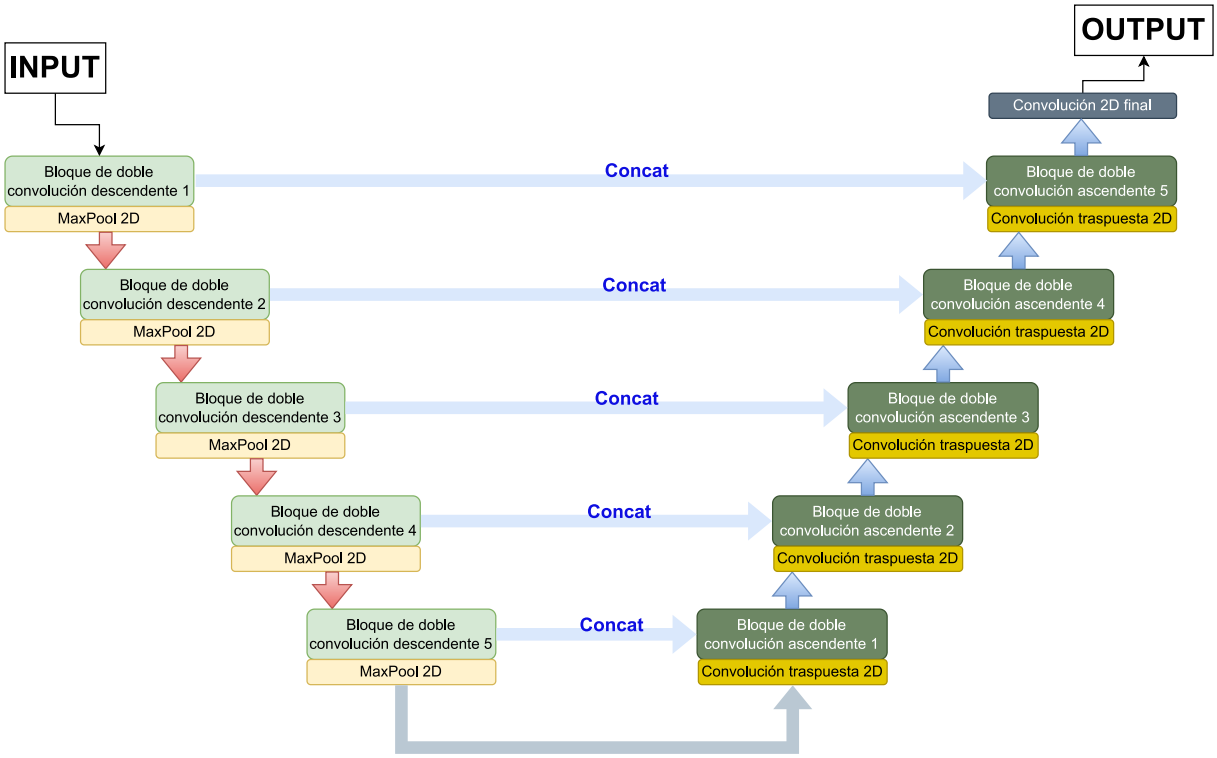


Figura 4.26: Arquitectura de la U-Net personalizada usada en este trabajo.

3. `ReLU(inplace=true)`: Aplica la función de activación ReLU (*Rectified Linear Unit* elemento a elemento.

4. `Conv2d(num_canales_entrada, num_canales_salida, kernel_size=(3,3), stride=(1,1), padding(1,1), bias=True`: Es equivalente a la primera capa del bloque, exceptuando que ahora ponemos el `bias` a `True` porque después no hay capa de `BatchNorm2d`.

5. `ReLU(inplace=true)`: Vuelve a aplicar ReLU.

Tras cada bloque de doble convolución descendente hay una capa de `Max Pooling`, que divide a la mitad el tamaño de los tensores en esa capa. Y antes de cada bloque de doble convolución ascendente hay una capa de convolución traspuesta, para ir devolviendo los tensores a sus dimensiones originales. Al final del último bloque, hay una última capa de convolución 2D para obtener el *output* de la red, que es la máscara de segmentación. Además, la salida de la última ReLU de cada bloque de doble convolución del *encoder* se concatena con la salida de cada convolución traspuesta del *decoder*, para crear las *skip-connections*.

Es una red bastante profunda ya que tiene un total de 54 capas y 31.033.602 parámetros y un tamaño total estimado en memoria de 2863 MB. Su estructura se detalla en la tabla 4.14. La primera columna indica el tipo de capa, la segunda columna indica la forma que

tiene el tensor de salida de esa capa y la tercera columna indica el número de parámetros entrenables que tiene esa capa.

Capa	**Forma del output**	**Número de parámetros**
1. Conv2d	[64, 512, 512]	576
2. BatchNorm2d	[64, 512, 512]	128
3. ReLU	[64, 512, 512]	0
4. Conv2d	[64, 512, 512]	36,928
5. ReLU	[64, 512, 512]	0
6. MaxPool2d	[64, 256, 256]	0
7. Conv2d	[128, 256, 256]	73,728
8. BatchNorm2d	[128, 256, 256]	256
9. ReLU	[128, 256, 256]	0
10. Conv2d	[128, 256, 256]	147,584
11. ReLU	[128, 256, 256]	0
12. MaxPool2d	[128, 128, 128]	0
13. Conv2d	[256, 128, 128]	294,912
14. BatchNorm2d	[256, 128, 128]	512
15. ReLU	[256, 128, 128]	0
16. Conv2d	[256, 128, 128]	590,080
17. ReLU	[256, 128, 128]	0
18. MaxPool2d	[256, 64, 64]	0
19. Conv2d	[512, 64, 64]	1,179,648
20. BatchNorm2d	[512, 64, 64]	1,024
21. ReLU	[512, 64, 64]	0
22. Conv2d	[512, 64, 64]	2,359,808
23. ReLU	[512, 64, 64]	0
24. MaxPool2d	[512, 32, 32]	0
25. Conv2d	[1024, 32, 32]	4,718,592
26. BatchNorm2d	[1024, 32, 32]	2,048
27. ReLU	[1024, 32, 32]	0
28. Conv2d	[1024, 32, 32]	9,438,208
29. ReLU	[1024, 32, 32]	0
30. ConvTranspose2d	[512, 64, 64]	2,097,664
31. Conv2d	[512, 64, 64]	4,718,592
32. BatchNorm2d	[512, 64, 64]	1,024
33. ReLU	[512, 64, 64]	0
34. Conv2d	[512, 64, 64]	2,359,808
35. ReLU	[512, 64, 64]	0
36. ConvTranspose2d	[256, 128, 128]	524,544
37. Conv2d	[256, 128, 128]	1,179,648
38. BatchNorm2d	[256, 128, 128]	512

39. ReLU	[256, 128, 128]	0
40. Conv2d	[256, 128, 128]	590,080
41. ReLU	[256, 128, 128]	0
42. ConvTranspose2d	[128, 256, 256]	131,200
43. Conv2d	[128, 256, 256]	294,912
44. BatchNorm2d	[128, 256, 256]	256
45. ReLU	[128, 256, 256]	0
46. Conv2d	[128, 256, 256]	147,584
47. ReLU	[128, 256, 256]	0
48. ConvTranspose2d	[64, 512, 512]	32,832
49. Conv2d	[64, 512, 512]	73,728
50. BatchNorm2d	[64, 512, 512]	128
51. ReLU	[64, 512, 512]	0
52. Conv2d	[64, 512, 512]	36,928
53. ReLU	[64, 512, 512]	0
54. Conv2d	[2, 512, 512]	130

Tabla 4.14: Arquitectura del modelo de red en U personalizado utilizado.

4.3. Funciones de pérdida

Una función de pérdida o *loss function* es una medida que se utiliza para cuantificar la discrepancia entre el valor predicho por un modelo y el valor real de los datos. El objetivo del aprendizaje es minimizar esta pérdida, ajustando los parámetros del modelo para que las predicciones se acerquen lo más posible a los valores reales. Hay muchos tipos de funciones de pérdida diferentes, y la elección de esta depende del tipo de problema que se esté intentando resolver. En esta sección se describen las funciones de pérdida usadas tanto en el problema de segmentación semántica como en el problema de clasificación binaria que le sigue.

4.3.1. Entropía cruzada

La función de pérdida que se ha utilizado para optimizar el modelo durante el entrenamiento, tanto para la segmentación semántica como para la clasificación binaria ha sido la **función de pérdida de entropía cruzada**, (en inglés, "*cross-entropy loss function*"). Esta función es una medida comúnmente utilizada para evaluar la diferencia entre dos distribuciones de probabilidad, en particular, la distribución de probabilidad que se calcula a partir de las predicciones de un modelo y la distribución de probabilidad real de los datos.

Si estamos intentando ajustar un modelo en un problema de n clases, el valor de esta función se calcula como

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i)$$

donde t_i es la etiqueta real y p_i es la probabilidad de la función Softmax para la clase i-ésima. Es decir, mide la discrepancia entre las predicciones de la red neuronal y las etiquetas verdaderas, penalizando las predicciones incorrectas. En un problema de clasificación binaria, esto es, donde solo hay 2 clases, 0 y 1, clase negativa y clase positiva, se usa la función de coste de entropía cruzada binaria (*binary cross-entropy loss*), que es un caso específico de la entropía cruzada general y se calcula como

$$L = -\frac{1}{N}\Big[\sum_{j=1}^{N}[t_j log(p_j) + (1 - t_j) log(1 - p_j)]\Big]$$

donde N es el número de datos, t_i es el valor real que toma valor 0 o 1 y p_i es la probabilidad de la función Softmax para el i-ésimo dato.

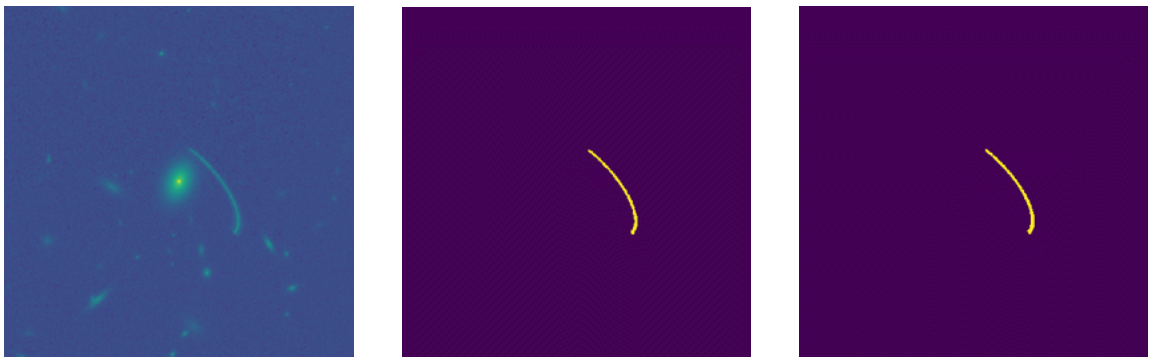
4.4. Métricas

Una métrica, en términos de *machine learning*, es una medida que sirve para evaluar o comunicar la calidad o efectividad del modelo una vez que ya ha sido entrenado, a diferencia de las funciones de pérdida, que se utilizan para guiar el proceso de entrenamiento. Hay muchos tipos diferentes, y su elección, al igual que con la función de pérdida, depende del tipo de problema a resolver. En este trabajo se han realizado dos tipos de tareas sobre los conjuntos de datos: segmentación semántica y clasificación, por lo tanto, en cada uno se han utilizado métricas diferentes para evaluar y comparar la calidad de los modelos entrenados. En segmentación se ha utilizado el coeficiente de *Dice*, mientras que en clasificación se han empleado un conjunto de métricas: *accuracy*, *precision*, *recall*, *F1-score* y área bajo la curva ROC.

4.4.1. Segmentación: *Dice*

La métrica que se ha utilizado para medir la calidad de los modelos de segmentación entrenados ha sido el **coeficiente de *Sørensen-Dice***, llamado comúnmente *Dice*. También existen otras métricas bastante usadas en problemas de segmentación semántica que no vamos a utilizar aquí como el coeficiente de Jaccard(también llamado *Intersection Over Union* (IOU)) o la exactitud de píxeles(*pixel accuracy*). El coeficiente de *Dice* es una medida de similitud utilizada en diversas áreas, como la biología, la ecología y la informática que fue propuesta inicialmente por Sørensen en 1948 para analizar la diversidad de especies en la vegetación de Dinamarca[35]. En nuestro contexto, esta métrica se calcula de la siguiente manera:

$$Dice = \frac{2 * TP}{2 * TP + FP + FN}$$



(a) *Input* (b) *Label* (c) *Output*

Figura 4.27: Inferencia para una muestra del conjunto de datos.

donde TP son los *true positives* o verdaderos positivos, FP son los *false positives* o falsos positivos y FN son los *false negatives* o falsos negativos. Estos conceptos se aplican estableciendo una clase positiva y una clase negativa:

- Un verdadero positivo es el resultado cuando el modelo predice correctamente la clase positiva.
- Un falso positivo es el resultado cuando el modelo predice incorrectamente la clase positiva, es decir, ha predicho como clase positiva algo que en realidad era clase negativa.
- Un falso negativo es el resultado cuando el modelo predice incorrectamente la clase negativa, es decir, ha predicho como clase negativa algo que en realidad era clase positiva.
- Un verdadero negativo es el resultado cuando el modelo predice correctamente la clase negativa.

Por lo tanto, el *Dice* mide la proporción de píxeles acertados de la red neuronal para una clase dada. En nuestro caso vamos a ignorar la clase de fondo (*background class*), ya que no se le suele dar importancia en los problemas de segmentación semántica.

En la figura 4.27 se muestra una muestra cualquiera del conjunto de datos. En esta figura, la primera columna (figura 4.27a) es la variable independiente o *input* que se le da al modelo, la segunda columna (figura 4.27b) es la etiqueta (*label)* o máscara segmentación, es decir, lo que el modelo debería devolver a partir del *input*. La tercera columna (figura 4.27c) es la variable dependiente u *output* inferido por uno de los modelos entrenados (de los que se hablará en el capítulo 4.4.2) para ese *input* dado. Al aplicar la fórmula del *Dice* mencionada anteriormente, obtenemos un resultado de 0.94, que es un valor bastante bueno en este tipo de problemas.

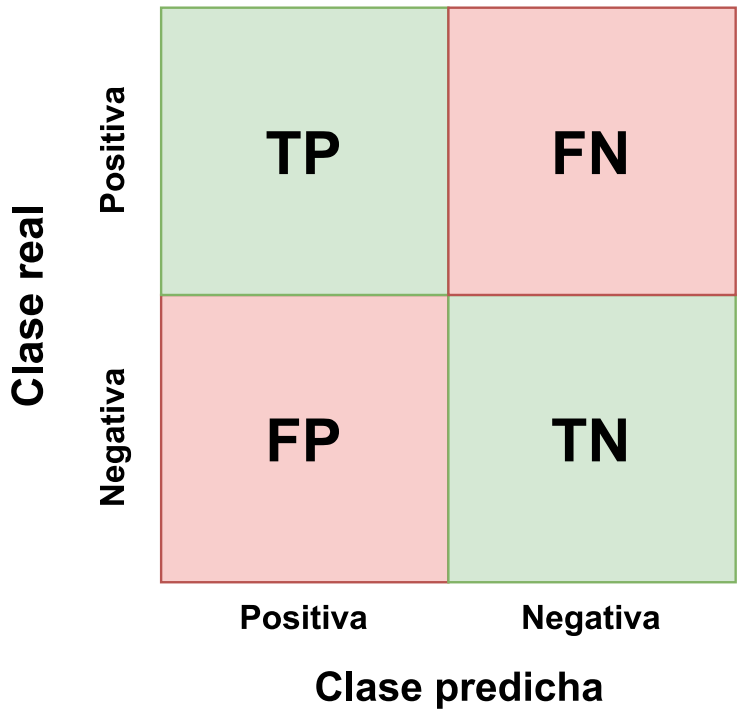


Figura 4.28: Matriz de confusión de ejemplo.

