Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Sistema de tracking visual basado en inteligencia artificial para la evaluación del bienestar animal

Artificial intelligence-based visual tracking system for animal welfare assessment

David Arteaga Sánchez

La Laguna, 14 de julio de 2023

D. **Jonás Philipp Lüke**, con N.I.F. X0581666-L Profesor Contratado Doctor de Universidad adscrito al Departamento de Ingeniería Industrial de la Universidad de La Laguna, como tutor

D. **Fernando Luis Rosa González**, con N.I.F. 43611314-W profesor Titular de Universidad adscrito al Departamento de Ingeniería Industrial de la Universidad de La Laguna, como cotutor

C E R T I F I C A N

Que la presente memoria titulada:

"Sistema de tracking visual basado en inteligencia artificial para la evaluación del bienestar animal"

ha sido realizada bajo su dirección por D. **David Arteaga Sánchez**, con N.I.F. 42266516Z

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

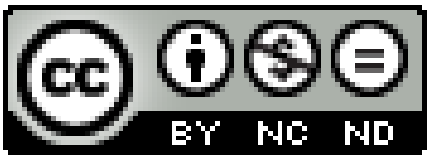
Agradecimientos

Agradezco a la Escuela Técnica Superior de Ingeniería y Tecnología así como a todos los profesores de la misma por la formación impartida durante todos estos años.

Agradezco también a Jonás Philipp Lüke por acompañarme durante todo el proyecto y ayudarme en todo lo que ha estado en su mano para llevarlo a cabo de la manera más satisfactoria posible.

Agradezco finalmente a mis padres y a toda mi familia por costear durante tantos años mis estudios, confiar en mí y apoyarme durante todo este proceso.

Licencia



Resumen

El principal objetivo de este trabajo ha sido garantizar el bienestar animal, en este caso en concreto de los ejemplares de Orcinus orca del famoso zoológico tinerfeño Loro Parque. Para ello planteamos la solución de crear un programa informático que fuera capaz de realizar un seguimiento de estos animales permitiendo así encontrar cualquier tipo de anomalía en su comportamiento.

Para evitar el uso de trackers convencionales decidimos utilizar técnicas de inteligencia artificial, más concretamente redes neuronales convolucionales ya que nos permitían obtener un buen resultado con los pocos datos de entrenamiento con los que contábamos.

Palabras clave: animal, orca, inteligencia, artificial, redes, neuronales, convolucionales

Abstract

The main objective of this work has been to ensure animal welfare, in this particular case of Orcinus orca specimens of the famous Tenerife zoo Loro Parque. For this we proposed the solution of creating a computer program that would be able to track these animals allowing to find any kind of anomaly in their behavior.

To avoid the use of conventional trackers, we decided to use artificial intelligence techniques, more specifically convolutional neural networks, since they allowed us to obtain a good result with the little training data we had.

Keywords: animal, tracker, artificial, intelligence, neural, networks, convolutional

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

El estudio del bienestar animal en un zoológico requiere de una serie de procesos que, entre otros, conllevan el seguimiento de individuos y la detección de patrones de comportamiento que permitan establecer pautas de socialización y conducta relacionadas con el bienestar. Esto requiere de una vigilancia continua y durante largos periodos de la población de animales. La realización de esta tarea de forma manual, puede implicar el visonado una cantidad ingente de grabaciones de vídeo o la presencia directa de un humano en el entorno de los animales.

La adecuada automatización de estos procesos permitiría reducir la cantidad de datos a almacenar, además de poderse abarcar periodos de tiempo mas extensos y además aplicarlo a entornos de mayor superifcie y con un número mayor de animales. Disponer de herramientas automáticas para el análisis y extracción de indicadores de bienestar o de comportamiento en tiempo real ayudaría a gestionar de una forma más adecuada las poblaciones de animales en entornos controlados.

La aparición del redes neuronales convolucionanales y el aprendizaje profundo (deep learning) en los últimos años, ha hecho evolucionar la inteligencia artificial, así como las técnicas de visión por computador. Dada la efectividad de estas técnicas, puede resultar de interés su uso en la automatización de las tareas anteriormente descritas.

En concreto, el problema que se pretende abordar es la monitorización continua mediante cámaras del entorno de las piscinas de Orca Ocean en Loro Parque en Tenerife. En dichas instalaciones habitan 4 cuatro ejemplares del *Orcinus orca*. Las instalaciones se componen de un total de 3 piscinas principales y una piscina médica. Dada la enorme extensión de las instalaciones, la inspección visual directa resulta compleja, por lo que están dotadas de cámaras cenitales que permiten observar y grabar los movimientos de los animales, para una posterior evaluación por parte de diferentes expertos, siendo esta tarea sumamente tediosa y no pudiendo prolongarse durante largos periodos de tiempo. A partir de las grabaciones de video se pueden extraer una serie de indicadores básicos tales como distancias recorridas, velocidades, la estimación de la energía consumida, interacciones entre individuos, tiempo en superficie y de inmersión, entre otros. La simple recolección y realización de estadísticas diarias de estos datos para cada inviduo, son de utilidad para la gestión de la población de animales, pues podrían permitir, junto con indicadores obtenidos por otros sistemas, conocer mejor estado de cada animal y de la población en su conjunto.

1.1. Objetivos

1.1.1. Objetivos generales

El principal objetivo de este trabajo es comprobar la viablidad y el funcionamiento de técnicas basadas en deep learning para realizar el seguimento (tracking) de los ejemplares de *Orcinus Orca* en las instalaciones de Orca Ocean. Este trabajo se limita a estudiar y evaluar técnicas de tracking y detección basadas en deep learning, su integración, así como la implementación de un sistema de almacenamiento de las trayectorias. No se aborará la extracción posterior de parámetros e indicadores a partir de los datos de las trayectorias y recorridos almacenados, ya que deberán abordarse en una fase de desarrollo posterior. Por tanto, el desarrollo de este trabajo se realiza de tal forma que pueda ser integrado con otras herramientas preexistentes, como la clasificación automática de vocalizaciones acústicas, u otras que pudieran añadirse en el futuro.

1.1.2. Objetivos específicos

Los objetivos específicos del desarrollo son los siguientes:

- Estudiar el uso de trackers basados en *deep learning* para hallar uno que se adecue al problema planteado.
- Utilizar modelos neuronales existentes y adaptarlos al problema planteado.
- Evaluar los modelos mediante distintas métricas.
- Portabilizar el código para que sea compatible con la arquitectura de una placa Nvidia Jetson y comprobar el rendimiento en este tipo de sistemas.
- Diseñar e implementar un sistema de almacenamiento que nos permita almacenar los resultados del tracking.
- Integrar la salida del software de seguimiento con el sistema de almacenamiento.

1.2. Metodología

Para alcanzar los objetivos especificados anteriormente se van a definir cuatro etapas en el desarrollo del proyecto:

1. La primera etapa consiste en la extracción y etiquetado de datos. Para ello, se han utilizado diversos vídeos de las cámaras cenitales ubicadas en las piscinas de Orca Ocean de Loro Parque. A partir de estos vídeos, se han obtenido numerosos fotogramas que se han etiquetado manualmente con el objetivo de prepararlos para el entrenamiento del modelo de inteligencia artificial que se utilizará posteriormente.
2. En la segunda etapa, se adapta un modelo preentrenado a los datos que se manejan en el problema, lo que se conoce como *transfer learnig*. Una vez realizado el entrenamiento, se realizarán pruebas para determinar cuál de ellos se adapta mejor al problema planteado.

3. La tercera etapa se basa en el desarrollo de una estructura de almacenamiento que permita recolectar el resultado que arroje el modelo.

4. En la cuarta y última etapa se procede a portabilizar el resultado del proyecto a una placa Jetson para evaluar el rendimiento de la red sobre este tipo de hardware.

1.3. Herramientas utilizadas

1.3.1. Python

El lenguaje principal de todo el proyecto, en el que está escrito la red convolucional y el algoritmo de tracking es Python. Python es un lenguaje interpretado, multiparadigma y multiplataforma de alto nivel que se destaca por su filosofía orientada a la legibilidad del código. Es ampliamente utilizado en el campo de la inteligencia artificial.

1.3.2. TensorFlow

TensorFlow es una biblioteca de software de código abierto desarrollada por Google Brain Team para el aprendizaje automático. Fue lanzado en 2015 y se ha convertido en una de las bibliotecas de aprendizaje automático más populares y utilizadas en la actualidad [1].

La principal característica de TensorFlow es su capacidad para crear modelos de aprendizaje profundo de forma eficiente y escalable. Ofrece una gran variedad de herramientas y funciones que permiten a los desarrolladores construir, entrenar y desplegar modelos de aprendizaje profundo en diferentes plataformas [1].

1.3.3. Anaconda

Anaconda es una distribución de Python que viene con múltiples librerías y herramientas preinstaladas que facilitan la tarea de desarrollar, gestionar y desplegar proyectos y paquetes científicos en diferentes entornos [2].

Anaconda se utiliza principalmente en disciplinas como la ciencia de datos, el aprendizaje automático y la ingeniería de software, ya que ofrece un conjunto completo de herramientas y librerías para trabajar con datos e implementar algoritmos de aprendizaje automático.

En este proyecto se ha utilizado Anaconda para la creación y gestión de entornos virtuales, así como para la instalación de los paquetes necesarios para el correcto funcionamiento de las tecnologías utilizadas.

1.3.4. FFmpeg

FFmpeg es una herramienta de conversión multimedia universal. Es capaz de leer una amplia variedad de entradas, incluidos dispositivos de captura/grabación en directo, filtrarlas y transcodificarlas a un gran número de formatos de salida [3].

FFmpeg es utilizado por muchos usuarios y desarrolladores debido a su capacidad para trabajar con una gran cantidad de formatos y codecs de audio y video. Además, su uso es gratuito y está disponible en múltiples plataformas, incluyendo Windows, Linux y MacOS.

1.3.5. Nvidia Jetson Xavier

Las NVIDIA Jetson son sistemas embebidos de alto rendimiento diseñados específicamente para aplicaciones de inteligencia artificial y aprendizaje profundo. Estos dispositivos están equipados con potentes módulos de procesamiento central (CPU) y gráficos (GPU), así como aceleradores dedicados para el procesamiento de redes neuronales.

La NVIDIA Jetson Xavier es una de las más avanzadas dentro de la línea Jetson. Utiliza un módulo de procesamiento central ARMv8 de 64 bits combinado con una GPU NVIDIA Volta con 512 núcleos Tensor, lo cual permite realizar cálculos complejos de manera eficiente en paralelo. Además, cuenta con 16 GB de memoria LPDDR4x para manejar grandes cargas de datos y un motor de inferencia de red neuronal acelerado por hardware que permite a los desarrolladores ejecutar rápidamente modelos de IA [4].

1.3.6. labelImg

Se ha utilizado labelImg como programa auxiliar para etiquetar las imágenes y generar el dataset del proyecto en formato YOLO. LabelImg es una herramienta gráfica de anotación de imágenes de código libre programada con Python usando la librería QT.

1.3.7. HDF5

HDF5 (Hierarchical Data Format 5) es un formato de archivo diseñado para almacenar y organizar grandes volúmenes de datos heterogéneos. Es ampliamente utilizado en el campo de la computación científica y la ingeniería para almacenar y compartir datos complejos [5].

HDF5 permite organizar los datos en una estructura jerárquica similar a un sistema de archivos, donde los conjuntos de datos individuales se denominan ”grupos” y pueden contener ”datasets” (conjuntos de datos) y ”atributos”. Los datasets pueden ser multidimensionales y admitir una amplia gama de tipos de datos, como enteros, flotantes, cadenas, etc [5].

1.3.8. SWIG

SWIG (Simplified Wrapper and Interface Generator) es una herramienta que se utiliza para generar automáticamente interfaces entre lenguajes de programación, como C/C++ y Python. La finalidad principal de SWIG es facilitar la integración de bibliotecas escritas en C/C++ con aplicaciones escritas en Python [6].

SWIG toma como entrada archivos de interfaz que contienen declaraciones de funciones y estructuras en el lenguaje de origen (C/C++) y genera código de envoltura que permite utilizar estas funciones y estructuras desde Python.

