ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Máster

REDES GENERATIVAS PARA LA RECONSTRUCCION DE IMÁGENES SUBMARINAS

(GENERATIVE NETWORKS FOR UNDERWATER IMAGES RECONSTRUCTION)

Para acceder al Título de

Máster Universitario en

Ingeniería de Telecomunicación

Autor: Luis Urresti de las Alas - Pumariño

Septiembre - 2020

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

MASTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE MASTER

Realizado por: Luis Urresti de las Alas – Pumariño

Director del TFM: Adolfo Cobo

Título: "Redes generativas para la reconstrucción de imágenes submarinas"

Title: "Generative networks for underwater images reconstruction**"**

Presentado a examen el día:

para acceder al Título de

MASTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Composición del Tribunal:

Presidente (Apellidos, Nombre): Tazón Puente, Antonio

Secretario (Apellidos, Nombre): Lomer Barboza, Mauro

Vocal (Apellidos, Nombre): Suárez Rodríguez, Almudena

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente	Fdo.: El Secretario
Fdo.: El Vocal	Fdo.: El Director del TFM (sólo si es distinto del secretario)
Vº Bº del Subdirector	Trabajo Fin de Máster Nº (a asignar por Secretaría)

Trabajo realizado en parte con los medios aportados por los proyectos TEC2016-76021-C2-2-R, PID2019-107270RB-C21, DeepRamp (Pleamar 2019, FB) y 2470S (PPNN 2017)

Agradecimientos

A mí familia, por darme la oportunidad de estudiar y creer en mi en todo momento.
A mis amigos, por estar ahí en las buenas y en las malas, siempre animándome.
A Marieta, por no dejar que me rinda. Por comprenderme en todo momento.
A Adolfo, por proporciónname un proyecto que me ha llegado a apasionar.

Resumen

En los últimos años, el *deep learning* se ha convertido en una de las ramas más destacadas del aprendizaje automático. Las técnicas de aprendizaje profundo han revolucionado campos como la medicina o la visión artificial, llegando a conseguir resultados muy superiores a los obtenidos con las técnicas tradicionales.
Es por este motivo que a lo largo del presente documento se busca aplicar técnicas del aprendizaje profundo, concretamente las redes generativas adversarias, para tratar de reconstruir las imágenes submarinas que recibe el piloto de un dron submarino durante su pilotaje. Y, de esta forma, a través de las técnicas de aprendizaje profundo, facilitar al piloto su labor.

Palabras clave: Aprendizaje profundo, redes neuronales artificiales, redes convolucionales, redes generativas adversarias, visión artificial.

Abstract

Deep learning has recently become one of the most prominent branches of machine learning. Learning techniques have revolutionized fields such as medicine or artificial vision, achieving results far superior to those obtained with traditional techniques.
This is the reason why the current document seeks to apply deep learning techniques, specifically adversary generative networks, to try to reconstruct the underwater images received by the pilot of an underwater drone during their piloting. And, thus, facilitate the pilot's work using deep learning techniques.

Keywords: Deep learning, artificial neural network, convolutional network, generative adversarial network, artificial vision.

Tabla de Contenido

Tabla de Figuras

Tabla de ecuaciones

Tabla de tablas

1. Introducción

En el año 2014, un estudiante de doctorado de la Universidad de Montreal discutía en un bar con sus compañeros la solución de un problema para el que no lograban obtener un resultado fiable. Estaban utilizando un generador para aumentar el número de datos, pero los producidos artificialmente mediante técnicas de *machine learning* eran poco realistas o tenían errores.

El estudiante, llamado Ian Goodfellow, defendía la idea, que sus compañeros tachaban de poco viable, de poner a competir a dos redes neuronales haciendo que una se encargase de la generación de los datos mientras la otra se ocupaba de decidir si los datos eran "reales" o no.

Ian Goodfellow es considerado el padre de las redes generativas adversarias (GANs) ya que, tras esa discusión en el bar de Montreal, demostró su idea y escribió el *paper Generative Adversarial Nets* [1] junto con sus compañeros de la Universidad de Montreal, siendo una de las mayores innovaciones en los últimos 15 años en el campo de *machine learning*.

Desde que en 2014 Ian Goodfellow y sus compañeros presentaran al mundo las redes generativas adversarias, estas se han utilizado para múltiples aplicaciones. Desde redes que hacen uso de la transferencia de estilo para crear nuevas obras de pintores ya fallecidos, como puede ser el caso de *CycleGAN* [2], una red neuronal capaz de convertir imágenes fotográficas en nuevas obras de Monet o Van Gogh, hasta redes como *StackGAN* [3], que es capaz de generar imágenes a partir de una descripción textual de lo que se está viendo.

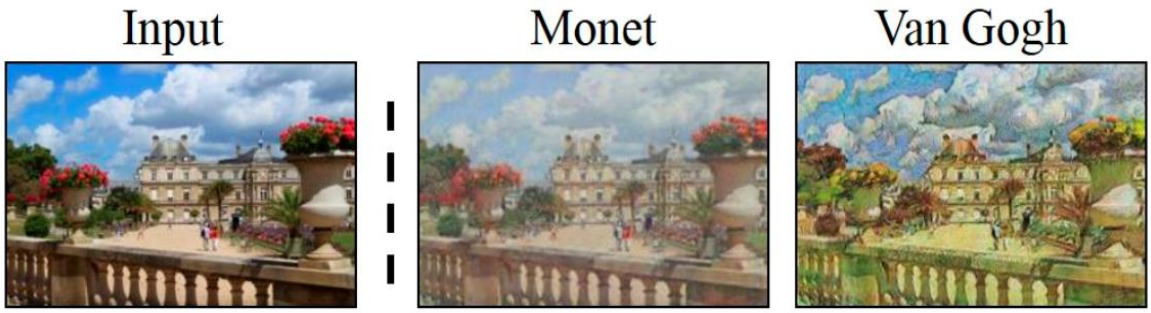

Figura 1 – Resultado de la CycleGAN [2]
imagen obtenida de https://junyanz.github.io/CycleGAN/

Las redes generativas no solo presentan un gran potencial generando o transformando imágenes, sino que también pueden ser aplicadas en la reconstrucción o regeneración de imágenes que previamente han sido dañadas. Es por este motivo por el que se eligió esta tecnología para para intentar solucionar el problema planteado, que no es otro que la reconstrucción de las imágenes captadas por un dron submarino, que presentan un escaso nivel de detalle y poca visibilidad, para, con esta reconstrucción, facilitar la labor del piloto en la conducción de dicho dron por el fondo marino.

2. Motivación y Estructura

Este trabajo, como se ha comentado en la introducción, se planteó con el objetivo de resolver un problema que surge en los drones submarinos al ser pilotados. Concretamente, el problema surge en las imágenes que recibe el piloto durante el manejo del dron; ya que estas imágenes cuentan con una resolución muy baja debido al poco ancho de banda con el que cuenta el cable umbilical que une al piloto con el dron. Además de este problema, las imágenes que recibe el piloto sufren el efecto *scattering* y la turbidez del agua, y debido a esto presentan un aspecto borroso y con poca claridad.

Para resolver el problema planteado se han utilizado las técnicas modernas del aprendizaje automático, como es la rama del aprendizaje profundo, y dentro de esta rama, como se partía de unas imágenes de entrada borrosas, con poco nivel de detalle y de muy baja resolución, se han utilizado redes generativas adversarias para reconstruir mediante un proceso de aprendizaje supervisado en el que la red tomará como referencia imágenes de alta calidad del fondo marino.

A lo largo de este documento se explicará el proceso que se ha seguido dividido en los siguientes capítulos:

- El tercer capítulo se utilizará como introducción para sentar las bases de los conceptos que se han aplicado para llevar a cabo la implementación realizada. Se comenzará explicando detalladamente en qué consiste una red neuronal artificial, centrándose, en primer lugar, en el funcionamiento de su unidad básica de procesamiento: la neurona. Posteriormente, nos centraremos en realizar un acercamiento al conocimiento jerarquizado: propiedad que otorga a las redes neuronales la capacidad de adquirir conocimientos complejos. Se tratarán los tres paradigmas del aprendizaje automático y las diferencias que se pueden encontrar entre cada uno de ellos. De esta forma, se habrán adquirido los conocimientos básicos para comprender el funcionamiento de las redes convolucionales, las cuales presentan una estructura necesaria de entender para comprender fácilmente el funcionamiento de las redes generativas adversarias, y el de la arquitectura U-NET, que es la que sigue el modelo *Pix2Pix*, el cual se ha utilizado para resolver el problema planteado.

- En el cuarto capítulo se explicarán las implementaciones realizadas, ya que existen dos formas de solventar el problema: mediante una única red o dividiendo el problema en dos redes. Se comenzará explicando la selección de los datos de entrenamiento y el proceso seguido para no condicionar el aprendizaje de la red; las diferentes redes que se han utilizado. Finalmente, se tratarán los resultados obtenidos por ambas implementaciones y se realizará una comparación entre ellas viendo los problemas encontrados.

- En el quinto capítulo se comentará una implementación adicional que se ha realizado. Esta consiste en la detección automática de la distancia entre dos puntos láser que emite el dron submarino. De esta forma se quiere facilitar aún más su labor al piloto

- Finalmente, en el sexto capítulo se recogerán las conclusiones del proyecto realizado y se marcarán algunas líneas futuras que se pueden seguir para continuarlo.

3. Estado del Arte

3.2. Redes Neuronales Artificiales

Las redes neuronales artificiales, ANN (Artificial Neural Networks), son sistemas informáticos que se han inspirado en el funcionamiento del sistema nervioso de los seres humanos.

Fueron creadas en la década de los 50, cuando Frank Rosenblatt aplicó la teoría de neuronas para diseñar el "Perceptrón", la que sería la primera red neuronal artificial [4]. Gracias a la gran mejora que ha habido en los últimos años tanto en las técnicas de diseño como de la tecnología, las redes neuronales han empezado a usarse en la gran mayoría de ámbitos; ya que son capaces de modelar comportamientos inteligentes.

Su estructura es compleja y avanzada, pero su funcionamiento se debe al trabajo conjunto de sistemas más simples. A cada uno de estos sistemas simples, en honor a las células cerebrales que imitan, se le denomina neurona.

La neurona es la unidad básica de procesamiento que se puede encontrar en el interior de una red neuronal; cada neurona cuenta con una serie de redes de entrada a través de las cuales reciben estímulos externos, los conocidos como "valores de entrada". Con estos valores, la neurona realizará un cálculo interno para generar un valor de salida que, a su vez, irá conectado a la entrada de una nueva neurona, formando de esta manera la red neuronal. Es decir, su función es exactamente igual a la de una función matemática que calcula un valor de salida en función de los datos de entrada.

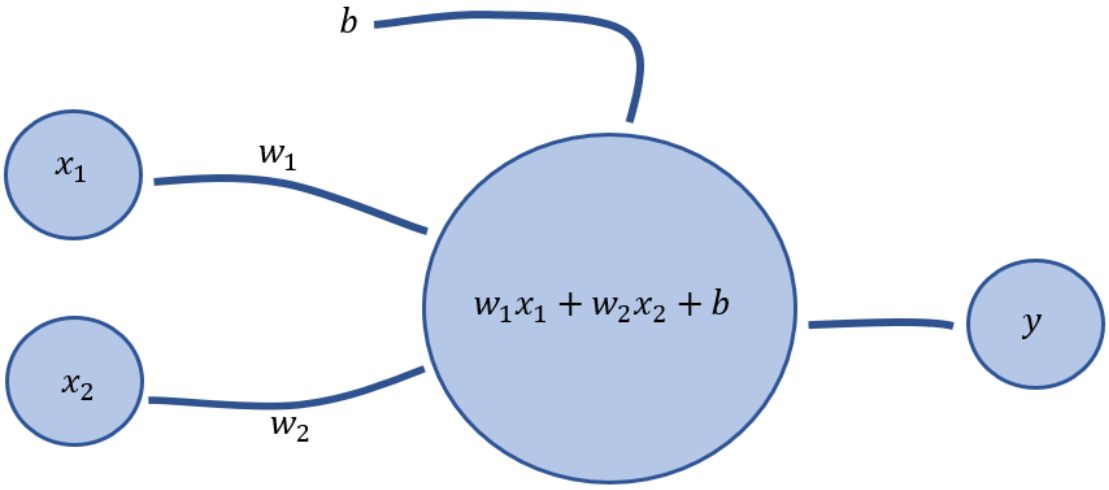

Figura 2 – Redes Neuronales Artificiales: Neurona

La operación que realiza la neurona en su interior consiste en obtener un único valor a través de la aplicación de una función de activación, γ. Esta función γ se activará en función de la suma ponderada de cada uno de los valores de entrada, con el peso que

tiene asociado la conexión que une el valor con la neurona más un sesgo b. Es decir, cada variable de entrada tiene asociado un valor que indica el nivel de intensidad con el que dicha variable afecta a la neurona; tal y como se puede observar en la figura que se encuentra sobre estas líneas (Figura 2). Son estos pesos los que actuarán como parámetros de nuestro modelo, que se van a ir modificando para mejorar el resultado obtenido. El valor de salida de la función de activación se puede calcular, por lo tanto, a través de la siguiente ecuación (Ecuación 1).

$$y = \gamma\left(\sum w_i x_i + b\right)$$

Ecuación 1– Función de activación [5]

Como se ha comentado, una ANN está compuesta por un conjunto de neuronas que actúan realizando pequeñas tareas más simples, para resolver, conjuntamente, un problema complejo. Dos neuronas se pueden colocar, una respecto a la otra, de dos formas distintas. La primera forma consiste en colocar una al lado de la otra, de forma que ambas neuronas se encuentren en la misma capa, recibiendo así los parámetros de la capa anterior. A la capa que recibe los parámetros de entrada se la denomina "capa de entrada", y a la última capa, que determina el valor de salida se la denomina "capa de salida". El resto de las capas que quedan entre la capa de entrada y la de salida son denominas "capas ocultas". Este sistema de capas se pude ver representado en la figura 3. La otra forma de colocar dos neuronas es de forma secuencial, es decir, una de las neuronas recibirá la información procesada por la neurona anterior, lo que permite a la red aprender conocimiento jerarquizado, gracias al cual las primeras capas pueden especializarse en elaborar conocimientos simples, que serán procesados en las siguientes capas elaborando cada vez un conocimiento más complejo y abstracto. La profundidad en el nivel de las capas es lo que da nombre al aprendizaje profundo o *deep learning.*

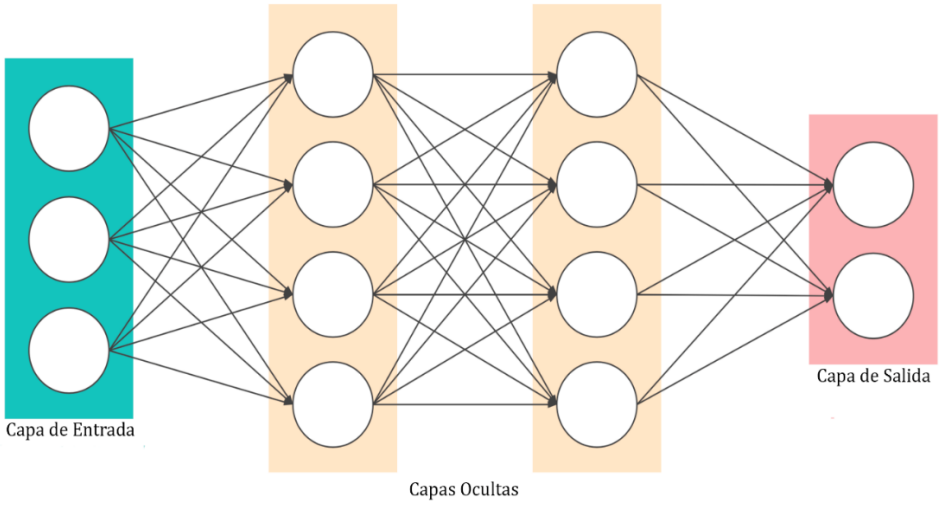

Figura 3 – Redes Neuronales Artificiales: Red Neurona
Imagen obtenida de www.medium.com

Un punto clave de la red neuronal es la función de activación. Como se ha explicado, cada neurona cuenta con una función de activación que calcula un valor que será entregado a las neuronas de la siguiente capa. Existen varias funciones de activación y la mayoría de ellas deben ser no-lineales; ya que, si las neuronas tuviesen como función de activación una función lineal, el resultado de la red podría simplificarse como la suma

de todas las funciones lineales, con lo cual, toda la red podría simplificarse como una única neurona que solamente sería capaz de resolver problemas lineales, perdiendo de esta forma el conocimiento jerarquizado y la capacidad para resolver problemas complejos.

Una de las funciones más simples es la función escalonada, que fija un valor umbral *y*, para cualquier valor inferior al valor umbral, el *output* de la función será 0. Por el contrario, si el valor de entrada es superior al valor umbral, el valor de salida será 1, tal y como se puede ver en la siguiente figura (Figura 4).

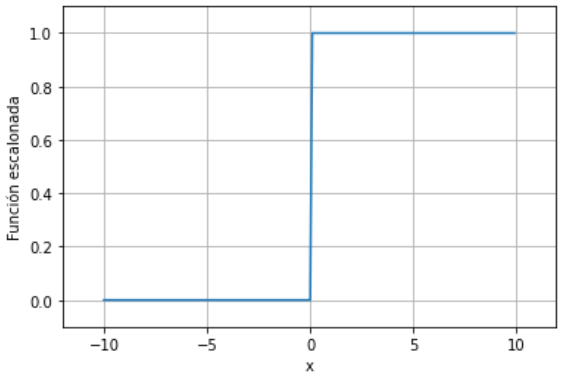

Figura 4 – Función escalonada

Esta función no produce un cambio gradual, sino que produce el cambio de forma instantánea, como un escalón, y eso perjudica el aprendizaje.

$$f(x) = \begin{cases} 0 \; si \; x < 0 \\ 1 \; si \; x \geq 0 \end{cases}$$

Ecuación 2 – Ecuación escalonada

La función sigmoide, al contrario que la función escalonada, produce un cambio de forma gradual; pero, al igual que esta, hace que los valores muy grandes saturen a uno y los valores muy pequeños saturen a cero.

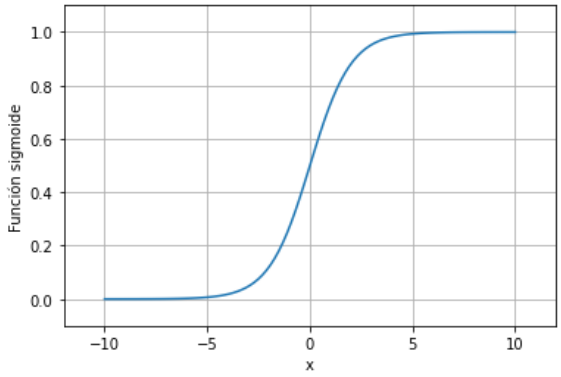

Figura 5 – Función sigmoide

Con lo que, además, sirve para representar probabilidades, pues el resultado siempre estará entre 0 y 1.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Ecuación 3 – Ecuación función sigmoide

La función de activación tangente hiperbólica es similar a la función sigmoide, pero cambiando su rango. Si la función sigmoide cuenta con un rango de $[0, 1]$, la función tangente hiperbólica cuenta con un rango $[-1, 1]$, tal y como se puede ver en la siguiente figura (Figura 6) y se demuestra con la ecuación 4.