4.4.2. Clasificación

En un problema de clasificación con 2 clases, como es el que se está tratando en este trabajo, también se tienen en cuenta los conceptos de verdadero (TP) y falso positivo(FP), y verdadero(TN) y falso negativo(TN), que se han explicado en el apartado anterior, con el mismo significado, solo que en este caso, se refiere a la categoría de la imagen entera, en vez de a la categoría de cada píxel.

Para analizar la bondad de un clasificador de forma rápida se suele utilizar un método de visualización llamado matriz de confusión. Consiste en una matriz cuadrada con tantas filas y columnas como clases a predecir tengamos en nuestro problema. Las columnas representan las clases reales del conjunto de datos y las filas representan las clases predichas por el modelo. Por lo tanto, en las casillas de la diagonal principal se representa el número de predicciones correctas, mientras que en el resto de casillas se representa el número de predicciones incorrectas. En la figura 4.28 se muestra el formato de una matriz de confusión para un problema de clasificación binaria, como es el que estamos tratando en este proyecto.

Además, a partir de estos datos se van construyendo una serie de métricas relacionadas y complementarias entre sí que permiten analizar en profundidad la calidad del clasificador.

Accuracy

La métrica de exactitud o *accuracy* es un valor entre 0 y 1 que indica cómo de bien detecta la clase positiva y negativa a la vez. Se calcula mediante la siguiente fórmula:

$$Accuracy = \frac{TP + TN}{Total\ muestras}$$

donde TP es el número de verdaderos positivos y TN es el número de verdaderos negativos. Es decir, se calcula como la suma total del número de etiquetas predichas correctamente por la red dividido entre el total de muestras. Un clasificador cuyo *accuracy* está por debajo de 0.5 es peor que un clasificador aleatorio.

Precision

La métrica de precisión o *precision* es un valor entre 0 y 1 que indica la precisión del modelo al detectar la clase positiva. Se calcula mediante la siguiente fórmula:

$$Precision = \frac{TP}{TP + FP}$$

donde TP es el número de verdaderos positivos y FP es el número de falsos positivos. Es decir, se calcula, de todas las predicciones de la clase positiva, cuantas ha acertado de verdad.

Recall

La métrica de recuperación, sensibilidad o *recall* es un valor entre 0 y 1 que indica la capacidad del modelo para detectar la clase positiva. Se calcula mediante la siguiente fórmula:

$$Recall = \frac{TP}{TP + FN}$$

donde TP es el número de verdaderos positivos, y FN es el número de falsos negativos. Es decir, calcula, de todas las clases positivas del conjunto de datos, cuántas se han predicho correctamente.

F1-score

La métrica *F1-score* combina las métricas de *precision* y *recall* en un solo valor, proporcionando una evaluación equilibrada del modelo que permite comparar varios entre sí de forma rápida. Se calcula mediante la siguiente fórmula:

$$F1 = \frac{2 * Recall * Precision}{Recall * Precision}$$

Esta medida es útil en situaciones donde se busca un equilibrio entre *precision* y *recall*, ya que un valor alto de *F1* indica que ninguna de las dos métricas de las que depende es extremadamente baja. En realidad, el *F1-score* y el coeficiente de Dice son métricas equivalentes, ya que ambas son una media armónica de la precisión(*precision*) y la recuperación(*recall*).

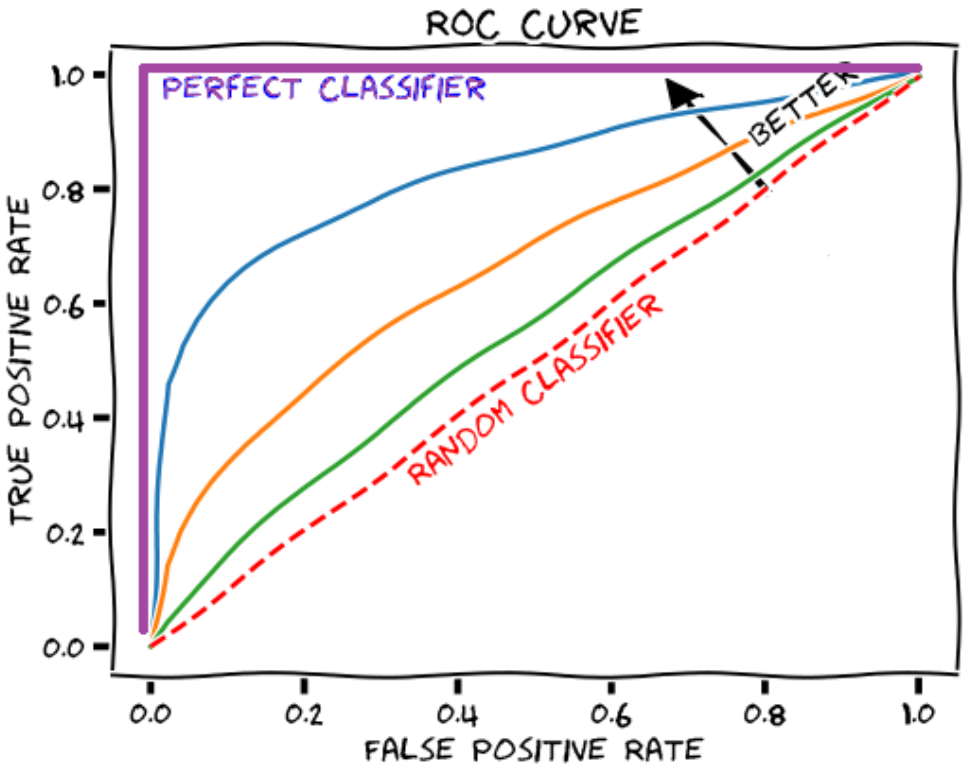


Figura 4.29: Explicación de una curva ROC[9].

Área bajo la curva ROC

En los apartados anteriores hemos supuesto que el umbral que distingue una predicción de la clase positiva de una predicción de la clase negativa es equidistante para ambas, es decir, está establecido en 0.5, ya que un clasificador da valores de predicción continuos en el intervalo $[0-1]$. Los valores de las predicciones no se pueden controlar de forma directa, ya que dependen de muchos factores, como el conjunto de datos, el tipo de modelo, o el algoritmo de aprendizaje. Sin embargo, lo que sí se puede controlar de forma explícita es el valor umbral a partir del cual las instancias se categorizan como clase positiva o negativa.

Una curva *Receiver Operating Characteristic* (ROC) es una representación gráfica que muestra la relación entre la tasa de verdaderos positivos (sensibilidad o *recall*) y la tasa de falsos positivos. Esta tasa de falsos positivos es una métrica más que se calcula mediante la siguiente fórmula:

$$FP\ rate = \frac{FP}{FP + TN}$$

Al ajustar el umbral de clasificación se obtienen diferentes pares de valores de tasa de verdaderos positivos y de falsos positivos. En el eje Y de la curva se representa la tasa de verdaderos positivos, y en el eje X la tasa de falsos positivos, como se muestra en la figura 4.29.

A partir de esta curva, se puede calcular el área bajo la curva (*Area under the ROC Curve* (AUC)), una medida entre 0 y 1 que indica el rendimiento general del modelo, así como su capacidad de discriminación entre la clase positiva y negativa. Un AUC con un valor igual a 1 representa un clasificador perfecto, y con un valor igual a 0.5, un clasificador aleatorio. Si está por debajo de este valor, entonces el modelo entrenado es peor que un clasificador aleatorio.

5: Discusión y comparación de resultados

En este capítulo se van a presentar los resultados obtenidos tras la experimentación realizada, tanto de segmentación como de clasificación. La presentación de estos resultados se va a dividir según la versión del *dataset* empleado, ya que este no era definitivo desde el inicio, sino que se ha ido refinando y mejorando durante el transcurso del proyecto. En concreto, se han realizado 3 versiones del *dataset*, cada una con imágenes más complejas que la versión anterior, y por lo tanto, más desafiantes para los modelos de aprendizaje profundo que hemos usado.

Para la generación de cada conjunto de datos se ha seguido la metodología explicada en el capítulo 3.2.6, y solo se diferencian en los parámetros que se han aleatorizado en el `script` a la hora de generarlos. Todas las imágenes y máscaras generadas tienen unas dimensiones de 512x512 píxeles y un solo canal de color.

5.1. Conjunto de datos 1

En este primer conjunto de datos algunos parámetros se han aleatorizado pero otros se han mantenido constantes, con el fin de empezar por imágenes con menor variabilidad. En cuanto a la generación de galaxias se han aleatorizado los siguientes parámetros:

- `re_bulge`: Radio efectivo del bulbo de la galaxia. Es un valor entre 1 y 9 píxeles.
- `re_disk`: Radio efectivo del disco de la galaxia. Es un valor entre 10 y 24 píxeles.
- `pa_disk`: Posición angular en grados del disco de la galaxia. Es un valor entre 0 y 359 grados.
- `ar_disk`: Cociente entre el semieje menor y mayor del disco, para darle aspecto elíptico. Es un valor entre 0.11 y 1.0.

Y en cuanto a la generación de las colas de marea se han aleatorizado los siguientes parámetros:

- `pa_stream`: Posición angular en grados de la cola de marea. Es un valor entre 0 y 359 grados.
- `ar_stream`: Cociente entre los ejes de la cola de marea. Es un valor entre 0.001 y 1.0.

Sin embargo, todos los valores relativos al brillo aparente se han mantenido constantes, en concreto:

- `bckg_mag`: Magnitud de brillo del fondo. Tiene un valor fijo de 29.5 mag.
- `mag_bulge`: Magnitud de brillo del bulbo de la galaxia. Tiene un valor fijo de 20 mag.
- `mag_disk`: Magnitud de brillo del disco de la galaxia. Tiene un valor fijo de 18 mag.
- `mag_stream`: Magnitud de brillo de la cola de marea. En esta sección, variaremos este valor. Primero entrenaremos con un *dataset* donde el valor de esta variable es 20 mag y, a continuación, crearemos otro *dataset* con un valor igual a 22 mag.

Con estos parámetros se ha generado un total de 1000 imágenes junto con sus máscaras para el conjunto de entrenamiento y 200 imágenes junto con sus máscaras para el conjunto de test. El conjunto de entrenamiento se divide a su vez en 2 partes en una proporción 80/20, donde el 80 % de las imágenes son para entrenamiento y el 20 % para validación. Todas estas imágenes se han generado estableciendo inicialmente el parámetro `mag_stream=20`.

5.1.1. Resultados de segmentación

A continuación, se ha entrenado el modelo descrito en el apartado 4.2 con los siguientes hiperparámetros:

- Épocas: 1000
- *Batch size*: 1
- *Learning rate*: 0.001

Después del entrenamiento se ha evaluado sobre el conjunto de test, obteniéndose un *Dice* medio de 0.928. En la figura 5.30 se muestran algunas inferencias sobre el conjunto de test. La primera columna es la variable independiente, es decir, la entrada de la red, la segunda columna es la máscara real y la tercera columna es la máscara predicha por el modelo entrenado, junto con el valor de Dice obtenido en la esquina inferior derecha.

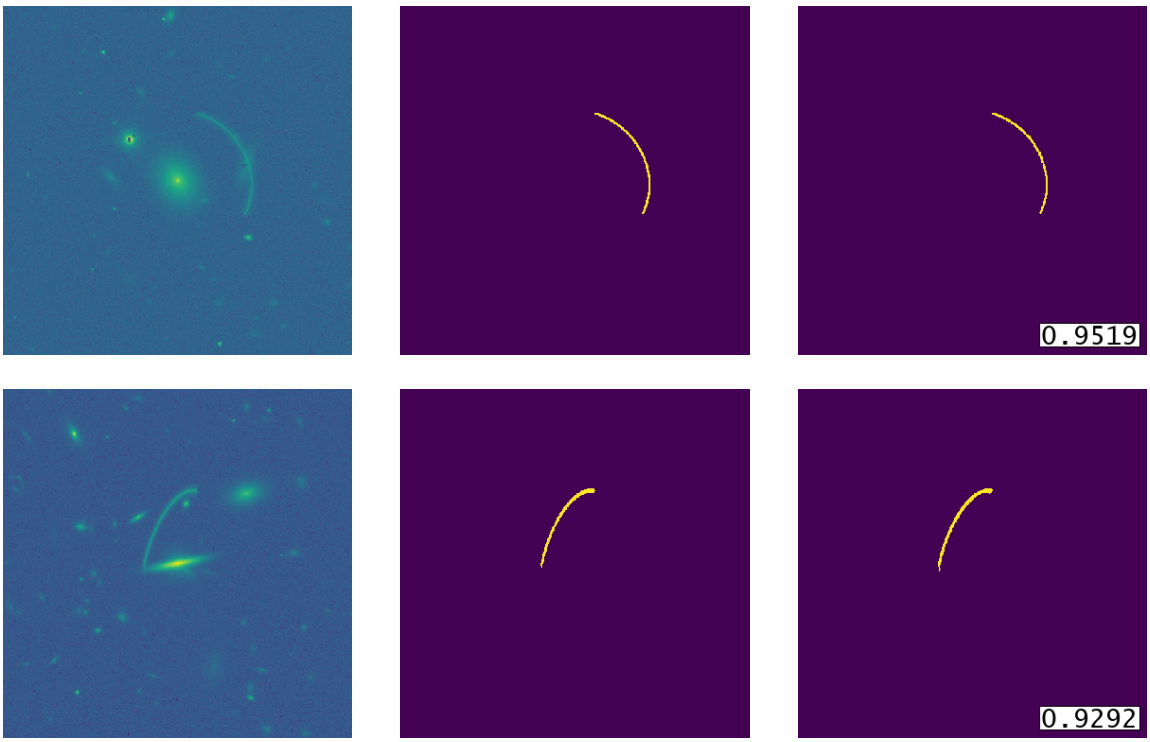


Figura 5.30: Inferencias con `mag_stream=20`

Después se volvió a realizar este mismo experimento con todos los parámetros iguales excepto uno: se generó un nuevo *dataset* con `mag_stream=22`, con el propósito de reducir los valores de brillo superficial de las colas de marea y hacerlas más realistas. Al evaluarlo sobre el conjunto de test, se obtuvo un *Dice* medio de 0.1413. Esta caída en el desempeño tiene una explicación, y es que las máscaras generadas con tan poco brillo a veces no llegaban al umbral mínimo para considerarlas como tal, por lo que en muchas de ellas, la máscara era o bien vacía, o bien se componía de muy pocos píxeles, lo que había provocado que la red no aprendiese bien. En la figura 5.31 se muestran algunos ejemplos de inferencias realizadas donde se puede observar la problemática existente.