El proceso de generación del wrapper implica la creación de un módulo de extensión de Python que proporciona una interfaz accesible y utilizable para las funciones y tipos definidos en el código fuente original. El código de envoltura generado por SWIG maneja la conversión automática de tipos de datos entre los dos lenguajes.

Capítulo 2

Técnicas de inteligencia artificial para el tracking visual

El seguimiento de objetos, más popularmente conocido como Multiple Object Tracking (MOT) es una tarea que cobra vital importancia en la rama de la visión por computador. Se basa en localizar múltiples objetos y mantener sus identidades recibiendo como entrada un vídeo [7].

El proceso que sigue el MOT puede desglosarse en dos partes claramente diferenciadas:

- **Detección**: Es la técnica basada en identificar y localizar los objetos de interés en cada frame del vídeo de entrada.
- **Tracking**: Se basa en asignar un ID único a cada objeto detectado permitiendo la diferenciación entre una detección y otra. Adicionalmente en esta etapa también se realiza una predicción de trayectorias en la que se estima la localización de la detección en frames futuros, con el objetivo de no perder la referencia.

En la sección 2.2, se describrá en mayor detalle el proceso y las técnicas de detección. Seguidamente en la sección 2.3 se describirán distintas técnicas de tracking. Sin embargo, para facilitar la comprensión de las siguientes secciones, en la sección 2.1,se introducirán brevemente las redes neuronales ariticiales.

2.1. Redes neuronales artificales

Una red neuronal artificial es un conjunto de nodos llamados neuronas artificiales que están conectados entre sí para transmitirse señales. Las neuronas artificiales se utilizan para procesar información de entrada y producir una salida. En su forma más básica, una neurona artificial recibe una serie de entradas $(x_1, x_2, ..., x_n)$, que son multiplicadas por unos pesos correspondientes $(w_1, w_2, ..., w_n)$. A continuación se realiza la suma ponderada a través de una función suma (Σ). Además se suma un término adicional conocido como "bias"(b). Finalmente, el resultado se pasa a través de una función no lineal llamada función de activación (ϕ), que produce la salida de la neurona [8]. Esto último permite modelar relaciones no lineales entre las entradas y las salidas de la red. En la figura 2.1, se puede observar una representación gráfica de este proceso.

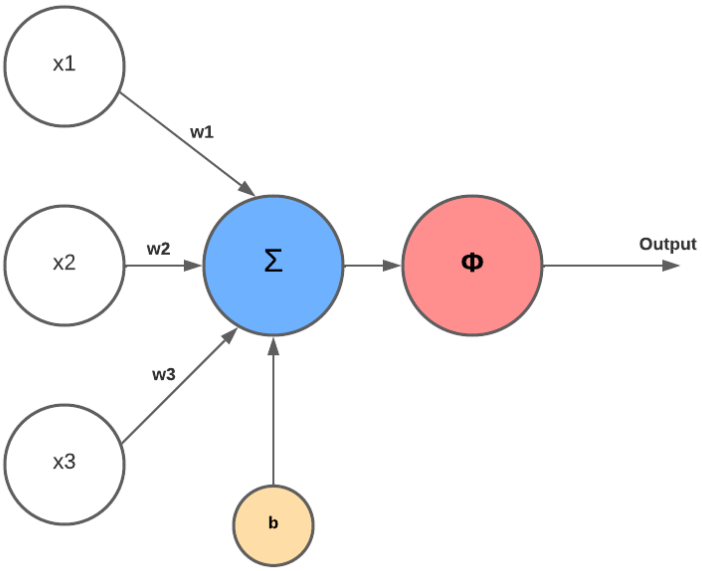


Figura 2.1: Diagrama de la estructura de una nuerona artificial

Las neuronas artificiales se organizan en capas para formar una red neuronal. Cada capa está compuesta por un conjunto de neuronas, como la descrita anteriormente.

Estas capas se conectan entre sí a través de conjuntos de conexiones, que se utilizan para transmitir información desde una neurona a otra. Cada neurona en una capa recibe entradas de las neuronas en la capa anterior y las pondera con los pesos asociados a la conexión correspondiente. Esto se puede observar con mayor detalle en la figura 2.2.

La primera capa, conocida como capa de entrada, recibe los datos de entrada de la red neuronal mientras que la última capa, conocida como capa de salida, produce la respuesta final de la red neuronal.

Las capas intermedias (o capas ocultas) se utilizan para procesar la información y extraer características relevantes del conjunto de datos de entrada. Cada capa oculta puede contener múltiples neuronas, y cada neurona en una capa oculta puede estar conectada a muchas neuronas en la capa anterior y muchas neuronas en la capa siguiente (figura 2.2). Esto permite que la red aprenda características complejas y abstracciones de los datos de entrada.

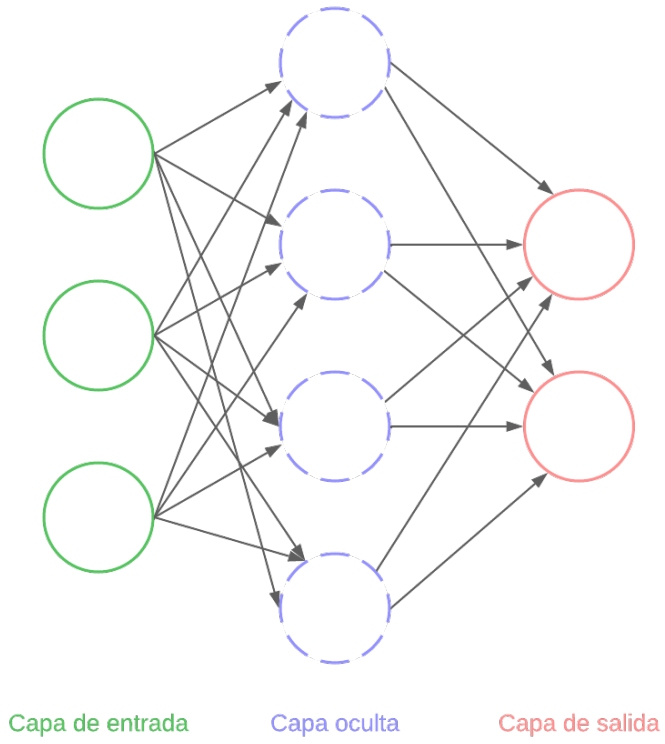


Figura 2.2: Ejemplo de red neuronal artificial

Para que una red neuronal artificial funcione correctamente, es decir, que dado una entrada la red proporcione el valor de salida esperado, debe llevarse a cabo un proceso que se conoce como entrenamiento, en el que se ajustan los valores de los pesos, w, para que la red aporte la salida deseada. Para ello, se debe recurrir a lo que se conoce como función de pérdida, que permite evaluar cómo de bien está funcionando la red. En el desarrollo de redes neuronales artificiales se pueden emplear distintos tipos de funciones de pérdida [9]. El ajuste de los pesos o entrenamiento se realiza mediante un algoritmo que se conoce como retropropagación o *backpropagation* [10].

2.1.1. Redes neuronales convolucionales

Una red neuronal convolucional (CNN) es una red neuronal artificial que pretende emular el comportamiento de la corteza visual primaria de un cerebro biológico. Es el tipo de red más utilizado en el campo de la visión por computador [11].

Las CNN están diseñadas específicamente para trabajar con datos de entrada que tienen una estructura jerárquica, como las imágenes, donde cada píxel está conectado a los píxeles cercanos para formar un patrón más grande.

Las CNN se componen de distintos tipos de capas, como capas de convolución, capas de pooling y capas completamente conectadas. Un ejemplo de una arquitectura de CNN simplificada puede verse en la figura 2.3 [12].

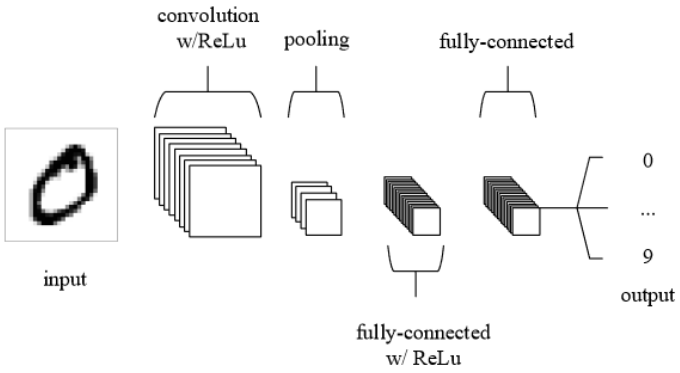


Figura 2.3: Ejemplo arquitectura de una Red neuronal convolucional [12]

La capa de convolución es la parte fundamental de una CNN ya que convoluciona los filtros o kernels a lo largo de la imagen de entrada para extraer características importantes, como bordes y texturas. Cada kernel es una matriz de números que se aplica de forma repetida sobre toda la imagen para producir una nueva imagen convolucionada [12].

Después de la capa de convolución, se usa una capa de pooling para reducir el tamaño de la imagen resultante. Esta capa agrupa regiones de la imagen y toma el valor máximo o promedio dentro de cada región para producir una nueva imagen más pequeña. La capa de pooling no solo reduce la cantidad de parámetros y cálculos necesarios en la red, sino que también ayuda a prevenir el sobreajuste o sobreentrenamiento de la red, ya que funciona como un regularizador [12].

Finalmente, las capas completamente conectadas procesan la información de las características extraídas por las capas anteriores para realizar la clasificación o regresión deseada. Una vez que se entrena una CNN en un conjunto de datos determinado, la red puede reconocer objetos, detectar características y clasificar nuevas imágenes con altos niveles de precisión [12].

En comparación con las redes neuronales tradicionales, las CNN son altamente eficientes para procesar datos de entrada de alta dimensionalidad, como imágenes y videos, ya que pueden aprovechar la estructura espacial de los datos de entrada. Esto es debido a que en una red convolucional, los pesos de la capa de convolución se comparten entre todas las unidades de la capa. Esto significa que cada unidad en la capa de convolución utiliza el mismo conjunto de pesos para procesar diferentes partes de la entrada. Por ejemplo, en una red convolucional para procesar imágenes, cada filtro de convolución se aplica a toda la imagen con el objetivo de detectar patrones en cualquier parte de ella.

Este enfoque tiene varias ventajas. En primer lugar, reduce significativamente el número de parámetros requeridos para entrenar la red, lo que resulta en modelos mucho más eficientes en términos de memoria y capacidad computacional. En segundo lugar, al compartir parámetros, la red convolucional puede aprender patrones similares en diferentes partes de la entrada, lo que mejora su capacidad para generalizar y detectar características en nuevas imágenes. [12].

2.2. Detección de objetos

Como se mencionó anteriormente, la detección de objetos es una técnica en el campo de la visión artificial que consiste en identificar y localizar objetos específicos en una imagen. Esta tarea se puede abordar mediante técnicas tradicionales o mediante técnicas basadas en redes neuronales artificiales. Por poner algunos ejemplos, entre los métodos tradicionales, cabe mencionar los siguientes:

- Scale Invariant Feature Transform (SIFT): es un algoritmo de visión por computadora utilizado para detectar y describir características invariantes a la escala en imágenes digitales. Fue propuesto por primera vez por el investigador David Lowe en 1999 [13].

  El objetivo principal de SIFT es encontrar puntos clave o "features"en una imagen que sean invariantes a la escala, la rotación y la iluminación. Estos puntos clave son seleccionados por su estabilidad y capacidad de ser reproducidos en diferentes imágenes.

- Speeded Up Robust Features (SURF): es un algoritmo de visión por computadora utilizado para detectar y describir características invariantes en imágenes digitales. Fue propuesto por Herbert Bay, Tinne Tuytelaars y Luc Van Gool en 2006 [14].

  Al igual que SIFT su objetivo es encontrar puntos clave o "features"en una imagen que sean invariantes a la escala, la rotación y la iluminación. Sin embargo, SURF utiliza una serie de técnicas más eficientes que permiten una detección y extracción de características más rápida.

- Features from Accelerated Segment Test (FAST): Es un algoritmo de detección de características en imágenes desarrollado por Edward Rosten y Tom Drummond en 2006, que se basa en la detección de cambios de intensidad en ciertos puntos clave. En lugar de analizar la imagen completa, el algoritmo busca una serie de puntos candidatos a ser características relevantes para la imagen. Estos puntos se identifican por su alta variación de intensidad en comparación con sus vecinos, lo cual es comunmente conocido como una esquina [15].