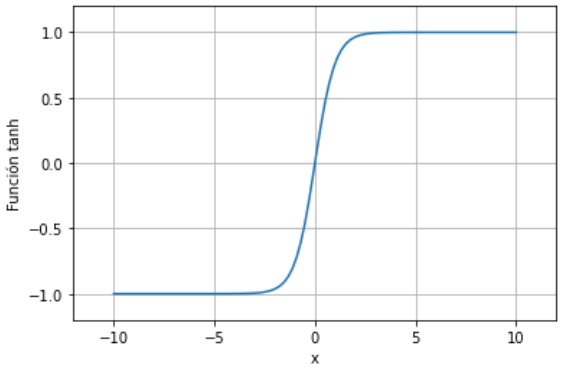

Figura 6 – Función tangente hiperbólica

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Ecuación 4 – Ecuación función tanh

Finalmente, se debe comentar la función de activación ReLU (Unidad Rectificadora Lineal), la cual tiene un comportamiento interesante, ya que actúa como una función lineal cuando el valor de entrada es positivo y es constante a cero cuando el valor es negativo; eliminando de esta forma los valores negativos que, en ocasiones, pueden no ser interesantes para el correcto funcionamiento de la red. Esta función, cuya representación se corresponde con la de la figura 7, se obtiene a partir de la ecuación (Ecuación 5) que se muestra a continuación.

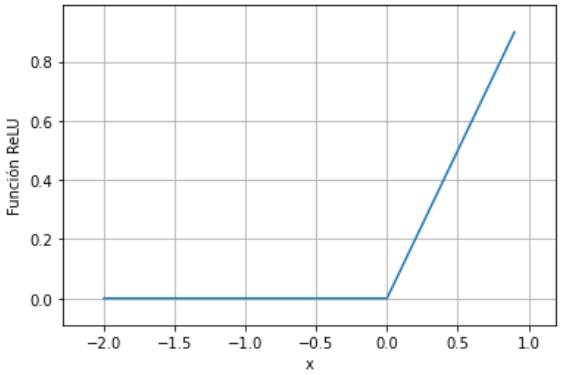

Figura 7 – Función ReLU

$$f(x) = \begin{cases} 0 \; si \; x < 0 \\ x \; si \; x > 0 \end{cases}$$

Ecuación 5 – Ecuación función ReLU

En la implementación que se ha realizado se ha utilizado concretamente una variante de la ReLU que se denomina Leaky ReLU, que sigue la premisa de no eliminar

completamente los valores negativos, ya que, su derivada cuando x toma valores negativos no es 0, sino que cuenta con una ligera pendiente, como se puede ver en la ecuación 6.

$$f(x) = \begin{cases} 0.01x & si\ x < 0 \\ x & si\ x > 0 \end{cases}$$

Ecuación 6 – Ecuación función Leaky ReLU

Llegados a este punto, se ha explicado que una red neuronal está compuesta por un conjunto de neuronas distribuidas en diversas capas, lo que le otorga a la red la capacidad de resolver problemas complejos y abstractos. Cada neurona cuenta con una función de activación mediante la que se modifican sus datos de entrada a través de la suma ponderada de estos con sus pesos correspondientes. Son los pesos los que a lo largo del entrenamiento se irán modificando y ajustándose para obtener el resultado deseado. El algoritmo que va modificando el valor de los pesos de las neuronas se denomina *backpropagation*, y es el que otorga a la red la capacidad del aprendizaje automático.

Este algoritmo surgió en 1986, cuando Rumelhart, Hinton y Williams redactaron el artículo *Learning representations by back-propagating errors* [6] y hasta este momento el *machine learning* era prácticamente una ciencia muerta, debido a que el algoritmo de aprendizaje automático que utilizaba el Perceptrón no era extensible a redes neuronales más complejas. Con la llegada de este algoritmo, se conseguía que una red neuronal autoajustara sus parámetros para así aprender una representación interna de lo que estaba procesando.

El algoritmo se basa en el método del descenso de gradiente [7], un algoritmo iterativo mediante el cual se puede ir minimizando el error hasta encontrar su mínimo, y emplea un ciclo de propagación en dos fases. En la primera fase, una vez se han aplicado unos valores de entrada estos se propagan a través de las capas de la red hasta generar una salida, que se compara con la salida deseada y, si estas no coinciden, se obtiene una señal de error. En la segunda fase, es la señal de error la que se propaga desde la salida hacia cada una de las neuronas de las capas ocultas, como se puede observar en la figura 8, que contribuyen directamente a la salida, repartiendo la señal de error en función de la influencia de la neurona en el resultado de la salida.

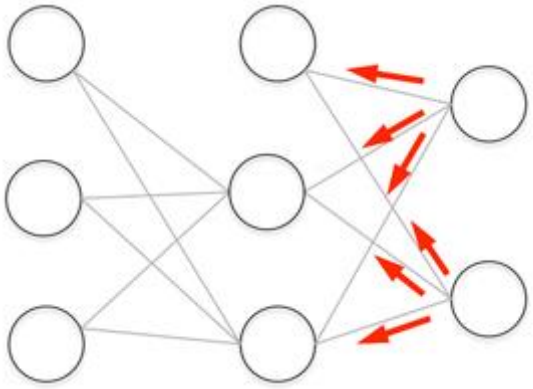

Figura 8 – Representación del algoritmo backpropagation
imagen obtenida de physics.bme.hu

Antes de existir el algoritmo de *backpropagation* el cálculo de la responsabilidad de cada neurona en el resultado se hacía utilizando la fuerza bruta, es decir, se preguntaba para

cada neurona cómo variaba el error cuando se introducía un cambio. Esta solución, además de muy exigente computacionalmente, era muy ineficiente. Gracias al algoritmo de *backpropagation*, propagando únicamente una vez el error hacia atrás se pueden obtener los errores de todas las neuronas que influyen en el resultado mediante los cuales se puede obtener el gradiente de toda la red neuronal y, con este, minimizar el error.

3.3. Paradigmas de Aprendizaje

Dentro del campo de *machine learning* existen tres grandes paradigmas dentro de los cuales se pueden englobar todos los algoritmos y técnicas de aprendizaje. Estos tres paradigmas son el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje reforzado, si bien es cierto que son los dos primeros los que engloban a la gran mayoría de algoritmos y técnicas.

3.3.1. Aprendizaje supervisado

El aprendizaje supervisado es un tipo de aprendizaje que se basa en descubrir la relación entre las variables de entrada y las variables de salida, es decir, el aprendizaje surge tras explicarle a estos algoritmos cuál es el resultado que se quiere obtener para un determinado valor de entrada. Tras mostrar al algoritmo una gran cantidad de opciones, este será capaz de obtener un resultado correcto incluso cuando se le muestre un valor de entrada que no ha visto nunca.

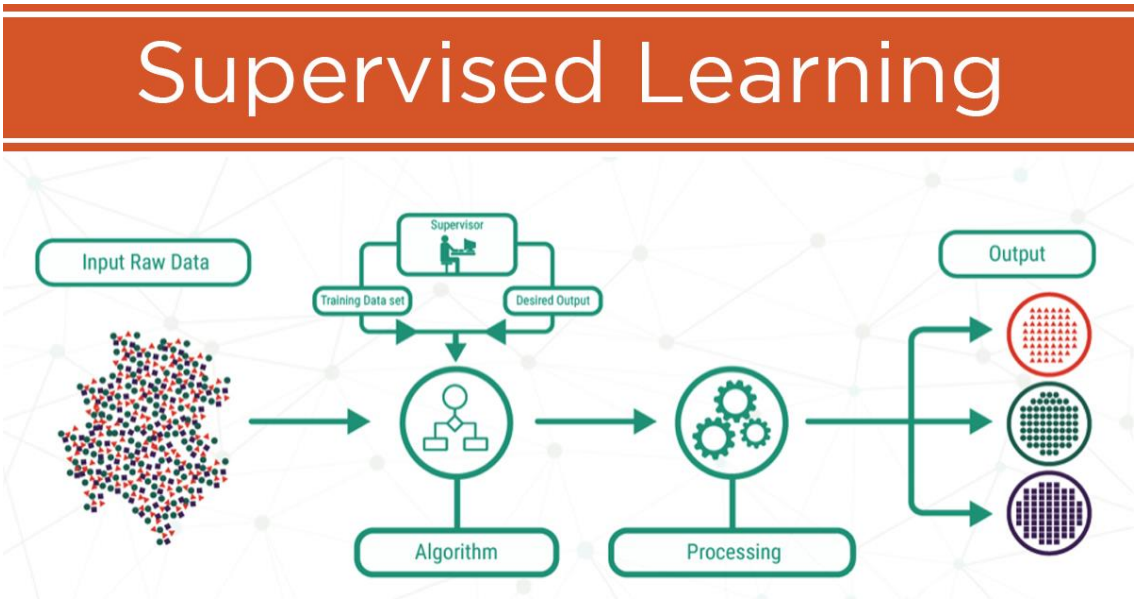

Figura 9 – Aprendizaje supervisado
(Imagen obtenida de www.gong-jj.com)

Para lograr obtener la relación entre los datos, los algoritmos de este paradigma requieren de un conjunto de datos que haya sido previamente etiquetado por un humano; de ahí el nombre de aprendizaje supervisado, en el que se especifique correctamente cual es la solución deseada para los datos de entrada especificados. Este conjunto de datos se conoce como conjunto de entrenamiento. A lo largo del entrenamiento, el algoritmo irá aprendiendo la relación entre los datos de entrenamiento y comprobando su aprendizaje calculando el resultado de datos con los

que no haya trabajado nunca, conjunto de test. Cuando esto se consiga el algoritmo, habrá aprendido. Este esquema es el que se muestra en la figura 9.

Dentro del aprendizaje supervisado también se distinguen el aprendizaje semi-supervisado, en el que a algunos de los datos de entrenamiento no están completos pero que aun así pueden ser usados para mejorar la calidad del modelo. Este tipo de aprendizaje está situado en un punto medio entre el aprendizaje no supervisado, que se tratará más adelante y el cual no utiliza datos de entrenamiento, y el aprendizaje supervisado.

Por otro lado, también se puede encontrar el aprendizaje débilmente supervisado en el que los datos son ruidosos, limitados o imprecisos. Este tipo de datos suele ser más sencillo y barato de conseguir, y por ello se puede utilizar un conjunto de entrenamiento de mayor tamaño.

3.3.2. Aprendizaje no supervisado

Por definición, el aprendizaje no supervisado es el paradigma que consigue producir conocimiento únicamente a partir de la información que se le proporciona a partir de los datos de entrada, sin necesidad de explicarle al sistema qué resultado queremos obtener. La dificultad de los algoritmos no supervisados es que no se cuenta con ningún tipo de respuesta para saber si el algoritmo está actuando correctamente. Pero, por el contrario, una gran ventaja de los algoritmos no supervisados frente a los supervisados es que los conjuntos de datos necesarios para entrenar el algoritmo son menos costosos de conseguir.

Lo que hace el paradigma del aprendizaje no supervisado es buscar patrones de similitud entre los datos de entrada, como se muestra en la siguiente figura (Figura 10). Los algoritmos más potentes dentro de este campo son capaces de identificar elementos comunes en los datos y reaccionar en función de la presencia o ausencia de dichos elementos comunes en cada nuevo bloque de datos.

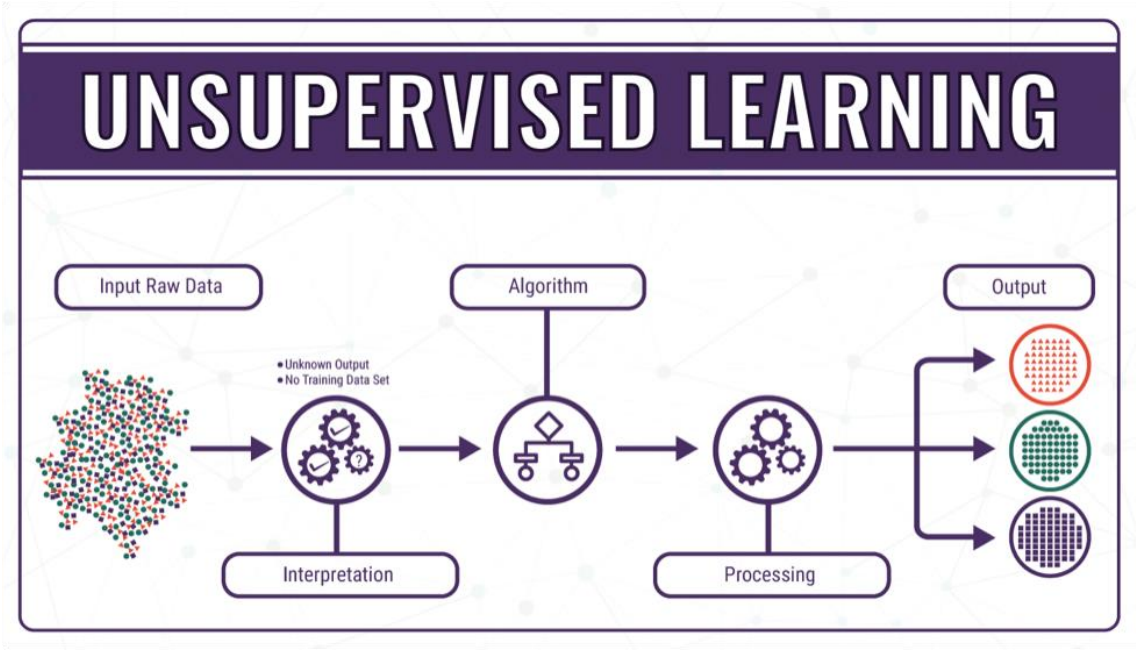

Figura 10 – Aprendizaje no supervisado
(Imagen obtenida de www.gong-jj.com)

3.3.3. Aprendizaje por refuerzo

Finalmente, el último de los tres paradigmas del *machine learning* es el aprendizaje por refuerzo. En esta clase de algoritmos, el desarrollador debe tomar medidas para maximizar una recompensa acumulativa gracias a la que el algoritmo va aprendiendo a través del ensayo y error.

Si el sistema no funciona y no es capaz de resolver la tarea correctamente, se le da un "premio" negativo. En cambio, si el sistema funciona correctamente y es capaz de resolver la tarea que está realizando se le otorga un "premio" positivo para, de esta forma, encontrar una buena solución final. Así, el sistema aprende cuando toma decisiones acertadas y cuando toma decisiones erróneas, como se puede ver en la siguiente figura (Figura 11).

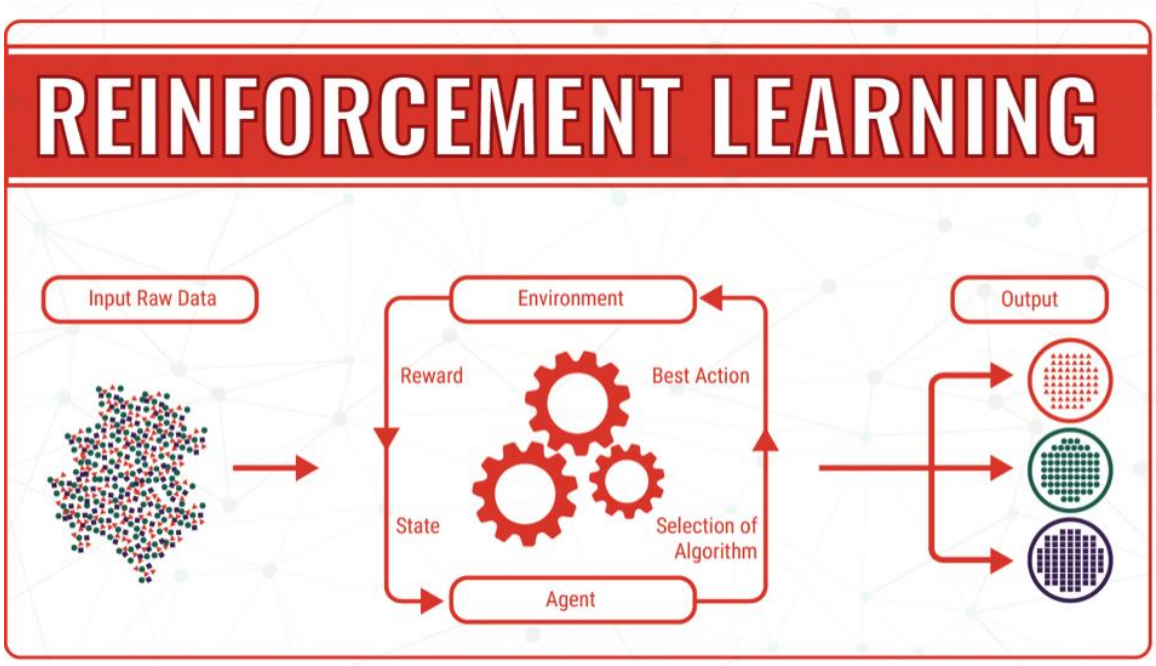

Figura 11 – Aprendizaje por refuerzo
(Imagen obtenida de www.gong-jj.com)

Si se realiza un entrenamiento adecuado, el sistema será capaz de tomar buenas decisiones y trasladar la perspectiva subjetiva humana a un proceso automático que mejorará continuamente su aprendizaje y su efectividad en las predicciones que tome.

3.4. Redes Neuronales Convolucionales

Las redes neuronales convolucionales o CNN (Convolutional Neural Networks) son un tipo de red neuronal artificial que se engloba dentro del paradigma del aprendizaje supervisado, por lo que las imágenes de entrada deben ser previamente etiquetadas por un humano. El nacimiento de estas redes se debe a la necesidad de poder procesar imágenes de una manera efectiva y eficiente. Su funcionamiento imita al ojo humano para identificar objetos o características de una imagen.

Como se ha explicado, lo que se busca es obtener la información más relevante y característica de la imagen. Para ello, la imagen pasa un proceso de compresión mediante el cual se extrae la información relevante que se puede observar en la imagen. De esta forma, mediante el proceso de entrenamiento la red neuronal irá detectando los patrones comunes en cada una de las imágenes de entrenamiento.

Para entender este funcionamiento, se debe recordar el funcionamiento de una imagen digital. Tomando como ejemplo una imagen que contiene, por ejemplo, el número 8, como la que se muestra en la siguiente figura (Figura 12), en un computador se representa como una matriz que va de 0 a 255 representando el valor de cada uno de los pixeles que forman la imagen.

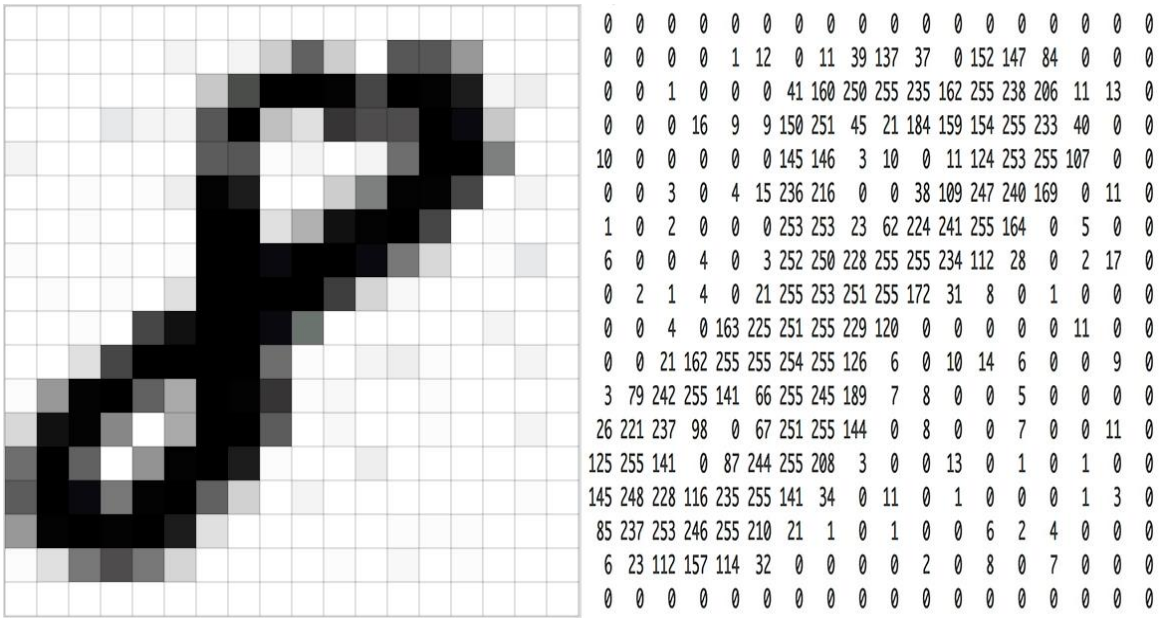

Figura 12 – Representación de una imagen digital
(Imagen obtenida de jetir.org)

Una vez se ha comprendido el funcionamiento de una imagen digital, se puede comprender cómo funciona una red convolucional. Partiendo de la imagen que actúa como dato de entrada, la red hará pasar la imagen por una primera de capa que cuenta con una serie de filtros que se irán especializando en extraer diferentes características de la imagen. Como se trata de la primera capa se especializará en extraer características simples (como pueden ser líneas o tonalidades dentro de la imagen). Tras los filtros existe un segundo tipo de capa que se conoce como *max polling*, que se encargará de eliminar aquella información que no es relevante, extrayendo de esta forma los datos que los filtros indicaron que eran más representativos. A medida que se avanza en la red, la imagen reduce su tamaño gracias a la capa de *max polling,* pero el número de filtros aumenta. En la Figura 13 se puede observar que la primera capa cuenta con 64 filtros, mientras que la segunda con 128 y la tercera con 256. Esto se debe a que las características que se extraen cada vez son más complejas, dejando de ser simples líneas o tonalidades y siendo capaz de extraer patrones complejos.