5.1.2. Resultados de clasificación

Después de obtener resultados insatisfactorios debido a la controversia en la elección de un umbral para determinar si un píxel corresponde o no a una cola de marea, se hace evidente la necesidad de implementar un proceso en etapas que incluya una fase inicial de detección de colas de marea a través de una clasificación binaria, y una segunda fase de segmentación únicamente sobre las imágenes en las que se había identificado cola de marea. Por lo tanto, se utilizó el *dataset* generado anteriormente con `mag_stream=20`, y se dividió en 2 clases:

- **Imágenes de galaxias sin cola de marea**: Durante la generación del *dataset* artificial se generan ficheros `.fits` de todas las etapas del *pipeline* mostrado ante-

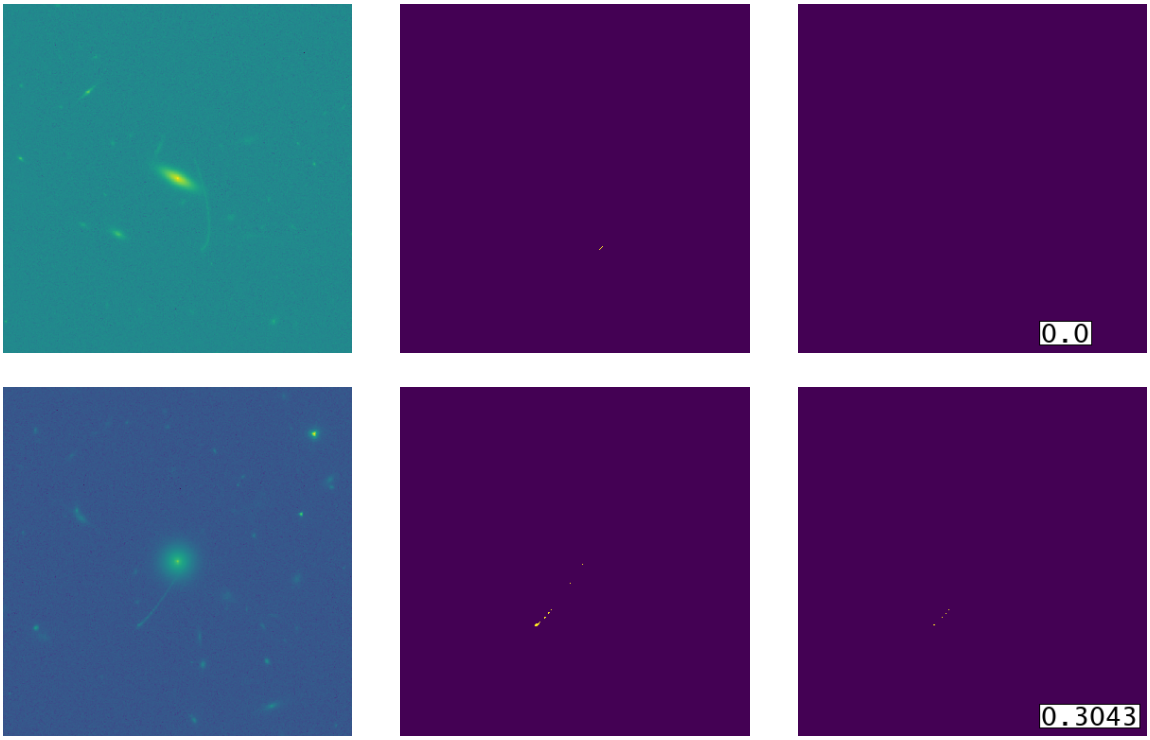


Figura 5.31: Inferencias con `mag_stream=22`

riormente en la figura 4.22, por lo que disponemos de las imágenes donde solo hay galaxia y no está todavía combinada con la cola de marea.

- **Imágenes de galaxias con cola de marea**: Utilizando las mismas que para la tarea de segmentación.

Las clases están divididas uniformemente 50/50 en el conjunto de entrenamiento. Sin embargo, en el conjunto de test, se ha dividido un 90 % sin cola de marea y un 10 % con cola de marea, ya que esta es la distribución real aproximada que nos encontraremos en las fotografías reales tomadas por el Euclid. A continuación, se ha entrenado una Resnet18 con los siguientes hiperparámetros:

- Épocas: 1000
- *Batch size*: 4
- *Learning rate*: 0.001

Después del entrenamiento se ha evaluado sobre el conjunto de test, obteniéndose los siguientes resultados:

- *Accuracy*: 0.99

- *Recall*: 1.0
- *Precision*: 0.929
- *F1-score*: 0.963
- Área bajo la curva ROC: 1.0

En la figura 5.32a se muestra la matriz de confusión obtenida con el conjunto de datos tienen `mag_stream=20`, junto con la curva ROC en la figura 5.32b.

También se realizó este proceso para el *dataset* generado con `mag_stream=22`, obteniéndose los siguientes resultados sobre el conjunto de test:

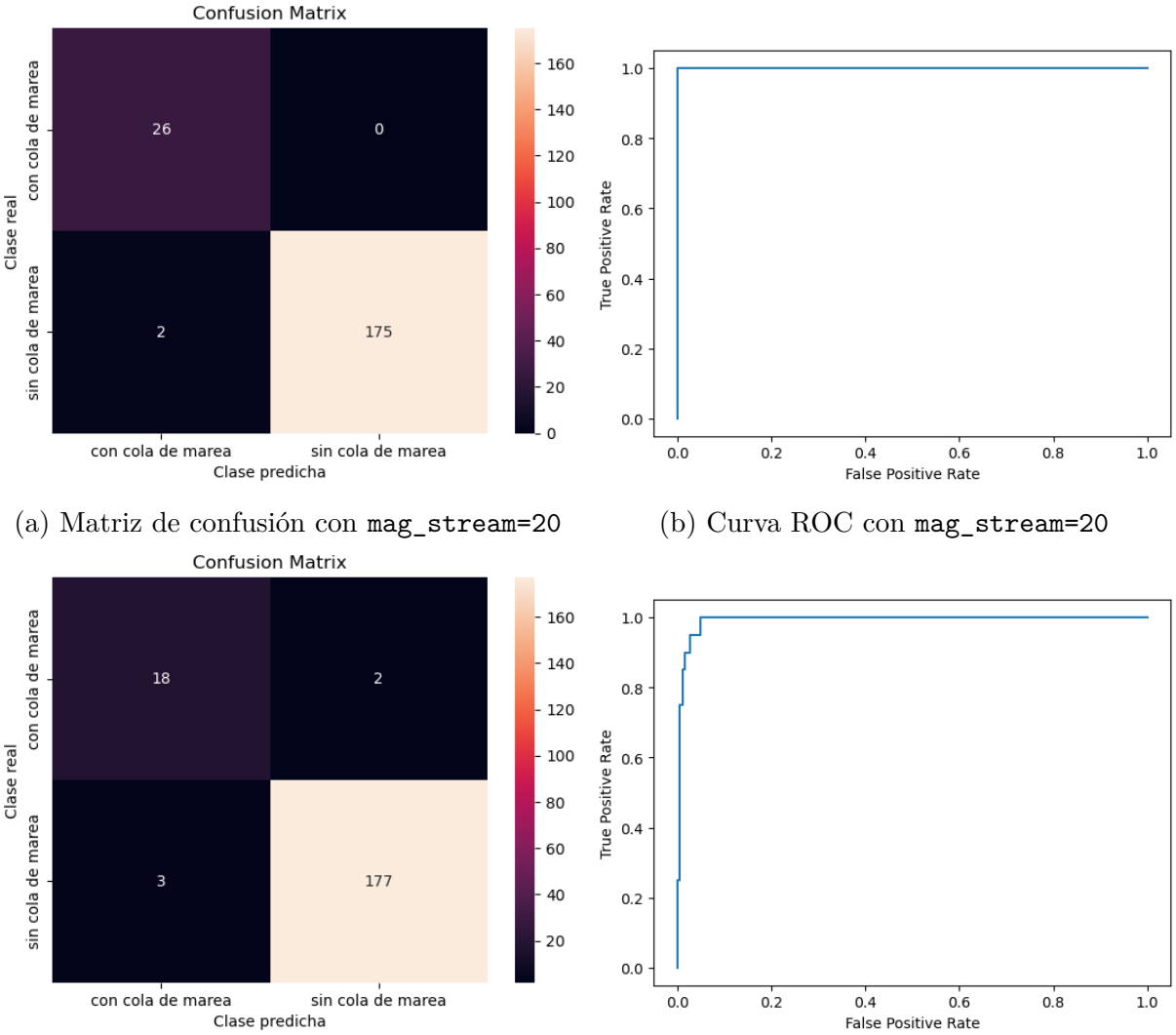
- *Accuracy*: 0.975
- *Recall*: 0.9
- *Precision*: 0.857
- *F1-score*: 0.878
- Área bajo la curva ROC: 0.991

En la figura 5.32c se muestra la matriz de confusión obtenida, junto con la curva ROC en la figura 5.32d. Como podemos observar, con ambos conjuntos de datos los resultados son bastante buenos, ya que la diagonal principal, que es donde la clase real coincide con la clase predicha, contiene la mayoría de las predicciones. Aun así, el conjunto de datos con `mag_stream=20` tiene resultados ligeramente mejores que el de `mag_stream=22`, ya que, en este último las colas de marea tienen menos brillo, por lo que son más difíciles de detectar para la red.

5.2. Conjunto de datos 2

Tras comprobar que los resultados no eran buenos con la primera versión del conjunto de datos cuando reducíamos el brillo de las colas de marea generadas(*incrementando* el parámetro *mag_stream*), se consideró que había que realizar ciertos cambios en el *script* de generación de imágenes para que el brillo superficial se redujera pero las máscaras obtenidas fueran correctas. Por ello, el profesor y cotutor de este trabajo Fernando Buitrago sugirió realizar algunas modificaciones al *script* de generación del conjunto de datos. Ahora se aleatorizan nuevos parámetros a mayores de los que ya se aleatorizaban con el *dataset 1*:

- `mag_bulge`: Magnitud de brillo del bulbo de la galaxia. Es un valor entre 19 y 21 mag.



(a) Matriz de confusión con `mag_stream=20`

(b) Curva ROC con `mag_stream=20`

(c) Matriz de confusión con `mag_stream=22`

(d) Curva ROC con `mag_stream=22`

Figura 5.32: Matrices de confusión y curvas ROC del *dataset* 1

- `mag_disk`: Magnitud de brillo del disco de la galaxia. Es un valor entre 17 y 19 mag.
- `potential_ini_angle`: Ángulo de inicio en grados de la cola de marea. Es un valor entre 0 y 359 grados.
- `potential_end_angle`: Ángulo de fin en grados de la cola de marea. Es un valor entre 0 y 359 grados pero siempre mayor que el ángulo de inicio.

Gracias a estos últimos dos parámetros, las colas de marea pueden tener cualquier número de grados, por lo que su amplitud será variable, lo que añadirá dificultad al aprendizaje de las redes. De momento, el `mag_stream` lo mantenemos fijo en 20 mag, pero ahora las colas de marea se modulan por una función $sen(2x)$ para hacerlas más realistas,

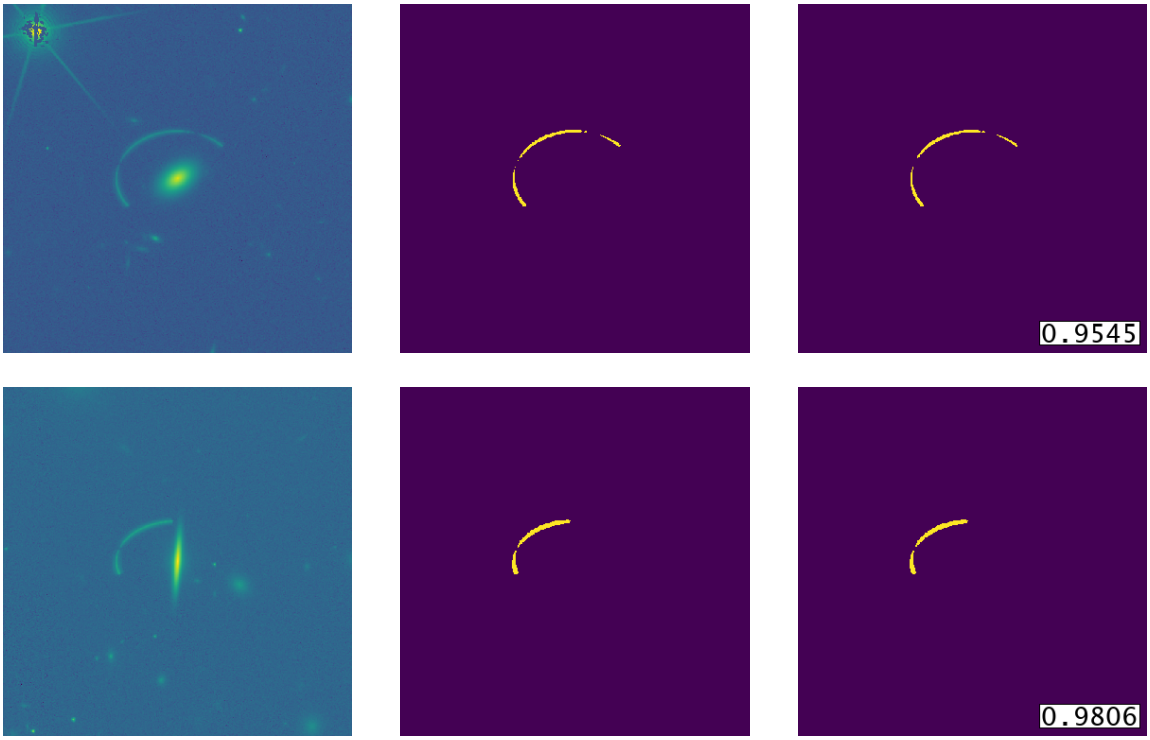


Figura 5.33: Inferencias con el *dataset* 2

de forma que hay zonas de la cola de marea muy poco visibles y otras bastante más visibles, en vez de estar distribuido uniformemente.

5.2.1. Resultados segmentación

Se ha generado un nuevo conjunto de datos de 2000 imágenes de entrenamiento con una división similar a la realizada en el apartado 5.1.1 y se ha entrenado también con los mismos hiperparámetros. El *Dice* medio obtenido sobre el conjunto de test es de 0.8924. En la figura 5.33 se muestran las inferencias obtenidas para algunas muestras del conjunto de test.

5.2.2. Resultados clasificación

Con este conjunto de datos también se ha realizado clasificación entre imagen con cola de marea y sin cola de marea, dividiendo el conjunto de entrenamiento en 50/50 y el conjunto de test en 90/10, con los mismos hiperparámetros y de la misma forma que en el apartado 5.1.2.

Se han entrenado 3 modelos diferentes de CNN, una ResNet18, una ResNet34 y una ResNet50. Los tres modelos han sido entrenados desde cero, es decir, no se ha hecho uso en ningún caso de *transfer learning*, sino que se han inicializado los pesos del modelo aleatoriamente. No obstante, esta inicialización aleatoria se ha realizado siguiendo las

instrucciones de la inicialización de Kaiming He[14], que es la manera recomendada de inicializar los pesos de redes neuronales muy profundas, como las ResNets que estamos usando en este caso.

