Mejora de imágenes submarinas mediante deep learning

Trabajo Final de Máster
Máster Universitario en Ciencia de Datos
Universitat Oberta de Catalunya

Autor: Leroy Deniz Pedreira
Director: Antoni Burguera Burguera
Profesor responsable: Albert Solé Ribalta

Universitat Oberta de Catalunya

Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos

Trabajo de Fin de Máster

Área: Visión por computador

Mejora de imágenes submarinas mediante deep learning

Autor: Leroy Deniz Pedreira

Director: Antoni Burguera Burguera

Profesor responsable: Albert Solé Ribalta

Barcelona, 23 de junio de 2023

"Mucha gente pequeña, en lugares pequeños,
haciendo cosas pequeñas, puede cambiar el mundo"

Eduardo Galeano, 1992

Ficha del Trabajo Final

Título del trabajo:	Mejora de imágenes submarinas mediante deep learning
Autor:	Leroy Deniz Pedreira
Director:	Antoni Burguera Burguera
Profesor responsable:	Albert Solé Ribalta
Fecha de entrega:	06/2023
Titulación o programa:	Máster Universitario en Ciencia de Datos
Área del Trabajo Final:	Visión por Computador
Idioma del trabajo:	Español
Palabras clave:	neural networks, image analysis, underwater environments
Código fuente:	https://github.com/leroydeniz/tfm
Kanban Board:	https://bit.ly/3nC4jGf
Versión:	v5.2

Índice

Índice de figuras

Índice de tablas

Índice de programas

Abstract

Mobile underwater robotics has made it easier for humans to perform, and sometimes allowed humans, to carry out tasks of analysis and recognition in underwater environments. With the advanced leaps in technology in recent years, better technological resources have become accessible and a greater computing capacity has significantly improved Autonomous Underwater Vehicle (AUV), resulting in the acquisition of a vast amount of information provided by images taken by these devices.

However, it is common in underwater environments that the conditions for capturing images are not optimal. As a result, there are a series of inconveniences, sometimes associated with the use of artificial light sources, ranging from excess lighting and even saturation or attenuation with distance, to loss of color depending on frequency.

This work investigates techniques for automatic optimization of underwater images, including analysis, design, implementation, and evaluation. It involves the analysis of various *deep learning* architectures which, with the appropriate configuration and training, enable the models obtained to generate a considerable improvement in the original images.

The ultimate goal is the design of a *deep learning* model capable of improving the quality of distorted underwater images, obtaining uniform, sharp and realistic results.

Keywords: image analysis, underwater environments, convolutional neural networks, machine learning, deep learning.

Resumen

La robótica móvil submarina ha facilitado a los humanos, y en ocasiones permitido, la realización de tareas de análisis y reconocimiento de espacios subacuáticos. Con los saltos de avanzada de la tecnología en los últimos años, se consigue el acceso a mejores recursos tecnológicos y una gran capacidad de computación, mejorando significativamente los vehículos autónomos submarinos (en inglés, *Autonomous Underwater Vehicles*, AUV), dando lugar a la obtención de una enorme cantidad de información proporcionada por las imágenes tomadas por estos dispositivos.

Sin embargo, es común que en entornos submarinos las condiciones para la captura de imágenes no sean óptimas. A raíz de esto, existen una serie de inconvenientes, en ocasiones asociados al uso de fuentes lumínicas artificiales, que van desde el exceso de iluminación, e incluso saturación o atenuación con la distancia, hasta la pérdida de color en función de la frecuencia.

Este trabajo tiene como línea de investigación el análisis, diseño, implementación y evaluación de técnicas de optimización automática de imágenes submarinas. Consiste en el análisis de diversas arquitecturas de *deep learning* que, con la configuración y el entrenamiento adecuado, los modelos obtenidos sean capaces de generar una mejora considerable en las imágenes originales.

El objetivo final es el diseño de un modelo de *deep learning* capaz de mejorar la calidad de las imágenes submarinas distorsionadas, obteniendo así resultados uniformes, nítidos y reales.

Palabras clave: análisis de imágenes, entornos submarinos, redes neuronales convolucionales, aprendizaje automático, aprendizaje profundo.

Capítulo 1

Gestión y planificación

1.1. Motivación

El aprendizaje profundo o *deep learning* (DL) en general, y el tratamiento de imágenes mediante estas técnicas en particular, es un campo en expansión desde hace ya unos años. Esto viene motivado por el aumento de la capacidad de procesamiento y la disponibilidad de hardware especializado, la generación masiva de datos e información que sirve como insumo para el aprendizaje, y por las soluciones que algunos algoritmos, con suficiente entrenamiento, son capaces de ofrecer. La combinación de todos estos factores han promovido el crecimiento cada vez más acelerado de este campo, llevando su uso a más sectores como la medicina, industria, investigación, entre otros.

Así pues, el DL, que utiliza algoritmos basados en Redes Neuronales Artificiales o *Artificial Neural Network* (ANN), permite a los modelos de inteligencia artificial el aprendizaje con un nivel de apreciación considerablemente destacados.

Además, el tratamiento de imágenes es un área que permite una interacción con el entorno muy similar a la que tenemos los seres vivos, y en particular los humanos, mediante el sentido de la vista, lo que añade un punto de interés adicional cuando los resultados alcanzan un nivel de significancia tal, siendo capaces de realizar apreciaciones iguales o mejores que nosotros mismos. El abordaje de este problema tiene dos principales motivos. Por un lado la especialización en la utilización de técnicas de procesamiento de imágenes, como una herramienta de interpretación y manipulación desde la óptica del *Machine Learning* (ML); por otro lado, dar las primeras pinceladas a la definición de una posible línea de investigación de cara a una posterior formación doctoral en el mundo de visión por computador.

1.2. Justificación

El proyecto plantea la problemática de la mejora de las imágenes submarinas, tomando como base un dataset ya mejorado y clasificado, que sirve de *input* para que los modelos aprendan a cómo alcanzar el objetivo propuesto. La topología planteada inicialmente es un encoder-decoder convolucional, lo que implica la utilización de técnicas de DL para la codificación y decodificación de las imágenes, siendo estos especialmente útiles a la hora de obtener información relevante de los datos de entrada, a veces, incluso, estructural o semántica.

En consecuencia, el uso de encoder-decoder convolucionales supone una profundización en modelos de ML, que resultan de especial interés como disparador a nuevas propuestas de soluciones análogas. Los encoder-decoder convolucionales tienen, además, la particularidad de poder generar imágenes. Es

decir, un encoder-decoder convolucional es capaz de construir imágenes partiendo de la representación latente aprendida por el modelo.

La mejora de imágenes es un problema sistemático en aquellos campos que basan una parte de su desarrollo en estas, como la medicina o la ciencia. La alta precisión de los recursos fotográficos en estas áreas, supone el éxito o no de pruebas o teorías. La eliminación de las distorsiones evita así las alteraciones generadas por el entorno, así como las imperfecciones inherentes en contextos con baja intensidad lumínica como los ambientes submarinos, incluso también bajo la presencia de partículas visibles en estos últimos. Por lo tanto, alcanzar un modelo capaz de mejorar imágenes submarinas, que es el espacio que nos ocupa en particular en este proyecto, puede contrbuir, en cierta medida, al desarrollo de esta rama.

1.3. Objetivos

1.3.1. Objetivo principal

El objetivo de este trabajo es la creación de un modelo basado en *deep learning*, capaz de mejorar imágenes submarinas.

1.3.2. Objetivos específicos

1. Análisis del estado del arte sobre procesamiento de imágenes mediante DL.
2. Diseño y desarrollo del modelo utilizando un encoder-decoder convolucional.
3. Evaluación cualitativa de los resultados obtenidos mediante el modelo generado y evaluación de su comportamiento.

1.4. Metodología

Para este proyecto y de cara a no destinar tiempo en una gran planificación inicial, se adoptará una metodología *agile* que permite, justamente, una planificación inicial somera, atentiendo a una mayor flexibilidad conforme el proyecto avanza. La herramienta elegida para la gestión del proyecto es Jira, que mediante el uso de panales *kanban* facilita identificar de forma visual el estado de situación en cada momento. El enlace de acceso a este panel se encuentra incorporado a la ficha del proyecto.

Las tareas serán agrupadas por *user stories* que permitirán evaluar el avance incremental, facilitando la dedicación de tiempo en la medida que se desarrollan. Esta metodología es muy útil a la hora de presentar el estado del proyecto de forma visual, atendiendo especialmente a la optimización, eficiencia y productividad en todas las etapas del proceso, asegurando el correcto cumplimiento de plazos y objetivos. Esta metodología de desarrollo ha sido elegida además por la flexibilidad a la hora de entregar las distintas versiones del producto según se avanza en la aplicación, lo que permite un feedback más

fluído en la relación con el tutor. También la comunicación es más informal y constante en metodologías *agile*, afianzando entonces la idea anterior en lo que respecta al feedback.

En cuanto a la metodología propuesta para el core de la *user story* sobre la creación del encoder-decoder convolucional, se propone una metodología *Cross Industry Standard Process for Data Mining* (CRISP-DM), que es un marco de trabajo perfectamente complementario a *kanban*. Mediante CRISP-DM puede asegurarse desde la etapa de planificación del proyecto el correcto abordaje del negocio y la obtención de los datos necesarios.

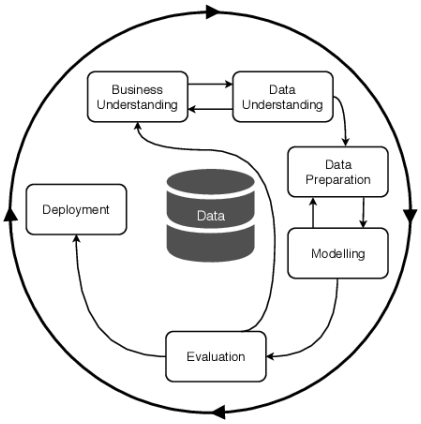


Figura 1.1: Proceso CRISP-DM.

Atendiendo a las seis fases de CRISP-DM, se compone entonces de entendimiento del negocio, entendimiento de los datos, preparación de los datos, modelado, evaluación y despliegue. Esta metodología se centra el la planificación ye ejecución de la minería de datos frente a *kanban* que se enfoca en la ejecución e implementación del proyecto.

1.5. Impacto en sostenibilidad, ético-social y de diversidad

Haciendo un análisis muy superficial sobre el objetivo de este proyecto, salen a la luz algunas apreciaciones clave a la hora de definir su aporte a la sostenibilidad. Por un lado, el uso de dispositivos robóticos en los océanos como fuente de obtención de datos es más confiable en recolección de información que la realizada por los humanos. No solamente por la fiabilidad, estabilidad, precisión y seguridad que estos robots brindan, sino además por la capacidad de alcanzar nuevos límites que para los humanos están lejos aún.

Estos dispositivos permiten el estudio de nuevos ecosistemas alejados de la vida del hombre, como espacios submarinos a grandes profundidades y con un nivel de presión significativa. Poder entender estos nuevos espacios y las poblaciones que en ellos viven, pueden dar pistas de nuevas técnicas de supervivencia, así como de características y elementos que permitan una mejor calidad de vida para la naturaleza en general.

Ciertamente, no solamente son útiles para lugares alejados, sino tabmién para aquellas zonas al alcance

del hombre, como regiones costeras. Su utilidad va desde la cuantificación de poblaciones de fauna hasta la extensión de praderas de algas o flora submarina con una resolución temporal y espacial. La calidad obtenida sería ampliamente mejor que la alcanzada por buzos humanos, cuyos resultados permitirían el establecimiento de correlaciones precisas entre la actividad humana y el crecimiento o decrecimiento de poblaciones o praderas.

La utilización de robots con cámaras incorporadas, favorece a la seguridad de los científicos cuyas líneas de investigación giran entorno a la geología submarina o ecología acuática. Evitar la exposición de las personas mediante el uso de equipamiento autónomo, permite además a estos profesionales centrarse en el análisis de los resultados y obtención de nueva información, sin poner en riesgo su vida o la intervención directa en un contexto que podría, en ocasiones, incluso ser hostil.

El impacto en la biodiversidad y la sostenibilidad es, pues, de un pequeño aporte del campo de la inteligencia artificial a disciplinas cuya vertiente práctica resulta sustancialmente más compleja, puesto que el entorno en el que estas se desarrollan se presentan condiciones no aptas para la vida humana.

1.6. Planificación

1.6.1. Calendario de fases

Fase	Descripción	Inicio	Fin	Semanas
1	Definición y planificación del trabajo final	01/03/2023	12/03/2023	01-02
2	Estado del arte del proyecto	13/03/2023	26/03/2023	03-04
3	Diseño e implementación del trabajo	27/03/2023	28/05/2023	05-13
4	Redacción de la memoria	29/05/2023	25/06/2023	14-17
5	Preparación y defensa pública	26/06/2023	14/07/2023	18-20

Tabla 1.1: Fases del proyecto.

1.6.2. EDT/WBS

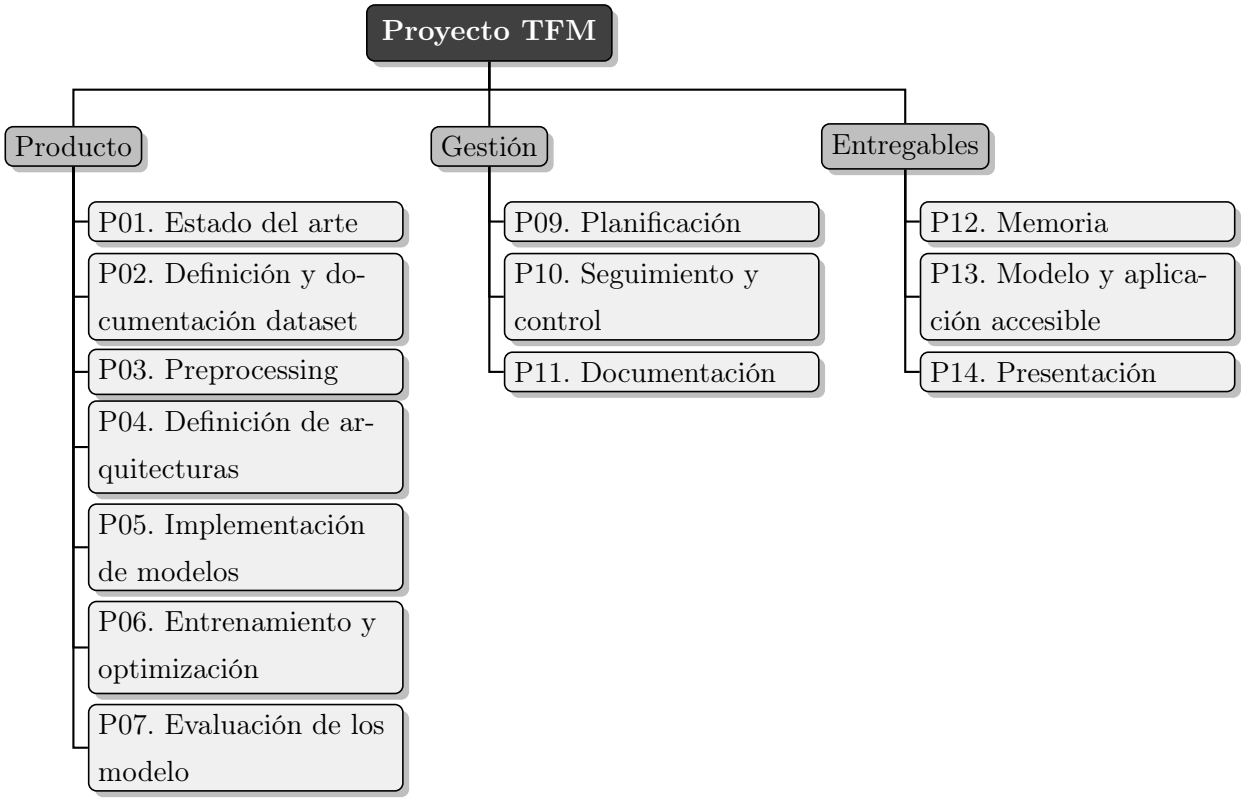


Figura 1.2: EDT/WBS.

1.6.3. Milestones

Se presentan a continuación las fechas de entrega límite de cada uno de los productos y/o versiones, desarrollados en el transcurso del proyecto.

#	Descripción	Deadline	Semana
M1	Entrega primer avance	12/03/2023	02
M2	Entrega segundo avance	26/03/2023	04
M3	Entrega del modelo	28/05/2023	13
M4	Entrega parcial de la memoria	11/06/2023	15
M5	Entrega final de la memoria	25/06/2023	17
M6	Entrega de la presentación	02/07/2023	18
M7	Defensa pública	14/07/2023	20

Tabla 1.2: Milestones.

1.6.4. Desglose en macro-tareas

Fase	Código	Descripción	Semanas
1	F1.P09.A01	Definición del alcance, objetivos y aspectos formales	01
1	F1.P09.A02	Definición del cronograma, EDT, *milestones* y tareas	01-02
1	F1.M1	Entrega de la planificación del proyecto	02
2	F2.P01.A01	Búsqueda bibliográfica	03
2	F2.P01.A02	Definición de la estructura del estado del arte	03
2	F2.P01.A03	Redacción del estado del arte	03-04
2	F2.P01.A04	Verificación del contenido pre-entrega	04
2	F2.M2	Entrega del estado del arte	04
3	F3.P02.A01	Búsqueda y elección de datasets disponibles	05
3	F3.P02.A02	Documentación del dataset final	05
3	F3.P03.A03	Definición del entorno de trabajo	05-06
3	F3.P03.A04	Filtrado del dataset original	06
3	F3.P03.A05	Carga y división del dataset	06
3	F3.P04.A06	Estudio de arquitecturas disponibles	06-07
3	F3.P04.A07	Estudio de viabilidad de *transfer learning*	06-07
3	F3.P04.A08	Definición de dos arquitecturas	07
3	F3.P05.A09	Implementación de modelo I	07-08
3	F3.P05.A10	Implementación de modelo II	07-08
3	F3.P06.A11	Entrenamiento de los modelos	09
3	F3.P06.A12	Ajuste de hiperparámetros	09
3	F3.P07.A13	Evaluación cualitativa y cuantitativa	10-11
3	F3.P07.A14	Exportación de modelo final	11
3	F3.P13.A20	Disponibilización de la app online	13
3	F3.M3	Entrega del modelo y app	13
4	F4.P12.A01	Justificación de modelos elegidos	14
4	F4.P12.A02	Documentación de pre-procesado de datos	14
4	F4.P12.A03	Resultados parciales de los modelos	14-15
4	F4.P12.A04	Análisis comparativo	14-15
4	F4.P12.A05	Ajustes finales primera versión	15
4	F4.M4	Entrega parcial de la memoria	15
4	F4.P12.A06	Corrección de sugerencias	16-17
4	F4.M5	Entrega final de la memoria	17
5	F5.P14.A01	Selección de información prioritaria	18
5	F5.P14.A02	Diseño de presentación	18
5	F5.M6	Entrega de la presentación	18
5	F5.M7	Defensa pública	20

Tabla 1.3: Desglose de tareas en semanas.

1.6.5. Diagrama de Gantt

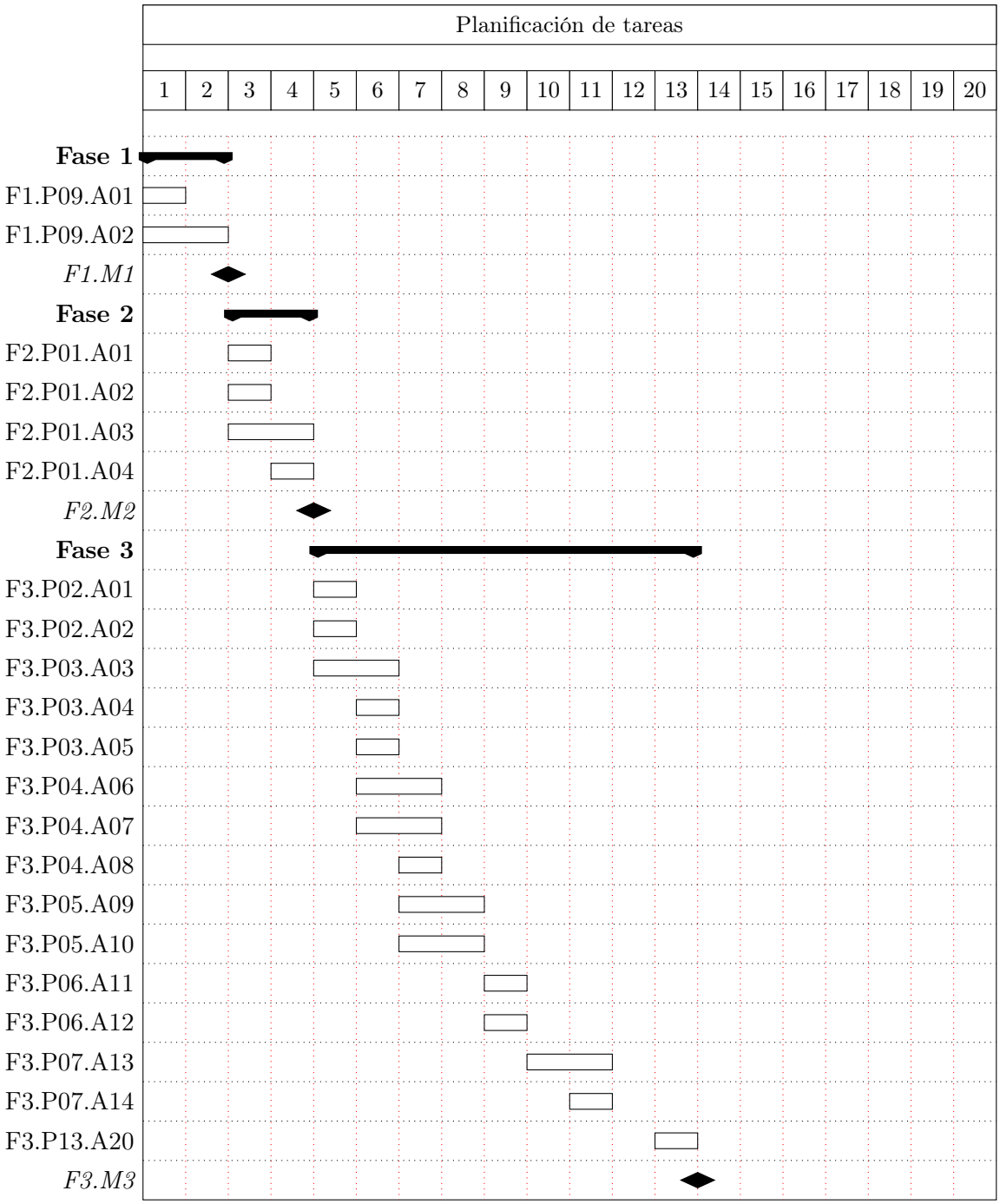


Figura 1.3: Diagrama de Gantt.

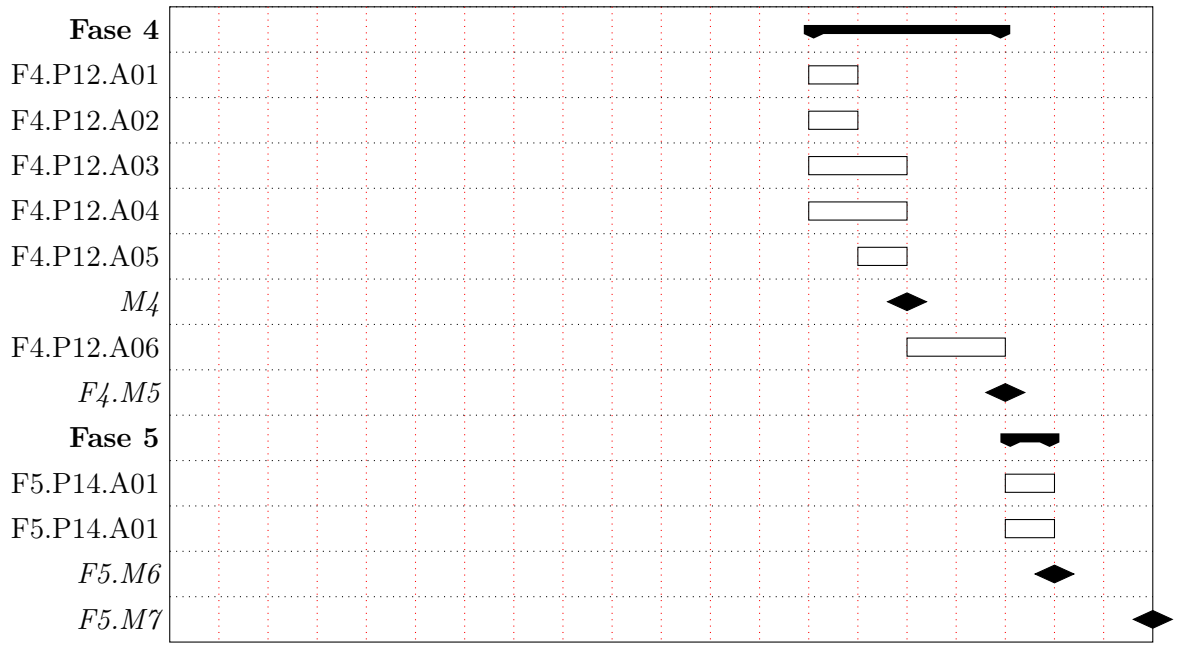


Figura 1.3: Diagrama de Gantt. (continuación de la página 7)

1.6.6. Gestión de riesgos

Tras un análisis a grandes rasgos, se identifican algunos riesgos en los que se puede incurrir durante el transcurso de este proyecto.

1. **Overfitting o underfitting**: conforme avanza la etapa de entrenamiento, el modelo puede caer en un sobreajuste o un subajuste, afectando directamente su capacidad de generalización sobre el *input* que recibe, cuyo resultado no es más que un mal rendimiento del modelo y aún peores resultados de cara a su uso. La solución para no caer en un desequilibrio puede ser, o bien el aumento de la cantidad de datos de entrenamiento (*data augmentation* con los datos que se dispone), o bien la detección anticipada tomando en cuenta las mejoras en el rendimiento parando el enternamiento cuando este se estabilice para evitar que memorice en vez de entrenar.

2. **Escasos datos de entrenamiento**: Este modelo requiere de las suficientes imágenes con vignetting para permitirle entender qué es y cuál es la salida que conforma las expectativas. Si bien hay una cantidad considerable de imágenes que pudieran ser utilizadas, su accesibilidad o los permisos sobre ellas podrían poner en juego la viabilidad del proyecto. Una solución acorde es aplicar técnicas de *data augmentation*, lo que facilitaría la creación de imágenes nuevas a partir de unas pocas, facilitando la tarea del modelo de reconocer distintas posiciones, colores o formas. Esta también es una solución válida para el problema de *overfitting*.

3. **Sesgo**: dado que las redes neuronales, y en especial las convolucionales, dependen en gran medida de los datos para su rendimiento, estos pueden contener cierto sesgo que afecte explícitamente la capacidad de generalización. Para solucionar esto, es importante el balanceo del dataset de tal

forma que se cubra una buena parte de las posibles imágenes que el modelo puede recibir.

4. **Capacidad de cómputo**: el entrenamiento de las redes neuronales suele requerir una gran cantidad de tiempo y de recursos para su entrenamiento. El procesamiento de todas las imágenes de entrenamiento, en función de los parámetros que se definen, afectan directamente al tiempo necesario para tal fin, por lo que contar con hardware suficiente es crucial. Esto plantea varias soluciones posibles, como el uso de *Azure Machine Learning* (AML), *Amazon web services* (AWS), *OVH cloud AI Notebooks* (OVH) entre otros servicios web, así como el uso de GPU local que mejore los tiempos.

5. **Demora en el entrenamiento**: la elección de hiperparámetros es absolutamente importante a la hora de entrenar estos modelos, ya que de ellos depende en buena medida alcanzar un modelo que generalice correctamente, así como del tiempo que llegar a este punto requerirá. El estudio previo de la casuística puede ser una solución que permita afinar estos parámetros iniciales, favoreciendo la forma de entrenar y el tiempo y eficiencia que sea capaz de conseguir.

1.6.7. Tecnologías

A continuación se describen las principales tecnologías utilizadas en este proyecto, eligiendo para esto las últimas versiones estables publicadas.

1.6.7.1. Machine y Deep Learning

1. **Tensorflow**. [v2.11.0] Es una biblioteca *open source* para ML y DL, que permite la creación de modelos complejos de manera sencilla, además de la representación de estos mediante grafos. Dos características fundamentales de esta biblioteca es su escalabilidad y que es multiplataforma. Es muy útil en cuestiones de clasificación de imágenes, *Natural Language Processing* (NLP), reconocimiento de voz, detección de objetos, entre otros.

2. **Keras**. [v2.11.0] Es una biblioteca en Python para *deep learning* de código abierto. Se basa en una API que facilita la construcción de modelos de redes neuronales. El objetivo de Keras es permitir la creación de modelos de DL con una sintaxis de gran usabilidad y fácil de interpretar, característica que lo ha hecho tan popular en este último tiempo. Su compatibilidad con las bibliotecas más usadas de DL permite la explotación de sus funcionalidades mediante el uso de Keras.

3. **Scikit-learn**. [v1.2.2] Es una biblioteca de Python para ML que facilita herramientas para resolver problemas de clasificación, regresión y agrupamiento de datos masivos, cuya sintaxis es flexible y fácil de leer. Posee una ampla gama de algoritmos implementados, como ser *k-Nearest Neighbors* (kNN), redes neuronales, árboles de decisión y/o regresión, entre otros muchos, así como algoritmos de evaluación como cross validation, holdout, etc.

1.6.7.2. Tratamiento de imágenes

1. **OpenCV**. [v4.6.0] Es una biblioteca *open source* de visión por computador para el procesamiento de imágenes, que proporciona herramientas para detección de objetos, extracción de características, seguimiento de objetos, entre otras muchas. Sus principales cualidades que la hacen tan útil en el campo de la investigación, es su capacidad de procesar grandes volúmenes de imágenes a tiempo real.

2. **Scikit-image**. [v0.20.0] Es una biblioteca de procesamiento de imágenes de código abierto para Python. Contiene una gran cantidad de algoritmos para procesamiento de imágenes, como ser filtrado, segmentación, transformaciones geométricas, detección de bordes, análisis de texturas, etc. Mantiene una buena integración con otras bibliotecas como Pandas, Numpy, SciPy, Matplotlib.

1.6.7.3. Representación gráfica

1. **Matplotlib**. [v3.7.1] Es una de las bibliotecas más populares de Python capaz de crear gráficos y visualizaciones de datos. Su flexibilidad permite la creación de muchos tipos distintos de gráficos como ser histogramas, diagramas de dispersión, gráficos de barras, entre otros. Se integra perfectamente con liberías de manejo de datos como Numpy y Pandas, lo que facilita el fliujo de trabajo en el desarrollo.

2. **Seaborn**. [v0.12.2] Es una biblioteca de Python basada en Matplotlib, que facilita la cración de gráficos estadísticos de alto nivel. Es una de las bibliotecas que más se utiliza en ciencia de datos para visualización, generando gráficos de cajas, violines, lineales, barras, dispersión, diagramas de calor, entre otros muchos. Se integra de manera nativa con Pandas, Numpy y matplotlib.

1.6.7.4. Lógica y manipulación de datos

1. **Pandas**. [v1.5.3] Es una biblioteca de código abierto para la manipulación de datos en tablas. Su estructura resultante son los DataFrames, capaces de realizar operaciones básicas como filtrados, agrupaciones, agregaciones, entre otras. Permite además una integración nativa con otras bibliotecas como Numpy.

2. **Numpy**. [v1.24.2] Es una biblioteca de Python para cálculo científico, que facilita de manera considerable el tratamiento de vectores y matrices. Es muy popular en análisis de datos, puesto que facilita la manipulación de grandes cantidades de datos y la aplicación de funciones entre ellos.

3. **Python**. [v3.11.2] Es un lenguaje de programación de alto nivel, interpretado, multiplataforma y *open source*, muy utilizado en el campo de la ciencia de datos por su amplia gama de frameworks y bibliotecas de inteligencia artificial y manipulación de datos. Su sintaxis es simple y cómoda.

Capítulo 2

Estado del arte

2.1. Introducción

Durante siglos, el hombre ha sentido una enorme curiosidad por conocer el mundo que lo rodea. Conforme la ciencia ha ido avanzando, las herramientas se han multiplicado y facilitado este objetivo, contribuyendo significativamente al descubrimiento de nuevos ecosistemas y, dentro de estos, nuevas especies que, hasta ese momento, eran desconocidas. Naturalmente, aquellos lugares cuyo costo de acceso implicaba un desafío, han tardado más en conocerse, dentro de los cuales están los entornos submarinos.