Sin embargo, en los últimos años la detección de objetos ha dado un salto cualitativo considerable debido al uso de redes neuronales convolucionales. Esto se logra mediante algoritmos de aprendizaje automático, que son entrenados para reconocer patrones y características distintivas de los objetos de interés. Se busca encontrar la ubicación y forma del objeto en la imagen, así como clasificarlo en una categoría previamente definida.

Los algoritmos más conocidos para la detección de objetos que utilizan una CNN son:

- Detección a partir de la clasificación: Se itera un área reducida dentro de la imagen y a cada área se le aplica la CNN para ver si logra alguna detección. Esta iteración se realiza hasta haber analizado la imagen en su totalidad. Este método tiene diversos inconvenientes como por ejemplo elegir el tamaño del área de la iteración, el tiempo de cómputo para realizarlo y la posibilidad de que dos objetos similares se encuentren cerca [16].

- R-CNN: Las R-CNN (Region Based Convolutional Neural Networks) surgen con la propuesta de: primero determinar regiones de interés dentro de la imagen (este proceso es conocido como selective search) y luego realizar la clasificación de imágenes sobre esas áreas utilizando una red pre-entrenada [16].

  Para procesar una imagen, se utiliza un algoritmo inicial que identifica las áreas de interés. Luego, estas regiones se someten a un análisis de clasificación mediante una red neuronal convolucional y un clasificador binario. De esta forma, se puede validar si las regiones pertenecen a las clases correctas y eliminar aquellas con baja confianza. Finalmente, un regresor se encarga de ajustar automáticamente la posición de cada localización para obtener los mejores resultados posibles [16].

- Mask R-CNN: Utiliza una red neuronal convolucional para detectar objetos en una imagen y luego utiliza otra CNN para generar una máscara que identifica la región exacta de la imagen donde se encuentra el objeto. La máscara se usa para aislar el objeto del fondo y proporciona una mayor precisión en la detección de los bordes y contornos del objeto. Este modelo ha sido muy popular en aplicaciones como el reconocimiento facial y la detección de objetos en imágenes médicas, entre otros casos de uso [16].

- Detección rápida con YOLO: YOLO (You Only Look Once) fue creado por Joseph Redmon, junto con sus compañeros en la Universidad de Washington, en el año 2016.

  El objetivo de YOLO es realizar la detección de objetos en una sola pasada a la imagen. Esto lo consigue dividiendo la imagen en una cuadrícula de tamaño fijo y aplicando la CNN a cada cuadro [16].

2.2.1. YOLO

YOLO (You Only Look Once) es un sistema de código abierto para la detección de objetos en tiempo real basado en deep learning desarrollado por Joseph Redmon en 2016. [17].

Se utiliza principalmente en la visión por computador y se basa en una red neuronal convolucional, cuya arquitectura se muestra en la figura 2.4, para identificar y localizar objetos en imágenes. A diferencia de otros algoritmos que requieren múltiples pasadas a través de una imagen, YOLO es capaz de detectar objetos en una sola pasada, lo que lo hace muy rápido y eficiente en términos de recursos de hardware.

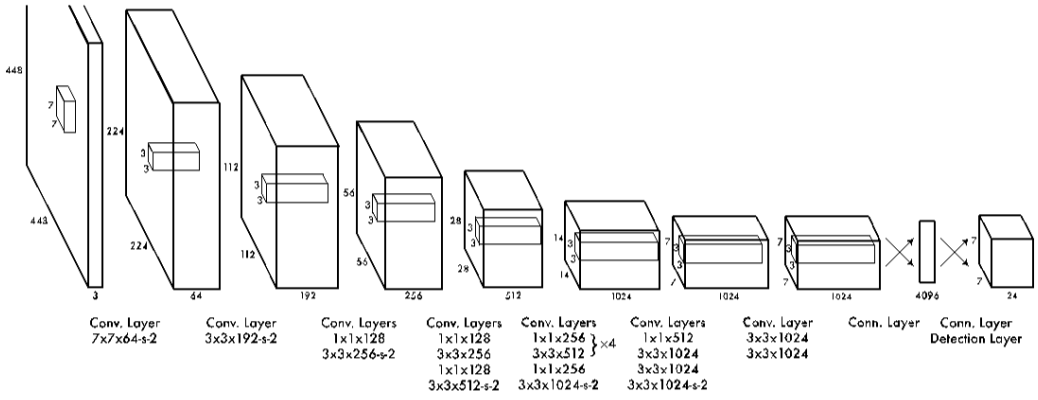


Figura 2.4: Estructura de la red convolucional que utiliza YOLO [18]

El proceso que realiza YOLO para la detección de objetos de una imagen es:

1. Redimensión de la imagen: YOLO recibe como entrada una imagen y la redimensiona a 448×448.

2. División de la imagen en rejillas: Comienza la principal característica distintiva de YOLO. Para llevar a cabo la detección se divide la imagen en una cuadrícula de $S \times S$ celdas (figura 2.5).

Figura 2.5: Primera etapa de la detección con YOLO [19]

3. Extracción de características: Luego, se utiliza una red neuronal convolucional para extraer las características de cada celda.

4. Predicción de cajas delimitadoras y confianza: Para cada celda, se realizan predicciones sobre las cajas delimitadoras (bounding boxes) que contienen los objetos y la confianza de que estas cajas sean correctas. Las cajas delimitadoras son rectángulos que rodean los objetos detectados y se definen por cuatro coordenadas: la posición, (x, y), del centro de la caja y su anchura y altura, (w, h). La confianza es un valor entre 0 y 1 que indica cómo de seguro está el modelo de que hay un objeto en esa celda y que la caja delimitadora es precisa. Para cada celda se predicen B cajas delimitadoras por lo que en total se calculan $S \times S \times B$ cajas (figura 2.6).

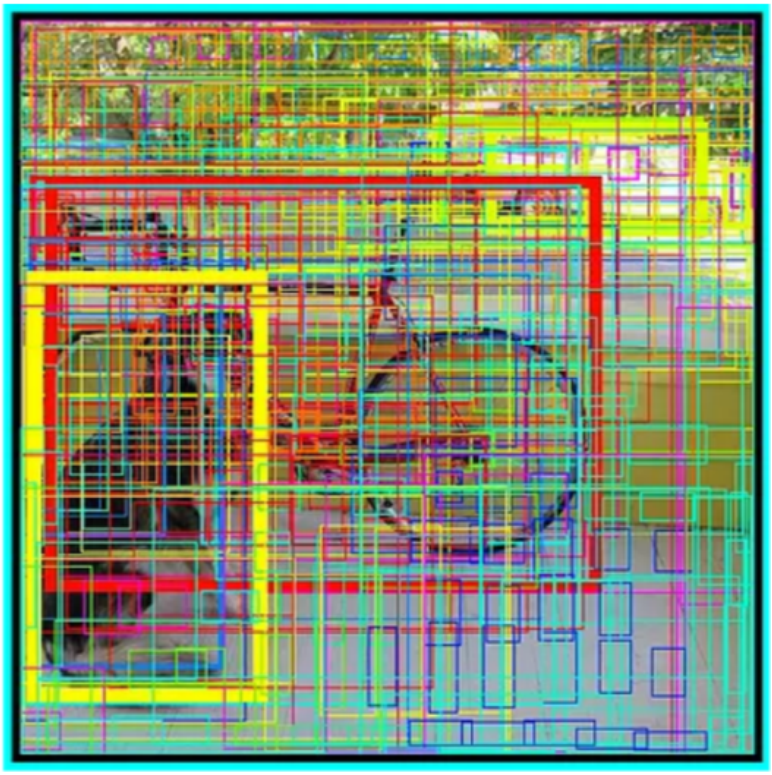


Figura 2.6: Segunda etapa de la detección con YOLO [19]

5. Predicción de clases: Para cada celda, también se realizan predicciones sobre las clases a las que pertenecen los objetos detectados. Estas predicciones se representan como un vector de probabilidades, donde cada elemento representa la probabilidad de que el objeto pertenezca a una clase específica.

6. Filtro de detecciones débiles: Finalmente, se aplica un filtro para eliminar las detecciones débiles. Esto se hace utilizando un umbral de confianza y eliminando las predicciones cuya confianza es menor a este umbral (figura 2.7). Además, si se detectan varias cajas delimitadoras para el mismo objeto, se selecciona la caja con la mayor confianza.

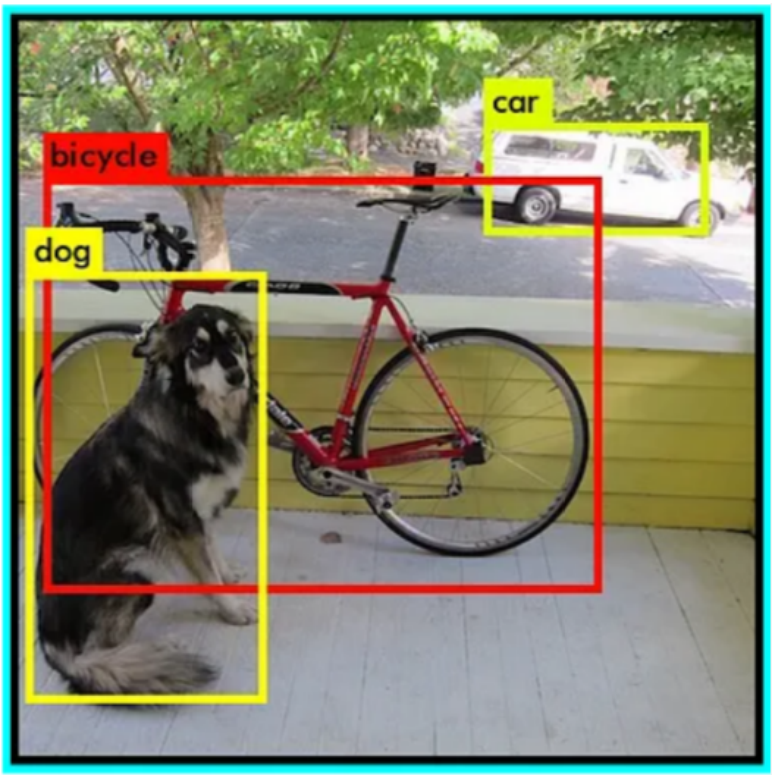


Figura 2.7: Tercera y última etapa de la detección con YOLO [19]

7. Retorno de las detecciones: Una vez realizado el proceso anterior en todas las celdas de la cuadrícula, se obtiene una lista de todas las cajas delimitadoras y las clases predichas para cada objeto detectado. Estas se pueden representar visualmente en la imagen original o utilizar para cualquier otra tarea de procesamiento posterior.

2.2.2. Versiones de YOLO

Existen diversas versiones del algoritmo YOLO por lo que es necesario realizar un estudio de las diferentes alternativas para ver cuál se adapta mejor al proyecto. Las opciones que se barajaron fueron:

- **YOLOv3**: Es la tercera versión oficial de YOLO, desarrollada por su creador original Joseph Redmon y su equipo en la Universidad de Washington y luego en la empresa de inteligencia artificial Yolo AI [20].
- **YOLOv3 tiny**: YOLOv3 Tiny es una versión reducida de la red neuronal utilizada en YOLOv3. A diferencia de la versión completa de YOLOv3, que tiene alrededor de 63 millones de parámetros, YOLOv3 Tiny tiene solo alrededor de 8.5 millones de parámetros. Fue diseñada para su uso en dispositivos con recursos limitados, ya que requiere menos memoria y potencia de procesamiento. Sin embargo, debido a que tiene menos parámetros, la precisión de detección de objetos puede ser ligeramente inferior a la de la versión completa de YOLOv3 [20].
- **YOLOv4**: Es la cuarta y última versión aceptada por la comunidad de la implementación del algoritmo YOLO. Fue creado por un equipo de investigadores liderado por Alexey Bochkovskiy de la Universidad de Washington. YOLOv4 mejora significativamente el rendimiento y la precisión de las versiones anteriores, incluyendo YOLOv3.

