



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Aplicación de técnicas de aprendizaje profundo a la localización
y seguimiento de objetos en entornos marítimos empleando cámaras
de video portátiles*

Grado en Ingeniería Mecánica

ALUMNO: D. Jesús Moreno Fernández

DIRECTORES: D. José Antonio González Prieto

D. Ramón Touza Gil

CURSO ACADÉMICO: 2022-2023

UniversidadeVigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Aplicación de técnicas de aprendizaje profundo a la localización
y seguimiento de objetos en entornos marítimos empleando cámaras
de video portátiles*

Grado en Ingeniería Mecánica
Intensificación en Tecnología Naval
Cuerpo General

UniversidadeVigo

RESUMEN

Encuadrado dentro de la Inteligencia Artificial, y particularmente en la rama de *Machine Learning*, este TFG pretende la aplicación de técnicas de aprendizaje profundo para resolver problemas relacionados con la captura de videos en entornos marítimos. Su objetivo es aplicar estas técnicas a la detección y seguimiento de embarcaciones, empleando dispositivos de grabación portátiles. Para ello se analizan los modelos de trabajo de *Machine Learning*, centrándose en el de visión por computador.

Concretamente, se ahonda en una de las posibles soluciones que se pueden emplear para este tipo de tareas, las redes neuronales artificiales (ANN). Dentro de este campo, se seleccionan las denominadas redes neuronales convolucionales (CNN), desarrolladas especialmente como sistemas de extracción de características en sistemas de datos estructurados, como imágenes o videos.

Por ello, el presente TFG busca el desarrollo de la solución completa. Es decir, parte de la creación de un *dataset* propio, formatea los datos de entrada, ejecuta los entrenamientos de un modelo, y tras ello, hace la validación de los resultados, buscando optimizar el mismo.

Finalmente, se justifica la elección del algoritmo YOLO como la mejor opción para acometer este trabajo, por su capacidad de obtener soluciones de reducido tamaño, que además operan a gran velocidad, sin tener que renunciar a un alto nivel de precisión en los resultados.

PALABRAS CLAVE

Machine learning, convolución, algoritmo, detección, seguimiento

AGRADECIMIENTOS

Me gustaría empezar los agradecimientos poniendo en valor el absoluto e incondicional apoyo que han supuesto mis dos tutores en este TFG, D. José Antonio y el CF Touza, dedicando gran parte de sus día a día a este trabajo, a pesar de la carga que sus labores docentes les propician. Especialmente a D. José Antonio, por inculcarme la relevancia que está teniendo la aplicación de técnicas de aprendizaje profundo, así como su pasión por ellas.

Por supuesto, agradecer a mi familia el apoyo y comprensión que siempre me han proporcionado a lo largo de toda la carrera, convirtiéndose siempre en puerto base de mi motivación. Quisiera agradecerles especialmente la educación que siempre me dieron, base de incalculable valor sobre la que se asientan todos los logros personales que pudiera alcanzar en la vida.

También me gustaría dedicar unas palabras a la Armada y su personal, por suponer una fuente inagotable de inspiración para mi siempre creciente vocación de servicio, siendo pilar fundamental en mi crecimiento personal y profesional.

No quiero olvidarme de mi estimada promoción 423 – 153, los cuales tendrán siempre un hueco en el pensamiento de este humilde servidor y con los que espero compartir una carrera llena de éxitos. En estos cinco años de travesía, los mejores momentos los he vivido con ellos.

Mis logros, siempre serán los de todos ellos.

CONTENIDO

Contenido	1
Índice de Figuras	4
Índice de Tablas.....	7
1 Introducción y objetivos	9
1.1 Contexto y motivación	9
1.2 Objetivos y alcance	9
1.3 Estructura de la memoria	10
2 Estado del arte	11
2.1 La Inteligencia Artificial.....	11
2.1.1 Definición	11
2.1.2 Contexto histórico.....	12
2.1.3 Inteligencia Artificial en la actualidad.....	13
2.2 <i>Machine Learning</i>	14
2.2.1 Definición	14
2.2.2 Aprendizaje supervisado.....	14
2.2.3 Aprendizaje no supervisado.....	15
2.2.4 Aprendizaje por refuerzo	16
2.2.5 <i>Deep Learning</i>	16
2.3 Visión por computador.....	17
2.3.1 Concepto de red neuronal	18
2.3.2 Redes neuronales convolucionales	23
2.3.3 Detección de objetos en imágenes	26
2.3.4 Algoritmo YOLO.....	28
2.4 <i>Deep Learning</i> en el ambiente marítimo.....	29
2.4.1 Datasets y tipos	29
2.4.2 <i>Singapore Maritime Datasets</i>	31
2.4.3 <i>ABOships onshore and onboard</i>	32
2.5 Aplicaciones para la Armada del S.XXI.....	34
2.5.1 Estado actual de estas técnicas en la Armada	34
2.5.2 Líneas de desarrollo de técnicas <i>Deep Learning</i> en la Armada.....	34
3 Desarrollo del TFG.....	37
3.1 Fase de formación	37
3.2 Dispositivos de grabación portátiles empleados	38
3.2.2 Cámara fotográfica con capacidad de zoom óptico	39

3.3 Entorno de trabajo y <i>softwares</i> empleados	39
3.4 Problemas a resolver	40
3.5 Justificación de los algoritmos empleados.....	41
3.5.1 Algoritmos para detección de objetos.....	41
3.5.2 Algoritmos para seguimiento de objetos en video.....	42
3.6 Prueba inicial.....	43
3.6.1 Etiquetado de imágenes de la <i>Webcam</i> de Cangas	43
3.6.2 Entrenamiento <i>dataset</i> de Cangas.....	44
3.7 Entrenamiento para <i>dataset</i> en Escuela Naval Militar.....	49
3.7.1 Selección del emplazamiento de grabación	49
3.7.2 Etiquetado de videos.....	50
3.7.3 Entrenamiento <i>dataset</i> de la Escuela Naval Militar	51
3.7.4 Introducción de nuevas imágenes al <i>dataset</i>	55
3.7.5 Obtención de videos con <i>tracking</i>	56
3.7.6 Reentrenamiento del modelo	58
3.7.7 Entrenamiento del modelo con <i>dataset</i> compuesto por 750 imágenes.....	59
3.7.8 Entrenamiento final del modelo en Google Colab	63
3.8 Instalación en dispositivo <i>Coral AI</i>	64
4 Resultados y validación.....	65
4.1 Resultados de la prueba realizada con el <i>dataset</i> de Cangas con YOLOv5	65
4.2 Validación de los resultados obtenidos en entorno VSCodium con YOLOv8n	66
4.3 Validación de los resultados obtenidos en entorno Google Colab con YOLOv8s	67
4.4 Resultados y validación general.....	70
5 Conclusiones y líneas futuras	73
5.1 Conclusiones extraídas.....	73
5.2 Objetivos alcanzados.....	74
5.3 Tendencias y líneas futuras	74
6 Bibliografía.....	77
Anexo I: Implicaciones Sociales, y/o Económicas, y/o Ambientales	83
Anexo II: Reflexiones Éticas y Sociales	85
Anexo III: Glosario de siglas y acrónimos	87
Anexo IV: <i>Script</i> para importación de imágenes	89
Anexo V: <i>Script</i> para <i>tracking</i>	91
Anexo VI: <i>Script</i> para exportación de imágenes	93

Anexo VII: <i>Script</i> para entrenamiento del modelo	97
--	----

ÍNDICE DE FIGURAS

Figura 2-1 ELIZA es considerado el primer asistente personal, capaz de tener una conversación con un interlocutor humano [4]	12
Figura 2-2 La domótica es la rama especializada en desarrollar sistemas capaces de coordinar el control de distintos elementos inteligentes [5].....	13
Figura 2-3 Diagrama que relaciona la IA con el Machine Learning [6]	14
Figura 2-4 Esquema de aprendizaje supervisado [6].....	15
Figura 2-5 Esquema genérico para un modelo de aprendizaje no supervisado [7]	15
Figura 2-6 Modelo de aprendizaje por refuerzo [7]	16
Figura 2-7 Comparativa de una red simple con una red neuronal de <i>Deep Learning</i> [10]	17
Figura 2-8 Comparativa entre el sistema de visión humano y la visión por computador	17
Figura 2-9 Comparativa entre detección y segmentación de objetos [13]	18
Figura 2-10 Representación de una red neuronal <i>perceptron</i> , particularmente de los <i>weights</i> asociados a las neuronas [17].....	19
Figura 2-11 Ejemplo de red neuronal recurrente (RNN) [18].....	20
Figura 2-12 Infografía que recoge algunos de los muchos tipos de red neuronales que existen [8].....	21
Figura 2-13 Ejemplo de regresión para una red con función de activación lineal	22
Figura 2-14 Línea temporal con los principales hitos en la historia de las CNN [21]	24
Figura 2-15 Proceso de convolución que aplica una CNN a una imagen de entrada [22]	25
Figura 2-16 Ejemplo de operación realizada por una <i>Convolutional layer</i> [21].....	25
Figura 2-17 En la figura se muestran los tipos de capas mencionados anteriormente, dentro de una CNN [21].....	26
Figura 2-18 Entre las aplicaciones de la detección de objetos, podemos encontrar alguna como la detección de peatones en un entorno urbano [23].....	27
Figura 2-19 Arquitectura de una red con 24 capas convolucionales [27]	28
Figura 2-20 Resultados del algoritmo YOLO aplicado a la detección de buques, particularmente en situación de niebla.....	30
Figura 2-21 Imágenes tomadas con un robot submarino con sistema <i>Deep Learning</i> integrado [33]	31
Figura 2-22 Esquema que incorpora los dos procesos realizados por <i>Singapore Maritime Datasets</i> para la detección y seguimiento de buques [34]	31
Figura 2-23 Seguimiento en video obtenido por <i>Singapore Maritime Dataset</i> [34].....	32
Figura 2-24 Ejemplo de una de las imágenes del <i>dataset</i> de ABOShips empleadas para la detección de buques en entornos marítimos [35]	33
Figura 2-25 Ejemplo de aplicación de técnicas de detección y seguimiento de buques, en los cristales del puente de un buque de la Armada (fuente: www.armada.defensa.gob.es)	35
Figura 2-26 Ejemplo de integración de sistemas de armas para proporcionar la máxima ayuda en la decisión del operador (fuente: periódico El Español)	36

Figura 3-1 Línea secuencial que acota los campos desarrollados en el presente trabajo (autoría propia)	37
Figura 3-2 Teleobjetivo portátil para smartphone de la marca Selvim	39
Figura 3-3 Cámara digital Kodak Pixpro AZ422 empleada para la obtención de imágenes	39
Figura 3-4 Modelos con los que permite entrenar la herramienta <i>software</i> Ultralytics HUB [41] ..	40
Figura 3-5 Imagen ejemplo de un etiquetado realizado con el programa label-studio (autoría propia)	44
Figura 3-6 Reparto de las imágenes que componen el <i>dataset</i> "Webcam Cangas" según el propósito que se le asigna a cada una de ellas (gráfica generada por Ultralytics HUB de autoría propia)	45
Figura 3-7 Gráfica generada por Ultralytics HUB que permite visualizar los resultados del mAP, recall y precision para cada uno de los epochs evaluados (gráfica generada por Ultralytics HUB de autoría propia)	46
Figura 3-8 Gráfica que muestra el <i>box loss</i> para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)	47
Figura 3-9 Gráfica que muestra el <i>class loss</i> para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)	48
Figura 3-10 Gráfica que muestra el <i>object loss</i> para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)	48
Figura 3-11 Imagen sometida a <i>test</i> tras haber entrenado el modelo con YOLOv5 en Ultralytics HUB (autoría propia)	49
Figura 3-12 Captura extraída de <i>Google Maps</i> que refleja el emplazamiento elegido para la generación del <i>dataset</i> (autoría propia)	50
Figura 3-13 Imagen ejemplo de las seleccionadas para realizar el entrenamiento del <i>dataset</i> de la Escuela Naval Militar (autoría propia)	51
Figura 3-14 Reparto de las imágenes que componen el <i>dataset</i> "Infantes" según el propósito que se le asigna a cada una de ellas (captura extraída de Ultralytics HUB)	52
Figura 3-15 Gráfica generada por Ultralytics HUB que permite visualizar los resultados del mAP, recall y precision para cada uno de los epochs evaluados (gráfica generada por Ultralytics HUB de autoría propia)	53
Figura 3-16 Gráfica que muestra el <i>box loss</i> para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)	54
Figura 3-17 Gráfica que muestra el <i>class loss</i> para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)	55
Figura 3-18 Gráfica que muestra el <i>object loss</i> para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)	55
Figura 3-19 Ejemplo de un <i>frame</i> de los videos obtenidos al ejecutar el <i>tracking</i> (autoría propia) ..	57
Figura 3-20 Ejemplo de error observado durante el <i>tracking</i> de los videos (autoría propia)	58
Figura 3-21 Fragmento del <i>script</i> empleado para el entrenamiento del modelo (autoría propia)	58
Figura 3-22 Fragmento del <i>script</i> empleado para el entrenamiento del modelo (autoría propia)	59
Figura 3-23 Matriz de confusión generada durante el entrenamiento de la iteración número 9 (autoría propia)	60

Figura 3-24 Gráfica que muestra el <i>precision</i> para cada clase de objeto detectado (autoría propia)	61
Figura 3-25 Gráfica que muestra el <i>recall</i> para cada clase de objeto detectado (autoría propia)	61
Figura 3-26 Gráfica generada por el <i>script</i> y que relaciona ambos parámetros (autoría propia).....	62
Figura 3-27 Imágenes generadas por el modelo de 750 imágenes durante la ejecución del <i>script</i> de entrenamiento (autoría propia).....	62
Figura 3-28 Esquema que detalla el proceso llevado a cabo durante el desarrollo del presente TFG (autoría propia).....	64
Figura 4-1 Imagen obtenida tras el entrenamiento del <i>dataset</i> de Cangas en Ultralytics HUB con YOLOv5 (autoría propia).....	66
Figura 4-2 Matriz de confusión generada durante el entrenamiento en entorno Google Colab (autoría propia)	67
Figura 4-3 Gráfica que muestra el <i>precision</i> para cada clase de objeto detectado (autoría propia).	68
Figura 4-4 Gráfica que muestra el <i>recall</i> para cada clase de objeto detectado (autoría propia)	68
Figura 4-5 Gráfica que representa la curva <i>precisión-recall</i> y que relaciona ambos parámetros (autoría propia).....	69
Figura 4-6 Recopilatorio de imágenes generadas por el modelo durante el entrenamiento en entorno Google Colab (autoría propia)	70
Figura 4-7 Error mostrado en uno de los videos generados con <i>tracking</i> que demuestra las limitaciones del modelo cuando no capta los objetos en su totalidad (autoría propia).....	71

ÍNDICE DE TABLAS

Tabla 1 Comparación del <i>dataset</i> ABOShips con otros similares en entornos marítimos [35]	33
Tabla 2 Comparativa entre las capacidades de ambas versiones de YOLO, en términos de detección de objetos [46].....	42
Tabla 3 Resumen de los parámetros a tener en cuenta en las iteraciones realizadas (autoría propia)	59
Tabla 4 Tabla comparativa de las diferentes versiones de YOLOv8 (fuente: GITHUB)	63

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Contexto y motivación

Las tecnologías que emplean inteligencia artificial están en auge y cada vez están más presentes en todos los ámbitos de nuestra vida, máxime, si hablamos de *Machine Learning*, y particularmente, si hablamos de *Deep Learning*.

Cabe destacar la relevancia de la que dota a estas técnicas el AJEMA en las líneas generales de la Armada 2022 [1], directiva en la que se refiere a los retos a los que hace frente ésta en los próximos años. En ellas se recoge la inminente necesidad de la Institución de impulsar el proceso de transformación digital, con el fin de contar con una Armada más “ágil, simple y flexible” [1]. Se menciona la obligatoriedad de optimizar el funcionamiento de la organización, desde su orgánica, hasta el empleo de tecnologías digitales en la capacidad de decisión de todos los niveles de mando, así como en el combate del presente, y del futuro. Habla especialmente de la incorporación de tecnologías como el tratamiento masivo de datos, la introducción de gemelos digitales, la capacidad de supercomputación, la automatización y el empleo de técnicas de inteligencia artificial. Por último, resalta que el fin último de todo esto debe ser incorporar esta transformación a la principal razón de ser de la Armada, que es su Flota, siendo progresiva su integración entre unidades de última generación, y otras de mayor veteranía.

Contextualizando este trabajo, se sitúa en la línea de investigación denominada *edge computing* [2]. Dicha línea de investigación tiene como objetivo el desarrollo de sistemas de IA distribuidos, donde sensores de distinta naturaleza (como es una cámara, en nuestro caso) realizan un procesamiento inteligente de los datos. Esto permite prescindir del tradicional procesamiento masivo en grandes bases de almacenamiento, y generar sistemas capaces de aprender y realizar tareas repetitivas de forma muy automatizada. Además, este procesamiento más cercano a los sensores que generan los datos, permite ahorrar y evitar problemas de traslado de información.

1.2 Objetivos y alcance

El objetivo que se propone este trabajo es aplicar técnicas de aprendizaje profundo (*Deep Learning* en su traducción al inglés) a la detección y seguimientos de objetos en entornos marítimos, empleando medios de grabación que pueda portar cualquier persona. La finalidad última que tiene es afianzar la detección y seguimiento de objetos en el entorno marítimo, es decir, buques en puerto o sus inmediaciones, para clasificar el tráfico que circula por una zona marítima. La particularidad de éste reside en hacerlo con medios de grabación portátil, como podrían ser una cámara de video o un teléfono móvil.

Con este fin se plantea el uso de técnicas de aprendizaje supervisado *Deep Learning*, seleccionando el algoritmo YOLO (algoritmo específico para detección y seguimiento de imágenes) como el más

adecuado para el problema que se desea resolver. Para ello se hará uso de varios softwares de etiquetado y entrenamiento de redes neuronales convolucionales (CNN), y se analizarán los parámetros que permitan determinar si nuestro sistema es capaz de acometer la tarea requerida.

El mayor reto de este trabajo será hacer frente a los problemas que propongan los medios de grabación portátil, y analizar los resultados para distintas cámaras y zooms escogidos (digital y óptico).

1.3 Estructura de la memoria

Una vez contextualizado, y presentados los objetivos que se propone este trabajo, se analiza la estructura que sigue el presente documento. Su finalidad no es otra que la de presentar el trabajo realizado en todas sus fases, los resultados obtenidos y las conclusiones a las que nos ha conducido. Por ello, este trabajo se divide de la siguiente manera:

- **Introducción y objetivos.** En él se expone el papel que juega la inteligencia artificial hoy en día, su importancia en el mundo actual y las posibilidades de explotación que puede tener la aplicación de técnicas de aprendizaje profundo para la resolución de tareas, concretamente en el entorno marítimo. Además, se presentan los objetivos que se propone el presente trabajo y su alcance. Por último, se expone la organización de la memoria.
- **Estado del arte.** En este segundo título se analiza el encuadre teórico en el que se sitúa el trabajo y conceptos que ayudan a entender la finalidad del mismo. Los contenidos que se revisan en él son los siguientes:
 - Introducción e historia de la inteligencia artificial, y su empleo en la actualidad.
 - Técnicas *Machine Learning*, así como métodos de aprendizaje y focalización expresa en *Deep Learning*.
 - Visión por computador. Conceptos de gran importancia como las redes neuronales convolucionales y el algoritmo YOLO, particularmente utilizado en detección y seguimiento de objetos.
 - Detección y seguimiento de objetos en el entorno marítimo. Qué es un *dataset* y cuáles podemos encontrar, además de centrarse en dos de gran relevancia: *Singapore Maritime Datasets* y *ABOships onboard and onshore*.
 - Aplicaciones para la Armada del S. XXI, centrándose en lo que ya hay implantado, así como sus posibilidades de desarrollo en distintos ámbitos.
- **Desarrollo del TFG.** En este capítulo se describe el proceso de formación previo al trabajo y los softwares empleados en las distintas etapas, así como la justificación del algoritmo empleado. Además, se describen detalladamente los dos problemas principales por los que pasa la resolución del problema expuesto, que son la detección de objetos en imágenes (buques en este caso), y el seguimiento de éstos en video. A su vez, estos dos problemas son divididos en dos, consistentes en una fase de etiquetado y una de entrenamiento de la red.
- **Resultados obtenidos y validación.** En el presente capítulo se exponen los resultados obtenidos, indicando que objetivos se han alcanzado y el porcentaje de cumplimiento de los problemas planteados al inicio del trabajo. También se exponen aquellos que no se han podido acometer, y posibles soluciones a aplicar en un futuro.
- **Conclusiones y líneas futuras.** En él se plantean las conclusiones extraídas tras la validación de los resultados obtenidos. Además, se plantean las posibles líneas futuras en las que podría derivar el desarrollo complementario de este trabajo y cómo optimizar los resultados obtenidos.

2 ESTADO DEL ARTE

Este capítulo analiza la contextualización en la que se encuadra el presente trabajo, analizando conceptos teóricos necesarios para el entendimiento del mismo, cómo son la Inteligencia Artificial y sus técnicas derivadas de *Machine Learning* y *Deep Learning*. Además, se añade un breve contexto histórico de los mismos y sus aplicaciones en la actualidad. Seguidamente se trata el concepto de red neuronal artificial y los distintos métodos de aprendizaje de éstas. Posteriormente, se explica el concepto de visión por computador y su gran importancia para resolver problemas de detección de objetos en imágenes, así como una presentación somera del algoritmo YOLO. Por otro lado, se exponen las particularidades de este tipo de tareas en el entorno marítimo y los *datasets*, entendidos como las bases de datos que emplean estas redes para aprender a resolver los problemas planteados. Finalmente, se revisa la actualidad de estas técnicas en el ámbito de la Armada y sus posibles líneas de desarrollo.

2.1 La Inteligencia Artificial

2.1.1 Definición

Partiendo de líneas generales, y ante la necesidad de ubicar el trabajo que nos atañe, encontramos necesario entender qué es la inteligencia artificial. Siendo ya presente, y por supuesto futuro, la inteligencia artificial es la habilidad con la que se pretende dotar a las máquinas de capacidades de razonamiento similar al de los seres humanos, sobre todo explotando campos como el del razonamiento lógico, la creatividad, el aprendizaje o la capacidad de planeamiento. [3]

Ya no parece futurista pensar que seamos capaces de desarrollar, en un futuro próximo, no solo sistemas capaces de almacenar y tratar datos, sino sistemas que perciban lo que ocurre en su entorno más próximo y sean capaces de relacionarse con él, ofreciendo respuestas a problemas, y definir un patrón de actuación propio que se adapte a infinidad de situaciones.

Si nos remitimos en primer lugar a una clasificación genérica, encontramos dos tipos de inteligencia artificial:

- Software: asistentes virtuales, análisis de imágenes, sistemas de reconocimiento de voz, etc. (es dónde ubicaríamos la rama que desarrolla este trabajo)
- Inteligencia artificial integrada: es la destinada a dotar a elementos mecánicos o máquinas, de habilidades que sustituyen, en la mayoría de los casos, la acción humana. Ejemplos de esto son los robots, drones, vehículos autónomos, etc. Cabe destacar que ésta siempre necesitará de software.

Una vez llegados a este punto, podríamos plantear por qué es tan importante el papel que juega la inteligencia artificial, o si de verdad es conveniente desde muchos puntos de vista. El principal motivo por el que la IA juega un papel crucial en la transformación digital es la cantidad de bases de datos de

las que se dispone en la actualidad, gracias al progreso tecnológico. De esto es culpable también el abismal desarrollo al que se ha visto sometido el campo de la informática, lo que permite eliminar muchas tareas repetitivas y encontrar soluciones (mediante algoritmos) a muchos problemas que se dan en la actualidad.

2.1.2 Contexto histórico

Para encontrar los orígenes de la inteligencia artificial tenemos que remontarnos a la Segunda Guerra Mundial, donde *Alan Turing*, tras descifrar en 1950 la máquina enigma, capaz de decodificar los mensajes cifrados alemanes, plantea en un artículo científico si las máquinas podrían llegar a pensar. Es entonces cuando presenta el Test de *Turing*". En dicho test, Turing quería comprobar la capacidad de una máquina de exhibir un comportamiento humano, por ejemplo, entablando una conversación con una persona. Estas máquinas serían programadas para generar respuestas similares a la de los humanos. Si tras cinco minutos de conversación, el examinador no era capaz de concluir si era una máquina u otro interlocutor, la máquina superaría la prueba. [4]

En 1956, la conferencia de *Dartmouth* reunió a científicos como *Marvin L. Minsky* y *John McCarthy* con el fin de aclarar términos y desarrollar ideas sobre las máquinas "pensantes". Sería este último quien eligió el término de inteligencia artificial para un nuevo campo de investigación.

Otro gran paso en este ámbito fue el desarrollo en 1966 de ELIZA (Figura 2-1). Aunque podamos pensar que los asistentes personales son inventos relativamente recientes, hace más de 50 años *Joseph Weizenbaum*, un profesor de informática de *Massachusetts*, desarrolló un programa capaz de dar conversación a su interlocutor y en cierto modo empatizar con los problemas de éste. Las dos técnicas que utilizó principalmente para conseguirlo fueron, por un lado, una base de datos de palabras claves para correlacionar los temas, y por otro lado, una base de datos de frases hechas sobre una gran cantidad de temas.

```
=====
EEEEEEEE L      IIIIII ZZZZZZZ AAA
E      L      I      Z      A  A
E      L      I      Z      A  A
EEEEEE L      I      Z      A  A
E      L      I      Z      AAAAAA
E      L      I      Z      A  A
EEEEEEEE LLLLLLL IIIIII ZZZZZZ  A  A
=====
ELIZA > (PLEASE TYPE IN ALL CAPS) WHAT'S YOUR NAME DEAR ?
AMIT
ELIZA > HI, I'M ELIZA. WHAT DO YOU WANT TO TALK ABOUT ?
AMIT > CAN I TALK ABOUT MY PROBLEM ?
ELIZA > SURE... ?
AMIT > DUE TO ON GOING EXAMS IN COLLEGE, I HAVE A LOT OF STRESS.
ELIZA > PLEASE RELAX A LITTLE. SLEEP WELL.
AMIT > THANKS FOR YOUR ADVICE.
ELIZA > NO MENTION.
AMIT > BYE
ELIZA > BYE AND KEEP IN TOUCH...
=====
```

Figura 2-1 ELIZA es considerado el primer asistente personal, capaz de tener una conversación con un interlocutor humano [4]

En 1979, un profesor en la Universidad *Carnegie Mellon* de Estados Unidos consigue crear un programa informático que vence al campeón del mundo de *backgammon* de la época, Luigi Villa. Esto fue sin duda un incentivo para seguir desarrollando el mundo de los videojuegos, haciéndolos cada vez más complejos.

Tres años más tarde, Japón invierte esfuerzos en desarrollar un programa de ordenadores de quinta generación que utilice técnicas de inteligencia artificial desarrolladas hasta el momento. El principal objetivo que se proponían con esta generación era integrar sistemas capaces de aprender y traducir

automáticamente de un idioma a otro, algo que nos parece básico hoy en día. Lamentablemente, diez años más tarde y sin apenas resultados, finaliza este proyecto y su fracaso se suma al resto.

Tras un “invierno” para la inteligencia artificial, causado en gran parte por el decremento de las inversiones y los escasos resultados obtenidos en los proyectos llevados a cabo, en 1997 IBM desarrolló un ordenador capaz de ganar al entonces campeón del mundo de ajedrez, *Garri Kaspárov*.

Entrando en el siglo XX, la Universidad de *Standford* consigue desarrollar un coche autónomo que ganó una competición de vehículos robots tras conducir 212 kilómetros por el desierto, sin ninguna persona al volante. Esto ocurrió en 2005, siendo 9 años más tarde cuando un ordenador superaría con éxito el Test de *Turing*. Este programa se llamaba *Eugene* y fue desarrollado en Rusia.

En 2016, la multinacional Microsoft lanza un *chatbot* capaz de aprender a partir de la interacción con las personas. Acabó siendo un fracaso ya que en un solo día se convirtió en racista, xenófobo y homófobo. Ese mismo año, un programa desarrollado por *Google DeepMind* venció al campeón mundial del famoso juego Go. Supuso uno de los hitos más importantes en cuanto a técnicas de aprendizaje profundo. Un año más tarde y siguiendo en el campo de los videojuegos, un software creado en la Universidad de *Carnegie Mellon* (EEUU) consigue vencer a rivales humanos en un torneo de póker en Texas.

2.1.3 Inteligencia Artificial en la actualidad

Como hemos visto en los apartados anteriores, la inteligencia artificial no solo ha evolucionado a una velocidad estratosférica, sino que además de tener un futuro prometedor, ya se encuentra entre nosotros en más aplicaciones de las que podríamos pensar.

Un ejemplo de ellos al alcance de todos es el comercio por internet. Son ya muchas las empresas que están aprovechando esta posibilidad para ofrecer recomendaciones personalizadas de compras a sus consumidores. Además, está siendo incluida en muchos comercios para optimizar los productos, los procesos logísticos o incluso disponer de inventarios inteligentes.

Otro de los ejemplos más comunes son los asistentes virtuales, evidenciando que la inteligencia artificial está al alcance de todos en nuestros teléfonos móviles.

No solo la tenemos a disposición personal, también podemos encontrar sistemas que están empezando a implementarla en servicio al funcionamiento de la sociedad. Muchas empresas están optando por el desarrollo de casas, vehículos e infraestructuras inteligentes (Figura 2-2) con el objetivo de optimizar el rendimiento de estos. Imagine un semáforo que sea capaz de evaluar el tráfico para determinar el paso de vehículos, o un termostato inteligente que fuese capaz de adaptar nuestro suministro de energía a los hábitos de consumo. [5]



Figura 2-2 La domótica es la rama especializada en desarrollar sistemas capaces de coordinar el control de distintos elementos inteligentes [5]

2.2 Machine Learning

2.2.1 Definición

Una vez comprendido el concepto de inteligencia artificial, se hace casi obligatorio hablar de *Machine Learning*. Este término es asociado dentro de la misma al subconjunto destinado a enseñar a las máquinas o computadoras (Figura 2-3), a aprender de los datos introducidos y llegar a optimizar los resultados mediante lo que conocemos como algoritmos de entrenamiento. De esta forma, los sistemas son capaces de encontrar soluciones que discriminen los elementos que conforman la entrada, con el fin de encontrar patrones y correlaciones en grandes *datasets*¹. [6]

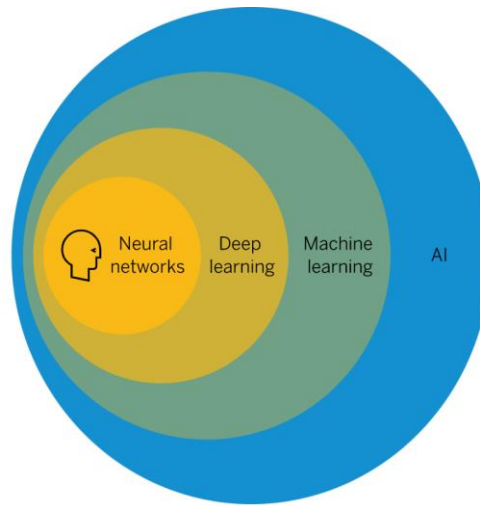


Figura 2-3 Diagrama que relaciona la IA con el Machine Learning [6]

2.2.2 Aprendizaje supervisado

Entre los cuatro modelos de Machine Learning que vamos a tratar, hablaremos primero del conocido como aprendizaje supervisado. En este modelo, la máquina aprende mediante ejemplos utilizando algoritmos supervisados, es decir, algoritmos en los que se dispone de los datos de entrada, que representan la pregunta que se la hace al sistema, así como de los datos de salida, que representan las respuestas deseadas, como se puede ver en la Figura 2-4. Procesando una gran cantidad de datos de entrada en las distintas capas, el sistema determina similitudes correlativas y diferencias hasta que es capaz de anticipar estos patrones de clasificación y ofrecer respuestas por sí solo [7]. Este método de aprendizaje es muy recurrente en aplicaciones con las que cualquier persona puede interactuar hoy en día, como las de recomendación para productos o análisis (como por ejemplo *Waze*, que analiza el tráfico y ofrece la solución de la ruta más rápida para diferentes horas del día). Dentro del aprendizaje supervisado existen dos tipos de aprendizaje principales:

- Clasificación: en este tipo de aprendizaje supervisado, la variable de salida del problema será una categoría o conclusión. El modelo recibe una o más entradas e intenta predecir el valor de uno o varios resultados. Esto lo hace a través de etiquetas categóricas que utiliza para llevar a cabo su entrenamiento, y con la finalidad de clasificar nuevos datos no vistos hasta el momento.
- Regresión: por otro lado, los modelos que resuelven problemas de regresión serán los encargados de determinar una recta o plano óptimo, que defina el conjunto de datos que se le han introducido. En estos modelos, la variable de salida es un valor real y continuo, como

¹ Este concepto se describe en subapartados posteriores.

por ejemplo una recopilación de pesos de una población, o los tiempos obtenidos en una carrera de atletismo. [8]

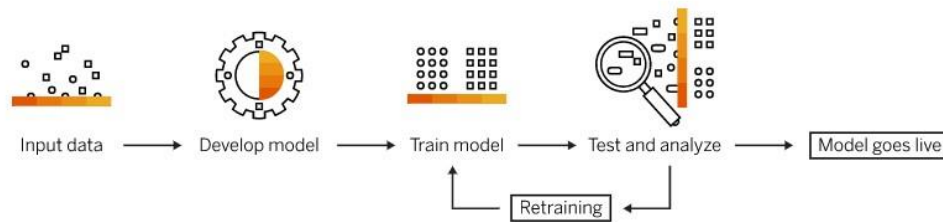


Figura 2-4 Esquema de aprendizaje supervisado [6]

En problemas complejos, la respuesta requerida al sistema inteligente suele ser una combinación de múltiples tipos básicos de salidas. Así, por ejemplo, en la detección de objetos en imágenes, cada uno de los objetos detectados se identifica mediante cuatro números que establecen la posición y dimensiones del rectángulo que enmarca el objeto. Además, representan una regresión múltiple por cada objeto detectado, así como la identificación del tipo de éste. En este tipo de problemas, el algoritmo debe decidir cuántos objetos hay en una imagen que se pretende analizar.