En la figura 5.34 se muestran las matrices de confusión y las curvas ROC obtenidas por estos 3 modelos sobre el conjunto de test. Para la ResNet18 se muestran la matriz de confusión y la curva ROC resultantes en las figuras 5.34a y 5.34b, respectivamente y los resultados de las métricas analizadas son las siguientes:

- *Accuracy*: 0.99
- *Recall*: 1.0
- *Precision*: 0.909
- *F1-score*: 0.952
- Área bajo la curva ROC: 0.999

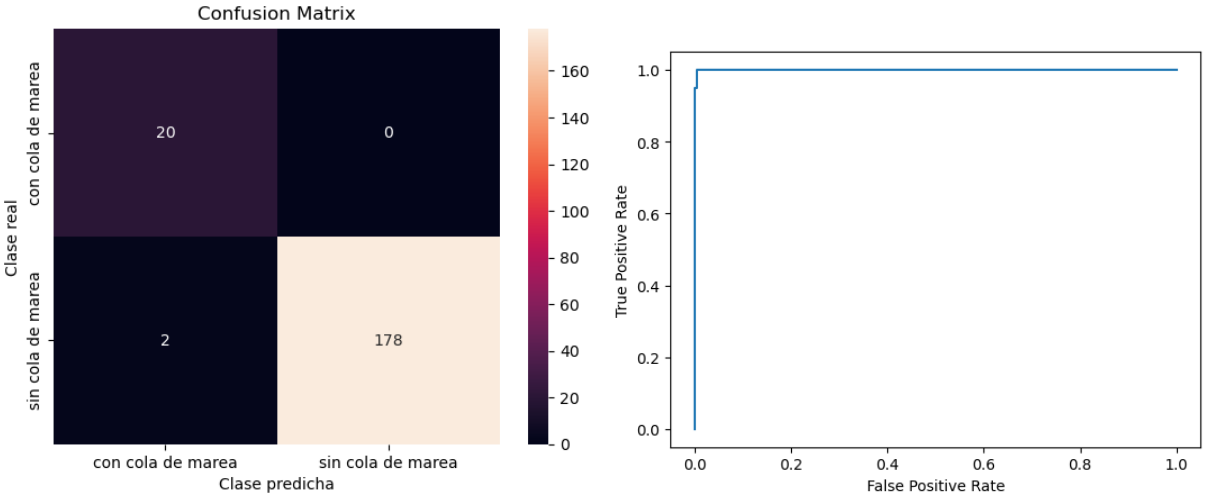
Para la ResNet34 se muestran la matriz de confusión y la curva ROC resultantes en las figuras 5.34c y 5.34d, respectivamente y los resultados de las métricas analizadas son las siguientes:

- *Accuracy*: 0.995
- *Recall*: 0.95
- *Precision*: 1.0
- *F1-score*: 0.974
- Área bajo la curva ROC: 1.0

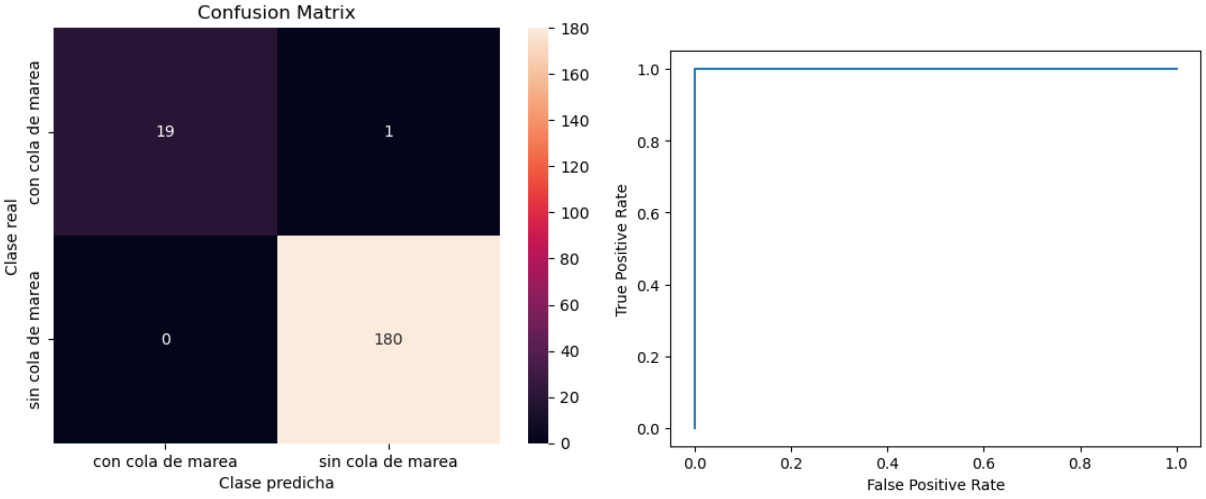
Por último, para la ResNet50 se muestran la matriz de confusión y la curva ROC resultantes en las figuras 5.34e y 5.34f, respectivamente y los resultados de las métricas analizadas son las siguientes:

- *Accuracy*: 0.99
- *Recall*: 0.95
- *Precision*: 0.95
- *F1-score*: 0.95
- Área bajo la curva ROC: 0.998

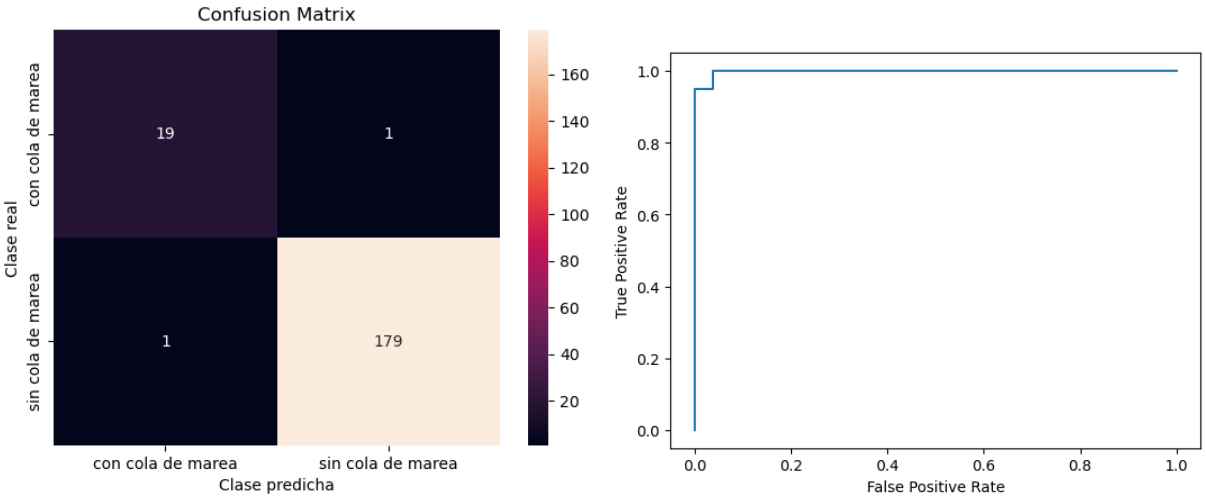
A la vista de estos resultados, el mejor modelo de los tres entrenados es la ResNet34, ya que es la que ha alcanzado un mayor valor en las métricas *F1* y área bajo la curva, aunque las diferencias no son muy abultadas entre los diferentes modelos.



(a) Matriz de confusión de ResNet18 (b) Curva ROC de ResNet18



(c) Matriz de confusión de ResNet34 (d) Curva ROC de ResNet34



(e) Matriz de confusión de ResNet50 (f) Curva ROC de ResNet50

Figura 5.34: Matrices de confusión y curvas ROC de ResNet18, ResNet34 y ResNet50 sin preentrenar con el *dataset* 2

5.3. Conjunto de datos 3

Para intentar hacer las simulaciones todavía más realistas y con más variabilidad, se han aleatorizado una serie de parámetros más en el *script* de generación de datos, todos relacionados con las colas de marea. En concreto, los nuevos parámetros aleatorizados son:

- `mag_stream`: Magnitud de brillo de la cola de marea. Es un valor que varía entre 19.8 y 20.2 mag, y que más tarde, se modula por la función seno, como en el *dataset 2*.
- `rr_stream`: Posición radial inicial de la cola de marea. Es un valor entre 80 y 120 grados.
- `width_stream`: Anchura de la cola de marea. Es un valor entre 15 y 25 píxeles.

5.3.1. Resultados segmentación

Se ha generado de nuevo un conjunto de datos de 2000 imágenes de entrenamiento con una división similar a la realizada en el apartado 5.2.1 y se ha entrenado también con los mismos hiperparámetros. En esta nueva versión del *dataset*, al haber randomizado el brillo de las colas de marea, algunas colas de marea no llegaban al umbral mínimo definido como para considerarla como *label* y por lo tanto, algunas máscaras tienen todos los píxeles a 0, que es un problema similar al que ocurría con el *dataset 1* cuando cambiamos el brillo de `mag_stream=20` a `mag_stream=22`. Sin embargo, en este caso, aquellas máscaras que tengan este problema no se van a considerar en el conjunto de entrenamiento y validación, ni a la hora de evaluar el *Dice* en el conjunto de test.

Tras entrenar nuestro modelo de segmentación, el *Dice* medio obtenido sobre el conjunto de test es de 0.799. En la figura 5.35 se muestran las inferencias obtenidas para algunas muestras del conjunto de test.

5.3.2. Resultados clasificación

Con este conjunto de datos también se ha realizado clasificación entre imagen con cola de marea y sin cola de marea, dividiendo el conjunto de entrenamiento en una proporción 50/50 y el conjunto de test en una proporción 90/10, con los mismos hiperparámetros y de la misma forma que en los apartados 5.1.2 y 5.2.2. Aquí también se han entrenado 3 modelos diferentes de CNN, una ResNet18, una ResNet34 y una ResNet50, desde cero, sin ningún tipo de preentrenamiento.

En la figura 5.36 se muestran las matrices de confusión y las curvas ROC obtenidas por estos 3 modelos sobre el conjunto de test. Para la ResNet18 se muestran la matriz de confusión y la curva ROC resultantes en las figuras 5.36a y 5.36b, respectivamente y los resultados de las métricas analizadas son las siguientes:

- *Accuracy*: 0.895

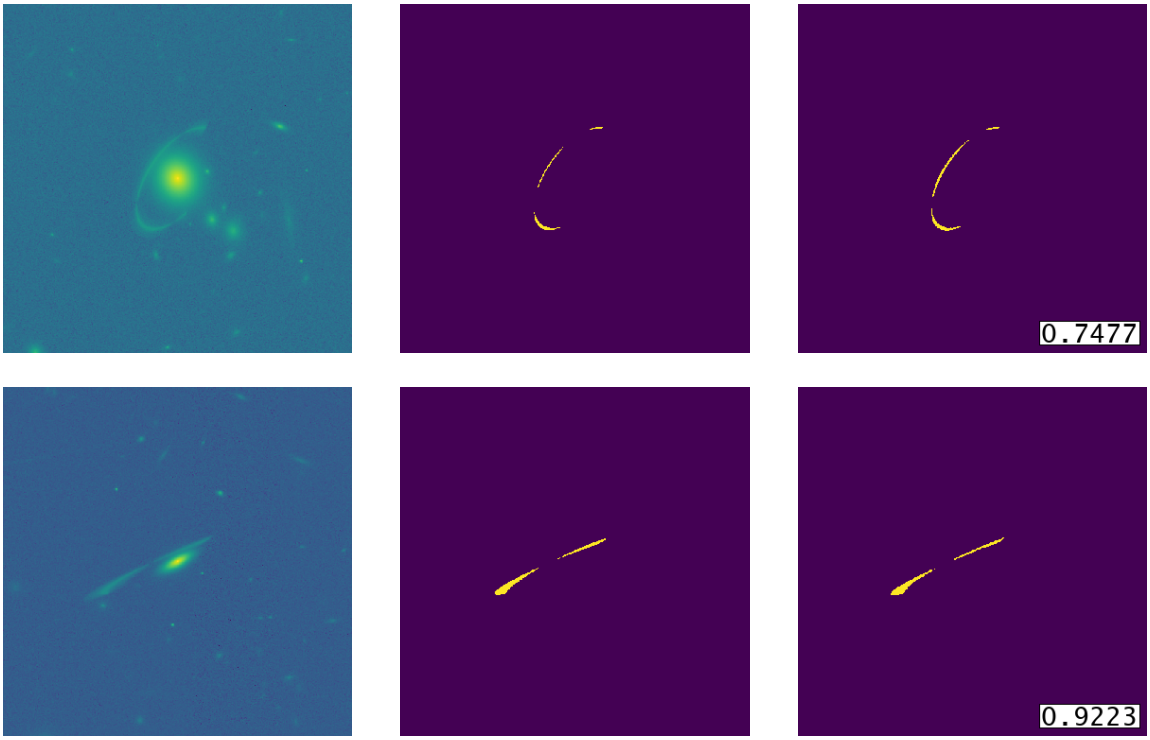


Figura 5.35: Inferencias con el *dataset* 3

- *Recall*: 0.85
- *Precision*: 0.485
- *F1-score*: 0.618
- Área bajo la curva ROC: 0.9225

Para la ResNet34 se muestran la matriz de confusión y la curva ROC resultantes en las figuras 5.36c y 5.36d, respectivamente y los resultados de las métricas analizadas son las siguientes:

- *Accuracy*: 0.93
- *Recall*: 0.9
- *Precision*: 0.6
- *F1-score*: 0.72
- Área bajo la curva ROC: 0.982

Por último, para la ResNet50 se muestran la matriz de confusión y la curva ROC resultantes en las figuras 5.36e y 5.36f, respectivamente y los resultados de las métricas analizadas son las siguientes:

		Segmentación	Clasificación					
Métrica:		**Coeficiente de Dice**	**F1-score**			**Área bajo la curva ROC**		
Modelo:		**Custom U-Net**	**ResNet18**	**ResNet34**	**ResNet50**	**ResNet18**	**ResNet34**	**ResNet50**
Dataset 1	mag_stream=20	0.928	0.963	-	-	1.0	-	-
	mag_stream=22	0.1413	0.878	-	-	0.9914	-	-
Dataset 2		0.8924	0.952	0.974	0.95	0.999	1.0	0.998
Dataset 3		0.799	0.6128	0.72	0.51	0.9225	0.982	0.941

Tabla 5.15: Resultados finales de la experimentación.

- *Accuracy*: 0.84
- *Recall*: 0.85
- *Precision*: 0.369
- *F1-score*: 0.51
- Área bajo la curva ROC: 0.941

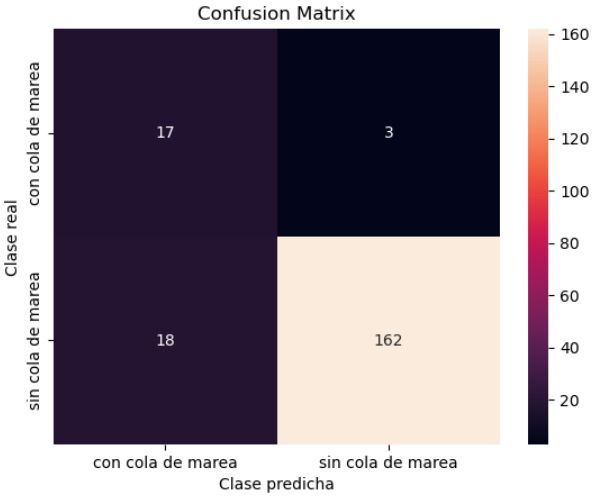
A la vista de estos resultados, el mejor modelo de los tres entrenados es también la ResNet34, ya que, al igual que con el *Dataset* 2, es la que ha alcanzado un mayor valor en las métricas *F1* y área bajo la curva. Sin embargo, en este caso, las diferencias son un poco mayores.

5.4. Comparación de resultados

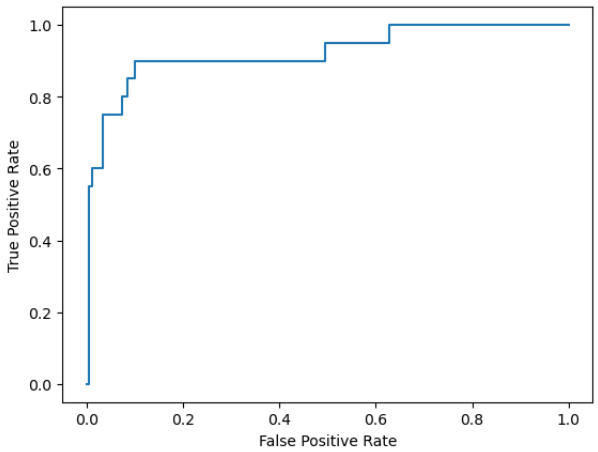
En esta sección se presentan tablas y gráficos comparativos de los resultados obtenidos hasta ahora. En la tabla 5.15 se muestra el resumen de todos los experimentos realizados en los tres niveles de *datasets*. Para la segmentación se muestra el valor obtenido del coeficiente de *Dice* y para la clasificación se muestran los valores de las métricas *F1-score* y AUC.