Esto se logra a través de varias mejoras técnicas, como el uso de una arquitectura de red neuronal más grande y compleja, así como técnicas avanzadas de aumento de datos y optimización [21].

- **YOLOv4 tiny**: es la versión reducida del algoritmo YOLOv4 que sigue los mismos principios de su versión completa pero con una arquitectura más pequeña y simplificada. Fue diseñado para ser utilizado en dispositivos con recursos limitados. Es el equivalente a YOLOv3 tiny pero con la versión 4 [21].

Aunque en nombre de la comunidad existan más versiones de YOLO posteriores a estas, no se han tenido en cuenta debido a que ninguna es considerada oficial por el mismo creador del algoritmo. Tan solo YOLOv3 y YOLOv4 lo son.

Entre las distintas versiones de YOLO existen diferencias en cuanto a redimiento y velocidad de ejecución, que deben ser consideradas en las aplicaciones desarrolladas que estén basadas en este algoritmo. En la figura 2.8, se muestra una comparativa de las versiones básicas de YOLOv3, YOLOv4 y otras arquitecturas de detección, pudiéndose comprobar que las direrencias son notables.

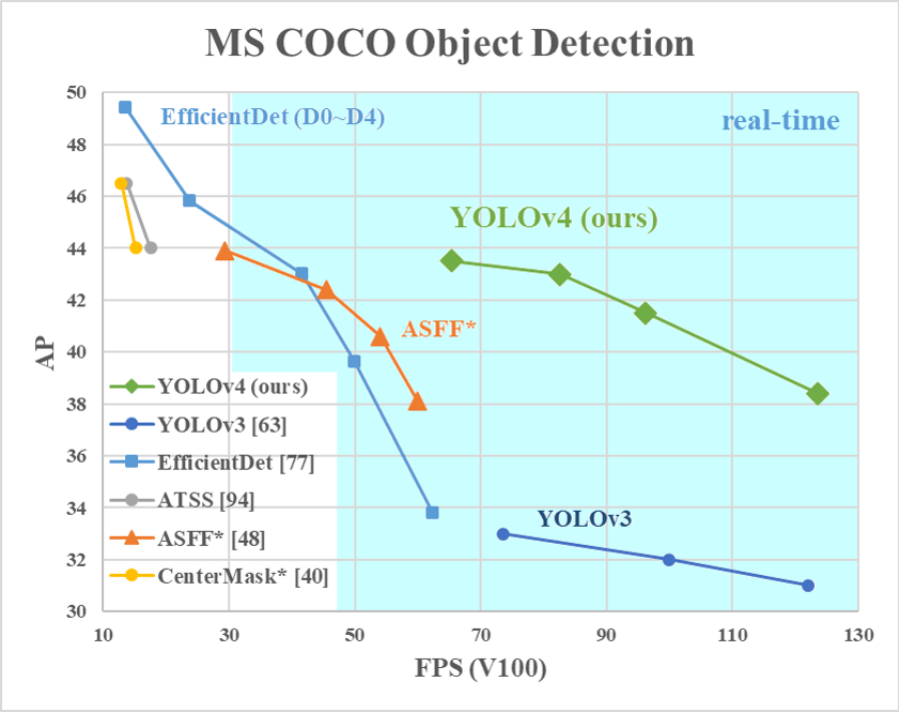


Figura 2.8: Comparación de rendimiento entre versiones de YOLO en una GPU Nvidia Tesla V100 [21]

2.3. Tracking

La segunda etapa del MOT es la etapa del tracking. En este proyecto hemos utilizado una implementación con inteligencia artificial mediante el uso de redes neuronales, sin embargo, tradicionalmente se han usado otros métodos menos sofisticados. Los más conocidos son:

- Mean Shift: Es un algoritmo de seguimiento de objetos. En el proceso se utiliza una ventana deslizante de tamaño fijo para buscar el objeto que se está rastreando.

En cada iteración, el centro de la ventana se mueve hacia el modo de densidad de probabilidad más alto dentro de la ventana actual, lo que resulta en la convergencia de la ventana hacia la posición del objeto. Este proceso se repite hasta que la ventana converge a una posición estable, lo que indica la ubicación del objeto en el fotograma actual [22].

- Optical Flow: Se trata de otra técnica utilizada para el tracking, esta vez se pretende estimar el movimiento aparente de los objetos en una secuencia de imágenes. La técnica funciona observando cómo cambia el brillo de los píxeles en una imagen a medida que se mueven a lo largo del tiempo. A partir de estas variaciones, se puede calcular la dirección y velocidad del movimiento de los objetos en la escena [23].

  Una vez calculado el optical flow, se puede utilizar para actualizar la posición del objeto que se está siguiendo y predecir su ubicación en los siguientes fotogramas.

Como algoritmo de tracking en este proyecto se ha optado por una aproximación con el uso de redes neuronales, en este caso se ha utilizado DeepSort. El nombre de DeepSort proviene de la combinación de dos términos. El seguimiento profundo y el algoritmo SORT (Simple Online and Realtime Tracking). DeepSort es por tanto una extensión del algoritmo de tracking SORT pero añadiendo el uso de seguimiento profundo, es decir, añadiendo deep learning (redes neuronales) al algoritmo [24].

Para poder comprender correctamente el funcionamiento de DeepSort es necesario entender las dos técnicas principales del algoritmo SORT. Estas son:

- **Algoritmo Húngaro.** Es una técnica utilizada comúnmente en la visión por computador. Se utiliza para encontrar la correspondencia óptima entre dos conjuntos de elementos. En el contexto del seguimiento de objetos, los elementos pueden ser las posiciones de los objetos en dos fotogramas diferentes [25].

- **Filtro de Kalman.** Es un algoritmo matemático utilizado para estimar el estado de un sistema dinámico a partir de medidas incompletas, ruidosas o imprecisas. El algoritmo del filtro de Kalman estima el estado actual del sistema y hace una predicción del estado futuro utilizando la información de las mediciones pasadas [26].

Con estas dos técnicas el algoritmo SORT realiza los dos procesos anteriormente comentados en la etapa del tracking: se asigna un identificador único a cada detección mediante el Algoritmo Húngaro y además se realiza una predicción de trayectorias para no perder dichas referencias a lo largo del vídeo mediante el Filtro de Kalman [27].

Mientras que el algoritmo SORT consigue un rendimiento aceptable en términos de seguimiento y precisión, también hay que asumir que en entornos en los que se encuentran problemas como oclusiones o diferentes puntos de vista, este puede perder o cambiar la referencia en diversas ocasiones. Debido a esto SORT no es un algoritmo completamente fiable en este tipo de entornos. Por ello, DeepSort utiliza redes neuronales para introducir otra métrica basada en la apariencia del objeto.

El proceso completo de DeepSort en el seguimiento de objetos sería:

1. Extracción de características: Una vez que se han detectado los objetos en la imagen, DeepSORT utiliza un modelo basado en redes neuronales siamesas para extraer características de aspecto de cada objeto. Estas características se utilizan posteriormente para identificar y hacer coincidir los objetos a lo largo del tiempo.

2. Asignación de identidad: Utilizando el algoritmo húngaro, se asigna una identidad única a cada objeto detectado en cada cuadro del video. El algoritmo húngaro busca minimizar la distancia entre las características de aspecto de los objetos en cuadros consecutivos del video.

3. Predicción de trayectorias: Para estimar las posiciones futuras de los objetos, DeepSORT utiliza un filtro de Kalman. El filtro de Kalman utiliza las posiciones anteriores de los objetos y las predicciones de movimiento para calcular la posición probable del objeto en el siguiente cuadro del video.

4. Actualización del estado: Una vez que se ha realizado la predicción de la trayectoria del objeto, se actualiza el estado del objeto con la nueva posición estimada y la identidad asignada por el algoritmo húngaro. Esta actualización del estado permite al sistema realizar un seguimiento continuo de los objetos a medida que se mueven a lo largo del video.

5. Eliminación de falsos positivos: Para evitar el seguimiento de objetos que no son relevantes, DeepSORT utiliza un umbral de confianza para descartar las detecciones con una probabilidad inferior a cierto valor. Esto ayuda a reducir la cantidad de falsos positivos y mejorar la precisión del seguimiento.

Capítulo 3

Implementación

El sistema de tracking visual para las piscinas de *Orca ocean* estará compuesto por varias cámaras que permitan cubrir la totalidad de la superfice desde un punto de vista cenital. El flujo de datos de cada una de las cámaras deberá alimentar un sistema de tracking con los bloques funcionales que se muestran en la figura 3.1.

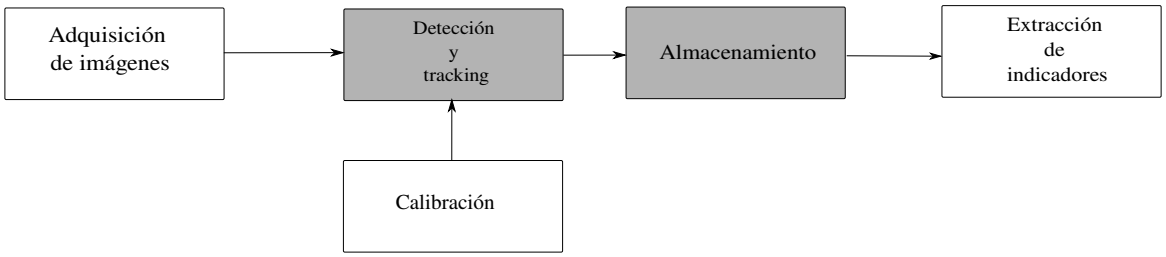


Figura 3.1: Diagrama de bloques del sistema de tracking visual.

El sistema está compuesto por un adquisidor de imágenes, una cámara, que alimentará al módulo de detección y un tracker, encargado de detectar a los animales dentro de las imágenes y seguir dichas detecciones a lo largo del tiempo. Este bloque debe tener como entrada la calibración de las cámaras con el fin de corregir las coordendas de pixel de las distintas imágenes y transformarlas a coordenadas métricas sobre un mapa de las piscinas. La salida con las trayectorias y otra información accesoria (*bounding boxes* y *keyframes*) deberá pasarse a un bloque encargado de almacenar y recuperar del disco dicha información, a la que debe acceder un último bloque, el de extracción de indicadores, para elaborar distintos indicadores de bienestar sobre los animales.

Dado que el desarrollo completo del sistema de tracking visual forma parte de un proyecto más amplio, en este trabajo se se centra únicamente en la implementación y evaluación de los dos bloques que se encuentran sombreados en la figura 3.1, es decir, la detección y el tracking, así como la estructura de almacenamiento de los datos. El desarrollo de estos dos bloques permitirá avanzar en el desarrollo del sistema completo, pues se utilizarán como herramienta para obtener datos de entrenamiento para refinar estos bloques en el futuro.

3.1. Detección y tracking

Como se ha indicado anteriormente, en este trabajo la detección y el tracking se va a realizar mediante técnicas basadas en redes neuronales artificiales. Más concretamente

se ha optado por utilizar una arquitectura basada en YOLO y DeepSORT, que se han descrito en las secciones 2.2.1 y 2.3 respectivamente.

Tal como se describió en la sección 2.1 las redes neuronales requieren de un proceso de entrenamiento que ajusta los pesos de la red para que produzca la salida deseada ante una entrada. Para la realización del entrenamiento se requiere un conjunto de entrenamiento. La obtención de este conjunto de datos se describe en la sección 3.1.1. Posteriormente, se realiza el entrenamiento y se selecciona la arquitectura de red más adecuada para el problema a tratar. Este proceso se describe en la sección 3.1.2.

3.1.1. Obtención de los datos de entrenamiento

Como el objetivo es entrenar un detector de ejemplares del *Orcinus orca* en las pisconas, el conjunto deberá ser una serie de imágenes en las cuales se hayan etiquetado las *bounding boxes* que contienen a los animales dentro de cada imagen.

Para obtener un conjunto de imágenes se ha procedido a la extracción de fotogramas de las cámaras cenitales de *Orca ocean*. Este proceso se ha realizado utilizando la herramienta FFmpeg (sección 1.3) con la que se ha extraído una imagen cada 20 segundos de vídeo de un conjunto de ficheros de video obtenidos de distintas cámaras y en distintas condiciones de iluminación.