Esta estructura con forma de embudo en la que la imagen se va comprimiendo a medida que los filtros van extrayendo las características más importantes se conoce como *encoder* y se trata de una de las estructuras más importantes en el campo de la visión

artificial, ya que permite la identificación y clasificación de imágenes, lo que tiene una gran cantidad de aplicaciones en ámbitos como la conducción autónoma, la detección de objetos o en el campo de la medicina.

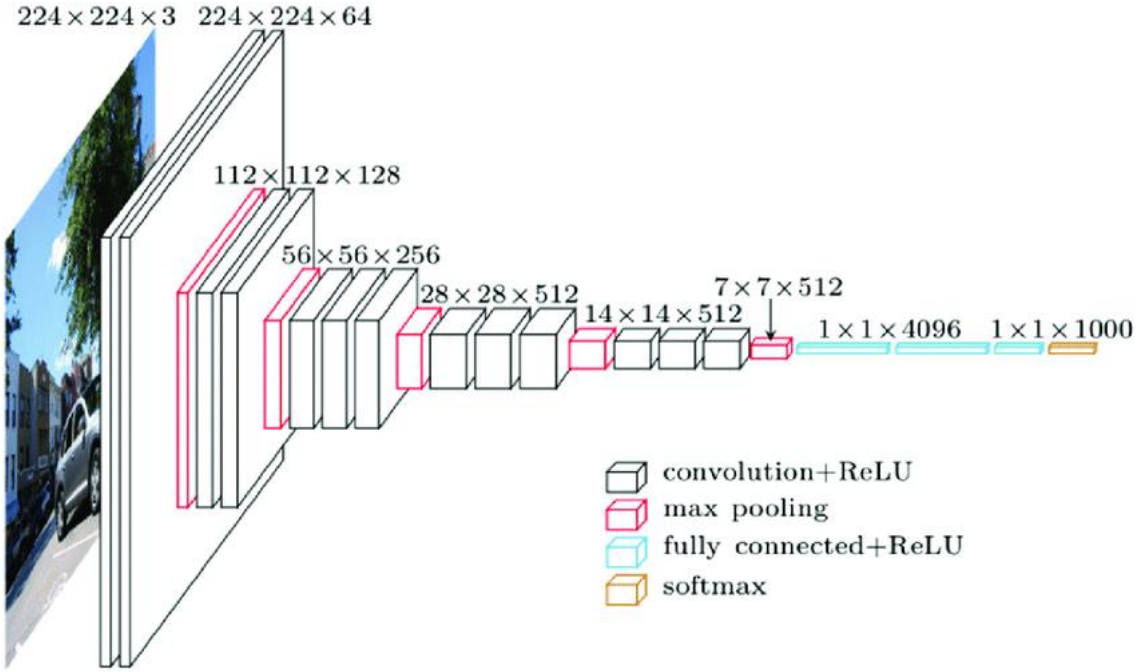

Figura 13 – Red Neuronal Convolucional
(Imagen obtenida de researchgate.net)

A pesar de que las redes neuronales convolucionales no son la solución al problema de la reconstrucción de las imágenes submarinas —ya que su función, como se ha comentado, es más de identificación o clasificación de imágenes que de generación—, su funcionamiento es interesante para comprender cómo las redes generativas adversarias son capaces de generar imágenes a partir de ruido, pues la estructura es similar a la vista en las redes convolucionales.

3.5. Redes Generativas Adversarias

Las redes generativas adversarias o GANs (Generative Adversarial Networks), por sus siglas en inglés, son un tipo de red neuronal artificial, que surgió en el año 2014 gracias a la insistencia de Ian Goodfellow, y tras su aparición en el campo del *machine learning* son consideradas la mayor revolución en los últimos años. Desde entonces su aportación en este campo únicamente ha ido en aumento.

Este tipo de redes neuronales están englobadas dentro del paradigma del aprendizaje no supervisado y su funcionamiento, sencillo pero potente, consiste en enfrentar a dos redes neuronales para que compitan en un constante juego de suma cero, es decir, la ganancia o la pérdida de una de las redes se compensa con la ganancia o pérdida de la opuesta.

De esta forma, la primera de las redes, la generativa, debe ser capaz de generar, normalmente a partir de ruido, datos completamente falsos; mientras que la segunda, la red discriminadora, debe tomar los datos generados artificialmente y tras compararlos con datos reales decidir si son reales o falsos, obteniendo a partir de esta comparación un vector de error, el cual conocerá la red generadora, lo que le permitirá ir mejorando en su tarea.

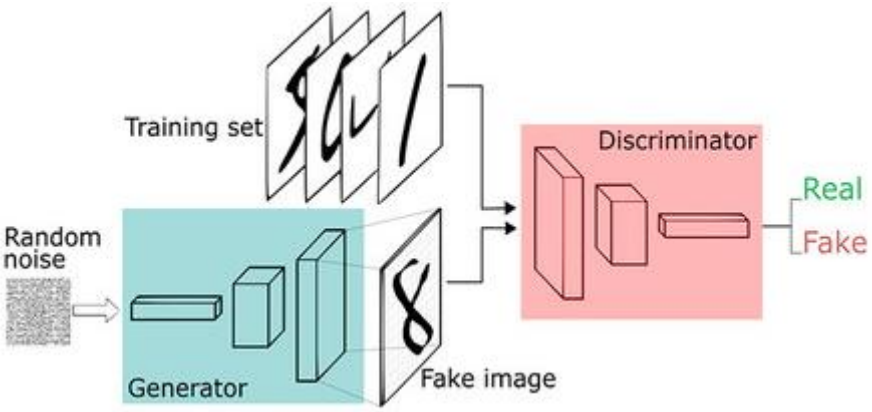

Figura 14 – Esquema GAN
(Imagen obtenida de cordobanoticias.net)

En la figura anterior (Figura 14), se muestra el esquema que sigue una GAN para conseguir crear, a partir de ruido aleatorio, imágenes correspondientes a los números del 0 al 9. Se puede observar que la estructura del generador es exactamente la misma estructura que se ha visto en el punto anterior de las redes convoluciones, pero en este caso la estructura está invertida, por lo que a ese tipo de red se le conoce como red deconvolucional, que no es más que un *encoder* invertido, un *decoder.* En este tipo de estructura todas las operaciones se invierten respecto a la estructura que se veía en las redes convolucionales: donde antes se iba reduciendo la información de la imagen ahora se va aumentando. De esta forma, el generador va prediciendo a partir de un valor inicial, que en este caso es ruido aleatorio, los valores que conformarían una imagen.

En primera estancia, cuando la red comienza el entrenamiento, el generador no tiene ningún conocimiento previo, por lo que generará una figura sin ningún detalle ni forma (parecida al ruido estático de las televisiones analógicas), como se puede ver en la imagen de la izquierda de la figura 15, que se corresponde con la primera época del entrenamiento de una red generativa. El discriminador comparará la imagen generada con una imagen de los datos de entrenamiento, determinando que la imagen es falsa. La red generadora únicamente sabrá el motivo por el cual esa imagen no es real; y, conociéndolo, irá mejorando en la tarea de generar imágenes. A medida que la red generadora mejora en su función y las imágenes generadas cada vez son más similares a las reales, la red discriminadora, a su vez, irá mejorando en su propia función de diferenciar imágenes cada vez más realistas, hasta llegar a un punto común en que ambas no puedan mejorar; momento en que la red discriminadora no distinga entre las imágenes generadas y las imágenes del conjunto de entrenamiento, por lo que la red generadora ya no podrá mejorar más y la red habrá sido entrenada por completo.

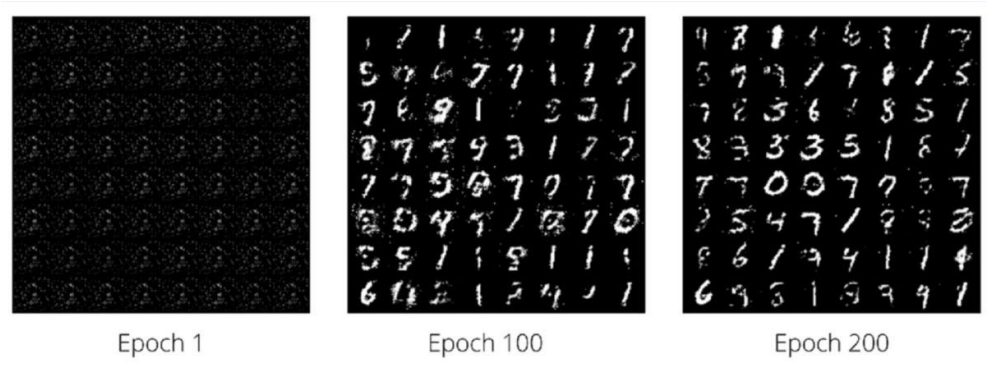

Figura 15 – Entrenamiento de una GAN
(Imagen obtenida de debbugercafe.com)

Se debe recordar que el problema planteado consistía en la reconstrucción de imágenes submarinas. La reconstrucción de imágenes se consigue generando nuevas imágenes, por lo que las redes generativas adversarias son el punto de partida perfecto para plantear la solución al problema. Si bien es cierto que, como se ha comentado, la red generadora crea los datos a partir de ruido aleatorio, por lo que el resultado a pesar de que será parecido a los datos del conjunto de datos también será aleatorio: no se puede controlar.

Por este motivo surgen las redes generativas adversarias condicionas, cGANs (Conditional Generative Adversarial Networks). Estas redes son una versión modificada de las GANs en las que además de introducir un vector de ruido al generador se concatena a este un vector de etiquetas, un vector de 1s y 0s donde la posición que sea 1 indica la etiqueta que se quiere generar. De esta forma, la red no solo aprenderá a generar datos a partir de ruido, sino que aprenderá a generar los datos en función del *input* que se le ha especificado a la red. Se está condicionando la salida en función de la entrada, condición que da nombre a la red.

Lo interesante de las cGANs es que no solo puedes condicionar la salida a través de un vector de etiquetas, sino que se puede condicionar a una cadena de texto, como es el caso de *StackGAN* [3], o, lo que realmente interesa para solucionar el problema planteado, condicionar la imagen de salida a una imagen de entrada, condicionando los pixeles de una imagen a los pixeles de otra imagen.

Con esta idea de trasladar los pixeles de una imagen a otra surgió en 2017 el articulo *Image-to-Image Translation with Conditional Adversarial Networks* [10] en el que los investigadores de Berkeley proponen el modelo *Pix2Pix* mediante el cual se ha implementado la solución del problema de la reconstrucción de imágenes submarinas.

3.6. Modelo *Pix2Pix*

El modelo *Pix2Pix* utiliza redes generativas adversarias condicionadas para partir de una primera imagen y conseguir una nueva imagen, trasladando de esta forma los pixeles de la primera imagen a un nuevo dominio. Este modelo cuenta con un gran número de aplicaciones, pues su abanico es muy amplio. Se puede usar tanto para convertir un dibujo trazado en una imagen real como para, por ejemplo, la restauración de fotografías en blanco y negro consiguiendo una imagen a color, como se puede observar en la figura 16.

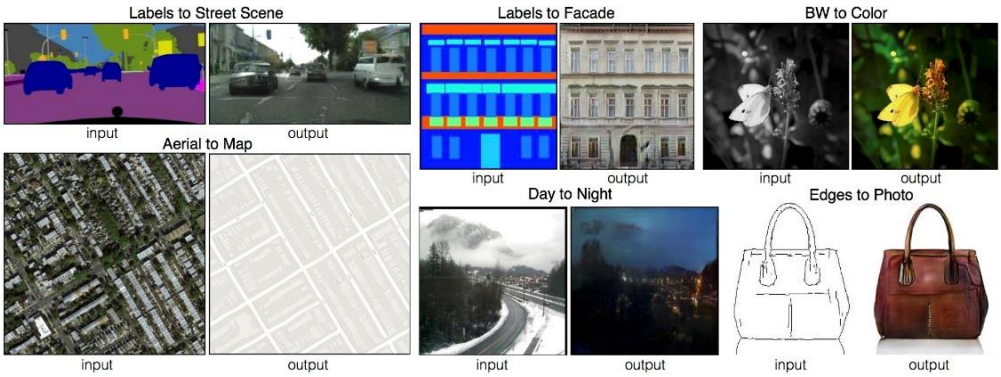

Figura 16 – Pix2Pix ejemplos de aplicación
Imagen obtenida de https://github.com/phillipi/pix2pix

Para conseguir estos resultados el modelo *Pix2Pix*, al igual que el resto de las redes generativas, utiliza un *decoder* a través del cual se introduce la información. Mediante una red convolucional invertida, esa información se va a ir incrementando capa tras capa hasta lograr predecir los pixeles de la imagen que se quiere generar. Pero, en este caso, se busca que la imagen de salida esté condicionada a una primera imagen de entrada, que también debe ser analizada por una red neuronal. Para lograr esto, antes del *decoder* se debe conectar un *encoder* que, al igual que ocurría con las redes convolucionales, irá procesando la imagen, eliminando la información que no es representativa a través de capas de filtros y de *max polling*, y quedándose con las características más representativas de esta. Esta arquitectura en la que la salida de un *encoder* se conecta a la entrada de un *decoder* se conoce como arquitectura en forma de reloj de arena, que es la forma que se puede observar en la figura 17.

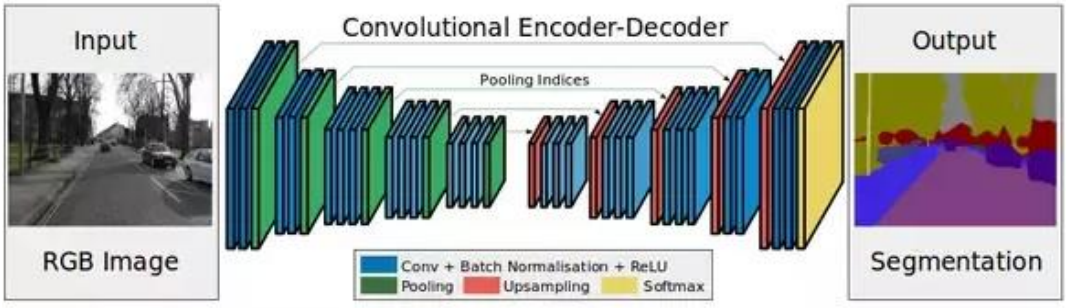

Figura 17 – Arquitectura del modelo Pix2Pix (Arquitectura U-NET)
Imagen obtenida de towardsdatascience.com

En este caso, lo que se está haciendo es tomar una imagen de entrada y comprimir la información de esta para que esa información comprimida, que especifica cuáles son las partes más representativas de la imagen de entrada, se pueda introducir en el *decoder* y a partir de ella se genere una nueva imagen. Pero existe un problema, y es que, como

la imagen que ha sido introducida en el *encoder* se comprime capa tras capa hasta llegar a la entrada del *decoder,* esta información, en ocasiones, no es suficiente para que el *decoder* pueda realizar su tarea correctamente. Es en este punto es donde tienen una gran relevancia las *skip connections*, conexiones de salto, que son las líneas de la figura 17 que unen el *encoder* con el *decoder.* Estas conexiones van a servir como atajo para que la información pueda llegar directamente a diferentes capas del *decoder*, y, de esta forma, la red neuronal deberá decidir en cada caso si con la información comprimida que le llega al *decoder* es suficiente o quiere utilizar la información "en crudo" que le proporcionan las *skip connections* para generar la imagen. Este tipo de conexiones que unen directamente diferentes capas de las dos redes que forman la arquitectura de reloj de arena se utiliza en la arquitectura U-NET, que es la que utiliza el modelo Pix2Pix, que básicamente concatena el *output* de cada nivel de procesamiento del *encoder* con su correspondiente nivel en el *decoder*.

El modelo *Pix2Pix* se ha utilizado para llevar a cabo la reconstrucción de las imágenes submarinas, ya que se busca condicionar la imagen de salida, una imagen clara y con mayor nivel de detalle, a la imagen dañada que recibe el piloto. Esto es lo que se tratará en el siguiente punto, que es el correspondiente a la implementación realizada.

4. Implementación: Reconstrucción de imágenes submarinas

A lo largo del capítulo anterior se han sentado las bases de las técnicas necesarias que se han utilizado para realizar esta implementación. Como se ha mencionado anteriormente, el problema planteado surge con las imágenes que recibe el piloto del dron submarino, y, por lo tanto, la solución que se busca es la reconstrucción de dichas imágenes. Estas imágenes cuentan con una muy baja resolución. Esto se debe a la baja capacidad que presenta el cable umbilical que une al dron con el piloto. Además de este problema, el efecto *scattering* y a la turbidez del agua causan que las imágenes que recibe el piloto presenten un aspecto borroso y con apenas nivel de detalle.

Figura 18 – Imagen de entrada

Por lo tanto, partiendo de una imagen de entrada de baja resolución, muy borrosa y sin apenas detalles, imitando a la foto que recibiría el piloto a través del cable umbilical (figura 18), se busca enseñar al sistema a generar imágenes con un mayor nivel de detalle y visibilidad. Esto se hará a través de una red generativa adversaria condicionada donde el discriminador comparará las imágenes generadas con imágenes de alta resolución (Figura 19), que serás las respectivas en alta calidad a la imagen de entrada que condiciona la generación.

Figura 19 – Imagen base

Como la imagen de entrada es de muy bajo nivel y sin prácticamente nivel alguno de detalle y el sistema tiene que aprender, prácticamente de cero, a generar imágenes con un nivel de detalle mucho mayor y mayor resolución, para la resolución de este problema se van a utilizar, como se ha dicho, redes cGAN que, como se ha explicado en el capítulo anterior, son un tipo de red que cuenta en realidad con dos redes neuronales que van a desempeñar funciones completamente diferentes. Mientras que una de ellas, la red generadora, se va a encargar de generar imágenes artificiales partiendo la de la imagen de entrada (Figura 18), la otra red, la discriminadora, se va a encargar de comparar la imagen generada por la primera red con la imagen base del *dataset* (Figura 19) y va a aprender a diferenciar si la imagen es real o ha sido generada artificialmente. Con lo cual, la red generadora tiene que ir mejorando su capacidad para crear imágenes artificiales para lograr engañar a la red discriminadora. La condición de la cGAN es que no va a generar imágenes aleatorias, sino que las va a generar en función de la imagen de entrada. Por lo tanto, la red aprenderá a generar la imagen correspondiente en función del *input* que se le suministre.

Para resolver este problema, existen dos tipos de implementación posibles. Por un lado, utilizar una reconstrucción en dos pasos, donde, en un primer paso, una red se encargue de reconstruir la imagen inicial eliminando los efectos producidos por el agua y consiguiendo una imagen con cierto nivel de detalle y visibilidad. Así, posteriormente, en un segundo paso, otra red se encargue de aumentar la resolución de la imagen generada mejorando de esta forma la calidad y aumentando el nivel de detalle de esta. Y, por otro lado, utilizar una reconstrucción en un único paso, donde una única red reconstruya la imagen y aumente la resolución de esta.