2.2.3 Aprendizaje no supervisado

En este método de aprendizaje, se dispone únicamente de datos de entrada como refleja la Figura 2-5. El objetivo del aprendizaje es determinar estructuras o patrones que existen en los datos de entrada, de forma que nos permita estudiar su comportamiento para un cierto dato introducido. Sistemas típicos en este ámbito son los algoritmos de clusterización, que determinan el número de categorías en las que puede ser organizada la información de entrada, así como a qué categoría pertenece un nuevo dato.

Podríamos asimilarlo al aprendizaje humano, en el que nuestra fuente de información no es solo el conocimiento que tenemos, sino que además nos basamos en otros sentidos como la experiencia o la intuición a la hora de tomar decisiones [7]. Además, a medida que se experimenta con más ejemplos de un determinado problema, su capacidad de categorizar e identificar se vuelve cada vez más precisa e intuitiva. Ejemplos cotidianos de este método de aprendizaje podrían ser el reconocimiento facial que incorporan nuestros dispositivos o la investigación de mercados financieros.

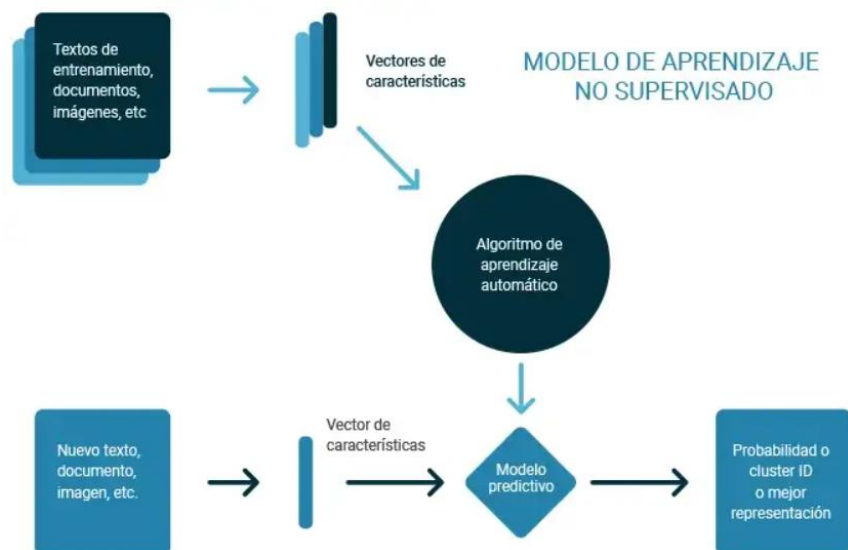


Figura 2-5 Esquema genérico para un modelo de aprendizaje no supervisado [7]

2.2.4 Aprendizaje por refuerzo

A diferencia de los modelos anteriores, especialmente el aprendizaje supervisado, en el aprendizaje por refuerzo el modelo no incluye una respuesta de referencia como entrada, sino que ofrece un conjunto de parámetros y límites establecidos, premiándolo (o no) con finales y estados potenciales [7]. Es decir, al igual que un animal doméstico, estaríamos enseñando a nuestra máquina por experiencia y recompensa, traduciendo esta recompensa en números y haciéndole saber al sistema que ese es el tipo de resultados que el algoritmo que genere debe buscar. Ejemplos de este modelo de aprendizaje los podemos encontrar en juegos desarrollados por ordenador.

MODELO DE APRENDIZAJE POR REFUERZO



Figura 2-6 Modelo de aprendizaje por refuerzo [7]

2.2.5 Deep Learning

Como hemos visto cuando definíamos el concepto de *Machine Learning*, es una rama subconjunto dentro de la inteligencia artificial (Figura 2-3), pero dentro de éste también podemos encontrar infinidad de subconjuntos, siendo el *Deep Learning* uno de ellos. Por tanto, definimos el *Deep Learning* como el proceso de *Machine Learning* llevado a cabo por una red neuronal artificial compuesta de distintos niveles jerárquicos. Así, la red aprende algo muy simple en sus niveles iniciales, generando una información más compleja que es transportada a los niveles superiores, de forma que cada capa aumenta la capacidad de abstracción del algoritmo, permitiéndolo mejorar sus resultados. Sin embargo, el precio a pagar, es que las necesidades computacionales relacionadas con el incremento exponencial de parámetros, dificulta enormemente su entrenamiento y aplicación en dispositivos con capacidades reducidas.

Un ejemplo de esto capaz de resolver un problema podría ser la diferenciación entre dos animales como pueden ser un perro y un gato. Esta metodología podría resolverlo, por ejemplo, estudiando los contrastes entre zonas claras y oscuras para determinar los bordes de una silueta. La primera capa, probablemente, resuelva el problema de buscar bordes entre zonas claras y oscuras. La siguiente capa podría intentar resolver con estos contrastes si detecta una oreja o unos ojos. La siguiente, con toda esta información, podría intentar resolver si se trata de un perro y un gato, y así sucesivamente.

En definitiva, nos permitirá resolver problemas complejos, a través una fragmentación en capas, tal y como refleja la Figura 2-7 de problemas más sencillos. Como podemos imaginar, su capacidad de aplicación y de explotación es enorme, y es por ello que muchas empresas están apostando ya por su investigación. [9]

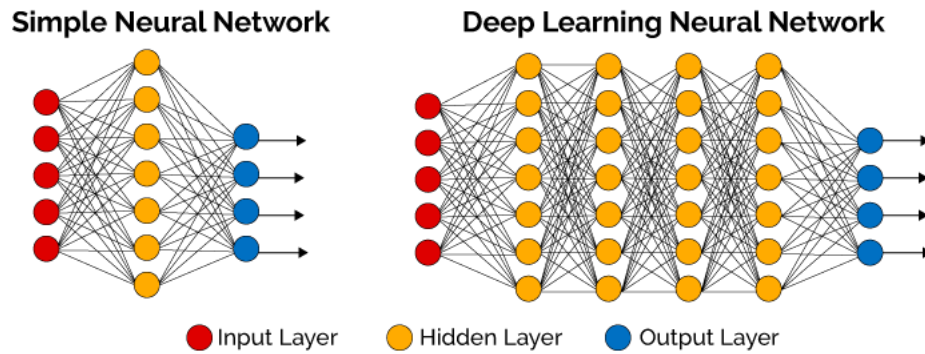


Figura 2-7 Comparativa de una red simple con una red neuronal de *Deep Learning* [10]

2.3 Visión por computador

Acotando el campo de actuación de este TFG, se reduce ya a uno de los múltiples propósitos que tiene la aplicación de técnicas *Machine Learning* y *Deep Learning*, que es el de visión por computador. Dotar a las máquinas de la capacidad de ver no es un proceso sencillo, ya que, si bien el ser humano posee un sistema de visión que le permite captar y procesar imágenes en tiempo real, no es una tarea sencilla de programar para ser empleada por dispositivos electrónicos. La visión por computador tiene, por tanto, la difícil tarea de reunir toda la información recogida por uno o varios sensores, procesarla en una unidad de procesamiento, para finalmente obtener unos datos de salida que le permitan obtener conclusiones (Figura 2-8). [11]

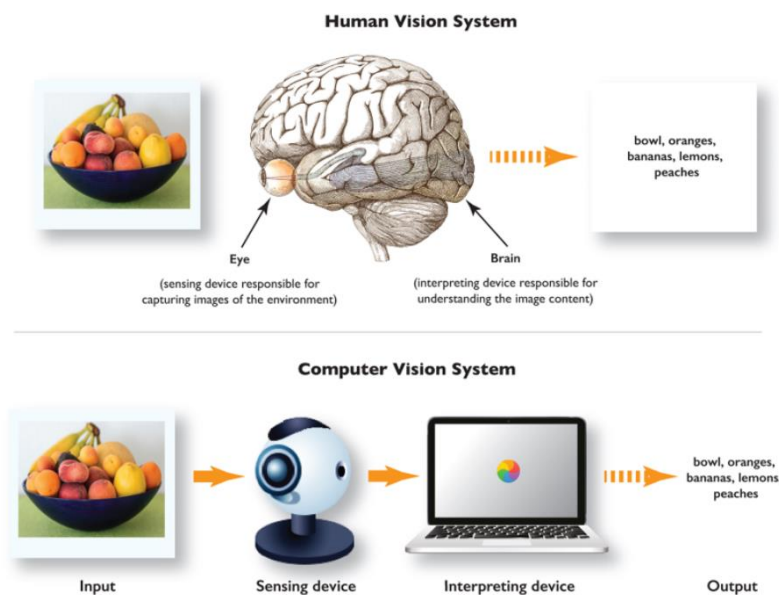


Figura 2-8 Comparativa entre el sistema de visión humano y la visión por computador

Si atendemos a sus posibilidades, podemos encontrar distintos tipos de problemas a los que puede hacer frente la visión por computador [12]. Algunos de los más importantes son los siguientes:

- **Clasificación:** es el problema más sencillo que pretende resolver la visión por computador. Para una de entrada, el modelo determina su clase, en base a las clases que ya ha aprendido, como por ejemplo, identificar si una foto representa un perro o un gato.
- **Detección de objetos:** el modelo aprende a localizar y señalar objetos en una imagen o video mediante cajas o *bounding boxes*. La combinación del problema de clasificación y la detección de objetos de forma simultánea, generan el problema básico al que hace frente el presente TFG. [13]

- **Segmentación:** en este caso, y a diferencia de la detección de objetos, el sistema sí que atribuye una clasificación a cada píxel. Ya no sólo trata de detectar objetos mediante cajas, sino que toma una imagen de entrada, y crea una nueva versión de la imagen, mostrando el conjunto de píxeles que corresponden a cada objeto clasificado, definiendo áreas como las representadas en la Figura 2-9. El mayor requerimiento de este tipo de problemas suele ser la necesidad de una alta resolución de las imágenes para obtener resultados efectivos. [14]

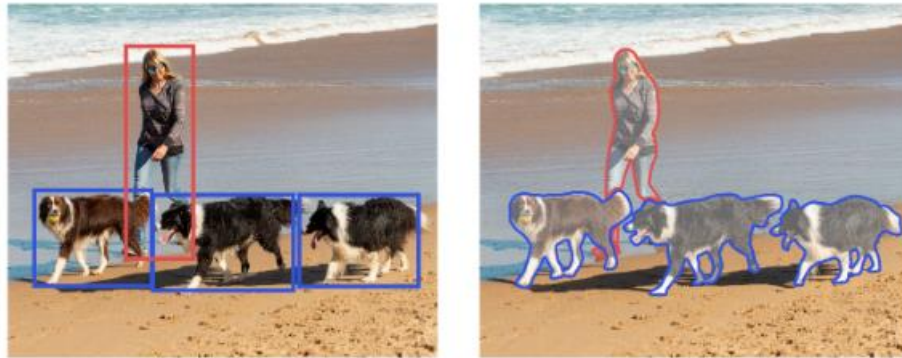


Figura 2-9 Comparativa entre detección y segmentación de objetos [13]

- **Seguimiento (*tracking*) de objetos en video:** los modelos dedicados a *tracking* resuelven el problema de identificar objetos y asignarles una identidad permanente, de manera que sean capaz de detectarlos y realizarles un seguimiento durante toda su aparición en el video. Este problema requiere de una fase de detección y clasificación de objetos, seguida por una fase de asignación de identificadores únicos a cada objeto.
- **Keypoint detection:** además de detectar objetos, en este tipo de problemas se pueden detectar *keypoints* o puntos clave de los objetos detectados. En el caso de la detección de personas, permitiría detectar las partes de su cuerpo tales como su cabeza, articulaciones o extremidades.
- **Gesture detection:** como su nombre indica, pretende detectar el cambio en los gestos humanos a través de técnicas de aprendizaje profundo. Hoy en día supone uno de los mayores retos en la resolución de problemas de visión por computador. [15]

A continuación, se hace un estudio sobre el concepto de red neuronal y sus tipos. Además, se profundiza en la estructura de aprendizaje que es de mayor interés para el presente TFG, el de red neuronal convolucional. Por otro lado, se aborda uno de los problemas que relaciona la visión por computador con el problema que plantea este trabajo, que es el de detección de objetos, repasando brevemente su historia y los algoritmos que se emplean en esta tarea. Finalmente, se hace lo mismo con el otro problema principal al que hace frente este trabajo, es decir, el problema de seguimiento de objetos en video.

2.3.1 Concepto de red neuronal

Para poder “enseñar” a las máquinas a procesar datos y generar patrones de conocimiento, podemos construir lo que conocemos como redes neuronales. Una red neuronal artificial (ANN) es un conjunto de neuronas que pretende imitar el modelo de funcionamiento de un cerebro biológico. Estas neuronas es lo que trataremos como nodos, agrupándose estos en múltiples capas operando en paralelo. En toda red neuronal encontramos siempre una capa de entrada (donde serán introducidos los datos a procesar), capas ocultas, encargadas de aportar un procesamiento determinado y ofrecer los datos de salida a la siguiente capa, y por último una capa de salida con los resultados obtenidos. Como veremos a continuación, el diseño y generación de estas redes neuronales a partir de lo que hemos llamado anteriormente como “entrenamiento”, resultará en un mejor reconocimiento de los datos y una experiencia de aprendizaje más eficiente. [16]

2.3.1.1 Tipos de redes neuronales

Las redes neuronales están en continuo desarrollo y es por ello por lo que podemos encontrar diversas arquitecturas (Véase Figura 2-12). Algunas de ellas son las siguientes: [16]

- **Perceptron neural networks:** este modelo de red neuronal fue el primero en desarrollarse y el más simple, por lo que sirve en gran medida para entender el resto de redes. Esta red funciona tomando una serie de *inputs* numéricos acompañados de lo que conocemos como *weights* o pesos (véase Figura 2-104). La multiplicación de éstos por los *inputs* de entradas, y en su conjunto, la suma de todos estos productos es lo que permitirá construir un polinomio que defina un patrón para la red neuronal. Además, a este polinomio se suma un término que nos permitirá modificar, en cierto modo, la línea que va a describir a éste, y que recibe el nombre de *bias*. Este polinomio es conocido como función de activación (véase “La función de activación”). Permite principalmente una clasificación binaria de una base de datos. [17]

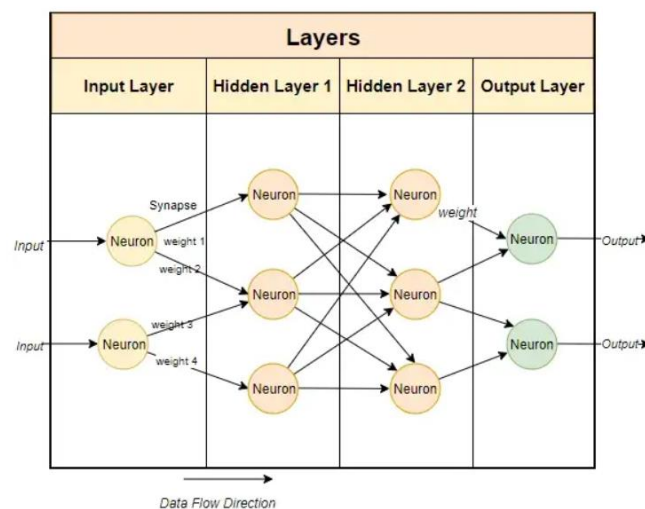


Figura 2-10 Representación de una red neuronal *perceptron*, particularmente de los *weights* asociados a las neuronas [17]

- **Recurrent neural networks:** este tipo de redes resuelven problemas utilizando secuencias de datos y son comúnmente empleados para cuestiones como la traducción de idiomas, reconocimiento de voz o análisis de videos. Ejemplos de ello pueden ser Siri, Google Translate o CAPTCHA. Al igual que el resto de redes, utilizan *training data* para aprender, con la diferencia de que ésta toma información de la experiencia adquirida en entrenamientos anteriores, a la hora de alimentar a la red neuronal. Es decir, actúan con un modelo de memoria de activación recurrente (hacia atrás). Por lo tanto, los *inputs* que recibe la capa ya no son independientes, sino que están influenciados por la activación de otras capas posteriores. Existen múltiples variantes de este tipo de red, pudiendo incluso construir una RNN bidireccional que tome datos tanto de los *inputs* como de los *outputs*. [18]

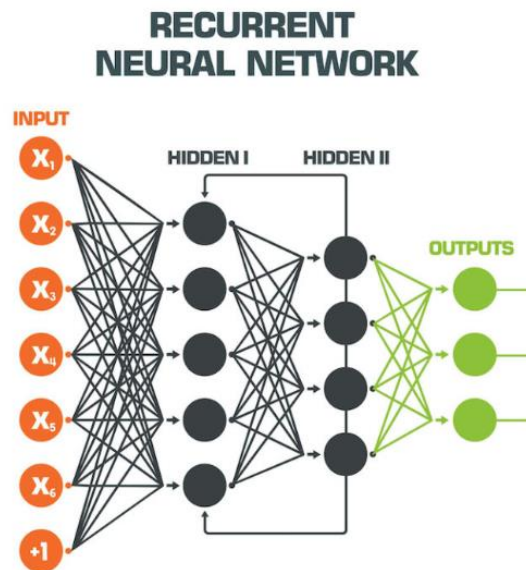


Figura 2-11 Ejemplo de red neuronal recurrente (RNN) [18]

- **Auto Encoder neural networks:** en este caso la red neuronal aprende a codificar y decodificar los datos mediante aprendizaje no supervisado [15]. El objetivo de este tipo de redes que utilizan el *encoder* es aprender a reducir la representación de una entrada, capturar los elementos más diferenciadores que le ayuden a solucionar el problema que quiere resolver, para posteriormente amplificar los datos hasta llegar a su capa de salida. La capa o capas ocultas que “comprimen” los datos de entrada se denominan *bottleneck* y son las más importantes de esta red. La mayor ventaja que ofrece esta red es que evita que el sistema memorice los datos de entrada, evitando así el *overfitting*². Por lo tanto, cuanto menor sea el *bottleneck*, menor será la capacidad de memorizar y obligará a la red a aprender por sí misma los patrones convenientes para la resolución del problema. [19]
- **Convolutional neural networks:** éstas son aquellas capaces de asignar tareas diversas a diferentes capas de la red, permitiendo multiplicar la velocidad de entrenamiento y la rapidez en la identificación de patrones de la red. Esta red será desarrollada en mayor profundidad en los siguientes subapartados.

² El *overfitting* es el exceso de modelado al que puede llegar una red cuando es entrenada en exceso, llegando incluso a modelar el ruido de ésta.

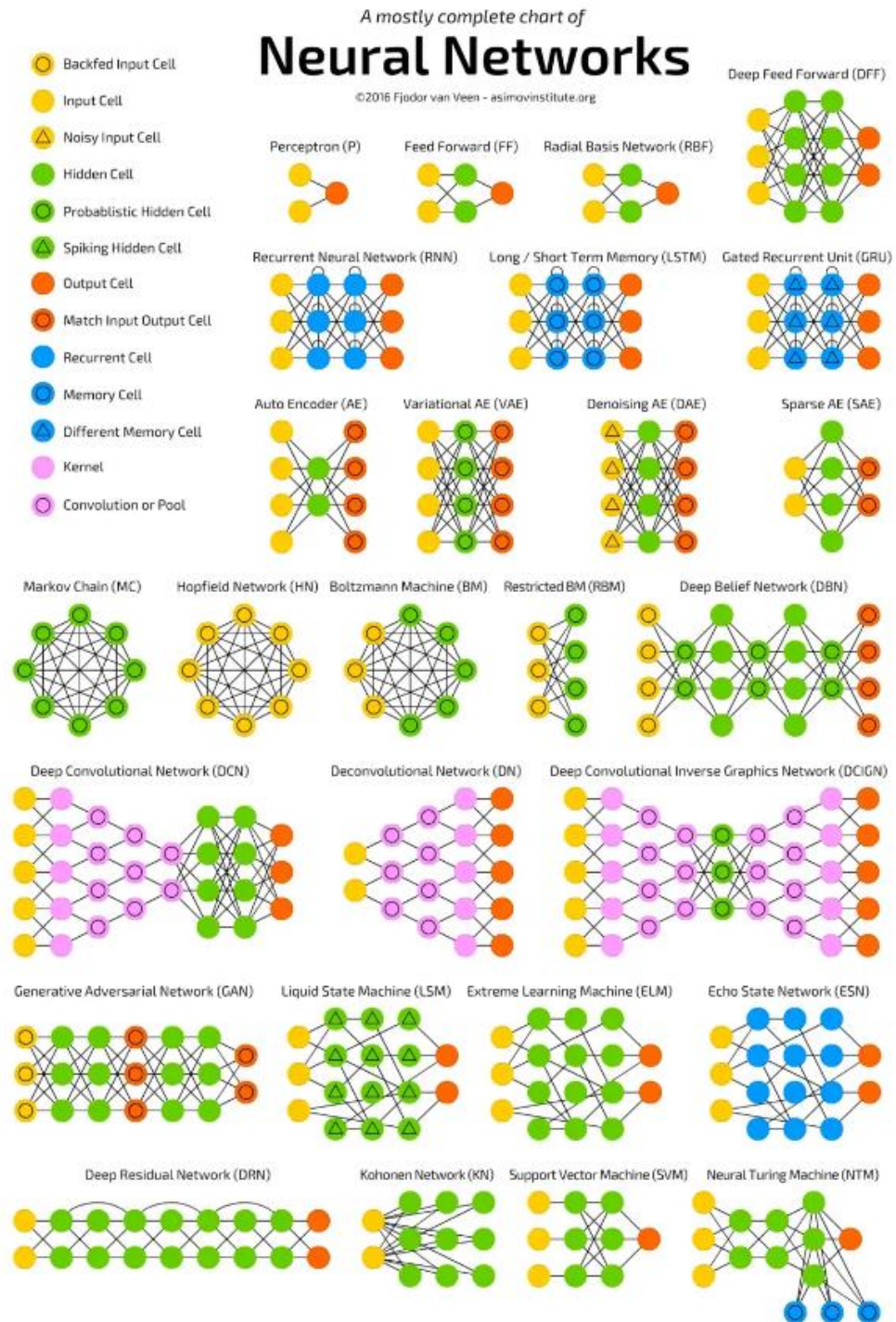


Figura 2-12 Infografía que recoge algunos de los muchos tipos de red neuronales que existen [8]

2.3.1.2 La función de activación

Como se revelaba anteriormente, la función de activación es una función no lineal que se aplica sobre la suma ponderada de una serie de pesos (*weights*) específicos de cada neurona, más el añadido de un determinado sesgo o *bias*. Si no se incluye la función de activación de carácter no lineal, los valores del procesamiento y la salida siempre van a generar un sistema basado en rectas, planos o hiperplanos, es decir, en una respuesta puramente lineal. Esto supone una limitación a la capacidad de aprendizaje del sistema, ya que existen problemas sencillos de clasificación (como la clasificación en el problema XOR) que no pueden ser resueltos mediante algebra lineal.

En otras palabras, lo que pretende una función de activación es decidir cuándo debe interactuar o no, una neurona en la capa, con el fin de descartar la linealidad de los valores de salida en la misma. A continuación, en la Figura 2-13, se adjunta un ejemplo del inconveniente que supone esta linealidad, ya que si bien la recta define, a groso modo, el comportamiento de los valores de entrada, está lejos de representar a valores que se sitúan muy alejados de ella, como por ejemplo el valor (8.0;30). [20]

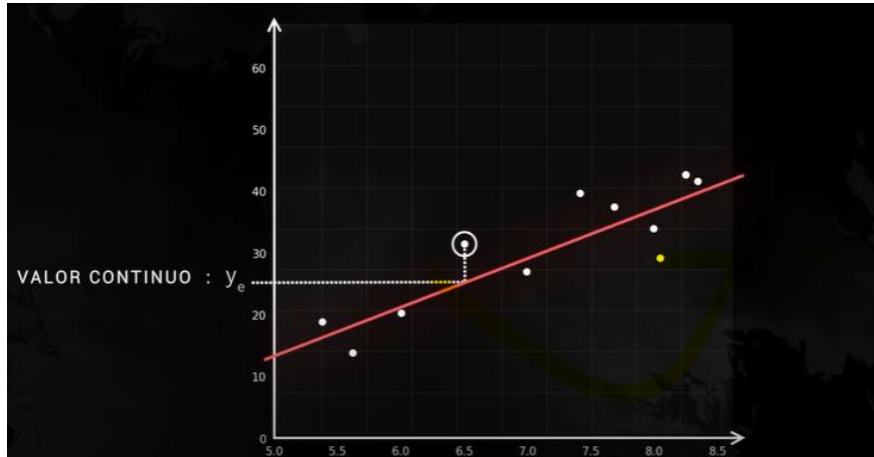
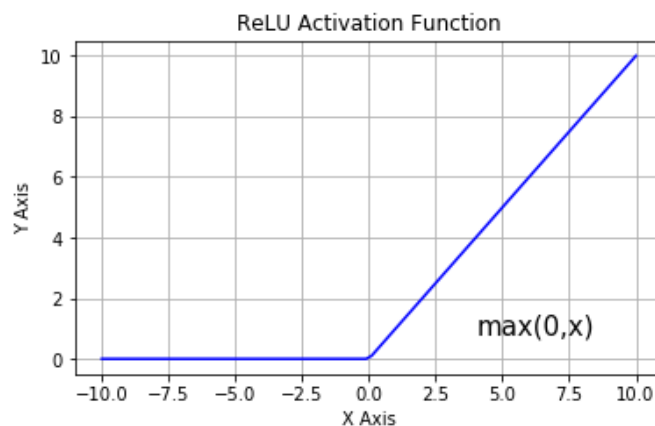


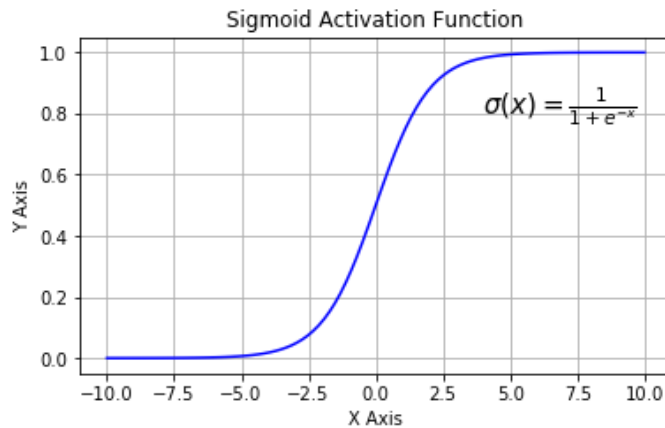
Figura 2-13 Ejemplo de regresión para una red con función de activación lineal

Es por ello, que la generalización de las respuestas que puede generar una red neuronal, sin incurrir en el denominado *overfitting* (véase nota al pie de página 2), es el mayor reto de entrenamiento al que es sometida, ya que se trata de evaluar la capacidad del sistema a generar respuestas con datos de entrada que provienen de una zona del conjunto de estudio para el cual no ha sido inicialmente entrenado. En consecuencia, existen distintos tipos de funciones de activación que facilitan esta generalización. Principalmente son cuatro:

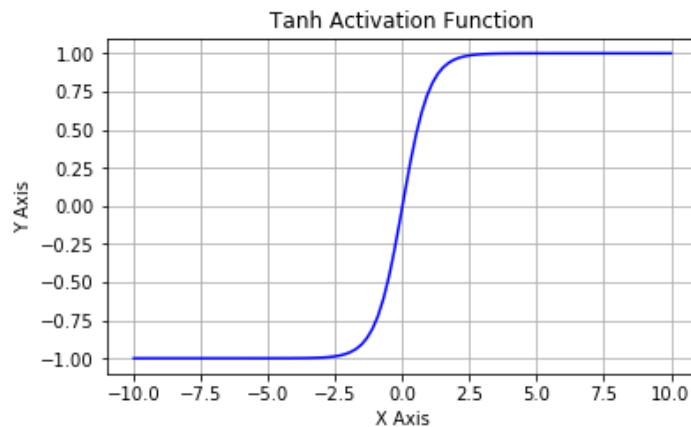
- **Escalonada o lineal:** como se ha desarrollado, genera una salida lineal en la neurona, concluyendo en una regresión que toma la forma de una recta. Es empleada en problemas que requieren agrupar los datos y reflejarlos en una regresión lineal.
- **ReLU:** también conocida como unidad lineal rectificada. Es la función de activación más empleada en capas ocultas de la red, principalmente por la sencillez de sus operaciones matemáticas. Permite anular, en cierta medida, la linealidad indeseada de la función de activación escalonada. Su funcionamiento se puede resumir en que, los valores de salida positivos, los mantiene positivos, y aquellos que son negativos, los transforma en 0.



- **Sigmoide:** esta función de activación permite describir formas curvas, siendo su rango de valores entre 0 y 1. Es de gran interés en la resolución de problemas de clasificación, ya que al ser sus datos de salida valores en el intervalo $[0,1]$, se puede introducir como última capa para determinar la probabilidad asociada a una clasificación binaria



- **Función tanh:** también conocida como función tangente hiperbólica, es similar a la función sigmoide, pero modificada matemáticamente. Su rango de valores oscila entre -1 y +1, y a diferencia de la anterior, ésta se emplea en capas ocultas de la red. Al variar sus datos de salida entre -1 y +1, su media es próxima a 0, por lo que ayudará a la red a centrar los datos, favoreciendo el aprendizaje de capas posteriores y permitiendo manejar datos con valores positivos y negativos.



2.3.2 Redes neuronales convolucionales

Expuesto el concepto de red neuronal artificial y sus distintas arquitecturas, este apartado pretende centrarse en la que se usará para la resolución de nuestro problema, que es la red neuronal convolucional (CNN). Para ello, se repasa ligeramente su historia y se desarrolla el concepto de red neuronal convolucional. Finalmente, se exponen distintos tipos de capas que podemos encontrar en su arquitectura.

2.3.2.1 Historia de las redes neuronales convolucionales (CNN)

Repasando brevemente el contexto en el que se desarrolló este modelo de red neuronal, debemos remontarnos a 1998, año en el que Yann LeCun introdujo una nueva arquitectura denominada LeNet, siendo una de las más comunes hoy en día. Sin embargo, debido a que en esa época los ordenadores no tenían gran capacidad para el procesamiento masivo de datos, como los generados por conjunto de imágenes o vídeos, no tuvieron gran éxito. [21]

No fue hasta 2010, cuando comenzó una verdadera revolución en el desarrollo de las CNN, ya que como podemos observar en la Figura 2-14, la aparición de un nuevo modelo de éstas fue prácticamente anual.