Una vez obtenido un conjunto de imágenes, se etiquetaron y delimitaron las *bounding boxes* de forma manual, para disponer de la salida deseada del detector para el entrenamiento y poder comprobar el rendimiento de éste una vez finalizado el entrenamiento. Como queremos trackear las orcas de la piscina la única clase que utilizaremos será la clase orca, aunque estos algoritmos permiten detectar y seguir difetentes clases de objetos. Para etiquetar las imágenes se utilizó la herramienta labelImg, descrita en la sección 1.3. En la figura 3.2 se muestra una captura de pantalla obtenida durante el proceso de de marcaje de las imágenes.

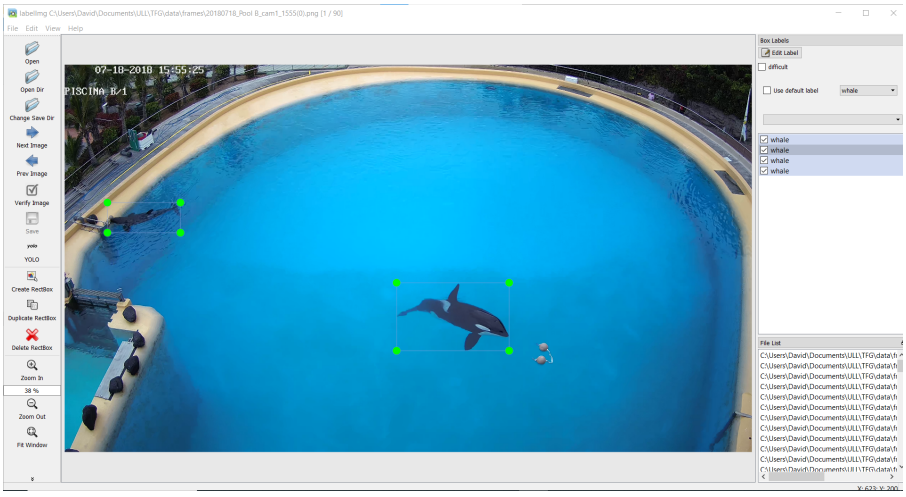


Figura 3.2: Etiquetado de datos en labelImg

Mediante el procedimiento descrito anteriormente, se han extraído un total de 2626 imágenes, sin embargo, ha habido que eliminar todas en las que no aparece una orca a detectar. Una vez se ha realizado esta limpieza y se han etiquetado todos los fotogramas, se generó un fichero .txt por cada imagen que contiene la información de las clases encontradas y las *bounding boxes* correspondientes.

Finalmente, el conjunto de datos se compone de 2389 imágenes de tamaño 960×540 etiquetadas en función de las apariciones de la única clase a detectar, es decir, una orca. Este conjunto de datos se ha dividido para que un 80 % (1911 imágenes) sea utilizado en el entrenamiento y un 20 % (478 imágenes) sean utilizadas para comprobar el rendimiento del detector y tracker sobre datos no conocidos por la red. En la tabla 3.1 se muestra el número de imágenes en función del número de orcas que aperecen en cada fotograma para el conjunto de entrenamiento y en la tabla 3.2 se muestra para el conjunto de prueba.

Nº de orcas en la imagen	Imágenes
1	1130
2	672
3	107
4	1
5	1

Tabla 3.1: Distribución de imágenes por número de orcas en el conjunto de entreno

Nº de orcas en la imagen	Imágenes
1	290
2	160
3	28
4	4

Tabla 3.2: Distribución de imágenes por número de orcas en el conjunto de prueba

3.1.2. Entrenamiento de redes neuronales

Se ha optado por evaluar el rendimiento de las arquitecturas neuronales basadas en YOLO+DeepSORT en sus distintas versiones (sección 2.2.2). Debido a que el número de imágenes del conjunto de entrenamiento descrito en la sección 3.1.1 es reducido, se ha optado por realizar un proceso de entrenamiento por transferencia (*transfer learning*) [28], una técnica de aprendizaje automático en la que se utiliza un modelo pre-entrenado en una tarea relacionada para resolver otra tarea distinta pero similar. En lugar de entrenar un modelo desde cero para cada tarea específica, se aprovecha el conocimiento previo del modelo pre-entrenado para acelerar y mejorar el entrenamiento del nuevo modelo.. Se han evaluado cuatro aquitecturas:

- YOLOv3+DeepSORT.
- YOLOv3tiny+DeepSORT.
- YOLOv4+DeepSORT.
- YOLOv4tiny+DeepSORT.

Para ello se ha utilizado una implementación de código abierto ya existente. Se ha extraído el código del modelo de un repositorio público de Github llamado yolov4-deepsort desarrollado por theAIGuysCode [29].

Este repositorio implementa las versiones YOLOv3 y YOLOv4 (así como sus versiones tiny) junto con DeepSORT, funcionando bajo TensorFlow teniendo así un Multi Object Tracker de alta calidad.

Una vez llegados a este punto, se ha procedido a entrenar cada uno de los modelos para que la red convolucional ajuste sus pesos internos con el objetivo de minimizar el error y aumentar la precisión. Por ello, tal y como se ha indicado anteriormente, se han utilizado los pesos por defecto de las implementaciones de YOLO como pre-entreno y se ha realizado el proceso de entrenamiento mediante el dataset formalizado anteriormente.

Las figuras 3.3, 3.4, 3.5 y 3.6 muestran la función de pérdida en función de las épocas de entrenamiento para cada una de las cuatro arquitecturas de red.