4.2. Implementación en dos pasos

Esta primera implementación que se ha realizado cuenta con dos redes generativas adversarias, es decir, en vez de usar una única red que realice tanto la reconstrucción como el aumento de resolución se han separado cada una de las tareas obteniendo al final una red que se encargue únicamente de realizar la reconstrucción (*reconsGAN*) y otra que se encargue de realizar el aumento de resolución (*srGAN*).

4.2.1. Reconstrucción de la imagen (reconsGAN)

Datos de entrenamiento

Como punto de partida para la resolución de este problema se contaba con alrededor de dos horas de grabación del fondo marino a una resolución de *Full HD,* $1920\ x\ 1080$ *pixeles*. Como a la hora de trabajar tanto la reconstrucción como el aumento de resolución se debe trabajar a nivel de *frame* se utilizó la biblioteca de visión artificial *OpenCV* para extraer los fotogramas de cada uno de los videos y de esta forma contar con un *dataset* de imágenes *per se*. El problema es que al haber extraído las imágenes de los videos la mayoría de estas imágenes son prácticamente idénticas al tratarse de fotogramas consecutivos, por lo que se han seleccionado una de cada cien imágenes del conjunto total, quedando de esta forma un conjunto de datos final de 1078 imágenes con el que se procederá a trabajar.

Como estas imágenes se corresponden con la imagen base, con un alto nivel de detalle, se ha usado la herramienta *Photoshop* para trasformar el *dataset* en imágenes similares a las que recibiría el piloto durante la sesión con el dron submarino. Para transformar esta imagen se le ha modificado el nivel de los canales RGB (concretamente, se ha disminuido el nivel de los canales Rojo (R) y Azul (B) y se ha aumentado el nivel del canal verde (G) para que el deterioro de la imagen se asemeje al que produce el efecto *scattering* del agua). Además, se han superpuesto capas no uniformes, con una opacidad en torno al 40%, para enturbiar la imagen. Posteriormente, se ha reducido la resolución de $1920\ x\ 1080$ a $300\ x\ 170$ pixeles, disminuyendo de esta forma el nivel de detalle y empeorando la calidad de la imagen; siendo el resultado final el que se puede ver en la figura 20.

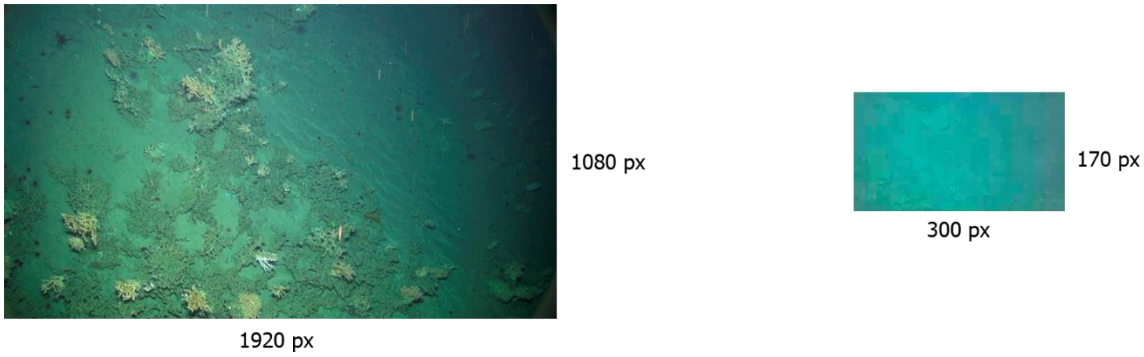

Figura 20 – Transformación de los datos

Tras este proceso se cuenta con dos conjuntos de imágenes bien diferenciados: por un lado, está el conjunto de datos de entrada formado por las imágenes modificadas mediante *Photoshop,* que serán las imágenes a partir de las cuales la red debe aprender a reconstruir para obtener una imagen con un mayor nivel de detalle; es decir, el *input* que recibe la cGAN para condicionarla a generar el resultado deseado. Y por otro lado está el conjunto de datos de las imágenes originales que actuarán como imagen objetivo (*target*) y serán las imágenes que utilizará la red discriminadora para compararlas con las imágenes resultantes de la red generadora y determinar si estas son reales o no.

Preprocesado de los datos

En el punto anterior se ha explicado cómo se han obtenido dos conjuntos de datos que se utilizarán para entrenar la red encargada de realizar la reconstrucción de la imagen. Pero, antes de que la red comience a trabajar con estos conjuntos de datos, estos van a sufrir un procesado previo.

En primer lugar, como el modelo *Pix2Pix* trabaja a nivel de *pixel* las imágenes sufrirán un reescalado, convirtiendo ambas imágenes en imágenes de tamaño $256\ x\ 256$. Como esta red (*reconsGAN*) únicamente se va a encargar de realizar la reconstrucción de la imagen, es más sencillo computacionalmente realizar esta reconstrucción con una menor cantidad de *pixeles*.

Posteriormente, y una vez que las imágenes han sido reescaladas, para que la computación sea más estable las imágenes se van a normalizar. Esto quiere decir que, si las imágenes vienen representadas como matrices de números racionales en un rango de $[0,\ 255]$ se van a modificar para trabajar con ellas en el rango de $[-1,\ 1]$. Esto se hace realizando la siguiente operación (ecuación 7), ya que al dividirlas por la mitad de 255 (127.5) las imágenes se quedarían en un rango de $[0, 2]$ y restándoles 1 pasarán a estar en el rango deseado.

$$image = (image/127.5)\ \ -1$$

Ecuación 7 – Ecuación normalización

Finalmente, la última transformación que sufrirán algunas imágenes es una que se recomienda en el artículo en que se explica el modelo *Pix2Pix* [10], y es una función que en el *paper* se denomina *random_jitter* y que se encarga de aumentar los datos del *dataset* original; es decir, partiendo de este *dataset* se realizan a las imágenes transformaciones aleatorias que las perturben, y de esta forma no se condiciona a la red a encontrar las imágenes de una determinada forma y posición.

En el *paper* se explica, que como primer paso se tome la imagen de partida, que en este caso es de $256\ x\ 256$, y se aumente ligeramente su tamaño (en el caso de esta implementación se ha aumentado el tamaño un 10%). Este proceso se puede observar en la figura 21.

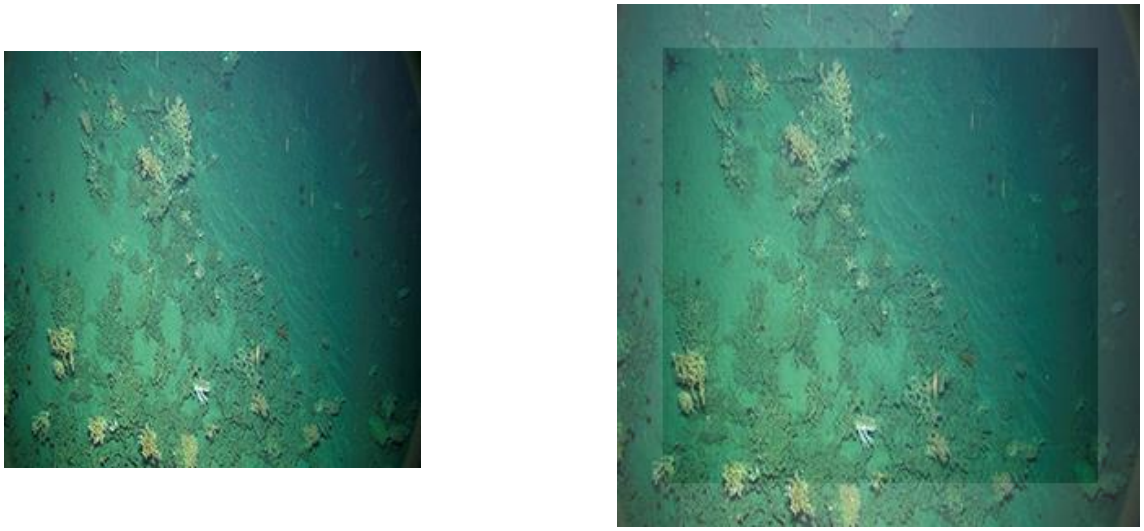

Figura 21 – Aumento de la imagen del 10%

Y, una vez la imagen se ha ampliado la imagen ligeramente, como segundo paso se seleccionará un fragmento de $256\ x\ 256$ aleatorio de la imagen aumentada, proceso que se puede ver en la figura 22.

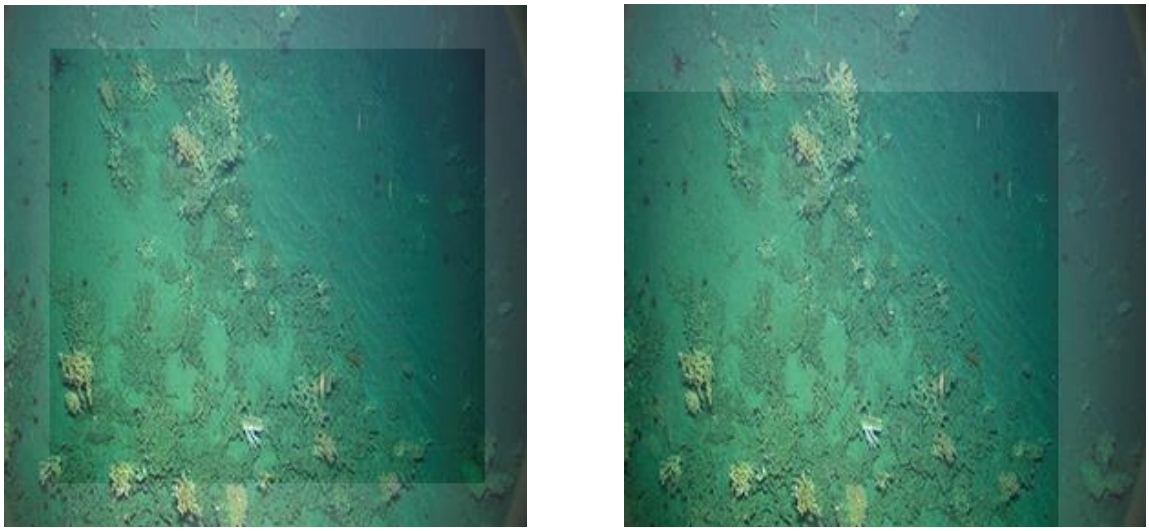

Figura 22 – Sesgado aleatorio de la imagen

Además, la función realiza al 50% de las imágenes un volteo horizontal de izquierda a derecha. Con esto se consigue tanto que el conjunto de datos aumente de forma artificial como que, como se ha comentado, no se condicione a la red de que las imágenes de entrada tengan que ser de una forma concreta.

Con esto, los datos ya han sido modificados y están listo para usarse en el entrenamiento de la red. Como se ha mencionado en el apartado anterior, el conjunto de datos con el que se cuenta es de 1078 imágenes. De estas 1078 imágenes se usará el 80% para el entrenamiento y el 20% para las pruebas.

Implementación de la red

El modelo *Pix2Pix*, como se ha visto, es un modelo generativo adversario, es decir, cuenta con dos redes enfrentadas en la que una, la generadora, trata de engañar a la otra, la discriminadora, en la creación de images artificiales. Además, este modelo genera la imagen condicionado a la imagen de entrada. Esto, como se ha explicado, lo hace gracias a su arquitectura U-NET, donde el generador cuenta con una red convolucional, *encoder*, para extraer las características más representativas de la imagen y seguido de este se encuentra una red deconvolucional, *decoder*, que a partir de la información extraída por el *encoder* generará la nueva imagen. Si la información que obtiene el *decoder* para generar la nueva imagen es insuficiente, la red puede hacer uso de la información que obtiene a partir de las *skip connections* y, de esta forma, contar con más información para generar la imagen.

Red generadora

En el *paper* donde se explica el modelo *Pix2Pix* se puede encontrar un anexo en donde se explica la estructura que deben seguir tanto el *encoder* como el *decoder*, se explican el número de capas que forma cada una de estas redes y como es la estructura de cada una de las capas. Por lo tanto, si se sigue la estructura que se indica en el anexo, el *encoder*, se organiza de la forma que se puede observar en la ecuación 8, en la que cada

bloque Ck se corresponde con un bloque formado por tres capas diferentes. En primer lugar, hay una capa de convolución que va a analizar la imagen, en segundo lugar, una capa de *batch normalization* que se encarga de realizar una normalización a nivel de lote y finalmente hay una capa de activación donde se puede encontrar una función ReLU en su variante Leaky ReLU. El número que acompaña indica el número de filtros que forman las capas de convolución que, como se vio en la explicación de las redes convolucionales, va aumentando de tamaño. De esta forma, la red es capaz de identificar patrones cada vez más complejos.

$$C64 - C128 - C256 - C512 - C512 - C512 - C512 - C512$$

Ecuación 8 – Estructura del encoder

Una vez conocida la estructura del *encoder* lo que se ha realizado a nivel de código es crear una pequeña función modular para crear cada uno de estos bloques en función del número de filtros y, de esta forma, poder generar el *encoder* de una manera más sencilla. Además, el *paper* especifica una serie de atributos de cada una de las capas, como, por ejemplo, que las convoluciones realizan un *downsample* (reducción de la muestra) de factor 2; es decir, que en cada paso se reduce el tamaño de la imagen a la mitad; que el primer bloque de 64 filtros no utiliza una capa de *batch normalization* o que los pesos, cuando se inicializan, deben inicializarse siguiendo una distribución Gaussiana de media 0 y desviación típica 0.02.

Una vez se ha creado la función modular que permitirá crear la parte del *encoder* de la arquitectura U-NET, se debe realizar la misma operación con la parte del *decoder*. Para esta otra sección, el artículo especifica una nueva estructura, que es la que se puede observar en la ecuación 9. En este caso, el *decoder* combina dos tipos de bloque diferentes: unos que son los denotados como CDk y que están formados, en este caso, por cuatro capas diferentes. La primera una capa de deconvolución o convolución inversa, es decir, en este caso en vez de realizar un *downsample* de la imagen se realizará un *upsample* de esta en un factor 2, el tamaño de la imagen se doblará en cada paso. La segunda capa con la que cuenta este tipo de bloques es una capa de *batch normalization*; la tercera es una capa de *dropout* del 50% cuya función es desactivar aleatoriamente una serie de neuronas, por lo que en cada iteración habrá una serie de neuronas que no se tendrán en cuenta ni en la propagación hacia delante ni en la posterior propagación hacia atrás (*backpropagation*) por lo que se obliga a las neuronas cercanas a no depender tanto de las neuronas desactivadas y de esta forma se ayuda a reducir el overfitting*. Finalmente, la última de las capas que forma este bloque es la capa de activación, que en este caso no se trata de una función Leaky ReLU, sino que simplemente es una ReLU. El otro tipo de bloque, son bloques idénticos a los que se encontraban en el *encoder,* pero en vez de que la información disminuya, esta aumentará.

$$CD512 - CD512 - CD512 - C512 - C256 - C128 - C64$$

Ecuación 9 – Estructura del decoder

*El overfitting es un sobre entrenamiento del algoritmo de aprendizaje lo que causa que la red funcione muy bien para unos determinados datos, generalmente los datos de entrenamiento, ya que termina memorizando estos datos sin llegar a generalizar a partir de estos para reconocer nuevos datos.

De modo que, al igual que con el *encoder,* se ha realizado una función modular para el *decoder,* dejando como parámetros de entrada el número de filtros y la aplicación o no de la capa de *dropout.* Con esto, se han creado dos funciones que describen cada uno de los bloques que forman tanto en *encoder* como el *decoder* dentro de la arquitectura de la U-NET, por lo que en este punto lo que hay que hacer es conectarlo y juntar todos estos bloques que en conjunto definen la red generadora.

Para ello, lo que se ha realizado a nivel de código es una nueva función, que se ha denominado *Generator*, a la que se le especifica en primer lugar el tipo de entrada que recibirá la red, que en este caso se trata de una imagen. A partir de ahí se define una lista que engloba a todas las capas correspondientes al *encoder*, especificando el número de filtros de cada una de ellas. Y, mediante un bucle *for*, se recorre esta lista haciendo que la salida de cada uno de los elementos sea la entrada del siguiente. Este proceso para conectar las capas se realiza también con las capas del *decoder*.

Además, al igual que se ha especificado la capa de entrada se debe especificar la capa de salida. Esta última capa será una capa deconvolucional al igual que las demás capas del *decoder,* pero tendrá una función de activación *tanh*, ya que la salida de este tipo de función de activación está englobada en el rango de $[-1,1]$, que si se recuerda es justamente el rango al que se habían normalizado las imágenes.

Finalmente, para terminar de formar la arquitectura U-NET queda habilitar las *skip connections*, para que de esta forma la red pueda decidir si quiere tomar la información comprimida que sale del *encoder* o tomar momentos previos de esta información en capas anteriores del *encoder.*

Siguiendo la implementación que especifica el *paper*, la red resultante es la que se puede ver en la figura 23, en la cual se pueden distinguir perfectamente la estructura de embudo de la red convolucional que se encarga de ir extrayendo las partes más representativas de la imagen, y su versión deconvolucional que se encarga de ir generando la información hasta generar la imagen final. Dicha imagen pasará, una vez generada, a la otra red que forma parte de la arquitectura de la red generativa adversaria: la red discriminadora.

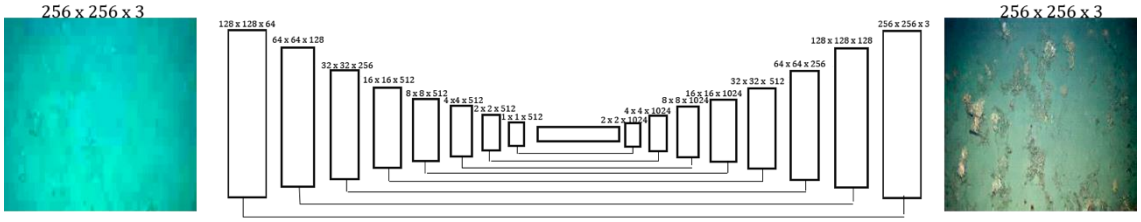

Figura 23 – Estructura red generadora

Si se realiza una prueba de la red generadora, el resultado que se obtiene se corresponde con el que se puede observar en la figura 24. En la imagen, se puede apreciar que esta está creada a partir de ruido aleatorio y se puede intuir la imagen a la que estaba condicionada.

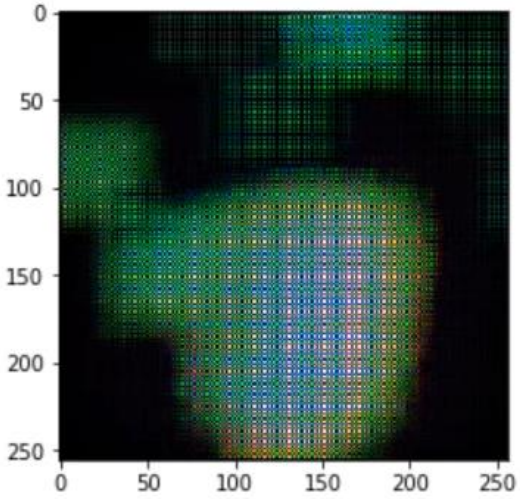

Figura 24 – Resultado test generador

Red discriminadora

La red discriminadora es la encargada de tomar la salida del generador y decidir si esta imagen es real o falsa a partir de lo que ella conoce como real. La red discriminadora que se especifica en el *paper* no es una red convolucional típica cuya salida sería un escalar, un 1 o un 0, que indique si la imagen es real o la imagen es falsa, sino que es un tipo de red convolucional que se conoce como *patchGAN*. Este tipo de red convolucional se fundamenta en la misma idea que la red convolucional típica, solo que, en vez de expresar el resultado como un escalar, determinando si la imagen es real o falsa, lo que devuelve es una cuadrícula donde se evalúan diferentes porciones, diferentes parches, de la imagen original. Con lo cual, en vez de obtener un único valor final, se obtiene una cuadrícula donde se indica qué partes de la imagen sí son realistas y cuáles no.