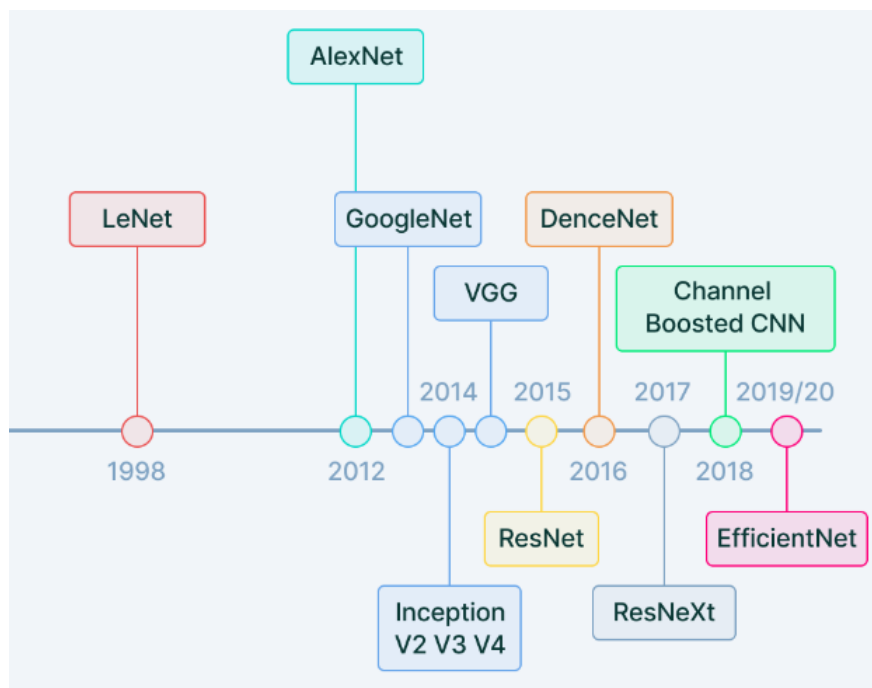


Figura 2-14 Línea temporal con los principales hitos en la historia de las CNN [21]

AlexNet aparece en 2012, siendo considerada como la “madre” de las CNN, creando además un referente para las que se desarrollan en años posteriores. Esta red neuronal estaba constituida únicamente de cinco capas convolucionales y tres capas totalmente conectadas. Fue el primer modelo de red neuronal convolucional que consiguió ser entrenada con éxito en ImageNet.³

2.3.2.2 Concepto de red neuronal convolucional

Cuando hablamos de visión por computador nos referimos a la manera en la que dotamos de “vista” a las máquinas. Las redes neuronales convolucionales (CNN), como hemos visto, son un tipo de red neuronal artificial que permite imitar mediante sus capas al ojo humano, todo esto con el fin de identificar y clasificar patrones repetitivos en datos estructurados, como puede ser la detección de objetos en imágenes. Empleando una jerarquía de capas, estas redes son capaces de especializar a las neuronas de sus capas, haciendo que se resuelvan problemas sencillos en cada una de ellas hasta llegar a un resultado mucho más complejo, que representa la respuesta deseada.

Para entender el concepto de red neuronal convolucional, es necesario desarrollar el concepto de convolución. Las convoluciones son operaciones matemáticas que permiten transformar (de forma numérica) unos datos de entrada, para obtener unos datos de salida distintos. Para ello, este tipo de redes toma un grupo de píxeles cercanos y les aplica una matriz conocida como “kernel”. Éste recorre la imagen que será sometida a las convoluciones y genera una nueva matriz de salida, que supondrá la entrada para la capa siguiente de neuronas ocultas. El conjunto de “kernels” (Figura 2-15) que le aplicaremos a nuestros datos de entrada se conocen como filtros, y nos permitirán obtener infinidad de nuevas imágenes “filtradas” que constituyen múltiples características de los datos originales. Esto ayudará, en el futuro análisis, así como en la discriminación y distinción de los objetos que le proporcionemos al sistema para que sean detectados en, por ejemplo, una imagen. [22]

³ ImageNet es una base de datos de acceso libre y gratuito, que incluye miles de imágenes puestas a disposición de cualquiera, para entrenar una red.

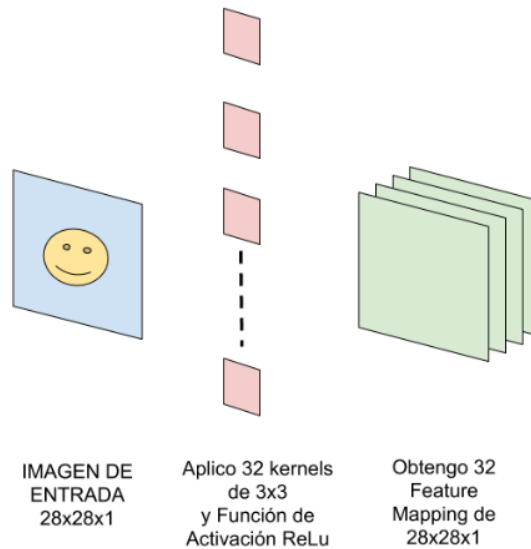


Figura 2-15 Proceso de convolución que aplica una CNN a una imagen de entrada [22]

2.3.2.3 Tipos de capas en las redes neuronales convolucionales

Aunque la principal característica de las CNN es que están compuestas de capas dónde tienen lugar una serie de convoluciones, no todas ellas son del mismo tipo. A continuación, se exponen distintos tipos de capas que contribuyen al funcionamiento de ésta, pero que no son puramente convolucionales. Algunos tipos son:

- **Convolutional layer:** en este tipo de capas, la red se limita a realizar convoluciones, es decir, operaciones matemáticas que generan un dato de salida. Como vemos en la Figura 2-16, el filtro va operando en todas las posiciones posibles de la matriz, hasta generar una nueva matriz. Este proceso se realiza en todas las capas convolucionales de la red, independientemente de que la preceda una capa del mismo tipo, o no. A la matriz resultante de todas estas operaciones realizadas por un filtro o *kernel* se le denomina *feature map* o mapa de características. Es por ello que a este tipo de capas de procesamiento basado en convoluciones, se las suele denominar como *feature extractors*.

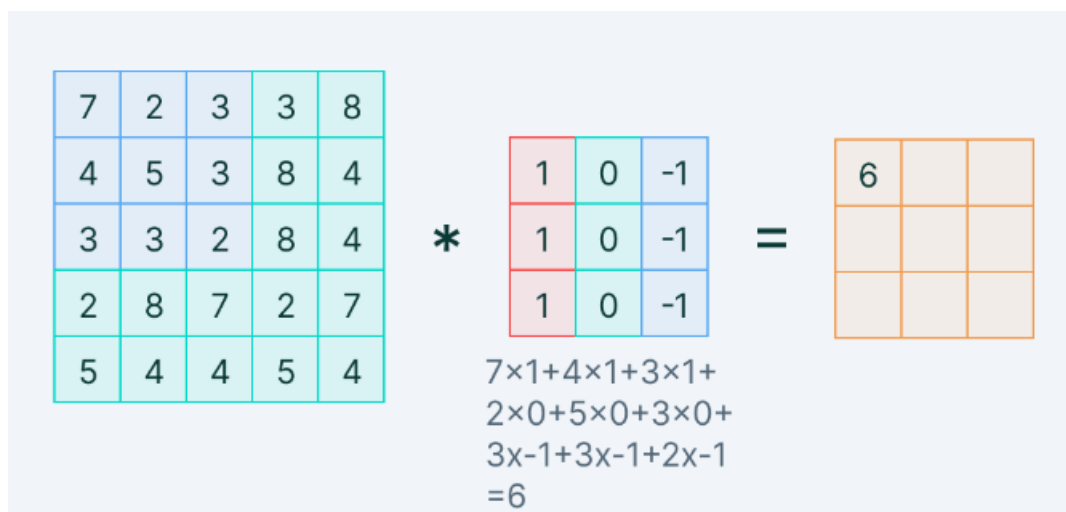


Figura 2-16 Ejemplo de operación realizada por una Convolutional layer [21]

- **ReLU layer:** en esta capa, mediante la función de activación ReLu (véase “La función de activación”), se cambia el valor de todos los números negativos por cero, previniendo así que el sumatorio pueda resultar nulo.
- **Pooling layer:** estas capas únicamente tienen el propósito de reducir el tamaño de la imagen representada, siendo colocadas entre dos capas convolucionales. Funciona haciendo pasar el filtro por cada posición posible de la matriz, y dependiendo de la finalidad que se le atribuya, tomará el mínimo, máximo o valor medio, de los valores leídos en ese instante de tiempo por el filtro. Permiten obtener versiones comprimidas de los datos de la capa anterior, en donde se ha eliminado parte del ruido presente en la imagen original.
- **Normalization layer:** como su nombre indica, este tipo de capas se dedica a normalizar el valor de salida de capas anteriores, obligando a la red a aprender con mayor independencia y evitar *overfitting* en el modelo. Se colocan entre las *convolutional layers* y las *pooling layers*, contribuyendo a la generalización de la red.
- **Fully-connected layer:** una vez procesadas las capas de convolución y las *pooling layers*, la información es “aplanada” mediante las *fully-connected layers*. Además, se encargan de devolver a la red a su forma original, haciendo que todas las neuronas de una misma capa tengan datos de entrada de toda la capa anterior. Un buen ejemplo de ello es la imagen que se muestra a continuación (Figura 2-17). [21]

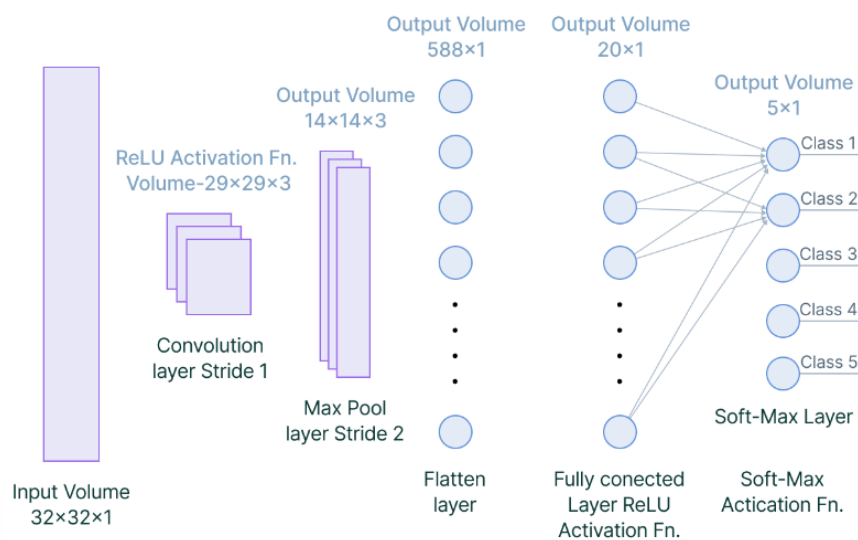


Figura 2-17 En la figura se muestran los tipos de capas mencionados anteriormente, dentro de una CNN [21]

2.3.3 Detección de objetos en imágenes

Una vez asimilado el concepto de red neuronal artificial, y específicamente el de red neuronal convolucional, se procederá a exponer una de sus muchas aplicaciones, y la más relacionada con el presente trabajo: la detección de imágenes. Este apartado se compondrá de dos subapartados, donde se expondrá un breve contexto histórico de su desarrollo, y se comentarán los principales algoritmos que se pueden emplear para esta tarea.

2.3.3.1 Introducción a la detección de objetos mediante Deep Learning

Hace menos de diez años, los problemas de detección de objetos en imágenes se resolvían con técnicas convencionales de *Machine Learning* como puede ser la búsqueda de similitudes por región, o la comparativa con histogramas de gradientes (HOG). Fue en 2013, cuando se empiezan a emplear las técnicas de *Deep Learning* para resolver este tipo de problemas, empleando recursos que utilizaban arquitecturas de redes neuronales para optimizar el proceso de detección. [23]

La aplicación de las redes neuronales para resolver este problema de la detección de objetos, ha llevado a la creación de dos formas de realizar esta detección: en dos fases (*two stages*) o una sola fase

(*one stage*). En la detección en dos fases, primero se divide la imagen de entrada en regiones de interés (*Region of Interest*), asignando varias cajas a una misma región. Tras esto, se clasifican las cajas generadas, borrando las que no son necesarias y procesando la CNN como se ha expuesto en el apartado anterior. Un ejemplo de este modo de funcionamiento fue el conocido algoritmo R-CNN.

El método que propone el *one stage* es hacer ambos procesos a la vez, desechando la división por regiones de interés, y haciendo directamente la clasificación de las cajas generadas. Uno de los ejemplos más utilizados es el algoritmo YOLO, que será analizado posteriormente.

Si bien el modo de funcionamiento puede parecer complejo, las aplicaciones de este tipo de soluciones en el mundo cotidiano son una realidad hoy en día. Entre estas aplicaciones podemos encontrar los sistemas de reconocimiento de caras, detección de peatones en un entorno urbano como muestra la Figura 2-18, video-cirugía inteligente o los vehículos autónomos.

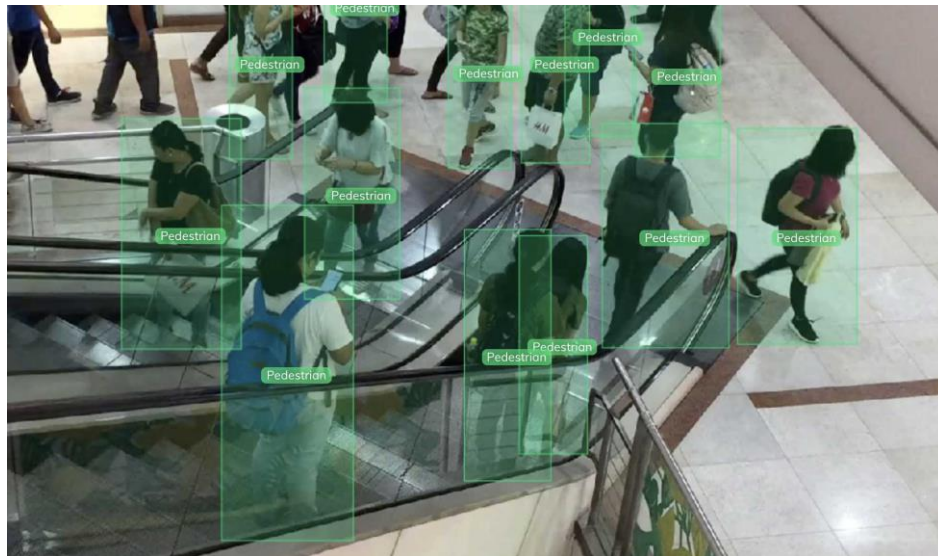


Figura 2-18 Entre las aplicaciones de la detección de objetos, podemos encontrar alguna como la detección de peatones en un entorno urbano [23]

2.3.3.2 Algoritmos empleados en la detección de objetos

Como se ha expuesto, la detección de objetos es solo una de las muchas aplicaciones que tienen las redes neuronales convolucionales. Además, como se vislumbra en el apartado anterior, este trabajo se centrará en el funcionamiento del algoritmo YOLO, por ser el elegido para el desarrollo del mismo. A pesar de ello, su empleo no nos exime de conocer otras posibilidades, en cuanto a algoritmos, que se podrían emplear para llevar a cabo la misma tarea. Algunos de ellos son: [24]

- **HOG:** desarrollado en 1986, es uno de los métodos tradicionales de detección de objetos, previo a la aparición del *Deep Learning*. Los histogramas de gradientes orientados funcionan obteniendo información útil de la imagen que procesan. De esta forma, si preprocesamos una imagen determinada obteniendo su mapa de gradientes, la red es capaz de determinar dónde se encuentran los cambios en éstos, concluyendo los bordes (o la característica particular que va a ayudar al sistema a encontrar el objeto deseado) y, por tanto, facilitando la detección. [25]
- **Fast R-CNN:** como se expuso en el apartado anterior, en la detección *two stages*, la imagen de entrada se dividía en regiones de interés que eran clasificadas a posteriori. Uno de los algoritmos que se empleaba para ello era el R-CNN (*Regions – CNN*), que ya reducía el número de regiones propuestas a 2000. Este algoritmo se limitó a acelerar los procesos de R-CNN, consiguiendo una mayor rapidez y calidad en la detección.
- **Faster R-CNN:** al igual que el anterior, el propósito de este algoritmo fue optimizar su predecesor, destacando que es el más sólido del grupo R-CNN. Éste era capaz de procesar la

imagen con una visión mucho más amplia, ya que dedicaba una red exclusiva a la propuesta de regiones de interés, mientras la red principal desarrollaba el *future map*.

- **SSD:** el algoritmo *Single Shot Detector* ya pasa a funcionar en una sola fase (*one stage*), consolidando las dos fases propias de los *two stages*. Además, es uno de los algoritmos empleados en tiempo real, que mejores resultados ofrece en cuanto a detección de objetos.
- **RetinaNet:** siendo un referente en detección *one stage*, este algoritmo es especialmente empleado en imágenes aéreas y satélites. RetinaNet nace como combinación de dos algoritmos ya existentes: *Feature Pyramid Network* (FPN) y *Focal Loss*. El primero de ellos, hace una detección variando la escala, de mayor a menor, con el objetivo de obtener la mayor información posible de la imagen. El segundo, es el elemento encargado de corregir los problemas habituales del *one stage*, en cuanto a exceso de *background-foreground*. Esto supone, en definitiva, eliminar del procesamiento todas las cajas generadas a elementos no convenientes. [26]
- **YOLO:** este algoritmo de funcionamiento *one stage*, es sin duda, el algoritmo de referencia en detección de objetos, ofreciendo al usuario múltiples posibilidades en este ámbito. A continuación, se analiza particularmente este algoritmo, y se exponen una serie de parámetros que se deben tener en cuenta a la hora de desarrollar y optimizar la red.

2.3.4 Algoritmo YOLO

Las redes neuronales convolucionales, cuando trabajan con imágenes, pueden entenderse como un sistema de procesamiento que dotan de vista a las máquinas. Esta capacidad hace que se abra todo un mundo de posibilidades en cuanto a detección de objetos. Concretamente el algoritmo YOLO (*You Only Look Once*) es capaz de hacer predicciones de “cajas detectoras” y clasificarlas a la vez, valorando las diferencias del objeto detectado respecto a algoritmos de clasificación que han sido entrenados con imágenes similares. [27]

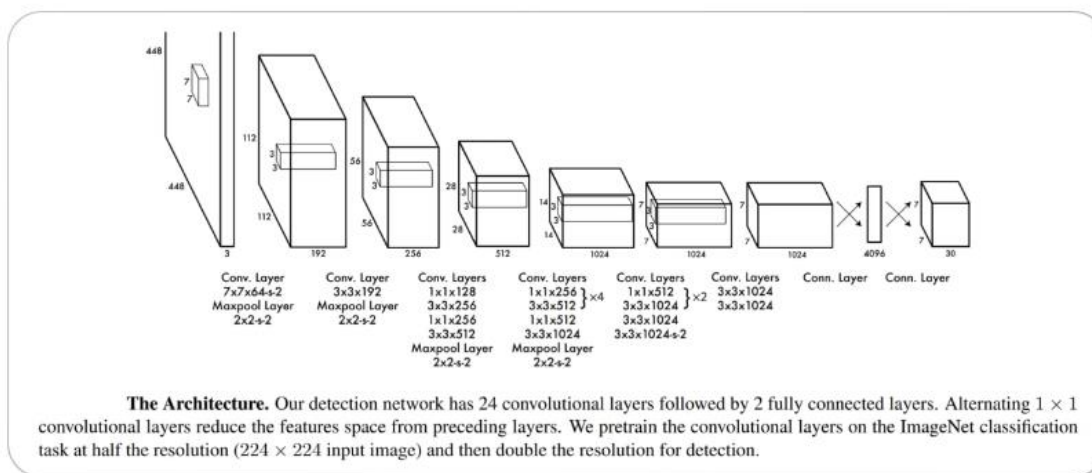


Figura 2-19 Arquitectura de una red con 24 capas convolucionales [27]

Este proceso de detección se realiza definiendo un cuadro delimitador alrededor del objeto a detectar, el cual será designado por cinco números: dos para las coordenadas de su centro, dos para las dimensiones del cuadro, y un último, para determinar la clase a la que pertenece el objeto (clasificación). Los parámetros a monitorizar durante el entrenamiento de la máquina son los siguientes: [28]

- **Intersection over Union (IoU):** representa la intersección sobre la unión de dos ventanas, midiendo el área común de solapamiento que existe entre ellas y dividiéndola entre la superficie que resulta de unir ambas superficies. Si ambas ventanas coinciden, el índice será 1, y por el contrario si no tienen nada en común, el índice será 0. Cuando hablamos de sistemas destinadas a localizar objetos, hablaremos de un índice positivo cuando sea superior a 0,5. Si no es así, será un índice negativo.

- **Precision:** este parámetro cuantifica el porcentaje de los objetos detectados por la red, es decir, es el cociente entre el número de objetos correctamente detectados con respecto al número total de objetos propuestos.
- **Recall:** este parámetro mide la relación entre el número de objetos detectados correctamente y el número total de objetos que deberían haber sido detectados.
- **Mean Average Precision:** también conocida por sus siglas como mAP, es el parámetro que nos permite medir la eficiencia de los sistemas con tareas de detección y localización de objetos. Es el resultado de la combinación de las variables anteriores a partir de la toma de resultados con múltiples condiciones en lo que respecta a una condición de valores límites que permiten establecer los niveles mínimos de similitud que deben tener los objetos detectados, como, por ejemplo, establecer un valor mínimo de *IoU* que permite eliminar detecciones que no cumplen con una similitud acordada. Estos resultados se obtienen en diferentes condiciones, y a partir de estos valores se generan los valores medios obtenidos, para finalmente obtener el mAP como una medida general del desempeño del modelo que está siendo entrenado.

2.4 Deep Learning en el ambiente marítimo

2.4.1 Datasets y tipos

Los *datasets* (o series de datos en su traducción al castellano) son conjuntos de datos tabulados (es decir, organizados en filas y columnas) generados en los sistemas de almacenamiento de datos estructurados. Para tratar volúmenes de datos como estos son necesarios sistemas capaces de ellos, ya que no es lo mismo analizar la edad de los componentes de un equipo de fútbol o la edad de los consumidores de todo el mercado online. La calidad de las bases de datos será crucial en los análisis posteriores, ya que todos los algoritmos y soluciones que definan nuestros sistemas, emanarán de la experiencia en el trato que han tenido con estos *datasets*.

Concretamente para reconocimiento de imágenes contamos con *COCO dataset*, un conjunto de datos de subtítulos, segmentación y detección de objetos a gran escala publicado por Microsoft. A continuación, se analizan los distintos tipos de *datasets* que podemos encontrar para las aplicaciones del ambiente marino: [29]

- **Archivo:** constituye un fichero independiente en el que se almacena la información con la que el sistema va a trabajar. La mayor ventaja de este tipo de *datasets* es la rapidez en el trabajo, ya que son visualizados y explotados de forma local. Por el contrario, tienen gran impedimento a la hora de conectar y escalar a otros *datasets* que no están integrados en el mismo sistema.
- **Folder:** es la suma de varios *datasets* almacenados en la misma carpeta, que además están conectados entre ellos. Entre sus restricciones encontramos el formato, debiendo compartir todos el mismo.
- **Database:** su concepto es similar al de los archivos, con la particularidad que éstos están específicamente diseñados para trabajar en un programa puntual. El archivo, si bien es uno, puede ser utilizado para distintos tipos de programas, mientras que las bases de datos no.
- **Web:** son los más accesibles y constituyen compilaciones de datos almacenados en un sitio web.

Si bien no es muy usual encontrar *datasets* particulares de entornos marítimos, existen ya aplicaciones que incorporan esta forma de tratamiento de datos para finalidades como la biología marina, seguridad marítima, batimetría, etc. A continuación, se exponen algunas de ellas: [30]

- **Vigilancia marítima:** el interés por la vigilancia marítima ha crecido sustancialmente en las últimas décadas, y supone un problema de gran importancia para garantizar la seguridad del transporte internacional y las misiones de defensa. En este aspecto, la visión por computación

puede ser una gran herramienta para solucionar algunas de las cuestiones que más afectan a ésta, como pueden ser la contaminación generada por el ser humano, derrames de combustible, detección y localización de siniestros en la mar, seguimiento de buques o detección de actividades sospechosas. Como veremos, además, no solo puede hacerse en la mar (el movimiento del buque es un gran impedimento para desarrollar sistemas que permitan hacer esta clase de actividades), sino que además se puede hacer en tierra, teniendo especial utilidad en muelles y puertos. [30]

- Vigilancia de puertos: en la actualidad, la tradicional vigilancia marítima se resume en un gran número de profesionales capaces de manejar el tráfico marítimo y velar por la seguridad en la navegación. Por ello, la visión por computación ha supuesto toda una revolución en este ámbito. Como demuestra el autor *Xingiang Chen* en su estudio “*Port ship detection in complex environments*”, el algoritmo YOLO es el mejor aliado en la detección de buques, siendo capaz de detectar y clasificar buques con un alto nivel de confianza tanto en condiciones de buena visibilidad, como con niebla, densidad de tráfico o a pequeña escala.

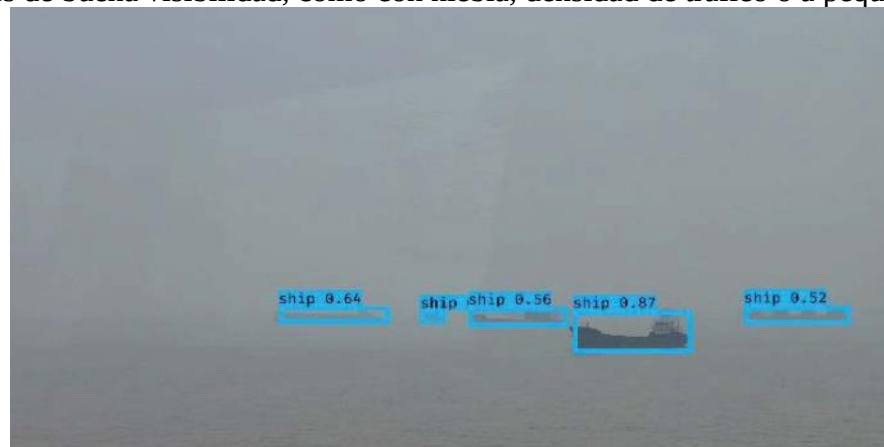


Figura 2-20 Resultados del algoritmo YOLO aplicado a la detección de buques, particularmente en situación de niebla

- Acústica submarina: otros *datasets* de gran interés en el entorno marino son los que recogen sonidos submarinos para detección de buques. Con estas grabaciones, un sistema podría aprender a clasificar buques en base al ruido que generan. Es cierto que las grabaciones también recogerán otros ruidos como el propio de la zona en la que se encuentran o ruido causante de fenómenos meteorológicos. Para ello es necesaria otra red que se dedique exclusivamente a eliminar este ruido. En este ámbito es destacable la investigación de *atlanTTic* [31], el Centro de Investigación en Tecnologías de Telecomunicación impulsado por la Universidad de Vigo, el cual tiene grandes *datasets* del tráfico marítimo de las Rías Bajas.
- Biología marina: también en el mundo subacuático podemos encontrar *datasets* que recogen la fauna marina de ecosistemas particulares. Son realmente útiles a la hora de estudiar el movimiento migratorio que hacen algunas especies, hábitos de alimentación, especies en peligro, etc. [32]

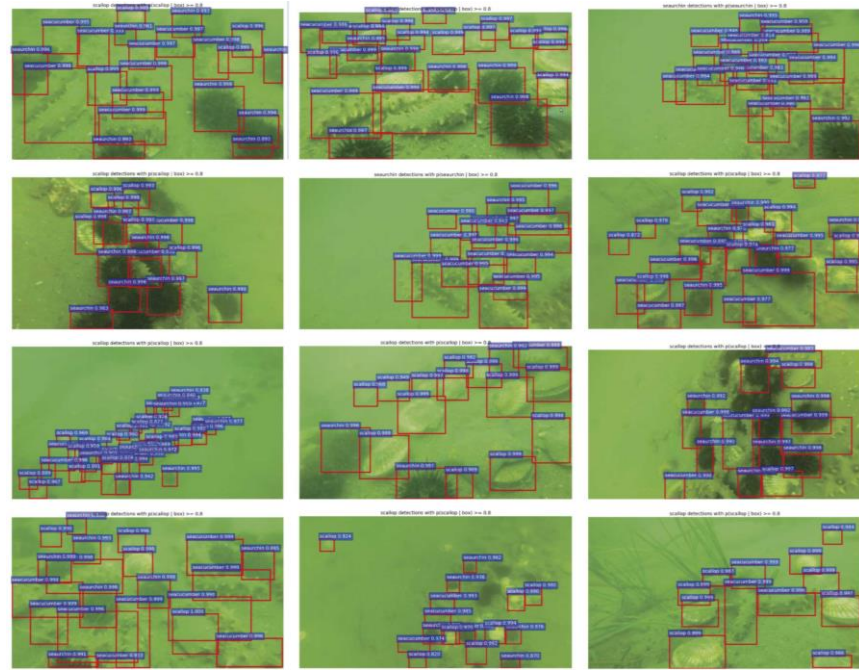


Figura 2-21 Imágenes tomadas con un robot submarino con sistema *Deep Learning* integrado [33]

2.4.2 Singapore Maritime Datasets

Este *dataset* lo abordamos por separado, por el gran interés que supone para este trabajo. En 2020, la autoridad portuaria de Singapore publicó *Singapore Maritime Datasets*. Esta compilación recoge videos realizados con cámaras 70D Canon, tanto a bordo de buques, como desde el puerto. Este *dataset* es de referencia para el mundo de las técnicas de aprendizaje profundo aplicadas en el entorno marítimo, ya que recoge imágenes con distintas condiciones (durante el amanecer, mediodía y el atardecer) y están etiquetadas, pudiendo emplearlo para el entrenamiento de nuestra red. [34]

Para entender mejor cómo funciona la detección y seguimiento de objetos (y de cara a introducir cómo funcionará nuestro sistema particular), usaremos de modelo el utilizado por *Singapore Maritime Datasets* reflejado en la siguiente figura. El proceso se divide principalmente en dos: *object detection* y *object tracking*.

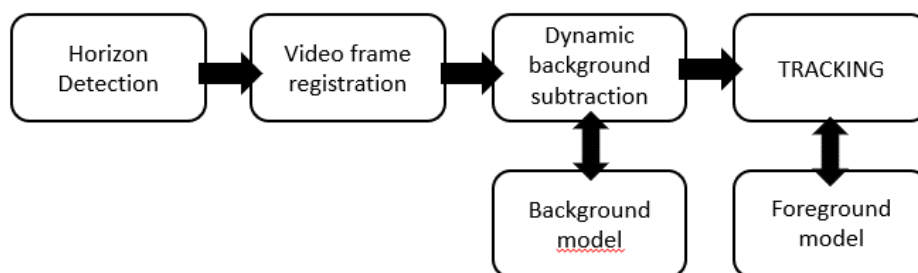


Figura 2-22 Esquema que incorpora los dos procesos realizados por *Singapore Maritime Datasets* para la detección y seguimiento de buques [34]

- *Object detection*: en esta fase, cada *frame* que constituye la grabación es considerado independiente y no tiene en cuenta ningún tipo de información temporal o relación con *frames* anteriores. Esta fase se divide en tres:
 - Detección de horizonte: teniendo en cuenta que el horizonte no siempre será nítido, el sistema usará diferentes mecanismos para resolver su detección.
 - Sustracción estática del *background*: este paso será importante de cara a eliminar elementos que afecten a nuestro plano, como pueden ser el movimiento de las olas,

- estelas, o movimientos de fauna animal, y también elementos estáticos que pueda haber en las inmediaciones de un puerto o muelle.
- Segmentación del primer plano: tras la sustracción, nuestro sistema resolverá problemas sencillos en sus distintas capas, siendo capaz de detectar formas en base a una detección primaria de líneas, curvas, ejes, contrastes, etc.
 - *Object tracking*: para resolver el problema del *tracking* esta fase consta nuevamente de varios pasos:
 - Empleo de la detección de horizonte: el principal propósito de la detección realizada en la fase anterior es permitir el registro consecutivo de *frames*, y compensar el movimiento de la cámara de uno a otro.
 - Sustracción dinámica del *background*: si en la fase anterior sustraíamos estáticamente el *background* de cada *frame*, la idea en este paso es hacerlo con mecanismos que eliminen la dinámica de este *background* para facilitar las tareas de detección de nuestro sistema y evitarle ruidos.
 - *Tracking*: en este paso, tras la sustracción de elementos estáticos y dinámicos del *background*, y tras haber detectado los objetos deseables, el sistema es capaz de realizar seguimientos a estos, como se puede ver en la Figura 2-23, acotando su espacio normalmente con un cuadro espacial o indicador.