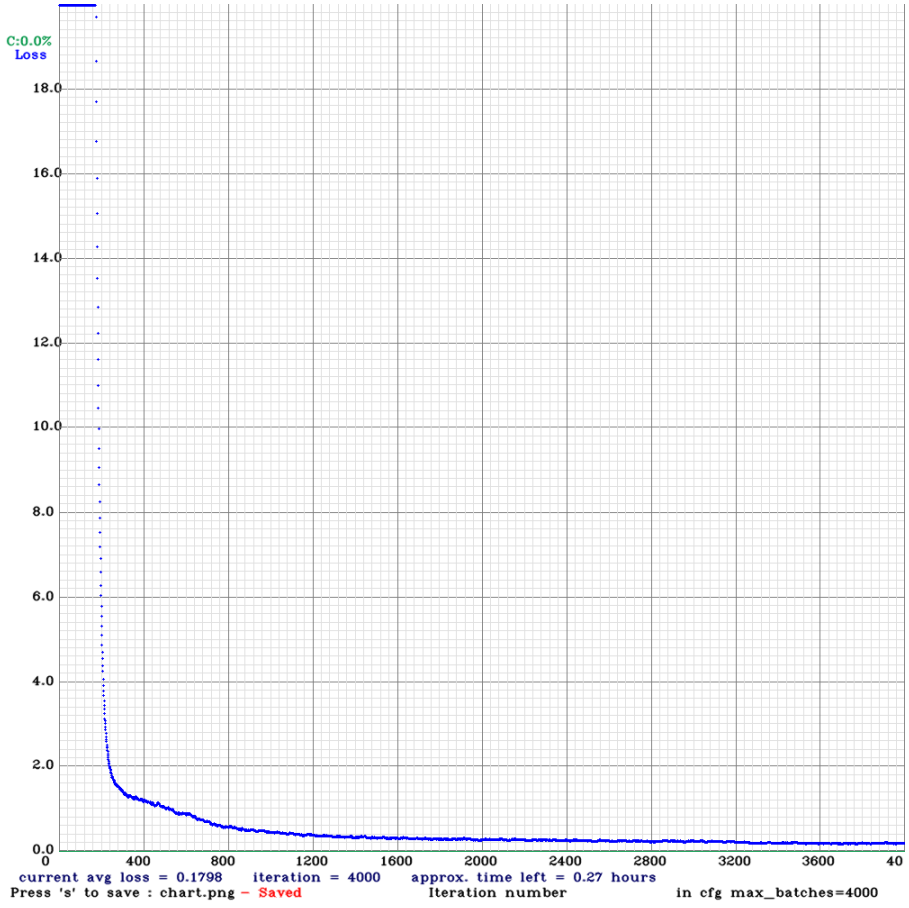


Figura 3.3: Gráfica de los resultados del entrenamiento de YOLOv3

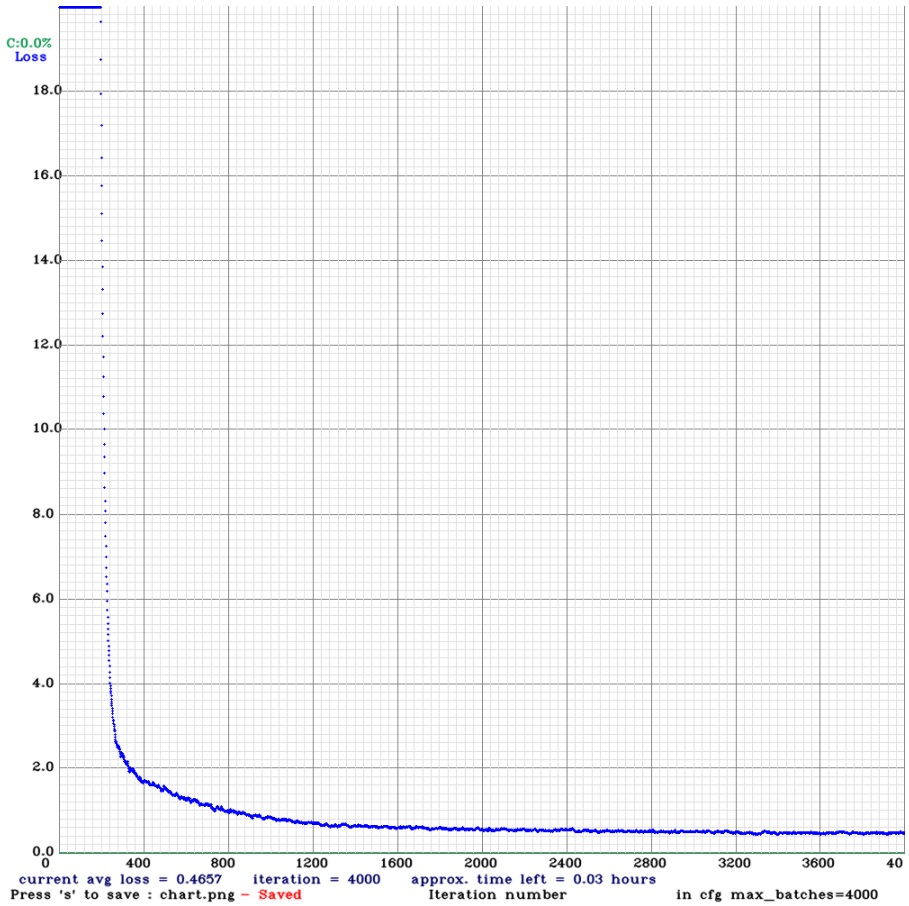


Figura 3.4: Gráfica de los resultados del entrenamiento de YOLOv3 tiny

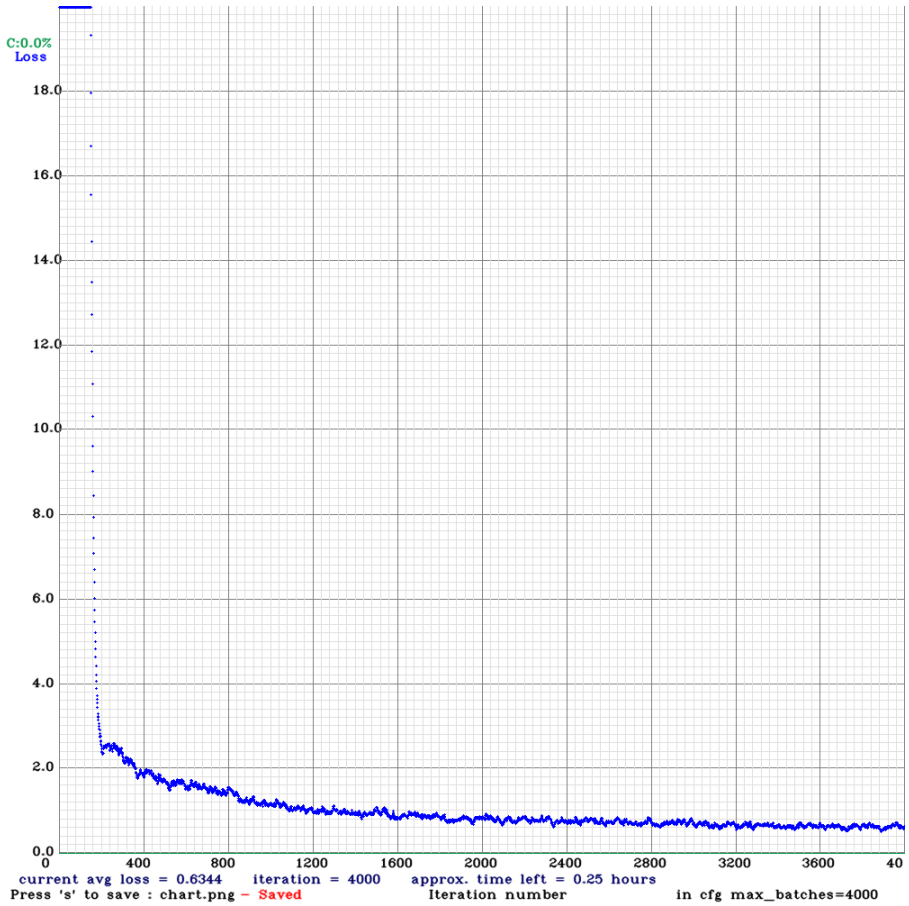


Figura 3.5: Gráfica de los resultados del entrenamiento de YOLOv4

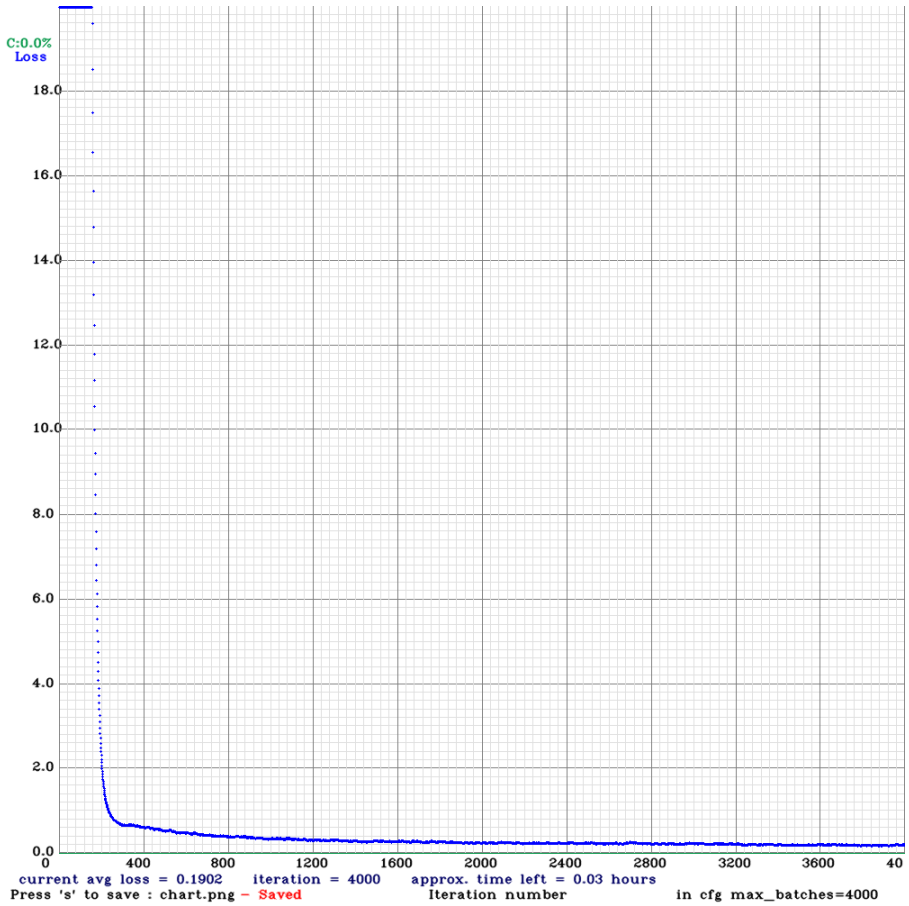


Figura 3.6: Gráfica de los resultados del entrenamiento de YOLOv4 tiny

Para todas las aquitecturas se han realizado 4000 iteraciones, aunque tal y como se observa a partir de las 1200 iteraciones el valor de la función de pérdida a apenas evoluciona.

3.2. Sistema de almacenamiento

Para poder almacenar el resultado arrojado por el módulo de detección y tracking, era necesario el desarrollo de una estructura de datos que permitiera guardar toda la información relevante del tracking animal. El objetivo del sistema de almacenamiento es que cumpla los siguientes requisitos funcionales:

- Debe permitir almacenar las trayectorias detectadas en tiempo real.
- Se debe contemplar la posibilidad de que existan varias trayectorias en el mismo vídeo.
- La trayectoria consistirá al menos en la tripleta de (x, y, timestamp), donde x,y indica la posición del animal y timestamp es la marca de tiempo.
- Se debe contemplar también la posibilidad de almacenar tanto las *bounding box* de las distintas detecciones en la imagen así como la imagen en cuestión.

La implementación se ha realizado con la librería HDF5 (Hierarchical Data Format 5), descrito en la sección 1.3, que utiliza un formato de archivo diseñado para almacenar y

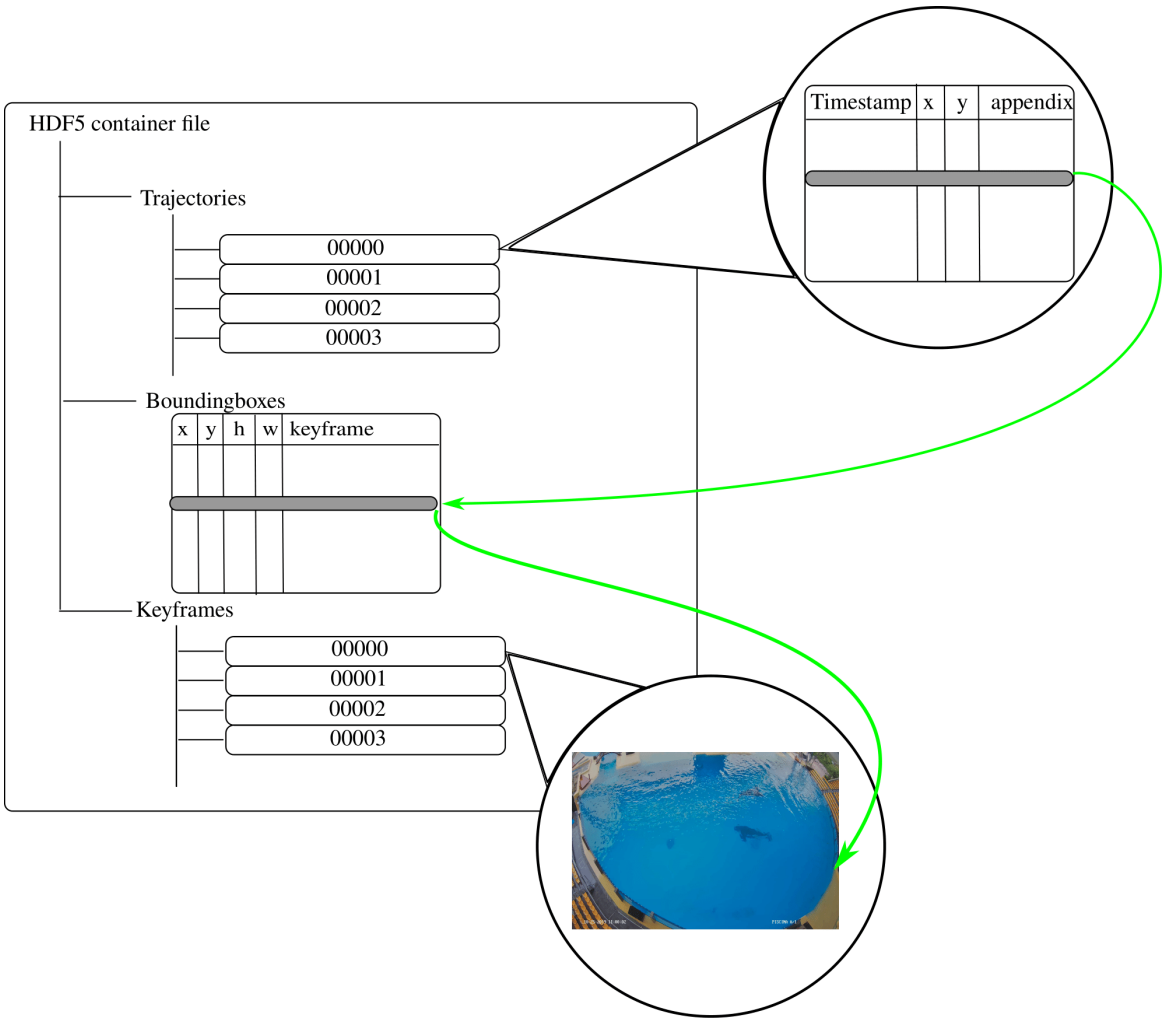


Figura 3.7: Diagrama de la estructura del almacenamiento de datos en HDF5

organizar grandes volúmenes de datos heterogéneos. La librería HDF5 está escrita en C y proporciona interfaces para varios lenguajes de programación, en este proyecto se ha utilizado la interfaz de C++. El formato HDF5 permite agrupar distintos conjuntos de datos y además posee la característica de poder referenciar desde un conjunto de datos, al estilo de un puntero de memoria, a otro conjunto de datos o a parte de este dentro del mismo fichero. En la figura 3.7 se explota esta característica estructurando los datos en tres grupos:

- Trajectories: Contendrá las trayectorias halladas por el modelo. Cada trayectoria será un array de puntos de trayectoria que a su vez estarán compuestos de:
  - Timestamp: Marca de tiempo.
  - X: Posición en el eje X del animal al que se le realiza el seguimiento en la marca de tiempo anteriormente descrita.
  - Y: Posición en el eje Y del animal al que se le realiza el seguimiento en la marca de tiempo anteriormente descrita.
  - Appendix: Se trata de una referencia hacia la bounding box, que encierra al animal, al que se le está realizando el seguimiento en esa trayectoria, en la marca de tiempo anteriormente descrita.

- Boundingboxes: Son los cuadros que encierran al animal en la detección. Cada uno está formado por:
  - X: Posición en el eje x de la esquina superior izquierda de la bounding box.
  - Y: Posición en el eje y de la esquina superior izquierda de la bounding box.
  - H: Altura en píxeles de la bounding box.
  - W: Ancho en píxeles de la bounding box.
  - Keyframe: Referencia a un screenshot del vídeo en la detección actual.
- Keyframes: Son screenshots del vídeo en momentos clave del seguimiento. La elección del momento clave a capturar dependerá del usuario, por ejemplo, cada vez que desaparezca una trayectoria o aparezca una nueva.

Para facilitar el acceso tanto de escritura como de lectura de los ficheros HDF5, se ha desarrollado un conjunto de clases en C++, basada en la librería de acceso HDF5. El diagrama de clases correspondiente se muestra en la figura 3.8 y se describen a continuación:

- trdb: Se trata de la clase principal del código y representa la base de datos de trayectorias. Es la clase encargada de crear o leer el fichero HDF5 que representa la base de datos así como también de escribir en él las entradas necesarias en función de la ejecución del programa.
- traject_t: Es la clase que representa un punto de una trayectoria de un animal. Está compuesto por una posición (x, y), un timestamp y una bbref_t.
- bbref_t: Es la clase que representa una referencia a la bounding box de una trayectoria.
- bbox_t: Es la estructura de datos que representa una bounding box. Está compuesta por un punto inicial (x,y), un alto, un ancho y una imgref_t.
- imgref_t: Es la clase que representa una referencia a una imagen del vídeo en el formato de archivos HDF5. Se asocia a una o más bbox_t para poder relacionar visualmente las bbox_t a los animales detectados.