Para la comunidad científica, el acceso al conocimiento que se encuentra bajo el agua presenta una serie de cuestiones asociadas en relación a la seguridad de las personas y la exactitud que estas son capaces de asegurar. El riesgo que supone la inmersión de personas en entornos poco controlados, representa un factor determinante a la hora de avanzar en determinados subcampos de la disciplina. Además, la dificultad no es únicamente del ecosistema sino que también va de la mano de la física, puesto que a medida que se desciende, la presión y la luz juegan un papel protagónico nada menor.

La tecnología ha sido, en este caso, la solución para el estudio de estos sitios muchas veces inaccesibles. La proliferación de recursos tecnólogicos y por consiguiente su accesibilidad en términos de costos, ha permitido el acceso a mejores dispositivos fotográficos, mejor capacidad de procesamiento y un sinfín de soluciones algorítmicas que acercan estos ecosistemas al conocimiento general.

2.2. Vehículos submarinos autónomos

En virtud de lo anterior, son los *Autonomous Underwater Vehicle* (AUV) los nuevos responsables de la exploración submarina. Los AUV son vehículos cuya función es realizar tareas en el océano, como exploración de aguas profundas, cartografía submarina, vigilancia de contaminación, entre otras. Estos dispositivos, como puede verse en el trabajo de Marco et al. [18], están equipados con sistemas de control y navegación autónomos, que le permiten tanto la movilidad como la realización de tareas específicas, y por descontado, la documentación a través de sus sensores, incluídos capturadoras de imágenes y vídeos.

Como ya se ha expuesto previamente, las capacidades de estos equipos han ido creciendo de manera drástica, permitiendo cada vez más tiempo de inmersión y resistencia a las condiciones ambientales [19].

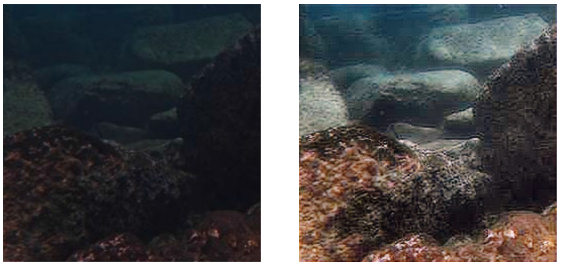
Estos vehículos resuelven varios de los problemas planteados a la vez. Por un lado, se evitan los riesgos que exponen la vida humana en entornos hostiles; también permiten el acceso a sitios cuyas profundidades traerían aparejadas una gran demanda de tecnología capaz de soportar las condiciones del entorno, y sobre todo, realizan un trabajo de recolección de información mucho más exhaustivo, durante más tiempo y con muchas menos limitaciones de lo que podría hacer el ser humano.

2.3. Antecedentes

Puesto que una buena parte de los AUV están equipados con cámaras, los recursos documentales han tomado protagonismo en la investigación submarina. Como resultado de la investigación de Xie et al. [11], la mayoría de los algoritmos hasta hace unos años se centraban en eliminar el ruido, partiendo todos de la premisa que las imágenes eran tomadas en un entorno iluminado. Las imágenes obtenidas en entornos más oscuros presentan más ruido y por tanto menos calidad.

Esto ha devenido en una focalización de la problemática, donde una buena cantidad de proyectos han intentado, y en muchos casos alcanzado, obtener buenos resultados para solventar este problema de la ausencia de luz [11], [17] y [15].

Cabe destacar que la investigación de Xie busca la clarificación de las imágenes subactuáticas con ruido, independientemente de la iluminación del entorno. Para realizar esta tarea, utilizan un dataset de imágenes de baja luminosidad y entrenan un algoritmo basado en DL, cuyos resultados pueden verse en la figura 2.1.



(a) **(b)**

Figura 2.1: Resultados obtenidos por Xie et al.[11]
(a) Imagen original
(b) Imagen procesada por Xie et al.

El modelo de DL utilizado es una *Local Descriptor Sequence Network* (LDS-Net), que se refiere a una red neuronal profunda capaz de emparejar características desde la perspectiva de visión por computador. Este emparejamiento no es otra cosa que la correspondencia entre patrones visuales en diferentes imágenes [20]. Las LDS-Net trabajan en conjunto con las redes neuronales convolucionales o *Convolutional Neural Network* (CNN), o redes neuronales recurrentes o *Recurrent Neural Network* (RNN) con el fin de extraer y codificar características locales en vectores de características.

Los modelos de LDS-Net se pueden utilizar para tareas de mejoramiento de imágenes, ya que son

capaces de aprender patrones de ruido y eliminarlos de las imágenes de entrada (*denoising*). Esta es una herramienta especialmente útil cuando las imágenes son tomadas en entornos oscuros, presentando más ruido que aquellas tomadas en entornos bien iluminados.

Otro trabajo de la misma línea trata justamente sobre la eliminación del efecto *vignetting*, en fotos tomadas con *smartphones* y utilizando filtros que alteran la imagen original [12]. Se basa en el uso de redes neuronales convolucionales y, posteriormente, análisis adicionales para el cálculo de los filtros a aplicar, a modo de obtener un nuevo filtro que aplicado vuelva la imagen a su toma original. En la figura 2.2 se pueden ver los resultados de este trabajo.

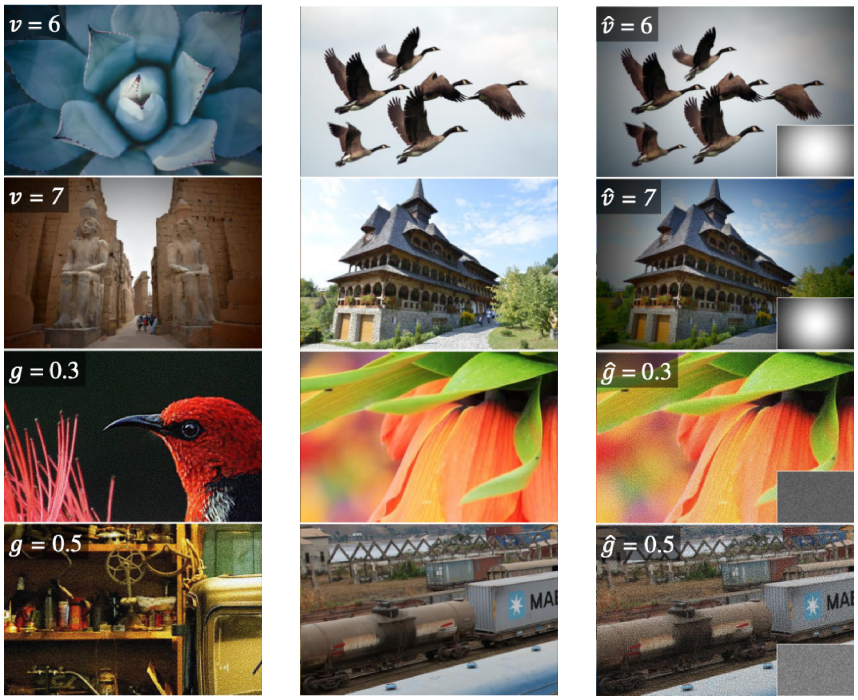


Figura 2.2: Transferencia de filtros por Abdelhamed et al.[12]
(a) Imagen filtrada (*input*)
(b) Imagen de contenido (*input*)
(c) Imagen filtrada (*output*)

El trabajo mencionado busca un filtro que presente una distribución lumínica Gaussiana, centrando la mayor cantidad de luz en el centro de la imagen y, a medida que se aproxima a los bordes, la luz va disminuyendo mediante una distribución normal.

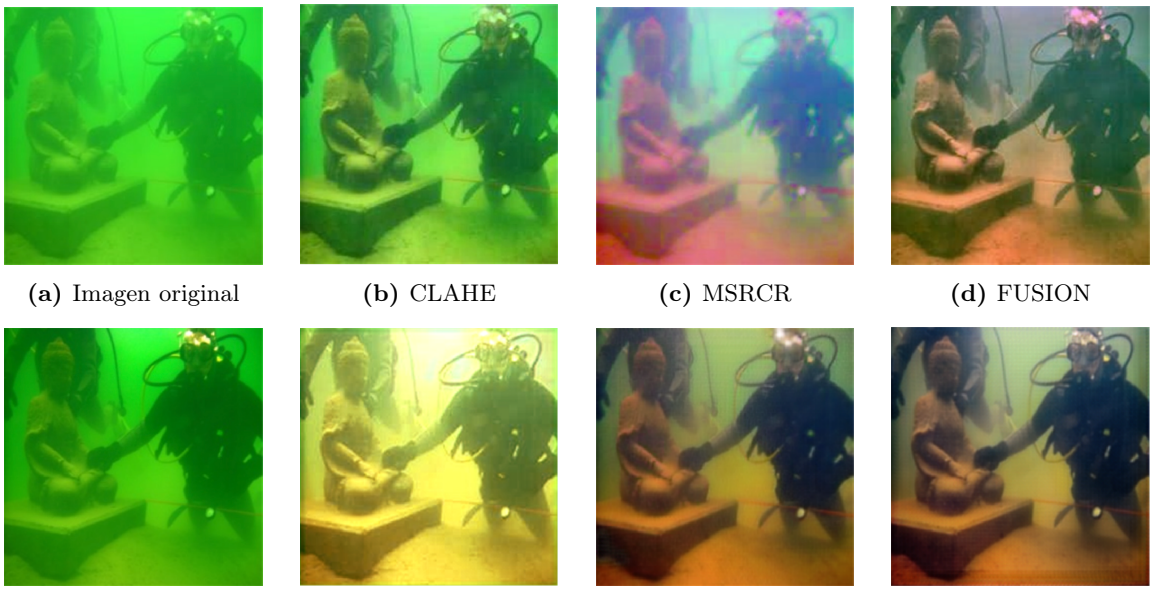
Por su parte, Hu [13] se apoya en los métodos de DL para mejorar la calidad de las imágenes submarinas. De esta forma, encuentra similitudes con respecto a los problemas presentes en las imágenes con *vignetting*, identificando, por ejemplo, la nitidez y claridad a causa del sesgo de color, con la tonalidad típica de las imágenes submarinas.

Este problema dificulta la labor de los AUV que, de poder saltar este inconveniente, serían capaces

de realizar tareas cada vez más complejas con un nivel de precisión sustancialmente mayor. Con esta solución, se reafirma la necesidad de utilizar estos vehículos como intermediador entre el mundo subacuático y la ciencia, recurriendo a las imágenes y vídeos obtenidos mediante estos con el objetivo de mejorar las oportunidades en ese entorno.

Para Hu [13], el desarrollo de los algoritmos de DL es una muy buena solución para el tratamiento de imágenes de cara al reconocimiento y clasificación, pero sobre todo, a la toma de decisiones a tiempo real permitiendo su autonomía. Puede verse cómo ha mejorado la calidad de las imágenes con los distintos tipos de procesamientos que se le aplica a una misma imagen en la figura 2.3.

Los resultados obtenidos, además de satisfactorios, apuntan a una mejora en la adaptabilidad y la robustez de estos dispositivos gracias a estas técnicas aplicadas; a establecimiento de juegos de datos más cohesivos y representativos de la realidad submarina y a una mejora significativa del rendimiento en tiempo real en términos de toma de decisión e investigación en ese contexto por parte de los AUV, a través de los sensores de vídeo. Otros, como describe Akkaynak y Treibitz [17] por ejemplo, utiliza estimadores de retrodispersión, considerando que para recuperar el color original, la refractancia de la luz no incide de manera homogénea en la imágen, sino que depende de la profuncidad de la que se encuentra cada subsección de esta.



(a) Imagen original **(b)** CLAHE **(c)** MSRCR **(d)** FUSION

(e) UDCP **(f)** UWCNN **(g)** UGAN **(h)** FGAN

Figura 2.3: Mejora del color por Hu et al.[13]

2.4. Arquitecturas de *deep learning*

Los proyectos más recientes que tienen por objetivo mejorar la calidad de las imágenes submarinas, utilizan mayoritariamente técnicas de aprendizaje profundo, tanto por los resultados prometedores que estas ofrecen, como por el poder de reconocimiento de patrones que estas son capaces de manejar.

El poder del *deep learning* ha ido mejorando notoriamente, ayudando a la ciencia a expandir sus horizontes un poco más allá. Las capacidades que nos ofrecen estos algoritmos han permitido, por ejemplo, tomar la primera foto de un agujero negro. Se han utilizado técnicas de DL para calcular su diámetro y la asimetría del ancho que lo rodea [27].

En base a todo lo hasta ahora expuesto, es importante detenerse a desarrollar algunos conceptos que son consumidos por las arquitecturas que se exponen en este apartado.

2.4.1. Redes neuronales convolucionales

Uno de los conceptos más destacados es el concepto de *Recurrent Neural Network*, ya que es el modelo de DL en reconocimiento de imágenes por excelencia. Estas redes están compuestas por capas que aplican la operación de convolución a la información que reciben.

Las CNN trata a las imágenes que procesa como una serie de componentes de manera espacial, por lo que es capaz de encontrar patrones a lo largo de todos los datos [25]. Estas se especializan en el reconocimiento de características espaciales y estructurales que lo definen [26]. Posteriormente, cada una de estas irán conectadas con capas de activación, como por ejemplo *softmax*, *sigmoid*, *tanh* [64] y, en los casos en los que el resultado sea una clasificación, todo este bloque de capas convolucionales irá seguido de capas densas o *Fully Connected* (FC), donde las últimas son capaces de encontrar patrones subyacentes en la información procesada.

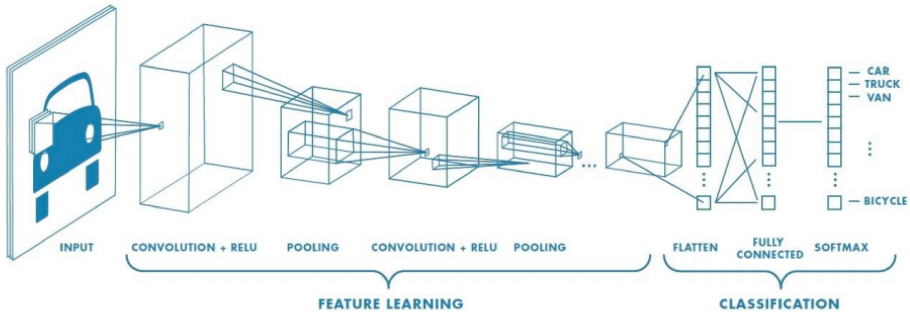


Figura 2.4: Esquema de una CNN [25].

Las redes convolucionales procesan la información y son capaces de reducir su dimensionalidad mediante el proceso de agrupamiento (*Pooling*) o escalado, aplicado a subregiones no superpuestas, cuyo objetivo es, además del ya mencionado, hacer suposiciones sobre las características agrupadas. Esta reducción favorece el tiempo de procesamiento a raíz de la reducción del número de parámetros a manejar.

2.4.2. Redes generativas adversarias

Dentro de las arquitecturas de *deep learning* más utilizadas, se encuentran las redes generativas adversarias o *Generative Adversarial Network* (GAN). Las GAN son modelos de inteligencia artificial,

compuestos por dos redes neuronales profundas, donde la generadora y la discriminadora compiten entre sí durante la etapa de entrenamiento [21].

El *feedback* de la red discriminadora es la entrada necesario para que la red generadora mejore su salida, aproximándose más a los casos reales con los que ha entrenado la primera red. A medida que la red generadora aproxima más el resultado a los esperados por la discriminadora, la pérdida de esta última comienza a disminuir, hasta el punto que la primera ya no es capaz de distinguir entre una imagen generada de una imagen real [24].

Estas redes tienen la peculiaridad de ser altamente inestables [22] y [24], dada la complejidad de la estructura y modelo. Esta inestabilidad viene dada por los cambios bruscos que puede tener una de las redes, debiendo la otra adaptarse a este cambio alterando sus pesos rápidamente, y esa inestabilidad se mantiene en el tiempo incluso incrementándose. Otro de los problemas más releventes es el *overfitting* en la red discriminadora, incapaz de reconocer una imagen que no sea una completamente real y clasificando como falsa aquellas generadas sintéticamente. Otros problemas comunes son la desaparición y explosión del gradiente así como también el problema de parada.

A través de la figura 2.5 se puede apreciar cómo se comportan las redes GAN, buscando generar una figura capaz de 'engañar' a la red discriminadora, partiendo de imágenes con ruido.

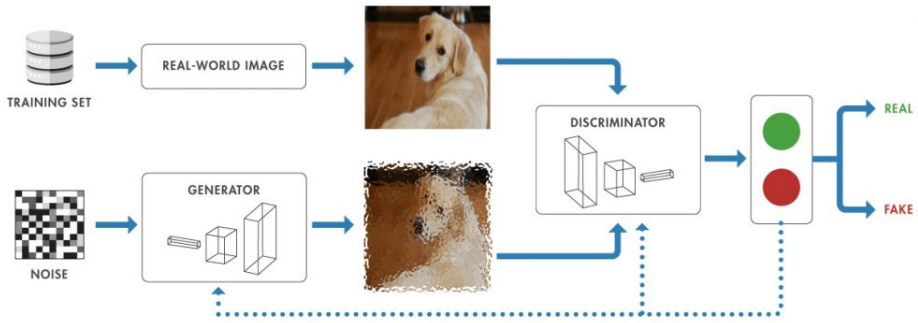


Figura 2.5: Cómo trabaja una red generativa adversaria (GAN) [23]

2.4.3. Redes convolucionales encoder-decoder

Otros modelos bastante utilizados son las redes neuronales convolucionales con una arquitectura *encoder-decoder*. Esta topología es muy utilizada en procesamiento de imágenes, consta de dos redes neuronales, una codificadora y una decodificadora, siendo al menos una de ellas una red neuronal convolucional. La primera comprime la imagen de entrada a su representación de características más reducida, que será la entrada del la segunda.

La red *encoder* utiliza capas convolucionales para la extracción de características, buscando la compresión de la información mediante convolución y *pooling*, mientras que la red *decoder* utiliza capas convolucionales transpuestas y, además, capas de *UpSampling* para la reconstrucción de la imagen final aumentando su tamaño. Su utilidad puede verse en la segmentación de imágenes como en el estudio

de Badrinarayanan et al. [28].

Una característica importante en este tipo de arquitecturas es el uso de técnicas de regularización como normalización por lotes o regularización L2, que permiten prevenir el *overfitting*, mejorando así la adaptación de la red a nuevos casos que aún no conoce. Además, son muy útiles en visión por computador por sus excelentes resultados a la hora de segmentar imágenes, eliminar el ruido y restaurar imágenes. Este último punto es fundamental para este proyecto.

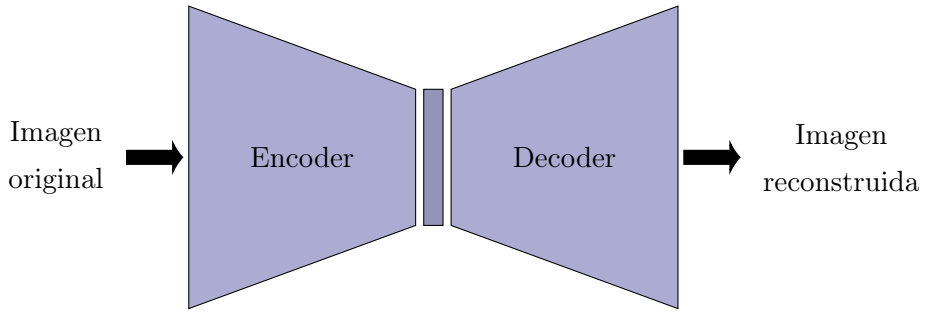


Figura 2.6: Estructura de una red encoder-decoder

Algunos proyectos reales que se han realizado utilizando esta arquitectura de DL, pueden ser la búsqueda de un modelo de traducción automática partiendo de redes neuronales de tipo *encoder-decoder* [30]; segmentación de imágenes [28], detección de anomalías para multi-sensores [31] y segmentación semántica de imágens mediante convolución [32].

2.4.4. Autoencoders convolucionales

Dentro de los modelos de *deep learning* no supervisado, los *Convolutional Autoencoder* (CAE) tienen por objetivo el aprendizaje de una imagen en una representación comprimida partiendo de los datos de la imagen original como entrada [33]. La salida de esta arquitectura, como su nombre lo indica, es la misma imagen partiendo, sin embargo, la verdadera importancia de estas está en la representación comprimida o espacio latente.

Un CAE puede considerarse un caso especial de la arquitectura *Encoder-Decoder* (ED), que utiliza capas convolucionales en lugar de capas *Fully Connected* tanto en el codificador como en el decodificador. Esta arquitectura le permite al modelo la captura de características desde la entrada, capaz de reproducirlas en la reconstrucción de la imagen de salida. En la imagen del estudio de Yasrab [29] se ve claramente la estructura.

Los modelos de CAE sueles ser más complejos que las redes ED, puesto que esta arquitectura en específico está compuesta por varias capas convolucionales y de muestreo, aumentando considerablemente la complejidad de la red [35].

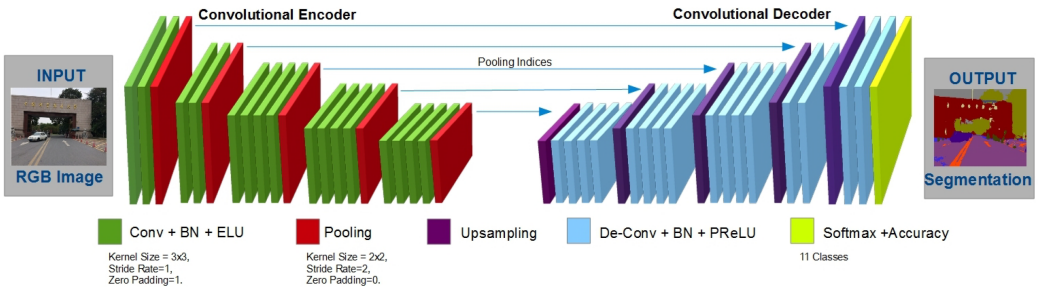


Figura 2.7: Estructura de autoencoder convolucional.
Fuente del modelo detallado [34] y [29]

2.5. Conclusiones

El *deep learning*, como se ha podido observar previamente, es una poderosa herramienta para la predicción, la clasificación y también la generación. Las redes neuronales profundas han llevado estas capacidades a nuevos niveles y la apuesta por ellas está siendo cada vez mayor. La diversidad de arquitecturas ha aflorado conforme ha ido afianzándose esta tecnología, dando paso a nuevos modelos que mejoras algunos aspectos, centrándose en la detección de nuevos patrones o comportamientos, o incluso generando una salida completamente nueva.

Los modelos de tipo *encoder-decoder*, como bien se ha indicado *ut supra*, son muy buenos a la hora del tratamieto de imágenes, eliminando ruido y restaurándolas, por lo que es una de las arquitecturas que se utilizará para este proyecto.

Ahora bien, en función de todo lo expuesto hasta aquí, podría ser posible la optimización de las imágene submarinas mediante alguna de estas arquitecturas, puesto que existen estudios previos que han conseguido mejorarlas de forma significativa y, además, otros como [12], proponen una solución extrayendo el *vignetting* como un filtro, invirtiendo los valores de este último para generar la imagen limpia.

La diversidad de causas por las que las imágenes se ven distorsionadas representan un inconveniente a la hora de su tratamiento. Sin embargo, las redes de tipo *encoder-decoder* son una arquitectura que ajusta perfectamente a las dimensiones del problema que se busca resolver. Su entrenamiendo permite el aprendizaje de estas redes sobre la relación entre la imágen distorsionada y una salida corregida. El entrenamiento de este modelo puede ser exhaustivo puesto que requiere de una gran cantidad de imágenes, con y sin distorsión.

Además, como alternativa, se evaluará la utilización de *transfer learning* con redes neuronales preentrenadas que faciliten el mejoramiento de las imágenes.

En conclusión, el estado del arte presentado sugiere la viabilidad en la optimización de imágenes submarinas, utilizando distintas arquitecturas convolucionales para la solución del problema planteado.

Capítulo 3

Procesamiento de datos

3.1. Sobre el *dataset*

Puesto que el objetivo de este proyecto es mejorar la calidad de imágenes submarinas, el *dataset* de datos de entrenamiento, validación y test será, naturalmente, un juego de datos de imágenes. Existen para esto varias opciones disponibles cuyo licenciamiento permite su uso bajo una diversidad de licencias abiertas. Sin embargo, para este proyecto particular, se utiliza *The EUPV dataset* [36], creado y disponibilizado por la Universidad de Minnesota, Estados Unidos.

El *dataset* está compuesto por tres grupos de imágenes submarinas. Estos tres grupos se denominan *underwater_dark*, que contiene imágenes parcialmente oscuras y con el tinte azul característico del entorno submarino; *underwater_scenes*, cuyas imágenes son tomadas en entornos más cerca de la superficie y con una variante de color verde; y *underwater_imagenet* que han sido imágenes distorsionadas utilizando modelos generativos.

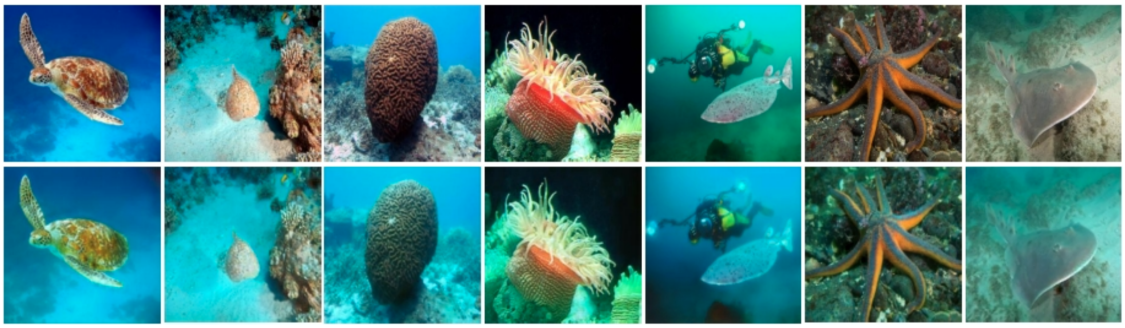


Figura 3.1: Ejemplos de pares de imágenes, debajo la original y arriba su ground truth. [36].

Este dataset contiene suficientes imágenes para el entrenamiento del modelo. Dentro de las ventajas que ofrece está la paridad en las imágenes, por lo que, a los efectos de la cardinalidad, se tiene el doble de imágenes iniciales. En otras palabas, cada imagen para el entrenamiento tiene su *ground truth* que permitirá al modelo mejorar según este último. Por tanto, en realidad se tiene el doble de imágenes en el dataset completo. En la tabla 3.1, puede verse el número total de elementos con lo que se cuenta originalmente de cara al entrenamiento.

Este *EUVP dataset* original tiene, además, un total de 6665 imágenes sin emparejar, de baja y buena calidad, que pueden ser de utilidad a la hora de probar los resultados del modelo final. Su utilidad será, en tal caso, una vía de evaluación visual de los resultados de los modelos entrenados, pero sin ninguna

capacidad de poder medir este en términos de métricas. El contenido del juego de datos está accesible para su descarga en Google Drive mediante este enlace.

Nombre del dataset	Entrenamiento	Validación	Total
Underwater Dark	5550	570	11670
Underwater ImageNet	3700	1270	8670
Underwater Scenes	2185	130	4500
Total	11435	1970	24840

Tabla 3.1: Distribución original de imágenes del dataset.

Por otro lado, como una forma de validar el producto final con datos de otra fuente, se cuenta con un dataset suficientemente grande y distinto llamado *The AQUALOC Dataset*[37]. El mismo es accesible mediante este enlace, pudiendo descargarse casi una centena de *gigabytes* de imágenes submarinas en dos entornos, bahías y enclaves arqueológicos. De este último *dataset*, se toman solamente un subconjunto de imágenes, que permitan la evaluación del modelo con casos significativamente distintos a los que ha entrenado inicialmente, visualizando de esta forma la capacidad de generalización del modelo.

Este *dataset* tiene la particularidad que las imágenes que lo componen están en escala de grises, lo que permite evaluar el comportamiento de los modelos, no solamente en términos de calidad y nitidez, sino también en cómo se comporta a la hora de predecir el color. También se recurre a imágenes extraídas de la web que no pertecezcan al *EUVP dataset* original, para evitar posibles patrones subyacentes en estas. Los resultados pueden accederse en el anexo B.

3.2. Construcción del dataset

El modelo que se busca debe ser capaz de mejorar imágenes en base a las distintas distorsiones que pueden presentarse en entornos submarinos. Por eso, partiendo de este juego de datos dividido en tres, se aunan en uno único, compuesto por las imágenes de todos ellos, facilitando así al modelo entrenar y reconocer patrones de falta de luz, cambios de frecuencia de color, efecto *vignetting*, entre otros.

Con el objetivo de construir un único juego de imágenes, se unificarán los tres *datasets* [36] cuyos archivos se encuentran en tres grupos distintos. A su vez, estos están subdivididos en dos, según si su contenido son imágenes distorsionadas o no -*trainA* y *trainB*, respectivamente-. El directorio *trainA* contiene las imágenes originales mientras que *trainB* sus *ground truths*.

Además, la nomenclatura de los archivos no siguen un patrón de nombres, por lo que la probabilidad de tener ficheros repetidos y generar errores de validación es alta, ya que estos están en distintos directorios y, por tanto, libre de la restricción que impide duplicados. Cada uno de los archivos de los grupos de imágenes distorsionadas, debería tener su homónimo en el directorio de imágenes mejoradas, aplicando así la regla de existencia en ambos sitios para considerar un caso proclive al entrenamiento.

Esta copia de los ficheros que componen el nuevo dataset emparejado conlleva además renombrar los archivos de forma numérica incremental para evitar superposiciones.

```
dataset_euvp/
├── Paired/
│   ├── underwater_dark/
│   │   ├── trainA/
│   │   └── trainB/
│   ├── underwater_imagenet/
│   │   ├── trainA/
│   │   └── trainB/
│   └── underwater_scenes/
└── Unpaired/
```

```
dataset/
├── test/
│   ├── enhanced/
│   └── original/
└── train/
    ├── enhanced/
    └── original/
```

Tabla 3.2: Estructuras origen y objetivo del dataset

La estructura de directorios también se modifica (tabla 3.2), puesto que se busca que el modelo aprenda sobre datos de los tres *datasets*. La proporcion final para *train* y *test* es de 90-10, considerando mediante el entrenamiento que el 20 % correspondiente a *validation* lo toma del porcentaje correspondiente al primero. De cara entonces a la representatividad de estos casos en todo el proceso, se separa la parte proporcional de cada uno de ellos copiando los fichero de sus directorios originales a la estructura que se presenta a continuación.

Para realizar este proceso, primero se valida al estructura de directorios de destino y, de no existir se crean y de contener ficheros, se eliminan. Una vez realizado esto, se distribuyen los nombres de los ficheros de cada *dataset* en una relación de 90-10 para *train* y *test*, tomando como referencia el directorio de imágenes distorsionadas. Finalmente, para cada uno de estos ficheros y sabiendo a qué *dataset* objetivo corresponde, se valida que exista su contraparte mejorada y, de estarlo, se realiza la copia del fichero.

Los aportes que cada *dataset* realiza a la conformación del dataset original se pueden ver en la tabla 3.3, así como el total de imágenes que componen el *dataset* con el que finalmente se entrena y prueba en la tabla 3.4.

Nombre del dataset	Destino	Aporte
Underwater Dark	train	4995
Underwater Dark	test	555
Underwater ImageNet	train	3300
Underwater ImageNet	test	370
Underwater Scenes	train	1966
Underwater Scenes	test	219
Total		11405

Tabla 3.3: Aportes y distribución por dataset.

Dataset	Clase	Total
train	original	10290
train	enhanced	10290
test	original	1147
test	enhanced	1147
Total		22874

Tabla 3.4: Cardinalidad de cada dataset procesado.

De este proceso se tiene entonces que, del *dataset* original, se purga poco menos del 8 % del total de imágenes que no tienen su par mejorado identificado, por lo que no nos son útiles de cara al entrenamiento de los modelos. En la figura 3.2 se ilustran un par de imágenes, cuya versión original presenta ruido y pérdida de frecuencia de color, mientras que la versión mejorada gana en nitidez y precisión.

El resultado de este proceso de limpieza, organización y distribución está reflejado en el Anexo A.1, donde se brinda el código relativo a la obtención de los datos y su estructura.