En la figura 5.37 se muestra un gráfico de líneas comparativo del coeficiente de *Dice* calculado para el modelo de segmentación obtenido en los distintos niveles de *dataset*, exceptuando el *dataset* 1 con `mag_stream=22`, ya que no aporta nada en esta comparativa. Como podemos observar, va decreciendo según aumentamos la dificultad y, en particular, hay un gran descenso del *dataset* 2 al 3, de casi un 0.1, lo que implica que la variabilidad de los datos ha aumentado considerablemente en la última versión del *script* de generación de imágenes sintéticas que hemos estado usando.

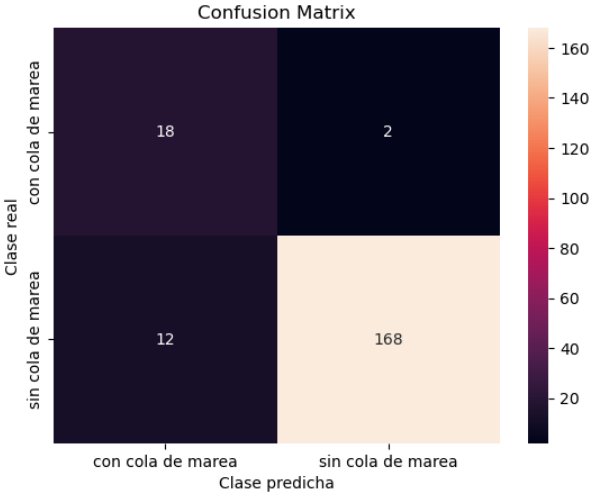
Por otro lado, en las figuras 5.38 y 5.39 se muestra la comparativa de las métricas de *F1-score* y AUC, respectivamente, de los distintos modelos de ResNet entrenados para cada versión del *dataset*. Al igual que en segmentación, la precisión de los modelos se va reduciendo a medida que aumentamos la complejidad de las imágenes sintéticas, habiendo un gran salto entre el *dataset* 2 y el 3. Además, la ResNet34 es ligeramente superior a la ResNet18 y a la ResNet50 en cuanto al desempeño alcanzado, por ello, vamos a elegirlo como mejor modelo de clasificación de cara a la siguiente sección de esta memoria.



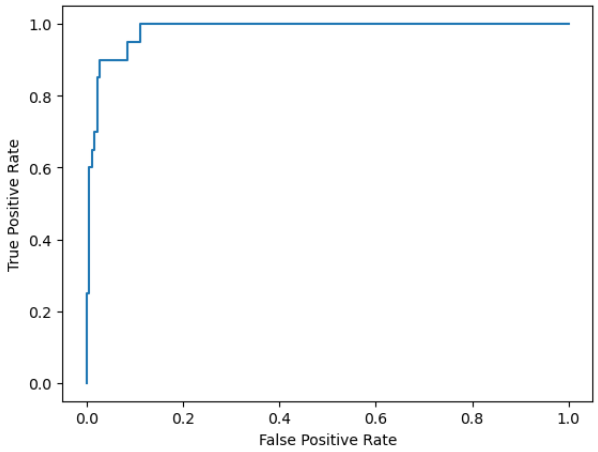
(a) Matriz de confusión de ResNet18



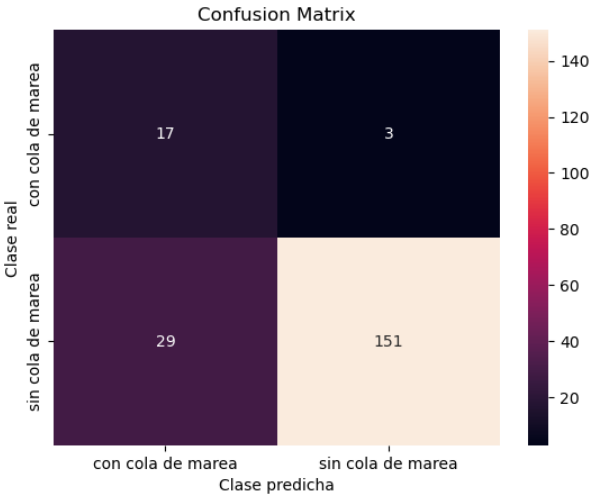
(b) Curva ROC de ResNet18



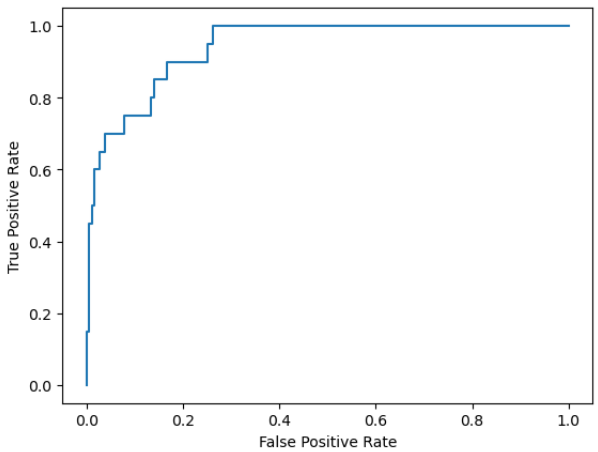
(c) Matriz de confusión de ResNet34



(d) Curva ROC de ResNet34



(e) Matriz de confusión de ResNet50



(f) Curva ROC de ResNet50

Figura 5.36: Matrices de confusión y curvas ROC de ResNet18, ResNet34 y ResNet50 sin preentrenar con el *dataset* 3.

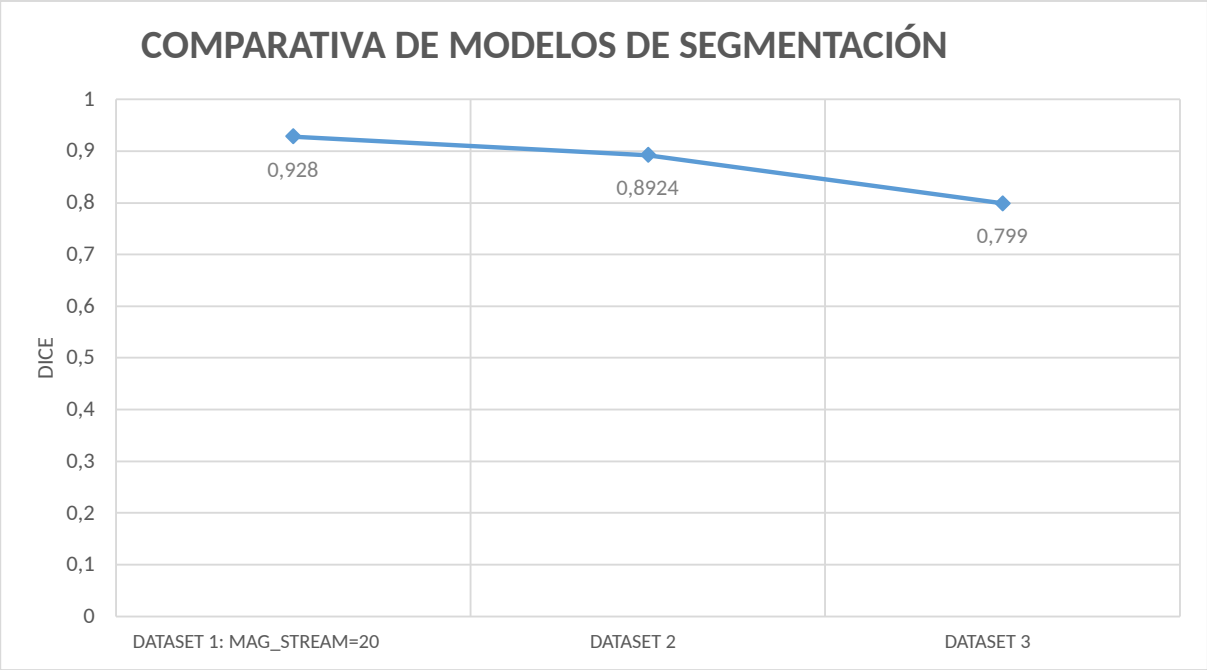


Figura 5.37: Gráfico comparativo del coeficiente de *Dice* de los modelos de segmentación entrenados.

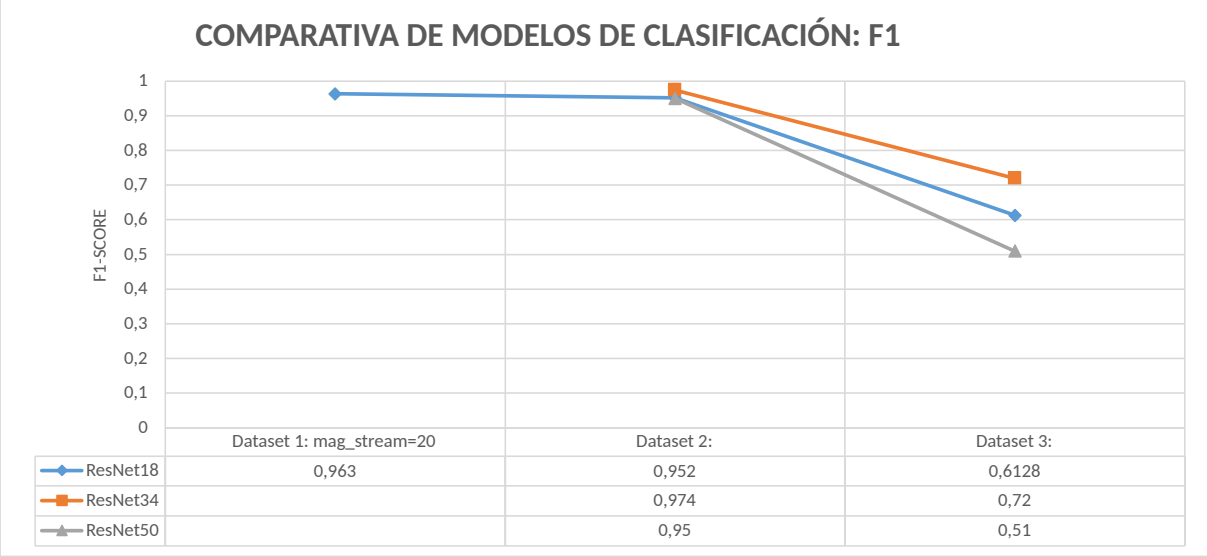


Figura 5.38: Gráfico comparativo del *F1-score* de los modelos de clasificación entrenados.

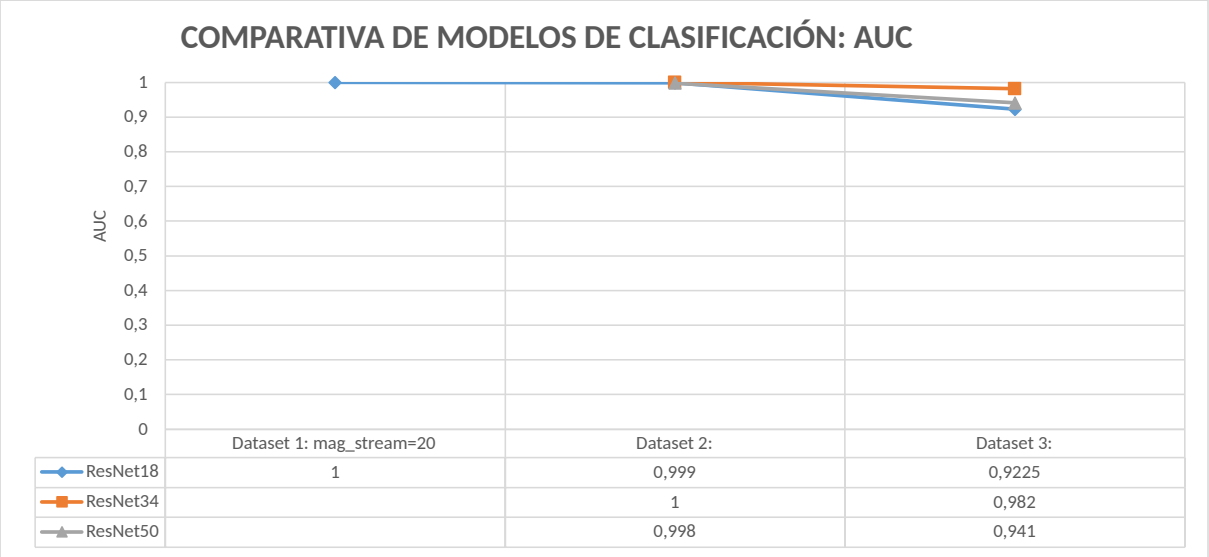


Figura 5.39: Gráfico comparativo del área bajo la curva ROC de los modelos de clasificación entrenados.

5.5. *Pipeline* completo de validación final

Para comprobar la eficacia de los modelos entrenados, tanto de segmentación como de clasificación, vamos a crear un conjunto de prueba final usando el *script* del *dataset* 3 y a evaluar el *pipeline* completo (clasificación+segmentación) sobre él. En la figura 5.40 se muestra un diagrama de flujo de este *pipeline*. Primero se pasarán todas las imágenes por la red de clasificación, calculando las métricas habituales. Las imágenes que la red de clasificación catalogue como que tienen cola de marea entrarán a la red de segmentación. Si estaban correctamente clasificadas, se podrá cuantificar el *Dice* a partir de su máscara, pero si no, igualmente se intentarán segmentar y se intentarán obtener conclusiones sobre lo que ha intentado segmentar y por qué la red previa las clasificó de esa forma. Dado que los conjuntos de datos que usamos son sintéticos y, por ello, podemos generar el número de ellos que queramos, hemos generado 1000 nuevas imágenes, de las cuales, 900 no tienen cola de marea, y 100 sí tienen cola de marea(junto con su máscara de segmentación).