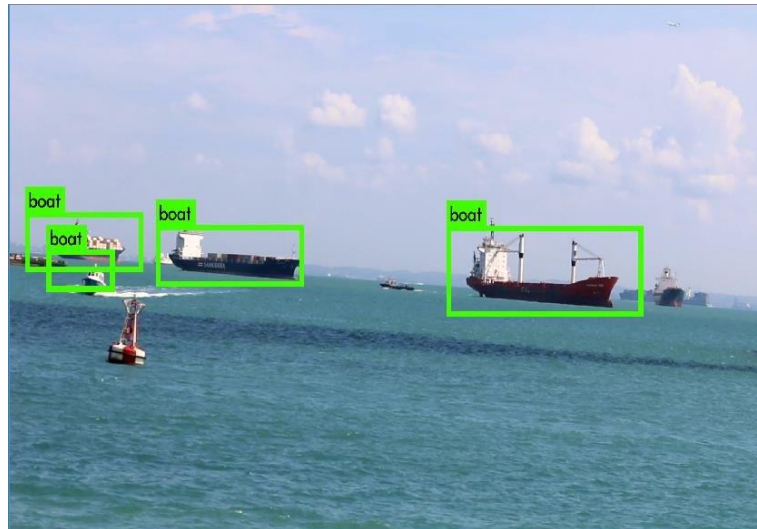


Figura 2-23 Seguimiento en video obtenido por *Singapore Maritime Dataset* [34]

2.4.3 ABOships onshore and onboard

Al igual que el anterior, se ha tomado la decisión de abordar aparte este *dataset*, debido al interés del mismo. Este *dataset* producto de *Abo Akademi University* (AAU), pone a disposición de las investigaciones una base de datos de más de 9880 imágenes, que a su vez incluyen casi 42.000 objetos anotados en ellas. Conscientes de las dificultades que presenta la detección de objetos en entornos marítimos, generadas la mayoría por las condiciones atmosféricas, cambios de fondo, o cambios de iluminación, AAU pone a servicio de la comunidad un *dataset* de gran valor formativo. En él, se clasifican hasta nueve tipos de barcos distintos, habiendo hecho un estudio previo de los que aparecen con mayor frecuencia en las grabaciones. Asimismo, se analizan los problemas que suponen un detrimento para el aprendizaje del modelo, siendo en la mayoría de los casos las condiciones atmosféricas, cambios de tamaño de los objetos, oclusión de buques, o el cambio de escala. La Figura 2-24 es un ejemplo de ellos. [35]

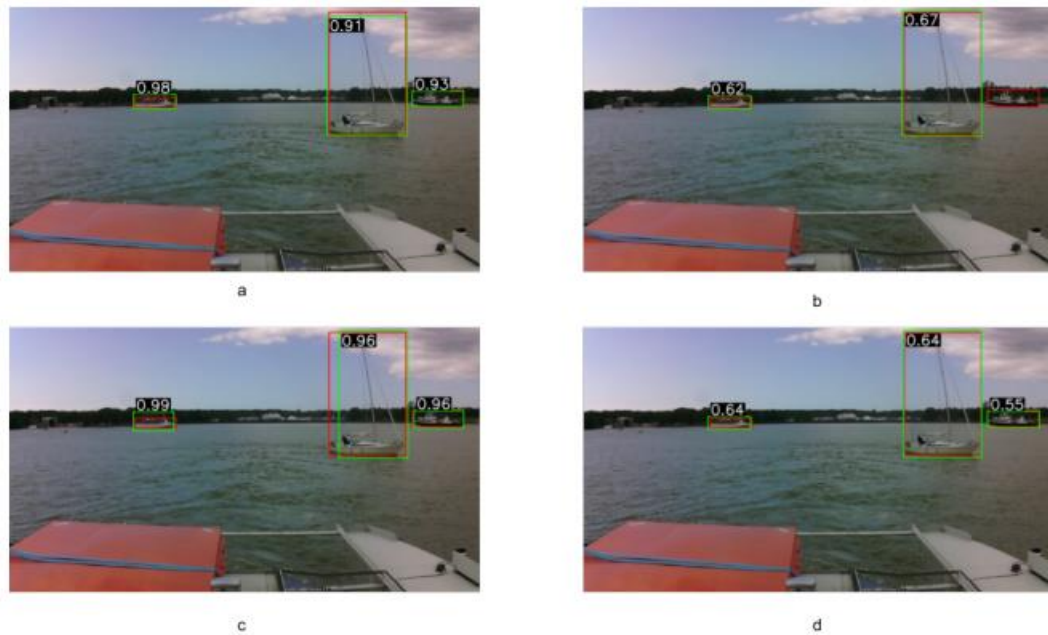


Figura 2-24 Ejemplo de una de las imágenes del *dataset* de ABOShips empleadas para la detección de buques en entornos marítimos [35]

En su documento acerca de este *dataset* “*An Inshore and Offshore Maritime Vessel Detection Dataset with Precise Annotations*”, hacen alusión al *one-stage* (véase “Introducción a la detección de objetos mediante Deep Learning”) como una de las claves en la optimización de detección de objetos.

Además, hace alusión a un término de gran importancia a la hora de entender el funcionamiento de grandes redes neuronales convolucionales, que es el de *transfer learning*. Este concepto se asocia al aprovechamiento de capas convolucionales dentro de una CNN, con el fin de emplearlas en otras redes convolucionales que tengan distintos propósitos. Poniendo un ejemplo, podemos imaginar un modelo que tenga como objetivo detectar rostros de personas en imágenes. Si tras esto, se requiere a nuestro modelo que lo que busque sean vehículos, habrá capas que no nos serán de utilidad. Por el contrario, habrá capas convolucionales que estarán muy especializadas en encontrar bordes, contrastes, o zonas curvas, y estas capas sí que serán útiles a la hora de detectar vehículos.

Uno de los mayores inconvenientes que implica el uso de las CNN, es la exigente necesidad de que los *datasets* sean de gran tamaño, ya que, de lo contrario, su experiencia de aprendizaje tiende a ser deficiente. Por ello, en el documento [35], el autor hace referencia a la incipiente necesidad de generar nuevos *datasets* que permitan nuevas experiencias de entrenamiento.

A continuación, en la Tabla 1, se expone una breve comparación entre las composiciones y objetos anotados en varios *datasets* relevantes, todos ellos especializados en detección de objetos en entornos marítimos.

Name	Total Images	Annotations	Ship Types
SeaShips	31455	40077	6
Singapore	17450	192980	6
MCShips	14709	26529	13
ABOShips	9880	41967	9

Tabla 1 Comparación del *dataset* ABOShips con otros similares en entornos marítimos [35]

2.5 Aplicaciones para la Armada del S.XXI

Como se adelanta en el primer capítulo de este TFG, el AJEMA refleja en sus Líneas Generales para la Armada [1], que la Institución no pretende alejarse de la inminente transformación digital a la que se está sometiendo la sociedad en general, y la industria en particular. Como se expone a continuación, la aplicación de técnicas de inteligencia artificial son ya una realidad en la Armada, siendo el Centro de Supervisión y Análisis de Datos de la Armada (CESADAR), y el Centro de Inteligencia Artificial de la Armada (CIA2), sus máximos exponentes en la actualidad. Además, se arrojan una serie de líneas que pueden ser objeto de desarrollo en un futuro próximo, y que supondrían importantes avances en muchas de las situaciones a las que la Armada se enfrenta cada día.

2.5.1 Estado actual de estas técnicas en la Armada

A la vanguardia de la investigación y desarrollo de estas técnicas, la Armada cuenta concretamente con dos organismos, siendo estos el CESADAR y el CIA2.

Uno de los pasos más importantes respecto a materia de inteligencia artificial, ha sido la reciente creación del mencionado Centro de Inteligencia Artificial de la Armada. Dicho centro pretende fomentar la investigación y difusión de la IA, estableciendo fuertes relaciones con la industria y centros de investigación, y participando en el desarrollo doctrinal de este campo de la ciencia [36]. Ya cuenta con algunos casos de éxito, como son el proyecto *SEDA* [37] de detección de objetivos en imágenes satélite. Cabe destacar este proyecto, ya que supone la implementación de una plataforma web de inteligencia geoespacial para el Ministerio de Defensa de España con capacidades como la detección automática de anomalías en dichas imágenes, anomalías que puedan comprometer la toma de decisiones en el ámbito de seguridad a nivel nacional. Cuenta además con otras líneas de proyecto como la clasificación de señales electromagnéticas en el espectro Radar.

Asimismo, la Armada cuenta con otro centro referente en la aplicación de IA y procesamiento masivo de datos, como es el Centro de Supervisión y Análisis de Datos de la Armada, o por sus siglas, CESADAR. Situado en el Arsenal de Cartagena, focaliza sus esfuerzos en el mantenimiento y sostenimiento predictivo [38] de los motores de las unidades de la Armada. Si bien la naturaleza de estas tareas no está intrínsecamente relacionada con la razón de ser de este trabajo, hay que comprender que la finalidad pretende ser la misma: evaluar la situación y responder ante anomalías que detecta un sistema, permitiendo la toma de decisiones de manera más ágil y sencilla.

Además, este tipo de técnicas pretenden ser empleadas en ámbitos tan particulares del mundo militar, como o es la búsqueda de minas [39]. En este aspecto, la inteligencia artificial permite desarrollar modelos que procesan grandes volúmenes de datos a gran velocidad, contemplando cambios de entorno y acciones del adversario en el campo de batalla. Una de las mayores ventajas de esta aplicación en un entorno tan específico como este, es reducir la presencia del ser humano en zona hostil, reduciendo así los riesgos al que se exponen.

Como se aprecia, el interés y desarrollo de la IA en la Institución están a la orden del día, y los resultados auguran que esta será la tendencia durante los próximos años.

2.5.2 Líneas de desarrollo de técnicas *Deep Learning* en la Armada

Además de lo expuesto en el apartado anterior, existen otras líneas de desarrollo a explotar en la Armada, respecto a la aplicación de técnicas de aprendizaje profundo. Este apartado pretende proponer algunas de esas aplicaciones, y motivar las razones que las convierten en líneas de interés para el crecimiento de la Institución. Estas aplicaciones son las siguientes:

- Vigilancia marítima en bases y estaciones navales: como ya se expuso en el análisis del estado del arte sobre los tipos de *datasets*, estos pueden ser empleados en tareas de vigilancia marítima. Especialmente en la Armada, una aplicación puede ser la instalación de cámaras de vigilancia que ofrezcan seguridad en los accesos a bases y estaciones navales, detectando

y *trackeando* los buques que transitan, y haciendo hincapié en aquellos que el sistema no reconoce como habituales.

- Incorporación de *displays* que ayuden al conocimiento de la situación de superficie: Una de las situaciones que requiere por igual a todos los buques de la Armada, es la navegación por aguas altamente transitadas. La posibilidad de incorporar estas técnicas a *displays* integrados en los cristales del puente, facilitarían en gran medida el conocimiento de la situación de superficie por parte del OGP, que tendría una concepción rápida del entorno sobre los 180° de visión que tiene por delante. La Figura 2-25 es un ejemplo de esta aplicación.



Figura 2-25 Ejemplo de aplicación de técnicas de detección y seguimiento de buques, en los cristales del puente de un buque de la Armada (fuente: www.armada.defensa.gob.es)

- Integración en sistemas de armas: otra aplicación que puede ser de gran utilidad, es la integración en sistemas de armas, como puede ser la estación remota de armas de la empresa Escribano [40]. De esta forma, el sistema facilitaría la toma de decisiones al operador, ofreciéndole una gran ayuda en la comprensión de la situación de superficie, en operaciones contra la piratería o situaciones de amenaza asimétrica ⁴. Un ejemplo de ello es la Figura 2-26.

⁴ El concepto de amenaza asimétrica hace referencia al vector definido como el empleo de medios diferentes, para evitar o negar las capacidades del oponente mediante la explotación de sus vulnerabilidades, obteniendo resultados desproporcionados. [52]



Figura 2-26 Ejemplo de integración de sistemas de armas para proporcionar la máxima ayuda en la decisión del operador (fuente: periódico El Español)

3 DESARROLLO DEL TFG

En este capítulo se desarrollan los pasos llevados a cabo hasta la culminación de este trabajo. Como se expone a continuación, ha sido necesario un periodo inicial de formación en la materia, debido a la naturaleza eminentemente contemporánea de los conceptos que se tratan en él. Por otro lado, se presentan las herramientas de *software* seleccionadas en las distintas fases del mismo, así como los problemas a los que es necesario enfrentarse en cada tarea y la justificación del algoritmo elegido. Posteriormente, se hace alusión al *dataset* que hemos denominado *Webcam de Cangas*, el cual ha sido empleado como una primera introducción a la detección de objetos en entornos marinos, a fin de entender los conceptos básicos asociados a ese problema, y así poder analizar y estimar los resultados que se esperan en las pruebas finales con el *dataset* definitivo. Por último, se detalla el proceso de etiquetado y entrenamiento al que es sometido nuestro modelo, una vez que se obtienen las grabaciones de las lanchas de instrucción y embarcaciones de Infantería de Marina en la Escuela Naval Militar.

Así, el objetivo de este desarrollo es la elaboración de un *dataset* y de una solución de seguimiento de objetos en video, grabados a partir de dispositivos portátiles, es decir, se trata de un trabajo en el que se pretende generar el ciclo completo asociado a una solución que emplea *Deep Learning*. De esta forma, el presente TFG no se centrará en el aprendizaje de los métodos de programación y creación de soluciones desde cero a partir de las plataformas existentes (*tensorflow*, *pytorch*, ...), sino que se ha optado por emplear las ventajas que ofrecen las técnicas de *transfer learning* para generar modelos operativos, con respecto al problema a resolver, empleando soluciones previas que hayan sido creadas y entrenadas previamente en bases de datos de imágenes genéricas (como *COCO dataset*), específicamente aquellas que hacen uso del algoritmo YOLO.

3.1 Fase de formación

Como se menciona en la introducción de dicho capítulo, previo procesamiento de los datos, ha sido necesario un periodo de formación en los conceptos empleados en este trabajo. La principal razón que ha motivado esta profundización es la relevancia actual que caracteriza a las técnicas de aprendizaje profundo, estando en un constante proceso de cambio. A continuación, se repasa cómo ha sido ese proceso de aprendizaje, tratando de vislumbrar cómo se especializa el presente trabajo, partiendo de la generalidad de IA, y finalizando con las particularidades de la tarea en la que se centra: la detección y seguimiento de objetos en imágenes.



Figura 3-1 Línea secuencial que acota los campos desarrollados en el presente trabajo (autoría propia)

Para llegar a los conceptos más particulares de nuestro trabajo, ha sido necesario empezar desde los más generales. En primer lugar, se ha hecho una investigación profunda sobre la inteligencia artificial y sus posibilidades, entre las que destacan las técnicas de *Machine Learning*. Con ello, se estudió las posibilidades que ofrecen estas técnicas, ya que, en definitiva, es enseñar a las máquinas a que aprendan a realizar tareas que cada vez resultan más complejas, lo cual implica la necesidad de que manejen conceptos cada vez más abstractos. Específicamente, se ha desarrollado la técnica de aprendizaje *Deep Learning*, que permiten dividir dicho aprendizaje en múltiples capas, construyendo un modelo de aprendizaje jerarquizado.

Prosiguiendo con la formación, fue necesario explorar detalladamente el problema de detección de objetos en imágenes. En la infinidad de problemas posibles que se pueden afrontar con técnicas de aprendizaje profundo, el seleccionado se encuadra en la visión por computador. Dado el problema que concierne a este trabajo, se ha optado por resolverlo con redes neuronales, de entre todas las opciones existentes. En esta fase de formación, se comprueba que se puede resolver mediante este medio, ya que permite “enseñar” a un sistema a detectar patrones de comportamiento en las características obtenidas para un conjunto de datos proporcionado.

El concepto de red neuronal artificial (ANN) está en continuo desarrollo, y esto hace que aparezcan constantemente nuevas arquitecturas que pretenden optimizar las ya existentes. Una de ellas, son las redes neuronales convolucionales (CNN), que aplican convoluciones (operaciones matemáticas) a los datos que son transferidos entre las múltiples capas ocultas que constituyen la red, consiguiendo extraer muchísima más información de las imágenes de entrada de la que extraen las ANN convencionales (véase “Redes neuronales convolucionales”). Las CNN son las elegidas para resolver la tarea del presente trabajo.

Posteriormente, asimilados los conceptos de visión por computador, detección de objetos en imágenes, y los conceptos de redes neuronales y convolucionales, se hace un estudio exhaustivo de los posibles algoritmos que facilitan la resolución de estas tareas. Por su versatilidad y fácil manejo, se elige YOLO (véase “Algoritmo YOLO”) como el más idóneo.

Finalmente, se afronta una etapa en la que se lleva a cabo un estudio de las posibilidades de los algoritmos de seguimiento, así como de la propia tarea de *tracking*. Además, se analizan sus deficiencias y vicisitudes generadas, y se procede a la optimización del modelo, empleando un entorno de trabajo distinto al utilizado en la detección de objetos en imágenes.

3.2 Dispositivos de grabación portátiles empleados

En el apartado dedicado a exponer los “Objetivos y alcance” del presente TFG, se aclaraba que la particularidad del mismo reside en hacerlo con dispositivos de grabación portátiles. Por ello, concretamente, se centrará en hacerlo con dos medios al alcance de la gran mayoría de personas: un teléfono móvil y una cámara fotográfica.

3.2.1 Teléfono móvil con objetivo portátil

Uno de los dispositivos con el que se tomarán las imágenes del *dataset* de la Escuela Naval Militar, es un teléfono móvil, un iPhone 13 en este caso. Debido a la distancia a la que se tendrán que tomar las imágenes, y con el objetivo de aumentar las prestaciones de nuestro sistema, se le acoplará un teleobjetivo portátil de la marca Selvim, como el mostrado en la Figura 3-2. Este acople permite disponer de un aumento de 22 veces el original, instalándose en el centro de la lente mediante una pinza. Además, cuenta con un trípode que garantiza la estabilidad durante la grabación de las muestras.



Figura 3-2 Teleobjetivo portátil para smartphone de la marca Selvim

3.2.2 Cámara fotográfica con capacidad de zoom óptico

El otro dispositivo con el que se obtienen las imágenes es una cámara digital de la marca Kodak, concretamente el modelo Pixpro AZ422. La principal diferencia con el dispositivo anterior, es que en este caso el zoom es óptico, sin necesidad de instalar un objetivo. Entre sus prestaciones encontramos la capacidad de grabar en HD a 30 fps, una calidad de 20 megapíxeles, y su zoom óptico permite hacer hasta 42 aumentos. Al igual que el teléfono móvil, es un dispositivo de gran accesibilidad para cualquier usuario, y la mayor ventaja que presentan ambos es su naturaleza portátil.



Figura 3-3 Cámara digital Kodak Pixpro AZ422 empleada para la obtención de imágenes

3.3 Entorno de trabajo y *softwares* empleados

Como se expone en apartados posteriores, este trabajo requiere de varias etapas de etiquetado de imágenes, así como varias fases de entrenamiento de la red y generación de modelos. Es por ello, que se hace evidente la necesidad de emplear herramientas de *software* que posibiliten la resolución de estas tareas. A continuación, se presentan brevemente las herramientas seleccionadas y algunas de las posibilidades que ofrecen:

- Label-studio: en términos de etiquetado, se ha optado por el programa Label-studio. Esta herramienta permite etiquetar datos multimedia, facilitando la importación de etiquetas diseñadas en lenguaje Python. Además de contar con una interfaz intuitiva para el desempeño de esta tarea, permite exportar el *dataset* etiquetado en distintos formatos, siendo YOLO de especial interés para el presente trabajo. [25]
- Ultralytics HUB: este programa, encontrándose aún en fase beta, permite importar *datasets* etiquetados y entrenarlos, obteniendo gráficas de gran interés de cara a la interpretación paramétrica del modelo. Hay que puntualizar que solo permite entrenar la detección de

objetos en YOLOv5 y sus variantes, por ser el creador del *software*, el mismo que generó este algoritmo. Al igual que el programa mencionado anteriormente, presenta una interfaz considerablemente intuitiva, y facilita al usuario (independientemente de su experiencia) el entrenamiento de su modelo, así como la representación de los resultados [41]. En la se presentan todos los tipos de modelo con los que permite entrenar esta herramienta *sotware*.

	YOLOv5x6 YOLOv5x6	 Public	COCO2017	mAP 55.8	size 270.0 MB
	YOLOv5m6 YOLOv5m6	 Public	COCO2017	mAP 51.3	size 69.0 MB
	YOLOv5n YOLOv5n	 Public	COCO2017	mAP 28.0	size 3.9 MB
	YOLOv5s YOLOv5s	 Public	COCO2017	mAP 37.4	size 14.1 MB
	YOLOv5m YOLOv5m	 Public	COCO2017	mAP 45.4	size 40.8 MB
	YOLOv5l YOLOv5l	 Public	COCO2017	mAP 49.0	size 89.3 MB
	YOLOv5x YOLOv5x	 Public	COCO2017	mAP 50.7	size 166.0 MB
	YOLOv5n6 YOLOv5n6	 Public	COCO2017	mAP 36.0	size 6.9 MB
	YOLOv5s6 YOLOv5s6	 Public	COCO2017	mAP 44.8	size 24.8 MB
	YOLOv5l6 YOLOv5l6	 Public	COCO2017	mAP 53.7	size 147.0 MB

Figura 3-4 Modelos con los que permite entrenar la herramienta *software* Ultralytics HUB [41]

- VSCodium: puede ser clasificado como un repositorio de *scripts*. Pese a ser prácticamente idéntico a Visual Code, éste no pertenece a Microsoft y es un editor de código *open source*, permitiendo ser instalado en cualquier sistema operativo. [42]
- Google Colab: siendo totalmente gratuito, y accesible a cualquier usuario que disponga de conexión a Internet, este *software* se ha posicionado como uno de los referentes en términos de investigación y aplicación de técnicas de aprendizaje profundo. Destaca por permitir el desarrollo de programas de *Python*, asimilando y ejecutando códigos, simplemente empleando un navegador web.

3.4 Problemas a resolver

Hasta el momento, se ha introducido el desarrollo del TFG con la fase inicial de formación afrontada, además de presentar los dispositivos y entornos de trabajo involucrados en el mismo. También se han acotado los márgenes en los que se mueve el presente trabajo, aclarando que lo hace en la tarea de visión por computador, específicamente empleando técnicas *Deep Learning*. Una vez establecidos estos elementos, este apartado se propone introducir los dos problemas principales a los que se enfrenta el desarrollo de los objetivos de este TFG, aunque sin profundizar en detalles. Las particularidades y

pesquisas propias de ambos problemas se irán revelando a medida que avanza el desarrollo de esta presentación del TFG. De esta forma, los principales problemas a resolver son dos:

- Detección y clasificación de objetos en imágenes: en esta fase, el modelo hace frente al problema de detectar objetos en imágenes. Para ello, el modelo habrá aprendido previamente qué clase de objetos tiene que detectar y clasificar, y particularmente en este trabajo, serán barcos y objetos propios del entorno marítimo.
- Seguimiento de objetos en video: una vez detectados los objetos en imágenes, el modelo aprende a realizarles *tracking* en video. Como se expone a continuación, los algoritmos que hacen posible acometer esta tarea, lo hacen atribuyéndoles una identidad a esos objetos que han sido detectados en cada *frame* o fragmento de video. De esta forma, el modelo aprende a correlacionar las detecciones, y permite mantener el seguimiento del objeto durante toda su aparición en una grabación.

3.5 Justificación de los algoritmos empleados

Expuestos los problemas que afronta el presente TFG de forma breve, se realiza un estudio de los algoritmos que permitirán obtener mayores probabilidades de éxito en el desarrollo de este trabajo. A continuación, se desarrolla la justificación de aquellos que han sido empleados en ambas tareas.

3.5.1 Algoritmos para detección de objetos

En el apartado del segundo capítulo se hace un estudio somero sobre las capacidades del algoritmo YOLO. La primera ventaja que se resalta del mismo es su procesamiento *one-stage* de las imágenes de entrada, fusionando en cierto modo la fase de detección de regiones de interés, junto con la predicción sobre la localización de los objetos. Además, tal y como se expuso antes, su arquitectura basada en capas convolucionales facilita el proceso de *transfer learning*, es decir, la capacidad de usar redes cuyas capas han sido pre-entrenadas en tareas similares, lo que aumentaba considerablemente su capacidad para ser empleadas en resolver tareas especializadas relacionadas con la detección de objetos en imágenes. Finalmente, se atribuía parte del éxito a la particularidad de *Non-Maximum Suppression* (NMS), el cual permite incrementar la precisión y eficiencia en la resolución de la tarea, simplemente estableciendo umbrales que descartan las *bounding boxes* generadas que resultan ser más redundantes.

Recorriendo su historia, se observó que estaba en continuo cambio, y es por ello que actualmente se encuentra en su versión YOLO v8. A continuación, se repasan las dos versiones con las que se trabaja en este TFG, que son YOLO v5, y el mencionado YOLO v8. [43]

- YOLOv5: su éxito radica en la facilidad de su construcción, su facilidad de uso, y los muchos documentos y validaciones que existen sobre él. La estructura que emplea es la que conocemos como *EfficientDet*, y permite alcanzar grandes precisiones y generalización para un número considerable de categorías [44]. Una diferencia a tener en cuenta respecto a su versión original, es que ha sido entrenado con un *dataset* que incluye 600 categorías distintas, mientras que YOLO fue entrenado solamente con 20. Además de esto, introduce un concepto denominado *spatial pyramid pooling*, el cual mejora exponencialmente la detección de objetos pequeños. Esto se consigue reduciendo la resolución espacial de los mencionados *feature maps*, es decir, la imagen que se obtiene al someter una imagen de entrada a una capa convolucional. Por último, una de las mayores ventajas que ofrece la elección de esta versión, es que es producto de Ultralytics HUB, lo que nos permite trabajar con él y entrenar nuestros modelos en un entorno web tan intuitivo como este.
- YOLOv8: es la versión más reciente de YOLO. Sin repetir las características de este algoritmo que han sido mencionadas anteriormente, cabe destacar que ha sido dotado con técnicas que le permiten una gran flexibilidad, haciendo posible su uso para modelos que han sido entrenados en versiones anteriores. Además, contiene un rango mayor de dispositivos CPU y GPU compatibles con su procesamiento, lo que lo hace más accesible

para usuarios con prestaciones computacionales menores. Nuevamente, una de sus mayores ventajas es la posibilidad de entrenarlo en el entorno web Ultralytics HUB, lo que proporcionará gráficas y resultados fácilmente interpretables, como se demuestra posteriormente. [45]

A continuación, en la Tabla 2 se muestra una comparativa entre las capacidades de ambas versiones, relativas a la detección de objetos.

Model Size	YOLOv5	YOLOv8	Difference
Nano	27.6	36.7	+32.97%
Small	37.6	44.6	+18.62%
Medium	45	49.9	+10.89%
Large	49	52.3	+6.73%
Extra Large	50.7	53.4	+5.33%

Tabla 2 Comparativa entre las capacidades de ambas versiones de YOLO, en términos de detección de objetos [46]

3.5.2 Algoritmos para seguimiento de objetos en video

Expuestos los algoritmos que permiten resolver los problemas de detección de objetos en el presente TFG, se presentan los dos algoritmos de los que se hará uso para resolver el problema de seguimiento de objetos en video. Cabe destacar, que ambos algoritmos ya se encuentran integrados en el *framework* de trabajo en YOLO en su versión 8. La mayor ventaja de ambos es su alto grado de interoperabilidad con YOLO v8, lo que permite una ejecución fácil y rápida, con un nivel alto de confianza en las detecciones. Estos algoritmos son los siguientes:

- *ByteTrack*: es uno de los algoritmos de *tracking* que será empleado en el presente TFG. Permite resolver problemas de seguimiento de varios objetos de forma simultánea, y lo hace identificando los objetos y atribuyéndoles una identidad en los videos. Para ello, realiza la detección de objetos en cada *frame*, correlacionando a posteriori las *bounding boxes* detectadas en cada uno, y atribuyendo una identidad o marca que permita asociarlo al mismo. Concretamente, *ByteTrack* permite realizar esta tarea mediante lo que se conoce como *IoU Tracker*, es decir, calcula los *IoU* de las *bounding boxes* de un frame, y los compara con los determinados en *frames* anteriores [47]. Uno de los problemas que presenta este algoritmo, es la pérdida de *bounding boxes* cuando se dan situaciones de oclusión. Para resolver esto, el algoritmo asocia un nivel de confianza a la detección que permite mantener el objeto detectado, si su nivel de confianza ha sido determinado como “alto” en *frames* anteriores. [48]
- *BoT-SORT*: es otro algoritmo que emplearemos en la tarea de *Multi-object Tracking* [49]. A diferencia de *ByteTrack*, este algoritmo combina la técnica *IoU Tracker* con otra conocida como *Re-identification Tracker (ReID)*. Esta segunda permite identificar los objetos a través de la estimación de su posición futura, y no comparando los *IoU* con otros detectados anteriormente con un nivel “alto” de confianza, como hace *ByteTrack*. Las características principales en las que se base el éxito de este algoritmo, son entre otras, la *Camera Motion Compensation* (compensación del movimiento de la cámara, en su traducción al castellano) y la fusión del sistema *ReID*, con elementos del sistema *IoU Tracker*. El motivo de esto es paliar las deficiencias del algoritmo cuando se presentan situaciones de abarrotamiento en la imagen, situaciones en las que recurre a comparar las estimaciones de posiciones futuras, propias de *ReID*, con detecciones previas clasificadas con un nivel “alto” de confianza, propias de *IoU Tracker*. [50]

3.6 Prueba inicial

Pasando a la parte eminentemente práctica del presente TFG, los primeros pasos en el entrenamiento de nuestro modelo se realizan con un *dataset* considerado de aprendizaje, compuesto por imágenes del puerto de Cangas de Morrazo, situado en la provincia de Pontevedra (Galicia, España). El objetivo principal de esta fase es reproducir los contratiempos y vicisitudes que nos presentará posteriormente el *dataset* seleccionado para la ejecución final de nuestro modelo, tanto en la fase de etiquetado, como en la de entrenamiento.

3.6.1 Etiquetado de imágenes de la Webcam de Cangas

Antes de comenzar a entrenar una red, es necesario determinar cuál va a ser el conjunto de datos que se le va a proporcionar. Como se revela anteriormente, esta prueba inicial será llevada a cabo con un *dataset* de menor tamaño compuesto por 300 imágenes, que nos permitirá obtener unas primeras impresiones. Dicho proceso de etiquetado se ha realizado junto a D. José Álvarez San Martín, autor del TFG “Aplicación de técnicas de aprendizaje profundo a la localización y seguimiento de objetos flotantes empleando drones con cámara”. Haciendo uso de esta experiencia acumulada, se pondrá a prueba el modelo con un *dataset* más particular, como se expone posteriormente. Por ello, la primera tarea que se acomete en este trabajo es el etiquetado del *dataset* empleado en la prueba inicial. Las particularidades de esta base de datos son las siguientes:

- Se trata de un entorno marítimo, donde se aprecia infraestructura propia de un puerto, así como faros, espigones, alumbrado y zona litoral.
- La posición de la toma de imágenes se mantiene fija, es decir, no hay variaciones ni de orientación, elevación o profundidad.
- El medio para la obtención de estas imágenes es una cámara de vigilancia.
- Las imágenes son tomadas durante todas las horas de un día encuadradas entre el orto y el ocaso, por lo que las condiciones meteorológicas y de luminosidad pueden variar de una imagen a otra.

Con el fin de obtener una mayor discriminación, se han dividido las etiquetas de las clases existentes, entre varios tamaños posibles. De esta forma, no consideraremos igual un *sailing_boat*, entendido como un velero de menor porte, que un *sailing_ship*, entendido como un velero de porte considerable. Esto dificulta el entrenamiento del modelo, pero asegura obtener una mayor generalización, y, por tanto, una mayor discriminación a la hora de clasificar los objetos detectados. Para este *dataset* se han diseñado unas etiquetas particulares al entorno en que han sido generadas las imágenes, ya que el tráfico marítimo de la zona se remite a mercantes, veleros, pesqueros, barcos de pasajeros o embarcaciones rápidas. Las etiquetas empleadas han sido las siguientes:

- *fish_boat*.
- *speed_boat*.
- *sailing_boat*.
- *fish_ship*.
- *passenger_ship*.
- *sailing_ship*.
- *passenger_vessel*.
- *merchant_vessel*.
- *jet_ski*.
- *surf*.
- *person*.

A pesar de todas las consideraciones y los esfuerzos realizados en el etiquetado, existen algunos aspectos a tener en cuenta a la hora de analizar los resultados. Estos resultados no dependerán exclusivamente de la variedad de etiquetas que hayamos diseñado, ni del nivel de ajuste de las cajas de

clasificación que hayamos podido alcanzar. Hay que tener otros detalles en cuenta, y que impiden en gran medida que nuestro modelo sea perfecto. Éstos son la resolución, tamaño y número de píxeles, contraste, brillo de la imagen, disparidad entre elementos de una misma clase, etc. En la Figura 3-5 se muestra un ejemplo de imagen con el etiquetado aplicado.