(a) Imagen *original* **(b)** Imagen *enhanced*

Figura 3.2: *Ejemplo de par de imágenes del dataset EUVP* [36]

3.3. Sobre la evaluación cualitativa

Como se ha comentado al comienzo de este capítulo, el juego de datos elegido para el entrenamiento de los modelos es *EUVP dataset*. Se ha reservado un 10 % del total de imágenes para evaluar los modelos generados y obtener las métricas de cada uno. Sin embargo, estos datos provienen de la misma fuente, lo que sugiere pudiera tener características similares a las utilizadas para el entrenamiento. Al tener sus respectivos *ground truths*, es posible obtener las métricas de error y similitud que permiten medir el modelo en términos cuantitativos.

Ahora bien, estas no son las imágenes que el modelo busca mejorar, sino otras cualesquiera donde no se conoce su respectiva mejora, por lo que aquí no hay métrica posible. Por eso, se utilizan imágenes

extraídas de la web, que presenten alguna distorsión y otras que no, para estudiar cualitativamente el comportamiento de los modelos con imágenes que realmente nunca ha visto.

Tal como se ha comentado previamente, también se cuenta con un segundo conjunto de datos que presenta una serie de características particulares, llamado *The AQUALOC Dataset* [37]. Su contenido son varios *gigabytes* de imágenes de bahías y parques arqueológicos submarinos. Algunas de sus particularidades son que muchas de estas imágenes presentan *vignetting*, lo que genera problemas de distorsión en la iluminación, además de problemas de nitidez. También es importante destacar que, este conjunto de imágenes están en escala de grises, lo que no se adapta exactamente al formato de entrada de los modelos entrenados. Dicho esto, se tratarán como un caso particular para estudiar el comportamiento de los modelos cuando el *input* no está previamente coloreado.

Este *dataset* no estará presente en el proceso de entrenamiento, por lo que el modelo no tendrá conocimiento de estas imágenes y, por lo tanto, permitirá medir qué tanta capacidad de generalización se es capaz de alcanzar con el entrenamiento previo. En la figura 3.3 se muestran algunas imágenes tomadas del *dataset* [37]. La evaluación cualitativa que se realiza con este juego de datos, permitirá la evaluación del modelo de cara a la mejora de la imagen, así como también, un primer intento de coloración aunque este no sea su objetivo primario.

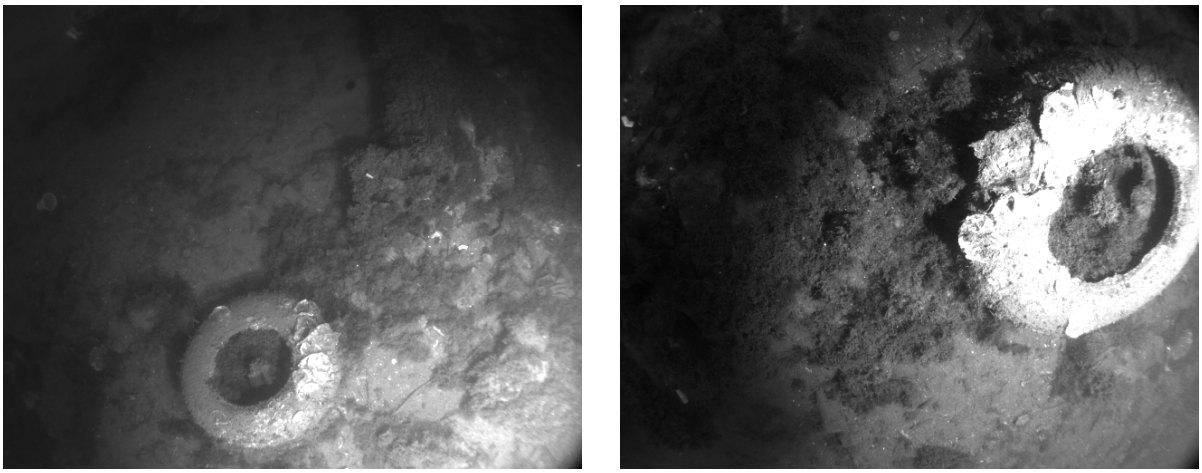


Figura 3.3: *Ejemplo de imágenes del dataset AQUALOC* [37]

Capítulo 4

Definición de la arquitectura

4.1. Características

En los últimos años, las arquitecturas basadas en *Convolutional Neural Network* han demostrado su eficiencia en una gran cantidad de tareas relacionadas con *computer vision* (CV), como la clasificación, segmentación y generación de imágenes, así como la detección de objetos presentes. Este auge está basado en la capacidad de las CNN para aprender de manera automática características relevantes de las imágenes mediante la operación de convolución, reduciendo la entrada y recordando patrones a partir de estas imágenes [58].

Además, las CNN pueden procesar eficientemente grandes volúmenes de datos [59], lo que las hace ideales para aplicaciones en tiempo real y para el procesamiento en paralelo en sistemas fuertemente distribuidos. Dada su efectividad y eficiencia, es altamente probable que las arquitecturas basadas en CNN se utilicen cada vez más en una amplia gama de aplicaciones de visión por computador de cara al futuro.

Tal y como se ha comentado previamente, dentro de las opciones disponibles, la arquitectura seleccionada para este modelo será un *Encoder-Decoder* convolucional cuya entrada son las imágenes del *dataset* EUVP procesadas y ordenadas.

Poniendo en contexto la arquitectura ED, estamos frente a un tipo de red neuronal muy buena en el procesamiento de imágenes, subdividida en dos partes. Por un lado un *encoder* y por otro un *decoder*.

El *encoder* es la red neuronal convolucional responsable de extraer mapas de características de la entrada [60]. Esto se consigue mediante la aplicación de una serie de capas convolucionales, con el fin de reducir de dimensionalidad del problema a una representación latente más pequeña. Estas variables latentes presentan una menor dimensión representativa que la real maximizando la retención de información útil. De modo que si el modelo la comprime, puede esperarse que estas nuevas variables contengan información esencial y distintiva de los datos de origen [70].

Esto tiene asociados una serie de beneficios, entre los que pueden destacarse las que se enumeran a continuación.

1. *Eficiencia computacional*, permitiendo así una disminución del tiempo de computo, puesto que a menos neuronas, menos operaciones y por lo tanto, menos tiempo [61].
2. *Extracción de características*, forzando a la red a que aprenda las características más distintivas de los datos.

El *decoder* por su parte, es la segunda red neuronal utilizada para la generación de la salida a partir de las características extraídas por el *encoder*. A diferencia de la red que le precede, esta expande la resolución de la imagen mejorando su precisión [63]. Esta arquitectura suele mantener una estructura de capas simétrica al *encoder*, tal y como se ha utilizado para este trabajo.

Otra de las principales características de esta arquitectura, es la mejora de las imágenes en la entrada, permitiendo eliminar ruido y devolviendo imágenes con mejor resultado visual [39].

Ahora bien, estas dos estructuras se fusionan en una única arquitectura que, como se ha dicho antes, es el *Encoder-Decoder*. La unión de ellas se da en una capa de *puente*, donde puede incluir en ocasiones una capa de *pooling*. Esta capa permite, entre otras cosas, aumentar la invariancia a la traslación y deformación de las características, asegurando la detección de estas incluso de haber alguna pequeña variación de movimiento o deformación.

4.2. Filtros convolucionales

En arquitecturas como *Encoder-Decoder* se utilizan filtros convolucionales. Es una técnica bastante común en el procesamiento de imágenes, siendo esta una de las capas fundamentales en la composición de la arquitectura de estas redes. Los filtros convolucionales son operaciones matemáticas que se aplican sobre las imágenes utilizando un *kernel* para realizar la operación de convolución.

Los *kernels* son matrices de pesos [66] utilizadas para procesar el filtro [26], aplicándose de forma deslizante a toda la imagen por pequeñas regiones. Estos filtros permiten la extracción de características más destacadas en cada una de estas regiones activas, es decir, sirven para la detección de patrones.

El desplazamiento de un *kernel* viene establecido por el *stride* o desplazamiento, que es el número de pixeles que se salta entre los píxeles centrales del *kernel* a la hora de aplicar la convolución. Dicho de otra forma, es la distancia entre los centros de dos *kernels*. Es un control a la cantidad de píxeles que se evitarán en cada paso de este proceso. Un *stride* de 1, implica una salida del mismo tamaño que la entrada (siempre que se considere el *padding* adecuado); por el contrario un *stride* mayor a 1 implica una reducción en el tamaño de la salida, entendiéndose como un submuestreo. El *stride* permite ajustar la cantidad de información extraída en la capa de convolución.

Los modelos implementados inicialmente se definen por la utilización de una variedad de filtros que pueden verse en la tabla 4.1. El tamaño del *kernel* está fijado en 3x3 para todos ellos así como el *stride* en 2. Estos valores son tomados como referencia a partir de [67] y [26], en el entendido que con ese tamaño se gana en eficiencia computacional, puesto que solo se consideran los pixeles que rodean al centro. El tamaño es lo suficientemente pequeño para capturar patrones simples, y el uso de muchos de ellos facilita la captura de una gran variedad de características distintas, frente a tamaños más grandes que capturan una única de estas.

Para cada uno de los valores de la tupla que se envía al generador de modelos, se crean en el *encoder* tantas capas convolucionales como elementos tenga la lista de filtros. Asimismo, estos filtros son

duplicados en sentido inverso en el *decoder*, por lo que se tienen el doble de capas convolucionales.

Los filtros convolucionales son utilizados, como ya se ha comentado, para la extracción de características. En el caso de una arquitectura ED, cada filtro convolucional es entrenado para la detección de una característica específica. Estas características son formas, texturas, bordes o patrones de las imágenes de entrada. La actualización de los pesos en los modelos generados se realiza mediante *Adam* [71], una combinación entre el algoritmo de *back-propagation* y optimización de gradiente descendente, que permite maximizar la precisión de la reconstrucción.

Cada un de las filas de la tabla 4.1 refleja un modelo distinto de la misma arquitectura con diferentes capacidades, pues la variación de filtros impacta en la forma de aprendizaje y la atención específica al reconocimiento de patrones.

#	Filtro	Kernel	Capas conv.
1	[32, 64, 128, 256]	3x3	6
2	[32, 64, 128]	3x3	6
3	[256, 128, 32]	3x3	6
4	[256, 128, 256]	3x3	6
5	[256, 256, 64]	3x3	6
6	[128, 8, 4]	3x3	6
7	[128,128,16]	3x3	6
8	[128,4]	3x3	4
9	[128,8]	3x3	4
10	[128,128,32]	3x3	6
11	[128,16]	3x3	4
12	[128,128,64]	3x3	6
13	[128,32]	3x3	4

Tabla 4.1: Filtros utilizados en los *Encoder-Decoder*.

A modo de ejemplo, tenemos entonces que el primer modelo es una arquitectura *Encoder-Decoder* con cuatro capas convolucionales en el *Encoder* con 32, 64, 128 y 256 filtros respectivamente, manteniendo el tamaño del *kernel* y el *sparse* invariantes, tal como ya se ha comentado, en 3x3 y 2 respectivamente. Esto es, que la primera capa obtendrá como resultado 32 mapas de características diferentes, la segunda 64, la tercera 128 y la cuarta y última 256.

El *Decoder* por su parte, tendrá exactamente la misma cantidad de capas, pero los filtros serán simétricos, por lo que se establece una primera capa con 256, una segunda con 128, una tercera con 64 y una última de 32 filtros.

Así pues, con el objetivo de visualizar de forma clara la arquitectura de cada una de estas configuraciones, se hace uso de la biblioteca *visualkeras* [68]. Además, a través de la función *plotModel* definida

en el anexo A.8, se permite la visualización de la información de cada una de sus capas.

Las figuras 4.1 y 4.2 presentan de una manera visual la segunda configuración de la arquitectura del *Encoder-Decoder*, para los tres filtros convolucionales, 32, 64 y 128.

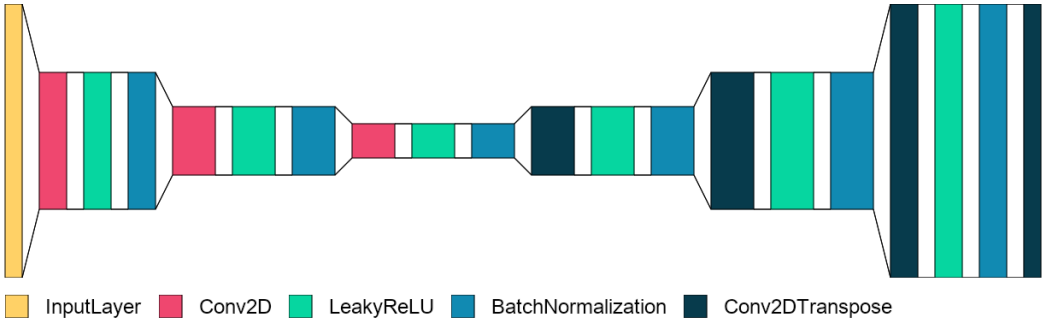


Figura 4.1: Arquitectura de capas en 2D.

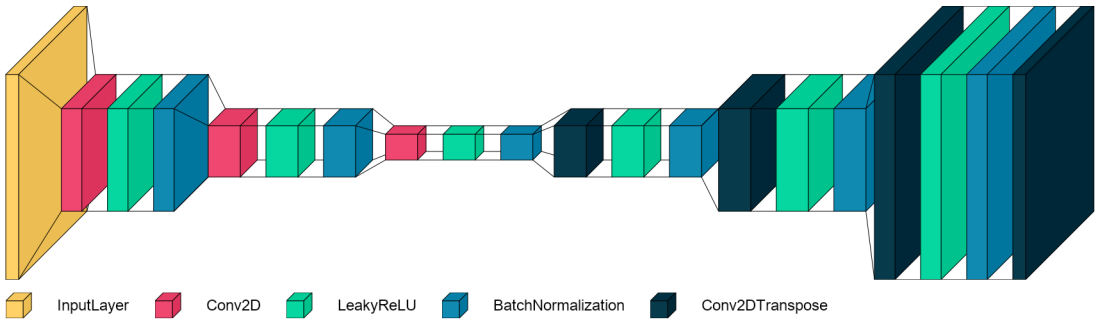


Figura 4.2: Arquitectura de capas en 3D.

Una vez vista gráficamente la arquitectura utilizada, es posible, gracias a la biblioteca cuya función homónima *plot_model* en el mismo anexo, generar los detalles de las capas de esta arquitectura. En la figura 4.3 puede apreciarse esta distribución de manera más informativa, ya que permite ver la evolución en términos de dimensiones y funciones.

La configuración de la figura 4.3, que por cuestiones espaciales se ha dividido en dos columas, comienza con la capa de entrada cuya entrada son imágenes de 320x320 en 3 canales. Seguidamente, por cada capa convolcuional creada, como su nombre lo indica, se realiza la aplicación de filtros mediante la operación de convolución, los siguen una capa de activación *LeakyRELU* para introducir no linearidad en el modelo.

Luego de sus capas de activación, se utilizan capas *BatchNormalization* que ayudan a acelerar la convergencia del modelo, estabilizar el proceso de entrenamiento y mejorar el rendimiento general del *Encoder-Decoder*. Ayudan también en la reducción del desvanecimiento o explosión del gradiente, normalizando las activaciones y manteniendo los rangos adecuados.

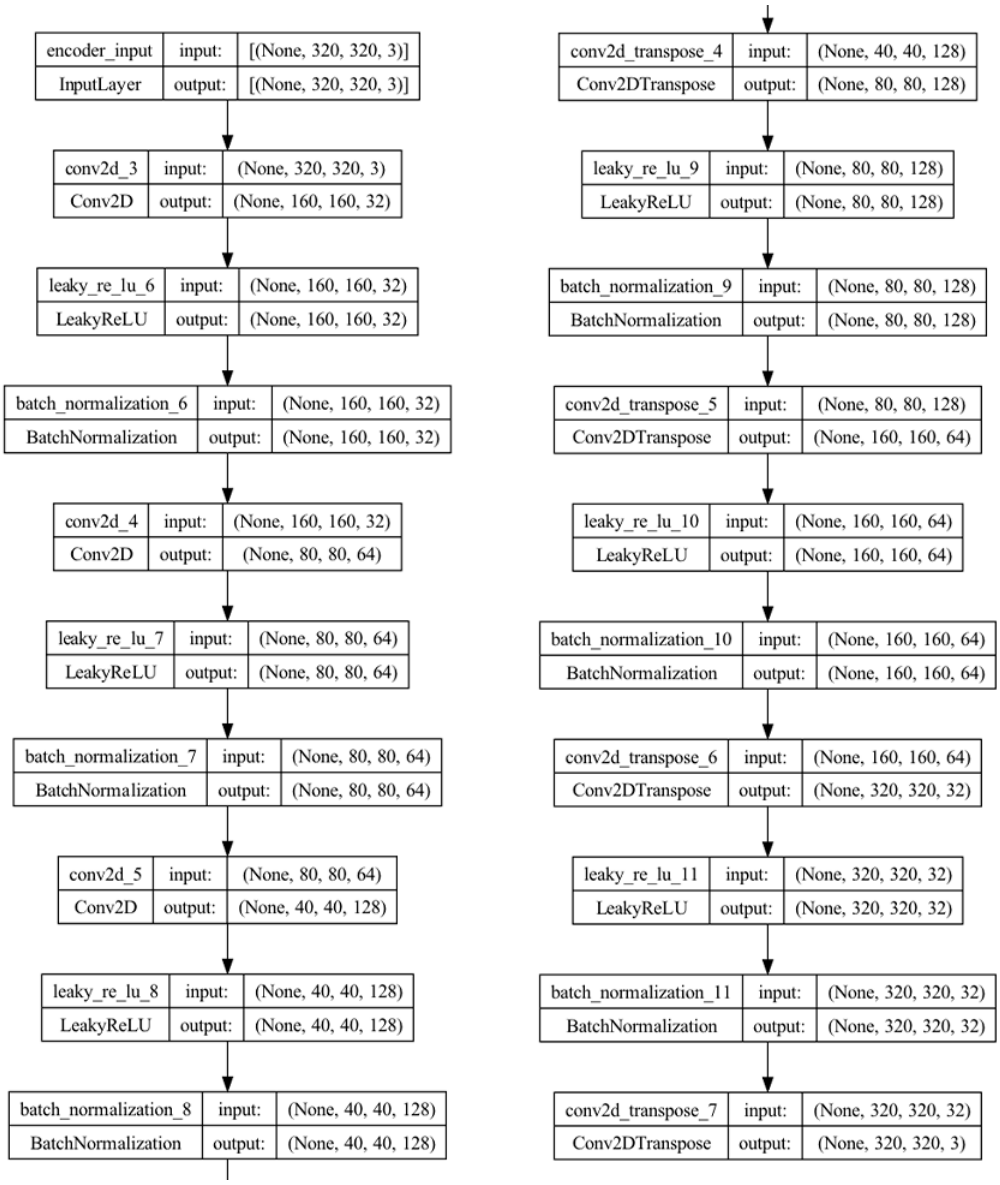


Figura 4.3: Detalles de las capas de la arquitectura.

Las capas del *encoder* pueden verse entonces en la medida que se sigue el patrón *Conv2D - LeakyReLU - BatchNormalization*. Ahora bien, las capas del *decoder* están definidas por el mismo orden pero, en lugar de capas convolucionales (*Conv2D*), utiliza capas de convolución transpuesta o *Conv2DTranspose*. Este tipo de capas aplican operaciones de convolución inversa para aumentar la resolución espacial de características.

Estas capas permiten la reconstrucción de imágenes y facilitan una salida de mayor tamaño, ideales en el proceso de decodificar los datos de cara a extender el espacio latente. En la figura 4.3, la primer columna refiere al *encoder*, mientras que la segunda refiere al *decoder*. En esta misma figura, puede verse la aplicación de los filtros convolucionales en el *output* de las capas convolucionales del *encoder*, así como en el orden inverso en las capas de convolución transpuestas en el *decoder*.

4.3. Callbacks

De cara al entrenamiento de los modelos y consecuentes análisis, los *callbacks* son herramientas, aunque prescindibles, muy útiles que facilitan estas tareas. El concepto de *callback* es ajeno a las redes neuronales, cuya idea es la de una función que se comunica con otra bajo una determinada acción. Dicho de otra forma, son funciones lanzadas por otras, bajo ciertas circunstancias. Es muy común este concepto en entornos como *JavaScript* donde los *frameworks* realizan llamadas a los componentes visuales de manera reactiva. Un *callback* es, en este contexto, una función que se ejecuta durante la etapa de entrenamiento de los modelos de *Machine Learning*, permitiendo así una personalización y control comportamental a tiempo real del modelo durante este proceso. Así pues, la utilización de *callbacks* adaptados a bibliotecas como *Keras* o *Tensorflow*, facilita la toma de decisiones en función de umbrales o datos específicos, brindando más flexibilidad al entrenamiento a poder redireccionar este proceso según sea necesario.

Los modelos entrenados en este proyecto tienen algunos *callbacks* implementados, que facilitan el flujo del entrenamiento. El primero de ellos es *Early Stopping*, cuyo objetivo es el de abandonar el proceso de entrenamiento cuando la métrica monitoreada parece ya no mejorar [40]. Este tipo de arquitecturas donde lo que se busca es la minimización de la pérdida (*loss*), para este caso particular en el juego de datos de validación (métrica *val_loss*), utiliza el modo *"min"* para identificar una cota inferior.

Además, este *callback* permite el argumento *patience* establecido en estos entrenamientos en tres. Esto se traduce en que el modelo seguirá entrenando hasta tres *epochs* más a pesar de que el valor de la métrica monitorizada no haya mejorado, permitiendo así un margen por si, por ejemplo, hubieran algunos *epochs* donde los cambios hayan sido muy bruscos y no alcanzara una buena corrección en los pesos rápidamente.

```
1 from tensorflow.keras.callbacks import EarlyStopping
2 early_stopping_callback = EarlyStopping(monitor='val_loss', patience=3, mode='min')
```

Programa 4.1: Definición del callback de Early Stopping

La línea utilizada para instanciar este *Early Stopping* está en el fragmento de código 4.1. En ella se puede apreciar la métrica monitorizada, la paciencia o margen de aceptación de desmejora, y el modo con la que calcularla. En este ejemplo, le entrenamiento debería parar si la métrica utilizada, en este caso *val_loss*, no disminuye en tres *epochs* seguidos.

El parámetro *patience* es, en definitiva, el número de *epochs* seguidos sin mejora de acuerdo con el criterio indicado, a partir del cual se corta el entrenamiento. Otro *callback* definido es *Plot Learning*, quien permite realizar las gráficas de las métricas establecidas al finalizar cada *epoch*. Para ciertos modelos, donde los tiempos de proecsamiento de datos pueden ser muy elevados, contar con este tipo de herramientas facilita de manera visual el control del progreso. Este fragmento de *callback* fue tomado de la Práctica 1 de la asignatura *Deep Learning* de la UOC. La instanciación del *callback* se realiza

mediante el fragmento en el programa 4.2.

```
plotting_callback = PlotLearning()
```

Programa 4.2: Definición del callback de Plot Learning

En la figura 4.4 puede verse visualmente un ejemplo del resultado esperado, con la salvedad que de cara a este trabajo, no se tiene *accuracy*, sino MSE y MAE. Al código fuente completo puede accederse mediante el Anexo A.2.

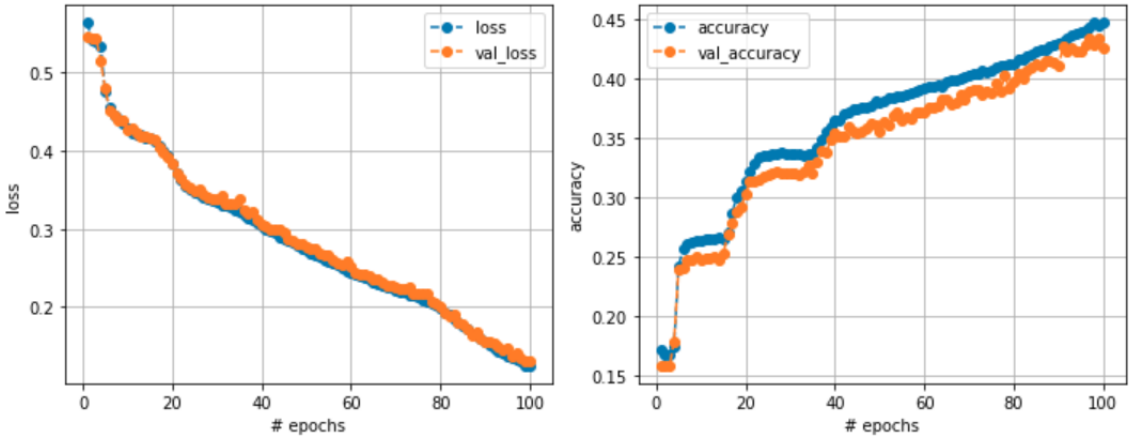


Figura 4.4: Ejemplo de salida del callback Plot Learning.

El tercer *callback* implementado está dedicado a la medición temporal del entrenamiento en sí. Almacena cada uno de los instantes temporales donde culmina un *epoch* y permite una trazabilidad del tiempo de ejecución completo y parcial. En el fragmento del programa 4.3 puede verse su instanciación. Como los demás, debe ser añadido a la lista de *callbacks* de la instrucción de entrenamiento.

```
timer_callback = TimerCallback()
```

Programa 4.3: Definición del callback de Timer

Finalmente, se utiliza *TensorBoard* como herramienta de visualización y monitoreo [7]. Es una biblioteca utilizada para examinar y comprender, tanto el rendimiento como el comportamiento de los modelos de *Machine Learning*. Permite un manejo interactivo mediante gráficas, histogramas, métricas, entre otros, de forma que facilita un acceso de visualización bastante amplio.

En el programa 4.4 se muestra, por un lado el proceso de instalación de la biblioteca *TensorBoard*, la preparación del entorno donde almacenará los logs, cómo instanciar el *callback* y por otro, dónde se añade dentro de la función de entrenamiento. Esto es accesible por medio de un navegador local a través de https://localhost:6006/.

```
!pip3 install tensorboard # Instalación del paquete
%load_ext tensorboard # Carga del módulo
!rm -rf ./logs/ # Vaciar los logs previos
log_dir = "./logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S") # Definir directorio logs
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
...
theModel.fit( ... callbacks=[tensorboard_callback, ... ]
```

Programa 4.4: Definición del callback de TensorBoard

4.4. Métricas

Esta arquitectura de *Encoder-Decoder* tiene por entrada una imagen en baja calidad. Además, tiene su *ground truth* en mejor resolución para la validación una ves se realiza la predicción. Esto permite ajustar los pesos según corresponda durante el proceso de entrenamiento. En este contexto, puede considerarse un tipo de aprendizaje supervisado, puesto que se cuenta con información de qué esperar como resultado por cada imagen que se procesa (pares de imágenes).

Podría considerarse un modelo de regresión que consiste en la predicción de la imagen de salida a partir de su correspondiente entrada, cuya mejora radica en la minimización de la diferencia entre la salida real y la salida esperada. De esta forma, se puede conseguir mediante el entrenamiento y una estructura de filtros adecuada, generar una imagen muy similar.

Los modelos obtenidos aquí, al plantearse como una suerte de problema de regresión, implican definir una serie de métricas que se utilizan para considerar si se está realizando una buena aproximación de la imagen esperada. Luego del entrenamiento, estos deben ser capaces de generalizar una salida para cualquier imagen que reciban. En función de los datos de entrenamiento, estos modelos sabrán procesar imágenes submarinas, por lo que los filtros visuales, inferencias de color y definición de formas que sean capaces de aplicar, estarán muy de la mano de este contexto.

Las métricas que se utilizan para medir la bondad de estos modelos son los que se están definiendo a continuación. Los valores obtenidos son clave a la hora de decantarse por un modelo u otro, y son la piedra angular sobre la que se apoya el análisis cuantitativo que permite elegir el modelo final.

4.4.1. Error Cuadrático Medio (MSE)

El error cuadrático medio o *Mean Squared Error* (MSE) es una métrica de proporciona una función cuadrática de pérdida, mediando el cuadrado de todas las diferencias entre cada valor real y el estimado. Esto permite penalizar con un valor significativamente mayor los errores por encima de la unidad (valores atípicos y *outliers*) y, además, minimizar los errores que estén por debajo de ella.

Es una métrica muy utilizada en ML para la medición de la calidad de los modelos de regresión, teniendo su base conceptual en la estadística. Como referencia entonces, cuanto menos sea el valor del MSE, más consistente será el modelo y, por tanto, mejor generalización y mejores resultados se

obtendrán en promedio. Utilizando la función *mean_ squared_ error* de la biblioteca *Scikit-image*, este cálculo se hace mediante imágenes de tres canales, normalizadas en valores en punto flotante entre 0 y 1.

La forma de calcularse está reflejada de forma compacta en la figura 4.5. La expresión es muy similar a la medida estadística de la varianza (σ^2) para el cálculo de incertidumbre.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Figura 4.5: Fórmula del MSE

A pesar de que el MSE es un métrica muy utilizada en el análisis de datos, cabe destacar algunas de sus desventajas que, en principio, debieran ser consideradas a la hora de evaluar los modelos resultantes. En primer lugar, es muy sensible a los valores atípicos, ya que, al elevar al cuadrado las diferencias, cualquier diferencia medianamente considerable elevaría el valor de la métrica hacia un valor sutancialmente más alto. Para reflejar la calidad, es necesario que se considere previamente la existencia de estos valores y cómo impacta en el resultado final.

Otro punto a destacar es que el MSE no es una medida absoluta, lo que implica que puede no dar datos precisos del modelo. Es decir, en un escenario donde los errores más grandes tengan una cierta prioridad frente a los errores más pequeños, el resultado arrojado por esta métrica no sería del todo fiable de cara a la precisión del modelo. Finalmente, considerar que el MSE no es una medida fácilmente interpretable, ya que no proporciona información de la significancia estadística del modelo. Sin embargo, es una herramienta muy útil si se utiliza acompañado de otras métricas.

4.4.2. Error Absoluto Medio (MAE)

El Error absoluto medio o *Mean Absolute Error* (MAE) es otra métrica muy utilizada para medir la calidad de un modelo de regresión. Sigue el mismo patrón que el MSE en cuanto a la medición de la diferencia entre los valores reales y los valores estimados. Sin embargo, no eleva al cuadrado estas diferencias, sino que le aplica el valor absoluto. Así, es capaz de medir la desviación del modelo pero reduciendo la sensibilidad de los valores atípicos. Esto es, que no penaliza los errores más significativos conforme sean más grandes, aunque con eso permite una cierta estabilidad en su presencia.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Figura 4.6: Fórmula del MAE

Análogamente al MSE, la forma de medir este error es mediante la función *mean_ absolute_ error* de la biblioteca *Scikit-image*, con imágenes de tres canales, normalizadas en valores en punto flotante entre

0 y 1. Igual que el MSE, cuanto menor sea el valor resultante del MAE, mejor será el modelo en su regresión, lo que se traduce en una generación de imágenes más cerca de las mejoradas. Esta métrica es útil cuando se busca minimizar el impacto de las anomalías en los datos. En síntesis, esta métrica se basa en la media de los valores absolutos de sus diferencias, tal y como puede verse en la figura 4.6.

4.4.3. Coeficiente de Correlación de Pearson (PCC)

El Coeficiente de Correlación de Pearson o *Pearson Correlation Coefficient* (PCC), es otra medida muy común de medir la correlación lineal, en este caso particular, entre la imagen resultado con respecto de la imagen mejorada esperada. El coeficiente varía en un rango de $[-1; 1]$, siendo -1 una correlación negativa perfecta y 1 exactamente lo contrario. Cuánto más próximo al 0, menos correlación existen entre ambas imágenes, es decir, que la diferencia entre ellas es mayor.

El PCC es calculado como la diferencia entre los píxeles de las dos imágenes. Por lo tanto, si este coeficiente es muy próximo a 1, las imágenes serán muy similares, mientras que de estar más próximo a 0, serán muy distintas. En general, obtener un coeficiente próximo a 1, se traduce e que el modelo ha logrado obtener una buena reconstrucción de la imagen esperada. En la figura 4.7 se puede ver la expresión matemática para el cálculo de esta métrica.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

Figura 4.7: Fórmula del PCC

4.4.4. Índice de Similitud Estructural (SSIM)

El Índice de Similitud Estructural o *Structural Similarity Index* (SSIM) es la cuarta métrica utilizada en el análisis de estos modelos, que busca medir la calidad de imagen evaluando la similitud estructural entre dos de la salida reconstruida en comparación con la imagen correspondiente mejorada. El SSIM compara la estructura local de las imágenes, es decir, cómo varía la intesidad de los píxeles en distintas ventanas a lo largo de la imágen. En cada una de estas ventanas se calculan tres valores, luminancia, contraste y estructura. La combinación de estas tres medidas en una fórmula (figura 4.8 permiten obtener el valor del SSIM.

