PROYECTO FIN DE GRADO

TÍTULO: Detección y seguimiento de truchas en vídeo con eliminación de fondo

AUTOR: Samuel Galán Munilla

TITULACIÓN: Grado en Ingeniería de Sistemas deTelecomunicación

TUTOR: Nicolás Sáenz Lechón

DEPARTAMENTO: IAC

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE: Cristóbal Tapia García

TUTOR: Nicolás Sáenz Lechón

SECRETARIA: Juana María Gutiérrez Arriola

Fecha de lectura:

Calificación:

La Secretaria,

Agradecimientos

Quisiera expresar mi agradecimiento a todas las personas que han contribuido de alguna manera al desarrollo de este PFG.

En primer lugar, me gustaría agradecer a mi tutor, Nicolás Sáenz, por sus instrucciones, reuniones y consejos a lo largo de todo el proceso. Gracias a él, he descubierto un nuevo mundo, la inteligencia artificial, la cual me ha resultado muy interesante y me motiva a seguir estudiándolo en el futuro.

A mis amigos y compañeros que me han acompañado durante toda la carrera y han hecho que estos años hayan sido una experiencia única. Desde mis compañeros de teleco hasta mi grupo "puchun". Deseo agradecer especialmente a Mauri, Aroia, Ainara y Lorena quiénes han sido mis cuatro pilares en este viaje.

Finalmente, quisiera agradecer a mi familia, porque gracias a ellos he llegado hasta aquí. Gracias por creer en mí y en vuestro apoyo y motivación constante en todas las etapas de mi vida.

Resumen

Este Proyecto de Fin de Grado (PFG) se centra en el desarrollo de un sistema para el seguimiento de truchas en una pecera mediante la utilización de técnicas de procesado de imagen por computadora y redes neuronales convolucionales (CNNs). La acuicultura es un sector con un enorme potencial económico para la economía y para la sostenibilidad alimentaria del planeta y se puede beneficiar del análisis automatizado de los comportamientos de los peces en cautividad.

El desarrollo de este sistema se llevará a cabo con el uso del algoritmo YOLO (You Only Look Once) para la detección de objetos en vídeo y la implementación de una técnica de zoom dinámico que optimizará el procesamiento del vídeo tanto en eficiencia como en calidad de detecciones.

Adicionalmente, en este PFG se desarrollará una interfaz gráfica de usuario (GUI) para que el usuario del sistema pueda realizar detecciones en vídeos sin el requerimiento de tener conocimientos sobre las tecnologías empleadas en este proyecto para implementar el sistema y descritas en esta memoria.

Los resultados obtenidos demuestran la eficacia y robustez del sistema final. Se realizan comparaciones tanto cuantitativas como cualitativas de todos los modelos generados para obtener un modelo final de calidad.

Se analizarán los errores obtenidos y mejoras posibles, además de trabajos futuros, consiguiendo obtener un modelo capaz de realizar una detección de objetos de calidad y fiable.

Abstract

This Final Degree Project (PFG) focuses on the development of a system for monitoring trout in a fish tank using computer image processing techniques and convolutional neural networks (CNNs). Aquaculture is a sector with enormous economic potential for the economy and for the food sustainability of the planet and can benefit from the automated analysis of the behaviours of fish in captivity.

The development of this system will be carried out with the use of the YOLO (You Only Look Once) algorithm for the detection of objects in video and the implementation of a dynamic zoom technique that will optimize video processing both in efficiency and quality of detections.

Additionally, in this PFG a graphical user interface (GUI) will be developed so that the system user can perform video detections without the requirement of having knowledge of the technologies used in this project to implement the system and described in this report.

The results obtained demonstrate the effectiveness and robustness of the final system. Both quantitative and qualitative comparisons of all the generated models are made to obtain a final quality model.

The errors obtained and possible improvements will be analysed, as well as future work, in order to obtain a model capable of performing quality and reliable object detection.