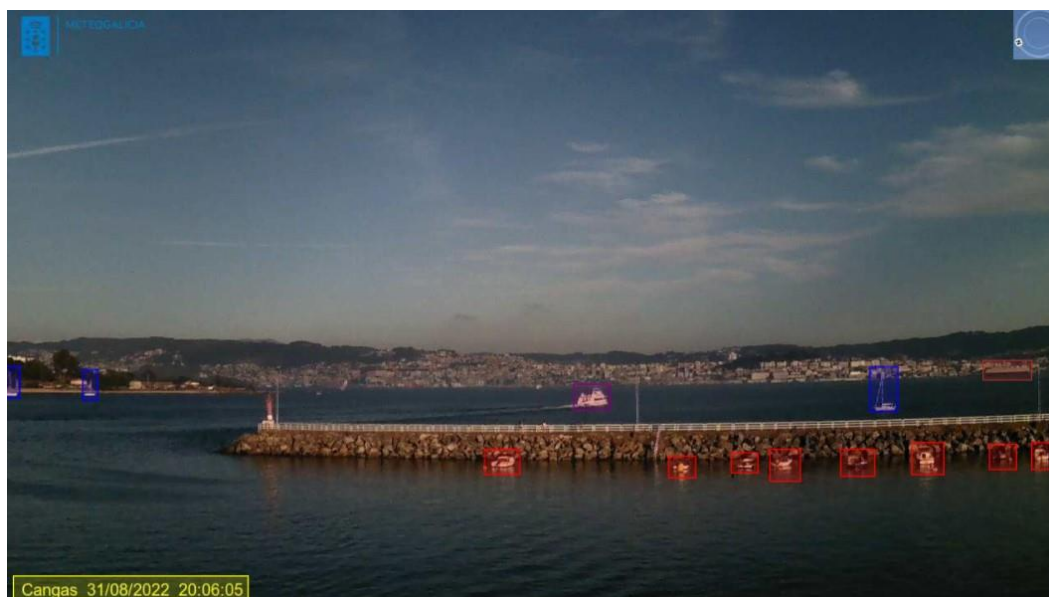


Figura 3-5 Imagen ejemplo de un etiquetado realizado con el programa label-studio (autoría propia)

3.6.2 Entrenamiento *dataset* de Cangas

Una vez realizado el etiquetado de las imágenes, serán procesadas en el programa Ultralytics HUB, en donde se dispone de modelos pre-entrenados del algoritmo YOLOv5. Dicho programa automatiza el reparto de nuestro *dataset* de prueba, dividiendo las imágenes en tres grupos, como podemos ver en la Figura 3-6. Cada uno de estos tres grupos generados tiene un propósito distinto, y cumple con una función particular en el aprendizaje de la red.

El primero de ellos, es el denominado *train*. Como se observa, es el grupo que abarca mayor número de imágenes, y será el seleccionado por la red para realizar su entrenamiento. Es importante que este grupo esté compuesto por imágenes que reflejen la máxima generalidad del *dataset*, de forma que la red no se entrene únicamente con imágenes que, por ejemplo, solo pertenezcan a unas horas fijas del día, sino que la variedad disponible suponga una experiencia de aprendizaje más completa de cara a su fase de validación.

El segundo grupo está constituido por las imágenes destinadas a la validación. Este conjunto de imágenes es empleado durante el entrenamiento de la red para comprobar que el aprendizaje permite generalizar correctamente los resultados sobre imágenes que no se emplean para el entrenamiento de los valores de los pesos de los filtros convolucionales, y permite optimizar el modelo al mismo tiempo que se entrena. De esta forma, la red accede ocasionalmente a este subgrupo, sin llegar nunca a aprender de él.

Por último, el modelo reserva un porcentaje menor de imágenes destinadas a la fase de *test*. La principal característica de esta compilación es que nunca ha sido vista por la red, hasta el momento en el que se someten a la fase de prueba final. Con ello se consigue poner a prueba a la red, y generar una serie de resultados que permitirán valorar la fiabilidad del modelo.

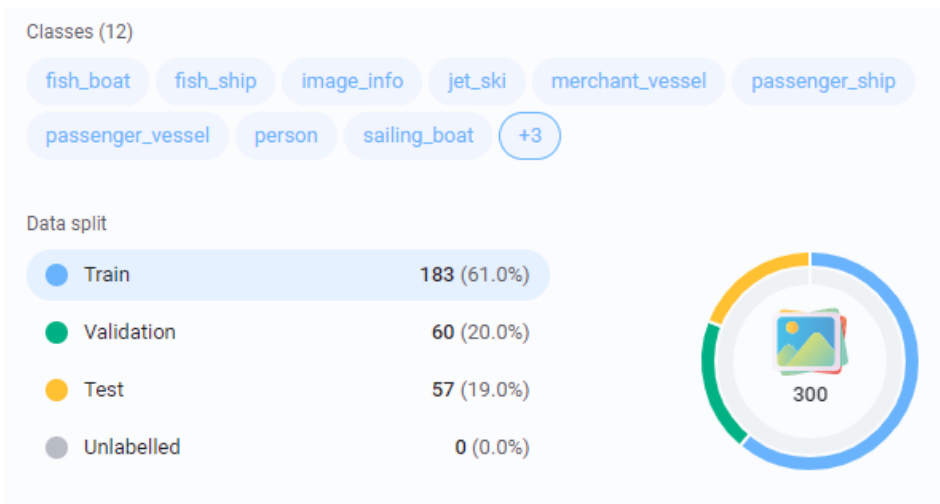


Figura 3-6 Reparto de las imágenes que componen el dataset "Webcam Cargas" según el propósito que se le asigna a cada una de ellas (gráfica generada por Ultralytics HUB de autoría propia)

La primera gráfica proporcionada en este entrenamiento es la que relaciona los valores del *recall* y *precisión*, con los resultados del mAP. Como vemos en la Figura 3-7, el mAP obtenido para un *IoU* de 0,5 es aparentemente bueno, mientras que si lo acotamos entre el 0,5 y el 0,95, su valor disminuye considerablemente. Por otro lado, la irregularidad de los valores de *precision* y *recall* obtenidos durante todo el entrenamiento permiten determinar que el modelo no es del todo fiable, tanto en detección como clasificación. Un aspecto a tener en cuenta es el número de *epochs* ordenado al modelo, lo cual evidencia que sería necesario incrementarlo para obtener mejores resultados y de mayor fiabilidad.

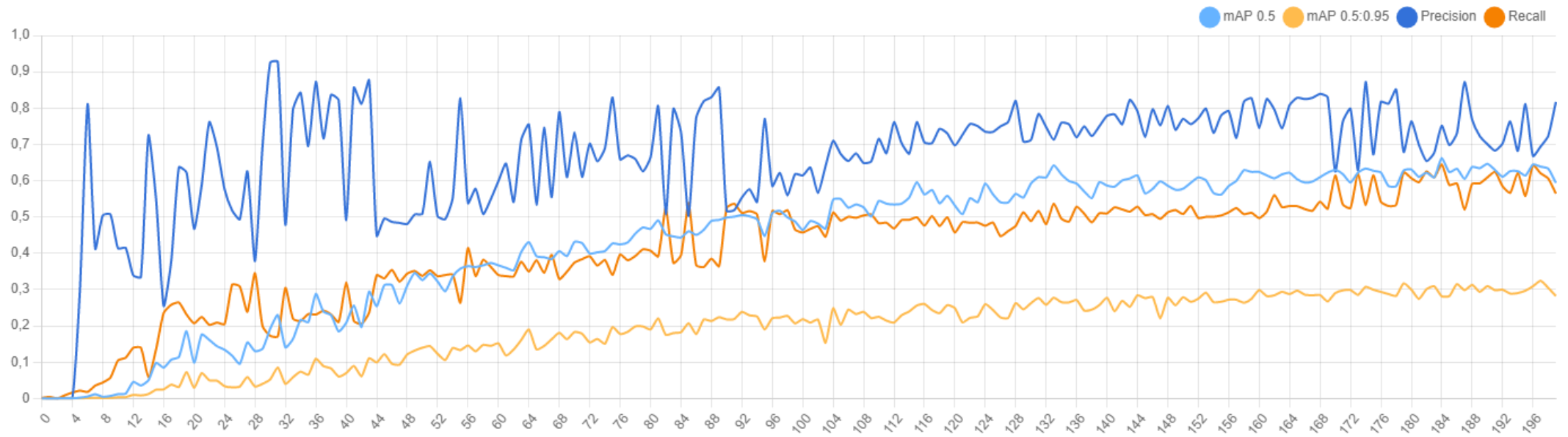


Figura 3-7 Gráfica generada por Ultralytics HUB que permite visualizar los resultados del mAP, recall y precision para cada uno de los epochs evaluados (gráfica generada por Ultralytics HUB de autoría propia)

Uno de los parámetros que ofrece el programa Ultralytics HUB para la valoración de nuestro modelo, es el *box loss* (pérdida de cajas, en su traducción al castellano). Este valor dado para cada una de las entradas, permite observar cómo va aprendiendo el modelo conforme aumenta la cantidad de imágenes que procesa. En la Figura 3-8 que se muestra a continuación, se observa como su valor va disminuyendo, tanto para el entrenamiento, como para la validación. Como se expuso en la aclaración anterior sobre el reparto del *dataset* (véase Figura 3-6), el modelo no conoce las imágenes reservadas para el *test*, por lo que en esta gráfica no está representado el *box loss* obtenido para éstas.

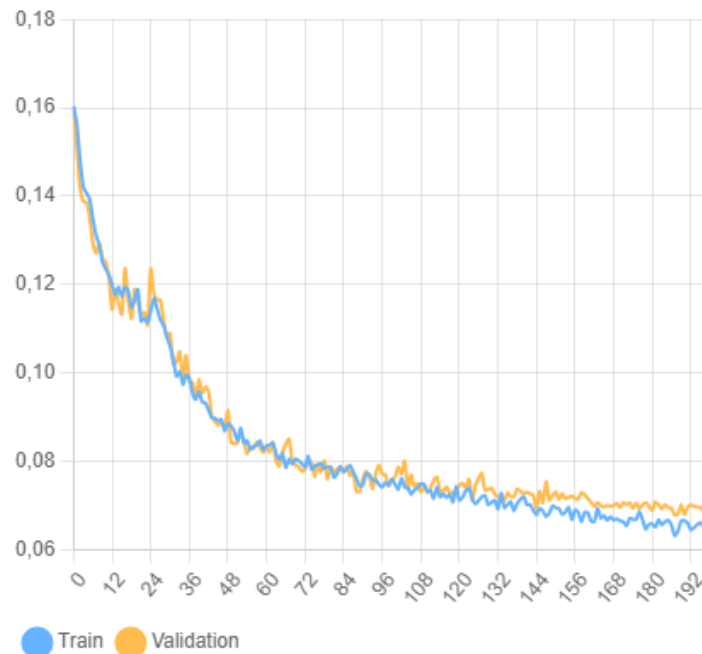


Figura 3-8 Gráfica que muestra el *box loss* para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)

Otro parámetro que permite analizar la capacidad o validez del modelo es el *class loss* (pérdida de clase o tipo, en su traducción al castellano), reflejado en la Figura 3-9. Este indicador cuantifica la capacidad que tiene el modelo para clasificar correctamente un objeto detectado (en base al etiquetado proporcionado), o si por el contrario lo hace erróneamente. Al igual que se observa en la gráfica del *box loss*, el número de errores va disminuyendo conforme obtiene mayor número de entradas.

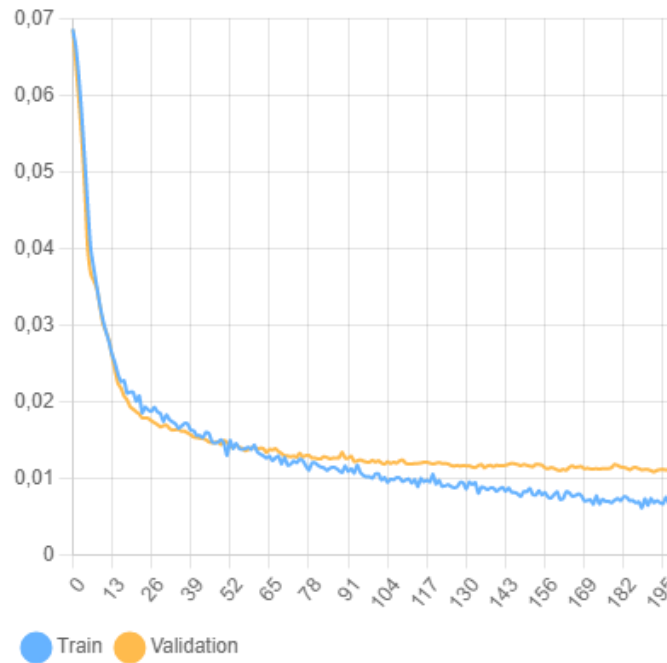


Figura 3-9 Gráfica que muestra el *class loss* para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)

Una vez analizados los parámetros *box loss* y *class loss*, el *software* genera otra gráfica que recoge el *object loss* (o pérdida de objetos, en su traducción al castellano). Como su nombre indica, este parámetro refleja la capacidad que tiene el modelo de detectar a todos los objetos presentes en una imagen de entrada. Nuevamente, y como refleja la Figura 3-10, este dato se va reduciendo conforme obtiene mayor número de entradas.

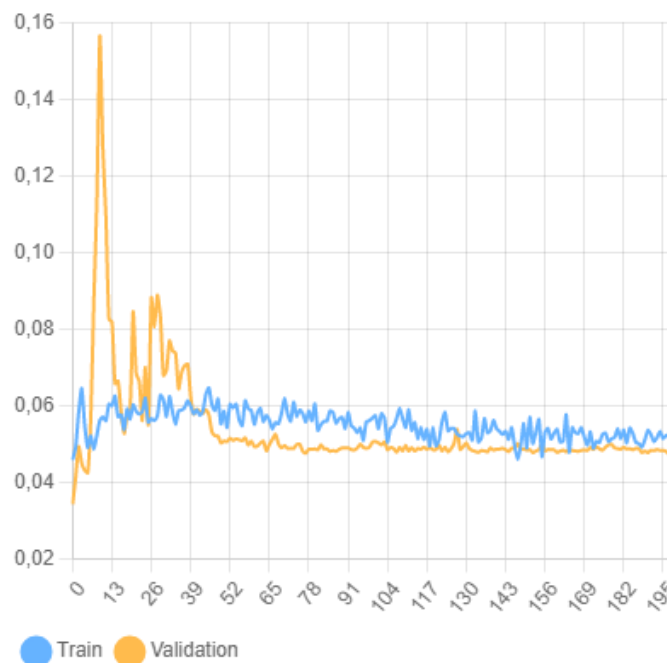


Figura 3-10 Gráfica que muestra el *object loss* para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)

Además, esta herramienta *software* ofrece un espacio dedicado a la evaluación de nuestro modelo, para el cual ha reservado una serie de imágenes con la que podemos comprobar el comportamiento de éste de forma visual. En la se muestra una de estas imágenes sometidas a *test*.

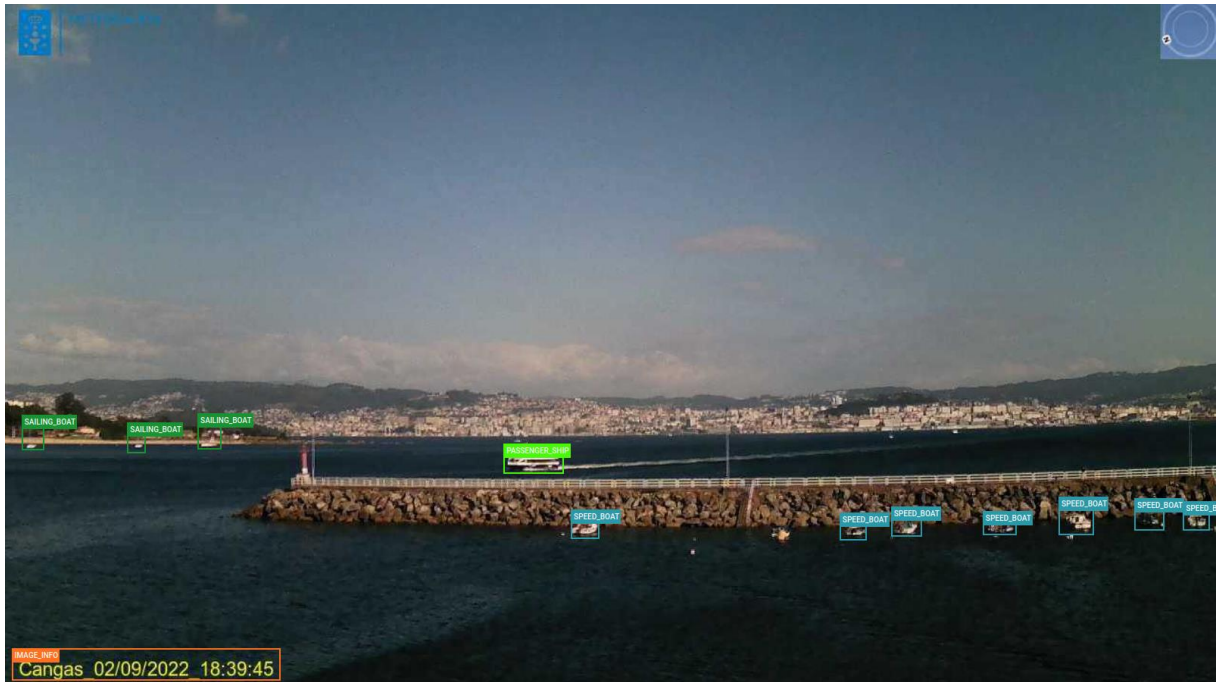


Figura 3-11 Imagen sometida a *test* tras haber entrenado el modelo con YOLOv5 en Ultralytics HUB (autoría propia)

3.7 Entrenamiento para *dataset* en Escuela Naval Militar

Como se adelantaba en el apartado anterior, en esta ocasión se realizarían las fases de etiquetado y entrenamiento para un *dataset* propio, generado en la Escuela Naval Militar en Marín, Pontevedra (Galicia, España). En los subapartados siguientes se justifica la elección del emplazamiento de grabación, se exponen las particularidades observadas durante esta fase de etiquetado, y se presenta una valoración inicial de los parámetros recogidos tras el entrenamiento del modelo con Ultralytics HUB. Posteriormente se procede a realizar el entrenamiento de modelos basados en la versión YOLOv8 nano de forma local, es decir, empleando *scripts* en *Python* y la librería que permite cargar y entrenar modelos para esta versión del algoritmo. En definitiva, se genera un proceso paralelo que permite un crecimiento del *dataset*, así como un entrenamiento del modelo hasta conseguir los resultados deseados. Dicho proceso iterativo será realizado hasta que se validen los resultados, y se obtengan suficientes datos que permitan arrojar conclusiones del presente TFG. Por último, se procede a entrenar el modelo empleando el algoritmo de detección de objetos YOLOv8 *small*, pero esta vez empleando el entorno de trabajo Google Colab como sistema de soporte para el entrenamiento, ya que la capacidad de los ordenadores en modo local no permite entrenar este tamaño de modelos.

Cabe resaltar que los *scripts* empleados han sido desarrollados por el tutor D. José Antonio González Prieto, tomándolos el autor del presente TFG para la aplicación de las tareas acometidas durante el desarrollo de este trabajo.

3.7.1 Selección del emplazamiento de grabación

Dado que este *dataset* será generado como parte del desarrollo del TFG, es importante establecer un punto de grabación que permita explotar las posibilidades de nuestro modelo. Para ello, se han realizado grabaciones de prueba en distintas posiciones en las inmediaciones de la Escuela Naval Militar. Entre ellas, se toma la indicada en la Figura 3-12, ubicada en un espigón que permite obtener imágenes, tanto de la dársena interior, como del exterior. Además, por la proximidad a las mismas, se dan situaciones de cruce y oclusión de embarcaciones, que son de interés para la generalización de nuestro modelo.

Como se aprecia en la Figura 3-12, las direcciones en la toma de imágenes serán tres. Con esto se busca probar las capacidades de generalización de los modelos generados, así como comprobar que con nuevas imágenes esta propiedad puede ser mejorada. De esta forma, si un modelo es entrenado sólo en

un entorno concreto, sus resultados serán poco fiables para cualquier otro entorno, por lo que este proceso iterativo nos permite entender mejor el funcionamiento de este tipo de sistemas de inteligencia artificial. El concepto de que el aprendizaje depende de la calidad de los datos, y que la generalización de resultados es una parte fundamental de los objetivos queda, de esta forma, reflejado en el propio proceso de creación de la solución general al problema planteado.

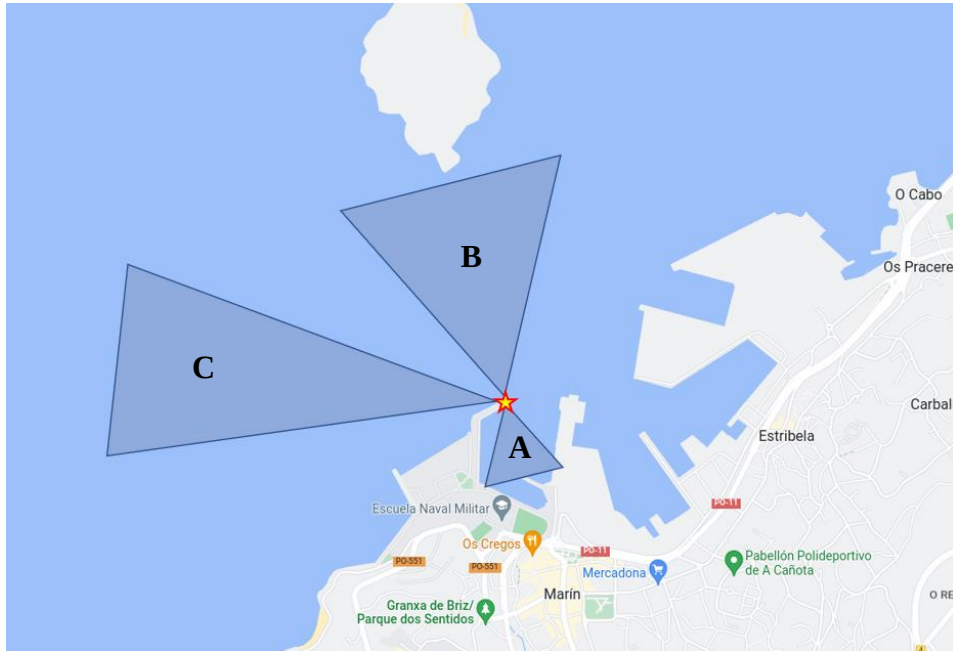


Figura 3-12 Captura extraída de *Google Maps* que refleja el emplazamiento elegido para la generación del *dataset* (autoría propia)

3.7.2 Etiquetado de videos

Elegido el emplazamiento donde se llevan a cabo las grabaciones, el siguiente proceso que se realiza es nuevamente el de etiquetado. Como se observa en la Figura 3-13, las embarcaciones habituales que se pueden encontrar en la dársena de la Escuela Naval Militar no son las mismas que proporcionaba el entorno marítimo propio del *dataset* de Cangas. Para ello, y siguiendo el mismo procedimiento, se han desarrollado nuevas etiquetas que permitan la clasificación propia de este entorno. De los videos generados, se obtienen sus *frames* y se construye un *dataset* con la elección de 300 imágenes. El tamaño del *dataset* se irá incrementando a medida que lo realimentamos para optimizar el modelo, como veremos en apartados posteriores. Las etiquetas diseñadas han sido las siguientes:

- *teaching_boat*: constituye las embarcaciones neumáticas utilizadas por instructores. Normalmente va una sola persona embarcada. Dado su aspecto, es fácilmente confundible con la clase *assault_boat*.
- *assault_boat*: comprenden las embarcaciones neumáticas utilizadas por los alumnos de Infantería de Marina de la Escuela Naval Militar. Suelen ir dotadas de entre 7 – 8 integrantes cada una.
- *sailing_boat*: esta clase está asociada a cualquier tipo de buque a vela que pueda aparecer en las imágenes del *dataset*.

Al igual que para la prueba inicial que se realiza con el *dataset* de Cangas, se emplea el software Label-Studio para realizar el etiquetado de las imágenes de entrada. Como se menciona en la introducción al apartado, el entorno empleado en la grabación de este *dataset* no es el mismo. Las particularidades propias de éste son las siguientes:

- En esta ocasión, el tiempo transcurrido entre las distintas imágenes proporcionadas al modelo es de apenas 15 minutos. A diferencia de la prueba inicial, no hay imágenes que presenten distintas condiciones meteorológicas o de luminosidad.
- Las imágenes generadas no son estáticas en su totalidad, proporcionando al modelo imágenes de videos originales que presentan movimiento, tanto en orientación, como en acercamiento. Esto supone que el encuadre de las imágenes no sea siempre el mismo.
- Respecto al medio de grabación, se lleva a cabo con un dispositivo de grabación portátil. En este caso, un teléfono móvil iPhone 13.



Figura 3-13 Imagen ejemplo de las seleccionadas para realizar el entrenamiento del *dataset* de la Escuela Naval Militar (autoría propia)

3.7.3 Entrenamiento *dataset* de la Escuela Naval Militar

Siguiendo el mismo proceso que realizábamos para la prueba inicial con las imágenes de Cangas, en esta fase se entrena el modelo con las imágenes del *dataset* generado por nosotros, una vez que han sido etiquetadas. De la misma forma, empleando el software Ultralytics HUB, se generan una serie de gráficas que permiten interpretar los resultados de nuestro modelo, así como los parámetros que proporcionan información sobre su funcionamiento.

Como se expuso en la prueba inicial del *dataset* de Cangas, el propio software se encarga de hacer un reparto de los datos de entrada, dividiéndolos en *train*, *validation* y *test*. Esto tiene como objetivo, presentar al modelo una mayor generalización de las imágenes de entrada, de manera que su aprendizaje sea el más completo a la hora de hacer frente a la fase de *test*. El reparto que nos proporciona Ultralytics HUB para el *dataset* de la Escuela Naval Militar es el siguiente:

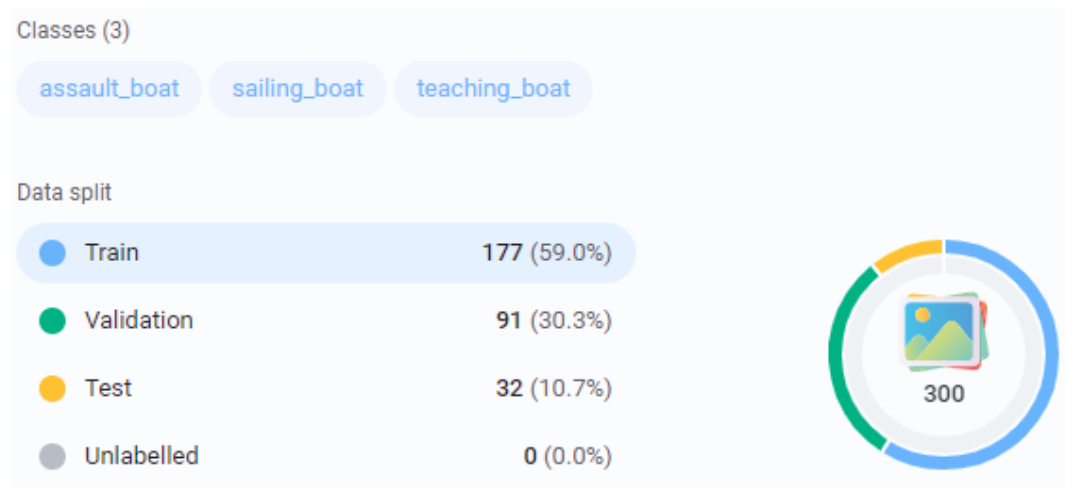


Figura 3-14 Reparto de las imágenes que componen el *dataset* "Infantes" según el propósito que se le asigna a cada una de ellas (captura extraída de Ultralytics HUB)

La primera gráfica proporcionada en este entrenamiento es la que relaciona los valores del *recall* y *precisión*, con los resultados del mAP. Como vemos en la Figura 3-11, el mAP obtenido para un *IoU* de 0,5 es aparentemente bueno, mientras que si lo acotamos entre el 0,5 y el 0,95, su valor disminuye considerablemente. Por otro lado, los valores de *precision* y *recall* obtenidos desde la entrada número 100 permiten determinar que el modelo es fiable, tanto en detección como clasificación. A diferencia del *dataset* utilizado en la prueba inicial, en este caso se ha realizado el entrenamiento con más de 400 *epochs*, lo que ha supuesto una experiencia de aprendizaje más completa para nuestro modelo.

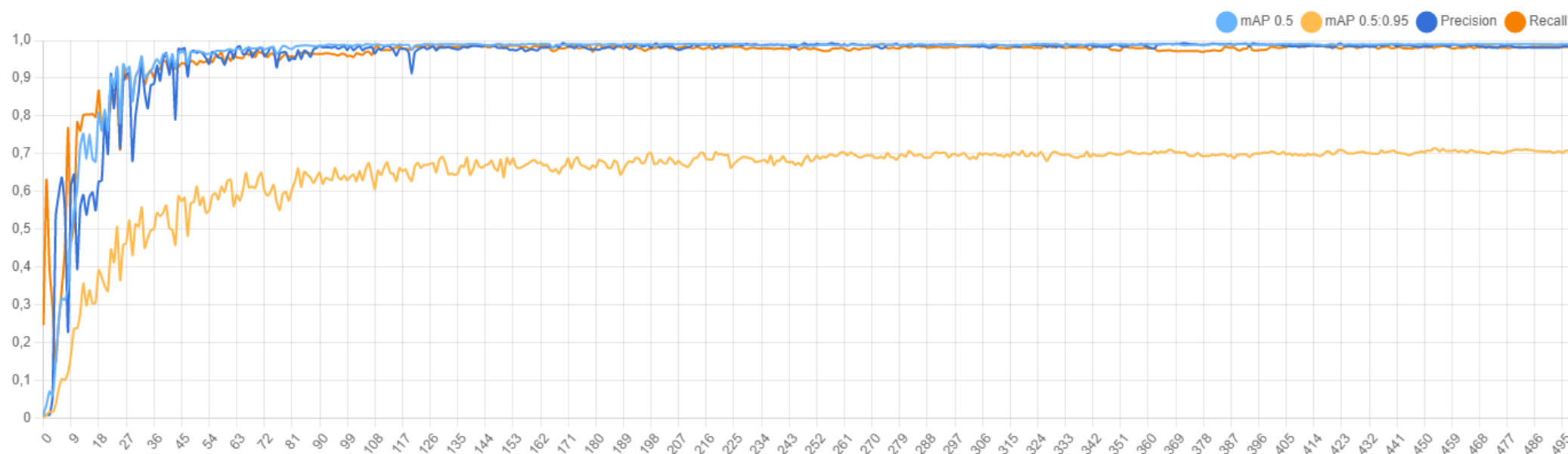


Figura 3-15 Gráfica generada por Ultralytics HUB que permite visualizar los resultados del mAP, recall y precision para cada uno de los epochs evaluados (gráfica generada por Ultralytics HUB de autoría propia)

Sin profundizar en el análisis de los resultados, cosa que se hará en el siguiente apartado de dicha memoria, se observa en la Figura 3-16 que esta vez la disparidad del *box loss* entre las entradas empleadas para el *train*, y las empleadas para el *validation*, es mucho menor. En esta ocasión, se sitúa por debajo del 0,04 rápidamente y se mantiene constante, llegando incluso a bajar de 0,02 a partir de la entrada 200, aproximadamente. Con el fin de no obtener conclusiones erróneas, hay que puntualizar que, en este caso, el número de etiquetas por imagen ha sido considerablemente menor, si lo comparamos con las imágenes empleadas en el *dataset* de Cangas. Por ello, se hace evidente el decrecimiento del *box loss* y la mejora de los resultados obtenidos.

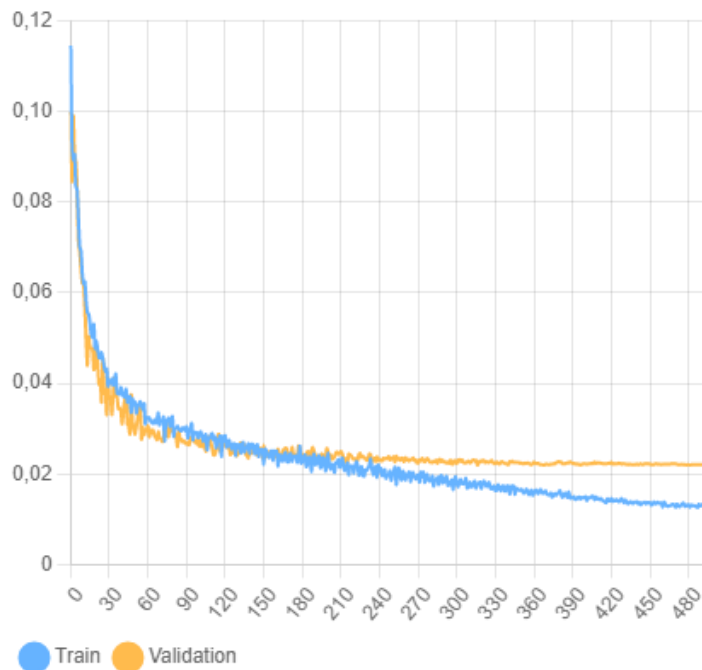


Figura 3-16 Gráfica que muestra el *box loss* para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)

Asimismo, se observa en la Figura 3-17 que sigue la misma tendencia para la gráfica del *class loss*, donde su valor, acaba siendo prácticamente nulo.