Para implementar esta red nuevamente se puede consultar el *paper* porque, al igual que pasaba con la red generadora, viene explicada su implementación. La *patchGAN*, como se ha explicado, no deja de ser una red convolucional que, en vez de terminar con un valor final, se corta antes para terminar con una especie de mapa de calor donde se especifiquen qué partes de la imagen creada son reales y cuales no en su comparación con la imagen original. La estructura de bloques que marca el artículo (ecuación 10), es similar para la red discriminadora a la que se vio para la red generadora, donde los bloques Ck estan formados por una primera capa de convolución en donde k, nuevamente, indica el número de filtros de esta capa. Tras la capa de convolución hay una capa de *batch normalization* y seguida de esta la capa de activación. Dicha capa es, en el caso del discriminador, como en el *encoder, Leaky ReLU.* Con lo cual, como su estructura es idéntica a la vista en el *encoder*, se puede coger la función que se creó para ese tipo de bloques y usarla para la función del discriminador.

$$C64 - C128 - C256 - C512 - C512$$

Ecuación 10 – Estructura patchGAN

Esta función, a la que se ha denominado *Discriminator*, cuenta con dos entradas diferentes; pues debe tomar la imagen generada por el generador para determinar si es real o falsa. Pero, al igual que el generador, está condicionado a la imagen real con la

que comparará la imagen generada. Por lo tanto, también debe tener acceso a dicha imagen.

Entonces, en primer lugar, se debe especificar que el discriminador recibirá como entrada dos imágenes, y como para pasarlas por la red no se puede pasar primero una y luego la otra, lo que se va a realizar es una concatenación de las dos. Por lo demás, el discriminador actúa como el *encoder* del generador, ya que tampoco en la primera capa se debe aplicar el *batch normalization.*

Finalmente, el *paper* especifica que se debe aplicar una última capa con convolución unidimensional, cuya función de activación debe ser una función sigmoide, porque se necesita únicamente un canal de información ya que, como se ha explicado, la salida será una imagen en la que se indicarán qué pixeles, por grupos, son reales y cuales no lo son.

Si, al igual que se hizo con la red generadora, se prueba el discriminador, en el resultado que se obtiene se puede observar el mapa que se forma y, como se indica, qué partes son más reales y cuáles no (figura 25).

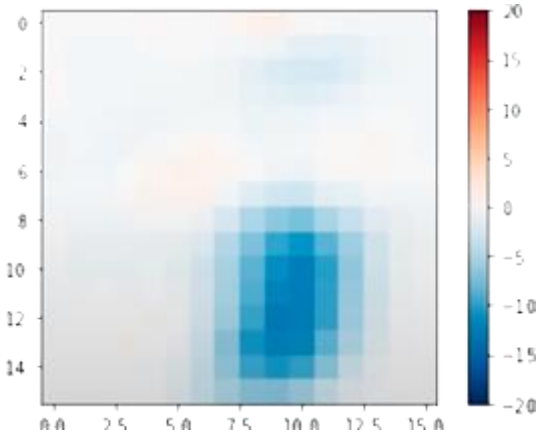

Figura 25 – Salida test discriminador

Los resultados que se muestran tanto en la figura 24, como en la 25 no son relevantes, ya que ni el generador ni el discriminador están entrenados. En este punto, ya se han descrito los elementos más importantes de la red generativa: los dos bloques que conforman el sistema de redes generativas adversarias. Lo que falta es diseñar una función de coste que va a unir el resultado de una con el resultado de la otra, lo que va a conseguir que durante el entrenamiento compitan entre ellas.

Funciones de coste y optimizador

Las funciones de coste van a permitir que cada una de las redes conozca el resultado de la otra y, gracias a esto, ver los puntos en los que fallan y poder ir mejorando en su tarea. Para generar las funciones de coste, a nivel de código, se va a hacer uso de la entropía cruzada binaria que va a calcular la entropía cruzada de cada uno de los pixeles de las imágenes que se están obteniendo. A partir de ahí se va a evaluar el comportamiento del discriminador y del generador por separado.

Al evaluar las pérdidas del discriminador, se van a tener en cuenta dos pérdidas diferentes, como se puede ver en la figura 26. Por un lado, está la pérdida real, que utiliza el resultado de la entropía cruzada para evaluar la diferencia entre dos cosas: el resultado del discriminador al evaluar una imagen real y el resultado perfecto. Este resultado sería una imagen, como la de la figura 25, en la que todos los pixeles indicaran que el resultado es real, es decir, todos los pixeles estarían a 1. Por lo tanto, se compara el resultado del discriminador al observar una imagen real con una matriz de 1s. La otra pérdida que se evalúa es la pérdida de salida del generador. Aquí lo que se compara es la diferencia entre el resultado del discriminador al evaluar la imagen que sale del generador y una imagen nuevamente como la de la figura 25, pero que en este caso indicará que toda la imagen es falsa, es decir, una matriz de 0s. Estos dos componentes, la real y la del generador, evalúan el comportamiento del discriminador. Por lo tanto, el resultado de cada una de ellas se suma y así se tiene el comportamiento del discriminador.

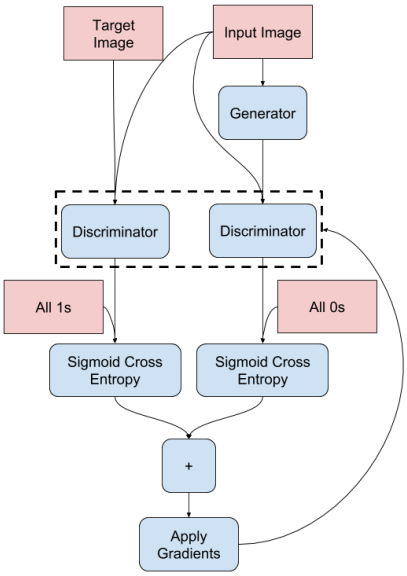

Figura 26 – Representación pérdidas discriminador

Por otro lado, el comportamiento de las pérdidas del generador, que se puede ver representado en la figura 27, es muy diferente. El generador tiene dos objetivos: el primero es generar la imagen más realista y el segundo es maximizar el error del discriminador. Entonces, por un lado, está el error adversario, que compara la imagen generada con lo que sería una imagen completamente real, es decir, una imagen de todos 1s. Este tipo de pérdida indica al generador si está engañando o no al discriminador. Y, por otro lado, está la pérdida absoluta que se obtiene de la comparación entre la imagen generada y la imagen real. En este caso, no ocurre como en el discriminador, en el que ambas pérdidas se sumaban por partes igual, sino que el *paper* especifica un hiperparámetro* *lambda* que indica si se quiere tener en cuenta más una pérdida u otra. En este caso, en el *paper* se propone un valor de *lambda* de 100, lo que significa que se estaría dando cien veces más peso a la perdida absoluta que al error adversario.

*Un hiperparámetro es parámetro que se configuran antes del entrenamiento pero que no forma parte del modelo como tal.

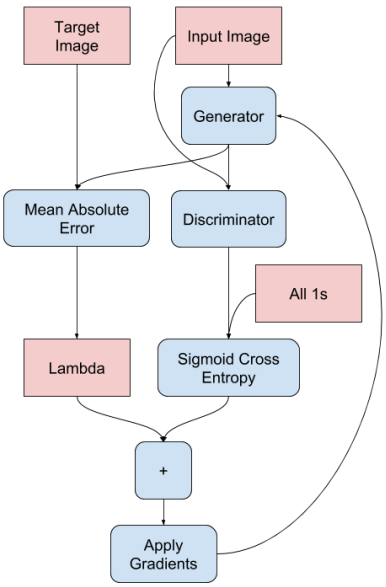

Figura 27 – Representación pérdidas generador

En este punto, se ha definido casi completamente el sistema, quedando solo por definir los optimizadores y así comenzar el entrenamiento.

Como se explicó en el capítulo anterior, es el algoritmo de *backpropagation* el que permite a la red que aprenda mediante la variación de los pesos de cada una de las neuronas, obteniendo el error final y buscando minimizarlo: es decir, buscar el valor óptimo de los pesos. Existen diferentes formas de optimizar estos pesos, en el *paper* con el que se está trabajando se indica que se utilice el optimizador de Adam (*Adaptive Moment Estimation*) además de los hiperparámetro de este.

Entrenamiento

El entrenamiento es el proceso que debe llevar a cabo la red para poder optimizar sus pesos y de esta forma lograr obtener los resultados esperados. Para esto se debe crear una función que junte todos los diferentes bloques que se han ido realizando (generador, discriminador, funciones de coste y optimizador). Esta función tendrá como parámetros de entrada las dos imágenes, es decir, la imagen entrada y la que se quiere obtener. Entonces, en primer lugar, el generador tomará la imagen de entrada y a partir de esta generará una nueva imagen. El discriminador tomará tanto la imagen generada como la imagen objetivo y comparará ambas imágenes. Las funciones de coste determinarán lo real o falsa que es la imagen generada en función de la salida del discriminador y del generador. Con esto, se ha realizado toda la propagación hacia delante, pero queda el proceso de *backpropagation*. Para realizar este proceso, se definen dos *GradientTape*, que es una herramienta de *keras* que permite de forma automática ir calculando los gradientes de cada una de las redes para poder realizar el proceso de optimización a través de los optimizadores definidos. Con esto se ha definido la función encargada del entrenamiento y ahora se debe realizar el mismo.

Resultados

En la implementación que se ha realizado, como se ha mencionado, se contaba con 1078 imágenes en total, de las que 862 serán imágenes de entrenamiento que se van a ejecutar durante 300 épocas. Una época es una propagación hacia delante y una propagación hacia atrás de todos los datos de entramientos, es decir, las 862 imágenes. El resto de las imágenes, las imágenes de test se van a utilizar para comprobar el funcionamiento de la red y ver cómo reacciona antes imágenes nunca vistas al final de cada época.

Durante el proceso de entrenamiento se mostrará el resultado que se obtiene al aplicar la red generadora al *dataset* de test, siento las siguientes figuras (figuras 28, 29 y 30), algunos de los resultados obtenidos durante este proceso.

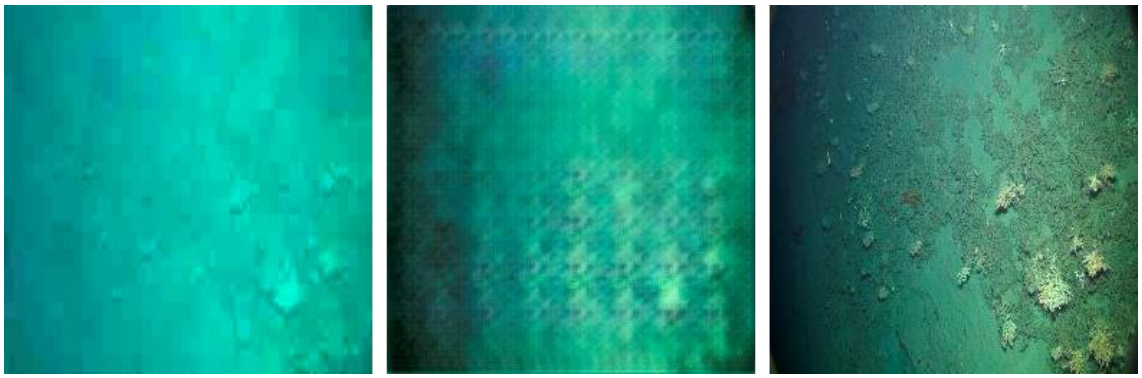

Figura 28 – Imagen de entrada, imagen generada, imagen objetivo. Época 0

Figura 29 - Imagen de entrada, imagen generada, imagen objetivo. Época 100

Figura 30 - Imagen de entrada, imagen generada, imagen objetivo. Época 200

Finalmente, tras concluir el entrenamiento, los resultados que se han conseguido (figuras 31 y 32) muestran que la reconstrucción obtenida cuenta con un buen nivel de detalle y que se ha recuperado completamente el contenido de la imagen. La red generadora ha logrado solventar con éxito tanto el efecto *scattering* del agua como su turbidez, y la imagen reconstruida se ve con claridad. Si bien es cierto que, en algunas ocasiones, la red presenta dificultades en distinguir las rocas del coral del fondo del mar; pero al final es un fallo leve, ya que lo que interesa es saber que hay algo y no tanto qué es lo que hay.

Figura 31 – Imagen de entrada, imagen generada, imagen objetivo.

Figura 32 - Imagen de entrada, imagen generada, imagen objetivo.

Además, se ha realizado una prueba reconstruyendo las imágenes de un vídeo, para comprobar si la red es capaz de reconstruir en tiempo real . Para ello, se ha transformado uno de los vídeos con los que se contaba inicialmente para que se asemeje al que recibiría el piloto del dron. Posteriormente, utilizando OpenCV se ha aplicado la reconstrucción a cada uno de los fotogramas del vídeo. Para esta reconstrucción se ha utilizado la herramienta de *Google Colaboratory*, que a través de la nube permite utilizar unas GPUs (Unidad de Procesamiento Gráfico) muy potentes. Concretamente, en esta prueba se contaba con una Nvidia Tesla K80 (24 GB de memoria GDDR5, 2.91 teraflops de rendimiento para operaciones de precisión doble y 8.73 teraflops para operaciones de precisión simple).

La transformación de las imágenes, al igual que ocurría con las imágenes de test, es muy sorprendente, ya que las reconstruye las imágenes prácticamente de la nada, generando un nivel de detalle y de visibilidad decente para esta resolución. El problema surge a nivel de *frame rate*, ya que, de media tarda en procesar cada fotograma 0.047 segundos lo que se transforma en un *frame rate* de 20 fotogramas por segundo cuando el video original tenía un *frame rate* de 30 fotogramas por segundo.

Rendimiento de la implementación reconsGAN sobre video	
Tiempo por frame [s]	0.0476
***Frame rate* máximo [fps]**	20.97

Tabla 1 - Rendimiento reconsGAN sobre video

En este punto de la implementación se ha resuelto el primero de los problemas que se planteó inicialmente; la imagen que recibía el piloto inicialmente ha sido correctamente reconstruida a través de la red generativa eliminando de esta forma el efecto scattering y la turbidez del agua. Ahora, se debe implementar la segunda de las redes generativas, que será la encargada de aumentar la resolución de estas imágenes de $256\ x\ 256$ a $1024\ x\ 1024$, para de esta forma no solo reconstruir las imágenes sino también aumentar su nivel de detalle y facilitar, de esta manera, la labor al piloto.

4.2.2. Aumento de resolución de la imagen (srGAN)

Como ya se comentó, las imágenes que recibía el piloto contaban con una resolución muy baja debido al poco ancho de banda con el que cuenta el cable umbilical que une al piloto con el dron submarino. Tras la reconstrucción de la imagen, a pesar de que la resolución se ha aumentado ligeramente, hasta $256\ x\ 256$, el problema no se ha solucionado del todo. Por este motivo, en la segunda parte de la implementación se va a trabajar en ampliar la resolución de la imagen para así conseguir aumentar el nivel de detalle y la calidad de esta.

En el ámbito del *deep learning* el aumento de la resolución de imágenes y vídeos se conoce como *super-resolution* (super resolución) y es una técnica que se está utilizando para la remasterización de películas antiguas [11] o de videojuegos [12].

Tradicionalmente, para el reescalado de imágenes existían numerosas técnicas como el reescalado bilineal, donde al aumentar la resolución lo que se hace es rellenar los pixeles nuevos mediante interpolaciones lineales entre los pixeles que sí se conocen, o el reescalado por proximidad, donde los pixeles nuevos se rellenan con el color del pixel más cercano. Este tipo de técnicas, si bien en algunos casos consiguen obtener un resultado aceptable, presentan una serie de problemas, como puede ser la pérdida de detalle y desenfoque con el reescalado bilineal o una imagen con patrones cuadriculados y con dientes de sierra en el caso del reescalado por proximidad (figura 33). Aunque es cierto que existen muchas más técnicas de reescalado, todas ellas utilizan la información local a nivel de pixeles y no llegan a entender el contenido de la imagen para realizar el reescalado.

Figura 33 – Reescalado por proximidad, reescalado bilineal, imagen original
Imagen obtenida de researchgate.net

En cambio, las técnicas de *deep learning* que se centran en el reescalado de imágenes tienen su base en la percepción y en la generación de la imagen, es decir, primero entender lo que hay en la imagen para posteriormente sintetizar o generar los pequeños detalles que la forman. Este tipo de técnicas no pueden, a partir de una imagen sin ningún nivel de detalle, aumentar su resolución correctamente, ya que, es en estos casos, la red tiende a sobreinterpretar la imagen e inventarse partes de esta que en realidad no existen. Estas técnicas alcanzan su máximo nivel cuando la imagen original aporta ciertos niveles de detalle. Es ahí cuando la red, a partir de las de imágenes con las que ha entrenado, es capaz de generar los pequeños detalles que en la imagen original no están debido a la baja resolución. Esto se puede ver reflejado en la figura 34: al principio no se aprecia diferencia, pero si uno se fija en la escala de las imágenes se puede comprobar cómo la imagen generada cuenta con prácticamente el cuádruple de resolución y, a pesar de esto, no han surgido los problemas que sí surgen con las técnicas tradicionales de reescalado de imagen; ya que la imagen sigue presentando una gran nitidez y ha ganado nivel de detalle.

Figura 34 – Ejemplo de super-resolution. A la izquierda imagen original, a la derecha la generada
Imagen obtenida de https://github.com/HasnainRaz/Fast-SRGAN

Para realizar esta implementación se ha aprovechado la red generativa adversaria que se realizó para la reconstrucción de las imágenes con unas ligeras modificaciones en el generador para poder generar imágenes de una mayor resolución.

Datos de entrenamiento

A pesar de que la estructura de la red es muy similar a la de la anterior implementación, los datos de entrenamiento son diferentes. En este caso, se deben tener como referencia imágenes de buena calidad de la resolución objetivo, que en este caso es

1024 x 1024. Se ha elegido esta resolución porque la red está diseñada para trabajar con imágenes cuadradas y porque es muy costoso computacionalmente entrenar una red para generar imágenes de resoluciones de alta calidad.

Entonces, partiendo de las 1078 imágenes que se eligieron para formar el conjunto de datos de la anterior red generativa, se han tomado las imágenes de $1080\ x\ 1920$ pixeles de resolución y se han reescalado mediante *Photoshop* a una resolución de 1024 x 1024 pixeles, de esta forma se tienen las imágenes base con las que realizará la comparación la red discriminadora.

Para obtener las imágenes de entrada de la red generadora se han tomado las imágenes base y se han reescalado a $256\ x\ 256$. Posteriormente se han reescalado nuevamente a 1024 x 1024 usando técnicas clásicas como la bilineal y la del pixel cercano. Esta operación se ha realizado varias veces, para que las imágenes pierdan nivel de detalle y calidad. El resultado que se ha obtenido es el que se puede ver en la figura 35, donde a la izquierda se puede ver la imagen de entrada a la red generadora, y a la derecha la imagen base con la que comparará la imagen generada el discriminador. Se puede apreciar el bajo nivel de detalle con el que cuenta la imagen de entrada y las diferencias entre ambas imágenes.