El rango en el que se mueve el SSIM va en $[-1; 1]$. Análogo al PCC, cuanto más próximo a 0 sea el resultado, más diferirán las imágenes, así como también, cuanto más sobre 1 ó -1, estarán directamente o inversamente en perfecta correlación. En la figura 4.8 se muestra la expresión matemática para calcular este coeficiente, donde x e y son las dos imágenes a comparar, μ_x y μ_y son las medias y σ_x y σ_y las desviaciones estándar de cada una de las imágenes. Por otr lado, σ_{xy} la covarianza entre ambas, C_1 y C_2 son constantes para evitar la división por cero y ajustar el rango de valores del SSIM, por ejemplo, $C1 = (k_1 L)^2$ y $C2 = (k_2 L)^2$. Finalmente, L es el rango de valores de la imagen (por ejemplo, $L = 255$ para imágenes en escala de grises de 8 bits) y $k_1 = 0.01$ y $k_2 = 0.03$ son constantes empíricas.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Figura 4.8: Fórmula del SSIM

En este contexto de la reconstrucción de imágenes, utilizar la métrica de SSIM es una buena práctica que facilita la interpretación de la bondad o no del modelo. Cuanto más alejado del cero esté, mejores resultados se consiguen.

4.4.5. Ecualización Adaptativa de Histograma (CLAHE)

La última métrica que se utiliza para la evaluación cuantitativa de estos modelos está basada en Ecualización Adaptativa de Histograma o *Contrast Limited Adaptive Histogram Equalization* (CLAHE). CLAHE es un método de procesamiento de imágenes utilizado para mejorar el contraste local de las mismas, basando su trabajo en el ajuste de la intensidad lumínica de cara a mejorar su contraste. CLAHE es una variante del método de ecualización de histogramas clásico que ajusta la iluminación de una imagen en su totalidad. Por su parte, esta versión realiza el proceso en subregiones de la imagen de forma independiente, lo que permite un mejor resultado ya que evita la amplificación del ruido en regiones de baja varianza.

En la figura 4.9 puede verse la instanciación de cómo trabaja CLAHE, donde A es la imagen, h y w son las medidas de alto y ancho de la ventana de paso, n es el número de *bins* del histograma y k es el coeficiente de mejora de contraste, utilizado para graduar la mejora de contraste del *output*. Esta divide la imagen en pequeños bloques, aplicando equalización de histograma a cada uno adaptativamente. A_r representa la imagen con un contraste mejorado, basándose de la imagen de entrada A. La segunda línea, realiza un ajuste de contraste adicional previo a devolver la imagen final, donde k es un factor de ajuste y lo que suma a la imagen A_r es una versión escalada de la diferencia entre la imagen original y aquella que se aplicó CLAHE ($k(A - A_r)$).

$$A_r = \text{CLAHE}(A, h, w, n)$$
$$A_{out} = A + k(A - A_r)$$

Figura 4.9: Corrección con CLAHE

Como se ha dicho previamente, CLAHE mejora el contraste de una imagen. En el contexto de este proyecto, es utilizado como un comparador entre la imagen original aplicándose CLAHE y sus diferencias con las imágenes reconstruida.

Ahora bien, esto permite definir si un modelo tiene mejores resultados que si a la imagen sólo se le aplica CLAHE. No es responsabilidad de este modelo colorear las imágenes, sino mejorar la calidad de las mismas en términos generales. En el código 4.5, está la función construida, que recibe las tres

imágenes, realiza las transformaciones correspondientes y devuelve la comparación de estas métricas utilizando MSE y MAE.

```
def diff_with_clahe(img_original, img_enhanced, img_predicted):
    num_pixels = img_original.shape[0] * img_original.shape[1]
    CLAHE = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))

    # Every image to grayscale
    img_enhanced_gray = cv2.cvtColor(np.uint8(img_enhanced * 255), cv2.COLOR_BGR2GRAY) /255.0
    img_predicted_gray = cv2.cvtColor(np.uint8(img_predicted * 255), cv2.COLOR_BGR2GRAY)/255.0
    img_original_gray = cv2.cvtColor(np.uint8(img_original * 255), cv2.COLOR_BGR2GRAY)

    # Apply CLAHE to original and normalize
    img_original_clahe_gray = CLAHE.apply(img_original_gray)/255.0
    img_original_gray = img_original_gray/255.0

    # If MSE_PRED<MSE_CLAHE: This Model is better than CLAHE
    MSE_PRED = mean_squared_error(img_predicted_gray, img_enhanced_gray)
    MSE_CLAHE = mean_squared_error(img_original_clahe_gray, img_enhanced_gray)
    MAE_PRED = mean_absolute_error(img_predicted_gray,img_enhanced_gray)
    MAE_CLAHE = mean_absolute_error(img_original_clahe_gray,img_enhanced_gray)

    return [MSE_PRED, MSE_CLAHE, MAE_PRED, MAE_CLAHE]
```

Programa 4.5: Función de comparación con CLAHE

Sintetizando el funcionamiento del código 4.5, recibe la imagen original, su *ground truth* y la imagen reconstruida. Convierte todo a escala de grises y le aplica *CLAHE* solamente a la imagen original. Finalmente, calcula las distancias utilizando MSE y MAE, entre el *ground truth* y la imagen original con *CLAHE* y también la primera con la imagen regenerada. En función de los errores que devuelva, se puede visualizar cuál de las dos técnicas funciona mejor en el contexto de escala de grises.

El costo de procesamiento de esta diferencia es bastante alto, por lo que se mide sobre una base representativa de imágenes que pueda extrapolarse a todo el conjunto en su totalidad. Se añade sí al conjunto de *test* donde para cada imagen, identifica qué algoritmo hubiera sido más eficiente en términos de aproximación a la imagen esperada.

4.5. Adaptaciones

4.5.1. Modelos

La arquitectura propuesta de tipo *Encoder-Decoder*, es la base para el entrenamiento con varias configuraciones de hiperparámetros. De cara a este proceso de entrenamiento, se harán cambios en los filtros, en cantidad y dimensiones, *kernel*, *epochs*, entre otros. Al resultado de cada entrenamiento con cada una de las configuraciones seleccionadas, por simplicidad se le llamará *"modelo"*. Este modelo es, en realidad, el fichero que contiene los anteriores parámetros entrenados.

La forma de comparar las métricas de cada uno de estos modelos y, por descontado, de utilizarlos luego en casos reales, implica su almacenamiento. Dicho esto, el entrenamiento prevé el almacenamiento de cuatro modelos, el *encoder*, el *decoder*, el modelo completo y un fichero que guarda las métricas obtenidas a lo largo de todo el entrenamiento.

Dentro del repositorio proporcionado, se incluyen los modelos dentro del directorio *saved_ models*, con extensión *h*5 y el formato propio de *Keras*, e información adicional del entrenamiento, en el mismo directorio, con extensión *pkl* y el formato propio de la biblioteca *Pickle*.

4.5.2. Imágenes

La definición de la arquitectura se ha tomado de un *Convolutional Autoencoder* [38], modificando una serie de características para adaptarlo a las necesidades específicas de este proyecto. Una de ellas implica un cambio a nivel de la validación.

El CAE original evaluaba el resultado del proceso de recreación contra la propia imagen de entrada. Sin embargo, como el objetivo de estos modelos aquí generados no es sólo reconstruir, sino además mejorar, implicaba cambios que permitan a estos actualizar sus pesos con respecto a los *ground truths* correspondientes a cada imagen de entrada. En el programa C.7, se muestra cómo la clase *Autogenerator* ahora permite también tomar pares de imágenes, extendiendo las funcionalidades del autoencoder original a un encoder-decoder con imágenes y sus *ground truths*. En la función *autobuild* en el programa C.10, a la hora de llamar a la función *get_filenames()*, le pasa el parámetro de las dos carpetas con imágenes, es decir, las imágenes distorsionadas y sus *ground truths*. A través de este programa, se facilita la creación de dos conjuntos de datos, uno de entrenamiento y otro de validación, y cada uno tiene sus pares de imágenes que servirán para mejorar la predicción.

A la hora de entrenar y evaluar los modelos, se le debe pasar las rutas de las carpetas que contienen los juegos de datos de validación y también de *test* para que este proceso genere además las métricas de los tres grupos de datos: *train*, *validation* y *test*. La inicialización del programa de entrenamiento y validación está en el anexo C.13. Otro cambio, aunque más superficial que el anterior, es modificar la extensión de las imágenes. Tal y como se puede ver en el *dataset EUVP*, las imágenes están en formato *jpg* mientras que el código espera imágenes *png*. El mismo se realiza mediante la constante *EXT_ IMAGE*. Esto implica, por un lado, el cambio de extensión esperada en el modelo, y luego una unificación y eventual transformación de los ficheros de entrada, ya que durante el proceso de evaluación, puede haber distintos tipos de formatos. Finalmente, se cierran las posibilidades de extensiones a *png* y *jpg* para evitar el tratamiento específico de otro tipo.

4.5.3. Funciones extras

En el código 4.6 se han definido una serie de funciones auxiliares que facilitan la interpretación, dando más claridad al formato en el que se muestran los resultados de cada proceso del entrenamiento y evaluación.

```python
from datetime import timedelta
from prettytable import PrettyTable

def print_table(title, headers, values):
    table = PrettyTable() # Create the table
    table.title = title
    table.field_names = headers
    table.add_rows(values) # Add rows
    print(table)

def format_time(timestamp):
    """ Function to plot time in format HH:MM:SS """
    tmp = str(timedelta(seconds=int(timestamp)))
    tmp = tmp.split('.')[0]
    return tmp

class Color_Text:
    """ Class to print in color """
    def bold(self, in_text):
        print (f"\033[1m {in_text} \033[0m")

    def red(self, in_text):
        print (f"\033[41m {in_text} \033[0m")

    def green(self, in_text):
        print (f"\033[42m {in_text} \033[0m")

text = Color_Text() # Object to print in color
```

Programa 4.6: Funciones auxiliares

Se han añadido algunas funciones sobre la estructura de *autoModel*, que facilitan la extracción de información como por ejemplo *count_params()* en el código 4.7, que devuelve la suma de todos los parámetros que el modelo debe entrenar, es decir, los del *encoder* y los del *decoder*.

```python
    def count_params(self):
        return sum((self.encoderModel.count_params(), self.decoderModel.count_params()))
```

Programa 4.7: Función count_params()

Además, a modo de centralizar la responsabilidad de procesar los datos de cada modelo, se crea la función *returnMetrics* (programa A.6 en anexo A.5) que procesa cada uno de los modelos entrenados devolviendo las métricas establecidas tal y como se verá en el siguiente capítulo. También es importante destacar que se controla si está realizando las métricas en *train* o *test*, ya que la información de MSE y MAE de train está almacenada en el *pkl* correspondiente al entrenamiento del modelo, mientras que todas las demás están en otro fichero dentro de *./models_metrics/*.

Capítulo 5

Entrenamiento

En el capítulo anterior se hizo referencia a la arquitectura *Encoder-Decoder*, utilizada para el desarrollo de un modelo que mejore imágenes submarinas. Ahora bien, una vez se han tenido las imágenes ordenadas, renombradas y separadas, el siguiente paso es el entrenamiento de los modelos. Como se ha comentado previamente, se han entrenado 13 modelos distintos, cuya diferencia principal radica en los filtros convolucionales que permiten la obtención de mapas de características.

Los modelos que han sido entrenados y probados están identificados con números, con el fin de evitar el uso de una nomenclatura compleja propensa a confusión. Los nombres de cada uno de ellos están definidos en la tabla 5.1 y, a partir de aquí, estos son referenciados a través de su identificador numérico asociado.

La estructura de sus nombres se compone de "*AUTOENCODER*", los filtros convolucionales que se aplican en el orden en que se procesan en el *encoder*, y al final el número de *epochs* máximos permitidos siguiendo el patrón "*EPOCHSXX*", donde XX es el valor correspondiente.

En todos los modelos entrenados en este proyecto, el número máximo de *epochs* es 30, por lo que el sufijo de todos estos es "*_EPOCHS30*". El programa que define estos nombres de manera automática es accesible en el anexo C.6.

Cabe recordar, al menos brevemente, que cada uno de los modelos aquí presentes se componen de tantas capas de convolución como el doble del número de filtros tenga. A modo de ejemplo, el modelo cuyo nombre es *AUTOENCODER_128_4_EPOCHS30*, se compone de una primera capa convolucional con 128 filtros y una segunda de 4 filtros, ambas en el *encoder*, mientras que en el *decoder* esta estructura es simétrica, presentando una capa convolucional de 4 filtros y una segunda consecutiva de 128. En total, este modelo tiene 4 capas convolucionales.

En las tablas 5.2 y 5.3, se muestran las métricas obtenidas para cada uno de los modelos entrenados. Para todos ellos se definieron un máximo de 30 *epochs*, además de *Early-Stopping* con un *patience* de 3. Como se puede ver en la tabla, ninguno alcanzó el máximo de *epochs* disponibles antes de estabilizar la pérdida, aunque prácticamente todos alcanzaron resultados, en general, bastante buenos.

La media de los *epochs* ha sido muy próxima a 17, lo que permite ver que, a excepción de unos pocos casos, se han mantenido en ese entorno. El hecho de que todos los modelos hayan parado el entrenamiento mediante *Early-Stopping* permite asegurarnos que estos han mejorado todo lo que han podido antes de hacer *overfitting*. Esto también se podría leer, en cierta forma, como alcanzar la mejor opción de modelo con la configuración y las características definidas.

Ahora bien, considerando que todos los modelos han sido estabilizados antes de los 30 *epochs* disponibles, un camino interesante sería tomar como primera referencia los modelos cuyos *epochs* ronden la media. De esta manera, se estaría asegurando que el modelo no ha visto demasiadas veces los mismos datos y pudiera estar memorizando, ni tampoco haberlos visto pocas y no poder generalizar correctamente.

Model ID	Model Name
Modelo 1	AUTOENCODER_32_64_128_256_EPOCHS30
Modelo 2	AUTOENCODER_32_64_128_EPOCHS30
Modelo 3	AUTOENCODER_256_128_3_EPOCHS30
Modelo 4	AUTOENCODER_256_128_256_EPOCHS30
Modelo 5	AUTOENCODER_256_256_64_EPOCHS30
Modelo 6	AUTOENCODER_128_8_4_EPOCHS30
Modelo 7	AUTOENCODER_128_128_16_EPOCHS30
Modelo 8	AUTOENCODER_128_4_EPOCHS30
Modelo 9	AUTOENCODER_128_8_EPOCHS30
Modelo 10	AUTOENCODER_128_128_32_EPOCHS30
Modelo 11	AUTOENCODER_128_16_EPOCHS30
Modelo 12	AUTOENCODER_128_128_64_EPOCHS30
Modelo 13	AUTOENCODER_128_32_EPOCHS30

Tabla 5.1: Nomenclatura y referencia de los modelos.

Model ID	Epochs	Filters	Params	Train time
Modelo 1	12/30	[32, 64, 128, 256]	1370499	6:36:43
Modelo 2	19/30	[32, 64, 128]	335747	2:20:15
Modelo 3	19/30	[256, 128, 32]	690755	2:23:12
Modelo 4	14/30	[256, 128, 256]	1789699	1:43:08
Modelo 5	10/30	[256, 256, 64]	1531011	1:13:36
Modelo 6	17/30	[128, 8, 4]	27467	2:05:57
Modelo 7	19/30	[128, 128, 16]	343715	2:39:16
Modelo 8	28/30	[128, 4]	17595	3:44:54
Modelo 9	19/30	[128, 8]	27283	3:05:06
Modelo 10	12/30	[128, 128, 32]	387651	1:34:14
Modelo 11	10/30	[128, 16]	47523	1:13:32
Modelo 12	10/30	[128, 128, 64]	489347	1:13:16
Modelo 13	22/30	[128, 32]	91459	2:41:09

Tabla 5.2: Resumen general de los modelos.

El entorno utilizado fue *OVHcloud*, implementado en un *Jupyter Notebook* con 10GB de SSD, 40GB de memoria RAM y 13vCores. Además, se ha incrementado la capacidad de cómputo contratando recursos adicionales como una GPU Nvidia Tesla V100S, con 32GB de memoria RAM dedicada. Por lo tanto, los tiempos que aquí se describen, están circunscritos a este espacio de trabajo.

5.1. Métricas durante el entrenamiento

El total de tiempo requerido para el entrenamiento completo asciende a más de 32 horas (32:34:18). El promedio de tiempo de entrenamiento fue de 2 horas, 30 minutos, 20 segundos (02:30:20), representado en la figura 5.1 con la línea roja.

El modelo 12 presentó el tiempo más bajo 01:13:16 seguido con apenas unos segundos de diferencia por el modelo 11. El modelo con tiempo de entrenamiento más alto ha sido el 1 con 6:36:43. La figura 5.1 representa la proporción de duración del entrenamiento considerando la más baja como la unidad. De esta forma, es visualmente fácil ver cuáles han tenido un tiempo de entrenamiento por encima de la media y, por el contrario, cuáles podemos asumir que han alcanzado la estabilidad en menos tiempo.

Otro factor a tomar en cuenta en el análisis de los modelos son los parámetros que se tiene que ajustar. El número de parámetros entrenables influye directamente en la capacidad de un modelo de capturar patrones y características, aunque también supone un riesgo en el tiempo que necesita su entrenamiento y es más propenso a *overfitting*. La media de los parámetros está próxima a 550.000, lo que, como indicador, permite definir una referencia sobre los parámetros.

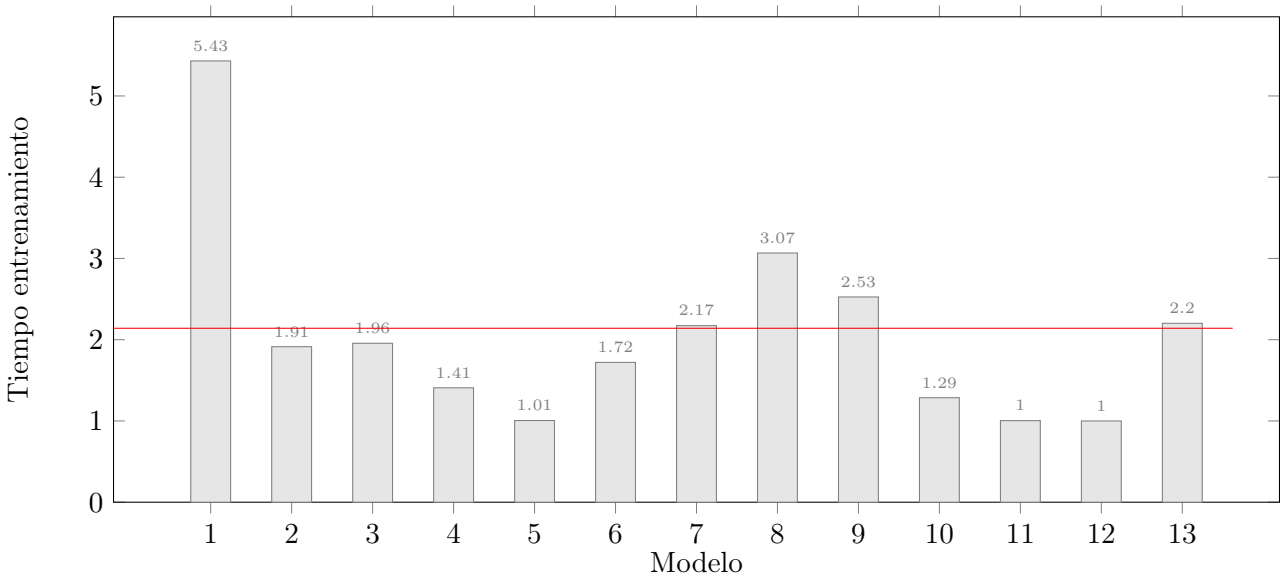


Figura 5.1: Proporción de tiempo de entrenamiendo por modelo

Los filtros convolucionales, responsables de la extracción de características de las imágenes, permiten a los modelos capturar más patrones y más complejos cuanto mayor sea el número aunque, análogamente a los parámetros, suponen un riesgo en el tiempo y en *overfitting*.

Ahora bien, para las imágenes generadas por cada modelo y en función de lo comentado en el capítulo anterior, se han utilizado una serie de indicadores que permiten comparar la imagen generada con respecto a su *ground truth*. Para recordar, este modelo *Encoder-Decoder* recibe una imagen distorsionada a causa de los efectos de la toma submarina como *input*, teniendo por salida una imagen reconstuida, siendo la misma que se le ingresa con mejor calidad. Este proceso es se puede validar mediante la distancia entre ambas imágenes, ya que se tiene una imagen mejorada como referencia de qué esperar como *output*.

En principio, solamente con la información a partir de los parámetros hasta ahora mencionados no se puede afirmar nada, por lo que ahora se pasa al análisis de las otras métricas. Esta nueva información está disponible en la tabla 5.3.

Las métricas que se han utilizado son *Mean Squared Error* (MSE) y *Mean Absolute Error* (MAE), ambas indican la distancia que hay entre la imagen regenerada y su respectivo *ground truth*. Estas son tomadas al finalizar cada *epoch* en los juegos de datos de *train* y *validation* por separados. En principio, las mismas deberían de ser dos curvas muy similares aunque en la práctica, las imágenes de *validation* pueden distar mucho de lo que el modelo ha conocido hasta ese momento.

Model ID	MSE train	MAE train	MSE val	MAE val	Pearson	SSIM
Modelo 1	0.009274	0.068365	0.009145	0.066077	0.918729	0.787904
Modelo 2	0.008461	0.063908	0.008258	0.062644	0.923734	0.824835
Modelo 3	0.008647	0.065110	0.008620	0.063935	0.924579	0.795288
Modelo 4	0.008696	0.064895	0.009280	0.066246	0.911523	0.819973
Modelo 5	0.009383	0.068113	0.009442	0.067038	0.911766	0.805491
Modelo 6	0.011763	0.078886	0.011933	0.077897	0.890482	0.648877
Modelo 7	0.009121	0.067244	0.009737	0.067655	0.909757	0.777985
Modelo 8	0.009641	0.069222	0.009247	0.066659	0.915163	0.799782
Modelo 9	0.009338	0.067831	0.009682	0.067235	0.913311	0.795536
Modelo 10	0.009279	0.067992	0.008868	0.065100	0.915132	0.794463
Modelo 11	0.009657	0.068883	0.009410	0.066904	0.914052	0.814415
Modelo 12	0.009339	0.068399	0.008675	0.063507	0.920720	0.802384
Modelo 13	0.008678	0.064837	0.009741	0.068637	0.913107	0.783521

Tabla 5.3: Métricas de resultado de entrenamiento.

En la figura 5.2 se pueden apreciar los resultados de este proceso de evaluación. La métrica *mse* -curva azul- representa el error MSE en el conjunto de imágenes de entrenamiento, mientras que *val_mse* -curva anaranjada-, presenta el mismo error MSE pero medido sobre el conjunto de datos de validación. Es decir, *mse* representa la pérdida calculada en cada iteración del entrenamiento sobre su propio conjunto de imágenes. Por otro lado, *val_mse* proporciona una medida del rendimiento del modelo en términos de generalización, ya que los datos sobre los que es realizada esta medida no son objeto del

proceso de entrenamiento.

En ella principamente puede notarse que en todos los modelos las métricas de entrenamiento son más estables y todas bastante similares. En ellas se puede apreciar una rápida disminución del error en las dos primeras *epochs* para posteriormente seguir reduciéndose pero ya a un ritmo más moderado. Estas curvas son suaves, lo que indica que el modelo está aprendiedo de menera efectiva a reconstruir los datos de entrada en sus dos primeras *epochs*, alcanzando luego una estabilidad sugiriendo que ha convergido a un estado óptimo hacia el final.

En cuanto a las curvas de error medidas en la partición de validación, estas presentan una serie de picos que no se ajustan demasiado a la curva de *train*. Esto puede sugerir que el modelo está sufriendo *overfitting*, ya que este se ajusta bastante bien a los datos de *train*, pero a la hora de predecir una imagen que nunca ha visto, no es capaz de realizar una correcta generalización.

Sin embargo, el nivel de error que todos estos modelos manejan es significativamente pequeño. Partiendo de las gráficas, se podría asumir que todos estos aproximan bastante bien, incluso las imágenes de *validation*, puesto que los valores de MSE fluctúan en una media de 0.009.

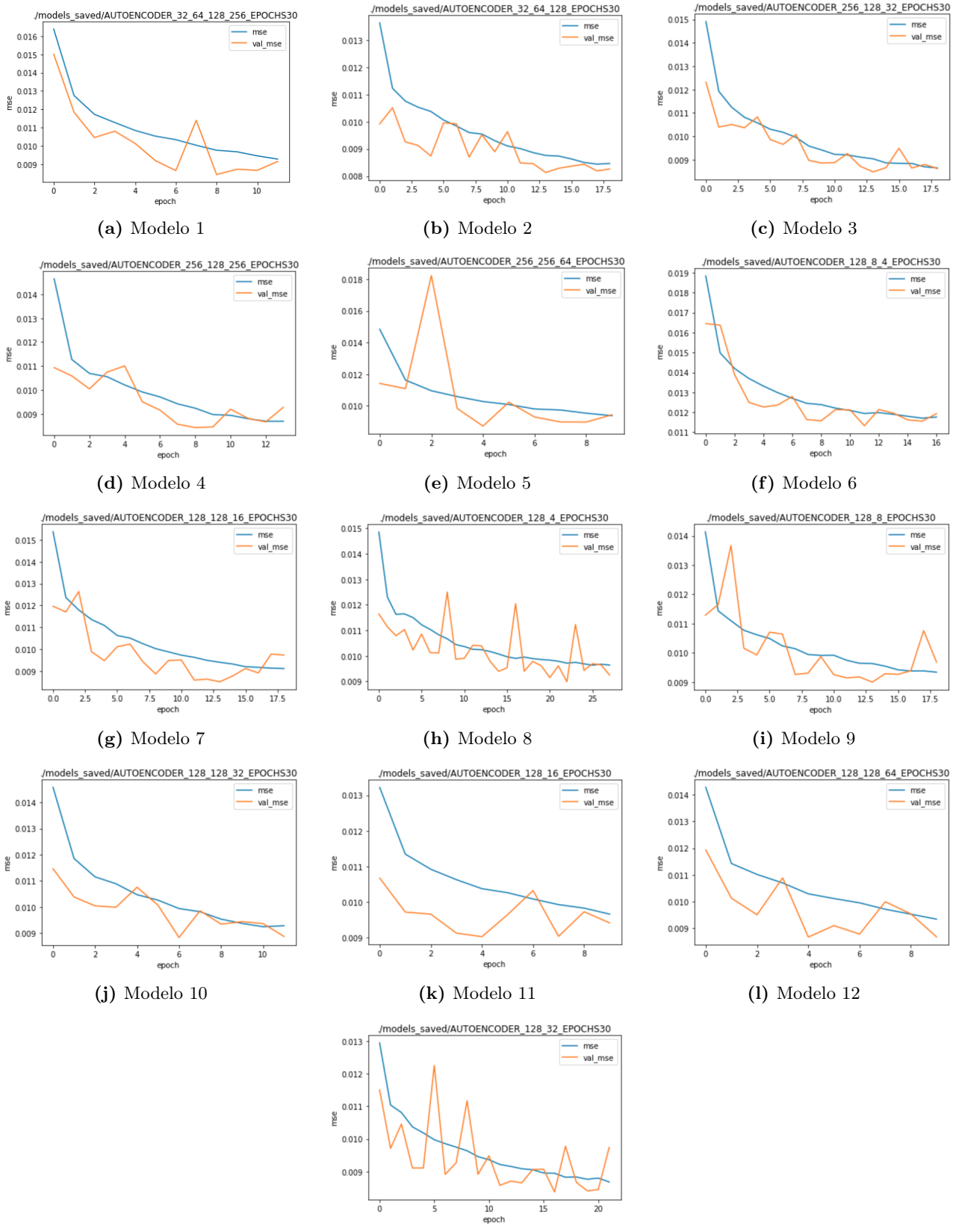
Desde una perspectiva visual, esto sugiene que las imágenes serán lo suficientemente parecidas para detectar que sí está generándola, aunque pudiera, eventualmente, perder detalles. Ahora bien, si se considera todo el conjunto de modelos entrenados, las medias de los resultados de MSE que se obtienen están en los intervalos $[0.008461; 0.011763]$ y $[0.008258; 0.011933]$ para *train* y *validation* respectivamente.

Esto se traduce en que los modelos generados pueden ser considerados bastante buenos, ya que logran disminuir el error a valores significativamente pequeños. Si se analizan ambos extremos del intervalo de *validation*, se puede entender que, en promedio, la imagen resultante de la imagen esperada, dista un 0.8 % en el mejor de los modelos y un 1.17 % en el que peor resultados devuelve.

Además del MSE, se ha considerado el MAE como métrica para medir el error de los modelos resultantes respecto del resultado esperado. Análogamente a la primera métrica utilizada, la evolución de esta segunda puede verse para cada modelo en la figura 5.3, cuyas gráficas son muy similares con a las de MSE. Los valores, en cambio, oscilan en un intervalo de valores de seis a ocho veces superior aunque mantienen un umbral más pequeño entre ellos.

No obstante, los valores del MAE siguen la misma tendencia que los del MSE. Por lo que a métricas se refiere, ambas son consistentes con respecto de los modelos entenados en función del primer error analizado. Tomadas las medias de estos errores en *train* y *validation*, los valores oscilan en los intervalos $[0.063908; 0.078886]$ y $[0.062644; 0.077897]$ respectivamente.

Esto podría leerse como una diferencia media de entre un 6.4 % y un 7.8 % entre la imagen generada y su *ground truth* en términos de error, en los casos de *validation* que la red nunca ha visto.



(a) Modelo 1 **(b)** Modelo 2 **(c)** Modelo 3

(d) Modelo 4 **(e)** Modelo 5 **(f)** Modelo 6

(g) Modelo 7 **(h)** Modelo 8 **(i)** Modelo 9

(j) Modelo 10 **(k)** Modelo 11 **(l)** Modelo 12

(m) Modelo 13

Figura 5.2: Resultados de MSE en los modelos.

La forma de medir el MAE es evidentemente distinta al MSE, por lo que tiene sentido que los valores puedan estar en otra escala. Sin embargo, estos valores indican que los modelos son aceptables, es decir, que ya no son tan buenos como indicaba el primer error.

De la misma forma que con el MSE, las curvas en *train* se comportan de una manera bastante estable, reduciendo gradualmente a partir de la segunda *epoch*. Sin embargo las curvas en *validation* tiene picos que pudieran sugerir además un problema de *overfitting*.

Aún así, esta métrica, a pesar de devolver valores más altos, permite ver que los modelos estarían prediciendo mejor los valores de *validation* frente a los de *train*, ya que la primera curva tiende a estar siempre por debajo de la segunda. Este detalle no es menor, puesto que ofrece información respecto de la capacidad de generalización del modelo y de cómo funciona regenerando imágenes que desconoce.

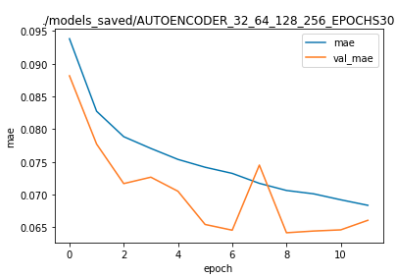
Ahora bien, se han considerado como herramienta de validación otras dos métricas: *Pearson Correlation Coefficient* (PCC) y *Structural Similarity Index* (SSIM). Ambas métricas devuelven resultados sobre las correlaciones entre las series que analizan.

Para este caso, se han considerado todos los valores involucrados en el entrenamiento, midiendo las similitudes entre la imagen reconstruida y su correspondiente *ground truth*. Como ya se ha explicado anteriormente, el rango de valores de las dos métricas oscila en el intervalo $[-1; 1]$, donde 1 es una correlación perfecta, -1 es una correlación inversamente perfecta y, cuanto más próximo al 0, no tendrán ninguna correlación, por lo que las imágenes serán cada vez más distintas.