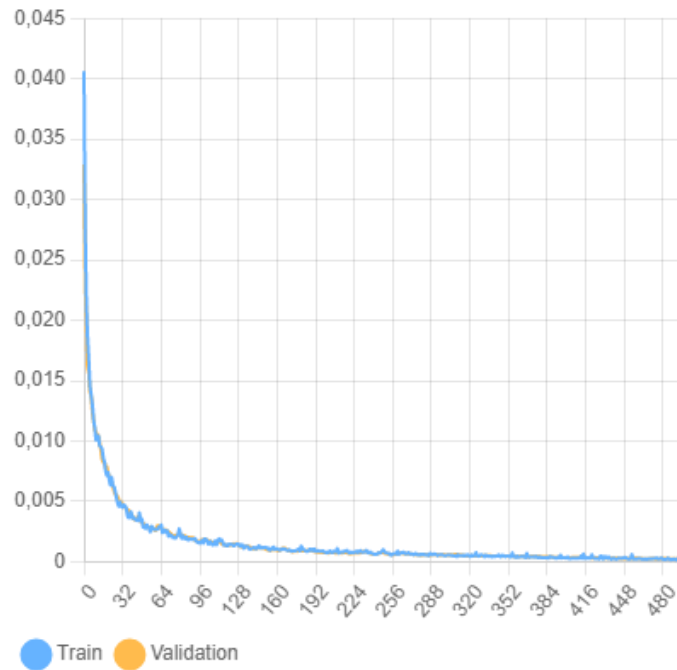


Figura 3-17 Gráfica que muestra el *class loss* para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)

En este caso, si bien el valor de las entradas empleadas para el *train* sí que se mantiene constante en su decrecimiento, en la Figura 3-18 se aprecia una ligera tendencia ascendente en los valores del *validation* a partir de la entrada número 200, aproximadamente.

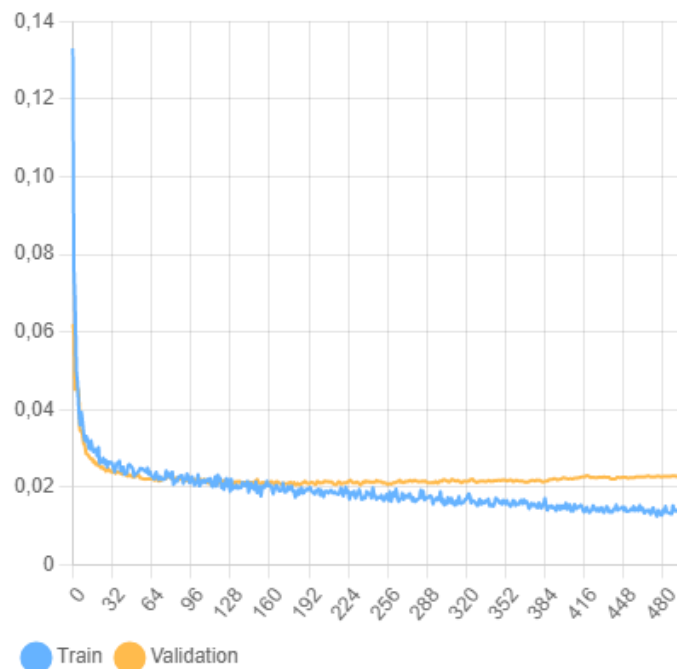


Figura 3-18 Gráfica que muestra el *object loss* para cada una de las entradas evaluadas (gráfica generada por Ultralytics HUB de autoría propia)

3.7.4 Introducción de nuevas imágenes al dataset

Una vez que se ha entrenado el modelo con Ultralytics HUB, se procede a ampliar el tamaño del *dataset* con el fin de optimizar nuestro modelo. Para ello, se pasa a trabajar con el entorno que permite emplear modelos del algoritmo YOLOv8, el cual aún no está integrado en la web de Ultralytics, por lo

que es necesario trabajar con este entorno mediante el uso de librerías que se ejecutan desde scripts desarrollados en *Python*. El objetivo es emplear un procedimiento iterativo, en donde, en cada paso se emplee el modelo generado en el paso anterior para generar las etiquetas de las nuevas imágenes seleccionadas, corrigiendo y validando dichas etiquetas en la herramienta Label-Studio.

Por ello, se introducen imágenes grabadas desde el mismo emplazamiento, pero esta vez con una orientación opuesta, hacia los exteriores de la Escuela Naval Militar. De esta forma, aumentan los elementos que tiene que analizar el modelo entrenado anteriormente, obligándole ahora a trabajar en dos entornos distintos. La introducción de nuevas imágenes en el *dataset* se realizará desde el entorno web Label-Studio, pero para ello, se extraerán imágenes de los videos incluidos en dicho *dataset*, mediante un *script* de *Python*, que puede ser consultado en el Anexo IV: *Script* para importación de imágenes.

El objetivo que se busca con este proceso es proporcionar imágenes al modelo, con un entorno distinto al que ha servido de entrenamiento. De esta forma, se importan nuevas imágenes al *dataset*, y se visualizan mediante Label-Studio. Estas imágenes recién incorporadas, incluyen también etiquetas generadas automáticamente por el *script*, que podrán ser corregidas en el entorno web antes de proceder a su exportación.

Este proceso de importación de nuevas imágenes se realiza progresivamente, al mismo tiempo que se visualizan los resultados.

3.7.5 Obtención de videos con tracking

Como se expone en el apartado de “Problemas a resolver” de este capítulo, una vez resuelto el problema de la detección de objetos en imágenes, nuestro modelo debe ser capaz de hacer seguimiento de éstos en video. Este seguimiento, será entendido como la capacidad del modelo de detectar un objeto y correlacionarlo en cada *frame* del video, de forma que le atribuya una identidad al mismo, y la mantenga durante toda su aparición en el video. Además, en la justificación de los algoritmos empleados en el presente trabajo, se elige YOLOv8 [51] por contar entre sus prestaciones con dos algoritmos de *tracking*, lo cual facilita el desarrollo de esta tarea. Es importante mencionar que el acceso a los videos que se van a mencionar de aquí en adelante, obligará o no a la descarga del mismo por parte del lector (depende de factores como el dispositivo, sistema antivirus, sistema operativo, etc.), siendo recomendable su descarga para evitar una pérdida notable de calidad en la imagen.

Para acometer esta tarea, haremos uso de un *script* de *Python* que será ejecutado nuevamente en VSCodium. Dicho *script* se puede consultar en el Anexo V: *Script* para *tracking*, y permite obtener videos con una interfaz como la que se observa en la Figura 3-19. Para obtener resultados de una experiencia realista y completa de aprendizaje, se ejecuta este *script* sobre videos grabados en ambos entornos, tanto el interior como el exterior de la dársena de la Escuela Naval Militar.



Figura 3-19 Ejemplo de un *frame* de los videos obtenidos al ejecutar el *tracking* (autoría propia)

Durante la obtención de estos videos, se observan una serie de anomalías, que serán interesantes de analizar en el capítulo dedicado a los resultados y validación del trabajo. Estas anomalías son las que debemos tomar como objetivo, a la hora de seleccionar nuevas imágenes en la realimentación de nuestro *dataset*. Los problemas observados más relevantes en la obtención de los videos de *tracking* son los siguientes:

- Pérdida de *bounding boxes* tras darse situaciones de oclusión o excesivo alejamiento del objeto.
- Equivocación del modelo en la clasificación del objeto detectado, según el perfil mostrado y la experiencia previa del modelo en el entorno examinado. Conforme se va realizando el proceso iterativo, la frecuencia de este tipo de errores es mucho menor. Se observa que el modelo, en las primeras iteraciones, confunde con facilidad los objetos *assault_boat* con los *teaching_boat*.
- Confusión de elementos propios del entorno, como rocas o fragmentos de islotes, con objetos detectables sobre los que ha sido entrenado el modelo. Un ejemplo de ello es la captura obtenida del video “VIDEO_EJEMPLO_ERROR_CONFUSION_ROCA_ZONA_C”, como se muestra en la Figura 3-20. Otro ejemplo de esto se puede observar en el video “VIDEO_EJEMPLO_ERROR_CONFUSION_ISLA_ZONA_B”, en cuyo caso se aprecia que el modelo, en un determinado momento confunde la cota de una isla situada en las inmediaciones del emplazamiento de grabación, con una embarcación de la clase *assault_boat*. Ambos videos pueden ser consultados en los siguientes enlaces:

https://drive.google.com/file/d/1fVE14M7pasp94XOcW3qSx51ROCZAIZRD/view?usp=share_link

https://drive.google.com/file/d/1pjMXi8rVYaf-XmDFjUCYIEZDN0Vx7q/view?usp=share_link

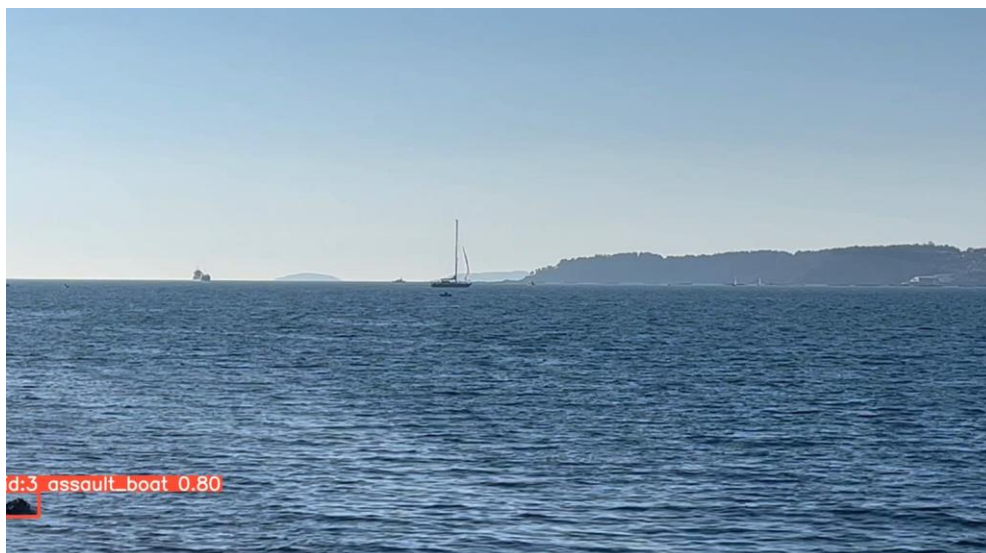


Figura 3-20 Ejemplo de error observado durante el *tracking* de los videos (autoría propia)

3.7.6 Reentrenamiento del modelo

Para cerrar el ciclo de este proceso iterativo, se recurre nuevamente a la ejecución de un *script* en VSCodium. Habiendo “realimentado” previamente el *dataset* empleado para entrenar a nuestro modelo, se analizan los resultados de los videos obtenidos con *tracking* en el apartado anterior. El *script* de *Python* que ejecutaremos en este caso (que se puede consultar en el Anexo VII: *Script* para entrenamiento del modelo), nos permite entrenar nuestro modelo de nuevo. Aquí radica la principal diferencia de esta fase, ya que los entrenamientos realizados hasta el momento con Ultralytics HUB, no permiten el reentrenamiento de un modelo, teniendo que crear uno nuevo para cada entrenamiento realizado. Como se puede deducir, esta experiencia de reentrenamiento permitirá que nuestro modelo acumule experiencia de entrenamientos anteriores, consiguiendo un producto trabajado y fiable en apenas varias iteraciones.

A continuación, se adjunta una tabla resumen que recoge los datos a tener en cuenta en cada uno de los entrenamientos realizados, pero antes de eso, es necesario entender algunos parámetros que ayudan a la interpretación de los mismos.

El primero de ellos, es el término *epoch* [52]. Como se puede consultar en el Anexo VII: *Script* para entrenamiento del modelo, o en la propia Figura 3-21, uno de los parámetros que debemos establecer es este. El *epoch* en *Machine Learning* es entendido como cada uno de los ciclos que completa el algoritmo en el entrenamiento de un determinado *dataset*. Por lo tanto, cuanto mayor sea el número de *epochs* establecidos, mayor será el entrenamiento del modelo, lo que a su vez requerirá mayor capacidad computacional. Cabe destacar que este incremento de *epochs* será directamente proporcional al tiempo invertido en el entrenamiento.

```
##### TRAIN CONFIGURATION #####
TRAIN_MODEL_NAME = "yolov8n.pt" #"lanchas_infantes_yolo8n.pt"#"yolov8s.pt" #"lanchas_infantes_yolo8s.pt"
TRAIN_EPOCHS = 800
TRAIN_IMAGE_SIZE = 800
TRAIN_RUN_NAME = "TRAIN_DATASET_393_YOLO8N"
TRAIN_DEVICE = "0" #"cpu" #
```

Figura 3-21 Fragmento del *script* empleado para el entrenamiento del modelo (autoría propia)

Otro concepto a tener en cuenta, es el que aparece encuadrado en rojo en la Figura 3-22, conocido como *patience*. Este parámetro permite establecer el número de *epochs* en el que el sistema debe concluir si está aprendiendo lo suficiente, o por el contrario le es imposible seguir haciéndolo. En caso de alcanzar ese valor y no conseguir entrenar al modelo lo suficiente, éste parará automáticamente el entrenamiento.

```
if __name__ == '__main__':
    try:
        print('***** TRAIN YOLO8 WITH DATASET *****')

        # Configure
        configuration = configure()

        # Create and train model
        model_path = configuration['models_path'] + configuration['model_version'] + "/" + TRAIN_MODEL_NAME
        print('model_path=', model_path)
        model = YOLO(model_path)
        yaml_source = configuration['images_yolo_path'] + "data.yaml"
        #print('yaml_source=', yaml_source)
        model.train(data = yaml_source, epochs = TRAIN_EPOCHS, imgsz = TRAIN_IMAGE_SIZE, device = TRAIN_DEVICE, patience = 100,
        batch = configuration['train_batch_size'], save = True, project = configuration['path'] + "RUNS/TRAIN/", name = TRAIN_RUN_NAME)
```

Figura 3-22 Fragmento del *script* empleado para el entrenamiento del modelo (autoría propia)

Comprendidos estos términos, en la Tabla 3 podemos observar parámetros de interés a la hora de analizar la progresión del modelo en el entrenamiento. Estos parámetros serán principalmente el número de iteración, el tamaño del *dataset*, el número de *epochs* establecidos, y el número de *epoch* alcanzado en el entrenamiento.

Nº iteración	Tamaño <i>dataset</i>	<i>Epochs</i> establecidos	<i>Epoch</i> alcanzado
1	400	1000	51
2	425	1000	51
3	450	1000	56
4	500	1000	208
5	550	1000	208
6	600	1000	208
7	650	1000	210
8	700	1000	210
9	750	1000	208

Tabla 3 Resumen de los parámetros a tener en cuenta en las iteraciones realizadas (autoría propia)

3.7.7 Entrenamiento del modelo con *dataset* compuesto por 750 imágenes

Como podemos observar por el número de *epochs* alcanzado, el modelo determina que no es capaz de ser optimizado en ese entrenamiento, por lo que se procede a hacer una nueva serie de videos de *tracking*, que nos permitirá evaluar la mejora obtenida. En esta ocasión, se expone un ejemplo de todos los elementos que genera la ejecución del *script* de *train*. Este concretamente refleja los resultados obtenidos en la iteración número 9, cuyo tamaño de *dataset* ha sido de 750 imágenes, y ha sido entrenado con el modelo de YOLO optimizado hasta el momento.

En cuanto al análisis paramétrico, el entrenamiento genera una matriz de confusión del modelo. Esta matriz que podemos observar en la Figura 3-23, refleja el porcentaje de acierto entre lo asignado y lo real en la tarea de detección de objetos, comparando el nivel de confusión entre las distintas clases proporcionadas para el entrenamiento, así como con el *background* de las imágenes.

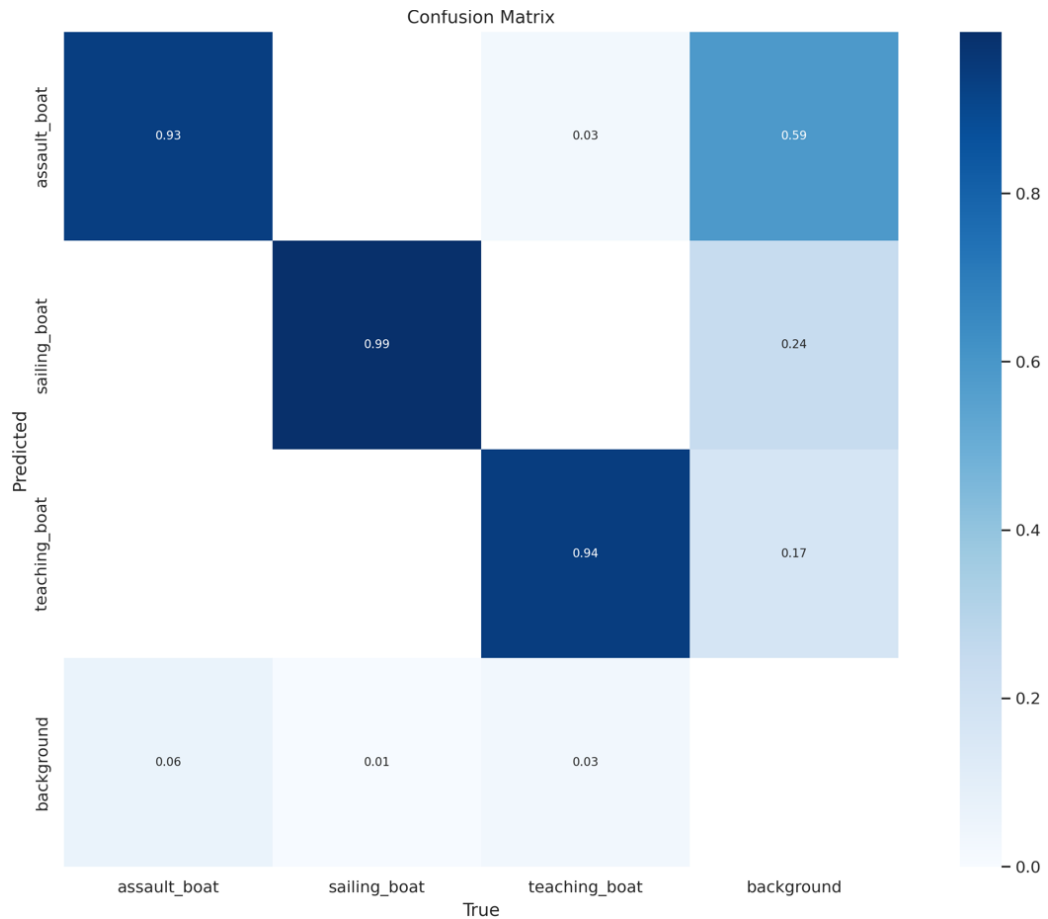


Figura 3-23 Matriz de confusión generada durante el entrenamiento de la iteración número 9 (autoría propia)

Continuando con el análisis paramétrico, la ejecución del *script* también proporciona una serie de gráficas que permiten el análisis de los parámetros *precisión* y *recall*. A continuación se presentan las gráficas de dichos parámetros, y tras éstas, la gráfica que relaciona ambos.

La Figura 3-24 muestra un ejemplo de la gráfica generada para el parámetro *precisión*. Recordando el significado de este parámetro, se definía como la cuantificación del porcentaje de los objetos detectados por la red, o lo que es lo mismo, el cociente entre el número de objetos correctamente detectados con respecto al número total de objetos propuestos.

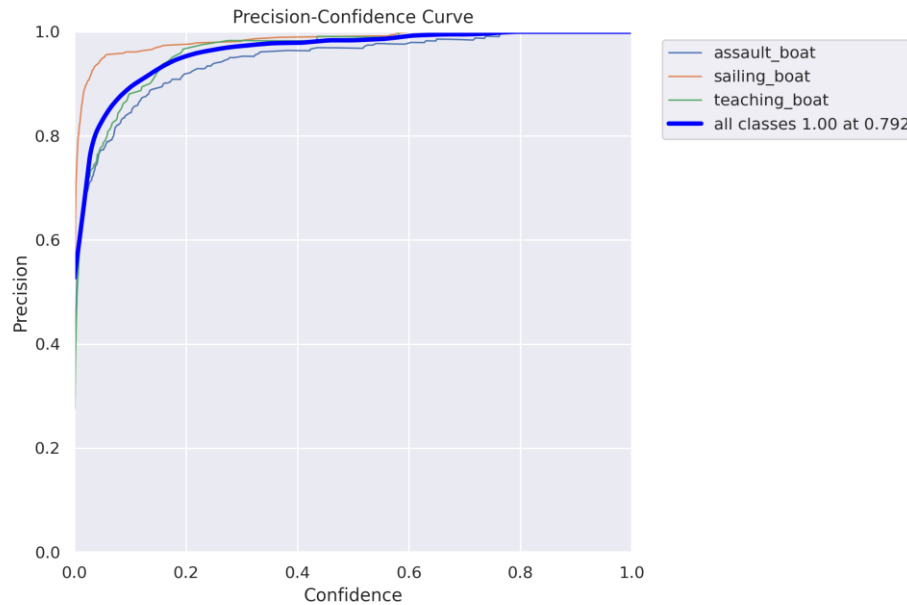


Figura 3-24 Gráfica que muestra el *precision* para cada clase de objeto detectado (autoría propia)

Asimismo, el modelo genera el mismo concepto de gráfico expuesto en la Figura 3-25, pero esta vez evaluando la capacidad de *recall* del mismo. Haciendo alusión de nuevo a este parámetro, cabe recordar que medía la relación entre el número de objetos detectados correctamente y el número total de objetos que deberían haber sido detectados.

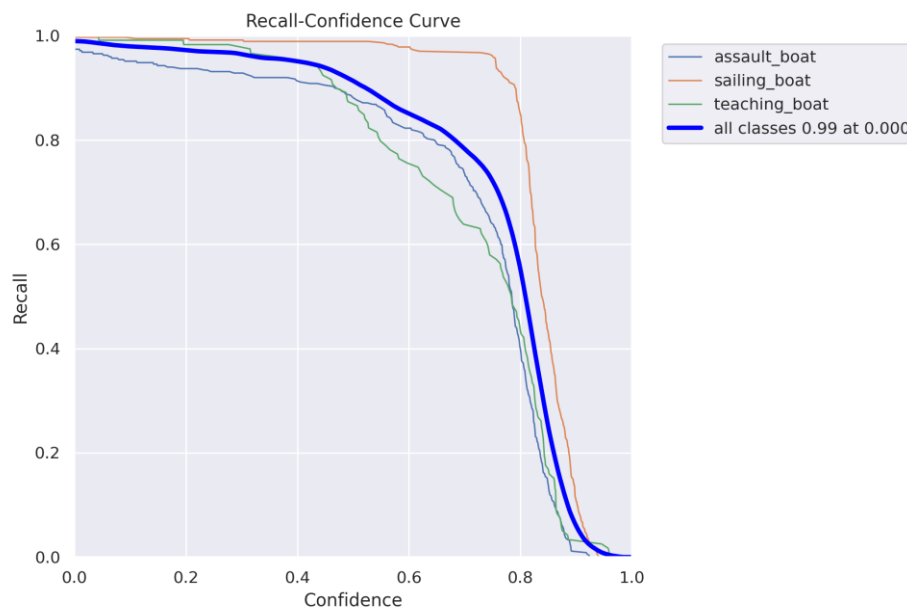


Figura 3-25 Gráfica que muestra el *recall* para cada clase de objeto detectado (autoría propia)

Como se adelantaba, el *script* genera una gráfica que relaciona el *precision* y el *recall*. Como se puede especular, la gráfica que generaría un modelo perfecto tendría una forma muy similar a la de un cuadrado, ya que estaría obteniendo el valor de la unidad en ambos parámetros. Conseguir esto supone una gran dificultad, ya que siempre que se intente optimizar los resultados de uno, conllevará el decrecimiento en los valores del otro.

Por ello, sin hacer un análisis profundo en los resultados obtenidos para este modelo en particular, la Figura 3-26 muestra que los valores fruto de este entrenamiento son razonablemente positivos, ya que la curva *precisión – recall* se asemeja a la forma cuadrada mencionada anteriormente.

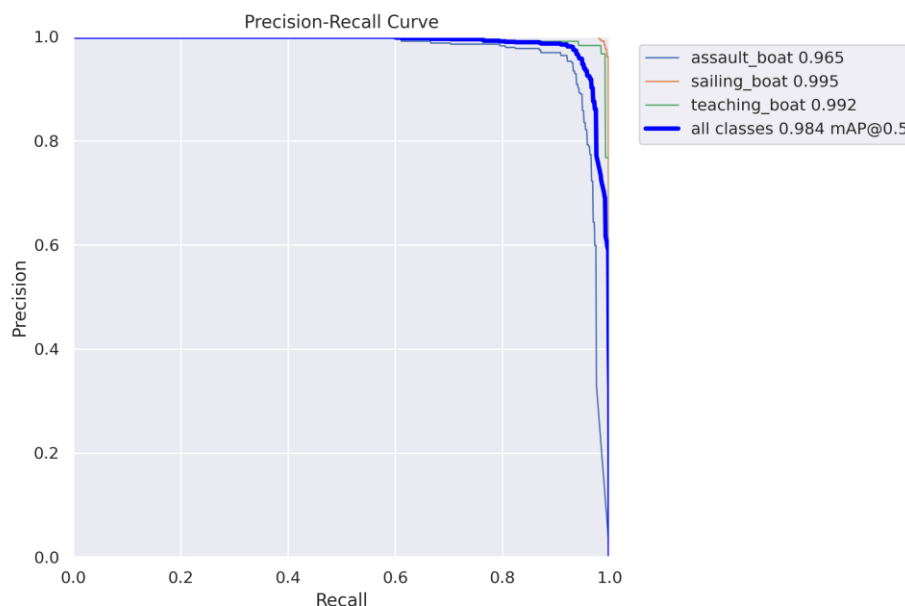


Figura 3-26 Gráfica generada por el script y que relaciona ambos parámetros (autoría propia)

Prosiguiendo con los elementos producto de este entrenamiento, el modelo proporciona una serie de imágenes que sirven de prueba para comprobar las capacidades que ha adquirido durante esta fase de *train*. Dado que genera una gran cantidad de ellas, en la Figura 3-27 se ha hecho un recopilatorio de algunas que el modelo ha tratado, mostrando además los resultados en varios entornos.

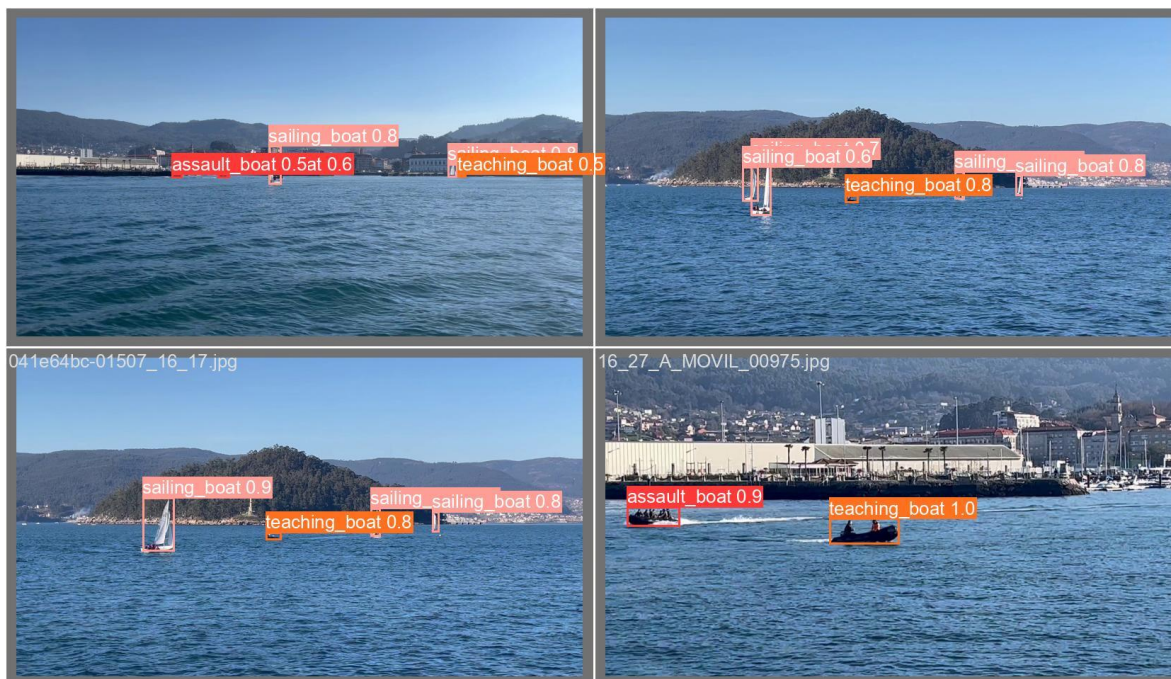


Figura 3-27 Imágenes generadas por el modelo de 750 imágenes durante la ejecución del script de entrenamiento (autoría propia)

Tras este entrenamiento del modelo, el proceso iterativo requiere nuevamente una comprobación de los resultados. Para ello, se han vuelto a *trackear* videos originales de los tres entornos que ha permitido nuestro emplazamiento de grabación, y los cuales se justificaron al comienzo del desarrollo del presente TFG. Los videos obtenidos se exponen en el capítulo dedicado a “Resultados y validación”.

Cabe mencionar, que este proceso iterativo de entrenamiento se realiza con una de las versiones de YOLO v8 que requiere menor capacidad de almacenamiento, como es YOLOv8n. El entrenamiento final

que se realice, una vez que se considere optimizado el modelo, será llevado a cabo con la versión YOLOv8s, que como podíamos observar en la Tabla 2 del apartado “Algoritmos para detección de objetos”, tiene capacidades considerablemente superiores a su versión nano en la resolución de estas tareas.

3.7.8 Entrenamiento final del modelo en Google Colab

Alcanzado el nivel de optimización deseado en nuestro modelo, se cambia nuevamente de entorno de trabajo, trasladando ahora el proceso de entrenamiento a la extensión *Google Colab*. A diferencia de los entrenamientos anteriores, en este entrenamiento se busca probar la respuesta de nuestro modelo, empleando una versión de YOLOv8 con mayor capacidad que el anterior. En este caso se ha tomado YOLOv8s, donde, como se puede observar en la , la principal diferencia radica en el número de parámetros que tiene en cuenta para el entrenamiento. Mientras que la versión YOLOv8n que entrenábamos en fases anteriores utiliza poco más de 3 millones de parámetros, la versión YOLOv8s lo hace con casi 12 millones. [53]

Model	size (pixels)	mAP ^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Tabla 4 Tabla comparativa de las diferentes versiones de YOLOv8 (fuente: GITHUB)

Esto requerirá adquirir un paquete adicional de la extensión *Google Colab*, que permita procesar el modelo disponiendo de más unidades de procesamiento. Por otro lado, en este entrenamiento se pretende entrenar el modelo con un algoritmo YOLO original, que no haya visto el *dataset* en ocasiones anteriores.

Por tanto, teniendo en cuenta la cantidad de unidades de procesamiento adquiridas, que permitirán un entrenamiento más prolongado, además del empleo de una versión que utiliza casi cuatro veces más parámetros que la usada hasta ahora, los resultados deben ser positivos. A continuación se exponen dichos resultados, de la misma forma en la que se hace en el subapartado “Entrenamiento del modelo con *dataset* compuesto por 750 imágenes”. Los resultados de este último entrenamiento se expondrán en el capítulo “Resultados y validación”.

Finalmente, en la Figura 3-28 se puede visualizar todo el proceso llevado a cabo durante la fase de desarrollo de este trabajo, sin considerar su integración en el dispositivo Coral AI, que se mencionará en el último apartado del presente capítulo.

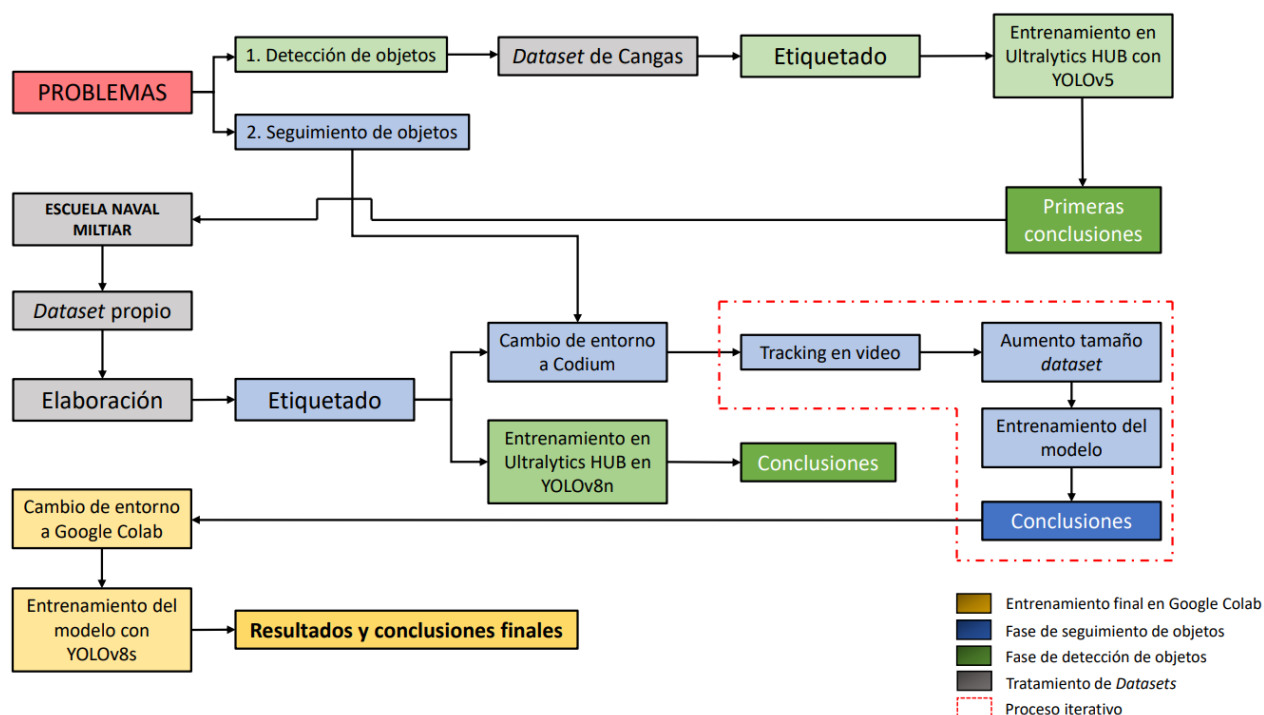


Figura 3-28 Esquema que detalla el proceso llevado a cabo durante el desarrollo del presente TFG (autoría propia)

3.8 Instalación en dispositivo *Coral AI*

Como hemos visto a lo largo del desarrollo, con un *dataset* elaborado de forma autónoma, y un proceso iterativo que ha permitido optimizar la resolución de tareas, hemos obtenido un modelo capaz de hacer frente a los dos problemas que se le proponían.