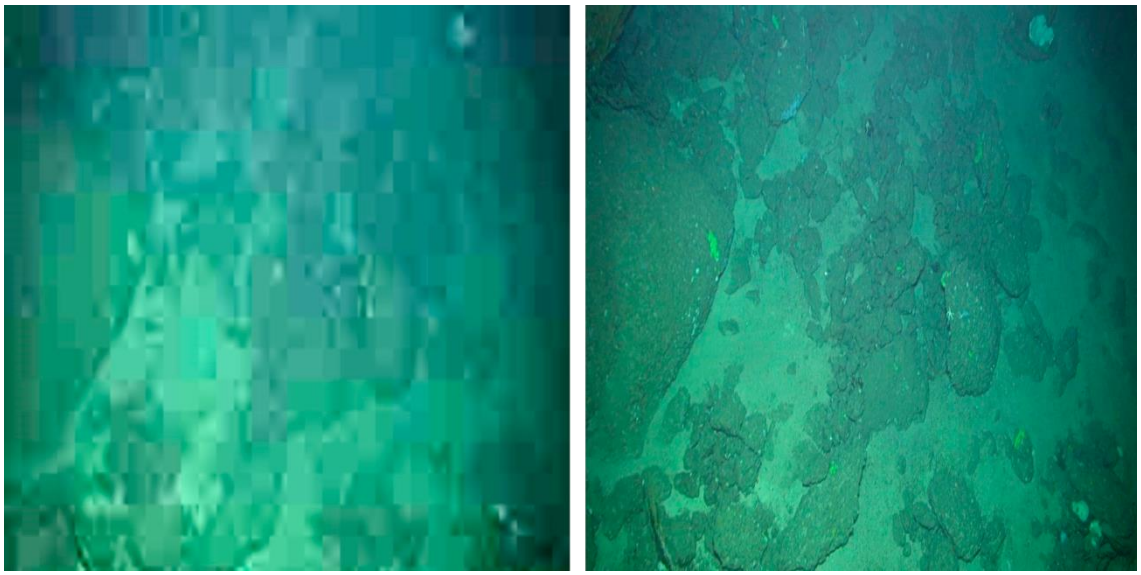

Figura 35 – Imagen de entrada (izquierda), imagen objetivo (derecha)

Implementación de la red

Como se ha mencionado en el punto anterior, la red que se va a utilizar es la misma que se utilizó para realizar la reconstrucción de las imágenes. Únicamente se han cambiado los datos de entrenamiento para entrenar la red en el objetivo deseado, y como en esta ocasión se trata de imágenes de mayor resolución, se deben modificar ligeramente el generador para que el *encoder* y el *decoder* puedan trabajar con imágenes de $1024\ x\ 1024$ pixeles de resolución. Y por el mismo motivo, para que el discriminador comprima más las imágenes y puedan encontrar un mayor número de patrones, debe ser modificado añadiéndole un par de capas más de convolución.

Por lo tanto, siguiendo la misma estructura *encoder – decoder* de la arquitectura U-NET, que mencionaba el *paper* de *Pix2Pix,* al *encoder* se le han añadido dos nuevas capas convolucionales de factor 2, y con 64 filtros, que reducirán la imagen en primera

instancia a $512\ x\ 512$ y posteriormente a $256\ x\ 256$. De esta forma, la imagen quedaría con las mismas dimensiones con las que funcionaba el *encoder* de la anterior red. Por lo que la estructura del *encoder* utilizando la nomenclatura del *paper* quedaría:

$$C64 - C64 - \ C64 - C128 - C256 - C512 - C512 - C512 - C512$$

Ecuación 11 – Estructura encoder srGAN

Por otro lado, en el *decoder* se ha realizado la misma operación. En la anterior red la imagen que salía del *decoder,* contando la capa de salida, tenía unas dimensiones de $256\ x\ 256$. Si, en esta ocasión, se busca que la imagen de salida tenga unas dimensiones de $1024\ x\ 1024$, se deberán crear dos nuevas capas que realicen deconvoluciones de factor 2 y de esta forma se aumentarán las dimensiones de la imagen generada. Por lo tanto, la estructura del *decoder* será la siguiente:

$$CD512 - CD512 - CD512 - C512 - C256 - C128 - C64 - C32 - C16$$

Ecuación 12 – Estructura decoder srGAN

Al discriminador, al igual que el generador, al tener que analizar imágenes de mayor resolución y para que sea capaz de comprimirlas al mismo punto que hacía antes se le deben añadir un par de capas de convolución que comprimirán la imagen de $1024\ x\ 1024$ a $512\ x\ 512$ y de $512\ x\ 512$ a $256\ x\ 256$. Por lo tanto, la estructura de su *encoder* queda de la siguiente manera:

$$C64 - C64 - C64 - C128 - C256 - C512$$

Ecuación 13 – Estructura encoder discriminador srGAN

El resto de la red (funciones de coste, optimizadores) es idéntico al utilizado en la red vista anteriormente.

Resultados

Para llevar a cabo el entrenamiento de esta nueva red, al igual que en la anterior, se contaba con 1078 imágenes en total, de las que 862 serán imágenes de entrenamiento que se van a ejecutar durante 300 épocas.

Durante el proceso de entrenamiento, al igual que con la anterior red, se mostrará el resultado que se obtiene al aplicar la red generadora al conjunto de datos de test. En las figuras 36, 37 y 38 se muestra como avanza el entrenamiento y como la red va aprendiendo a generar mejores resultados.

Figura 36 – Imagen de entrada, imagen generada, imagen objetivo. Época 0

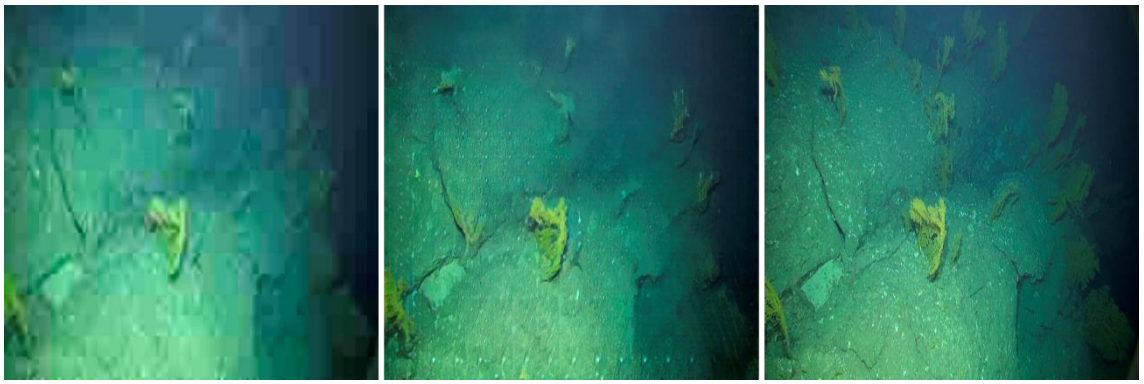

Figura 37 – Imagen de entrada, imagen generada, imagen objetivo. Época 100

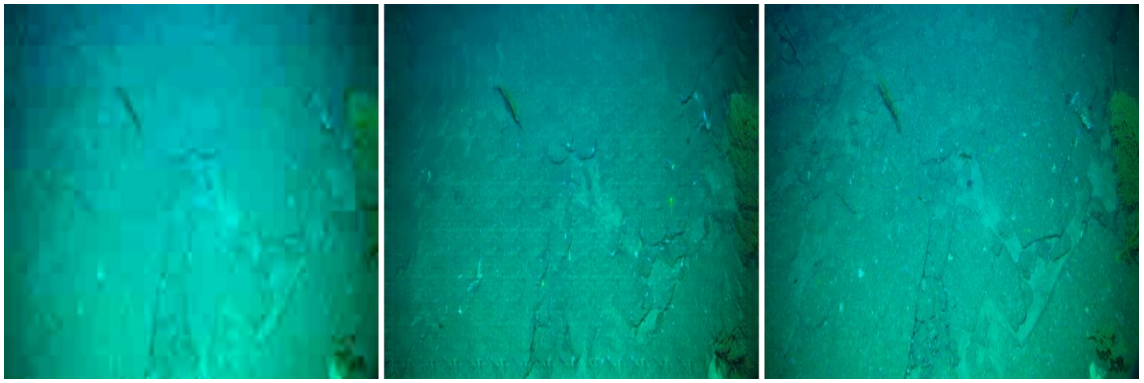

Figura 38 – Imagen de entrada, imagen generada, imagen objetivo. Época 200

Tras concluir el entrenamiento, se ha conseguido que las imágenes de salida de la red presenten un buen nivel de detalle con la resolución de 1024 x 1024 pixeles, como se puede ver en las siguientes figuras (39 y 40).

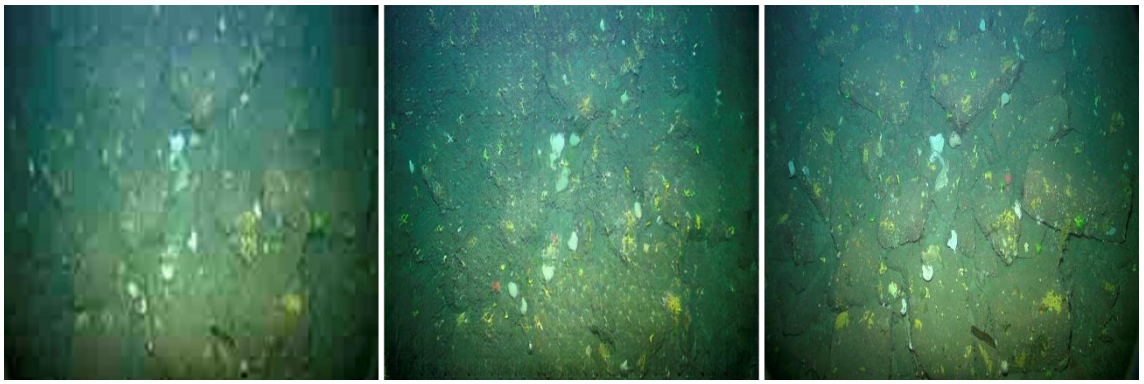

Figura 39 - Imagen de entrada, imagen generada, imagen objetivo.

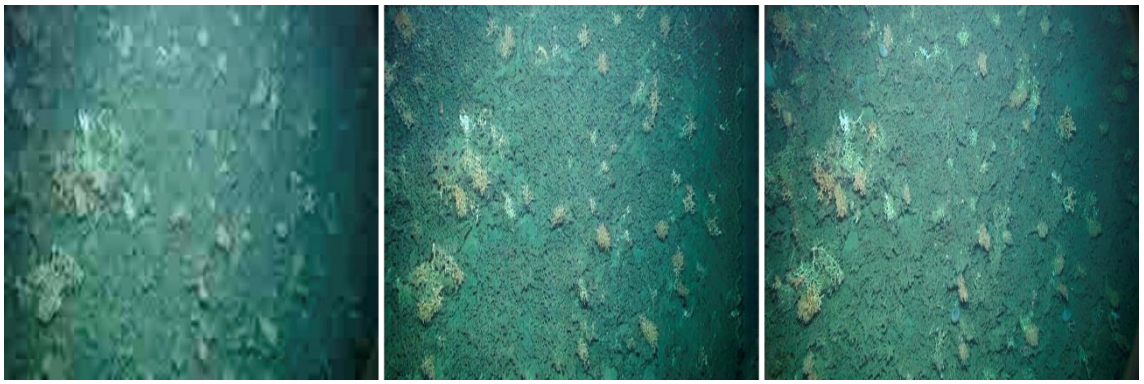

Figura 40 - Imagen de entrada, imagen generada, imagen objetivo.

Por otro lado, al igual que se hizo con la red encargada de realizar la reconstrucción, se ha realizado la prueba de aplicar la red a cada uno de los fotogramas de un vídeo, para comprobar cuánto tiempo tarda en aumentar la resolución de cada uno de estos y qué *frame rate* se puede obtener.

Rendimiento de la implementación srGAN sobre video	
Tiempo por frame [s]	0.0842
***Frame rate* máximo [fps]**	11.86

Tabla 2 - Rendimiento srGAN sobre video

Esto se ha comprobado con un vídeo que funciona a 30 fotogramas por segundo, y la prueba se ha realizado nuevamente en el entorno de *Google Colaboratory.* En esta sesión se contaba con una GPU Nvidia Tesla T4, que cuenta con un rendimiento de 8.1 TFLOs en operaciones de doble flotante y de 65 TFLOs con flotante simple, 16 GB de memoria GDDR6 y un ancho de banda de 320 GB/s.

A la vista de los resultados, a pesar de la mejora en la unidad de procesamiento gráfico, el tiempo para aumenta la resolución de cada fotograma a través de la red realizada es prácticamente el doble que en la reconstrucción de este. Esto se debe a que esta red debe trabajar con imágenes de una mayor resolución y, por lo tanto, la red requiere de un mayor cómputo para obtener la salida.

4.2.3. Resultados

A lo largo de este capítulo se ha explicado en primer lugar, muy detalladamente, la red generativa adversaria que se ha realizado, basándose en el modelo *Pix2Pix,* para la reconstrucción de la imagen. Posteriormente, aprovechando la red ya realizada para la reconstrucción se le han introducido unas pequeñas variaciones para utilizar esta red para aumentar la resolución de las imágenes.

Se han mostrado los resultados obtenidos por cada una de estas redes de forma individual. Pero, lo que realmente se quiere comprobar con esta implementación son sus resultados de forma conjunta. A primera vista, y viendo los resultados individuales, no debe haber ningún problema en lo relativo a la reconstrucción y *upscaling* de las imágenes. La primera red es capaz de reconstruir la imagen, eliminado todos los problemas que presentaban las imágenes de entrada como la turbidez del agua y el efecto *scattering* que producía está en los canales de color de la imagen. Tras la

reconstrucción, la segunda red puede tomar a su entrada una imagen con mucho mas nivel de detalle y, a partir de esta, aumentar la escala sin incurrir en la mayoría de los problemas de las técnicas tradicionales de *upscaling*.

A continuación, se muestra la imagen de entrada, es decir, la imagen con la resolución original ($300\ x\ 170$) y los problemas de turbidez del agua, bajo nivel de detalle, lo que produce una escasa visibilidad. Esto sería lo que vería el piloto sin aplicar la tecnología implementada:

Figura 41 – Imagen original piloto

Figura 42 - Imagen reconstruida completamente

Y, partiendo de esta imagen, se muestra la imagen ya reconstruida (figura 42) y con la resolución aumentada a $1024\ x\ 1024$. A pesar de que la reconstrucción falla en los bordes de la imagen, ya que se aprecian los cambios que ha producido la red, y que la calidad de esta es mejorable, el nivel de visibilidad y detalle que se puede apreciar en la imagen en comparación con la imagen inicial (figura 41) es muy significativo, y se puede dar la reconstrucción de la imagen como exitosa.

Una vez se ha comprobado el resultado de ambas redes en conjunto, se ha probado, al igual que se hizo por separado, a comprobar su funcionamiento al reconstruir y aumentar la resolución de cada uno de los fotogramas de un vídeo para comprobar su rendimiento. Observando los resultados que se obtuvieron al realizar esta misma prueba, pero con cada una de las redes por separado, se puede intuir que la red encargada de realizar el aumento de resolución de las imágenes al trabajar con imágenes de mayor tamaño va a crear un cuello de botella, limitando el número de fotogramas que pueda alcanzar el vídeo resultante.

Esta prueba, al igual que las anteriores, se ha realizado utilizando *Google Colaboratory* y concretamente, en la sesión en la que se realizó la medida se contaba con una Nvidia Tesla T4, que es la misma con la que se contaba en la prueba de la *srGAN*. En las siguientes ecuaciones (Ecuación 16) se puede observar cómo las dos redes en conjunto tardan una media 0.095 segundos en modificar cada fotograma lo que repercute en un *frame rate* de 10.45 fotogramas por segundo.

Rendimiento de la implementación en dos pasos sobre video	
Tiempo por frame [s]	0.095
***Frame rate* máximo [fps]**	10.45

Tabla 3 - Rendimiento implementación en dos pasos sobre video

4.3. Implementación en un paso

En esta segunda implementación, en lugar de utilizar dos redes generativas adversarias que realicen por separado la reconstrucción de las imágenes y el aumento de la resolución de estas, se utilizará una única red que realice las dos funciones. Esta red (*reconsSRGAN)* contará con una estructura similar a la vista anteriormente donde el generador tomará como entrada las imágenes que recibiría el piloto, pero en este caso no solo tendrá que reconstruir las imágenes, sino que además tendrá que aumentar la resolución de estas.

4.3.1. Implementación de la red

Preprocesado de los datos

El conjunto de datos de entrenamiento con el que se cuenta es el mismo con el que se disponía para la implementación en dos pasos. Y, al igual que se ha realizado en la anterior implementación, a este conjunto de datos se le realiza un pequeño preprocesado previo al entrenamiento, que en este caso contará con una pequeña diferencia. Donde en la implementación en dos pasos las imágenes se reescalaban a $256\ x\ 256$, en este caso este reescalado será a $1024\ x\ 1024$. De esta forma, se tomará la imagen inicial que recibiría el piloto, de una resolución de 300 x 170, y será reescalada

a 1024 x 1024, donde el resultado será el que se muestra en la figura 43. Por lo demás, al conjunto de entrenamiento se le realizará el mismo procesado que se realizaba para la anterior implementación, es decir, las imágenes serán normalizadas y se aumentará el conjunto de datos mediante volteos (*flips* horizontales) y sesgados aleatorios de las imágenes.

Figura 43 – Imagen inicial reescalada a 1024 x 1024

Red generadora

El generador, toma la imagen inicial que previamente ha sido preprocesada para, en primer lugar, comprimirla a través de *encoder* y extraer sus características más representativas y partiendo de esas características generar a través de *decoder* una nueva imagen reconstruida y con una resolución de 1024 x 1024 pixeles. Para realizar esto sigue la misma estructura que se explica en el *paper* de *Pix2Pix*, en donde el *encoder* y el *decoder* están formados por diferentes bloques. Donde cada bloque está formado por una capa primera capa de convolución, en la que va variando el número de filtros, una segunda capa de *batch normalization* y finalmente una capa de activación, en el caso del *decoder* existe una capa previa de *dropout*, donde en el *encoder* se puede encontrar una función Leaky ReLU y en el *decoder* una ReLU. Si se representa la

estructura que sigue tanto el *encoder* como el *decoder* en una ecuación utilizando la nomenclatura que utiliza el *paper*, esta sería:

$$C64 - C64 - \ C64 - C128 - C256 - C512 - C512 - C512 - C512$$

Ecuación 14 – Encoder de la implementación en dos pasos

$$CD512 - CD512 - CD512 - C512 - C256 - C128 - C64 - C32 - C16$$

Ecuación 15 - Decoder de la implementación en dos pasos

Red discriminadora

La red discriminadora tomará como valores de entrada tanto la imagen generada por el generador como la imagen objetivo que es la que se quiere conseguir y con la que se tiene que comparar. Igual que se realizaba en la implementación en dos pasos, el discriminador debe concatenar estas dos imágenes e ir comprimiéndolas para extraer la información más relevante de estas y poder generar el mapa de parches en el que se mostrarán las diferencias entre las imágenes.

Al igual que la red generadora, el discriminador de esta implementación tiene la misma estructura *patchGAN* que la vista en la anterior, donde se utiliza una red convolucional para ir comprimiendo la imagen. Esta red convolucional está formada por una serie de bloques idénticos a los utilizados en el *encoder* del generador. La única diferencia está en el número de capas, ya que el discriminador cuenta con menos capas porque no busca comprimir completamente la imagen sino mostrar las diferencias en el mapa de parches que obtiene como salida.

4.3.2. Resultados

Esta segunda implementación se ha entrenado durante 350 épocas. Nuevamente se contaba con 862 imágenes de entrenamiento y 216 imágenes de test que probaran la red al final de cada una de las épocas. En las figuras 44, 45, 46 y 47 se muestra el resultado de aplicar la red a las imágenes de test durante diferentes fases del entrenamiento.