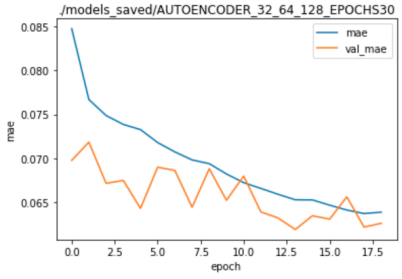
Tal como se puede ver en los gráficos de la figura 5.4, ambas métricas están, a excepción del modelo 6, bastante próximas. Este modelo que destaca, lo hace por tener unos valores bajos de SSIM y PCC, lo cual, al estar más próximo a 0 que todos los demás, se traduce en que las imágenes resultantes distan más de sus correspondientes *ground truths*. Incluso más, este modelo presenta, como puede verse en la tabla 5.3, los resultados más altos de MSE y MAE, lo que indica que el error es mayor entre la imagen generada con respecto a la que se buscaba otbtener, en comparación con los resultados alcanzados por los demás modelos.

Hasta este punto, ya habrían elementos suficientes para descartar el modelo 6 como candidato al modelo final.

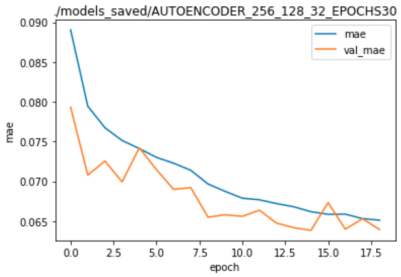
Dentro de los resultados por encima de la media, se puede ver que al modelo 5 presenta el mejor en términos del SSIM, sin embargo en PCC está por debajo de la media.



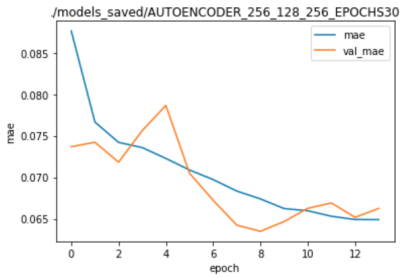
(a) Modelo 1



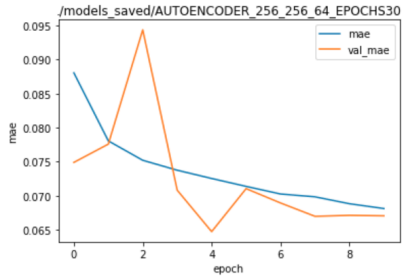
(b) Modelo 2



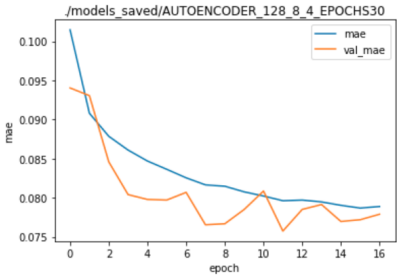
(c) Modelo 3



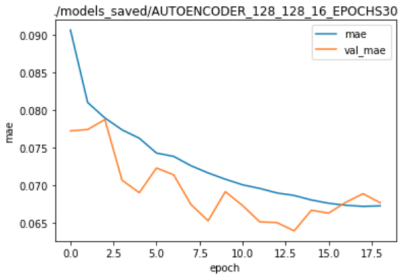
(d) Modelo 4



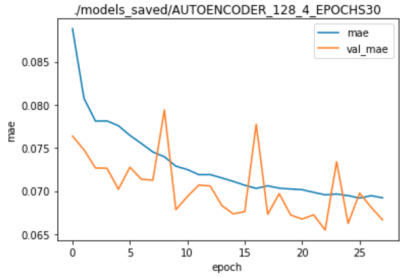
(e) Modelo 5



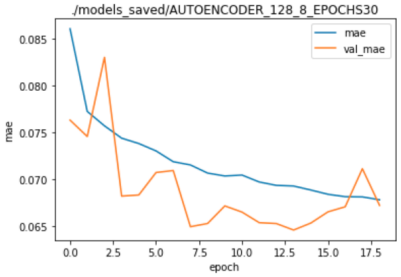
(f) Modelo 6



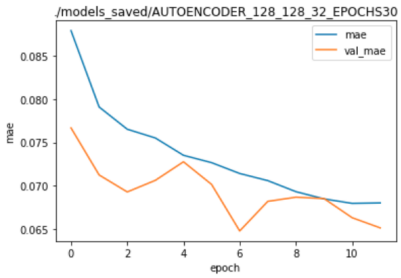
(g) Modelo 7



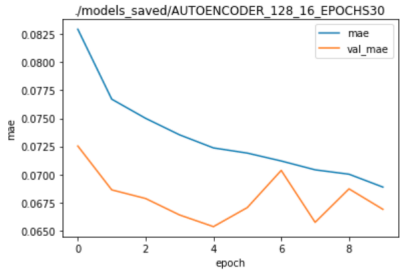
(h) Modelo 8



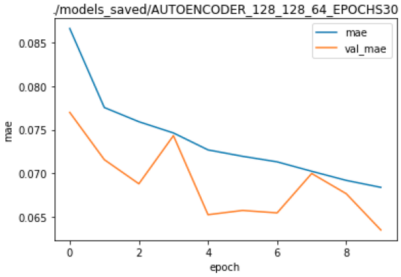
(i) Modelo 9



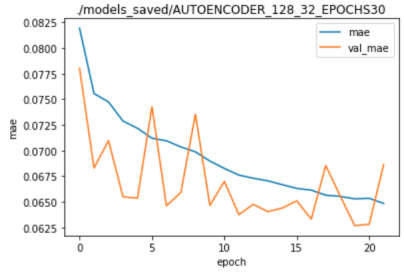
(j) Modelo 10



(k) Modelo 11



(l) Modelo 12



(m) Modelo 13

Figura 5.3: Resultados de MAE en los modelos

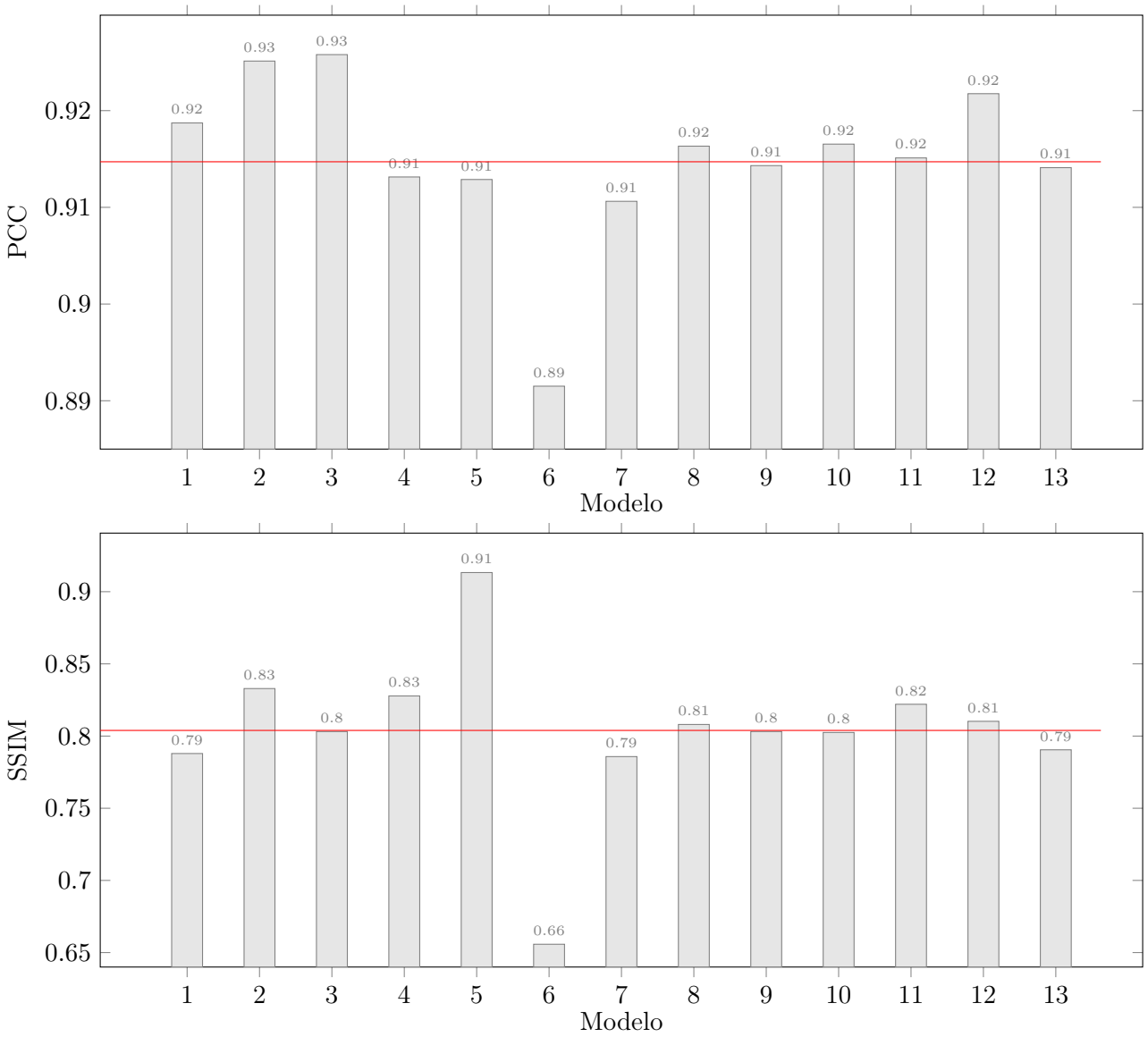


Figura 5.4: Medias de PCC y SSIM por modelo.
Valores de referencia en la tabla 5.3.

El modelo que mejor valor tiene de ambos índices es el modelo 2, ya que en ambos están por encima de la media, aproximándose a 1 en buena medida. Tomando como referencia también los valores del MSE y MAE, el modelo 2 destaca por tener los valores más bajos de ambas métricas, tanto en *train* como en *validation*. Se recuerda que en estos valores, la proximidad a 0 es la dirección de referencia para medir la disminución del error.

5.2. Análisis con CLAHE

Finalmente, se ha realizado un análisis utilizando *Contrast Limited Adaptive Histogram Equalization* (CLAHE) como algoritmo conocido. Para esto, se transforman temporalmente todas las imágenes a

escala de grises, puesto que sobre ellas es la comparación. Se busca con esto cuantificar los errores entre, por un lado, la imagen regenerada y su *ground truth* (MSE_PRED, MAE_PRED), y por otro la imagen original aplicándosele CLAHE y su *ground truth* (MSE_CLAHE, MAE_CLAHE). El resultado que se espera es capaz de medir para cada modelo si generan imágenes mejores o peores que si se hubiera aplicado CLAHE a las imágenes originales.

Como se puede notar, las columnas MSE_CLAHE y MAE_CLAHE en la tabla 5.4 tienen el mismo valor, esto se debe a que no involucra al modelo en sí, sino que es el resultado de medir la distancia entre la imagen original y su *ground truth*. Es decir, ambas imágenes son las mismas para todo el proceso de entrenamiento y validación. Distinto es para las columnas MSE_PRED y MAE_PRED donde ahí sí se mide la distancia respecto de la imagen regenerada.

Model ID	MSE CLAHE	MSE PRED	MAE CLAHE	MAE PRED
Modelo 1	0.017033	0.003047	0.102287	0.041633
Modelo 2	0.017033	0.002593	0.102287	0.038146
Modelo 3	0.017033	0.003129	0.102287	0.041951
Modelo 4	0.017033	0.002754	0.102287	0.038949
Modelo 5	0.017033	0.002803	0.102287	0.040006
Modelo 6	0.017033	0.004550	0.102287	0.050788
Modelo 7	0.017033	0.003451	0.102287	0.044528
Modelo 8	0.017033	0.003066	0.102287	0.042149
Modelo 9	0.017033	0.002953	0.102287	0.041862
Modelo 10	0.017033	0.002941	0.102287	0.040244
Modelo 11	0.017033	0.002824	0.102287	0.039448
Modelo 12	0.017033	0.002866	0.102287	0.040322
Modelo 13	0.017033	0.003064	0.102287	0.042232

Tabla 5.4: Métricas de resultado de entrenamiento.

Así, midiendo los errores utilizando MSE y MAE, las métricas obtenidas en *train* están disponibles en la figura 5.4. La interpretación de esta tabla se realiza mediante las comparaciones de los mismos errores por modelo. Todos estos tienen un MSE_PRED significativamente mejor que su MSE_CLAHE o, en otras palabras, que las imágenes generadas son bastante más próximas a lo esperado que si únicamente se les aplicara CLAHE a la imagen original.

Si el error es medido con MSE, todos los modelos generados debieran ser, en media, mejores que el algoritmo CLAHE, siendo los modelos 2 y 4 los que menor valor de error presentan o, dicho de otra forma, los que imágenes más próximas a lo esperado devuelven.

De la misma forma, cuando el error es medido con MAE, los valores también son menores, lo que indica que los modelos son mejores que únicamente la aplicación de CLAHE como algoritmo de mejora. Si

bien el intervalo de errores es mayor, esto puede deberse a que esta métrica utiliza los valores absolutos de las distancias, frente al MSE que penaliza de forma cuadrática los errores grandes, pero también a los pequeños restándoles importancia.

En este caso, podría interpretarse como un sumatorio creciente de errores pequeños sin penalización, lo que explicaría por qué el MSE es menor. De todas formas, el valor en el que oscilan es de aproximadamente 0.04, lo que sugiere que la proximidad entre las imágenes regeneradas por los modelos y la imagen esperada, ambas en escala de grises, distaría en términos de error un 4

En este sentido, el modelo 2 podría sí ser candidato a modelo final desde el punto de vista de las métricas obtenidas, ya que es el que mejor comportamiento ha reflejado en sus valores hasta el momento. En cualquier caso, es conveniente supeditar la decisión al comportamiento en *test* y también al visual de los modelos. Desde un punto de vida conceptual, mejores métricas implican mejores modelos y mejores resultados, pero para esto es conveniente ver a los modelos en acción.

Capítulo 6

Evaluación de los modelos

Una vez realizado el análisis respecto de las métricas de *train* y *validation*, corresponde cederle el protagonismo a la generalización. En este capítulo, el foco se centra en el comportamiento de los modelos y, en especial, del **modelo 2**, con datos que nunca ha visto, reservados en *test* desde el momento previo al entrenamiento (distribución de imágenes en tabla 3.2).

Naturalmente, los resultados obtenidos en base a las métricas de *train* facilitan una primera toma de contacto con los resultados esperados, sin embargo, es con este dataset donde se ve el comportamiento real de los modelos. Tal y como se ha explicado en el capítulo correspondiente al procesamiento de los datos, las imágenes de *test* son un subconjunto disjunto del utilizado para el entrenamiento. Es decir, estas imágenes han sido tomadas del mismo dataset de imágenes original pero han sido separadas puesto que no han sido vistas por los modelos durante la etapa de entrenamiento.

Es importante insistir en que los datos con los que aquí se mide el rendimiento han sido extraídos del mismo conjunto de imágenes como un sunconjunto desconocido para los modelos. Dada esta procedencia, podrían eventualmente contener ciertos patrones o características similares al primer conjunto, indetectables a la vista humana pero fácilmente reconocibles a nivel de un modelo de *deep learning*, impidiendo así generar completamente los resultados.

Dicho esto, luego del análisis de *test*, que permitirá obtener métricas ya que se cuenta con sus respectivos *ground truth*, se prueban los modelos con imágenes externas que, en principio, no comparten procedencia.

6.1. Métricas en test

Ahora bien, análogamente al capítulo anterior, se realiza la medición del comportamiento de los modelos mediante las métricas definidas pero ahora en *test*. En la tabla 6.1 se puede visualizar estas métricas que, como era de esperar, tienen un comportamiento muy similar a *train*. Así, en la tabla 6.2 están las métricas de la comparación en escala de grises utilizando CLAHE.

Los valores de MSE y MAE, en el modelo 2 siguen siendo los más pequeños, lo que indica que su comportamiento en la reconstrucción de imágenes es bastante bueno, con valores de 0.008472 y 0.063244 respectivamente. También presenta el valor más alto en SSIM, y respecto de la medición a través de PCC, dista del primero recién en la cuarta cifra significativa.

Además, el modelo 6 que previamente se ha podido comprobar que era el que peores resultados devolvía, se vuelve a confirmar también en *test* que mantiene esta característica.

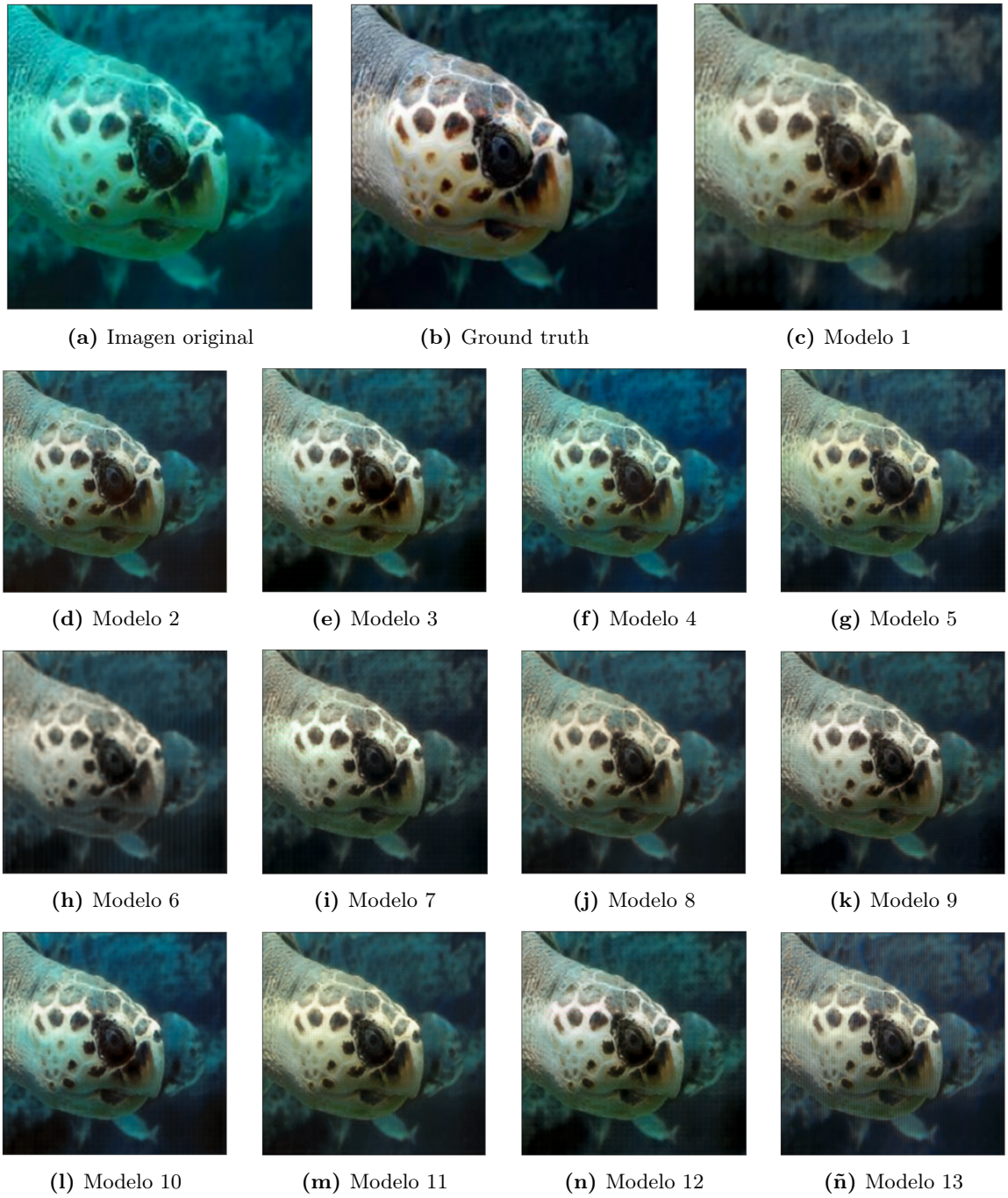
Model ID	MSE test	MAE test	PCC	SSIM
1	0.009569	0.067752	0.914016	0.786425
2	0.008472	0.063244	0.922195	0.824236
3	0.008812	0.064670	0.922429	0.794851
4	0.009210	0.066235	0.912712	0.819085
5	0.010107	0.069004	0.907456	0.803591
6	0.011876	0.077881	0.885512	0.647095
7	0.010517	0.070337	0.904492	0.776750
8	0.009634	0.068337	0.911337	0.798623
9	0.009954	0.068613	0.909313	0.794341
10	0.009143	0.066179	0.914125	0.794583
11	0.009976	0.068015	0.910186	0.813163
12	0.009202	0.065107	0.917253	0.802026
13	0.009962	0.069178	0.908995	0.781683

Tabla 6.1: Métricas por modelo en test.

Model ID	MSE CLAHE	MSE PRED	MAE CLAHE	MSE PRED
1	0.016602	0.003092	0.101041	0.042116
2	0.016602	0.002672	0.101041	0.038769
3	0.016602	0.003192	0.101041	0.042452
4	0.016602	0.002796	0.101041	0.039389
5	0.016602	0.002850	0.101041	0.040479
6	0.016602	0.004594	0.101041	0.051245
7	0.016602	0.003466	0.101041	0.044896
8	0.016602	0.003122	0.101041	0.042709
9	0.016602	0.003000	0.101041	0.042444
10	0.016602	0.003022	0.101041	0.040777
11	0.016602	0.002875	0.101041	0.039985
12	0.016602	0.002910	0.101041	0.040822
13	0.016602	0.003097	0.101041	0.042842

Tabla 6.2: Métricas utilizando CLAHE.

Las métricas de comportamiento de los modelos en *test* son similares a *train* y, entre ellas, son bastante parecidas también. La variación de valores de los errores se mantiene en intervalos muy pequeños, lo que sugiere que las imágenes que generan, desde un punto de vista puramente numérico, serían similares. No obstante, visualmente sí mantienen algunas diferencias cuando se comparan entre ellas.



(a) Imagen original **(b)** Ground truth **(c)** Modelo 1

(d) Modelo 2 **(e)** Modelo 3 **(f)** Modelo 4 **(g)** Modelo 5

(h) Modelo 6 **(i)** Modelo 7 **(j)** Modelo 8 **(k)** Modelo 9

(l) Modelo 10 **(m)** Modelo 11 **(n)** Modelo 12 **(ñ)** Modelo 13

Figura 6.1: Resultados visuales de imágenes en test

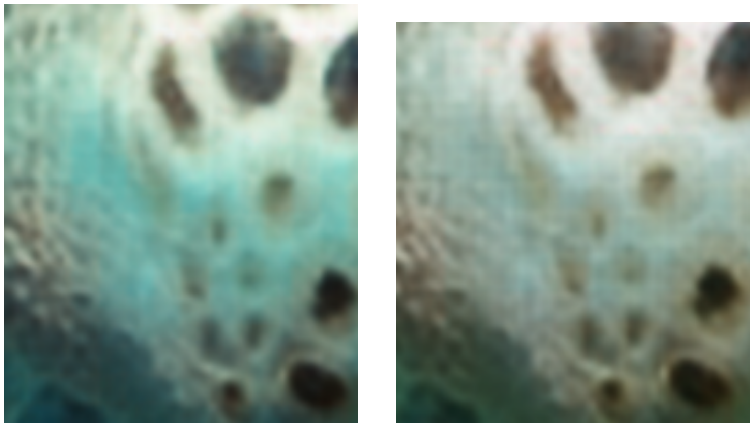
Un ejemplo de las diferencias visuales que se han comentado previamente puede verse en la figura 6.2, donde la imagen generada por el modelo 10 (figura 6.2-a) presenta un mayor corrimiento del azul frente a la imagen generada por el modelo 12 (figura 6.2-b).

Para explicitar un poco más, el corrimiento del azul refiere a un fenómeno óptico que se sucede en la propagación de la luz en el agua, causado por el efecto Doppler [65]. El agua actúa a modo de filtro y absorbe ciertos colores cuyas longitudes de onda son más cortas, como el azul en este caso. En las

tomas submarinas es un efecto muy común, sucediendo también con el verde. Los colores más cálidos como el rojo o el anaranjado suelen penetrar a más profundidad que estos colores.

Estas dos imágenes no pueden ser utilizadas para afirmar ninguna generalización, pero sí es observable que el modelo 10 es menos eficiente en la atención del error causado por la pérdida de frecuencia, en comparación con el modelo 12 que ha sido capaz de eliminarlo en gran medida. Como se puede apreciar en las imágenes de la figura 6.1, la imagen 6.1-a muestra la entrada de la red y 6.1-b su *ground truth*, contra la que se realiza la medición de errores posteriores. Como estos modelos se caracterizan por ser de aprendizaje supervisado, se les ha suministrado la salida esperada contra la cual validar el error y corregir sus pesos.

En la figura 6.1 pueden verse algunos de los problemas de visión submarina que se han comentado al comienzo de este trabajo. Por un lado, puede verse, como se ha mencionado antes, el efecto Doppler en acción cuando las frecuencias de azul y rojo se han ido perdiendo antes en el agua. Por otro lado, el fondo se presenta más oscuro y la cabeza de la tortuga, al estar en primer plano presenta más luz, lo que sugiere la presencia del efecto *vignetting*. Las imágenes presentan atenuación a muy poca distancia de la tortuga, generando un entorno más difuso y menos proclive a una ganancia en nitidez. En otros ejemplos, sobre todo en el conjunto de datos originales llamado *underwater_dark*, estos efectos son muy claros. Otro ejemplo con los errores problemente más evidentes se presentó en la figura 3.2.



(a) Fragmento del modelo 10 **(b)** Fragmento del modelo 12

Figura 6.2: Corrimiento de color azul

Partiendo de los datos de la tabla 6.3, se tendría que la imagen 6.1-n, generada por el modelo 12 es la más próxima en términos de métricas. También es fácil apreciar que la imagen 6.1-h, generada por el modelo 6, es la que más lejos está de aproximarse a lo que se busca. De todas formas, visualmente y a la escala que este documento es capaz de ilustrar, no son apreciables los detalles que hacen visibles la distinción entre todas las figuras generadas por todos los modelos.

Tomando estas imágenes como un ejemplo particular, se obtienen las métricas asociadas a la generación, disponibles en la tabla 6.3. De ellas se desprende que, a partir de los errores, quien mejor ha aproximado su *ground truth* ha sido el modelo 12. Esto no quita que, en la media, el modelo 2 sea quien mejor

generaliza y reconstruye las imágenes.

Model ID	MSE	MAE	PCC	SSIM	MSE CLAHE	MAE CLAHE
1	0.006815	0.068244	0.966855	0.740031	This model	This model
2	0.006312	0.098746	0.969733	0.754184	This model	This model
3	0.004967	0.058029	0.970828	0.768321	This model	This model
4	0.008611	0.076969	0.946401	0.754439	This model	This model
5	0.006557	0.069852	0.970775	0.749709	This model	This model
6	0.006783	0.071065	0.967806	0.668779	This model	This model
7	0.004482	0.054452	0.973974	0.769541	This model	This model
8	0.006683	0.069413	0.968624	0.744170	This model	This model
9	0.004153	0.053472	0.976663	0.772169	This model	This model
10	0.006886	0.069501	0.957466	0.756048	This model	This model
11	0.005364	0.062542	0.974411	0.756538	This model	This model
12	0.003929	0.052944	0.977631	0.775313	This model	This model
13	0.007061	0.071555	0.968245	0.701254	This model	This model

Tabla 6.3: Métricas del caso de test.

6.2. Análisis con CLAHE

Con respecto a la comparación realizada utilizando CLAHE, de la misma forma que en los resultados de *test*, para esta imagen los errores siguen el mismo patrón descrito previamente.

Las diferencias entre la imagen original y su *ground truth*, ambas en escala de grises frente a la imagen regenerada y su *ground truth*, en escala de grises también, es más próxima en todos los modelos generados siempre que semida con MSE. De la misma forma sucede con MAE, donde los modelos superan ampliamente la utilización de CLAHE.

Las métricas que se obtienen en general de los modelos podrían sugerir que los modelos 2 y 4 son los candidatos a modelo final. Ambos tienen las mejores métricas aunque en la imagen generada de prueba, el modelo 2 parece ajustar mejor. Como el entorno de *test* es donde los modelos demuestran qué tan bien podrían funcionar en entornos reales, el modelo final sería el 2.

6.3. Comportamiento con imágenes externas

Finalmente, previendo estos casos futuros donde las imágenes no pertenecen al dataset, se genera un juego de datos de prueba que permitan visualizar el comportamiento con datos que nunca antes han visto. En la figura 6.3 se han extraído los resultados de los modelos 2 y 6, mejor y peor modelo respectivamente, de forma que permita una rápida visualización de las imágenes regeneradas por cada uno.

(a) Imagen original 1 **(b)** Reconstrucción Modelo 2 **(c)** Reconstrucción Modelo 6

(d) Imagen original 2 **(e)** Reconstrucción Modelo 2 **(f)** Reconstrucción Modelo 6

(g) Imagen original 3 **(h)** Reconstrucción Modelo 2 **(i)** Reconstrucción Modelo 6

(j) Imagen original 4 **(k)** Reconstrucción Modelo 2 **(l)** Reconstrucción Modelo 6

Figura 6.3: Resultados visuales de imágenes externas

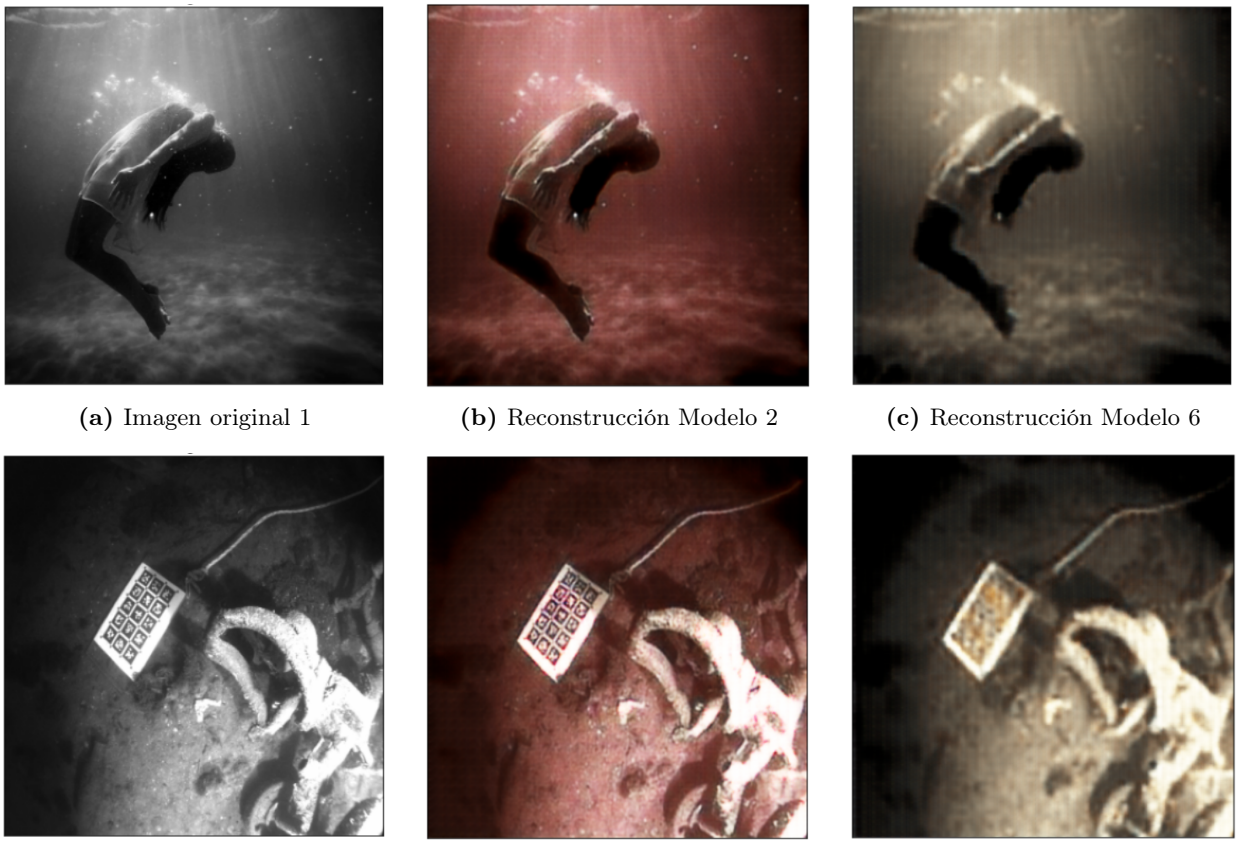
Las reconstrucciones del modelo 2 muestran de forma visual resultados más claros y nítidos que los del modelo 6, lo cual tiene todo el sentido según las métricas obtenidas de los modelos previamente.

Los resultados completos de todos los modelos pueden verse en el anexo B.