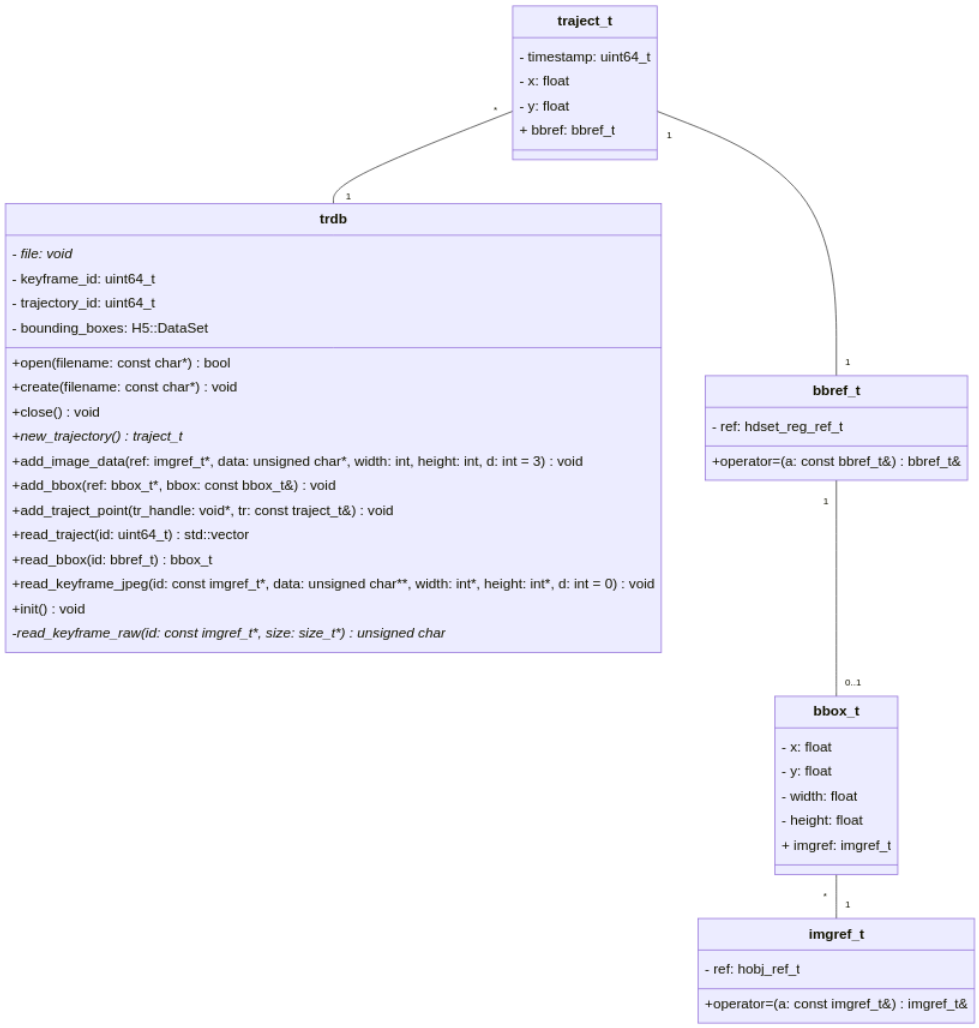


Figura 3.8: Diagrama de clases para acceder a los ficheros que almacenan las trayectorias.

3.2.1. API de acceso para Python

Aunque el desarrollo de la API para acceder a los ficheros de datos se ha hecho en C++ porque el sistema de producción finalmente se implementará en este lenguaje, para realizar la integración con los códigos de prueba y entrenamiento de las redes neuronales, que generalmente se desarrollan en Python, se ha decidido crear un *wrapper* en Python para la API de acceso descrita en la sección anterior.

Una de las principales necesidades de utilizar un *wrapper* en este contexto es abordar las diferencias inherentes entre los lenguajes. Estas diferencias pueden incluir la forma en que los lenguajes manejan los tipos de datos, las estructuras de datos y la gestión de memoria. Por ejemplo, Python es un lenguaje interpretado con tipado dinámico, mientras que C++ es un lenguaje compilado con tipado estático. Esto implica que los datos y las llamadas a funciones deben ser adaptados para garantizar una interacción coherente y sin errores entre los dos lenguajes.

Aquí es donde entra en juego el *wrapper*: actúa como una capa de abstracción que facilita la traducción de las llamadas y datos entre los dos lenguajes.

En este caso particular, se ha utilizado la herramienta SWIG (Simplified Wrapper and Interface Generator), descrita en la sección 1.3, para generar automáticamente el wrapper. Como puede apreciarse en la figura 3.9, gracias a SWIG, se puede utilizar la

librería de acceso desarrollada en C++ con una interfaz en Python que nos permite utilizar algunas ventajas del lenguaje como tipado dinámico y programación funcional, por lo que ahora la comunicación con el código del tracker es mucho más sencilla. En este código de ejemplo, se lee la primera trayectoria de la base de datos y se itera por cada punto de la trayectoria imprimiéndolo junto con la imagen que tiene asociada a la bounding box.

```
 1 import pytrdb as db
 2 import numpy as np
 3 import matplotlib.pyplot as plt
 4
 5 db.trdb_init();      # se inicializa la bbdd
 6
 7 trd = db.trdb();
 8
 9 trd.open('../build/tests/test.h5')  # se abre el fichero de la bbdd
10
11 trj=trd.read_traject(0)      # se lee la primera trayectoria
12
13 for t in trj:              # se lee cada punto de la trayectoria
14     print(t)               # se imprime el punto de la trayectoria
15     bounding_box = 0
16     bb=trd.read_bbox(t.bounding_box)
17     img = read_image_jpeg(trd, bb.keyframe)
18
19
20     plt.imshow(img)
21     plt.show()             # se imprime la imagen
22
23 trd.close()
```

Figura 3.9: Código de ejemplo de uso del wrapper de Python para el acceso a un fichero de la base de datos.

Capítulo 4

Evaluación y resultados

4.1. Evaluación de los modelos neuronales

Dado que existen diferentes alternativas para el módulo de detección y tracking, se han realizado distintas pruebas de rendimiento para cada una de ellas, lo que permite determinar cuál es la que mejor se adapta al problema planteado. Para ello se utilizan una serie de métricas descritas en la sección 4.1.1. Los resultados de las pruebas realizadas se pueden ver en la sección 4.1.2

4.1.1. Métricas

Para poder elegir un modelo adecuado, es necesario conocer el rendimiento y la efectividad de cada uno de ellos en el problema planteado. Aunque en rasgos generales unos puedan ser mejores que otros, es necesario realizar pruebas empíricas que lo demuestren, ya que en función del contexto algunas implementaciones podrían superar a otras.

Para elegir un modelo como el definitivo, se suelen analizar tres métricas:

- Rendimiento del modelo: Es necesario conocer a qué velocidad se ejecuta el modelo en tiempo real. Debido a que uno de los objetivos del proyecto es implementar el resultado en un sistema embebido, no se pueden ignorar los requerimientos de hardware de la solución. La métrica utilizada en este caso son los frames por segundo (fps).

- Precisión del detector: Para lograr un buen tracking es necesario conseguir una buena detección. Por lo tanto, será necesario analizar hasta qué punto el detector funciona correctamente con el problema planteado. Para esta parte la métrica utilizada es el mean average precision (mAP). El cálculo del mAP implica primero calcular la precisión media (AP) de cada clase de objeto detectada en la imagen, y luego tomar el promedio de estas APs para obtener el valor final del mAP. En este caso solo existe una clase, por lo que el AP coincidirá con el mAP.

- Calidad del tracker: Finalmente, será necesario obtener una métrica que indique cómo funciona el tracker dentro del contexto planteado. Normalmente los trackers utilizan como métricas el MOTA (Multiple Object Tracking Accuracy) y el MOTP (Multiple Object Tracking Precision).

- MOTA: Mide la proporción de errores cometidos por el algoritmo de seguimiento en relación con el número total de objetos reales presentes en la escena. Un valor más alto de MOTA indica una menor cantidad de errores [30]. La fórmula para calcularla es:

$$MOTA = 1 - (FN + FP + ID)/GT$$

  Donde:

  - FN (False Negatives) son los objetos reales no detectados.
  - FP (False Positives) son los objetos que se detectan pero que en realidad no existen.
  - ID (Identity Switches) son los errores en la asignación de identidades a objetos.
  - GT (Ground Truth) es el número total de objetos reales en la escena.

- MOTP: Mide la precisión promedio de la localización de los objetos en cada cuadro individualmente. Se calcula como la media de la distancia euclidiana entre las posiciones predichas y las posiciones reales de los objetos [30]. La fórmula para calculara es:

$$MOTP = 1/T \sum t = 1, T \sum i = 1, nd(i, t)$$

  Donde:

  - T es el número total de cuadros en el vídeo.
  - n es el número total de objetos detectados en todo el seguimiento.
  - d(i,t) es la distancia euclidiana entre la posición predicha y la posición verdadera del objeto i en el cuadro t.

  Estas son las métricas más utilizadas para conocer la calidad de un algoritmo de tracking, sin embargo, en un dataset propio es muy complicado obtenerlas por lo que como métrica simplificada utilizaremos la diferencia entre tracks resultantes del modelo y los tracks reales que debería haber predicho. Esta métrica no es tan precisa, sin embargo, sirve para tener una idea de la calidad del tracker.

Estas pruebas se han realizado con dos sistemas de hardware diferentes: una GPU Nvidia Geforce GTX 1050 con 3GB de memoria VRAM y con 1518 MHz de frecuencia de reloj, que se ejecuta en una estación de trabajo de escritorio, y una placa de desarrollo Jetson Xavier con una GPU NVIDIA Volta con 512 núcleos NVIDIA CUDA y 64 núcleos Tensor, apta para el desarrollo de sistemas embebidos.

4.1.2. Resultados de las pruebas

Primero se ha analizado la velocidad de cada modelo. Como podemos apreciar en los resultados aportados en la tabla 4.1 en la GTX 1050 podría implementarse prácticamente cualquier modelo de manera más o menos eficiente, sin embargo, en el sistema embebido Jetson Xavier no. Por lo tanto, las únicas implementaciones viables en los dos hardware serían el YOLOv3 Tiny + DeepSORT y el YOLOv4 tiny + DeepSORT por lo que el resto de pruebas se realizarán con estas dos versiones.

Modelo	GTX 1050	Jetson Xavier
YOLOv3 + Deepsort	10.94 fps	2.38 fps
YOLOv3 tiny + Deepsort	25.80 fps	5.91 fps
YOLOv4 + Deepsort	14.72 fps	2.97 fps
YOLOv4 tiny + Deepsort	71.91 fps	13.97 fps

Tabla 4.1: Resultados de rendimiento promedio en los tres vídeos de cada uno de los modelos

El segundo paso ha sido evaluar la calidad de las detecciones realizadas. Es importante señalar que estas no incluyen la parte de tracking (DeepSORT), aunque se supone que a mayor calidad en las detecciones, también será mayor la calidad en el proceso de tracking completo. Para las pruebas del detector se ha utilizado el conjunto de datos de prueba de 478 imágenes, descrito en la sección 3.1.1. Como se puede apreciar en la tabla 4.2, el detector con YOLOv4 tiny tiene una precisión media considerablemente más alta que la que utiliza YOLOv3 tiny, siendo un 12.58 % más preciso.

Model	mAP
YOLOv3 tiny	83.54 %
YOLOv4 tiny	94.05 %

Tabla 4.2: Resultados de mAP en el detector de objetos

Finalmente se ha evaluado módulo de detección y tracking en su conjunto. Para ello se han utilizado 3 vídeos de 2 minutos de duración cada uno de los cuales se conocía el número de trayectorias esperadas. La prueba realizada ha consistido en determinar la cantidad de trayectorias detectadas por el módulo de tracking. Esto permite obtener una idea sobre la cantidad de pérdidas de identidad que tiene cada uno de las alternativas que se han considerado. Las tablas 4.3, 4.4 y 4.5 muestran los datos obtenidos con las dos alternativas consideradas para cada uno de los videos de test.