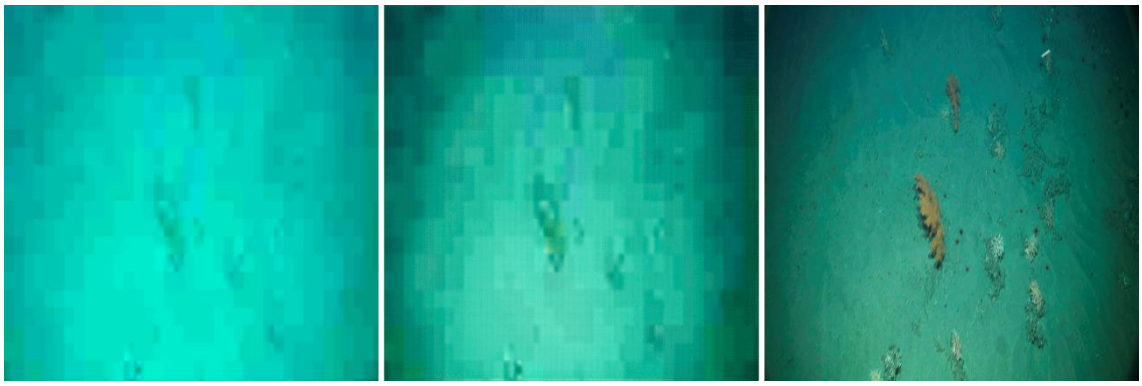

Figura 44 - imagen de entrada, imagen generada e imagen objetivo. Época 0

Figura 45 - imagen de entrada, imagen generada e imagen objetivo. Época 100

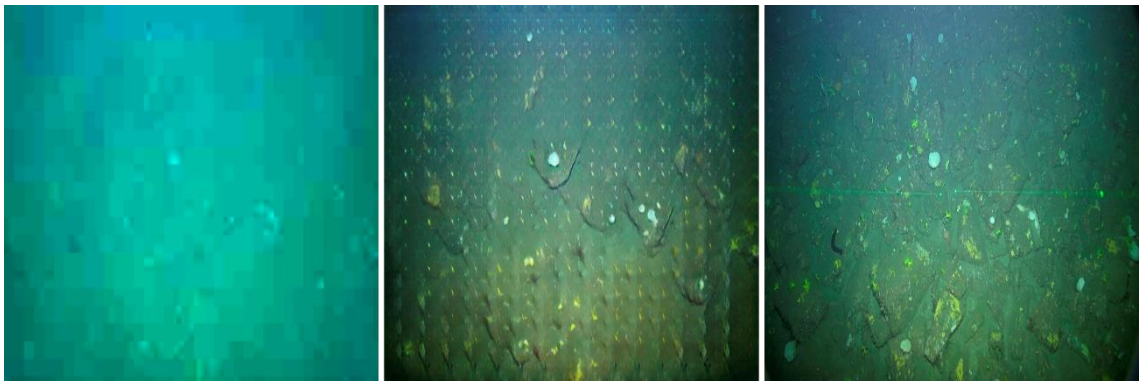

Figura 46 - imagen de entrada, imagen generada e imagen objetivo. Época 200

Figura 47 - imagen de entrada, imagen generada e imagen objetivo. Época 300

Finalmente, tras concluir el entrenamiento se puede observar (figura 48) que la reconstrucción es aceptable sobre todo considerando el punto de partida de la imagen inicial y que además de reconstruir la imagen se ha aumentado la resolución de esta. Si bien es cierto al igual que ocurría con la reconstrucción en dos pasos la red tiende a confundir el coral del fondo del mar con rocas y piedras que se puedan encontrar ahí. Así todo, este error es un error leve porque como ya se comentó en la implementación en dos pasos lo que se busca es saber que ahí hay algo y no tanto que es lo que hay. Por otro lado, si se observan los bordes de la imagen de la figura 49, se puede observar que se generan artefactos al igual que ocurría en la implementación en dos pasos. A pesar de estos fallos, la red consigue reconstruir la imagen con éxito, pues partiendo de una imagen sin nivel de detalle y sin visibilidad, la red ha aprendido no solo a reconstruir la imagen, sino que también ha triplicado su resolución.

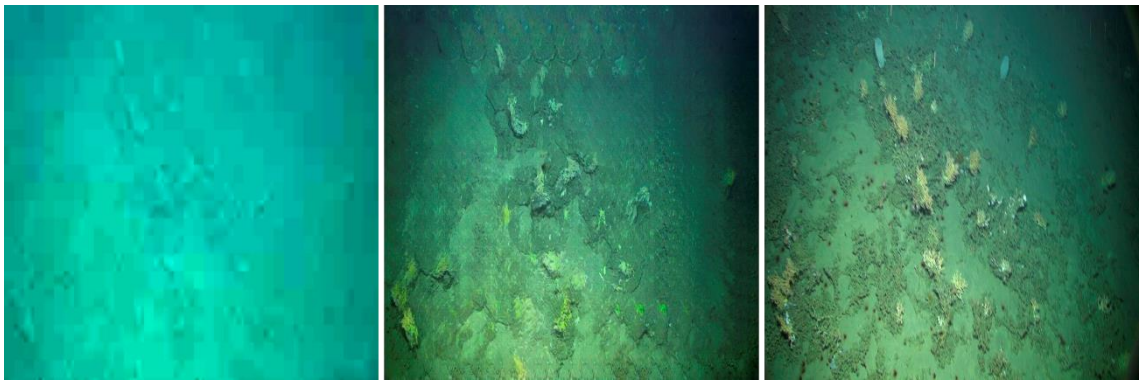

Figura 48 - imagen de entrada, imagen generada e imagen objetivo. Época 350

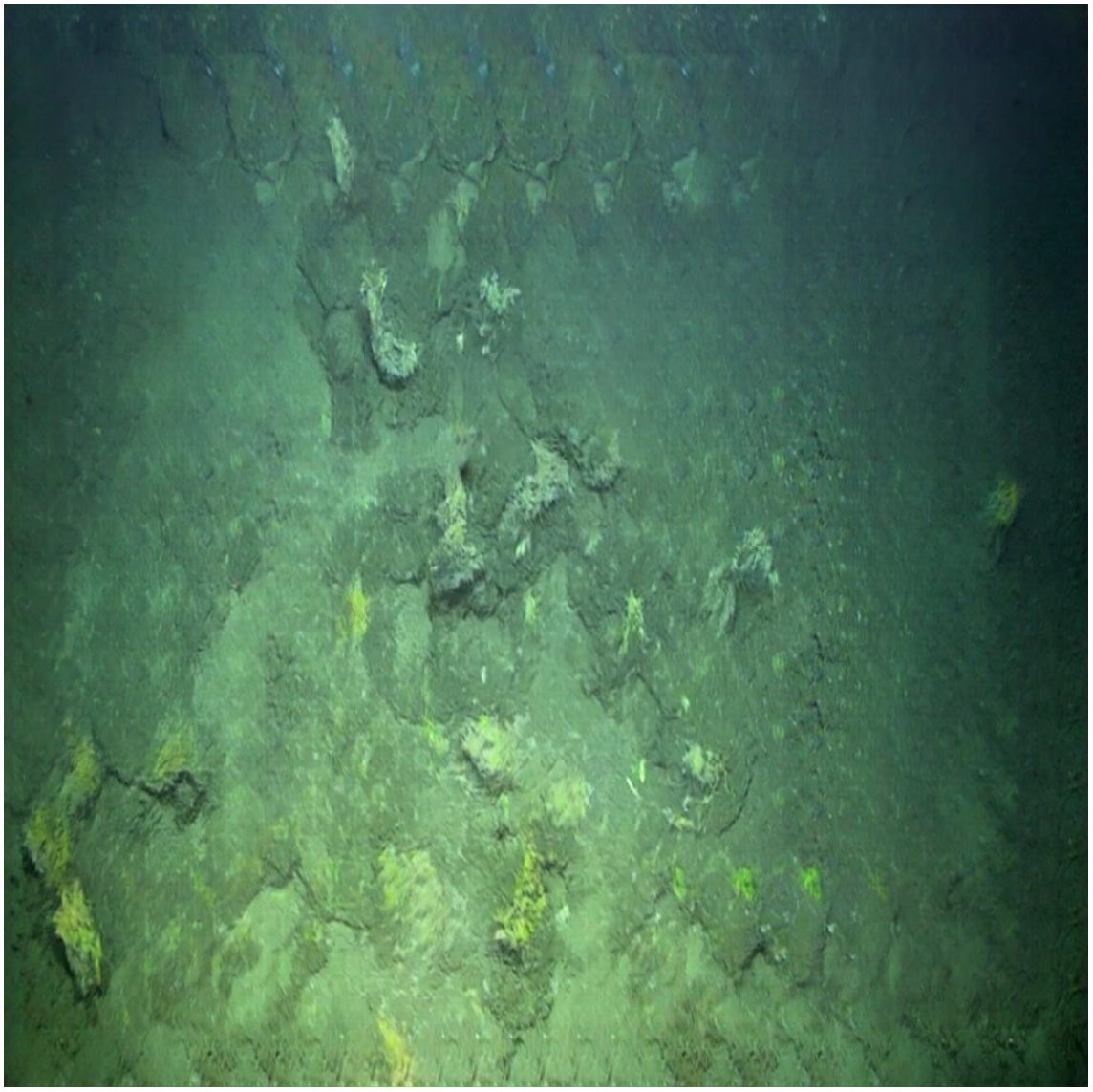

Figura 49 - Imagen reconstruida

Una vez se han comprobado los resultados que consigue la red en la reconstrucción y el *upscaling* de la imagen se debe comprobar cómo funciona esta red trabajando con un video en tiempo real. Para ello se ha hecho uso de la GPU Nvidia Tesla T4, que proporciona *Google* a través de *Google Colaboratory,* donde se ha probado el funcionamiento de esta segunda implementación. La red consigue procesar cada

fotograma en un tiempo medio de 0.082 segundos lo que se convierte en 12.093 fotogramas por segundo.

Rendimiento de la implementación en un paso sobre video	
Tiempo por frame [s]	0.082
***Frame rate* máximo [fps]**	12.093

Tabla 4 - Rendimiento de la implementación en un paso sobre video

4.4. Comparativa de las implementaciones

Una vez se han explicado las dos implementaciones que se han realizado y se han mostrado sus resultados. Se debe realizar una comparación entre ambas para ver los puntos fuertes de cada una y así comprobar que implementación consigue mejores resultados.

En primer lugar, se realizará una comparación de la calidad de reconstrucción de la imagen, es decir, como de bien se ha reconstruido y se ha aumentado la resolución de esta. Para ello ha realizado la reconstrucción de la misma imagen (figura 50) utilizando ambas implementaciones.

Figura 50 – imagen de entrada para la comparación

La figura 51 es el resultado de aplicar la reconstrucción, a la imagen de la figura 50, utilizando ambas redes. La imagen de la izquierda de la figura 51 se corresponde con la reconstrucción de la imagen utilizando la implementación en dos pasos, la imagen de la derecha se corresponde con la implementación en un paso y la imagen central se corresponde con la imagen original. En esta figura se puede observar que ambas implementaciones reconstruyen la imagen de una forma similar. El fondo de la imagen, que en la imagen de entrada (figura 50) no tienen ningún nivel de detalle y no se distingue nada, lo reconstruyen como "agua", sin distinguir las algas que forman parte de la roca. La reconstrucción sí que distingue la roca del frente de la imagen y el alga que esta sobre ella. Y, teniendo en cuenta la imagen original, en la que como se ha mencionado no se distingue nada, se puede considerar que ambas reconstrucciones son exitosas.

Figura 51 - Implementación 1 vs Implementación 2

Si se compara la reconstrucción de la implementación en dos pasos con la imagen original, se pueden mostrar las diferencias entre ambas imágenes. Esto se muestra en la figura 52, donde la imagen de la izquierda se corresponde con la reconstrucción de la imagen. La imagen central se corresponde con la imagen original y la imagen de la derecha se corresponde con la diferencia entre ambas imágenes.

Figura 52 – Reconstrucción en dos pasos, imagen original y diferencia entre ellas

Si se realiza la misma operación con la imagen reconstruida mediante la implementación en un paso, el resultado que se obtiene (figura 53) es similar al visto en la figura anterior, ya que los errores en la reconstrucción son comunes para ambas implementaciones.

Figura 53 - Reconstrucción en un paso, imagen original y diferencia entre ellas

Como conclusión, a la vista de los resultados observados, se puede decir que ambas implementaciones reconstruyen las imágenes de una forma muy parecida. Ambas implementaciones consiguen una reconstrucción más fiel a la original en las zonas centrales de la imagen, ya que, en la imagen de entrada, son las zonas que cuentan con

mayor visibilidad y nivel de detalle. Mientras que en las zonas posteriores o de los laterales la reconstrucción no consigue ser tan efectiva.

Una vez se ha comprobado la calidad de la reconstrucción de la imagen, concluyendo que ambas reconstruyen las imágenes de igual forma. Se puede comprobar que tal funciona cada una de las implementaciones reconstruyendo imágenes en tiempo real y así determinar su rendimiento. Para ello, se debe comparar el tiempo que necesita cada una de las implementaciones para reconstruir un fotograma en igualdad de condiciones, es decir, utilizando la misma GPU.

En estas pruebas realizadas, se ha utilizado la GPU Nvidia Tesla T4 que proporciona el entorno de *Google Colaboratory*, y tras la reconstrucción del mismo video de entrada se han obtenido los resultados que se pueden observar en la siguiente tabla

	Implementación dos pasos	**Implementación un paso**
Tiempo medio por *frame* [s]	0.095	0.082
***FPS* alcanzados [fps]**	10.45	12.093

Tabla 5 - Comparación de rendimiento entre implementaciones

Observando los resultados se puede ver que la implementación que realiza en un solo paso la reconstrucción y el aumento de la resolución es ligeramente más rápida al realizar la reconstrucción de la imagen, y por lo tanto puede reproducir un video en tiempo real con una mayor tasa de fotogramas por segundo.

A la vista de los resultados obtenidos, el rendimiento de la implementación en un paso es ligeramente superior al de la implementación en dos pasos. Porque tras reconstruir la imagen se obtiene un resultado muy similar, pero al aplicar la reconstrucción en tiempo real la implementación en dos pasos reconstruye los fotogramas $13\,ms$ más rápido.

5. Implementación Adicional: Estimación automática de distancia

Inicialmente, se planteó como principal objetivo la reconstrucción de imágenes submarinas para facilitar el pilotaje de un dron submarino. Una vez se cumplió con este objetivo, se buscaron implementaciones adicionales que pudiesen complementarlo y de esta manera ayudar aún más al piloto con su labor. Por este motivo se decidió complementar el objetivo principal con una nueva implementación que permitiese estimar automáticamente la distancia entre los dos puntos laser con los que cuenta el dron, para así proporcionar al piloto una referencia mediante la cual obtener datos importantes como la profundidad a la que se encuentre el dron.

La estimación automática de distancia es un problema clásico dentro del campo de la visión artificial. Para resolver este tipo de problemas se debe, en primera instancia, detectar el objeto del cual quieres estimar su distancia para, posteriormente, calcular esta distancia a partir de las coordenadas en las que se encuentra dicho objeto.

Como se vio en el capítulo II, la detección de objetos es una de las aplicaciones más comunes para las redes convolucionales debido a su capacidad para extraer las características más relevantes de las imágenes con las que trabajan. Es por esto por lo que se decidió utilizar este tipo de redes para conseguir realizar con existo la detección del láser para posteriormente poder estimar la distancia entre ellos.

Entrenar de cero una red convolucional para que distinga objetos requiere de un tiempo de entrenamiento muy largo debido a que las redes convolucionales tienden a ser muy profundas y entonces son muchos los parámetros a ajustar. Además, se debe contar con un conjunto de datos muy amplio y diverso para no condicionar a la red y caer en el *overfitting*. Por este motivo, normalmente se hace uso de la transferencia de aprendizaje, que permite utilizar redes que han sido previamente entrenadas en la detección de objetos, y aprovechar así la capacidad que adquirieron durante su entrenamiento para detectar formas y patrones. Y, de esta manera, se puede conseguir entrenar una red para la detección del objeto que interesa sin la necesidad de emplear una gran cantidad de tiempo en el entrenamiento, ni contar con un gran conjunto de datos.

Para esta implementación se ha utilizado You Only Look Once (YOLO) [13] que es un sistema de código abierto que utiliza redes convolucionales para detectar objetos, siendo la detection de objetos en tiempo real su principal virtud ya que solo requiere ver cada imagen una única vez, tal y como indica su nombre. Como todas las redes convolucionales, YOLO forma parte del paradigma de aprendizaje supervisado y previo al entrenamiento se deben etiquetar las imágenes indicando mediante un recuadro (*bounding box)* donde se encuentra el objeto en la imagen, como se puede ver en la figura 54.

Para llevar a cabo la detección de objetos, YOLO en primer lugar divide la imagen en una cuadricula de S filas y S columnas. Para posteriormente tratar de predecir el número de *bounding boxes* que hay en cada una de las celdas, y tratar de calcular la probabilidad de cada una de ellas. La mayoría de las *bounding box* que se encuentran cuentan con un

nivel de probabilidad muy bajo, y por esto se aplica un umbral para descartar esas celdas. Finalmente, al resto de cajas restantes se les aplica un paso de *non-max suppression* en el que se elimina las cajas duplicadas, quedando únicamente aquellas que tienen mayor probabilidad. Este proceso se puede ver representado en la figura 55, donde al comienzo del proceso se ve como la imagen se divide en celdas, donde luego se buscan las *bounding box* y se calcula su probabilidad para finalmente eliminar las que se encuentren por debajo de un umbral y así detectar finalmente el objeto.

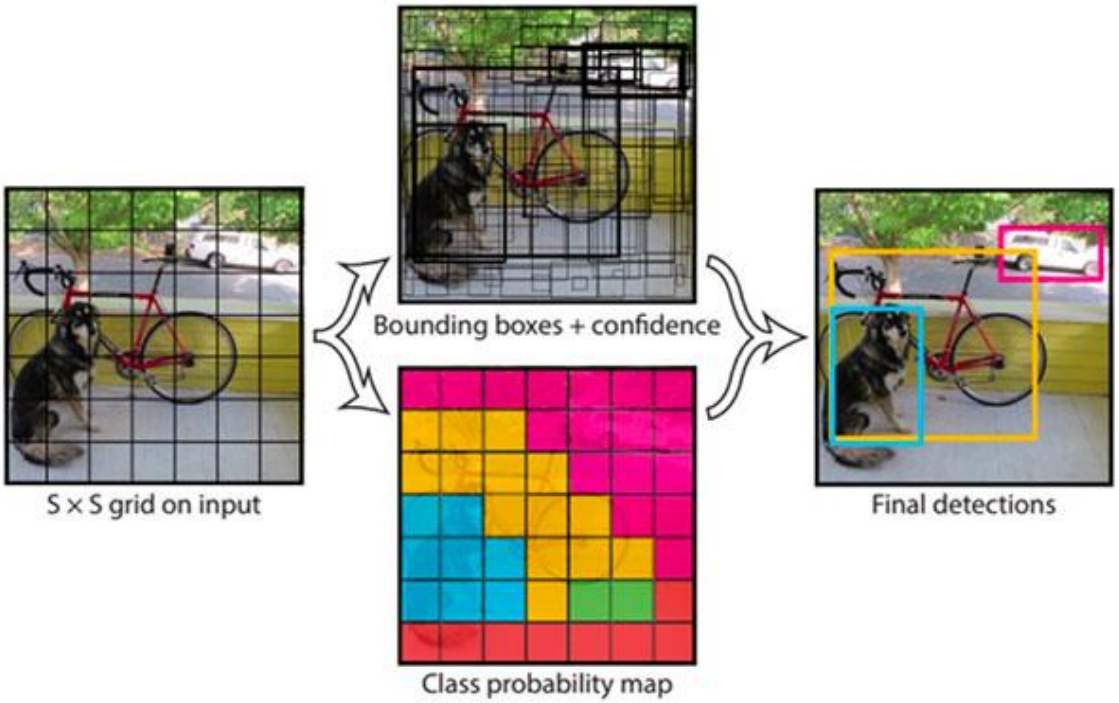

Figura 54 - Funcionamiento YOLO
(Imagen obtenida de medium.com)

Actualmente YOLO cuenta con varias versiones que han ido mejorando su funcionamiento e incrementando su rendimiento. Para esta implementación se ha utilizado la quinta versión: YOLOv5 [14], que fue publicada el 27 de mayo del 2020. Esta es la última versión del sistema YOLO y se caracteriza por su velocidad tanto para inferir las imágenes como durante el entrenamiento.