A partir de las imágenes presentes en la figura 6.3, puede verse fácilmente que el modelo 2 es significativamente superior, aunque por descontado aún sería conveniente ir un poco más allá con su entrenamiento. El primer detalle visual que puede apreciarse es que no es capaz de predecir bien el azul de fondo, mientras que sí ha aprendido a mejorar la calidad y rebajar la frecuencia de verdes.

6.4. Imágenes en escala de grises

Es importante recalcar nuevamente que no es responsabilidad explícita de estos modelos aprender a colorear perfectamente, sino mejorar la calidad en términos generales. Sin embargo, en muchos de los casos sí han aprendido a hacerlo. A modo de ejemplo, el anaranjado es un color que puede predecir y lo acentúa en los resultados.



(a) Imagen original 1 **(b)** Reconstrucción Modelo 2 **(c)** Reconstrucción Modelo 6

(d) Imagen original 2 **(e)** Reconstrucción Modelo 2 **(f)** Reconstrucción Modelo 6

Figura 6.4: Comportamiento en escala de grises

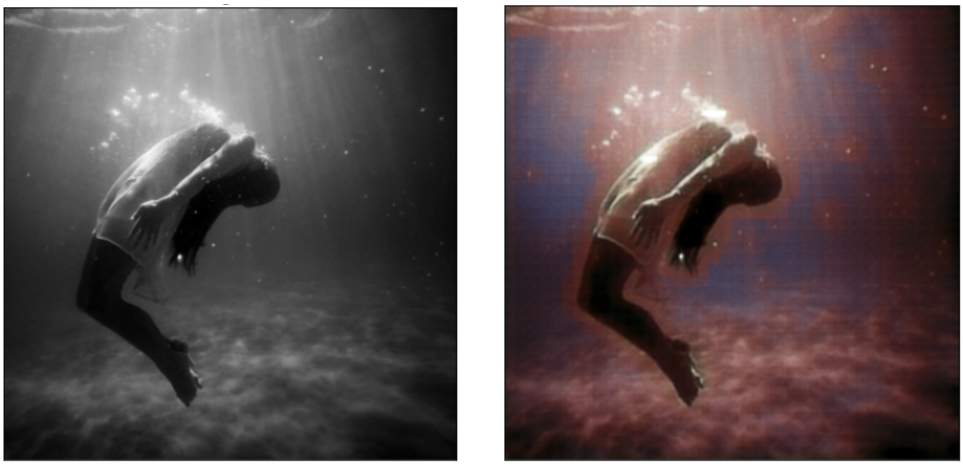
Tomando en cuenta los resultados que ha obtenido aquí, hay un caso especial que no se ha contemplado

como tal, sino que se busca medir el comportamiento del modelo cuando ocurre. Este caso se da cuando recibe imágenes en escala de grises.

Como las imágenes en escala de grises tienen la particularidad de tener un único canal, a diferencia de las imágenes en color que tienen 3 (o eventualmente 4 si incluyen *alpha*), la solución ha sido triplicar la misma capa para crear una imagen en tres canales. Al ser esto el input de los modelos generados, buscará no solamente mejorar la imagen, sino además colorearla bajo los criterios que ha aprendido.

Las imágenes probadas en escala de grises pueden verse en la figura 6.4, donde puede verse el comportamiento de los modelos 2 y 6 en acción. El modelo 2 que intenta mejorar la calidad con los mejores resultados obtenidos por los 13 modelos entrenados, tiende a tintar de rojo la imagen en escala de grises. Una característica que sí puede apreciarse visualmente es que los modelos han aprendido a iluminar mejor las imágenes, mejorando así el contraste respecto de la figura original.

No obtante, sabiendo que la chica de la figura 6.5-a está sumergida, se asume que el color que debería rodearla es una variante de azul. El único que ha conseguido aproximar un poco este color es el modelo 4. Es decir, que de cara a imágenes en escala de grises, podría ser este una buena alternativa si se entrena un poco más.



(a) Imagen original 1 **(b)** Reconstrucción Modelo 4

Figura 6.5: Comportamiento del modelo 4 prediciendo azul

Una alternativa para sí asignarle la responsabilidad a los modelos de entrenar y aprender a colorear podría ser, además del juego de datos como tal usado, añadir las mismas imágenes en escala de grises y su *ground truth* para que así, conforme avanza el entrenamiento, sea capaz de predecir los colores.

Capítulo 7

Conclusiones y trabajo futuro

Este proyecto ha tenido por objetivo la reconstrucción mejorada de imágenes submarinas, que permita, al menos en parte, solventar los problemas típicos de este entorno. Los modelos resultantes del entrenamiento han demostrado que la arquitectura *Encoder-Decoder* es altamente eficiente para abordar los desafíos asociados al procesamiento de este tipo de imágenes. Los resultados obtenidos respaldan el uso del *deep learning* como una herramienta poderosa de cara al tratamiento de problemas asociados a esta problemática de visión por computador.

El conjunto de datos utilizado presenta un juego de imágenes submarinas y sus respectivos *ground truths*. Esto ha permitido entrenar los modelos bajo un paradigma de aprendizaje supervisado, capaz de aproximar de manera significativa la reconstrucción mejorada en su capacidad de generalización en unas pocas *epochs*. El uso de la arquitectura *Encoder-Decoder* ha facilitado la resolución de algunos problemas en este entorno como ser falta de luz, ruido, pérdida de frecuencia de colores como el rojo típicamente, corrimiento del azul, falta de contraste, entre otros.

Se han realizado las evaluaciones correspondientes, utilizando algunas métricas que permiten medir de forma cuantitativa la distancia a las imágenes reconstuidas. Los valores que se han obtenidos son buenos en general, lo que se traduce en que los resultados visuales deberían tener una aproximación suficiente a lo que se espera. Desde un punto de vista cualitativo, las imágenes externas generadas a través de los modelos enternados tienen también resultados considerables. En este caso, no es posible realizar métricas puesto que no se tienen imágenes con las que medir esa diferencia.

Los trece modelos entrenados han permitido explorar distintas configuraciones de hiperparámetros, obteniendo una serie de resultados, algunos de ellos con una ganancia en términos de calidad bastante buena. Esto es, además, la base sobre la que seguir trabajando en la búsqueda de nuevas configuraciones capaces de obtener aún mejores resultados posibles. Hay disponibles una serie de herramientas como ser *transfer learning* o *skip connections*, que permitirían un gran abanico de pruebas, que por varios motivos, siendo el tiempo como principal de ellos, trascienten a este trabajo.

De cara al trabajo a futuro, se ha considerado que el módulo previsto para compartimentar y disponibilizar la funcionalidad aquí desarrollada por medio de un *docker*, pasara a formar parte de tareas de trabajo futuro. No obstante, ha quedado disponible de forma pública en el repositorio de GitHub, el notebook y los datos necesarios para la utilización del código aquí desarrollado, bajo una licencia Creative-Commons.

En conclusión, este proyecto busca demostrar el gran potencial del *deep learning* en el procesamiento de imágenes submarinas. Basándose en los resultados obtenidos y dejando una serie de consideraciones a

futuro, se abren nuevas líneas de trabajo que podrían ofrecer avances significativos en el procesamiento de imágenes en estos entornos. Las particularidades del entorno sugieren, por ejemplo, un foco importante en el procesamiento de imágenes con muy baja luz, puesto que a grandes profundidades es una de los grandes escollos en el reconocimiento de espacios. El uso de fuentes lumínicas artificiales acarrean problemas como el *vignetting* que invita a un nuevo estudio más profundo sobre su tratamiento.

La exploración de nuevas configuraciones e incluso la propia explotación de estas mismas, podría llevar al desarrollo de soluciones aplicables a entornos productivos. Estas soluciones podrían aportar mejoras a la investigación y exploración submarina, así como el monitoreo del medio ambiente haciendo al menos una pequeña contribución a los Objetivos de Desarrollo Sostenibles (ODS).

Desde un punto de vista personal, este proyecto abre una puerta al mundo del *deep learning* y de visión por computador, permitiendo conocer distintas aplicaciones y alternativas de uso de estas herramientas. El crecimiento inexorable de modelos capaces de interactuar con el entorno o, cuando menos, de facilitar los recursos para el trabajo humano en él, facilita tomar como objetivo una formación aún más allá de cara a un doctorado.

Bibliografía

[1] Hamed Habibi Aghdam y Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks*. New York, United States: Springer Publishing, 2017.

[2] Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 3rd ed. O'Reilly Media, nov. de 2022.

[3] Joseph Howse y Joe Minichino. *Learning OpenCV 5 Computer Vision with Python: Tackle computer vision and machine learning with the newest tools, techniques and a algorithms, 4th Edition*. Packt Publishing - ebooks Account, mar. de 2023.

[4] Fernando Martínez-Plumed et al. «CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories». En: *IEEE Transactions on Knowledge and Data Engineering* 33 (2021), págs. 3048-3061.

[5] Jojo Moolayil. *Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python*. Apress, dic. de 2018.

[6] Derrick Mwiti. *Deep learning with TensorFlow and Keras*. Independently published, nov. de 2022.

[7] TensorFlow Team. *Empiece a utilizar TensorBoard*. Ene. de 2022. URL: https://www.tensorflow.org/tensorboard/get_started?hl=es-419.

[8] Sebastian Raschka et al. *Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python (English Edition)*. 1.ª ed. Packt Publishing, feb. de 2022.

[9] Adrian Rosebrock et al. *OCR with OpenCV, Tesseract, and Python*. 1.ª ed. PyImageSearch, 2020.

[10] Durga Nooka Venkatesh Alla et al. «Vision-based Deep Learning algorithm for Underwater Object Detection and Tracking». En: *OCEANS 2022 - Chennai*. 2022, págs. 1-6. DOI: 10.1109/OCEANSChennai45887.2022.9775438.

[11] Yaofeng Xie et al. «Lighting the darkness in the sea: A deep learning model for underwater image enhancement». En: *Frontiers in Marine Science* 9 (ago. de 2022). DOI: 10.3389/fmars.2022.921492. URL: http://dx.doi.org/10.3389/fmars.2022.921492.

[12] Abdelrahman Abdelhamed et al. «Extracting Vignetting and Grain Filter Effects from Photos». En: *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (ene. de 2022). DOI: 10.1109/wacv51458.2022.00013. URL: https://doi.org/10.1109/wacv51458.2022.00013.

[13] Kai Hu et al. «An Overview of Underwater Vision Enhancement: From Traditional Methods to Recent Deep Learning». En: *Journal of Marine Science and Engineering* 10.2 (feb. de 2022), pág. 241. ISSN: 2077-1312. DOI: 10.3390/jmse10020241. URL: http://dx.doi.org/10.3390/jmse10020241.

[14] Dorotea Potoc y Davor Petrinovic. «Creating a synthetic image for evaluation of vignetting modeling and estimation». En: *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)* (mayo de 2022). DOI: 10.23919/mipro55190.2022.9803709. URL: http://dx.doi.org/10.23919/mipro55190.2022.9803709.

[15] Xiaoxiang Yuan et al. «Vignetting Correction of Post-Earthquake UAV Images». En: *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium* (jul. de 2018). DOI: 10.1109/igarss.2018.8517825. URL: http://dx.doi.org/10.1109/igarss.2018.8517825.

[16] Yan Mo et al. «A Robust UAV Hyperspectral Image Stitching Method Based on Deep Feature Matching». En: *IEEE Transactions on Geoscience and Remote Sensing* 60 (2022), págs. 1-14. DOI: 10.1109/tgrs.2021.3123980. URL: http://dx.doi.org/10.1109/tgrs.2021.3123980.

[17] Derya Akkaynak y Tali Treibitz. «Sea-Thru: A Method for Removing Water From Underwater Images». En: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (jun. de 2019). DOI: 10.1109/cvpr.2019.00178. URL: http://dx.doi.org/10.1109/cvpr.2019.00178.

[18] D. B. Marco, A. J. Healey y R. B. McGhee. «Autonomous underwater vehicles: Hybrid control of mission and motion». En: *Autonomous Robots* 3.2-3 (1996), págs. 169-186. DOI: 10.1007/bf00141153. URL: http://dx.doi.org/10.1007/bf00141153.

[19] Carlos C. Insaurralde. «Autonomic management for the next generation of Autonomous Underwater Vehicles». En: *2012 IEEE/OES Autonomous Underwater Vehicles (AUV)* (sep. de 2012). DOI: 10.1109/auv.2012.6380726. URL: http://dx.doi.org/10.1109/auv.2012.6380726.

[20] Yunyun Dong et al. «Local Deep Descriptor for Remote Sensing Image Feature Matching». En: *Remote Sensing* 11.4 (feb. de 2019), pág. 430. DOI: 10.3390/rs11040430. URL: http://dx.doi.org/10.3390/rs11040430.

[21] Hao Ye et al. «Deep Learning-Based End-to-End Wireless Communication Systems With Conditional GANs as Unknown Channels». En: *IEEE Transactions on Wireless Communications* 19.5 (feb. de 2020), págs. 3133-3143. DOI: 10.1109/twc.2020.2970707. URL: https://ieeexplore.ieee.org/ielx7/7693/9090043/08985539.pdf.

[22] Martin Arjovsky, Soumith Chintala y Léon Bottou. «Wasserstein GAN». En: *arXiv (Cornell University)* (ene. de 2017). URL: https://arxiv.org/pdf/1701.07875.

[23] Laura Martinez Molera. *Synthetic Image Generation using GANs*. Dic. de 2021. URL: https://blogs.mathworks.com/deep-learning/2021/12/02/synthetic-image-generation-using-gans/.

[24] Guillermo Iglesias Hernández y Edgar Talavera Muñoz. *CómicGAN: Generación de ilustraciones con redes GAN de crecimiento progresivo*. 2021.

[25] Felipe Esteban Silva Piña. *Detección automática de grietas basada en imágenes de puentes de concreto a través de modelos Encoder-Decoder*. 2021.

[26] Anna Bosch Rué, Jordi Casas Roma y Toni Lozano Bagén. *Deep Learning*. Editorial UOC, jul. de 2019.

[27] Roberto Perkins. «Caltech Researchers Help Generate First Image of Black Hole at the Center of Our Galaxy». En: (mayo de 2022). URL: https://www.caltech.edu/about/news/caltech-researchers-help-generate-first-image-of-black-hole-at-the-center-of-our-galaxy.

[28] Vijay Badrinarayanan, Alex Kendall y Roberto Cipolla. «SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (dic. de 2017), págs. 2481-2495. DOI: 10.1109/tpami.2016.2644615. URL: http://dx.doi.org/10.1109/tpami.2016.2644615.

[29] Robail Yasrab. «ECRU: An Encoder-Decoder Based Convolution Neural Network (CNN) for Road-Scene Understanding». En: *Journal of Imaging* 4.10 (oct. de 2018), pág. 116. DOI: 10.3390/jimaging4100116. URL: http://dx.doi.org/10.3390/jimaging4100116.

[30] Kyunghyun Cho. «On the Properties of Neural Machine Translation: Encoder-Decoder Approaches». En: (sep. de 2014). URL: https://arxiv.org/abs/1409.1259.

[31] Pankaj Malhotra. «LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection». En: (jul. de 2016). URL: https://arxiv.org/abs/1607.00148.

[32] Liang-Chieh Chen. «Encoder-Decoder with Atrous Separable Convolution for Semantic...» En: (feb. de 2018). URL: https://arxiv.org/abs/1802.02611.

[33] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[34] Dylan Snover et al. «Deep Clustering to Identify Sources of Urban Seismic Noise in Long Beach, California». En: *Seismological Research Letters* 92 (dic. de 2020). DOI: 10.1785/0220200164.

[35] Jonathan Masci et al. «Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction». En: *Artificial Neural Networks and Machine Learning – ICANN 2011*. Ed. por Timo Honkela et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, págs. 52-59. ISBN: 978-3-642-21735-7.

[36] Md Jahidul Islam, Youya Xia y Junaed Sattar. «Fast Underwater Image Enhancement for Improved Visual Perception». En: *IEEE Robotics and Automation Letters (RA-L)* 5.2 (2020), págs. 3227-3234.

[37] Maxime Ferrera et al. «AQUALOC: An underwater dataset for visual–inertial–pressure localization». En: *The International Journal of Robotics Research* 38.14 (oct. de 2019), págs. 1549-1559. DOI: 10.1177/0278364919883346.

[38] Antoni Burguera. «Lightweight Underwater Visual Loop Detection and Classification using a Siamese Convolutional Neural Network». En: *IFAC-PapersOnLine* 54.16 (ene. de 2021), págs. 410-415. DOI: 10.1016/j.ifacol.2021.10.124. URL: https://doi.org/10.1016/j.ifacol.2021.10.124.

[39] Francisco Charte. «Autoencoders ¿Qué son, para qué sirven y cómo funcionan?» En: (ene. de 2021). URL: http://cemixugrmadoc.ugr.es/pages/10-banners/siadef/sesion14_transparencias/.

[40] Keras Team. *Keras documentation: EarlyStopping*. URL: https://keras.io/api/callbacks/early_stopping/.

[41] Cosmin Ancuti, Christophe De Vleeschouwer y Philippe Bekaert. «Color Balance and Fusion for Underwater Image Enhancement». En: *IEEE Transactions on Image Processing* 27.1 (ene. de 2018), págs. 379-393. DOI: 10.1109/tip.2017.2759252. URL: https://doi.org/10.1109/tip.2017.2759252.

[42] Dana Berman et al. «Underwater Single Image Color Restoration Using Haze-Lines and a New Quantitative Dataset». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (ago. de 2021), pág. 1. DOI: 10.1109/tpami.2020.2977624. URL: https://doi.org/10.1109/tpami.2020.2977624.

[43] Keming Cao, Yan-Tsung Peng y Pamela C. Cosman. «Underwater Image Restoration using Deep Networks to Estimate Background Light and Scene Depth». En: abr. de 2018. DOI: 10.1109/ssiai.2018.8470347. URL: https://doi.org/10.1109/ssiai.2018.8470347.

[44] James Chiang y Ying-Ching Chen. «Underwater Image Enhancement by Wavelength Compensation and Dehazing». En: *IEEE transactions on image processing* 21.4 (abr. de 2012), págs. 1756-1769. DOI: 10.1109/tip.2011.2179666. URL: https://doi.org/10.1109/tip.2011.2179666.

[45] Cicero Dos Santos. *Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts*. Ago. de 2014. URL: https://aclanthology.org/C14-1008/.

[46] Zhenqi Fu. *Underwater Image Enhancement via Learning Water Type Desensitized Representations*. Feb. de 2021. URL: http://arxiv.org/abs/2102.00676.

[47] Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton. «ImageNet classification with deep convolutional neural networks». En: *Communications of The ACM* 60.6 (mayo de 2017), págs. 84-90. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[48] Yann LeCun, Yoshua Bengio y Geoffrey E. Hinton. «Deep learning». En: *Nature* 521.7553 (mayo de 2015), págs. 436-444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[49] Chongyi Li, Saeed Anwar y Fatih Porikli. «Underwater scene prior inspired deep underwater image and video enhancement». En: *Pattern Recognition* 98 (feb. de 2020), pág. 107038. DOI: 10.1016/j.patcog.2019.107038. URL: https://doi.org/10.1016/j.patcog.2019.107038.

[50] C. J. Prabhakar y P. U. Praveen Kumar. «An Image Based Technique for Enhancement of Underwater Images». En: *arXiv (Cornell University)* (dic. de 2012). DOI: 10.48550/arxiv.1212.0291. URL: http://arxiv.org/abs/1212.0291.

[51] Ahmad Salman et al. «Fish species classification in unconstrained underwater environments based on deep learning». En: *Limnology and Oceanography-methods* 14.9 (sep. de 2016), págs. 570-585. DOI: 10.1002/lom3.10113. URL: https://doi.org/10.1002/lom3.10113.

[52] Kuldeep Singh y Rajiv Kapoor. «Image enhancement using Exposure based Sub Image Histogram Equalization». En: *Pattern Recognition Letters* 36 (ene. de 2014), págs. 10-14. DOI: 10.1016/j.patrec.2013.08.024. URL: https://doi.org/10.1016/j.patrec.2013.08.024.

[53] Xiwen Deng et al. «An underwater image enhancement model for domain adaptation». En: *Frontiers in Marine Science* 10 (abr. de 2023). DOI: 10.3389/fmars.2023.1138013. URL: https://doi.org/10.3389/fmars.2023.1138013.

[54] Xunxiang Yao et al. «Underwater Image Enhancement Using Deep Transfer Learning Based on a Color Restoration Model». En: *IEEE Journal of Oceanic Engineering* 48.2 (ene. de 2023), págs. 489-514. DOI: 10.1109/joe.2022.3227393. URL: https://doi.org/10.1109/joe.2022.3227393.

[55] Naresh Kumar et al. «Underwater Image Enhancement using deep learning». En: *Multimedia Tools and Applications* (mayo de 2023). DOI: 10.1007/s11042-023-15525-4. URL: https://doi.org/10.1007/s11042-023-15525-4.

[56] Cristo Jurado-Verdu et al. «Convolutional autoencoder for exposure effects equalization and noise mitigation in optical camera communication». En: *Optics Express* 29.15 (jul. de 2021), pág. 22973. DOI: 10.1364/oe.433053. URL: https://doi.org/10.1364/oe.433053.

[57] Mizuho Nishio et al. «Convolutional auto-encoder for image denoising of ultra-low-dose CT». En: *Heliyon* 3.8 (ago. de 2017), e00393. DOI: 10.1016/j.heliyon.2017.e00393. URL: https://doi.org/10.1016/j.heliyon.2017.e00393.

[58] Yann LeCun et al. «Gradient-based learning applied to document recognition». En: *Proceedings of the IEEE* 86.11 (ene. de 1998), págs. 2278-2324. DOI: 10.1109/5.726791. URL: https://doi.org/10.1109/5.726791.

[59] Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». En: vol. 25. Dic. de 2012, págs. 1097-1105. URL: http://books.nips.cc/papers/files/nips25/NIPS2012_0534.pdf.

[60] Jawadul H. Bappy et al. «Hybrid LSTM and Encoder–Decoder Architecture for Detection of Image Forgeries». En: *IEEE transactions on image processing* 28.7 (ene. de 2019), págs. 3286-3300. DOI: 10.1109/tip.2019.2895466. URL: https://doi.org/10.1109/tip.2019.2895466.

[61] Hu Chen et al. «Low-Dose CT With a Residual Encoder-Decoder Convolutional Neural Network». En: *IEEE Transactions on Medical Imaging* 36.12 (jun. de 2017). DOI: 10.1109/tmi.2017.2715284. URL: https://doi.org/10.1109/tmi.2017.2715284.

[62] Hui Zou y Trevor Hastie. «Regularization and variable selection via the elastic net». En: *Journal of The Royal Statistical Society Series B-statistical Methodology* 67.2 (abr. de 2005), págs. 301-320. DOI: 10.1111/j.1467-9868.2005.00503.x. URL: https://doi.org/10.1111/j.1467-9868.2005.00503.x.

[63] Naftaly Wambugu et al. «A hybrid deep convolutional neural network for accurate land cover classification». En: *International journal of applied earth observation and geoinformation* 103 (dic. de 2021), pág. 102515. DOI: 10.1016/j.jag.2021.102515. URL: https://doi.org/10.1016/j.jag.2021.102515.

[64] Aston Zhang et al. «Dive into Deep Learning». En: *arXiv preprint arXiv:2106.11342* (2021).

[65] Antonio Arias Ruano et al. *El efecto Doppler y el corrimiento al rojo y al azul*. URL: https://www.ucm.es/data/cont/docs/136-2015-01-27-El%5C%20efecto%5C%20Doppler.pdf.

[66] Pedro Querejeta Simbeni. «Procesamiento digital de imágenes». En: jul. de 2015. URL: http://lcr.uns.edu.ar/fvc/NotasDeAplicacion/FVC-QuerejetaSimbeniPedro.pdf.

[67] Yen Po-Wei et al. *Real-time Super Resolution CNN Accelerator with Constant Kernel Size Winograd Convolution*. Ago. de 2020. URL: https://ieeexplore.ieee.org/abstract/document/9073972.

[68] Paul Gavrikov. *visualkeras*. https://github.com/paulgavrikov/visualkeras. 2020.

[69] Lutz Roeder. *Netron app*. 2021. URL: https://netron.app/.

[70] Jordi De la Torre Gallart. *Modelos generativos*. Inf. téc. 2023.

[71] Zijun Zhang. «Improved Adam Optimizer for Deep Neural Networks». En: *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. 2018, págs. 1-2. DOI: 10.1109/IWQoS.2018.8624183.

Anexo A

Códigos fuente

A.1. Preprocesado y distribución de imágenes

En este anexo se muestra el código utilizado para la construcción y distribución de los juegos de datos, basados en las imágenes del *Dataset EUVP*[36].

```
!gdown 1rowzy-pSgYnmwN7Y16OtpnrgyDJYjuFD # Download dataset zip from Google Drive
if os.path.exists("./Paired.zip"): # Unzip downloaded file and remove
    !unzip -q -o Paired.zip -d dataset_euvp
    os.remove("./Paired.zip")

# General paths
general_path = "./dataset_euvp/Paired/"
datasets = ["underwater_scenes", "underwater_imagenet", "underwater_dark"]
dataset_path = "./dataset"
image_id = 1 # Var to rename each image

# Target directories
target_dirs = ["train", "test"]
target_subdirs = ["original", "enhanced"]

# Create or clean target directories
for target_dir in target_dirs:
    for target_subdir in target_subdirs:
        tmp_path = f"{dataset_path}/{target_dir}/{target_subdir}/"
        os.makedirs(tmp_path, exist_ok=True) # Create the directory tree if not exists
        for f in os.listdir(tmp_path):
            os.remove(os.path.join(tmp_path, f)) # Remove every previous file

for subdataset in datasets:
    # Source directories definition path
    source_path = f"{general_path}/{subdataset}"
    source_trainA, source_trainB = f"{source_path}/trainA/", f"{source_path}/trainB/"

    # Get source filenames to copy
    original_files = os.listdir(source_trainA)

    # Get 90% filenames to train and validation, and 10% to test
    total_90, total_10 = int(len(original_files) * 0.9), int(len(original_files) * 0.1)

    # Split the previous list in order to save into original and enhanced directories
    source_trainA_files_train = original_files[:total_90] # 90% train + validation
    source_trainA_files_test = original_files[total_90:] # 10% test

    for source_a in [ ["train", source_trainA_files_train], ["test", source_trainA_files_test]]:
        for file in tqdm(source_a[1], ncols=110, desc=f"Copying from {subdataset} to {source_a[0]}"):
```

```
            if os.path.exists(f"{source_trainB}{file}"): # Check if selected image has it enhanced
                shutil.copy(f"{source_trainA}{file}", f"{dataset_path}/{source_a[0]}/{target_subdirs[0]}/{
    ↪ image_id}.jpg") # Copy original
                shutil.copy(f"{source_trainB}{file}", f"{dataset_path}/{source_a[0]}/{target_subdirs[1]}/{
    ↪ image_id}.jpg") # Copy enhanced
                image_id += 1 # Next image value

# Check the number of images in each directory
to_list = []
for dirs in target_dirs:
    for subdir in target_subdirs:
        to_list.append([dirs, subdir, len(os.listdir(f"{dataset_path}/{dirs}/{subdir}"))])

print_table(["Dataset", "Class", "# of images"], to_list)
```

Programa A.1: Función de preprocessing de imágenes

El resultado de este preprocesado y distribución se imprime a continuación, en la figura A.1, mostrando además el número de imágenes presentes en cada directorio.

```
+---------+----------+-------------+
| Dataset |  Class   | # of images |
+---------+----------+-------------+
|  train  | original |    10291    |
|  train  | enhaced  |    10291    |
|   test  | original |     1144    |
|   test  | enhaced  |     1144    |
+---------+----------+-------------+
```

Figura A.1: Resultado del proceso de distribución.

A.2. Callback Plot Learning

En este anexo se adjunta el fragmento de código del *callback* utilizado para el seguimiento visual del entrenamiento, facilitando la evaluación de las métricas prácticamente a tiempo real. Este fragmento de *callback* fue tomado de la Práctica 1 de la asignatura *Deep Learning* de la UOC.

```
class PlotLearning(Callback):
    """ Callback to plot metrics during the training process """

    def __init__(self, showLR=False):
      self.showLR = showLR # show or not learning rate in each epoch

    def on_train_begin(self, logs={}):
        self.metrics = {}
        for metric in logs:
            self.metrics[metric] = []

    def on_epoch_end(self, epoch, logs={}):
        for metric in logs:
            if metric in self.metrics:
                self.metrics[metric].append(logs.get(metric))
            else:
                self.metrics[metric] = [logs.get(metric)]

        metric = [x for x in logs if ('val' not in x) and ('lr' not in x)]
        if self.showLR:
          metric.append('lr')

        f, axs = pyplot.subplots(1,len(metric),figsize=(10,4))
        clear_output(wait=True)
        for i,ax in enumerate(axs):
          ax.plot(range(1, epoch + 2), self.metrics[metric[i]],'o--',label=metric[i])
          try:
            ax.plot(range(1, epoch+2), self.metrics['val_' + metric[i]], 'o--', label='val_'+metric[i])
          except:
            pass
          ax.set_xlabel('# epochs')
          ax.set_ylabel(metric[i])
          ax.legend()
          ax.grid()

        pyplot.tight_layout()
        pyplot.show()
```

Programa A.2: Callback Plot Learning

A.3. Análisis con CLAHE

Uno de los análisis que se han realizado ha sido la medición de errores entre los modelos generados y el algoritmo CLAHE. La función a continuación, permite cuantificar esta diferencia entre, por un lado, lso *ground truths* de las imágenes, y por otro, la imagen reconstruida y la imagen original aplicándole CLAHE.