Model	Total tracks	Real tracks
YOLOv3 tiny + Deepsort	43	3
YOLOv4 tiny + Deepsort	10	3

Tabla 4.3: Resultados del tracker con el vídeo test 1

Model	Total tracks	Real tracks
YOLOv3 tiny + Deepsort	37	2
YOLOv4 tiny + Deepsort	41	2

Tabla 4.4: Resultados del tracker con el vídeo test 2

Model	Total tracks	Real tracks
YOLOv3 tiny + Deepsort	36	1
YOLOv4 tiny + Deepsort	26	1

Tabla 4.5: Resultados del tracker con el vídeo test 3

Podemos ver que de promedio el modelo con YOLOV4 tiny + DeepSORT pierde menos veces el track. Esto es debido a que el detector es más preciso y para realizar un buen tracking es necesario un buen detector. Por lo tanto, debido a los resultados de las pruebas realizadas, se ha decidido que el modelo más adecuado a utilizar sea el que utiliza YOLOv4 tiny + DeepSORT.

4.2. Funcionamiento del sistema de almacenamiento de trayectorias

Una vez implementado el sistema de almacenamiento que guarde los resultados arrojados por el modelo y el wrapper de Python que permita la comunicación entre ambos códigos, es hora de hacer una prueba de todo el proyecto en conjunto para ver su comportamiento.

Para ello se ha ejecutado el modelo en un vídeo de prueba de 15 minutos de duración, con el objetivo de tener una estimación de los recursos de infraestructura, sobretodo a nivel de almacenamiento, necesarios para utilizar el software desarrollado. Para ello se ha realizado el almacenamiento de las trayectorias detectadas junto con las *bounding boxes* y los *keyframes* cada 60 frames. Teniendo en cuenta que el video tiene una tasa de 20 fps, esto implica el almacenamiento de 1 imagen por segundo. La prueba ejecutada ha generado un fichero en formato HDF5 de 51.6MB. En ella se han almacenado:

- 106 trayectorias
- 218 bounding boxes
- 201 keyframes

En estas condiciones, cabe esperar que el número de *keyframes* almacenado sea 300, pero al no haber detección en parte de la secuencia de video utilizada este número es menor.

Finamente, se han comprobado los tiempos de lectura y escritura sobre el fichero, promediando los tiempos de 100 intentos, arrojando los siguientes resultados:

- Trayectorias:
  - Escritura: 0.08 ms
  - Lectura: 0.04 ms
- Bounding boxes:
  - Escritura: 0.13 ms
  - Lectura: 0.06 ms
- Keyframes:
  - Escritura: 5.8 ms
  - Lectura: 5.1 ms

Se observa que estos tiempos son suficientemente pequeños como para realizar un almacenamiento continuo incluso de una secuencia de video a 20 fps puesto que en este caso se debería procesar un frame aproximadamente cada 50 ms.

Capítulo 5

Conclusiones y líneas futuras

Después del estudio realizado y a la vista de los resultados obtenidos, las conclusiones son las siguientes:

- Se han evaluado cuatro modelos neuronales basados en YOLO+DeepSORT para el tracking de ejemplares del orcinus orca, seleccionando el modelo más eficiente en cuanto a tiempo de ejecución y precisión.
- El modelo que mejor rendimiento presenta es el YOLOv4+DeepSORT, porque tiene menos pérdidas de identidad y permite una mayor tasa de frames por segundo.
- Se ha evaluado el tiempo de ejecución de los modelos neuronales sobre un sistema embebido dotado de GPU, comprobándose que es viable el uso de este tipo de hardware para esta aplicación, aunque sería necesario reducir ligeramente la tasa de muestreo de video.
- Los modelos neuronales evaluados no permiten la reidentificación de individuos, aunque si permite determinar la trayectoria de los animales visibles en cada momento, sin identificarlos, pudiéndose producir péridas de identidad.
- Se ha diseñado e implementado un sistema de almacenamiento basado en el formato de fichero HDF5 para almacenar las trayectorias de los animales en la piscina, junto con las *bounding boxes* y ciertos *frames* de la secuencia de video.
- El sistema de almacenamiento permitirá recopilar datos para un posterior entrenamiento de modelos más avanzados.
- El trabajo desarrollado abre la posibilidad de realizar estudios de larga duración sobre la población de animales de *Orca ocean*, determinando indicadores de bienestar sobre las trayectorias calculadas.

En el futuro se abordarán lo siguiente:

- Uso de hardware más potente para usar las vesiones completas de YOLO (no las *tiny*).
- Desarrollar y mejorar la precisión del algoritmo de seguimiento entrenando redes neuronales a medida que permitan la reidentificación de animales, lo que permitiría asignar las distintas secuencias detectadas a un animal. Esto se haría utilizando como conjunto de entrenamiento secuencias de datos obteninadas con la implementación actual.

- Entrenamiento del algoritmo DeepSORT para mejorar o incorporar la reidentificación de orcas, ya que debido a la cantidiad de reducida de datos disponibles no fue posible, aunque este trabajo servirá como herramienta para extraer tales datos.
- Se deberán desarrollar indicadores de bienestar a aprtir de las trayectorias obtenidas, ya que este aspecto se quedaba fuera del ámbito definido para este trabajo.

Capítulo 6

Summary and Conclusions

After the conducted study and considering the obtained results, the conclusions are as follows:

- Four neural models based on YOLO+DeepSORT for tracking orcinus orca specimens were evaluated, selecting the most efficient model in terms of execution time and accuracy.
- The model that presents the best performance is YOLOv4+DeepSORT, as it has fewer identity losses and allows for a higher frame rate.
- The execution time of the neural models on an embedded system equipped with a GPU was evaluated, confirming the viability of using this type of hardware for this application.
- The evaluated neural models do not allow for individual re-identification, although they do allow determining the trajectory of visible animals at each moment without identifying them, which may result in identity losses.
- A storage system based on the HDF5 file format has been designed and implemented to store the animal trajectories in the pool, along with the bounding boxes and certain frames of the video sequence.
- The storage system will enable collecting data for further training of more advanced models.
- The developed work opens up the possibility of conducting long-term studies on the population of animals in Orca Ocean, determining well-being indicators based on the calculated trajectories.

In the future, the following will be addressed:

- The use of more powerful hardware to utilize the full versions of YOLO (not the tiny versions).
- Developing and improving the accuracy of the tracking algorithm by training custom neural networks that allow for animal re-identification, enabling the assignment of different detected sequences to an animal. This would be done by using the current implementation’s data sequences as a training set.

- Training the DeepSORT algorithm to improve or incorporate the re-identification of orcas, as it was not possible due to the limited amount of available data, although this work will serve as a tool to extract such data.
- Developing well-being indicators based on the obtained trajectories.

Capítulo 7

Presupuesto

En la siguiente tabla, se muestra el presupuesto de ejecución de este trabajo de fin de grado. Para los siguientes cálculos, se toma el valor de 15€ la hora como tarifa para un Ingeniero Informático.

Objeto	**Coste**
Licencias de software	0,00 €
Placa de desarrollo Jetson Xavier NX	1.400,00 €
Amortización de equipos informáticos	133,00 €
Salario (300 horas x 15 €)	4.500,00 €
Consumibles informáticos(Estimado)	20,00 €
Material de oficina(Estimado)	10,00 €
Suministros generales (Estimado)	40,00 €
Subtotal	6.103,00 €
IGIC tipo general (7 %)	427,21 €
TOTAL	**6530,21 €**

Bibliografía

[1] Colaboradores de Wikipedia. *TensorFlow*. 2021. url: https://es.wikipedia.org/wiki/TensorFlow.

[2] Anaconda Inc. *Anaconda*. 2023. url: https://www.anaconda.com/.

[3] *FFmpeg. (s.f.).* url: https://ffmpeg.org.

[4] NVIDIA. *The World's smallest AI supercomputer*. url: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/.

[5] Colaboradores de Wikipedia. *Hierarchical data format*. url: https://en.wikipedia.org/wiki/Hierarchical_Data_Format.

[6] Colaboradores de SWIG. *Simplified wrapper and interface generator*. url: https://www.swig.org/.

[7] «Multiple object tracking: A literature review». En: (2022). url: https://arxiv.org/abs/1409.7618.

[8] A. M. Montesinos López O. A. amd López y J. Crossa. «Fundamentals of Artificial Neural Networks and Deep Learning». En: (2022).

[9] J. Brownlee. *Loss Functions and Their Use In Neural Networks*. url: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9.

[10] Colaboradores de Wikipedia. *Backpropagation*. 2023. url: https://en.wikipedia.org/wiki/Backpropagation.

[11] Colaboradores de Wikipedia. *Red neuronal convolucional*. 2023. url: https://es.wikipedia.org/wiki/Red_neuronal_convolucional.

[12] K O'Shea. «An Introduction to Convolutional Neural Networks». En: (2015). url: https://arxiv.org/abs/1511.08458.

[13] Colaboradores de Wikipedia. *Scale-invariant feature transform*. 2019. url: https://es.wikipedia.org/wiki/Scale-invariant_feature_transform.

[14] Colaboradores de Wikipedia. *SURF*. 2023. url: https://es.wikipedia.org/wiki/SURF.

[15] Colaboradores de Wikipedia. *Features from accelerated segment test*. 2022. url: https://en.wikipedia.org/wiki/Features_from_accelerated_segment_test.

[16] M. Moin. *Object Detection: A journey from R-CNN to Mask R-CNN and YOLO*. 2023. url: ttps://medium.com/augmented-startups/object-detection-a-journey-from-r-cnn-to-mask-r-cnn-and-yolo-698ba097d490.

[17] Colaboradores de Wikipedia. url: https://es.wikipedia.org/wiki/YOLO.

[18] J. Redmon et al. «You Only Look Once: Unified, Real-Time Object Detection». En: (2016), págs. 779-788. doi: 10.1109/CVPR.2016.91.

[19] J Redmon. «You Only Look Once: Unified, Real-Time Object Detection». En: (2015). url: https://arxiv.org/abs/1506.02640.

[20] J. Redmon. *YOLO: Real-Time Object Detection*. url: https://pjreddie.com/darknet/yolo.

[21] Alexey Bochkovskiy, Chien-Yao Wang y Hong-Yuan Mark Liao. «YOLOv4: Optimal Speed and Accuracy of Object Detection». En: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. url: https://arxiv.org/abs/2004.10934.

[22] D. Comaniciu y P. Meer. «Mean shift: a robust approach toward feature space analysis». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.5 (2002), págs. 603-619. doi: 10.1109/34.1000236.

[23] Daniel B. Bung y Daniel Valero. «Optical flow estimation in aerated flows». En: *Journal of Hydraulic Research* 54.5 (2016). Ed. por Taylor & Francis, págs. 575-580. doi: 10.1080/00221686.2016.1173600. eprint: https://doi.org/10.1080/00221686.2016.1173600. url: https://doi.org/10.1080/00221686.2016.1173600.

[24] Nicolai Wojke, Alex Bewley y Dietrich Paulus. «Simple Online and Realtime Tracking with a Deep Association Metric». En: *CoRR* abs/1703.07402 (2017). arXiv: 1703.07402. url: http://arxiv.org/abs/1703.07402.

[25] H. W. Kuhn. «The Hungarian method for the assignment problem». En: *Naval Research Logistics Quarterly* 2.1-2 (1955), págs. 83-97. doi: https://doi.org/10.1002/nav.3800020109. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

[26] Yan Pei et al. *An Elementary Introduction to Kalman Filtering*. 2019. arXiv: 1710.04055 [eess.SY].

[27] R. Kanjee. *DeepSORT - deep learning applied to object tracking*. 2022. url: https://medium.com/augmented-startups/deepsort-deep-learning-applied-to-object-tracking-924f59f99104.

[28] Fuzhen Zhuang et al. «A Comprehensive Survey on Transfer Learning». En: *CoRR* abs/1911.02685 (2019). arXiv: 1911.02685. url: http://arxiv.org/abs/1911.02685.

[29] theAIGuysCode. *TheAIGuysCode/yolov4-deepsort: Object Tracking implemented with Yolov4, DeepSort, and tensorflow*. url: https://github.com/theAIGuysCode/yolov4-deepsort.

[30] Keni Bernardin, Alexander Elbs y Rainer Stiefelhagen. «Multiple Object Tracking Performance Metrics and Evaluation in a Smart Room Environment». En: 2006.