La red de clasificación que se va a usar es la ResNet34, ya que es la que mejor desempeño ha obtenido, como se ha comentado en la sección anterior. La red de segmentación será la red en U entrenada con el último conjunto de datos. Los resultados de las métricas analizadas son las siguientes:

- *Accuracy*: 0.96
- *Recall*: 0.92
- *Precision*: 0.742
- *F1-score*: 0.821

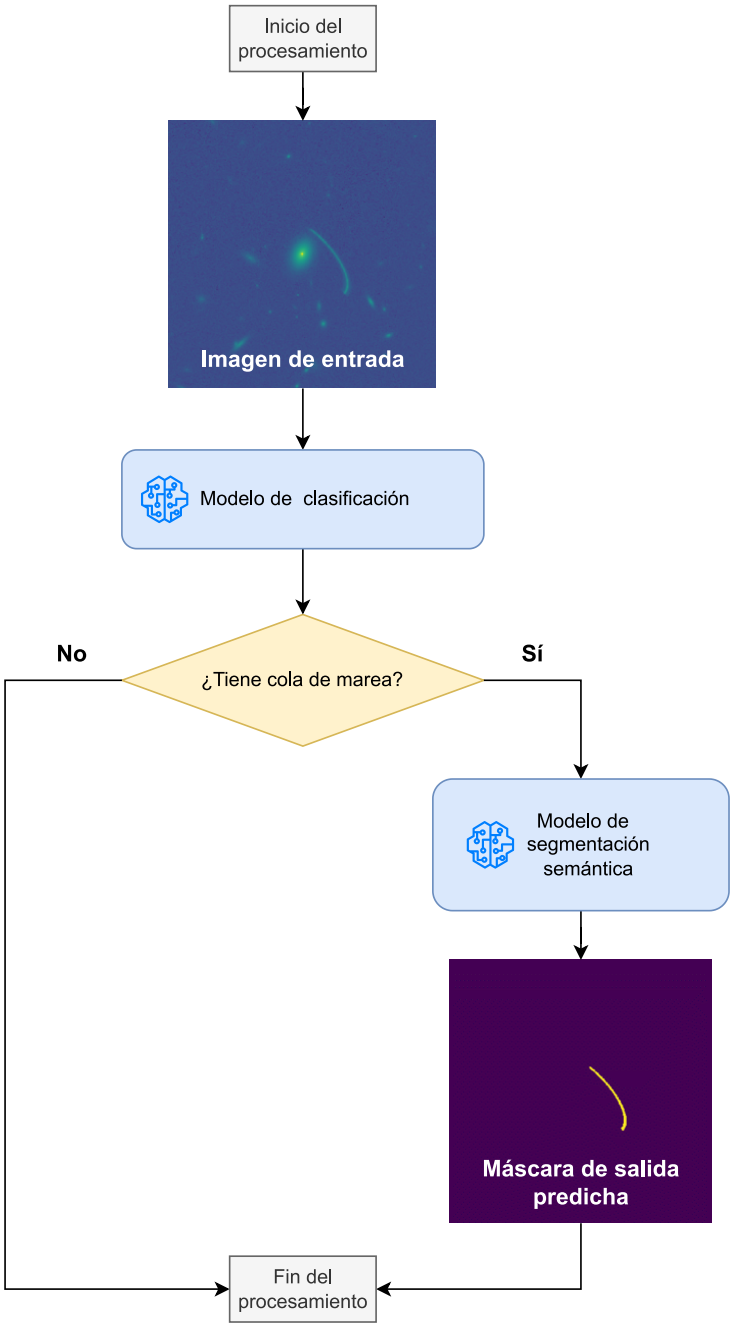
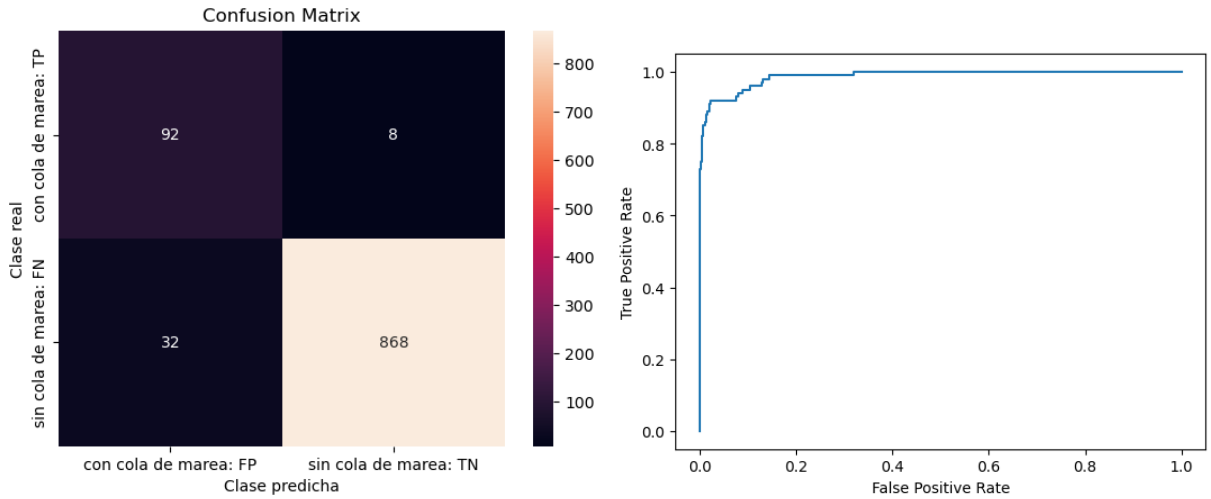


Figura 5.40: Diagrama de flujo del *pipeline* de clasificación y segmentación de colas de marea



(a) Matriz de confusión de con el *dataset* de test final

(b) Curva ROC de la ResNet34 con el *dataset* de test final

Figura 5.41: Matrices de confusión y curvas ROC del *pipeline* de validación final

- *Área bajo la curva ROC*: 0.987
- *Dice*: 0.83

En la figura 5.32c se muestra la matriz de confusión obtenida, junto con la curva ROC en la figura 5.32d. En el apéndice B se muestran todas las imágenes con cola de marea (y sus etiquetas) de este conjunto de datos de prueba final y las inferencias realizadas por la red de segmentación entrenada.

En esta última fase hemos podido comprobar que, para un conjunto de datos de prueba mayor, las métricas de clasificación se han mantenido similares con respecto a los valores obtenidos con el conjunto de prueba del *dataset* 3, mostrados en el apartado 5.3.2. Sin embargo, el *Dice* obtenido en esta prueba ha sido ligeramente superior: 0.83 frente a 0.799. Esto se debe a que la red de segmentación solo ha recibido las imágenes con colas de marea que la red previa ha clasificado correctamente. Si el clasificador ha errado significa que esas imágenes eran más confusas y complejas para él, y por lo tanto, también lo serían para el segmentador, por lo que esa pequeña subida de en el *Dice* implica que no ha segmentado imágenes con colas de marea dudosas.

Ante esto, cabe plantearse si debemos relajar el umbral del clasificador para permitir más falsos positivos, aún sabiendo que confundirán al segmentador, o si debemos mantener esta política, que hace que el segmentador solo reciba imágenes muy claras y por lo tanto, halle los píxeles de la cola de marea de forma muy confiable.

6: Conclusiones y líneas de trabajo futuras

En este último capítulo se presentan las conclusiones obtenidas tras la realización del proyecto, así como las líneas de trabajo futuras que podrían complementar y mejorar el trabajo realizado.

6.1. Conclusiones

El principal objetivo establecido al inicio de este proyecto era conseguir modelos de aprendizaje profundo que fueran capaces de clasificar y segmentar colas de marea en imágenes astronómicas con un desempeño razonable, y tras la discusión de resultados mostrada en el capítulo 4.4.2, podemos concluir que hemos conseguido alcanzar este objetivo, tanto en clasificación como en segmentación.

En la labor de investigación previa a este trabajo, realizada en la asignatura de I+d+i, obteníamos métricas de *Dice* sobre test superiores al 0.95, algo muy poco común en problemas de segmentación semántica. Sin embargo, el conjunto de datos generado no era lo suficientemente realista y no se asemejaba a imágenes de colas de marea reales, sino que era más bien una prueba de concepto, que indicara que las técnicas que estamos utilizando eran adecuadas en el contexto del problema a resolver.

Por ello, uno de los retos de este proyecto era mantener un compromiso entre la calidad de los modelos entrenados y el realismo de las imágenes artificiales generadas. Es cierto que los niveles de *Dice* y de precisión obtenidos han ido empeorando ligeramente a medida que aumentábamos la complejidad, variabilidad y realismo de las imágenes astronómicas generadas, pero siempre se han mantenido por encima de los valores previstos inicialmente.

En las últimas fases de este proyecto conseguimos generar un conjunto de datos artificiales(sección 5.3) que satisfacía las condiciones necesarias para ser comparable a datos reales tomados por telescopios espaciales, aunque el *Dice* en segmentación había bajado hasta un 0.8 aproximadamente, un valor más común en problemas de segmentación semántica y que no se suele considerar como un resultado deficiente.

Otra cuestión a destacar es que hemos sido capaces de entrenar modelos desde cero, tanto en segmentación como en clasificación, sin emplear *transfer learning*. En segmentación, debido a haber creado una arquitectura personalizada específica para este problema, no había otra posibilidad que entrenar el modelo desde cero. Por otro lado, aunque en las tareas de clasificación hemos utilizado arquitecturas muy conocidas y usadas en el ámbito de la visión artificial como son las ResNets, no hemos hecho uso de los modelos preentrenados, sino que hemos establecido los valores de los pesos de forma aleatoria mediante la inicialización de Kaiming He.

Por lo tanto, hemos logrado obtener resultados bastante satisfactorios en este problema específico sin depender de datos ajenos a nuestro proyecto de investigación, ya que hemos generado todos los datos nosotros mismos y los hemos utilizado para entrenar los modelos desde cero.

6.2. Líneas de trabajo futuras

La razón de tener como objetivo el disponer de modelos que sean capaces de segmentar colas de marea en imágenes astronómicas realistas, es porque, tanto el alumno autor del trabajo como los tutores responsables, tienen la intención de proseguir con este trabajo y realizar una publicación científica partiendo de la base de conocimiento obtenida.

Como hemos anticipado en la sección 3.1.3, en julio de 2023, la ESA lanzará el telescopio espacial Euclid, que se encargará, entre otras muchas cosas, de estudiar y fotografiar el espacio profundo. Nuestro siguiente paso es acceder a simulaciones oficiales del telescopio Euclid para evaluar los modelos entrenados y, posteriormente, disponer de las imágenes reales que este telescopio tomará y evaluar de nuevo la calidad de nuestros modelos.

Lo esperable es que el *Dice* y el *accuracy* obtenidos con imágenes reales sea mucho peor que con los conjuntos de datos sintéticos, por lo que habrá que realizar una gran labor de refinamiento de las técnicas usadas, como el preprocesado y el *data augmentation*, y de reentrenamiento de los modelos de forma iterativa, aunque partiendo de la base metodológica de experimentación utilizada y del código de entrenamiento y de evaluación de modelos desarrollado en este trabajo.

Además, existen numerosas arquitecturas de aprendizaje profundo muy reconocidas y exitosas para la segmentación semántica y la clasificación, aparte de las las redes en U y redes residuales, que no hemos utilizado en nuestra investigación. En el futuro, por lo tanto, habrá que crear un *grid* de experimentación mucho más amplio para encontrar la combinación de hiperparámetros óptima para el problema que estamos resolviendo.

Apéndices

Apéndice A

Repositorio de código

En este apéndice se incluye la documentación necesaria para que el lector de este documento pueda replicar cualquier experimento de los realizados en este proyecto y modificar los aspectos que considere oportunos. Todo el código usado en este trabajo está subido en el siguiente repositorio de GitLab: https://gitlab.inf.uva.es/dardela/tfm-segmentacion-de-colas-de-marea.git. Los conjuntos de datos generados y los modelos entrenados no están subidos al repositorio remoto debido a su tamaño conjunto. La estructura de este repositorio está organizada de la siguiente forma:

- `Dataset1`: Directorio que contiene el *script* de generación de imágenes y los Jupyter Notebook usados en el nivel de experimentación del conjunto de datos 1(sección 5.1). Tiene la siguiente estructura:
  - `create_mock_tidal_tails_dataset_1_mag_stream_20.sh`: *Script* en línea de comandos de Unix usado para generar imágenes artificiales mediante funciones de GNUAstro para el *dataset* 1 con el parámetro `mag_stream=20`.
  - `Segmentacion_mag_stream_20`: Directorio que contiene los Jupyter Notebook usados para entrenar y evaluar los modelos de segmentación con el *dataset* 1 con el parámetro `mag_stream=20`.
  - `Clasificacion_mag_stream_20`: Directorio que contiene los Jupyter Notebook usados para entrenar y evaluar los modelos de clasificacion con el *dataset* 1 con el parámetro `mag_stream=20`.
  - `create_mock_tidal_tails_dataset_1_mag_stream_22.sh`: *Script* en línea de comandos de Unix usado para generar imágenes artificiales mediante funciones de GNUAstro para el *dataset* 1 con el parámetro `mag_stream=22`.
  - `Segmentacion_mag_stream_22`: Directorio que contiene los Jupyter Notebook usados para entrenar y evaluar los modelos de segmentación con el *dataset* 1 con el parámetro `mag_stream=22`.

  - `Clasificacion_mag_stream_22`: Directorio que contiene los Jupyter Notebook usados para entrenar y evaluar los modelos de clasificacion con el *dataset* 1 con el parámetro `mag_stream=22`.

- `Dataset2`: Directorio que contiene el *script* de generación de imágenes y los Jupyter Notebook usados en el nivel de experimentación del conjunto de datos 2(sección 5.2). Tiene la siguiente estructura:

  - `create_mock_tidal_tails_dataset_2.sh`: *Script* en línea de comandos de Unix usado para generar imágenes artificiales mediante funciones de GNUAstro para el *dataset* 2.
  - `Segmentacion`: Directorio que contiene los Jupyter Notebook usados para entrenar y evaluar los modelos de segmentación con el *dataset* 2.
  - `Clasificacion`: Directorio que contiene los Jupyter Notebook usados para entrenar y evaluar los modelos de clasificacion con el *dataset* 2.

- `Dataset3`: Directorio que contiene el *script* de generación de imágenes y los Jupyter Notebook usados en el nivel de experimentación del conjunto de datos 3(sección 5.3). Tiene la siguiente estructura:

  - `create_mock_tidal_tails_dataset_3.sh`: *Script* en línea de comandos de Unix usado para generar imágenes artificiales mediante funciones de GNUAstro para el *dataset* 3.
  - `Segmentacion`: Directorio que contiene los Jupyter Notebook usados para entrenar y evaluar los modelos de segmentación con el *dataset* 3.
  - `Clasificacion`: Directorio que contiene los Jupyter Notebook usados para entrenar y evaluar los modelos de clasificacion con el *dataset* 3.

- `Auxiliar`: Directorio que contiene distintos *scripts* y ficheros usados por todos los conjuntos de datos. En concreto contiene los siguientes elementos:

  - `GenerarMascaras.ipynb`: Jupyter Notebook utilizado para generar las máscaras de segmentación a partir de los ficheros `.fits` generados por el *script* de GNUAstro.
  - `inserting_tidal_tails.py`: *Script* en Python para modificar el fondo de las imágenes generadas por GNUAstro y añadir, mediante `MontagePy`, un fragmento de una imagen real. Este *script* requiere tener en su mismo directorio un fichero `.fits` que representa una región del cielo fotografiada por un telescopio espacial. La imagen usada no se ha subido al repositorio remoto debido a que su tamaño es de aproximadamente 14GB.
  - `psf_visSC3ovs6_centered_realpixscale.norm.fits`: Fichero `.fits` que contiene la PSF necesaria para generar las imágenes simuladas.

- `PipelineFinal`: Directorio que contiene el código utilizado en la evaluación de los mejores modelos finales mediante el *pipeline* compuesto de clasificación y segmentación. En concreto contiene el siguiente elemento:
  - `Pipeline_clasificacion+segmentacion.ipynb`: Jupyter Notebook que contiene el código Python necesario para ejecutar el *pipeline* compuesto de clasificación y segmentación y visualizar sus resultados.

Apéndice B

Inferencias del *Dataset* final

En la figura B se muestran los imágenes y etiquetas del conjunto de datos final y las inferencias realizadas por la red de segmentación entrenada. Se muestran agrupados en grupos de 3 (en 2 columnas), de izquierda a derecha, la imagen de entrada a la red, la máscara real usada como etiqueta y la máscara segmentada por la red.