Esta diferencia bsuca media la diferencia estructural de las imágenes, evitando las diferencias en el coloreado, por eso se convierten las tres imágenes a escala de grises previo al cálculo.

```python
def diff_with_clahe(img_original, img_enhanced, img_predicted):
    num_pixels = img_original.shape[0] * img_original.shape[1]
    CLAHE = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))

    # Every image to grayscale
    img_enhanced_gray = cv2.cvtColor(np.uint8(img_enhanced * 255), cv2.COLOR_BGR2GRAY) /255.0
    img_predicted_gray = cv2.cvtColor(np.uint8(img_predicted * 255), cv2.COLOR_BGR2GRAY)/255.0
    img_original_gray = cv2.cvtColor(np.uint8(img_original * 255), cv2.COLOR_BGR2GRAY)

    # Apply CLAHE to original and normalize
    img_original_clahe_gray = CLAHE.apply(img_original_gray)/255.0
    img_original_gray = img_original_gray/255.0

    # If MSE_PRED<MSE_CLAHE: This Model is better than CLAHE
    MSE_PRED = mean_squared_error(img_predicted_gray, img_enhanced_gray)
    MSE_CLAHE = mean_squared_error(img_original_clahe_gray, img_enhanced_gray)
    MAE_PRED = mean_absolute_error(img_predicted_gray,img_enhanced_gray)
    MAE_CLAHE = mean_absolute_error(img_original_clahe_gray,img_enhanced_gray)

    return [MSE_PRED, MSE_CLAHE, MAE_PRED, MAE_CLAHE]
```

Programa A.3: Función de medición de errores contra CLAHE

A.4. Cálculo de métricas de todas las imágenes

De cara a obtener elementos que permitan el análisis y la decisión de qué modelo elegir, se implementa la función *calculate_ metrics()* que realiza el cálculo de las metricas definidas, reutilizando funciones *ad-hoc* o definidas en las propias bibliotecas

```
def calculate_metrics(allNames, dataset):

    models_metrics = {}
    for i, baseName in enumerate(allNames):
        models_metrics[baseName]={}

        theModel=AutoModel()
        theModel.load(baseName)

        # Calculate average pcc and ssim for all images
        pcc_list, ssim_list = [], []

        # CLAHE and Prediction comparision
        mse_clahe_list, mae_clahe_list, mse_pred_list, mae_pred_list = [], [], [], []

        # Errors in train are in each history model file
        if dataset == "test":
            mse_list, mae_list = [], []

        for f in tqdm(os.listdir(f"./dataset/{dataset}/original/"), ncols=110, desc=f"Processing {dataset}/#{i
    ↪ +1}"): # Source dataset
            # Read the images
            img_original = img_as_float(imread(f"./dataset/{dataset}/original/"+f))
            img_original = resize(img_original,(320,320)) # Change the image dims
            img_original = np.expand_dims(img_original, axis=0) # Add a new dim

            img_enhanced = img_as_float(imread(f"./dataset/{dataset}/enhanced/"+f))
            img_enhanced = resize(img_enhanced,(320,320)) # Change the image dims
            img_enhanced = np.expand_dims(img_enhanced, axis=0) # Add a new dim

            stdout_default = sys.stdout
            sys.stdout = io.StringIO() # Hide output temporary
            img_predicted = theModel.predict(img_original)  # Assuming predict() returns the predicted image
            sys.stdout = stdout_default # Restore output again

            img_original = np.squeeze(img_original, axis=0) # Removethe added dim
            img_predicted = np.squeeze(img_predicted, axis=0) # Removethe added dim
            img_enhanced = np.squeeze(img_enhanced, axis=0) # Removethe added dim

            # Calculate Pearson
            pcc = np.corrcoef(img_enhanced.flatten(), img_predicted.flatten())[0][1]
            pcc_list.append(pcc)

            # Calculate Structural Similarity
            ssim = structural_similarity(img_enhanced, img_predicted, win_size=7, channel_axis=-1, data_range
    ↪ =1.0)
            ssim_list.append(ssim)

```

```
            # Calculate metrics with CLAHE
            clahe = diff_with_clahe(img_original, img_enhanced, img_predicted)
            mse_pred_list.append(clahe[0])
            mse_clahe_list.append(clahe[1])
            mae_pred_list.append(clahe[2])
            mae_clahe_list.append(clahe[3])

            # Update test metrics
            if dataset == "test":
                # Calculate MSE test
                mse = mean_squared_error(img_predicted.flatten(), img_enhanced.flatten())
                mse_list.append(mse)

                # Calculate MAE test
                mae = mean_absolute_error(img_predicted.flatten(), img_enhanced.flatten())
                mae_list.append(mae)

        # Update metrics values
        models_metrics[baseName]['PCC'] = np.mean(pcc_list)
        models_metrics[baseName]['SSIM'] = np.mean(ssim_list)

        models_metrics[baseName]['CLAHE'] = {}
        models_metrics[baseName]['CLAHE']['MSE'] = np.mean(mse_clahe_list)
        models_metrics[baseName]['CLAHE']['MAE'] = np.mean(mae_clahe_list)

        models_metrics[baseName]['PRED'] = {}
        models_metrics[baseName]['PRED']['MSE'] = np.mean(mse_pred_list)
        models_metrics[baseName]['PRED']['MAE'] = np.mean(mae_pred_list)


        # Save test errors
        if dataset == "test":
            models_metrics[baseName]['MSE'] = np.mean(mse_list)
            models_metrics[baseName]['MAE'] = np.mean(mae_list)

    with open(f'./models_metrics/models_metrics_{dataset}.pkl', 'wb') as f:
        pickle.dump(models_metrics, f) # Save metrics as pkl
```

Programa A.4: Función de cálculo de métricas

Esta función es llamada para cada dataset una vez por el costo en términos de tiempo que conlleva. Por eso, se verifica previamente que los ficheros *pkl* que contienen la información no estén presente antes de lanzarlas. La variable *allNames* contiene todos los modelos entrenados por el la aplicación previamente.

```
# Update metrics only if necessary, as it is a time-consuming process.
if not os.path.exists("./models_saved/models_metrics_train.pkl"):
    calculate_metrics(allNames, "train")

if not os.path.exists("./models_saved/models_metrics_test.pkl"):
    calculate_metrics(allNames, "test")
```

Programa A.5: Chequeo de métricas existentes

A.5. Obtención de métricas

A través de la función *returnMetrics()*, se procesan los ficheros cuyo contenido son las métricas de cada modelo, construyendo un JSON que contendrá toda esta información. Para cada uno de los modelos entrenados y en función del juego de datos que se quiera estudiar, prepara los datos y los devuelve según si se está analizando *train* o *test*.

```
def returnMetrics(baseName, modelId, dataset):
    """ Return a list with general metrics"""
    theModel=AutoModel()
    theModel.load(baseName)

    # Load PKL file
    with open(baseName+'_HISTORY.pkl', 'rb') as f:
        data_history = pickle.load(f)
    to_return = {}

    # Process information
    to_return['model_name'] = f"#{modelId}"
    data = baseName.replace("./models_saved/AUTOENCODER_", "")
    data = data.replace("EPOCHS", "")
    data = data.split("_")
    to_return['epochs'] = f"{len(data_history[0]['mse'])}/{data[-1]}"
    to_return['filters'] = [int(i) for i in data[:-1]]
    to_return['params'] = theModel.count_params()
    to_return['training_time'] = format_time(data_history[1])

    # Train errors in each model history
    if dataset == "train":
        to_return['mse_train'] = f"{data_history[0]['mse'][-1]:.6f}"
        to_return['mae_train'] = f"{data_history[0]['mae'][-1]:.6f}"
        to_return['mse_val'] = f"{data_history[0]['val_mse'][-1]:.6f}"
        to_return['mae_val'] = f"{data_history[0]['val_mae'][-1]:.6f}"

    # PCC & SSIM for every model in train and test pkg
    with open(f'./models_metrics/models_metrics_{dataset}.pkl', 'rb') as f:
        x = pickle.load(f)

    # Test errors in the same pkl
    if dataset == "test":
        to_return['mse_test'] = f"{x[baseName]['MSE']:.6f}"
        to_return['mae_test'] = f"{x[baseName]['MAE']:.6f}"

    # For everyone
    to_return['pcc'] = f"{x[baseName]['PCC']:.6f}"
    to_return['ssim'] = f"{x[baseName]['SSIM']:.6f}"
    to_return['CLAHE_DIFF'] = {}
    to_return['CLAHE_DIFF']['CLAHE'] = {}
    to_return['CLAHE_DIFF']['CLAHE']['MSE'] = f"{x[baseName]['CLAHE']['MSE']:.6f}"
    to_return['CLAHE_DIFF']['CLAHE']['MAE'] = f"{x[baseName]['CLAHE']['MAE']:.6f}"
    to_return['CLAHE_DIFF']['PRED'] = {}
    to_return['CLAHE_DIFF']['PRED']['MSE'] = f"{x[baseName]['PRED']['MSE']:.6f}"
    to_return['CLAHE_DIFF']['PRED']['MAE'] = f"{x[baseName]['PRED']['MAE']:.6f}"
```

```
47
48        # Return information
49        return to_return
50
```

Programa A.6: Función de devolución de métricas

A.6. Cálculo de diferencia entre imágenes individuales

Durante la etapa de validación de los resultados en *test*, donde se cuenta con los *ground truths* correspondientes, se calculan las diferencias una a una según la imagen que se esté procesando y el modelo con el que se esté prdiciendo cada vez. Esta función busca la reducción a una pequeña expresión que unifique las métricas que han de ser calculadas en esta etapa.

```
def img_differences(img_original, img_enhanced, img_predicted):
    # Check images have the same dimensions
    if img_original.shape != img_enhanced.shape and img_enhanced.shape != img_predicted.shape:
        text.red("Images have different dimensions.")
        return None

    mae = mean_absolute_error(img_enhanced.flatten(),img_predicted.flatten()) # Mean Absolute Error (MAE)
    mse = mean_squared_error(img_enhanced.flatten(), img_predicted.flatten()) # Mean Squared Error (MSE)
    pcc = np.corrcoef(img_enhanced.flatten(), img_predicted.flatten())[0][1] # Pearson Correlation Coefficient
    ssim = structural_similarity(img_enhanced, img_predicted, channel_axis=-1, data_range=1.0)
    clahe = diff_with_clahe(img_original, img_enhanced, img_predicted) # Get clahe measures (MSE/MAE)
    return {"mae":mae,
            "mse":mse,
            "pcc":pcc,
            "ssim":ssim,
            "clahe_mse_pred":clahe[0],
            "clahe_mse_clahe":clahe[1],
            "clahe_mae_pred":clahe[2],
            "clahe_mae_clahe":clahe[3]
           }
```

Programa A.7: Chequeo de métricas existentes

A.7. Visualización de imágenes para test y externas

La diferencia más significativa entre las imágenes de *test* y las externas es que las primeras tienen sus *ground truths* y, por lo tanto, se obtendrán sus métricas. Esta función permite la visualización de las imágenes en los directorios */dataset/test* y */external*. De esta forma, se reutiliza para el cálculo de las métricas o no.

```
def visualize_predictions(batchOriginal, batchEnhanced, batchPrediction, num_samples=1, show_metrics=True,
    ↪ model_name="Model", haveGroundTruths=True):
    n_cols = 3 if haveGroundTruths else 2
    fig, axis = pyplot.subplots(nrows=num_samples, ncols=n_cols, figsize=(25, 9*num_samples))
    text.bold(f"Model {model_name}")
    params = {'xticks':[], 'yticks':[]}
    axs_idx = 0
    for i in range(num_samples):
        ax_pos = 0
        out1, out2, out3 = img_as_float(batchOriginal[i]), img_as_float(batchEnhanced[i]), img_as_float(
    ↪ batchPrediction[i])
        if show_metrics: diff = img_differences (batchOriginal[i], batchEnhanced[i], batchPrediction[i])

        ax = axis[i][ax_pos]
        params['title'] = 'Original'
        ax.set(**params)
        ax.imshow(out1)
        ax_pos +=1

        if haveGroundTruths:
            ax = axis[i][ax_pos]
            params['title'] = 'Enhanced'
            ax.set(**params)
            ax.imshow(out2)
            ax_pos +=1

        ax = axis[i][ax_pos]
        params['title'] = 'Predicted'
        ax.set(**params)
        ax_pos +=1

        if show_metrics:
            best_filter_mse= "This Model" if diff['clahe_mse_pred'] < diff['clahe_mse_clahe'] else "CLAHE"
            best_filter_mae= "This Model" if diff['clahe_mae_pred'] < diff['clahe_mae_clahe'] else "CLAHE"
            ax.text(0.5, -0.34, f"MSE: {diff['mse']:.6f}\nMAE: {diff['mae']:.6f}\nPCC: {diff['pcc']:.6f}\nSSIM
    ↪ : {diff['ssim']:.6f}\n\nPRED vs. CLAHE\nMSE: {best_filter_mse}\nMAE: {best_filter_mae}", size=12, ha="
    ↪ center",  transform=ax.transAxes)
        ax.imshow(out3)

    pyplot.subplots_adjust(hspace=0.5) # Adjust vertical scale
    pyplot.show()
```

Programa A.8: Visualización de imágenes para test y externas

A.8. Visualización de arquitectura

A.8.1. Utilizando Visualkeras y plot_model

El objetivo de este programa es la generación de imágenes, que describan las distintas configuraciones de la arquitectura propuesta. Así pues, para cada uno de los modelos entrenados, se guardan las imágenes en 2D, 3D y su estructura en capas. En el nombre de la imagen, *XX* corresponde al número identificador del modelo entrenado.

- */models_images/model_XX_plot_2d.png* : imagen con la arquitectura en 2D.
- */models_images/model_XX_plot_3d.png* : imagen con la arquitectura en 3D.
- */models_images/model_XX_info.png* : imagen con las capas de la arquitectura.

A modo de ejemplo, tomando como referencia la configuración del modelo 2, se muestra el resultado de estas imágenes.

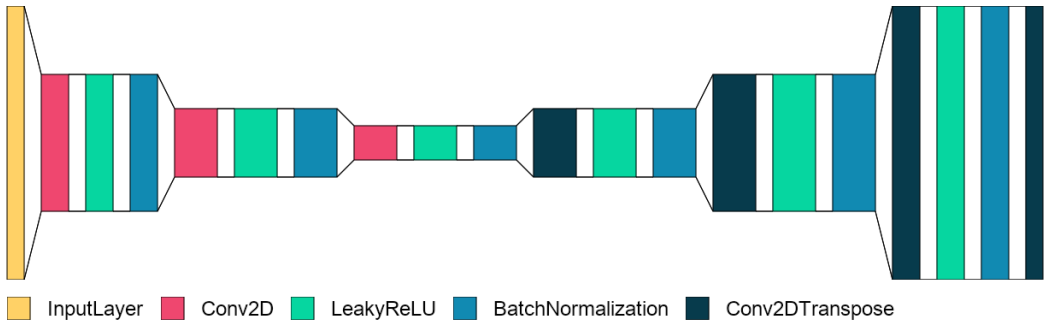


Figura A.2: Arquitectura de capas en 2D.

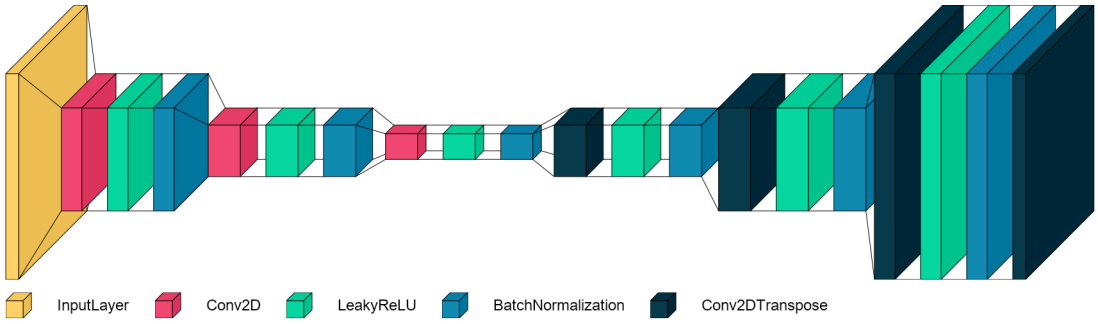


Figura A.3: Arquitectura de capas en 3D.

Los modelos son cargados por Keras mediante la función *load_ model*, mientras que para combinar ambas redes neuronales, es necesario procesar capa a capa de la segunda, ya que Keras por defecto entiende a este último como una única capa funcional. De ahí que en las líneas 5 a 7 del programa A.9 se desglose de esta forma.

```
1   def plotModel(i, baseName):
2   theModel_enc = load_model(f"{baseName}_ENC.h5")# Load encode model
3   theModel_dec = load_model(f"{baseName}_DEC.h5")# Load decode model
4
5   x = theModel_enc.output
6   for layer in theModel_dec.layers[1:]:
7       x = layer(x)
8
9   theModel = Model(inputs=theModel_enc.input, outputs=x)
10
11  # Plot layers with VisualKeras
12  font = ImageFont.truetype("fonts/arial.ttf", 24)
13  visualkeras.layered_view(theModel,
14                           to_file=f"models_images/model_{i+1}_plot_2d.png",
15                           draw_volume=False,
16                           legend=True,
17                           font = font,
18                           spacing=20,
19                           padding=20,
20                           scale_xy=1,
21                           scale_z=1,
22                           max_z=50)
23  visualkeras.layered_view(theModel,
24                           to_file=f"models_images/model_{i+1}_plot_3d.png",
25                           draw_volume=True,
26                           legend=True,
27                           font = font,
28                           spacing=40,
29                           padding=20,
30                           scale_xy=1,
31                           scale_z=1,
32                           max_z=50)
33
34  # Print information layers schema
35  plot_model(theModel,
36             to_file=f"models_images/model_{i+1}_info.png",
37             show_shapes=True,
38             show_layer_names=True)
39
40 for i, curName in enumerate(allNames):
41  plotModel(i, curName)
```

Programa A.9: Visualización de arquitectura de capas

A.8.2. Utilizando Netron

Alternativamente, es posible utilizar *Netron* [69] para la visualización de algunos modelos puntuales. Dado que es una aplicación externa, la apertura de todos estos modelos implican varias pestañas, por lo que su uso queda a consideración de lois modelos específicos que quieran visualizarse.

```
netron.start('models_saved/AUTOENCODER_128_128_16_EPOCHS30.h5')
```

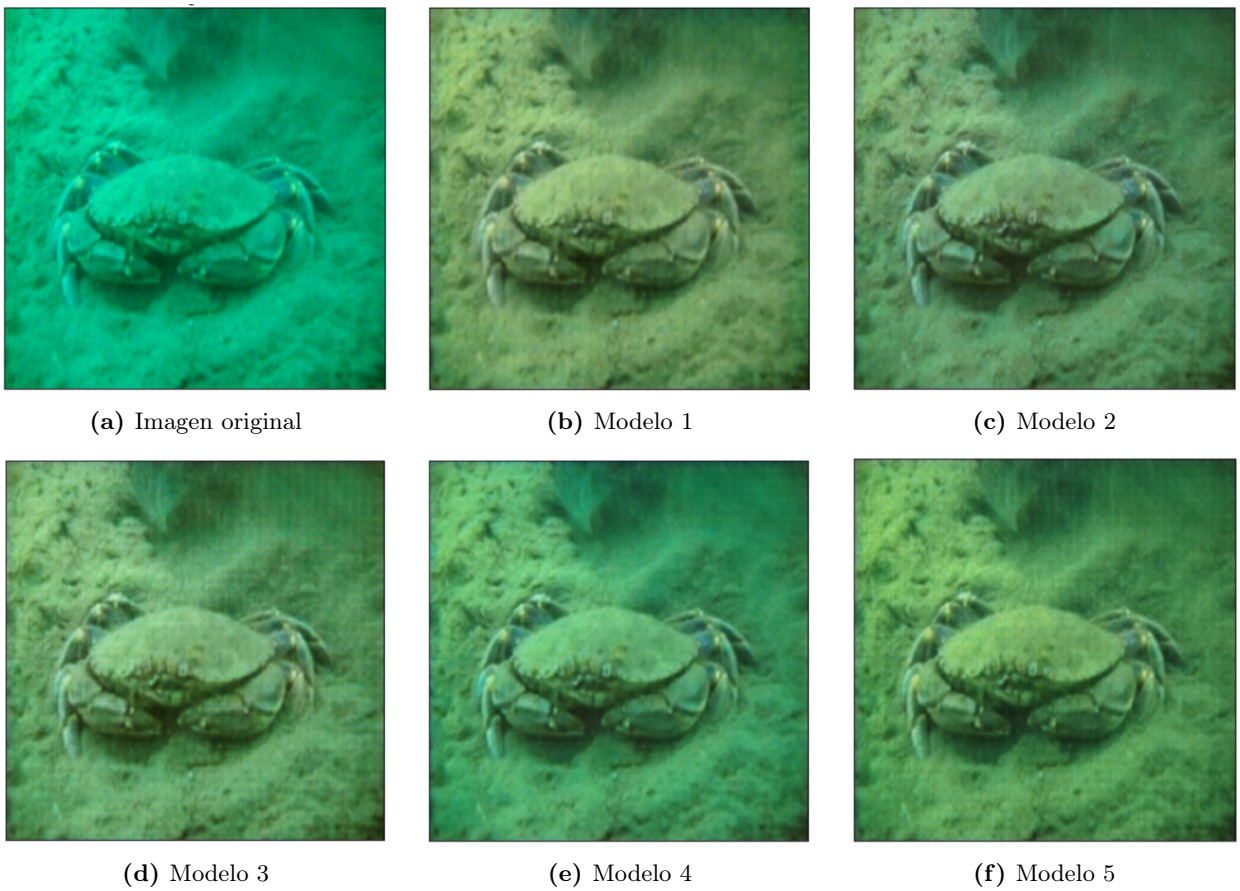
Programa A.10: Visualización de arquitectura con Netron

Anexo B

Resultados visuales

En este anexo, se presentan todas las imágenes resultado del proceso de regeneración por todos los modelos entrenados. Todas las imágenes que aparecen en este apartado son a color en tres canales. Las mismas pertenecen a entornos submarinos, buscando mejorarlas en cuanto a calidad, color, nitidez.

B.1. Resultados visuales imagen 1

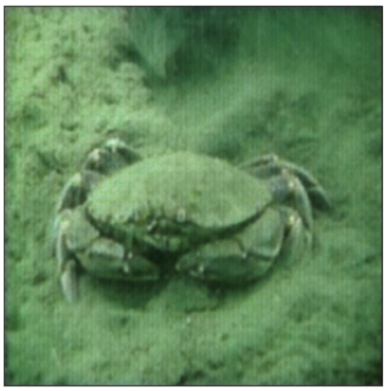


(a) Imagen original **(b)** Modelo 1 **(c)** Modelo 2

(d) Modelo 3 **(e)** Modelo 4 **(f)** Modelo 5

Figura B.1: Resultados visuales de la imagen 1

(g) Modelo 6



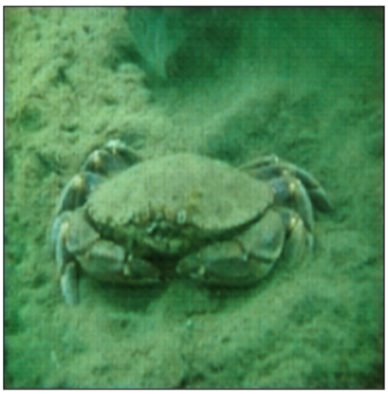
(h) Modelo 7

(i) Modelo 8

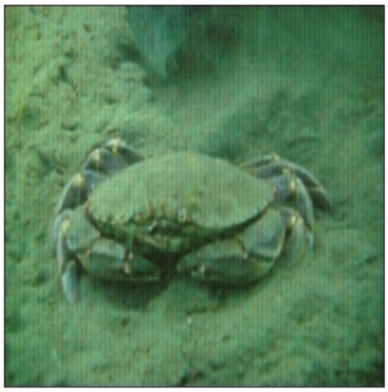
(j) Modelo 9

(k) Modelo 10

(l) Modelo 11



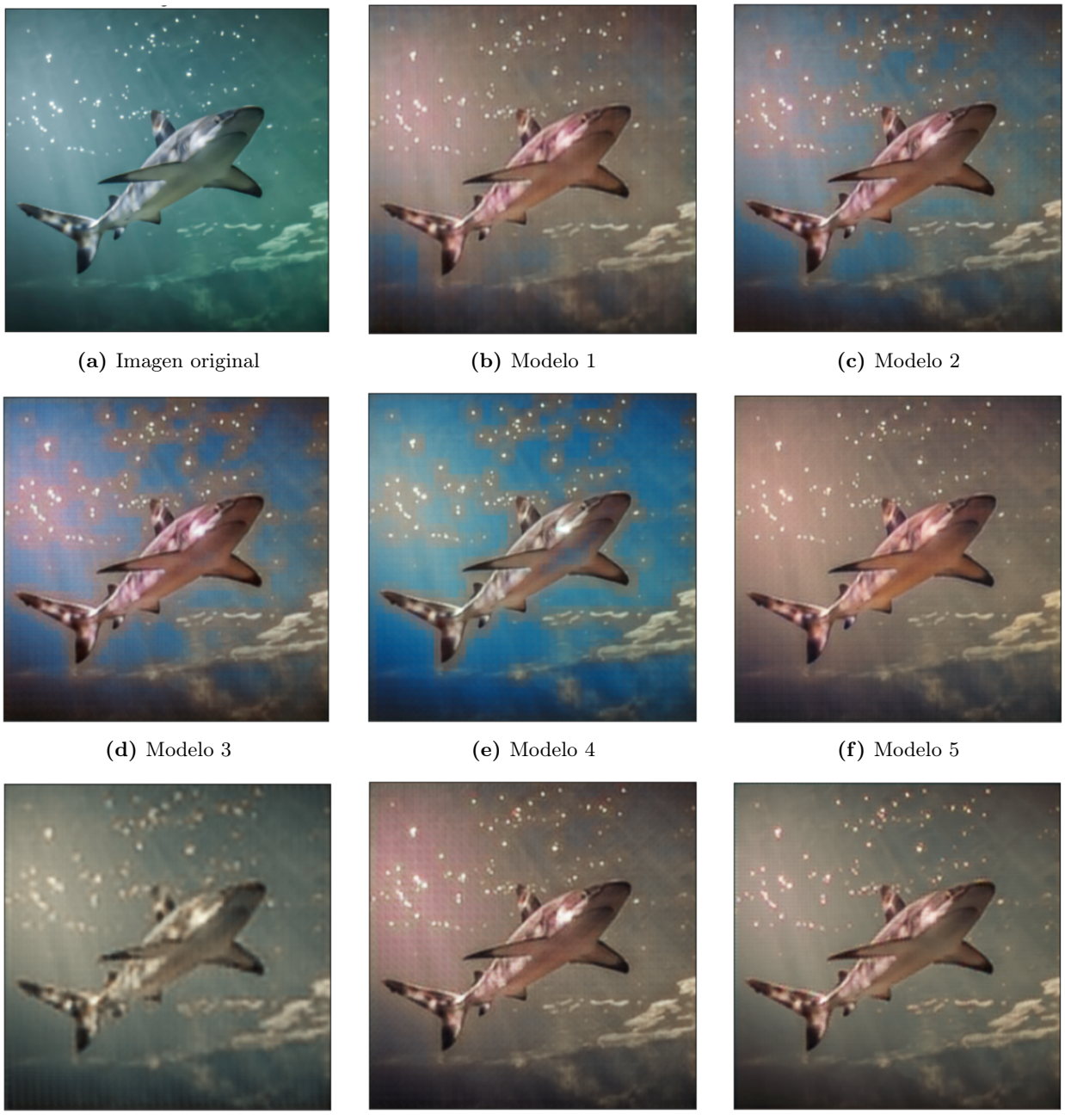
(m) Modelo 12



(n) Modelo 13

Figura B.1: Resultados visuales de la imagen 1 (continuación de la página 79)

B.2. Resultados visuales imagen 2



(a) Imagen original **(b)** Modelo 1 **(c)** Modelo 2

(d) Modelo 3 **(e)** Modelo 4 **(f)** Modelo 5

(g) Modelo 6 **(h)** Modelo 7 **(i)** Modelo 8

Figura B.2: Resultados visuales de la imagen 2

(j) Modelo 9

(k) Modelo 10

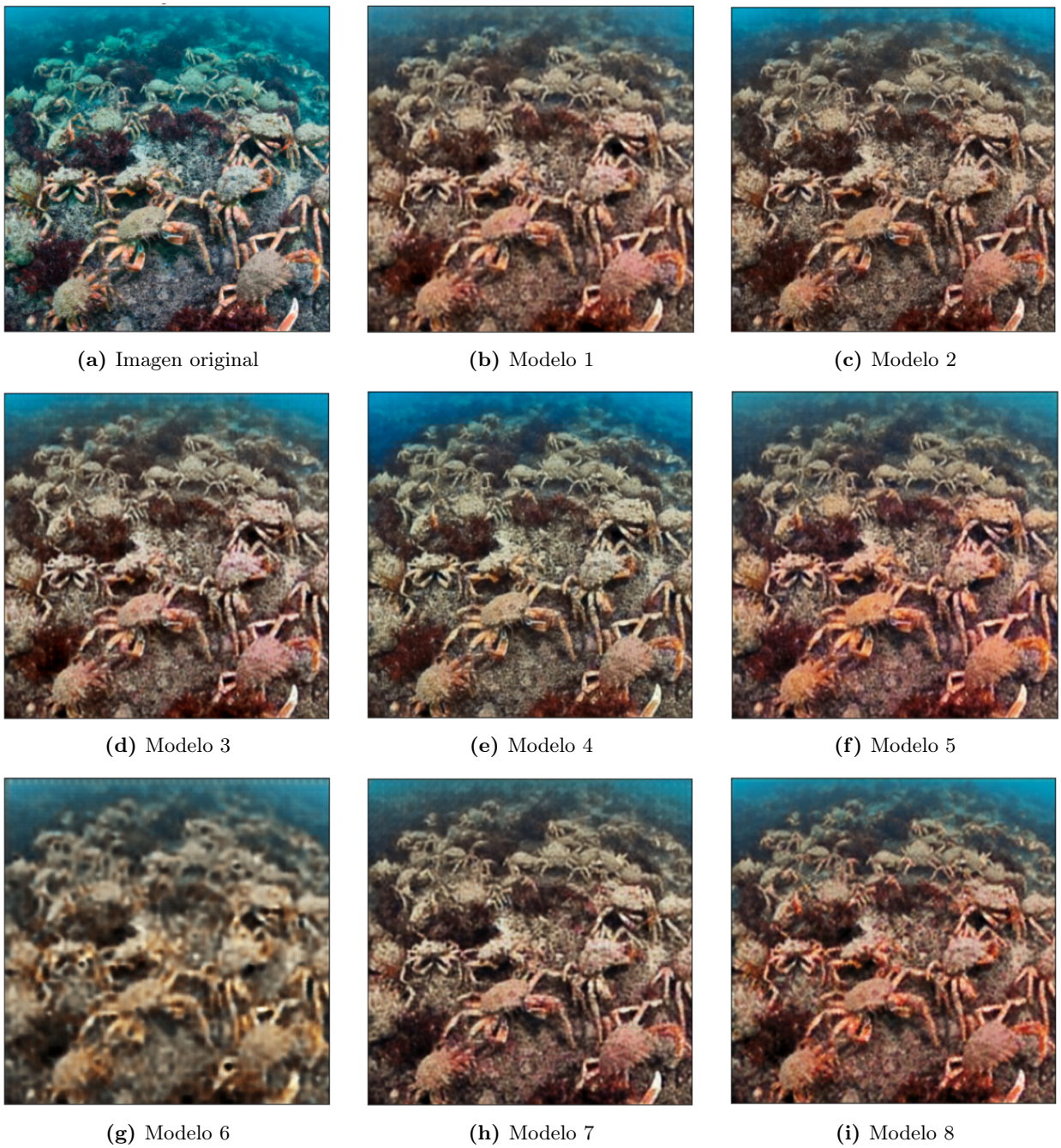
(l) Modelo 11

(m) Modelo 12

(n) Modelo 13

Figura B.2: Resultados visuales de la imagen 2 (continuación de la página 81)

B.3. Resultados visuales imagen 3



(a) Imagen original **(b)** Modelo 1 **(c)** Modelo 2

(d) Modelo 3 **(e)** Modelo 4 **(f)** Modelo 5

(g) Modelo 6 **(h)** Modelo 7 **(i)** Modelo 8

Figura B.3: Resultados visuales de la imagen 3

(j) Modelo 9

(k) Modelo 10

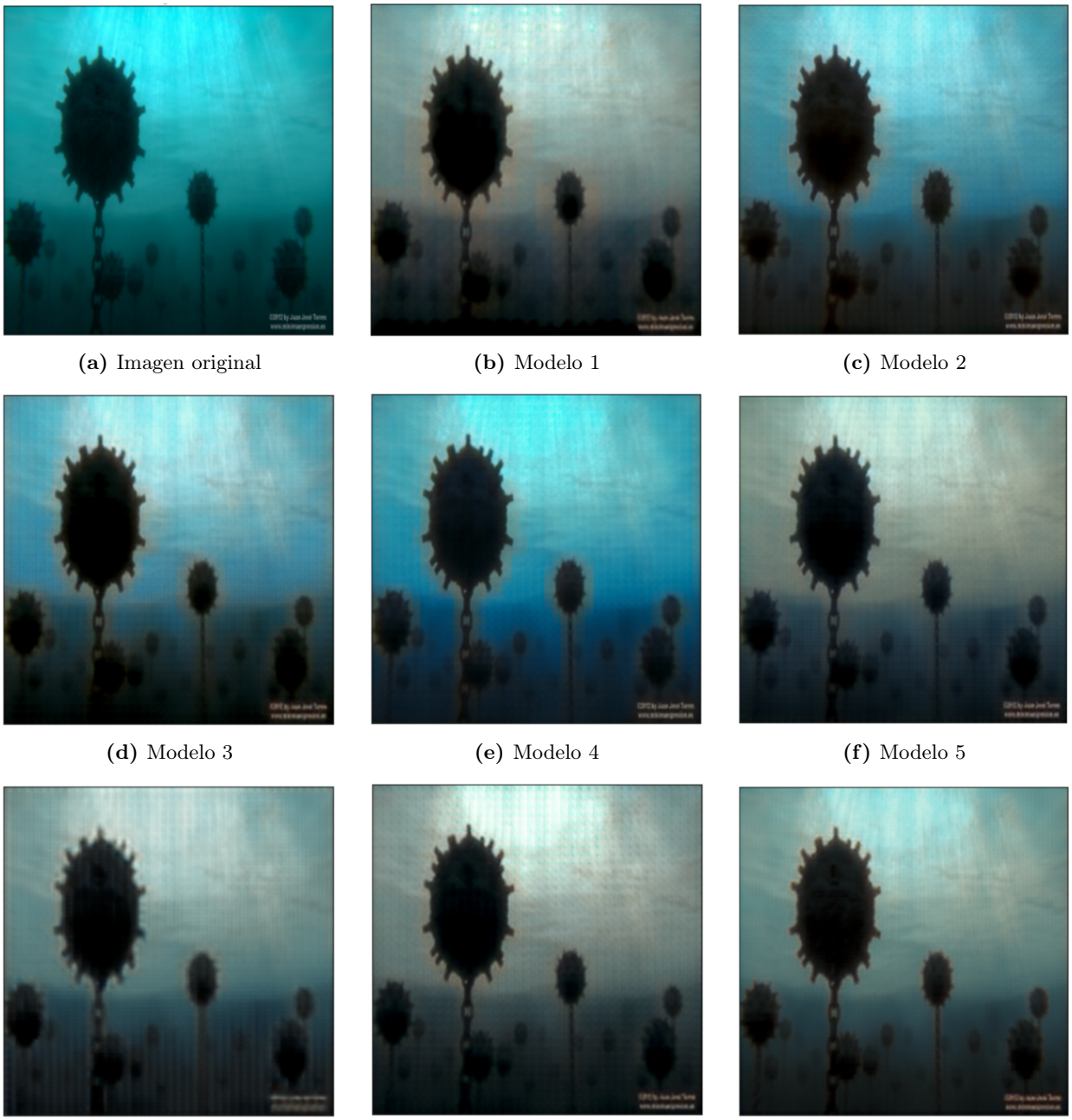
(l) Modelo 11

(m) Modelo 12

(n) Modelo 13

Figura B.3: Resultados visuales de la imagen 3 (continuación de la página 83)