5.2. Conjunto de datos y entrenamiento

Como se ha comentado anteriormente, YOLO es una red convolucional y como tal necesita un conjunto de datos correctamente etiquetado debido a que requiere un aprendizaje supervisado. El etiquetado para la detección de objetos consiste en indicar en que coordenadas de la imagen se encuentra el objeto a detectar mediante la creación de una *bounding box.* Para crear las *bounding boxes* sobre las imágenes se ha utilizado un *software* libre llamado LabelImg que permite recuadrar sobre la imagen el objeto e indicar a que clase pertenece, en este caso se creado la clase “laser”, y automáticamente el *software* se encarga de crear el fichero correspondiente donde se indican las coordenadas de este objeto y la clase a la que pertenece.

Para el entrenamiento de esta red, como se hace uso de la trasferencia de aprendizaje, se han utilizado únicamente 150 imágenes, donde en cada una de ellas se ha indicado la posición de los láseres. En una primera prueba se probó a etiquetar los láseres por

separado pero el *bounding box* resultante era demasiado pequeño y la probabilidad de detección, por lo tanto, muy baja. Por este motivo, se realizó un segundo intento creando un único *bounding box* para ambos láseres.

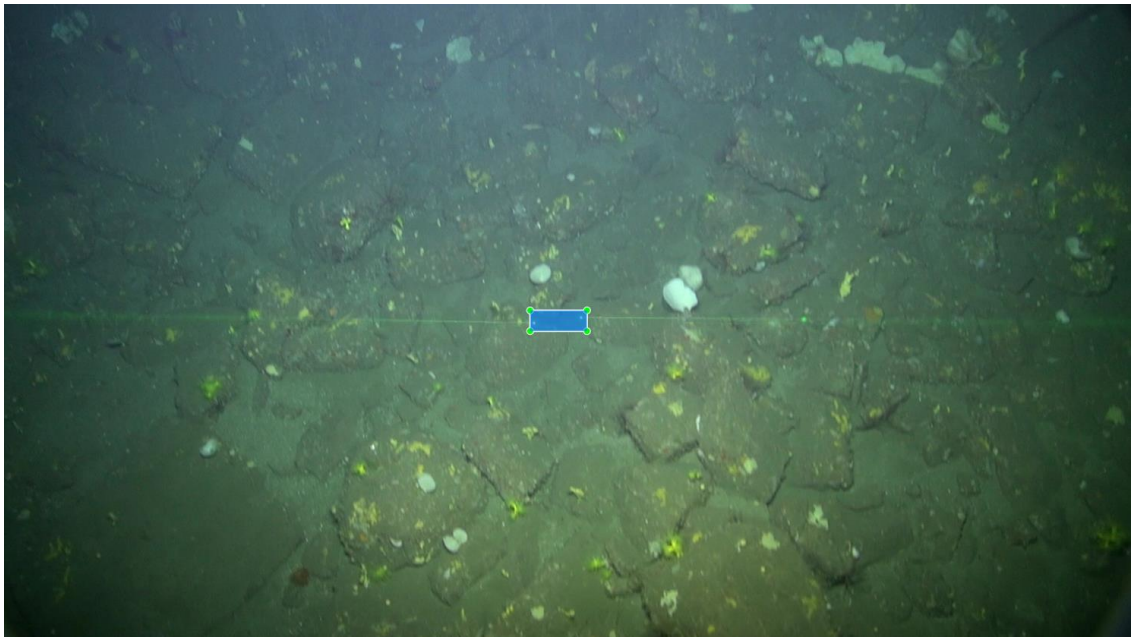

Figura 55 - Imagen del conjunto de entrenamiento

Una vez se han etiquetado todas las imágenes, se puede comenzar el entrenamiento de la red. Para ello, ya que se va a hacer uso de la transferencia de aprendizaje, se toman los pesos de la red preentrenada. Y, de esta forma, no se necesita que el entramiento requiera un gran número de épocas. Concretamente esta red se ha entrenado durante 50 épocas.

5.3. Resultados

En este apartado se evaluarán los resultados obtenidos tras la conclusión del entrenamiento. Para ello se ha comprobado la capacidad de detección de modelo entrenado en el conjunto de datos de validación, es decir, imágenes que el modelo nunca ha visto durante su entrenamiento. Para estas imágenes el modelo presenta una confianza que varía mucho en función del fondo marino sobre el que se proyecten los láseres. Si, por ejemplo, se trata de un fondo de arena y sin grandes contrastes, la confianza que presenta el modelo es entorno al 88% como se puede ver en la figura de la izquierda de la figura 56. En cambio, si el fondo es arena, pero con mucho contraste por diferentes partículas, la red puede creer que estas partículas también son láseres y fallar en su detección, esto se puede ver representado en la figura derecha de la figura 56, donde el modelo detecta múltiples láseres.

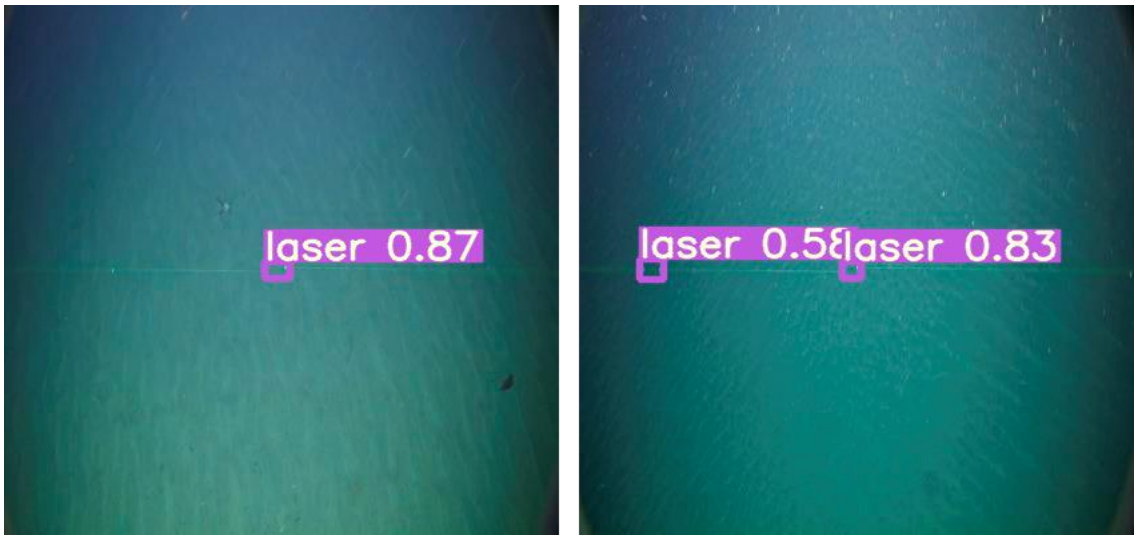

Figura 56 - Resultado 1: Fondo arena

Por otro lado, si el fondo es más rocoso, donde el láser destaca menos, la confianza del modelo baja. En este tipo de fondos, el modelo o bien detecta el láser, pero con una confianza entre el 40% y el 60%, o bien comete el error de detectar múltiples láseres.

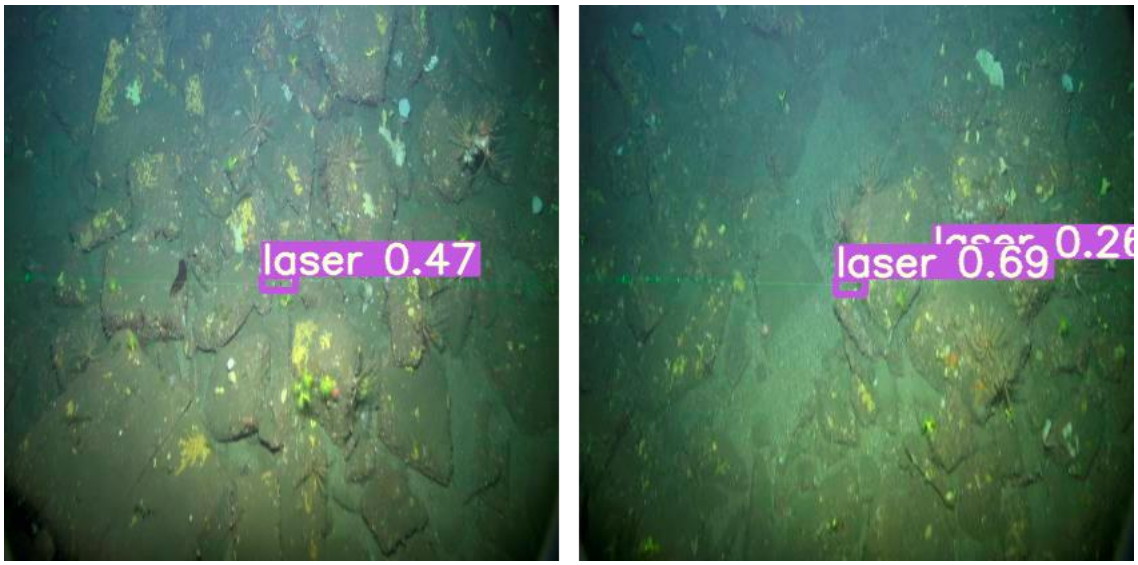

Figura 57 - Resultado 2: Fondo roca

No hay que olvidar, que el objetivo de esta implementación no era detectar los láseres sino la distancia entre ellos. La red actualmente detecta el objeto y calcula las coordenadas donde este se encuentra. Para obtener la distancia, simplemente se deben tomar las coordenadas x del recuadro que engloban el par de láseres y realizar una resta entre ella de tal forma que se obtendrá la distancia. Esta distancia siempre será orientativa pues si no se tiene una referencia no se puede hacer la conversión al sistema métrico internacional.

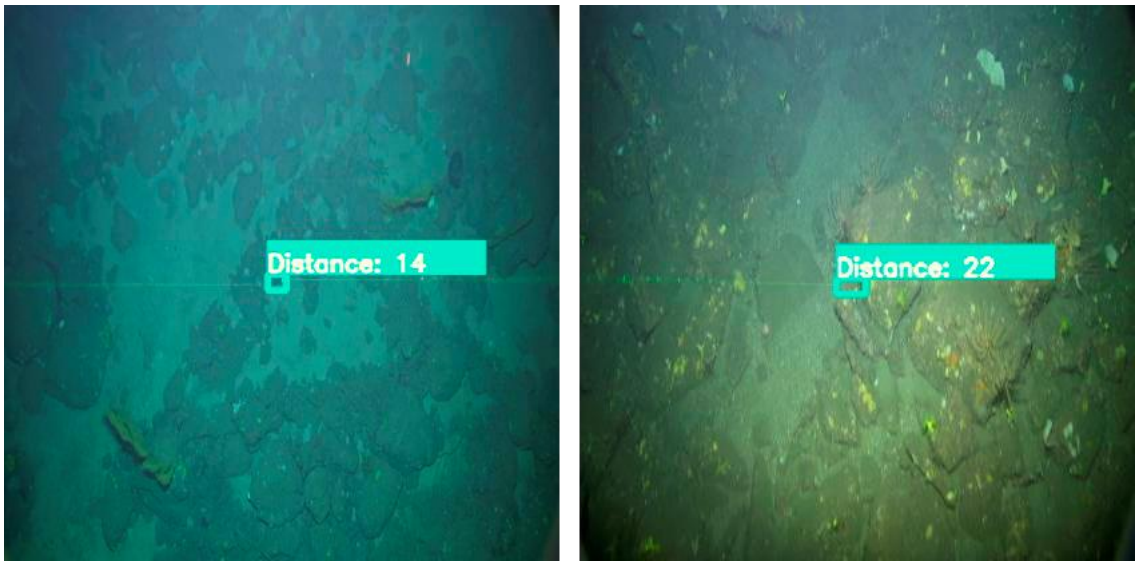

Figura 58 - Resultado 3: distancia

Finalmente, se debía comprobar su funcionamiento en tiempo real pues es donde va a ser utilizado realmente. YOLO es un sistema que está preparado para correr sobre video ya que solo necesita ver la imagen una vez para realizar detección del objeto. Cada fotograma se procesa en 0.021 segundos, lo que permite un *frame rate* máximo de 47.61 fps utilizando Nvidia Tesla K80, que cuenta con 24GB de memoria GDDR5.

Rendimiento YOLOv5 sobre video	
Tiempo por frame [s]	0.021
***Frame rate* máximo [fps]**	47.61

Tabla 6 - Rendimiento YOLOv5 sobre video

Como se puede observar en la tabla 6, el *frame rate* al que funciona el video no es un problema. No obstante, si existe un problema a la hora de detectar el laser en cada uno de los fotogramas. Como se ha visto en los resultados mostrados (figura 56 y 57), la confianza de la red varía en función del terreno sobre el que se proyecte el láser. Por lo tanto, si el umbral de confianza que se marca es alto la red no determinará la posición del láser en todos los fotogramas. Para solucionar esto, una solución que se propone es ir almacenando la distancia que va obteniendo la red. De esta forma, si la distancia disminuye o aumente se puede calcular el paso con el que lo hace. Y como para el dron es físicamente imposible realizar cambios de nivel bruscos, teniendo el paso con el que aumenta o disminuye la distancia se puede estimar que distancia existe entre los dos puntos laser en los instantes en que la red no es capaz de determinarla.

6. Conclusión y líneas futuras

El objetivo marcado en un inicio era solventar un problema con el que conviven los pilotos de drones submarinos, y que dificulta gravemente su trabajo: las imágenes que reciben del dron durante su pilotaje cuentan con una muy baja visibilidad debido a algunos problemas, como son el escaso ancho de banda del cable umbilical –lo que limita la resolución de estas imágenes–, la turbidez del fondo marino, (que afecta a la visibilidad de las imágenes que recibe el piloto, dificultando el pilotaje del dron, ya que el piloto no logra ubicarse), el efecto scattering del agua, que perturba los canales de color de las imágenes haciendo que las imágenes pierdan nitidez, etc. Estos problemas hacen que la imagen que reciba el piloto sea prácticamente inútil y que tenga que pilotar el dron a ciegas. Viendo estos problemas, se ha buscado aplicar técnicas modernas de aprendizaje automático para la reconstrucción de estas imágenes.

Para ello, a lo largo del tercer capítulo se ha realizado una pequeña introducción a las redes neuronales artificiales, donde se ha explicado detalladamente cómo se forman estas redes y su funcionamiento. Una vez se ha explicado el funcionamiento de las redes neuronales se han explicado las redes convolucionales, un tipo de red muy utilizada en el campo de la visión artificial y el procesamiento de imágenes, y especialmente, las redes generativas adversarias, en las cuales se basan las implementaciones realizada, que se han explicado en el cuarto capítulo.

Estas implementaciones utilizan redes generativas, las cuales se han basado en el modelo *Pix2Pix*, un modelo presentado en 2017 en el que se utilizan redes generativas condicionales para generar, reconstruir o transformar imágenes. La primera de las implementaciones utiliza dos redes enfocadas a realizar dos tareas distintas pero complementarias. Mientras la primera se encarga de reconstruir la imagen para que esta recupere visibilidad y aumente su nivel de detalle, la segunda se encarga de aumentar la resolución de estas imágenes para, de esta forma, obtener como resultado imágenes completamente reconstruidas como las que se han visto en el apartado en que se comentan los resultados. Por otro lado, la segunda implementación utiliza una única red que se encarga tanto de reconstruir las imágenes como de aumentar su resolución. Finalmente se han comparado ambas implementaciones para comprobar cuál de ellas obtiene un mejor rendimiento. Tras esta comparación se ha determinado que, a la segunda implementación, la que realiza todo en una única red, obtiene mejores resultados.

Resultados que, como se ha visto a lo largo del texto, reconstruyen correctamente la imagen, aumentando claramente su visibilidad y el nivel de detalle de esta a la par que se aumenta su resolución; la cual se triplica a lo largo de la reconstrucción. Sí es cierto que, a la hora de trabajar con vídeo, el resultado no es tan bueno, ya que, aunque las imágenes se reconstruyen correctamente, *el frame rate* que puede mantener el vídeo de salida llega justo a los 12 fotogramas por segundo. Esto abre una vía de trabajo, ya sea perfeccionando la red para optimizar su rendimiento.

Por último, se ha realizado una implementación complementaria, que no estaba marcada como objetivo principal pero que también se baja en ayudar y facilitar a los pilotos de drones submarinos su labor, detectando con éxito los láseres del dron y calculando la distancia entre ellos. Lo que permite al piloto tener una referencia y, a partir de ella, pude obtener otros datos de su interés.

Este proyecto siempre se planteó con el objetivo de resolver el problema de la calidad y escasa visibilidad con la que cuentan las imágenes que recibe el piloto durante el pilotaje de un dron submarino. A lo largo del mismo se han explicado las soluciones propuestas para la resolución de este problema y se han mostrado los resultados de las dos implementaciones realizadas, terminando con una comparación entre ambas. A la vista de estos resultados se puede concluir que las implementaciones realizadas solucionan el problema inicial y facilitan de esta manera su labor al piloto. Así todo, existen numerosas posibilidades mediante las cuales mejorar estas soluciones propuestas. A continuación, se comentan algunas de ellas para continuar o mejorar este proyecto:

1. **Aumentar el conjunto de datos de entrenamiento**. Como se ha explicado al comienzo de este capítulo para la implementación realizada se contaba con unas dos horas de video del fondo marino. Si bien este conjunto de datos es suficiente para una primera implementación, si se consiguen aumentar el conjunto de datos con imágenes de diferentes fondos marinos (arena, roca, coral etc.), aumentando de esta forma el conjunto de datos de entrenamiento. Se lograría que la red sea capaz de comprender mejor que es cada cosa que ve en las imágenes, mejorando de esta forma su capacidad para reconstruir.
2. **Probar otros modelos para comparar resultados**. Las implementaciones realizadas se han basado en el modelo *Pix2Pix* propuesto en 2017. Con este modelo se han conseguido buenos resultados, pero existen numerosos modelos creados para la reconstrucción o generación de imágenes. Es recomendable realizar pruebas con otros modelos y así comparar resultados.
3. **Utilizar técnicas de localización y mapeo simultaneo (SLAM) para conocer el trayecto realizado por el dron.** Los algoritmos SLAM permiten trazar en tiempo real un mapa del movimiento realizado por un vehículo. Este tipo de técnicas están siendo utilizadas por los vehículos autónomos para trazar mapas de entornos desconocidos. Por ello, los pilotos del dron submarino se pueden beneficiar de estas técnicas para poder orientarse en entornos desconocidos o de baja visibilidad.

7. Referencias

[1] I. Goodfellow, J. Pouget-Abadie, Mehdi Mirza, B. Xu «*Generative Adversarial Nets*», jun. 2014.

[2] «*Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*». https://junyanz.github.io/CycleGAN/

[3] H. Zhang, « *hanzhanggit/StackGAN*».

[4] F. Rosenblatt, «*The perceptron, a probabilistic model for Information storage and Organization in the brain*». Cornell Aeronautical Laboratory, 1958

[5] R. Salas, «*Redes Neuronales Artificiales*». Departamento de Computación, Universidad de Valparaíso, 1998

[6] D. Rumelhart, G. Hinton y R. Williams «*Learning representations by back-propagating errors*». Universidad de California, 1986.

[7] «*Reducción de la pérdida: Descenso de gradientes*», Google Developers

[8] «*Supervised vs Unsupervised vs Reinforcement Learning*», Intellipaat Blog, dic. 26, 2019.

[9] «*CS231n Convolutional Neural Networks for Visual Recognition*».

[10] «Image-to-Image Translation with Conditional Adversarial Networks». https://phillipi.github.io/pix2pix/

[11] E. Pérez, «*124 años después, escalan a 4K 60fps el clásico de los hermanos Lumière de 1986 utilizando inteligencia artificial*» *Xataka,* feb. 05, 2020.

[12] J. Vincent, «*Artificial intelligence is helping old video games look like new*», *The Verge*, abr. 18, 2019.

[13] Glenn Jocher et al. *Ultralytics/yolov5: v3.0 Zenodo, 2020.*

[14] S. Sumikura, M. Shibuya, k. Sakurada, «*OpenVSLAM: A versatile visual SLAM framwork*», oct. 10, 2019