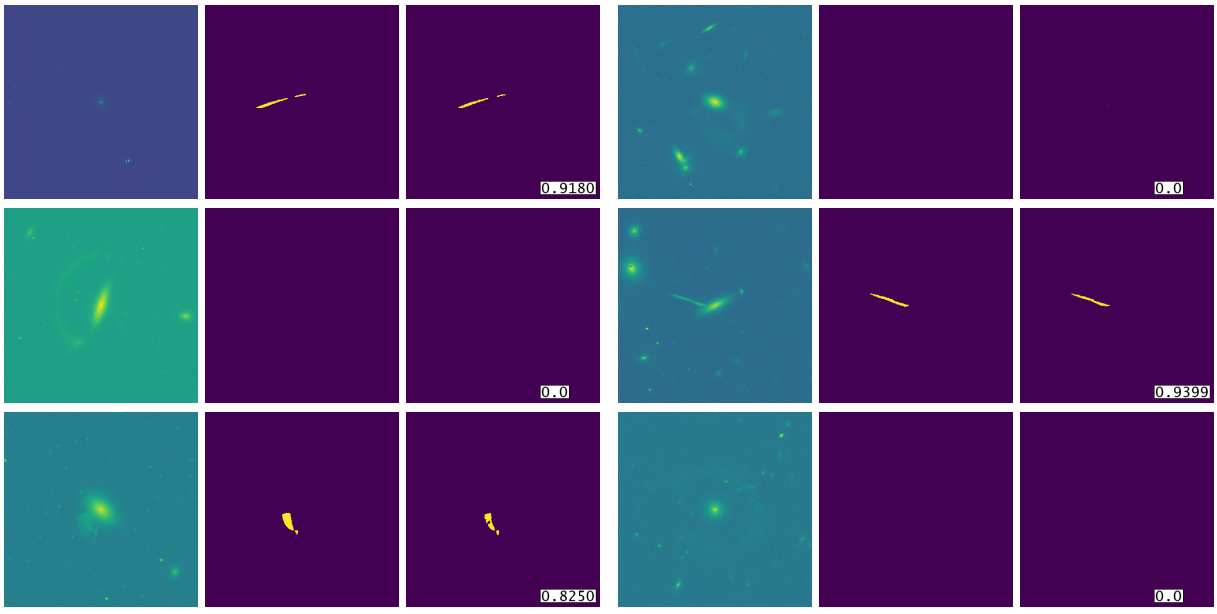


Figura B.1: Inferencias con el *dataset* final

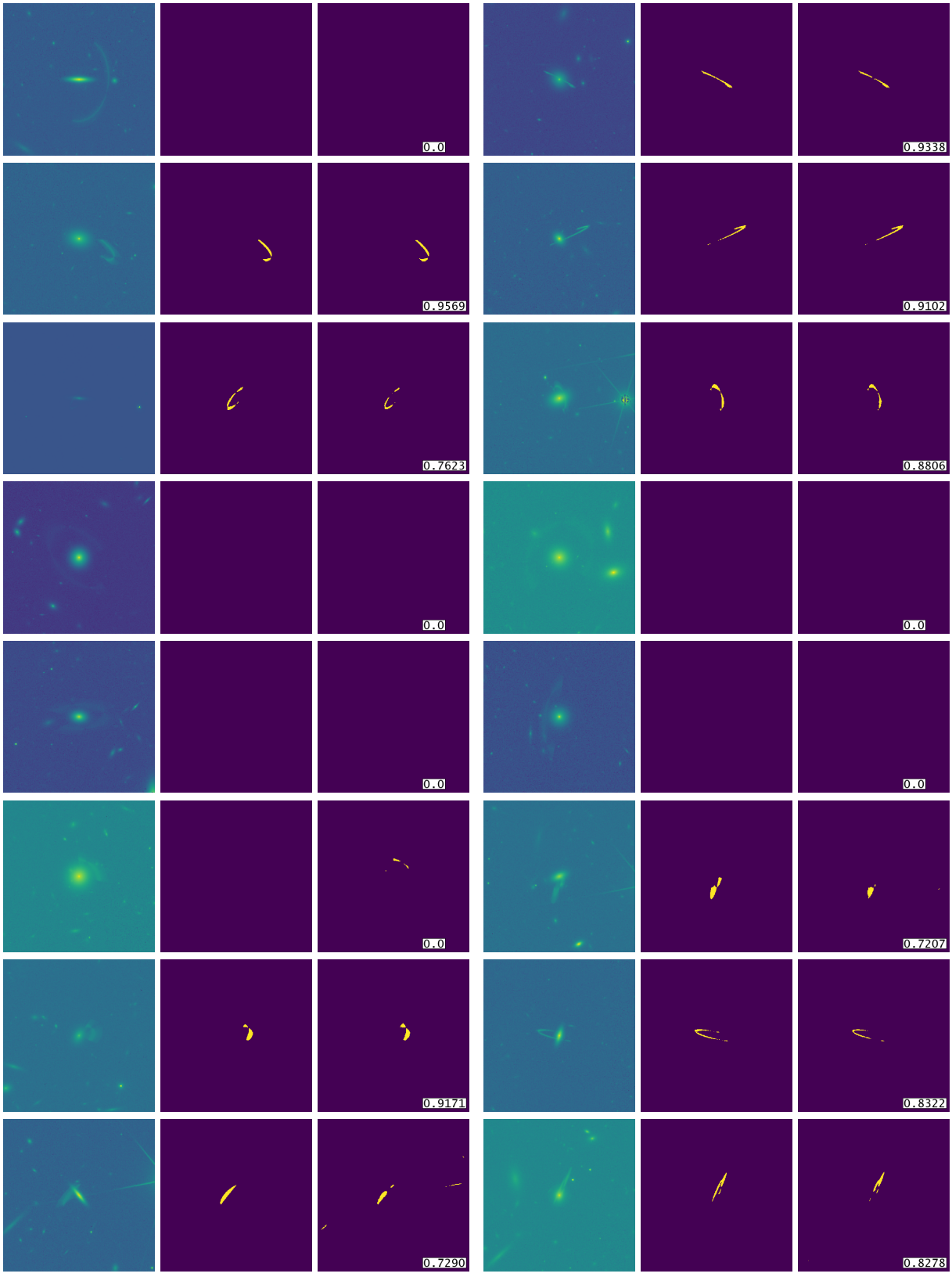


Figura B.1: Inferencias con el *dataset* final

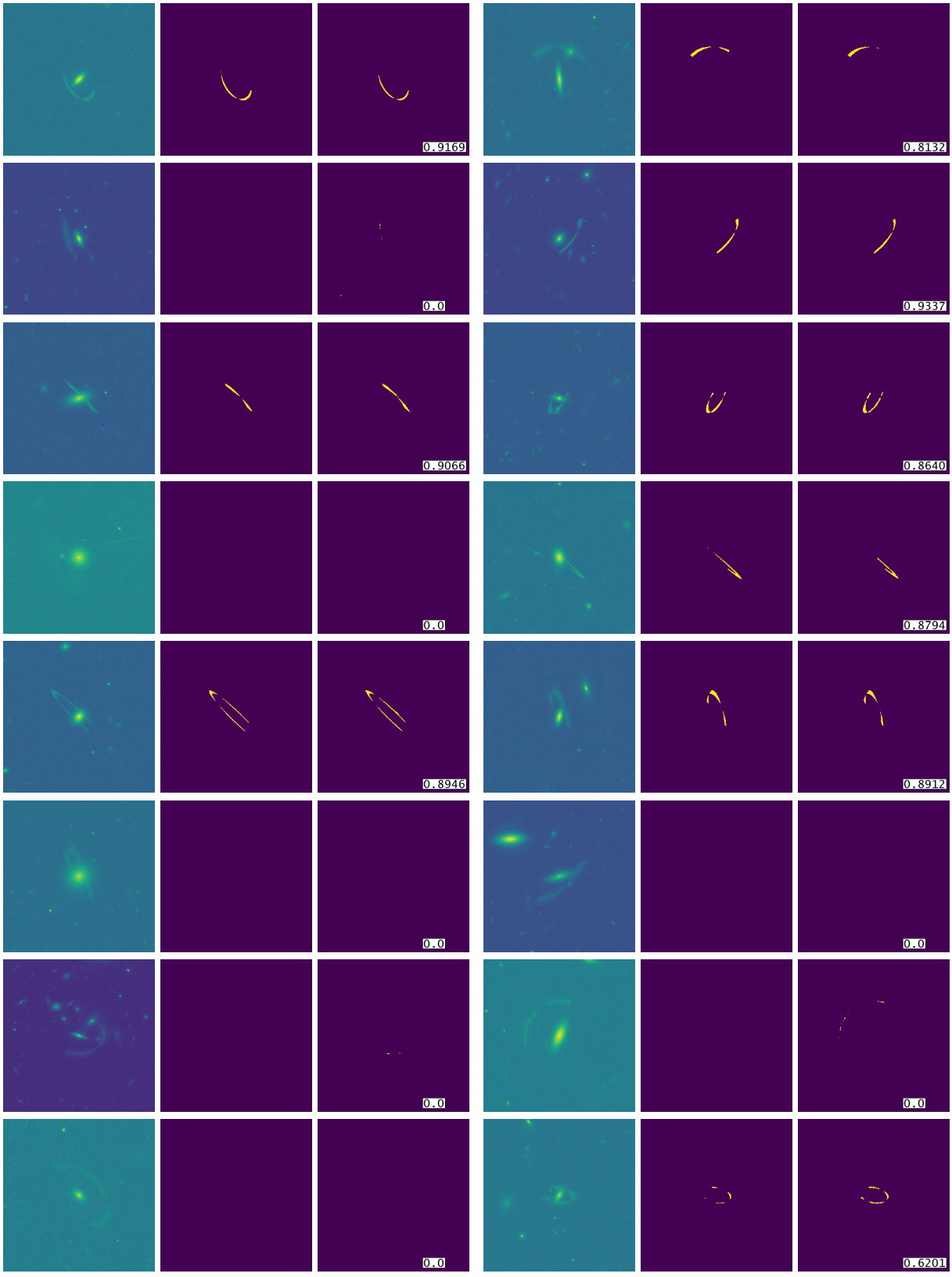


Figura B.1: Inferencias con el *dataset* final

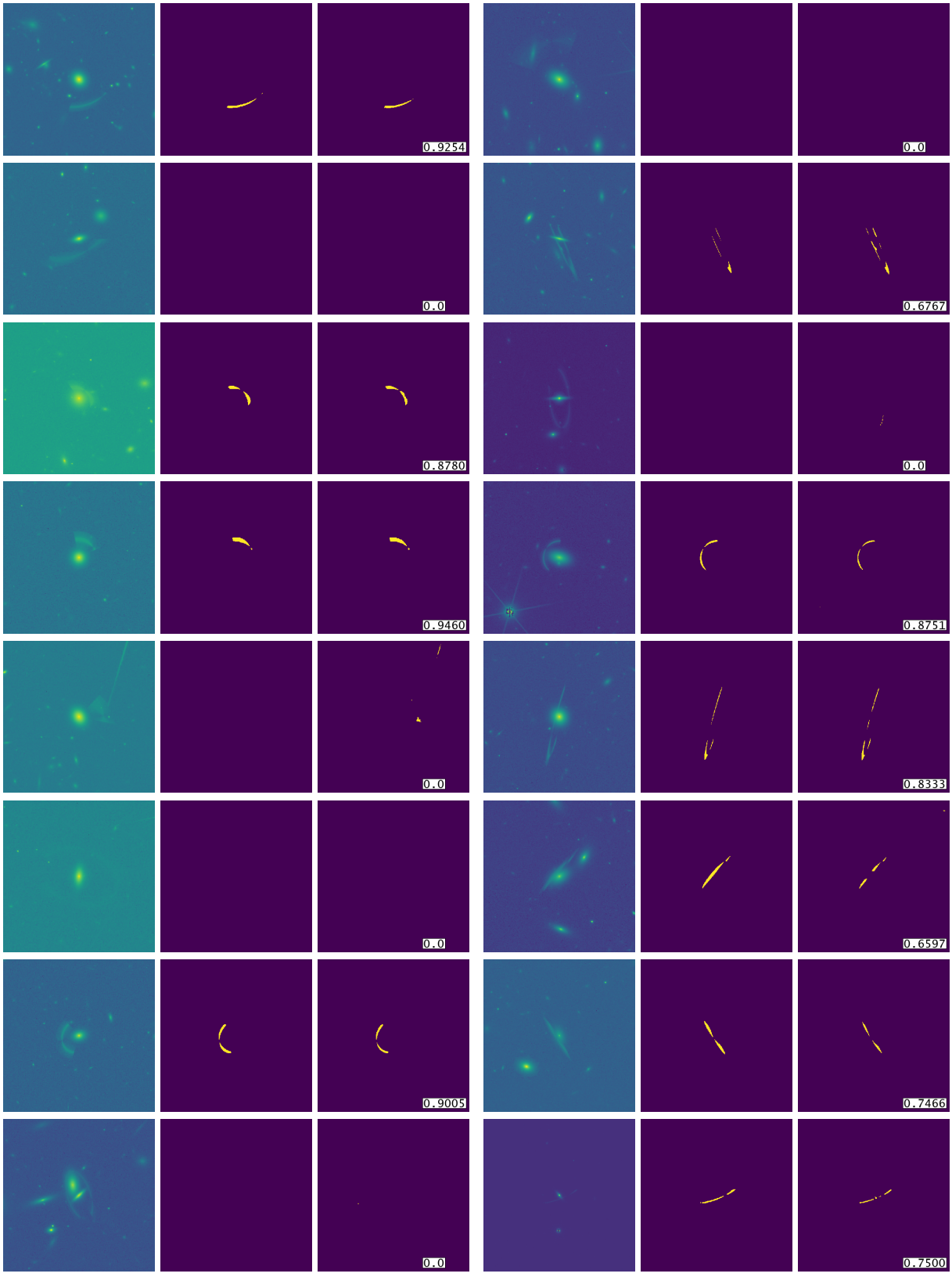


Figura B.1: Inferencias con el *dataset* final

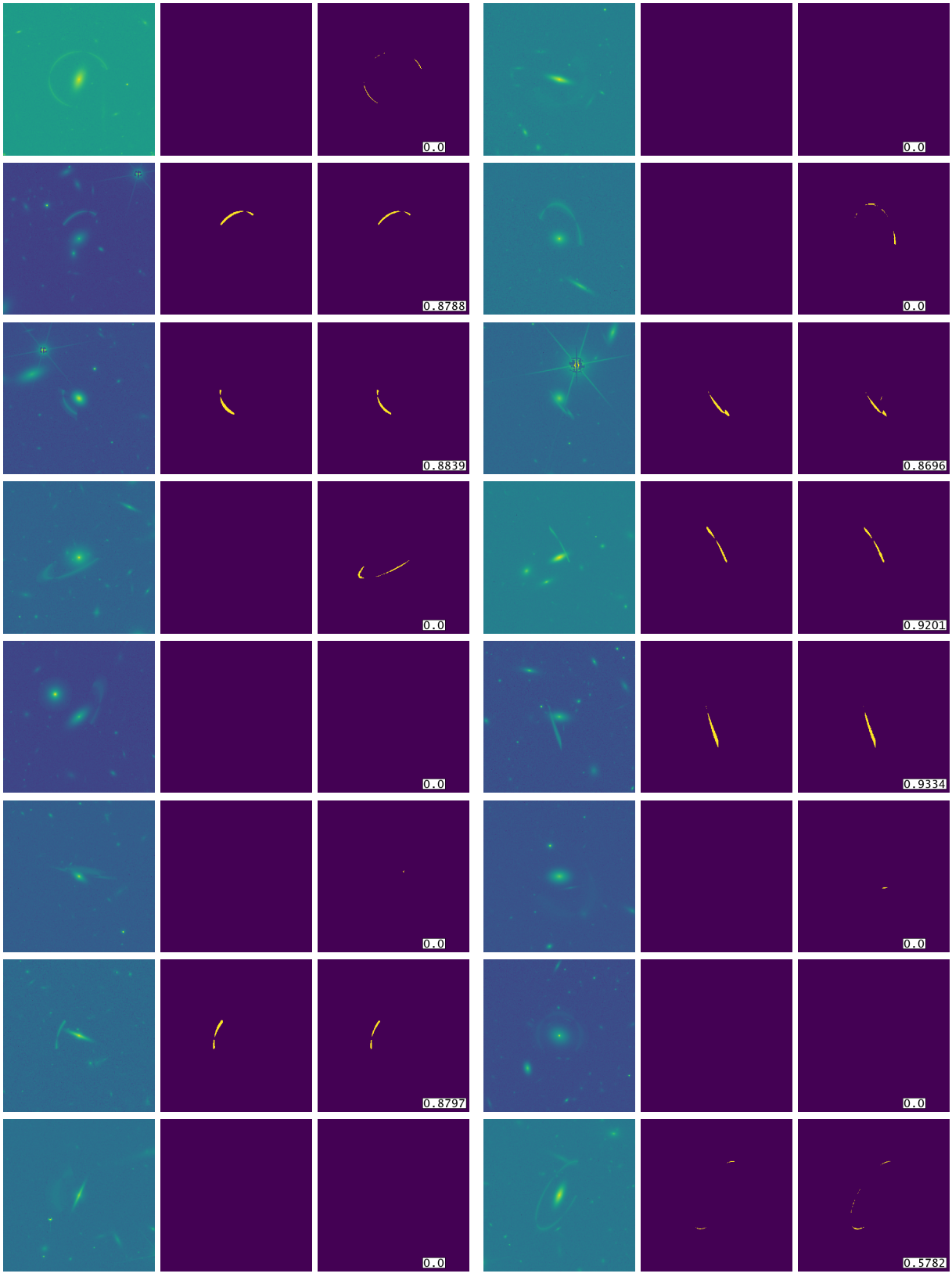


Figura B.1: Inferencias con el *dataset* final

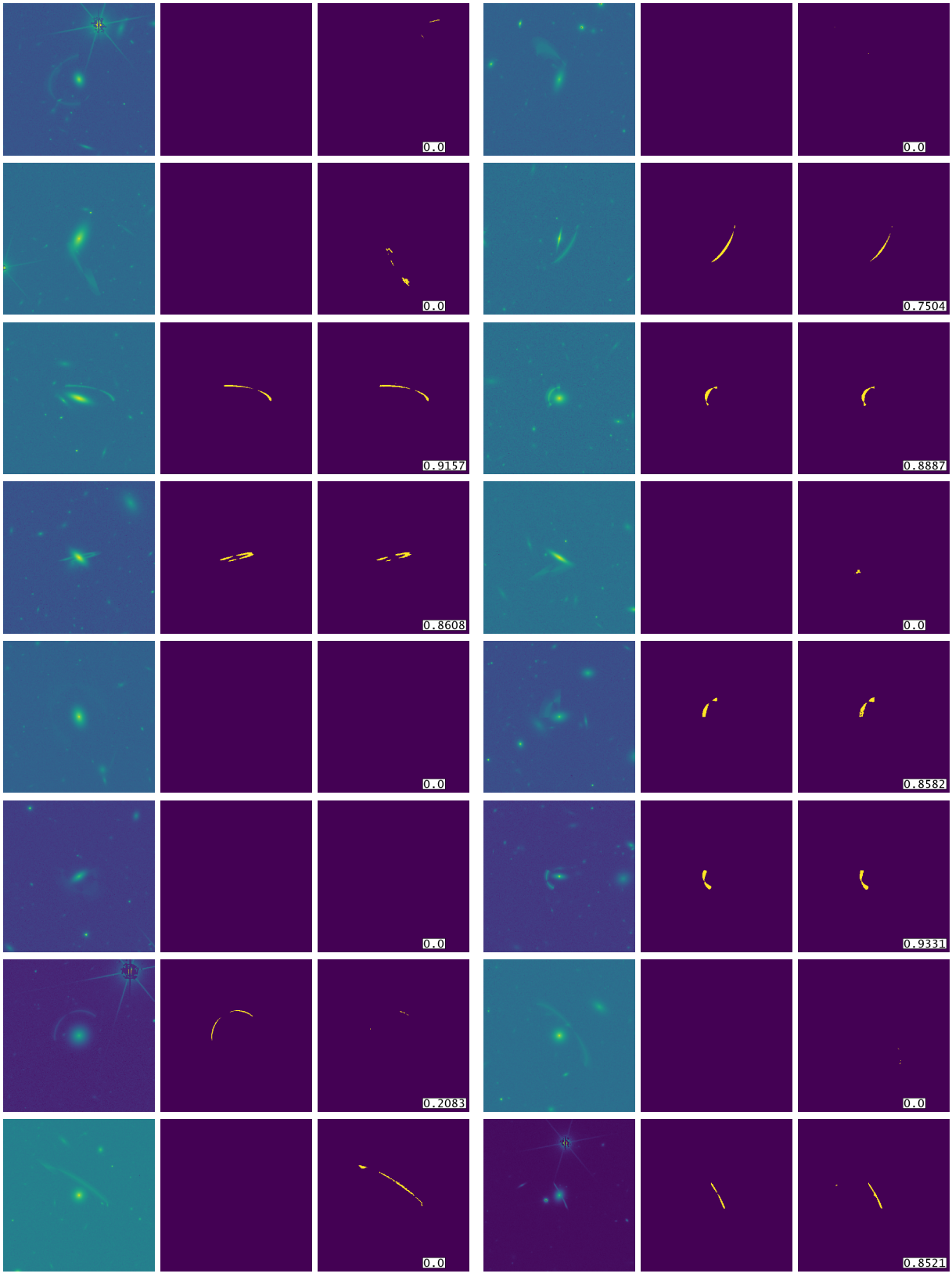


Figura B.1: Inferencias con el *dataset* final

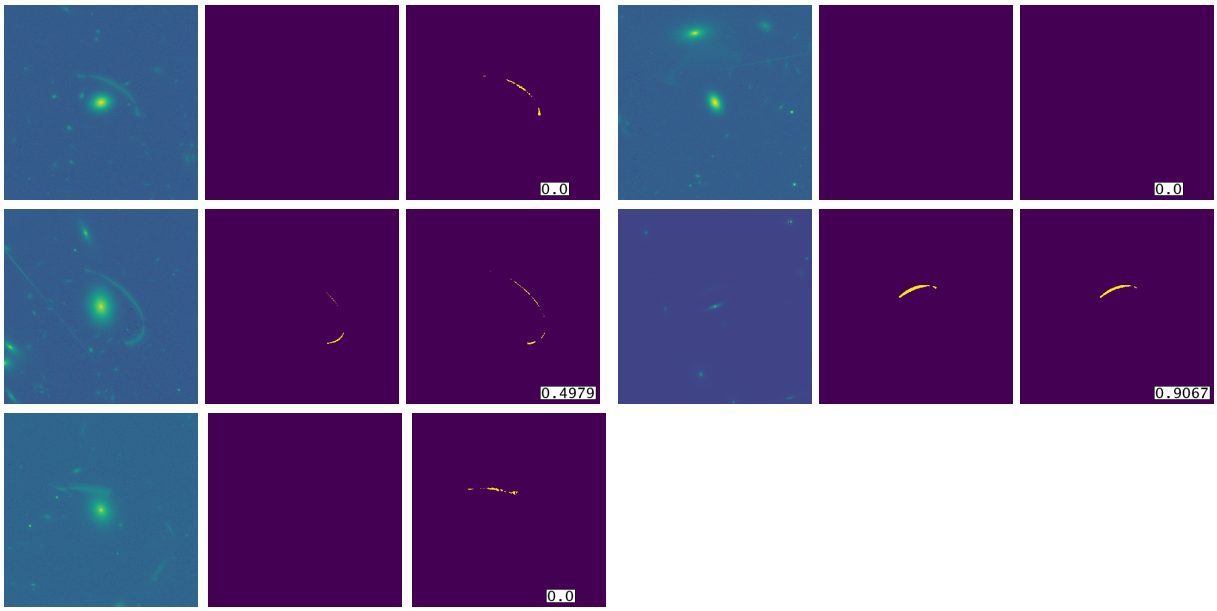


Figura B.1: Inferencias con el *dataset* final

Siglas

2dF *2 degree Field*. VI, 13

AUC *Area under the ROC Curve*. 46, 58

CNN *Convolutional Neural Network*. 21, 23, 53, 56

CPU *Central Processing Unit*. 31

EDT *Estructura de desglose del trabajo*. 4, 6

ESA *European Space Agency*. 65

FITS *Flexible Image Transport System*. 29

GPU *Graphics Processing Unit*. 30, 31

IOU *Intersection Over Union*. 41

LSST *Legacy Survey of Space and Time*. 14

Mpc *Megapársec*. 12, 13

PMBOK *Project Management Body of Knowledge*. 3

PMI *Project Management Institute*. 3

PSF *Point Spread Function*. 33, 68

ReLU *Rectified Linear Unit*. VI, 26, 27

ROC *Receiver Operating Characteristic*. VII, 4, 6, 41, 45

SCORE *SCrum Of REsearch*. 8

Bibliografía

[1] Agency, E. S. Euclid science archive system.

[2] Akhlaghi, M., and Ichikawa, T. Noise-based detection and segmentation of nebulous objects. *ApJS 220* (Sept. 2015), 1.

[3] Beckwith, S. V. W., Stiavelli, M., Koekemoer, A. M., Caldwell, J. A. R., Ferguson, H. C., Hook, R., Lucas, R. A., Bergeron, L. E., Corbin, M., Jogee, S., Panagia, N., Robberto, M., Royle, P., Somerville, R. S., and Sosey, M. The Hubble Ultra Deep Field. *The Astronomical Journal 132*, 5 (Nov. 2006), 1729–1755.

[4] Borne, K. D. Astroinformatics: A 21st century approach to astronomy, 2009.

[5] Chollet, F. Building autoencoders in keras. *Keras Blog* (2016).

[6] Colless, M., and et al. The 2dF Galaxy Redshift Survey: Final Data Release. *Monthly Notices of the Royal Astronomical Society 352*, 3 (2004), 825–883.

[7] Consortium, E. Euclid and the origin of the accelerating universe.

[8] de Vaucouleurs, G. Recherches sur les Nebuleuses Extragalactiques. *Annales d'Astrophysique 11* (Jan. 1948), 247.

[9] Draelos, R. Measuring performance: Auc (auroc).

[10] Dumoulin, V., and Visin, F. A guide to convolution arithmetic for deep learning, 2018.

[11] Freeman, K. C. On the Disks of Spiral and S0 Galaxies. *The Astrophysical Journal 160* (June 1970), 811.

[12] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A.,

del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature 585*, 7825 (Sept. 2020), 357–362.

[13] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

[14] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[15] Hicks, M., and Foster, J. Adapting scrum to managing a research group.

[16] Jacob, J. C., Katz, D. S., Berriman, G. B., Good, J. C., and Laity, A. C. Montage: A grid-enabled image mosaicking workflow system. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing* (2004), IEEE, pp. 40–40.

[17] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C. Jupyter notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (2016), F. Loizides and B. Schmidt, Eds., IOS Press, pp. 87 – 90.

[18] Lawtomated. A.i. technical machine vs deep learning, n.d.