B.4. Resultados visuales imagen 4



(a) Imagen original **(b)** Modelo 1 **(c)** Modelo 2

(d) Modelo 3 **(e)** Modelo 4 **(f)** Modelo 5

(g) Modelo 6 **(h)** Modelo 7 **(i)** Modelo 8

Figura B.4: Resultados visuales de la imagen 4

(j) Modelo 9

(k) Modelo 10

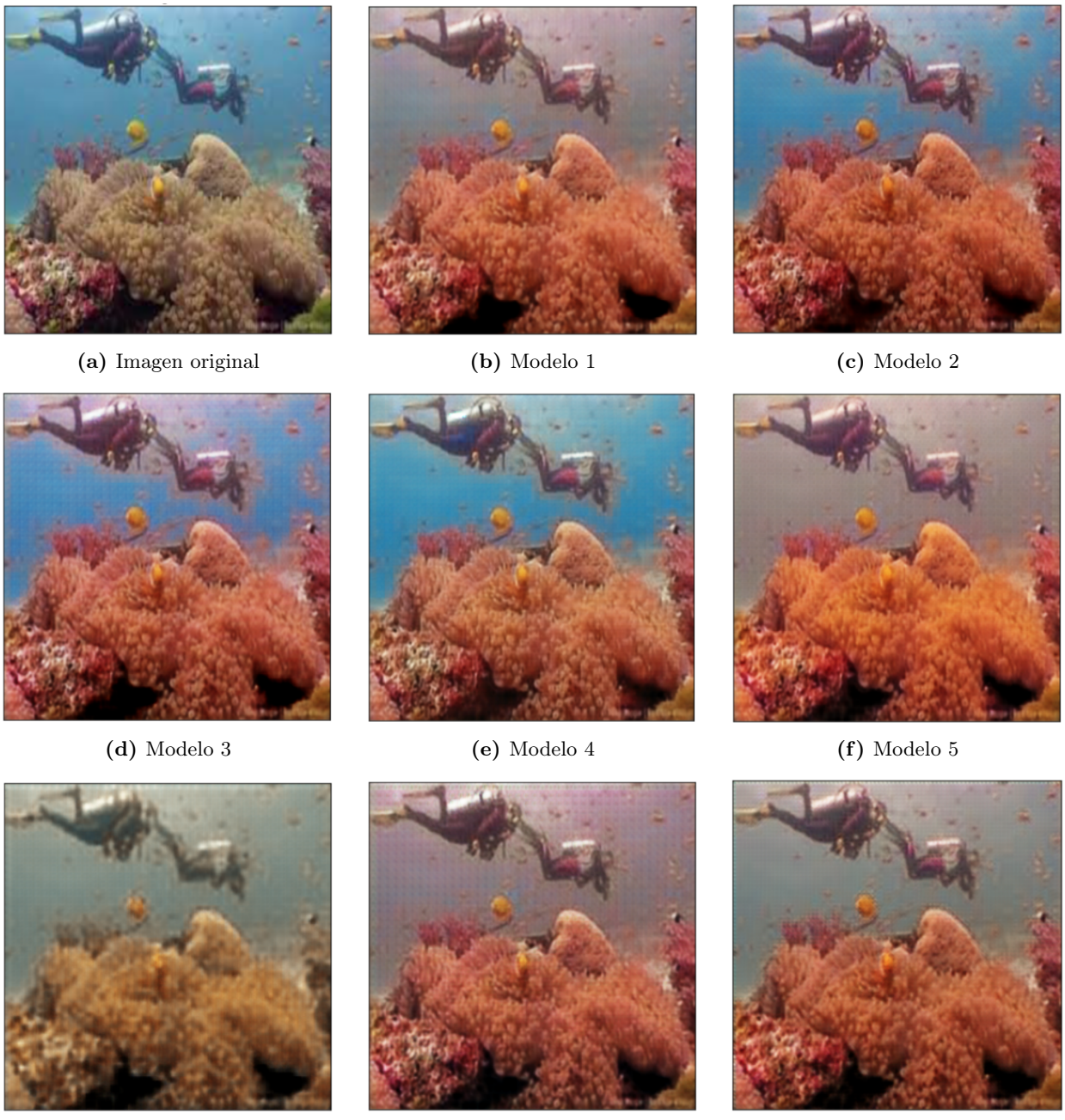
(l) Modelo 11

(m) Modelo 12

(n) Modelo 13

Figura B.4: Resultados visuales de la imagen 4 (continuación de la página 85)

B.5. Resultados visuales imagen 5

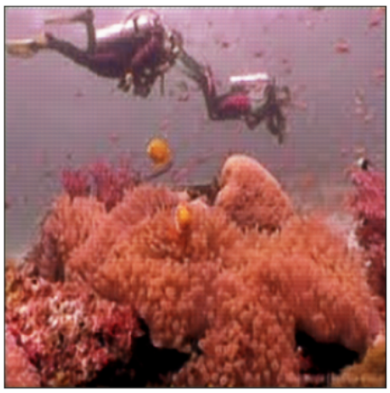


(a) Imagen original **(b)** Modelo 1 **(c)** Modelo 2

(d) Modelo 3 **(e)** Modelo 4 **(f)** Modelo 5

(g) Modelo 6 **(h)** Modelo 7 **(i)** Modelo 8

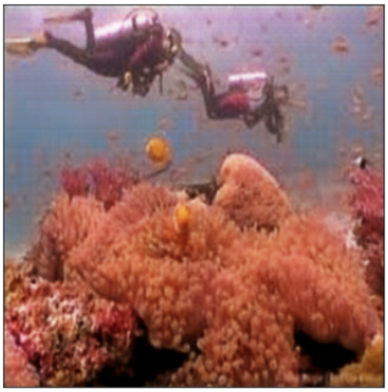
Figura B.5: Resultados visuales de la imagen 5



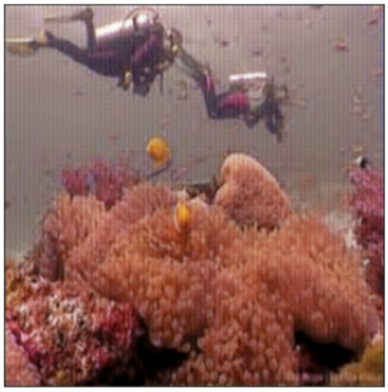
(j) Modelo 9

(k) Modelo 10

(l) Modelo 11



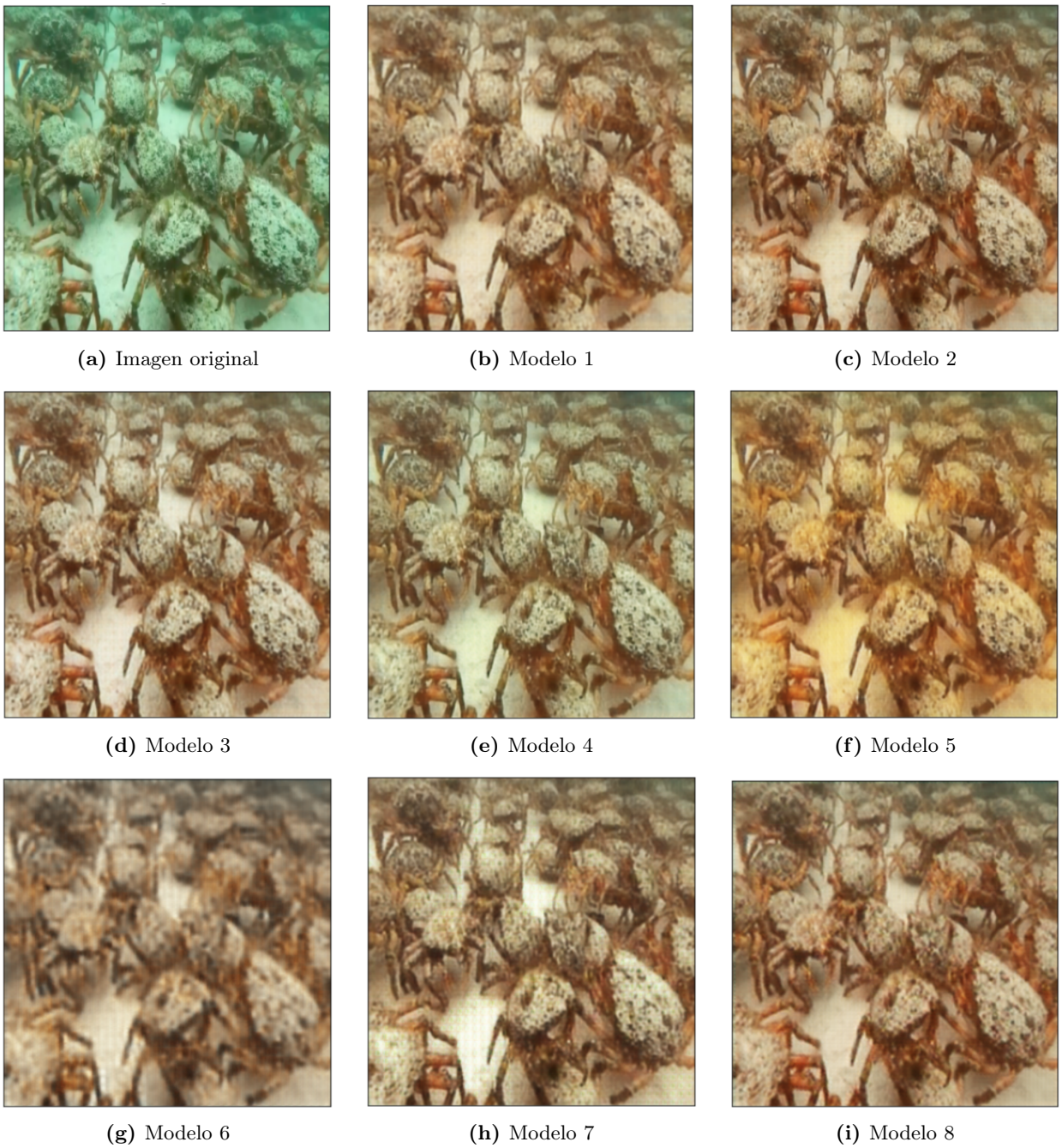
(m) Modelo 12



(n) Modelo 13

Figura B.5: Resultados visuales de la imagen 5 (continuación de la página 87)

B.6. Resultados visuales imagen 6



(a) Imagen original **(b)** Modelo 1 **(c)** Modelo 2

(d) Modelo 3 **(e)** Modelo 4 **(f)** Modelo 5

(g) Modelo 6 **(h)** Modelo 7 **(i)** Modelo 8

Figura B.6: Resultados visuales de la imagen 6

(j) Modelo 9

(k) Modelo 10

(l) Modelo 11

(m) Modelo 12

(n) Modelo 13

Figura B.6: Resultados visuales de la imagen 6 (continuación de la página 89)

Anexo C

Códigos del encoder-decoder

Los códigos que aquí se presentan, son una adaptación del *Autoencoder* [38] que han sido la base de la arquitectura *Encoder-Decoder* desarrollada.

C.1. Código de utilidades

Este código define una función que calcula la pérdida de contraste (contrastive loss) utilizando las etiquetas verdaderas y las predicciones, con una margen opcional, y devuelve el valor de pérdida resultante.

```
def contrastive_loss(y, preds, margin=1):
    # Explicitly cast the true class label data type to the predicted one.
    y = tf.cast(y, preds.dtype)
    # Calculate both losses (for loops and non loops)
    squaredPreds = K.square(preds)
    squaredMargin = K.square(K.maximum(margin - preds, 0))
    # Compute the contrastive loss
    loss = K.mean((1-y) * squaredPreds + y * squaredMargin)
    return loss
```

Programa C.1: Cálculo de contrastative-loss

Crea una imagen que contiene todas las imágenes de la entrada.

```
def montage(theBatch,doPlot=True,savePath=None):
    # Determine the input dimensions
    if len(theBatch.shape)==4:
        (nItems,nRowsImage,nColsImage,nChan)=theBatch.shape
    elif len(theBatch.shape)==3:
        (nItems,nRowsImage,nColsImage)=theBatch.shape
        nChan=1
    else:
        sys.exit('Input shape must be (n,r,c,ch) or (n,r,c) where n is the number of images, r and c are the
    ↪ number of rows and columns and ch is the number of channels. This last dimension can be ommited for
    ↪ grayscale images.')

    # Determine the output dimensions
    nCols=int(np.ceil(np.sqrt(nItems)))
    nRows=int(np.ceil(nItems/nCols))

    # Create the output matrix
    outImage=np.zeros((nRows*nRowsImage,nCols*nColsImage,nChan)).astype('float')

    # Convert the batch to float images
    theBatch=img_as_float(theBatch)

    # Build the output matrix
    batchIndex=0
    for r in range(nRows):
        if batchIndex>=nItems:
            break
        for c in range(nCols):
            if batchIndex>=nItems:
                break
            outImage[r*nRowsImage:(r+1)*nRowsImage,c*nColsImage:(c+1)*nColsImage]=theBatch[batchIndex]
            batchIndex=batchIndex+1

    # If plot requested, do it
    if doPlot:
        pyplot.figure(figsize=(20,20))
        pyplot.imshow(outImage)
        pyplot.axis('off')
        pyplot.show()

    # If save requested, do it
    if not (savePath is None):
        imsave(savePath,outImage)

    return outImage
```

Programa C.2: Creación de un buffer de imágenes

Este código define una función que muestra una barra de progreso con el porcentaje de completitud basado en un valor actual y un valor máximo.

```
def progress_bar(curValue,maxValue):
    thePercentage=curValue/maxValue
    curSize=int(50*thePercentage)
    sys.stdout.write('\r')
    sys.stdout.write(f"[{'='*curSize:<50}] {int(thePercentage*100)}%")
    sys.stdout.flush()
```

Programa C.3: Definición de barra de progreso

Este código define una función que obtiene los nombres de archivo de una ruta específica con una extensión determinada, los mezcla de forma aleatoria y los divide en dos conjuntos según una proporción dada. Si no se solicita una división, la función devuelve la lista completa de nombres de archivo.

```
def get_filenames_original(thePath,theExtension,smallSplitRatio=None):
    # Get all the filenames and shuffle them
    fileNames=[os.path.join(thePath,f) for f in os.listdir(thePath) if os.path.isfile(os.path.join(thePath,f))
    ↪  and f.endswith(theExtension)]
    np.random.shuffle(fileNames)

    # If no split requested, return the list
    if smallSplitRatio is None:
        return fileNames

    # Divide the file list in two sets
    numFiles=len(fileNames)
    cutIndex=int(smallSplitRatio*numFiles)

    # Return the two sets
    return fileNames[cutIndex:],fileNames[:cutIndex]
```

Programa C.4: División del conjunto de imágenes

Este código define una función que obtiene los nombres de archivo de las rutas originales y mejoradas, con una extensión específica, los mezcla de forma aleatoria y los divide en dos conjuntos según una proporción dada. Los nombres de archivo se almacenan en pares, representando la ruta del archivo original y la ruta del archivo mejorado. Si no se solicita una división, la función devuelve la lista completa de pares de nombres de archivo.

```
def get_filenames(originalPath, enhancedPath, theExtension, smallSplitRatio=None):
    # Get all the filenames from original an enhanced paths and shuffle them
    fileNames = []
    for f in os.listdir(originalPath):
        if os.path.isfile(os.path.join(originalPath,f)) and f.endswith(theExtension) and \
           os.path.isfile(os.path.join(enhancedPath,f)) and f.endswith(theExtension):
           fileNames.append( (os.path.join(originalPath,f), os.path.join(enhancedPath,f)) )
    np.random.shuffle(fileNames)

    # If no split requested, return the list
    if smallSplitRatio is None:
        return fileNames

    # Divide the file list in two sets
    numFiles=len(fileNames)
    cutIndex=int(smallSplitRatio*numFiles)

    # Return the two sets
    return fileNames[cutIndex:],fileNames[:cutIndex]
```

Programa C.5: Obtención de los ficheros por dataset

Esta función crea un nombre a cada modelo según sus características.

```
def build_model_basename(thePath,thePrefix,theList,theEpochs):
    # Build the model file name base
    baseName=os.path.join(thePath,thePrefix)
    for curItem in theList:
        baseName += f"_{curItem}"
    baseName+=f"_EPOCHS{theEpochs}"
    return baseName
```

Programa C.6: Definición del nombre de modelo

Clase AutoGenerator: Generador de datos simple (Sequence) para alimentar un Encoder-Decoder Convolucional.

```
class AutoGenerator(Sequence):
################################################################################
# CONSTRUCTOR
# Input: fileNames   - List of the names of the image files in the dataset.
#        imgSize     - Desired image size
#        batchSize   - The size of the batches.
#        doRandomize - Randomize file order at the end of each epoch and
#                      apply some basic data augmentation (True/False)
################################################################################
    def __init__(self,fileNames,imgSize=(64,64),batchSize=10,doRandomize=True):
        # Store input parameters
        self.fileNames=fileNames
        self.imgSize=imgSize
        self.batchSize=batchSize
        self.doRandomize=doRandomize

        # Get the number of images
        self.numImages=len(fileNames)

        # Additional initializations
        self.on_epoch_end()

    def on_epoch_end(self):
        """ CALLED AFTER EACH TRAINING EPOCH """
        if self.doRandomize:
            np.random.shuffle(self.fileNames)

    def __len__(self):
        """GET THE NUMBER OF BATCHES"""
        return int(np.ceil(self.numImages/float(self.batchSize)))

    def random_flip(self, curImage, curRandom):
        """ OUTPUT ONE DATA-LABEL PAIR. SINCE THIS IS AIMED AT AN AUTOENCODER,
            DATA AND LABEL ARE THE SAME. """
        if curRandom<.25:
            curImage=curImage[::-1,:]
        elif curRandom<.50:
            curImage=curImage[:,::-1]
        elif curRandom<.75:
            curImage=curImage[::-1,::-1]
        return curImage

    def __getitem__(self,theIndex):
        X_original, X_enhanced = [], []

        # Compute start and end image indexes in the requested batch
        bStart=max(theIndex*self.batchSize,0)
        bEnd=min((theIndex+1)*self.batchSize,self.numImages)

        # If randomization is requested, simple data augmentation will also be performed.
        if self.doRandomize:
            theRandoms=np.random.random(bEnd-bStart)
```

```
53
54          # For each image in the batch
55          for i in range(bStart,bEnd):
56              # Read the image or images
57              if type(self.fileNames[i]) == str:
58                  originalImage=img_as_float(imread(self.fileNames[i]))
59                  enhancedImage = None
60              else:
61                  originalImage=img_as_float(imread(self.fileNames[i][0])) #[0,1]
62                  enhancedImage=img_as_float(imread(self.fileNames[i][1]))  #[0,1]
63
64              # Preprocess images
65              if originalImage.shape[:2]!=self.imgSize: # Resize only if necessary
66                  originalImage=resize(originalImage,self.imgSize)
67
68              if (enhancedImage is not None) and (enhancedImage.shape[:2]!=self.imgSize):
69                  enhancedImage=resize(enhancedImage,self.imgSize)
70
71              # Apply random flip if randomization requested
72              if self.doRandomize:
73                  curRandom=theRandoms[i-bStart]
74                  originalImage = self.random_flip(originalImage, curRandom)
75                  if enhancedImage is not None:
76                      enhancedImage = self.random_flip(enhancedImage, curRandom)
77
78              X_original.append(originalImage) # Append the resulting image(s)
79              if enhancedImage is not None:
80                  X_enhanced.append(enhancedImage)
81
82          if enhancedImage is not None:
83              return np.array(X_original), np.array(X_enhanced) # Return the original an enhanced images
84          else:
85              # Return the same data as X and y
86              return np.array(X_original), np.array(X_original)
87
88      def plot_batch(self,batchNum): # PLOT THE SPECIFIED BATCH
89          montage(self.__getitem__(batchNum)[0])
```

Programa C.7: Código de AutoGenerator

Clase ModelWrapper: utilizada para facilitar el acceso a los modelos, definida como clase base para cada uno.

```
class ModelWrapper:

    def __init__(self):
        """ CONSTRUCTOR """
        self.theModel=None
        self.trainHistory=None
        self.trainTime=None
        self.evaluationResults=None
        self.metricsNames=None

    def create(self):
        """ CREATES AND COMPILES THE MODEL """
        pass

    def fit(self,*args,**kwargs):
        tStart=time.time()
        self.trainHistory=self.theModel.fit(*args,**kwargs).history
        self.trainTime=time.time()-tStart
        self.metricsNames=self.theModel.metrics_names

    def evaluate(self,*args,**kwargs):
        self.evaluationResults=self.theModel.evaluate(*args,**kwargs)
        return self.evaluationResults

    def predict(self,*args,**kwargs):
        return self.theModel.predict(*args,**kwargs)

    def summary(self):
        self.theModel.summary()

    def is_saved(self,baseName):
        return os.path.exists(baseName+'.h5')

    def print_evaluation(self):
        """ PRINTS THE MODEL EVALUATION RESULTS """
        if self.evaluationResults is None:
            text.red('[ ERROR ] Cannot print evaluation since the model has not been evaluated.')
        else:
            for i in range(len(self.evaluationResults)):
                print(f"{self.metricsNames[i]} : {self.evaluationResults[i]:.3f}")

    def save(self,baseName,forceOverwrite=False):
        """ SAVES MODEL AND HISTORY TO DISK IF POSSIBLE """
        if forceOverwrite or (not self.is_saved(baseName)):
            self.theModel.save(baseName+'.h5')
            with open(baseName+'_HISTORY.pkl','wb') as outFile:
                dump([self.trainHistory,self.trainTime,self.evaluationResults,self.metricsNames],outFile)
            return True
        else:
            text.red('[ SAVING ABORTED ] Model file already exists. Use forceOverwrite=True to overwrite it.')
            return False

```

```
53      def load(self,baseName):
54          """ LOADS MODEL AND HISTORY FROM DISK IF POSSIBLE """
55          if self.is_saved(baseName):
56              self.theModel=load_model(baseName+'.h5')
57              with open(baseName+'_HISTORY.pkl','rb') as inFile:
58                  [self.trainHistory,self.trainTime,self.evaluationResults,self.metricsNames]=load(inFile)
59          else:
60              text.red('[ LOADING ABORTED ] Model file not found.')
61
62      def plot_training_history(self,plotTitle='TRAINING EVOLUTION'):
63          """ PLOTS THE TRAINING HISTORY """
64          # Get the keys in trainHistory to be plotted together
65          nonVal=[theKey for theKey in self.trainHistory if not theKey.startswith('val_')]
66          theVal=[theKey[4:] for theKey in self.trainHistory if theKey.startswith('val_')]
67          pairsList, labelsList = [], []
68          for curKey in nonVal:
69              if curKey in theVal:
70                  theVal.remove(curKey)
71                  pairsList.append([curKey,'val_'+curKey])
72              else:
73                  pairsList.append([curKey])
74              labelsList.append(curKey)
75
76          for curKey in theVal:
77              pairsList.append(['val_'+curKey])
78              labelsList.append(curKey)
79
80          for i in range(len(pairsList)): # Plot everything
81              pyplot.figure()
82              for curItem in pairsList[i]:
83                  pyplot.plot(self.trainHistory[curItem])
84
85              pyplot.title(plotTitle)
86              pyplot.xlabel('epoch')
87              pyplot.ylabel(labelsList[i])
88              pyplot.legend(pairsList[i],loc='best')
89              pyplot.show()
```

Programa C.8: Definición de clase ModelWrapper

Wrapper simple para facilitar el acceso a un modelo de Keras

```
class AutoModel(ModelWrapper):

    def __init__(self):
        self.encoderModel=None
        self.decoderModel=None
        super().__init__()

    def create(self,inputShape=(64,64,3),theFilters=[16,16,32]):
        """ CREATES THE MODEL """
        encoderInput=Input(shape=inputShape,name='encoder_input') # Create the encoder layers
        x=encoderInput

        for curFilter in theFilters:
            x=Conv2D(curFilter,kernel_size=(3,3),strides=2,padding='same')(x)
            x=LeakyReLU(alpha=0.2)(x)
            x=BatchNormalization()(x)
        encoderOutput=x

        # Create the decoder layers, learning rate grande 0.001
        decoderInput=Input(shape=tuple(encoderOutput.get_shape()[1:]), name='decoder_input')
        x=decoderInput
        for curFilter in theFilters[::-1]:
            x=Conv2DTranspose(curFilter,kernel_size=(3,3),strides=2,padding='same')(x)
            x=LeakyReLU(alpha=0.2)(x)
            x=BatchNormalization()(x)

        # Changing output size into channels
        decoderOutput=Conv2DTranspose(inputShape[2],kernel_size=(3,3),padding='same', activation='sigmoid')(x)
    ↪  #[-1,1]->tanh, #[-3,3] -> sin activation

        # Create the models
        self.encoderModel=Model(encoderInput,encoderOutput,name='encoder_model')
        self.decoderModel=Model(decoderInput,decoderOutput,name='decoder_model')
        self.theModel=Model(encoderInput,self.decoderModel(self.encoderModel(encoderInput)),name='
    ↪ autoencoder_model')

        # Compile them
        self.theModel.compile(optimizer='adam', loss='mse', metrics=['accuracy','mse','mae'])
        self.encoderModel.compile(optimizer='adam', loss='mse', metrics=['accuracy','mse','mae'])
        self.decoderModel.compile(optimizer='adam', loss='mse', metrics=['accuracy','mse','mae'])

    def summary(self):
        super().summary()
        self.encoderModel.summary()
        self.decoderModel.summary()

    def count_params(self):
        return sum((self.encoderModel.count_params(), self.decoderModel.count_params()))

    def save(self,baseName,forceOverwrite=False):
        if super().save(baseName,forceOverwrite):
            self.encoderModel.save(baseName+'_ENC.h5')
            self.decoderModel.save(baseName+'_DEC.h5')
```

```
52
53      def load(self,baseName):
54          self.encoderModel=load_model(baseName+'_ENC.h5')
55          self.decoderModel=load_model(baseName+'_DEC.h5')
56          super().load(baseName)
57
58      def encode(self,theImages):
59          return self.encoderModel.predict(theImages)
60
61      def decode(self,theFeatures):
62          return self.decoderModel.predict(theFeatures)
```

Programa C.9: Definición de clase AutoModel

Clase para construcción, entrenamiento, evaluación y almacenamiento de un modelo.

```
def autobuild(pathTrain, pathTest, pathModels, extImage='png',
              inputShape=(64,64,3), theFilters=[128,128,64],
              numEpochs=100,splitVal=0.2):

    # Create the autoencoder object
    theModel=AutoModel()

    # Build the file name corresponding to that model
    baseName=build_model_basename(pathModels, 'AUTOENCODER', theFilters, numEpochs)
    print(f'PROCESSING MODEL {baseName}')

    # If the model is already saved, do nothing
    if theModel.is_saved(baseName):
        text.red('[ ABORTING ] The model seems to be already trained and saved.')
        return baseName

    # Get the training and validation file names
    trainFileNames,valFileNames=get_filenames(pathTrain+"original",pathTrain+"enhanced",extImage,splitVal)

    # Create the training and validation data generators
    trainGenerator=AutoGenerator(trainFileNames,imgSize=inputShape[:2],doRandomize=True)
    valGenerator=AutoGenerator(valFileNames,imgSize=inputShape[:2],doRandomize=False)

    theModel.create(inputShape=inputShape,theFilters=theFilters) # Create the model and compile it
    text.green('TRAINING THE MODEL') # Train the model

    timer_callback = TimerCallback()
    plotting_callback = PlotLearning()
    early_stopping_callback = EarlyStopping(monitor='val_loss', patience=3, mode='min')

    log_dir = "./logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

    theModel.fit(x=trainGenerator,validation_data=valGenerator,epochs=numEpochs, callbacks=[plotting_callback,
    ↪  timer_callback, early_stopping_callback, tensorboard_callback])

    # Prepare test data
    testFileNames=get_filenames(pathTest+"original", pathTest+"enhanced",extImage) # Get the test file names
    testGenerator=AutoGenerator(testFileNames,imgSize=inputShape[:2],doRandomize=False) # Build the test data
    ↪ generator
    text.green('EVALUATING THE MODEL')
    theModel.evaluate(testGenerator) # Evaluating
    theModel.save(baseName) # Saving
    return baseName
```

Programa C.10: Construcción, entrenamiento, evaluación y almacenamiento

Imprime y plotea la arquitectura, información de entrenamiento y evaluación sobre un modelo almacenado.

```
def autoshow(i, baseName, testBatch=None):
    theModel=AutoModel() # Create the autoencoder object

    if not (theModel.is_saved(baseName)): # If the model is already saved, do nothing
        text.red(f"[ ABORTING ] Model #{i} file {baseName} not found. Please execute autobuild before.")
        return

    text.green(f'SHOWING MODEL #{i}  >>  {baseName.replace("./models_saved/","")}')
    theModel.load(baseName) # Load the model

    # Show modelwrapper built-in stats
    theModel.summary()                          # Print the summary
    theModel.plot_training_history(baseName)    # Training stats
    theModel.print_evaluation()                 # Evaluation stats

    # Process one batch of test images and plot the results if requested
    if not (testBatch is None):
        theFeatures=theModel.encode(testBatch) # Encode the batch
        thePredictions=theModel.decode(theFeatures) # Decode the batch
        jointData=np.vstack((testBatch,thePredictions)) # Plot both data and predictions together
        montage(jointData)
```

Programa C.11: Impresión de métricas y resultados

Genera las predicciones en base a un modelo almacenado.

```
def predictBatch(i, baseName,testBatch=None):
    theModel=AutoModel() # Create the autoencoder object

    if not (theModel.is_saved(baseName)): # If the model is already saved, do nothing
        text.red(f"[ ABORTING ] Model #{i} file {baseName} not found. Please execute autobuild before.")
        return

    text.green(f'SHOWING MODEL #{i}  >>  {baseName.replace("./models_saved/","")}')
    theModel.load(baseName) # Load the model

    # Process one batch of test images and return predictions
    if not (testBatch is None):
        theFeatures=theModel.encode(testBatch) # Encode the batch
        thePredictions=theModel.decode(theFeatures) # Decode the batch
        return thePredictions
```

Programa C.12: Generación de predicciones

Define los parámetros restantes y entrena todos los modelos si aún no lo están.

```
MODEL_INPUT_SHAPE=(320,320,3) # Model input shape.
MODEL_EPOCHS=30 # Number of epochs to train
EXT_IMAGE='jpg' # Image file extension
SPLIT_VAL=0.2 # Ratio of train images to use as validat.
MODEL_FILTERS = [
                    [32, 64, 128, 256],
                    [32, 64, 128],
                    [256, 128, 32],
                    [256, 128, 256],
                    [256, 256, 64],
                    [128, 8, 4],
                    [128,128,16],
                    [128,4],
                    [128,8],
                    [128,128,32],
                    [128,16],
                    [128,128,64],
                    [128,32]
                ]
# Paths
PATH_TRAIN='./dataset/train/'     # Train images path
PATH_TEST='./dataset/test/'       # Test images path
PATH_MODELS='./models_saved/'     # Model storage path

# Train, evaluate and save autoencoders using all the filters in MODEL_FILTERS
allNames=[]
for curFilters in MODEL_FILTERS:
    baseName=autobuild(PATH_TRAIN,PATH_TEST,PATH_MODELS,EXT_IMAGE,MODEL_INPUT_SHAPE,curFilters,MODEL_EPOCHS,
    ↪ SPLIT_VAL)
    allNames.append(baseName)
```

Programa C.13: Función principal

Leroy Deniz

Mejora de imágenes submarinas mediante deep learning

Trabajo Final de Máster
Máster Universitario en Ciencia de Datos
Universitat Oberta de Catalunya
Julio 2023