Es por ello que uno de los propósitos del presente TFG es hacer viable la adaptación de un modelo de *Deep Learning* a una unidad de procesamiento menor, como puede ser un dispositivo *Coral AI*, capaz de transportar dicho modelo a dispositivos menores que puedan ser transportados fácilmente. Resaltar que esta tarea ha sido imposible acometerla debido a las vicisitudes propias de este sistema, en cuanto a la compatibilidad ente el formato de nuestro modelo, y el que requiere el propio dispositivo. Aún así, se espera poder realizarlo antes de la exposición oral de este trabajo.

4 RESULTADOS Y VALIDACIÓN

En este capítulo, se procede al análisis de los resultados obtenidos en las distintas etapas del desarrollo. Sólo se comentarán aquellos que permitan extraer conclusiones posteriormente, y aquellos que por sus particularidades sean objeto de especial interés para la validación del modelo. Para ello, el presente capítulo se divide principalmente en tres partes. La primera de ellas se centra en los resultados obtenidos en la prueba inicial para el *dataset* de Cangas, cuyo modelo fue entrenado con YOLOv5. En el segundo se procede a la validación de los resultados obtenidos mediante la ejecución de *scripts* en *Codium*, en cuyo caso se entrenó al modelo con YOLOv8n y se obtuvieron las primeras imágenes de seguimiento. En el siguiente apartado se hace lo mismo, pero habiendo entrenado el modelo en el entorno Google Colab y con una versión de mayor capacidad de YOLO, siendo ésta YOLOv8s. Por último, se procede a una validación y análisis general de los resultados obtenidos, tomando aquellos parámetros que son esenciales para la comprensión de dichos resultados.

4.1 Resultados de la prueba realizada con el *dataset* de Cangas con YOLOv5

Dado que este *dataset* sólo se emplea en una de las primeras fases del desarrollo de este trabajo, los únicos resultados que ofrece esta prueba son relativos a la tarea de detección de imágenes. A pesar de esto, permite visualizar las capacidades del algoritmo ante una cantidad mayor de objetos que con la que cuenta el *dataset* elaborado en la Escuela Naval Militar.

La refleja una de las imágenes extraídas en la fase de *test*, es decir, imagen que el modelo no ha visto hasta que se le proporcionado para evaluar su comportamiento. En ella se aprecia una alta capacidad de detección de objetos, objetos en su mayoría de pequeño tamaño, siendo de especial relevancia la cantidad de ellos que es capaz de detectar al mismo tiempo. Es importante recordar que este modelo ha sido entrenado con un *dataset* únicamente de 300 imágenes, y que lo ha hecho con YOLOv5, una versión que ofrece menores prestaciones.



Figura 4-1 Imagen obtenida tras el entrenamiento del *dataset* de Cangas en Ultralytics HUB con YOLOv5 (autoría propia)

4.2 Validación de los resultados obtenidos en entorno VSCodium con YOLOv8n

Como se expone en el capítulo de desarrollo, la principal diferencia entre esta fase y la de prueba inicial radica en el *dataset* empleado. Al tomar un *dataset* propio, es necesario un procesamiento previo de los datos que permita escoger aquellas imágenes que favorezcan la generalización del modelo.

Un factor común a toda la elaboración del *dataset* ha sido la no variación de las condiciones meteorológicas de los videos grabados. Es importante tener en cuenta esta particularidad de cara a la validación de este modelo, ya que todos los resultados obtenidos han sido en unas condiciones meteorológicas favorables, y dichos resultados podrían no ser tan positivos en caso de que dichas condiciones fueran desfavorables.

Por otro lado, un elemento que permite asignar una gran fiabilidad al modelo es la variación del entorno que muestran las imágenes. En esta ocasión, el modelo no conoce sólo una vista, como es el caso del entorno observado en la prueba inicial con el *dataset* de Cangas. La elección de tres vistas distintas desde el mismo lugar de grabación, son un claro aliciente en el entrenamiento del modelo.

A continuación, a fin de validar este modelo, se adjunta un ejemplo de video desde cada una de las tres posibles vistas que ofrece el emplazamiento de grabación. Estos videos han sido denominados como “VIDEO_TRACK_MODELO_750_ZONA_A”, “VIDEO_TRACK_MODELO_750_ZONA_B” y “VIDEO_TRACK_MODELO_750_ZONA_C”, y se pueden visualizar en los siguientes enlaces en su respectivo orden:

https://drive.google.com/file/d/1cWrkJ9M8rLLsB3dHAcSEJiheEV5toRy7/view?usp=share_link

https://drive.google.com/file/d/1BAxyiwIlgrkn2cCaZHySQ7fjNRCbojRX/view?usp=share_link

https://drive.google.com/file/d/133Eie9NKZw8FFalnjcXBUW48vUtxMtX/view?usp=share_link

4.3 Validación de los resultados obtenidos en entorno Google Colab con YOLOv8s

Con el fin de validar el modelo obtenido en el entrenamiento realizado con YOLOv8s, en este apartado se exponen los resultados de dicho entrenamiento, analizando las gráficas e imágenes de prueba generados durante el mismo. Pese a realizarse, en este caso, en el entorno Google Colab, los elementos proporcionados por el entorno para el análisis del modelo son prácticamente los mismos. Por tanto, se exponen algunas de las gráficas que permiten analizar la paramétrica del modelo, así como las imágenes de prueba que permiten hacer una interpretación más visual de los resultados.

Empezando con la paramétrica, en la matriz de confusión reflejada en la Figura 4-2, los resultados son muy similares a los obtenidos con el modelo optimizado del entrenamiento con YOLOv8n. Como se puede observar, respecto a la clase *sailing_boat*, el modelo ya obtiene la unidad en la matriz, no dando lugar a confusión con otra clase de objeto. Otro aspecto a tener en cuenta, es el grado de confusión que sigue mostrando el modelo entre las distintas clases y el *background*. Esto se debe en gran parte a que el *background* de nuestro *dataset* no es el horizonte, ya que, al situarse nuestro emplazamiento de grabación en las inmediaciones de una ría, en la mayoría de ocasiones habrá elementos naturales o terrestres propios del entorno. También hay que tener en cuenta, que las condiciones meteorológicas existentes en los videos que conforman el *dataset* son favorables en general, por lo que esta matriz de confusión no es representativa de aquellos casos en los que las condiciones fueran desfavorables. Se puede deducir que en dicho caso, la confusión entre las clases de objetos y el *background* podría ser aún mayor.

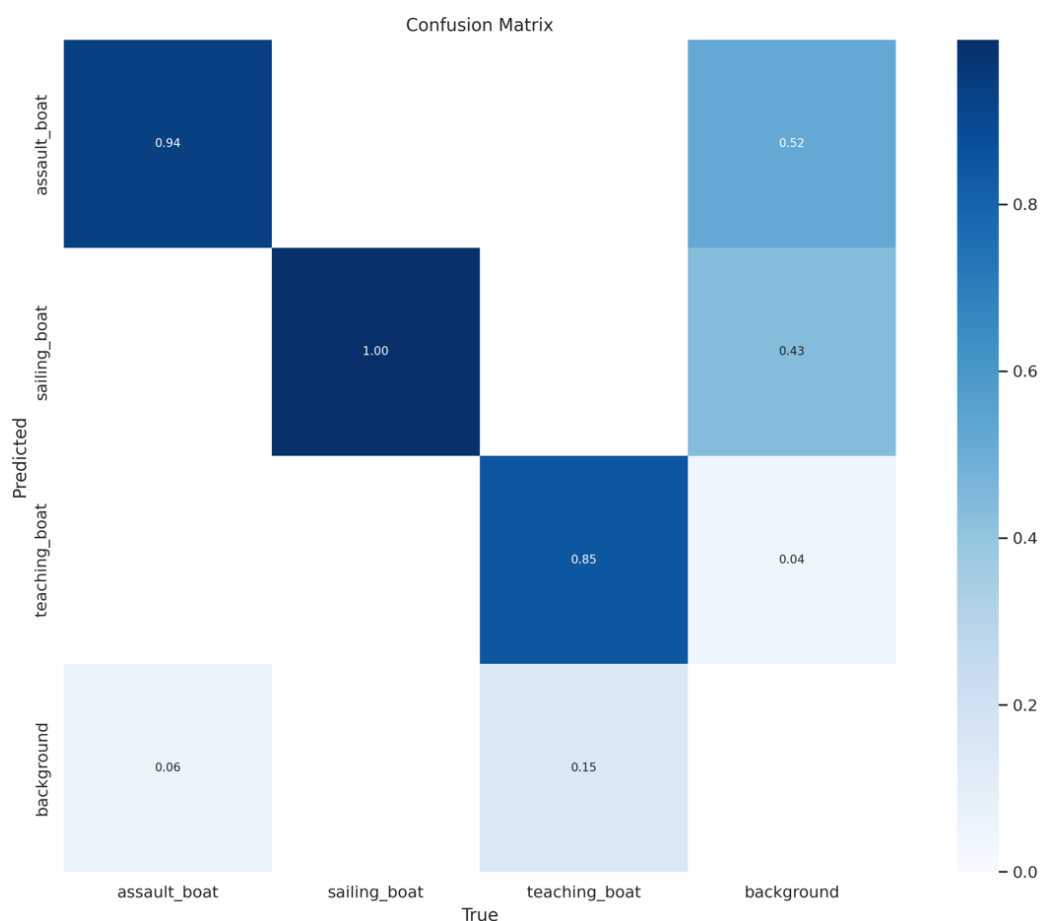


Figura 4-2 Matriz de confusión generada durante el entrenamiento en entorno Google Colab (autoría propia)

Al igual que vimos en el apartado “Entrenamiento del modelo con *dataset* compuesto por 750 imágenes”, contamos con tres gráficas de gran interés para analizar el comportamiento de nuestro modelo. Empezando por la de *precision*, en la Figura 4-3 podemos observar que el modelo ofrece unos índices altos en este parámetro, siendo mayor de 0,9 desde un nivel de confianza de 0,1,

aproximadamente. En este caso, los resultados obtenidos con YOLOv8s son mejores que los que podíamos observar en la Figura 3-24, demostrando una fiabilidad considerablemente positiva.

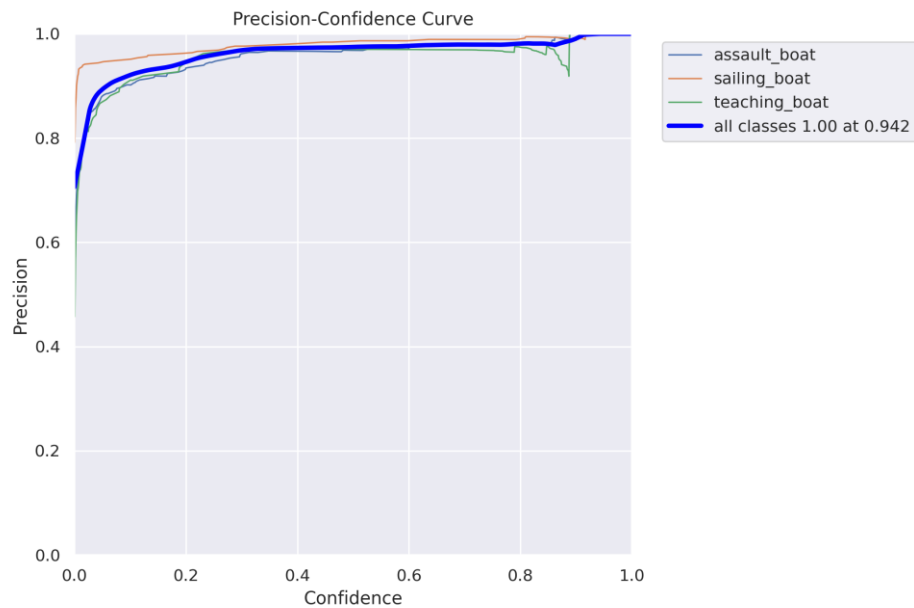


Figura 4-3 Gráfica que muestra el *precision* para cada clase de objeto detectado (autoría propia)

Respecto al parámetro *recall*, la Figura 4-4 muestra los resultados para este entrenamiento. Teniendo en cuenta que es su capacidad de acierto en la detección respecto al total que debería haber detectado para esa clase, se aprecia un ligero decremento en su punto de partida, siendo compensado en la fase final. Se puede afirmar que las curvas descritas para los tres tipos de clases presentan una mayor constantancia, y además, el modelo muestra una mayor consistencia en el intervalo de confianza 0,6 – 0,8 para las clases *assault_boat* y *teaching_boat*, respecto al reflejado en la Figura 3-25 del entrenamiento optimizado en *Codium*.

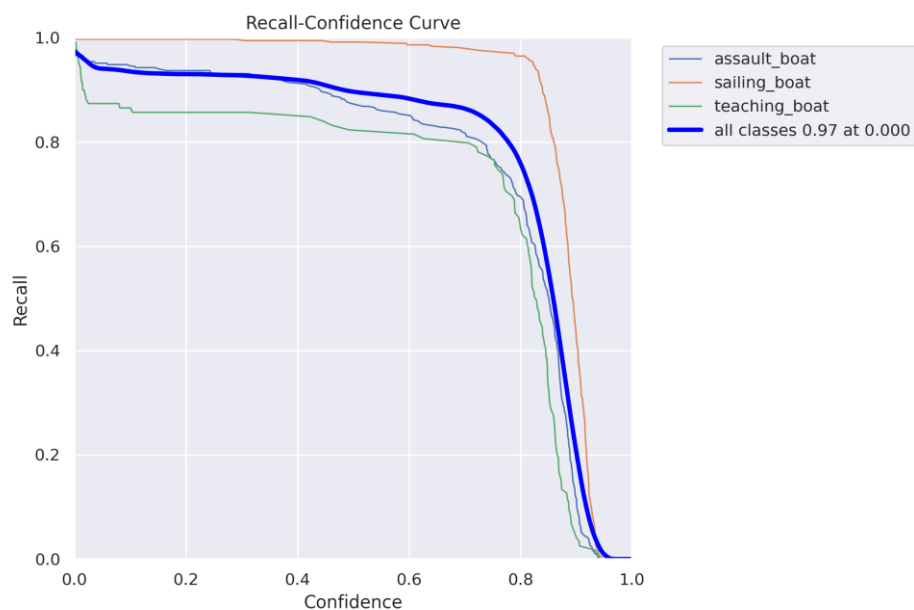


Figura 4-4 Gráfica que muestra el *recall* para cada clase de objeto detectado (autoría propia)

Para finalizar con el análisis paramétrico, en la Figura 4-5 se expone nuevamente la curva *precisión-recall* descrita por este modelo. Se aprecia que, en términos generales, dicha curva describe una forma muy similar a la descrita en la Figura 3-26, aunque se pueden extraer varios apuntes de este caso. El

primero es la pérdida de capacidad en las curvas propias de las clases *assault_boat* y *teaching-boat*, consecuencia principal de que en este caso el modelo no ha sido reentrenado, sino que ha tomado el *dataset* elaborado y lo ha entrenado con un modelo de YOLOv8 que no había interactuado con este *dataset*. Por otro lado, vemos que el resultado en la clase *sailing_boat* alcanza casi la unidad en ambos ejes.

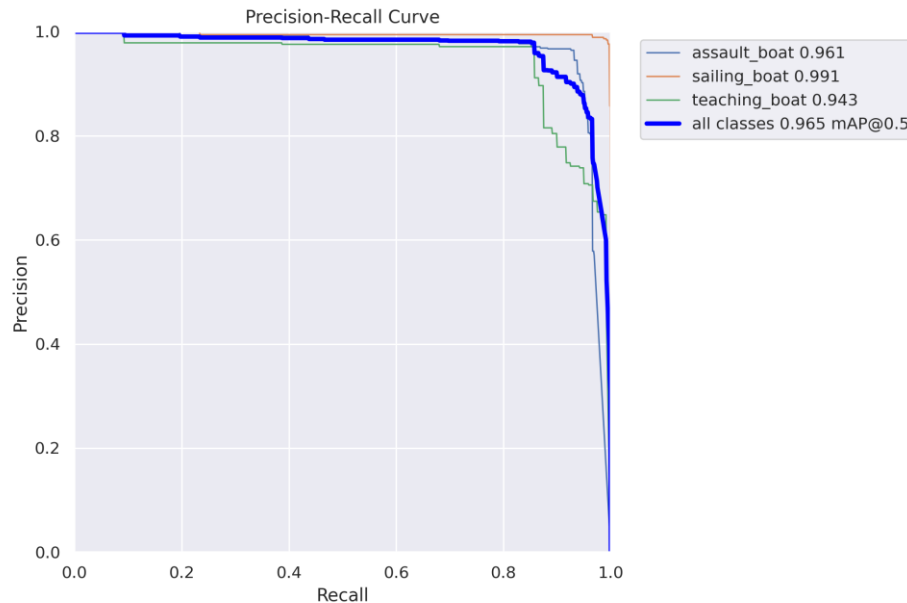


Figura 4-5 Gráfica que representa la curva *precisión-recall* y que relaciona ambos parámetros (autoría propia)

Por último, en la Figura 4-6 se recoge un recopilatorio de algunas imágenes generadas durante el entrenamiento. Aunque se puede deducir tras haber analizado los resultados paramétricos, también visualmente el modelo demuestra un comportamiento positivo en la detección y seguimiento de objetos en videos. Además, en los videos “VIDEO_VALIDACION_YOLOV8S_COLAB_ZONA_A” y “VIDEO_VALIDACION_YOLOV8S_COLAB_ZONA_C” depositados en la dirección que se adjunta a continuación:

https://drive.google.com/file/d/18PEauegymJEn9fVojtnEMcd2iXfLt8k2/view?usp=share_link

https://drive.google.com/file/d/1fyemMJscUC4MZmv88fpsLWorUHHPMQ4n/view?usp=share_link

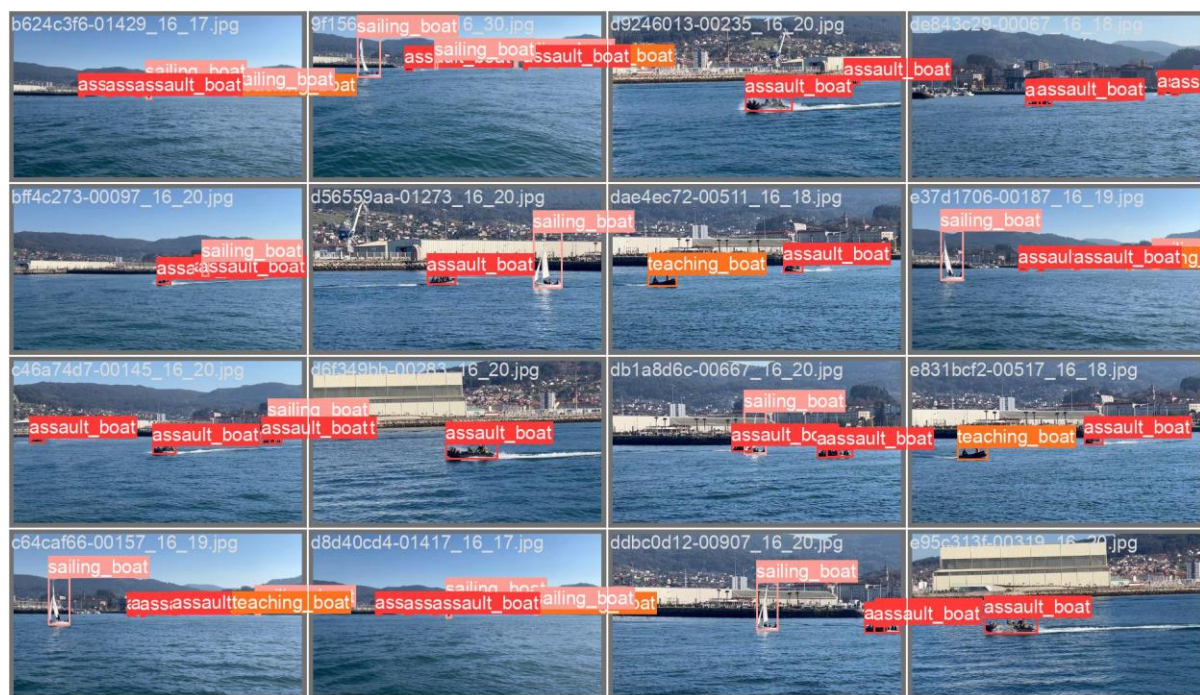


Figura 4-6 Recopilatorio de imágenes generadas por el modelo durante el entrenamiento en entorno Google Colab (autoría propia)

4.4 Resultados y validación general

Tratando de evaluar el grado de validación final que ofrece nuestro modelo y los resultados del mismo, es importante analizar algunas variables que permiten hacerlo de forma más precisa.

En primer lugar, hay que considerar el número de parámetros que tienen en cuenta ambas versiones de YOLO con las que se ha trabajado, YOLOv8n y YOLOv8s, siendo de 3,2 y 11,2 millones de parámetros. Son números que reflejan la compleja composición de las redes neuronales convolucionales utilizadas por nuestro modelo, y que permiten imaginar el grado de especialización de cada una de sus capas.

Respecto al tamaño de la base de datos escogida, nuestro *dataset* elaborado contaba con 300 imágenes de partida. Tras concluir el proceso iterativo, acabó contando con 750 imágenes. Cabe mencionar que los resultados no dependen directamente de dicho tamaño, ya que repercutirá de manera más positiva para un mismo modelo su reentrenamiento, antes que el incremento de imágenes introducidas.

Referente al tamaño de las imágenes del *dataset*, se ha observado que el tamaño asimilable por el modelo para el entrenamiento, varía según la capacidad computacional de la unidad procesadora donde se realiza. En este caso, no ha sido un impedimento para el entrenamiento de nuestro modelo y cumple con un tamaño adecuado al que poseen las imágenes originalmente.

Otra variable que es importante tener en cuenta es la calidad de las imágenes. Una calidad pobre de las imágenes empleadas dará lugar a una mayor dificultad en la especialización de las capas de la red que conforman el modelo. Nuestro modelo en particular está compuesto de imágenes tomadas desde dispositivos de grabación portátil, que en la actualidad cuentan con resoluciones altamente aceptables para acometer este tipo de tareas.

Asimismo, se ha sometido al último modelo entrenado en el entorno Google Colab a una prueba de *tracking*, empleando para su validación un video que no haya visualizado el modelo anteriormente. Dicho video tiene las siguientes particularidades:

- Ha sido obtenido con otro medio de grabación distinto al de las imágenes que conforman el *dataset*, concretamente ha sido con una cámara digital.
- Ha sido realizado un día distinto.
- Se ha mantenido la similitud de los objetos a detectar, respecto a los establecidos en el entrenamiento del modelo.

Este video se denomina “VIDEO_EXTERNO_TRACKING_YOLOV8S_ZONA_B” y se puede consultar en el siguiente enlace:

https://drive.google.com/file/d/1dhFV6ypenlr3VOCFLDV5HhEnZaeMf5PE/view?usp=share_link

Analizando brevemente los resultados de dicho *tracking*, se puede determinar que generalmente son positivos. Realiza el seguimiento de los objetos detectados con relativa solvencia, a pesar de los constantes cambios de zoom y las vibraciones apreciables en el dispositivo de grabación durante la toma de imágenes. Como se puede observar en la Figura 4-7, durante la grabación de este video hay un acercamiento a un objeto que limita la captación total del objeto por parte del modelo, conllevando su equivocación. Esto es interesante de cara a una futura realimentación del *dataset*, quedando patente la necesidad de incrementar la generalización de los posibles perfiles y puntos de vista que caracterizan a los objetos que pretendemos que detecte el modelo.



Figura 4-7 Error mostrado en uno de los videos generados con *tracking* que demuestra las limitaciones del modelo cuando no capta los objetos en su totalidad (autoría propia)

Teniendo en cuenta todos los análisis realizados hasta el momento, se puede concluir que nuestro modelo cumple con unos requisitos más que aceptables para validar su aplicación en el propósito deseado, el de detectar y seguir objetos en entornos marítimos, con imágenes obtenidas mediante dispositivos de grabación portátil.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones extraídas

En el presente TFG ha quedado patente la revolución que está suponiendo la aplicación de técnicas de inteligencia artificial a nuestra vida cotidiana, haciendo especial mención a las técnicas de aprendizaje profundo. Esta revolución permite sustituir muchas de las tareas repetitivas que realiza el ser humano hoy en día, tareas que también son propias del entorno marítimo.

Por lo tanto, la primera conclusión extraída de este TFG es que estos sistemas ofrecen un procesamiento de datos masivo respecto al que ofrecen las personas, multiplicando las capacidades en cuanto a automatismos se refiere. De esta forma, se puede disponer de menos personal dedicado a estas tareas y emplearlo en otras donde sean realmente requeridos, ahorrando esfuerzos en algo que un sistema hará de forma mucho más eficaz.

En términos de capacidad computacional, estos modelos trabajan con millones o incluso billones de parámetros, por lo que es importante tener en cuenta el tamaño de los datos que vamos a utilizar. Esto afectará también en los modelos que empleemos, máxime si pretendemos hacerlo en dispositivos portátiles y de poco almacenamiento, como puede ser Coral AI, ya que limitará en cierto modo las aspiraciones del modelo.

Otra conclusión extraída es la importancia de considerar los condicionantes del entorno, tales como la luminosidad, las condiciones meteorológicas, los contrastes, la diversidad del *dataset*, el *background*, etc. Como se puede deducir, estos sistemas son entrenados para tareas muy concretas y alcanzando grandes niveles de especialización. Entendido esto, hemos podido concluir que esta particularidad resulta en un mayor proceso de aprendizaje del modelo, redundado en su entrenamiento cuantas veces sea necesario.

En cuanto a las tareas acometidas en este trabajo, especialmente aquellas centradas en resolver el problema de la detección de objetos, se han extraído varias conclusiones que son de utilidad para proyectos derivados de este TFG. Una es relativa a los objetos que detecta el modelo. Como se ha apreciado, el modelo únicamente trabajará para detectar aquellos objetos en los que ha sido entrenado, desechando el resto. Esto requiere un preprocesamiento de los datos, en el que se analice qué objetos son objetivos de interés y cuáles no, orientado al diseño de etiquetas que aplicaremos sobre el *dataset* escogido o elaborado. Otro factor a tener en cuenta en la resolución de este problema, ha sido el empleo de zoom en los dispositivos de grabación, concluyendo que el modelo detecta y clasifica mucho mejor cuando es capaz de abarcar la silueta completa del objeto sobre el que ha sido entrenado.

Respecto a la resolución de las tareas de tracking, la aplicación del modelo nos ha permitido arrojar varias conclusiones. La primera es relativa al movimiento de la fuente que observa/graba, detectando que no es determinante si el movimiento es pequeño, y manteniendo una fuerte capacidad de seguimiento. Esto permite ser optimista en cuanto a la aplicación de sistemas de este tipo, no sólo en emplazamientos en tierra, sino también embarcado en plataformas navales. Otra característica común al comportamiento de estos sistemas, es su dificultad para discernir los objetos en situaciones de oclusión, y por tanto mantener su seguimiento. Si bien, por la experiencia adquirida durante el desarrollo, podemos afirmar que es un mal combatible mediante el entrenamiento del modelo, en los *trackings* realizados a videos con los que no ha trabajado éste, los objetos son fácilmente enmascarables en situaciones de oclusión.

Referente al empleo de dispositivos de grabación portátil, podemos concluir que ha supuesto un éxito, permitiendo una adquisición fácil y cómoda para una amplia variedad de usuarios, y generando unos resultados fiables con *datasets* que requieren muy poco preprocesamiento de los datos.

Por último, el impacto que podría tener en algunas de las funciones que desarrolla en la actualidad la Armada. No es difícil elucubrar situaciones en las que sería de gran ayuda contar con sistemas capaces de facilitar la toma de decisiones, y en este aspecto, las técnicas de IA se postulan como la principal herramienta para dotar al personal de una mayor agilidad. Si a esto sumamos que estos sistemas empleen técnicas de aprendizaje profundo, basadas en el caso de las CNN, en operaciones matemáticas, la fiabilidad de los mismos será considerablemente superior a sistemas que utilicen otros medios.

5.2 Objetivos alcanzados

Haciendo un análisis imparcial de los objetivos logrados en este trabajo, en términos generales, se puede considerar positivo el alcance del mismo.

El primer objetivo que se puede dar por satisfecho es el uso de técnicas de *Machine Learning*, teniendo en cuenta la ventaja que supone aprovechar el *transfer learning* de otras redes neuronales, que ha permitido obtener un modelo que cumple con la función deseada. Además, se ha conseguido empleando un *dataset* de elaboración propia, y particular al entorno que nos concierne, que en este caso ha sido el entorno marítimo.

Además, cabe destacar el proceso de aprendizaje al que nos hemos enfrentado, haciendo frente a las vicisitudes que proponen las redes neuronales, obteniendo conclusiones sobre ellas y aplicándolas a la hora de optimizar nuestro modelo.

Otro objetivo a tener en cuenta ha sido el empleo de dispositivos de grabación portátil, como un teléfono móvil o una cámara digital. Si bien ya existen *datasets* de gran tamaño generados por múltiples instituciones en el mundo, en este TFG se demuestra que se pueden resolver tareas de detección y seguimiento de objetos con poco requerimiento material a nuestro alcance, simplemente empleando dispositivos del que puede disponer cualquier usuario actualmente.

Por último, querría mencionar las raíces que asienta el presente TFG, postulándose como base para el desarrollo de futuros trabajos que ahonden aún más en la aplicación de estas técnicas, y su aprovechamiento (especialmente por parte de la Armada) para acometer, de forma más cómoda y segura, muchas de las tareas que se realizan hoy en día en el entorno marítimo.

5.3 Tendencias y líneas futuras

Finalizado el desarrollo del modelo, y analizados los objetivos y conclusiones alcanzados en el presente TFG, este apartado pretende arrojar algo de luz sobre las posibles líneas futuras a las que aspira una continuación del mismo. Por ello, se plantean distintas cuestiones que pueden ser tomadas como base para una mayor explotación de este trabajo, tratando de asentar unas bases que permitan obtener un producto de calidad, y que pueda ser empleado por la Armada en un futuro próximo. Dichas líneas son las siguientes:

- Como se ha observado durante todo el desarrollo del trabajo, una ampliación del tamaño del *dataset* incurre en una mayor fiabilidad y optimización del modelo. Por ello, una línea de desarrollo de este TFG sería alimentar al modelo con un *dataset* más completo, no sólo en tamaño, sino en diversidad, permitiendo alcanzar mayores niveles de generalización. Esto permitiría incrementar exponencialmente las posibilidades de aplicación de nuestro modelo.
- Otro punto de gran interés a implementar es el *keypoint detection*, abriendo la posibilidad a la detección de armas, personas, elementos característicos de los buques, etc. Si bien es atractivo, el mayor impedimento a esta aplicación radicaría en la exigencia en la calidad de las imágenes, lo que, dada la naturaleza del presente TFG, y que aspira a la aplicación de técnicas de aprendizaje profundo con dispositivos de grabación portátil, sería de gran dificultad.
- Continuando con las líneas de desarrollo en el entorno marítimo, otra que sería de relevante aplicación sería la determinación de la posición GPS de los objetos/buques detectados. El principal objetivo de dicha línea sería hacerlo mediante un sistema que permitiera determinar la posición respecto a los elementos que están realizando el *tracking*, combinando los resultados de hacerlo con varios dispositivos portátiles a la vez.
- Por último, como en cualquier trabajo de investigación en el entorno marítimo, es de gran interés la incorporación de datos que faciliten la comprensión de la situación de superficie, permitiendo decodificar situaciones que pueden ser de peligro en la mar. Especialmente, los datos *AIS* (*Automatic Identification System*) que incorporan la mayoría de buques que navegan en el mundo. Esto permitiría incorporar a los buques detectados, datos en tiempo real tales como eslora, manga, rumbo, velocidad, o naturaleza del buque, combinándolo con el seguimiento de los mismos.