[19] Martin, G., Bazkiaei, A. E., Spavone, M., Iodice, E., Mihos, J. C., Montes, M., Benavides, J. A., Brough, S., Carlin, J. L., Collins, C. A., Duc, P. A., Gó mez, F. A., Galaz, G., Hernández-Toledo, H. M., Jackson, R. A., Kaviraj, S., Knapen, J. H., Martínez-Lombilla, C., McGee, S., O'Ryan, D., Prole, D. J., Rich, R. M., Román, J., Shah, E. A., Starkenburg, T. K., Watkins, A. E., Zaritsky, D., Pichon, C., Armus, L., Bianconi, M., Buitrago, F., Busá, I., Davis, F., Demarco, R., Desmons, A., García, P., Graham, A. W., Holwerda, B., Hon, D. S. H., Khalid, A., Klehammer, J., Klutse, D. Y., Lazar, I., Nair, P., Noakes-Kettel, E. A., Rutkowski, M., Saha, K., Sahu, N., Sola, E., Vázquez-Mata, J. A., Vera-Casanova, A., and Yoon, I. Preparing for low surface brightness science with the vera c. rubin observatory: Characterization of tidal features from mock images. *Monthly Notices of the Royal Astronomical Society 513*, 1 (apr 2022), 1459–1487.

[20] Martí nez-Delgado, D., Cooper, A. P., Román, J., Pillepich, A., Erkal, D., Pearson, S., Moustakas, J., Laporte, C. F. P., Laine, S., Akhlaghi, M., Lang, D., Makarov, D., Borlaff, A. S., Donatiello, G., Pearson, W. J., Miró-Carretero, J., Cuillandre, J.-C., Domínguez, H., Roca-Fàbrega, S., Frenk, C. S., Schmidt, J., Gómez-Flechoso, M. A., Guzman, R., Libeskind, N. I., Dey, A., Weaver, B. A., Schlegel, D., Myers, A. D., and Valdes, F. G. Hidden depths in the local universe: The stellar stream legacy survey. *Astronomy & Astrophysics 671* (mar 2023), A141.

[21] Nes, F. L. V., and Bouman, M. A. Spatial modulation transfer in the human eye. *J. Opt. Soc. Am. 57*, 3 (Mar 1967), 401–406.

[22] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[23] Price-Whelan, A. M., Sipőcz, B., Günther, H., Lim, P., Crawford, S., Conseil, S., Shupe, D., Craig, M., Dencheva, N., Ginsburg, A., et al. The astropy project: Building an open-science project and status of the v2. 0 core package. *The Astronomical Journal 156*, 3 (2018), 123.

[24] *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, 7 ed. Project Management Institute, 2021.

[25] Renacuajo, G. Galaxia renacuajo — Wikipedia, la enciclopedia libre, 2022.

[26] Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. *CoRR abs/1505.04597* (2015).

[27] Rubin, O. V. Tea fiber installation. [Fotografía], 6 2022.

[28] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[29] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development 44*, 1.2 (2000), 206–226.

[30] Sánchez, H. D., Martin, G., Damjanov, I., Buitrago, F., Huertas-Company, M., Bottrell, C., Bernardi, M., Knapen, J. H., Vega-Ferrero, J., Hausen, R., Kado-Fong, E., Población-Criado, D., Souchereau, H., Leste, O. K., Robertson, B., Sahelices, B., and Johnston, K. V. Identification of tidal features in deep optical galaxy images with convolutional neural networks. *Monthly Notices of the Royal Astronomical Society 521*, 3 (mar 2023), 3861–3872.

[31] Sérsic, J. L. Influence of the atmospheric and instrumental dispersion on the brightness distribution in a galaxy. *Boletin de la Asociacion Argentina de Astronomia La Plata Argentina 6* (Feb. 1963), 41–43.

[32] Sérsic, J. L. *Atlas de Galaxias Australes*. 1968.

[33] Smithsonian Astrophysical Observatory. SAOImage DS9: A utility for displaying astronomical images in the X11 window environment. Astrophysics Source Code Library, record ascl:0003.002, Mar. 2000.

[34] Sola, E., Duc, P.-A., Richards, F., Paiement, A., Urbano, M., Klehammer, J., Bí lek, M., Cuillandre, J.-C., Gwyn, S., and McConnachie, A. Characterization of low surface brightness structures in annotated deep images. *Astronomy & Astrophysics 662* (jun 2022), A124.

[35] Sørensen, T. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content. *Det Kongelige Danske Videnskabernes Selskab 5*, 4 (1948), 1–34.