En definitiva, las técnicas de *Machine Learning* están llamadas a ser claves en el desarrollo de sistemas que automaticen muchas de las tareas repetitivas que se realizan hoy en día en la Armada, y en el mundo marítimo en general. Como hemos visto, su desarrollo y aplicación no están exentos de vicisitudes, y su optimización requiere de mayor capacidad computacional a medida que pretendemos obtener modelos más competentes.

6 BIBLIOGRAFÍA

En esta sección figurarán todas las referencias, sean recursos web, libros, artículos, etc., incluyendo la información de autores, título de la obra, nombre de la publicación, año, edición y enlace más fecha de último acceso en el caso de referencias a recursos online.

- [1] Almirante Jefe de Estado Mayor de la Armada, «Armada Española,» 11 enero 2022. [En línea]. [Último acceso: 25 enero 2023].
- [2] G. Boesch, «viso.ai,» 2023. [En línea]. Available: <https://viso.ai/edge-ai/edge-ai-applications-and-trends/>. [Último acceso: 5 febrero 2023].
- [3] «Noticias Parlamento Europeo,» 8 septiembre 2020. [En línea]. Available: <https://www.europarl.europa.eu/news/es/headlines/society/20200827STO85804/que-es-la-inteligencia-artificial-y-como-se-usa>. [Último acceso: 25 enero 2023].
- [4] CICE, «Escuela Profesional de Nuevas Tecnologías,» [En línea]. Available: https://www.cice.es/blog/articulos/historia-evolucion-la-inteligencia-artificial/#inicio_contenido. [Último acceso: 26 enero 2023].
- [5] «Telefónica Tech - Think Big/Empresas,» 14 febrero 2020. [En línea]. Available: <https://empresas.blogthinkbig.com/domotica/>. [Último acceso: 10 febrero 2023].
- [6] «SAP,» [En línea]. Available: <https://www.sap.com/latinamerica/insights/what-is-machine-learning.html>.
- [7] J. L. Gonzalez, «Medium,» 8 febrero 2018. [En línea]. Available: <https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2>. [Último acceso: 20 febrero 2023].
- [8] R. Greyrat, «Barcelona Geeks,» 5 julio 2022. [En línea]. Available: <https://barcelonageeks.com/regresion-y-clasificacion-aprendizaje-automatizado-supervisado/>. [Último acceso: 8 marzo 2023].
- [9] Carlos García Moreno, «INDRA,» [En línea]. Available: <https://www.indracompany.com/es/blogneo/deep-learning-sirve>. [Último acceso: 26 enero 2023].

- [10] Data Science Academy, «¿Qué son las redes neuronales artificiales o Deep Learning?,» de *Deep Learning Book*.
- [11] H. Bandyopadhyay, «V7 Labs,» 02 febrero 2023. [En línea]. Available: <https://www.v7labs.com/blog/what-is-computer-vision>. [Último acceso: 20 enero 2023].
- [12] «Coral,» 2020. [En línea]. Available: <https://www.coral.ai/examples/>. [Último acceso: 26 enero 2023].
- [13] «Qualcom developer network,» [En línea]. Available: <https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk/learning-resources/image-segmentation-deeplab-neural-processing-sdk/classification-object-detection-segmentation>.
- [14] I. Mansouri, «Medium,» Analytics Vidhya, 29 diciembre 2019. [En línea]. Available: <https://medium.com/analytics-vidhya/computer-vision-part-6-semantic-segmentation-classification-on-the-pixel-level-ee9f5d59c1c8>. [Último acceso: 08 marzo 2023].
- [15] Z. Z. E. C. Rui Ma, «Hindawi,» 10 febrero 2021. [En línea]. Available: <https://www.hindawi.com/journals/complexity/2021/6679746/>. [Último acceso: 20 febrero 2023].
- [16] J. Finance, «Inteligia-Artificial.dev,» Formación Europea para LATAM y España, 6 enero 2021. [En línea]. Available: <https://inteligencia-artificial.dev/tipos-redes-neuronales/>. [Último acceso: 2026 enero 2023].
- [17] A. Bhardwaj, «Towards Data Science,» 11 octubre 2020. [En línea]. Available: <https://towardsdatascience.com/what-is-a-perceptron-basics-of-neural-networks-c4cfea20c590>. [Último acceso: 20 enero 2023].
- [18] «IBM,» [En línea]. Available: <https://www.ibm.com/topics/recurrent-neural-networks>. [Último acceso: 10 febrero 2023].
- [19] H. Bandyopadhyay, «V7 Labs,» 3 enero 2023. [En línea]. Available: <https://www.v7labs.com/blog/autoencoders-guide>. [Último acceso: 12 febrero 2023].
- [20] R. Greyrat, «Barcelona Geeks,» 5 julio 2022. [En línea]. Available: <https://barcelonageeks.com/funciones-de-activacion-en-redes-neuronales/>. [Último acceso: 25 enero 2023].
- [21] P. Baheti, «V7 Labs,» 3 enero 2023. [En línea]. Available: <https://www.v7labs.com/blog/convolutional-neural-networks-guide>.
- [22] «Aprende Machine Learning,» 29 noviembre 2019. [En línea]. Available: <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>. [Último acceso: 25 enero 2023].
- [23] Alberto Rizzoli, «V7 Labs,» 19 enero 2023. [En línea]. Available: <https://www.v7labs.com/blog/object-detection-guide>. [Último acceso: 10 febrero 2023].
- [24] «SuperAnnotate,» 12 octubre 2021. [En línea]. Available: <https://www.superannotate.com/blog/yolo-object-detection>. [Último acceso: 10 febrero 2023].
- [25] «Programador clic,» [En línea]. Available: <https://programmerclick.com/article/2325787641/>. [Último acceso: 27 enero 2023].

- [26] «ArcGIS,» [En línea]. Available: <https://developers.arcgis.com/python/guide/how-retinanet-works/>. [Último acceso: 25 enero 2023].
- [27] R. Kondu, «V7 Labs,» 19 enero 2023. [En línea]. Available: <https://www.v7labs.com/blog/yolo-object-detection>. [Último acceso: 20 febrero 2023].
- [28] R. R. Abril, «La máquina oráculo,» [En línea]. Available: <https://lamaquinaoraculo.com/computacion/deteccion-de-objetos/>. [Último acceso: 20 febrero 2023].
- [29] R. Cañadas, «AbDatum,» 14 febrero 2022. [En línea]. Available: <https://abdatum.com/tecnologia/dataset>. [Último acceso: 15 febrero 2023].
- [30] C. Han, «Hindawi,» 25 noviembre 2021. [En línea]. Available: <https://www.hindawi.com/journals/jat/2021/5808206/>. [Último acceso: 26 enero 2023].
- [31] «atlanTTic,» [En línea]. Available: <https://atlanttic.uvigo.es/underwaternoise/>. [Último acceso: 25 enero 2023].
- [32] «Find Open Data,» Natural Englad, 3 marzo 2022. [En línea]. Available: <https://www.data.gov.uk/dataset/bfc23a6d-8879-4072-95ed-125b091f908a/marine-habitats-and-species-open-data>.
- [33] J. Y. H. Z. C. W. Fenglei Han, «Hindawi,» 25 febrero 2020. [En línea]. Available: <https://www.hindawi.com/journals/mpe/2020/3937580/>. [Último acceso: 5 marzo 2023].
- [34] D. R. L. R. Dilip K. Prasad, «Kaggle - Singapore Maritime Datasets,» 17 noviembre 2016. [En línea]. Available: <https://www.kaggle.com/datasets/mmichelli/singapore-maritime-dataset>. [Último acceso: 25 enero 2023].
- [35] B. Iancu, V. Soloviev, L. Zelioli y J. Lilius, «Abo Akademi,» 5 febrero 2021. [En línea]. Available: <https://research.abo.fi/en/publications/aboships-an-inshore-and-offshore-maritime-vessel-detection-datase>. [Último acceso: 20 febrero 2023].
- [36] Redacción, «infodefensa,» 22 marzo 2022. [En línea]. Available: <https://www.infodefensa.com/texto-diario/mostrar/3502925/armada-espanola-apuesta-adquirir-soberania-tecnologica-inteligencia-artificial>. [Último acceso: 16 marzo 2023].
- [37] «Universidad Pontificia Comillas,» 2021. [En línea]. Available: <https://ckh.comillas.edu/comunidad/ckhexplorer/recurso/obtencion-procesamiento-y-analisis-de-imagenes/236a081f-dbcc-4d8d-9ec2-84fc3aaccdce>. [Último acceso: 16 marzo 2023].
- [38] F. L. L. José Daniel González-Aller Lacalle, «Evolución del CESADAR,» *Revista General de Marina*, p. 14, agosto-septiembre 2018.
- [39] J. M. L. Ros, «Inteligencia Artificial aplicada a la guerra de minas navales,» *Revista General de Marina*, p. 14, junio 2022.
- [40] «Foro Naval,» 23 mayo 2022. [En línea]. Available: <https://foronaval.com/2022/05/23/sentinel-2-0-de-escribano-la-primera-estacion-remota-de-armas-rws-operada-por-la-armada/>. [Último acceso: 10 marzo 2023].
- [41] «Ultralytics HUB,» [En línea]. Available: <https://hub.ultralytics.com/signin>. [Último acceso: 15 marzo 2023].

- [42] M. Merino, «Genbeta,» 09 junio 2020. [En línea]. Available: <https://www.genbeta.com/desarrollo/vscodium-version-vs-code-libre-telemetry-microsoft>. [Último acceso: 09 marzo 2023].
- [43] J. Świeżewski, «Appsilon,» 22 mayo 2020. [En línea]. Available: <https://appsilon.com/object-detection-yolo-algorithm/>. [Último acceso: 20 febrero 2023].
- [44] R. Kundu, «V7 Labs,» 02 marzo 2023. [En línea]. Available: <https://www.v7labs.com/blog/yolo-object-detection>. [Último acceso: 20 febrero 2023].
- [45] S. Rath, «LearnOpenCV,» 10 enero 2023. [En línea]. Available: <https://learnopencv.com/ultralytics-yolov8/>. [Último acceso: 8 marzo 2023].
- [46] S. Rath, «LearnOpenCV,» 10 enero 2023. [En línea]. Available: <https://learnopencv.com/ultralytics-yolov8/>. [Último acceso: 09 marzo 2023].
- [47] S. Sah, «Medium,» 9 febrero 2023. [En línea]. Available: <https://siddharthksah.medium.com/real-time-object-tracking-and-segmentation-using-yolov8-with-strongsort-ocsort-and-bytetrack-180eef43354a>. [Último acceso: 23 marzo 2023].
- [48] S. Mandal, «Qure AI,» 08 agosto 2022. [En línea]. Available: <https://blog.qure.ai/notes/byte-track-multi-object-tracking>. [Último acceso: 20 febrero 2023].
- [49] A. Petrovicheva, «OpenCV,» 27 octubre 2020. [En línea]. Available: <https://opencv.org/multiple-object-tracking-in-realtime/>. [Último acceso: 08 marzo 2023].
- [50] R. O. B.-Z. B. Nir Aharon, «School of Electrical Engineering, Tel-Aviv University,» 07 julio 2022. [En línea]. [Último acceso: 8 marzo 2023].
- [51] P. Skalski, «roboflow,» 10 enero 2023. [En línea]. Available: <https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/>. [Último acceso: 23 marzo 2023].
- [52] Simplilearn, «Simplilearn,» 14 febrero 2023. [En línea]. Available: <https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-epoch-in-machine-learning>. [Último acceso: 10 marzo 2023].
- [53] N. Buhl, «Encord,» 7 febrero 2023. [En línea]. Available: <https://encord.com/blog/yolov8-for-object-detection-explained/>. [Último acceso: 21 marzo 2023].
- [54] «Aqua Fundació,» 13 diciembre 2021. [En línea]. Available: <https://www.fundacionaquae.org/wiki/impacto-economico-la-inteligencia-artificial/>. [Último acceso: 17 marzo 2023].
- [55] J. M. García, «La inteligencia artificial tiene un impacto ambiental mucho mayor del que se creía,» *La Vanguardia*, 17 junio 2019.
- [56] «Do Better,» 04 junio 2021. [En línea]. Available: <https://dobetter.esade.edu/es/machine-learning-regulacion>. [Último acceso: 10 marzo 2023].
- [57] N. Adaloglou, «AI SUMMER,» 21 enero 2021. [En línea]. Available: <https://theaisummer.com/cnn-architectures/>.
- [58] J. C. C. Ibáñez, «Armada - Revista General de Marina,» marzo 2021. [En línea]. Available: <https://armada.defensa.gob.es/archivo/rgm/2021/03/RGM03cap10.pdf>. [Último acceso: 10 marzo 2023].

- [59] Ministerio de Defensa, «Ministerio de Defensa,» [En línea]. Available: https://www.defensa.gob.es/misiones/en_exterior/actuales/listado/otan-sea-guardian.html. [Último acceso: 10 marzo 2023].
- [60] Ministerio de Defensa, «Ministerio de Defensa,» [En línea]. Available: https://www.defensa.gob.es/misiones/en_exterior/actuales/listado/atalanta.html. [Último acceso: 10 marzo 2023].

ANEXO I: IMPLICACIONES SOCIALES, Y/O ECONÓMICAS, Y/O AMBIENTALES

En este anexo se recogen algunas de las implicaciones sociales, económicas y ambientales que se pueden deducir de este trabajo, así como de la aplicación de técnicas de IA en general. Como se expone a continuación, algunas se pueden prever por la propia naturaleza de estas técnicas, pero hay otras no tan evidentes que también pueden suponer un gran impacto.

En el ámbito económico – industrial, podemos encontrar varias implicaciones fruto de la aplicación de estas técnicas. A diferencia de lo que algunos sectores de la sociedad opinan, la implementación de la IA y el *Machine Learning* no necesariamente resultan en la pérdida de puestos de trabajo, permitiendo la capacitación de los sistemas, y a su vez, el desplazamiento del personal a tareas más estratégicas y que supongan un menor tedio [54]. Algunas de esas implicaciones son las siguientes:

- Una característica propia de estos proyectos es la necesidad de capacidad computacional para llevarlos a cabo, pudiendo requerir una inversión en *hardware* y *software* que no esté al alcance de todos los sectores. En el caso particular de la Armada, esto podría verse afectado por la adaptación del presupuesto económico propio de la época.
- Otra implicación es el requerimiento de personal cualificado para desarrollar estas técnicas. Si bien en el sector privado no es tan difícil encontrarlo, en organismos públicos o instituciones como la Armada, donde convive tecnología de distintas épocas, es necesario adaptar la formación del personal para adaptarlo a estas nuevas técnicas. La necesidad de formación en el personal, deriva en un evidente incremento de los costes.
- Por último, los costes derivados de ser procesos no depurados. Hay que ser conscientes de que es una tecnología emergente, y por ello sus procesos muchas veces no cuentan con el nivel de validación que ya tienen las tecnologías asentadas. Por poner un ejemplo en el entorno marítimo, podríamos imaginar una situación en la que un sistema producido con técnicas de *Deep Learning*, fuera el encargado de organizar el tráfico de acceso a un puerto. En caso de darse situaciones de colisión o conflicto, los armadores deberán afrontar altos costes, por no mencionar la responsabilidad penal, la cual sería difícil de concluir.

Por otro lado, en el aspecto medioambiental, existen muchas implicaciones que son difíciles de dilucidar en un primer momento. Como se expone a continuación, aunque muchas de ellas sean positivas, la huella ecológica que generan estos sistemas no es nula, siendo, en algunas ocasiones, mayor a la que generan algunas tecnologías que están en entredicho actualmente.

- Una implicación directa puede ser la mejora en los sistemas de detección de basura y grandes zonas contaminadas en el entorno marítimo. Siendo nuestro modelo entrenado para detectar y hacer seguimiento a buques, también podría hacer lo propio con otros objetos marinos. Esto, además de repercutir en el aspecto ambiental, lo hace en el ámbito de la seguridad, convirtiendo los mares en áreas más navegables y seguras.
- Al igual que la basura puede ser objeto de detección y seguimiento en entorno marítimo, otra de las aplicaciones que se revisó en el estado del arte del presente TFG era la biología marina. En el campo de la fauna marina, estos sistemas pueden suponer otra vía de seguimiento para especies en peligro de extinción, o aquellas que sean objeto de interés para la extracción de conclusiones del ámbito marino.
- Sin embargo, no todas las implicaciones ambientales son positivas. Según investigadores de la Universidad de Amherst (Massachusetts, Estados Unidos), la huella ecológica de los sistemas que emplean IA sería mucho mayor de la esperada [55]. En este estudio se analiza la energía que consumen las unidades que permiten este procesamiento de datos, y arroja el dato de que pueden llegar a emitir 284 toneladas de dióxido de carbono equivalente (CO₂e),

lo que supondría la contaminación que generan 47 ciudadanos españoles en todo el año. También se recoge el impacto ambiental fruto del entrenamiento de redes neuronales, considerando los costes de *hardware*, mantenimiento, consumo de aire acondicionado, electricidad, etc. En este caso, pone de ejemplo una red que necesitó ser entrenada durante 274.000 horas, lo que sería equivalente en términos de CO2 a más de 300 vuelos de ida y vuelta entre Madrid y New York.

Por último, no ahondaré en las implicaciones sociales a fin de evitar caer en la redundancia de las implicaciones tratadas en el Anexo II: Reflexiones Éticas y Sociales.

ANEXO II: REFLEXIONES ÉTICAS Y SOCIALES

En este anexo se recoge una breve reflexión sobre las implicaciones ético - sociales del presente TFG. Para ello, se empieza acotando el contexto social de este trabajo como mejora a la seguridad en un entorno tan particular, como es el marítimo. Esto se hace implicando a la Armada, como una de las garantes de la seguridad marítima a nivel nacional, y su adaptación a los tiempos, haciendo frente a los retos y vicisitudes que continuamente se le presentan. Además, se hace un breve análisis de las repercusiones éticas que tienen la aplicación de las técnicas *Machine Learning* en la sociedad actual, dejando claro que no todo está a favor del desarrollo de las técnicas *Deep Learning*.

El compromiso de la Armada con la defensa de España, y particularmente con la seguridad de sus ciudadanos, es un hecho incuestionable. En las últimas décadas, España y la Armada han tenido que afrontar tiempos de bajo presupuesto económico, consecuencia entre otros motivos, de la crisis económica que achacó a todo el mundo en el año 2008. Reitero aun así, que esto nunca fue una excusa, y que el compromiso de la Armada con la seguridad de sus compatriotas ha permanecido intacto. Este compromiso no ha estado exento de esfuerzos, que, en mayor o menor medida, han cambiado los conceptos de Seguridad y Defensa.

Por otro lado, como se puede apreciar a través de operaciones militares europeas como las ya consolidadas EUNAVFOR “*Atalanta*”⁵ o “*Sea Guardian*”⁶, el concepto de seguridad cada vez es de mayor conciencia colectiva, y no tanto individual. Los países, conscientes de que una mayor seguridad colectiva, asegura una mayor seguridad individual, hacen grandes esfuerzos por alcanzar un alto grado de interoperabilidad, asegurando de esta forma la protección de sus intereses, más allá de sus fronteras. La Armada, también consciente de esto, ha cooperado de la manera más eficaz posible en cada momento, adecuando los medios y presupuestos a los tiempos a los que hacía frente.

En este marco, y como se expone en la introducción del presente TFG, la Institución está firmemente comprometida con la transformación digital [1] a todos los niveles, tanto en su Flota, como en sus niveles más administrativos. Esta transformación supone también un gran cambio en el concepto de seguridad, ya que hace uso de técnicas de gran relevancia en la actualidad, como la Inteligencia Artificial o el *Machine Learning*, para obtener sistemas más seguros y de mayor fiabilidad, incrementando siempre la capacidad operativa de sus unidades, que tienen como fin último la defensa de España y sus legítimos intereses.

Por ello, este anexo pretende poner en relieve la aplicación de técnicas de aprendizaje profundo desde el punto de vista social, ya que, no estando el entorno marítimo exento de peligros y riesgos, promueve el desarrollo de sistemas que contribuyen a mejorar la situación de seguridad en la mar, sin olvidar que este es uno de los principales cometidos de la Armada.

Por otro lado, desde el punto de vista ético, el desarrollo de este trabajo y concretamente la aplicación de técnicas *Machine Learning*, conllevan una serie de repercusiones que pueden encontrar rechazo por parte de determinados sectores de la sociedad. A continuación se analizan algunas de ellas:

- El primer aspecto a considerar, es el hecho de que los modelos y redes neuronales artificiales (ANN) cada vez carguen con más peso en la toma de decisiones de alto riesgo. El rechazo de este aspecto está infundado en el hecho de que, si bien estos sistemas cada vez actúan con mayor exactitud y precisión, los modelos de los que parten cada vez son de mayor complejidad, siendo más difícil su verificación y validación. Como se ha visto en el desarrollo de este trabajo, la optimización de un modelo requiere de un gran compromiso en

⁵ Operación de la Unión Europea enfocada a la lucha contra la piratería en el Océano Índico. [55]

⁶ Operación de la OTAN, focalizada en el conocimiento del entorno marítimo y con el fin de disuadir y luchar contra el terrorismo en el Mar Mediterráneo. [54]

el proceso iterativo, de forma que su comportamiento final sea adecuado, y por tanto, el deseado.

- Otro punto a analizar, es la interpretabilidad de los objetivos con el que se emplean estas técnicas, siendo muy diferentes si comparamos el entorno público con el privado. Como bien refleja “Do Better” en su artículo “*Machine learning, políticas públicas y bienestar social: retos y soluciones*” [56], al ser una tecnología emergente, sus particularidades son de dominio público, lo que hace muy difícil establecer sistemas que garanticen la seguridad y responsabilidad civil de las organizaciones que emplean estas técnicas. Poniendo un ejemplo, si nos centramos en las organizaciones públicas, su mayor incentivo será el uso de estas técnicas para mejorar los servicios públicos y asignar recursos de una forma más eficiente. Por el contrario, en las organizaciones privadas prima alcanzar la eficiencia y la focalización en el usuario, así como resultados más personalizados, con el fin último del beneficio económico. En definitiva, esta disparidad genera falta de buenas prácticas, equidad, seguridad y privacidad en los procesos que emplean técnicas de aprendizaje profundo.

En resumen, en el plano ético y al igual que el resto de procesos tecnológicos, la aplicación de técnicas de aprendizaje profundo requiere una revisión de la forma en que son entendidas, pretendiendo asegurar la responsabilidad civil y equidad de sus prácticas. Desde la perspectiva social, su empleo por parte de instituciones públicas como la Armada, aseguran un buen empleo, contribuyendo a la mejora de capacidades tan importantes como la defensa de España y sus ciudadanos, y la garantía de seguridad en entornos marítimos.

ANEXO III: GLOSARIO DE SIGLAS Y ACRÓNIMOS

AAU: *Abo Akademi University*

AIS: *Automatic Identification System*

AJEMA: Almirante Jefe de Estado Mayor de la Armada

ANN: *Artificial Neural Network*

CAPTCHA: *Completely Automated Public Turing test to tell Computers and Humans Apart*

CNN: *Convolutional Neural Network*

CESADAR: Centro de Supervisión y Análisis de Datos de la Armada

CIA2: Centro de Inteligencia Artificial de la Armada

FPN: *Feature Pyramid Network*

HOG: *Histograms of Oriented Gradients*

IA: Inteligencia Artificial

IoU: *Intersection over Union*

mAP: *mean Average Precision*

NMS: *Non-Maximum Suppression*

OGP: Oficial de Guardia en Puente

ResNet: *Residual Network*

RNN: *Recurrent Neural Network*

SORT: *Simple Online and Real-time Tracking*

SSD: *Single Shot Detector*

XOR: Puerta exclusiva OR

YOLO: *You Only Look Once*

ANEXO IV: SCRIPT PARA IMPORTACIÓN DE IMÁGENES

```
# -*- coding: utf-8 -*-
import sys, traceback, os, shutil
from ultralytics import YOLO
from configuration import configure
from functions import video_to_images

##### TRACK CONFIGURATION #####

IMPORT_VIDEO_NAME = "16_21_A_MOVIL.MOV"
IMPORT_IMAGE_NAME = "16_21_A_MOVIL"
IMPORT_IMAGE_SIZE = 800
IMPORT_FRAME_START = 88
IMPORT_FRAME_INTERVAL = 10
IMPORT_NUM_IMAGES_MAX = 200
IMPORT_MODEL_NAME = "lanchas_infantes_yolo8n_550.pt"
IMPORT_SAVE_ANNOTATIONS = True

#####

if __name__ == '__main__':
    try:
        # Configure
        configuration = configure()

        # Create and train model
        video_path = configuration['videos_dataset_path'] + IMPORT_VIDEO_NAME
        print('video_path=', video_path)
        annotation_model_path = configuration['models_path'] +
configuration['model_version'] + "/" + IMPORT_MODEL_NAME
        images_path = configuration['images_import_path'] + IMPORT_IMAGE_NAME + "/"
        print('images_path=', images_path)
        if (os.path.isfile(images_path)):
            shutil.rmtree(images_path)
        os.makedirs(images_path, exist_ok=True)
        video_to_images(video_path = video_path, images_path = images_path,
tasks_path = configuration['tasks_label_studio_path'],
        image_base_name = IMPORT_IMAGE_NAME, frame_interval =
IMPORT_FRAME_INTERVAL, frame_start = IMPORT_FRAME_START,
        num_images_max = IMPORT_NUM_IMAGES_MAX, save_annotations =
IMPORT_SAVE_ANNOTATIONS, annotation_model_path = annotation_model_path, labels
= configuration['labels'],
        image_size = IMPORT_IMAGE_SIZE)
    except KeyboardInterrupt:
        print('KEYBOARD INTERRUPT !!! ', str(traceback.format_exc()))
        sys.exit(0)
    except:
        print('MAIN ERROR : ', str(traceback.format_exc()))
```

```
sys.exit(0)
```

ANEXO V: SCRIPT PARA TRACKING

```
# -*- coding: utf-8 -*-
import sys, traceback
from ultralytics import YOLO
from configuration import configure

##### TRACK CONFIGURATION #####

TRACK_VIDEO_NAME = "16_23_A_MOVIL.MOV"#"16_20_A_MOVIL.MOV"
TRACK_SAVE_NAME  = "16_23_A_MOVIL_YOLO8N_400" #"16_20_A_MOVIL_YOLO8N_391"
TRACK_SAVE_RESULTS = True # False #
TRACK_MODEL_NAME  = "lanchas_infantes_yolo8n_400.pt"
#"lanchas_infantes_yolo8n_355.pt"
TRACK_TRACKER_NAME = "bytetrack.yaml"#"botsort.yaml"#
TRACK_IMAGE_SIZE  = 800

#####

if __name__ == '__main__':
    try:
        # Configure
        configuration = configure()

        # Create model
        model_path = configuration['models_path'] + configuration['model_version'] +
        "/" + TRACK_MODEL_NAME
        #print('model_path=', model_path)
        model = YOLO(model_path)

        # Track video
        track_video_path = configuration['videos_dataset_path'] + TRACK_VIDEO_NAME
        #print('track_video_path=', track_video_path)
        model.track(source = track_video_path, show = True, save =
TRACK_SAVE_RESULTS, project = configuration['path'] + "RUNS/TRACK/",
name = TRACK_SAVE_NAME, imgsz = TRACK_IMAGE_SIZE,
tracker=TRACK_TRACKER_NAME)
    except KeyboardInterrupt:
        print('KEYBOARD INTERRUPT !!! ', str(traceback.format_exc()))
        sys.exit(0)
    except:
        print('MAIN ERROR : ', str(traceback.format_exc()))
        sys.exit(0)
```


ANEXO VI: SCRIPT PARA EXPORTACIÓN DE IMÁGENES

```
# -*- coding: utf-8 -*-
import sys, traceback, time, os, glob, shutil, zipfile
import numpy as np
from configuration import configure

##### TRACK CONFIGURATION #####
EXPORT_PROJECT_NAME = "project.zip"

#####

if __name__ == '__main__':
    try:
        print('***** EXPORT LABEL STUDIO TO YOLO
*****')

        # Configure
        configuration = configure()

        # Read images
        images = glob.glob(configuration['images_label_studio_path'] + '*.jpg')
        num_images = len(images)

        if num_images == 0:
            print ('Formating from Label Studio. Empty source images : ' ,
configuration['images_label_studio_path'])
            sys.exit(0)

        print ('Formating from Label Studio. Number of images: ' , num_images)

        # Remove YOLO destiny directory and create new template
        shutil.rmtree(configuration['images_yolo_path'])
        shutil.copytree(configuration['templates_yolo_path'],
configuration['images_yolo_path'])

        # Unzip project zip file
        project_file_path = configuration['export_path'] + EXPORT_PROJECT_NAME
        project_yolo_path = configuration['export_path'] + 'YOLO/'
        os.makedirs(project_yolo_path, exist_ok=True)
        shutil.rmtree(project_yolo_path)

        yolo_labels_path = project_yolo_path + 'labels/'
        #print('project_file_path=',project_file_path)
        #print('yolo_labels_path=',yolo_labels_path)
        #shutil.unpack_archive(project_file_path)
        with zipfile.ZipFile(project_file_path, 'r') as zip_ref:
            zip_ref.extractall(project_yolo_path)
        labels = glob.glob(yolo_labels_path + '*.txt')
```

```
num_labels = len(labels)
print ('Formating from Label Studio. Number of labels: ' , num_labels)
print('=====')
=====')
image_counter = 1
for image_path in images:
    image_name = os.path.basename(image_path)
    label_name = image_name.replace("jpg","txt")
    label_path = yolo_labels_path + label_name

    if not (os.path.isfile(label_path)):
        image_reset_path = configuration['images_reset_path'] + image_name
        print('*****')
        print('LABEL NOT FOUND FOR image_path=', image_path, ' moving to :',
image_reset_path)
        print('*****')
        shutil.move(image_path, image_reset_path)
        continue

    prob = np.random.rand()

    if prob < 0.6:
        target_path = 'train/'
    elif prob < 0.9:
        target_path = 'valid/'
    else:
        target_path = 'test/'

    image_target_path = configuration['images_yolo_path'] + target_path +
'images/'
    label_target_path = configuration['images_yolo_path'] + target_path +
'labels/'

    print('-----', str(image_counter), '-----')
--')
    image_counter += 1
    print('image_path=', image_path)
    #print('label_path=', label_path)

    #print('image_target_path=', image_target_path)
    #print('label_target_path=', label_target_path)

    shutil.copy2(image_path, image_target_path)
    shutil.copy2(label_path, label_target_path)

    # Remove temporal directory
    shutil.rmtree(project_yolo_path)
except KeyboardInterrupt:
    print('KEYBOARD INTERRUPT !!! ', str(traceback.format_exc()))
    sys.exit(0)
```

```
except:  
    print('MAIN ERROR : ', str(traceback.format_exc()))  
    sys.exit(0)
```


ANEXO VII: SCRIPT PARA ENTRENAMIENTO DEL MODELO

```
# -*- coding: utf-8 -*-
import sys, traceback
from ultralytics import YOLO
from configuration import configure

##### TRAIN CONFIGURATION #####
TRAIN_MODEL_NAME = "yolov8n.pt" #"lanchas_infantes_yolo8n.pt"#"yolov8s.pt"
#"lanchas_infantes_yolo8s.pt"
TRAIN_EPOCHS = 800
TRAIN_IMAGE_SIZE = 800
TRAIN_RUN_NAME = "TRAIN_DATASET_600_YOLO8N"
TRAIN_DEVICE = "0" #"cpu" #

#####

if __name__ == '__main__':
    try:
        print('***** TRAIN YOLO8 WITH DATASET *****')

        # Configure
        configuration = configure()

        # Create and train model
        model_path = configuration['models_path'] + configuration['model_version'] +
        "/" + TRAIN_MODEL_NAME
        print('model_path=', model_path)
        model = YOLO(model_path)
        yaml_source = configuration['images_yolo_path'] + "data.yaml"
        #print('yaml_source=', yaml_source)
        model.train(data = yaml_source, epochs = TRAIN_EPOCHS, imgsz =
TRAIN_IMAGE_SIZE, device = TRAIN_DEVICE, patience = 100,
        batch = configuration['train_batch_size'], save = True, project =
configuration['path'] + "RUNS/TRAIN/", name = TRAIN_RUN_NAME)

    except KeyboardInterrupt:
        print('KEYBOARD INTERRUPT !!! ', str(traceback.format_exc()))
        sys.exit(0)
    except:
        print('MAIN ERROR : ', str(traceback.format_exc()))
        sys.exit(0)
```