Índice de figuras

Lista de acrónimos

AI/IA: Artificial Inteligence (Inteligencia Artificial)

CNN: Convolutional Neural Network (Red neuronal convolucional)

CPU: Central Processing Unit (Unidad central de procesamiento)

CVAT: Computer Vision Annotation Tool

DL: Deep Learning (Aprendizaje profundo)

FPS: Frames per second (Fotogramas por segundo)

GPU: graphics processing unit (Unidad de procesamiento gráfico)

GUI: Graphical User Interface (Interfaz gráfica de usuario)

HTML: HyperText Markup Language (Lenguaje de marcas de hipertexto)

IoU: Intersection over Union (Intersección sobre unión)

mAP: Mean Average Precision (Precisión media)

MCP: neurona artificial de McCulloch-Pitts

ML: Machine Learning (Aprendizaje automático)

MTS: MPEG Transport Stream

ODS: Objetivos de Desarrollo Sostenible

OpenCV: Open Source Computer Vision Library

PFG: Proyecto Fin de Grado

RAM: Random Access Memory (Memoria de acceso aleatorio)

ReLU: Rectified Linear Unit

RGB: Red Green Blue (Rojo Verde Azul)

TPU: Tensor Processing Unit (Unidad de procesamiento tensorial)

VLC: VideoLan Client

YAML: yet another markup language (Otro lenguaje de marcas más)

YOLO: You Only Look Once

Índice

1. Introducción

La acuicultura, un sector que encierra numerosas posibilidades en términos económicos y de sostenibilidad, se presenta como un campo fascinante que demanda exploración y mejora continua. Dentro de este contexto, se destacan tareas de gran interés, entre las cuales se encuentra el análisis del comportamiento de los peces en cautividad. La comprensión de estos patrones de comportamiento no solo contribuye al bienestar de los animales, sino que también impacta directamente en la calidad del producto final destinado al mercado alimenticio.

Para estudiar el comportamiento de las truchas en cautividad, se utiliza una prueba denominada "*open field*", en la cual se permite a las truchas nadar libremente en un tanque. Esta prueba es fundamental para analizar sus movimientos y determinar su carácter y estado de salud. Sin embargo, realizar este análisis manualmente es muy costoso en términos de tiempo. Automatizar este análisis podría generar grandes beneficios a los expertos en acuicultura, permitiendo un monitoreo eficiente y preciso del comportamiento de estos peces. En este proyecto, disponemos de vídeos grabados de este tipo de prueba que queremos analizar.

La herramienta principal para llevar a cabo este proyecto será la técnica del *tracking* o rastreo de movimiento [1]. Esta herramienta consiste en identificar, seguir y predecir objetos en secuencias de imágenes o videos de cierta duración. El principal objetivo es mantener un seguimiento continuo de la posición, velocidad u otras características del objeto (truchas, en nuestro caso) a rastrear dentro un espacio.

Para abordar este desafío, se emplearán redes neuronales convolucionales (CNNs), específicamente diseñadas para tareas de visión por computadora. Las CNNs han revolucionado el mundo de la inteligencia artificial, permitiendo resolver problemas de manera más eficiente y precisa debido a su capacidad de aprender representaciones de datos a partir de los píxeles de una imagen.

El sistema desarrollado no solo emplea técnicas de visión por computadora tradicionales, sino que también introduce una innovación específica para mejorar la eficiencia del modelo. Se ha implementado una técnica de zoom dinámico que permite centrar el procesamiento de la imagen en el área donde se encuentra la trucha. De este modo, se reduce el tamaño de la imagen a procesar, reduciendo el tiempo de procesamiento de esta. Además, no solo se mejora la eficiencia, sino que también hay una mejora en la precisión de la detección, debido a que orientamos al modelo hacia la zona donde se encuentra la trucha.

Finalmente, se desarrollará una interfaz gráfica de usuario (GUI) para que cualquier usuario, sin tener conocimientos de programación ni de los temas mencionados anteriormente, pueda realizar un seguimiento de la trucha de manera sencilla e intuitiva e invertir su tiempo en el análisis del comportamiento del animal.

1.1 Objetivos

Los principales objetivos de este proyecto son los siguientes:

- Desarrollar un algoritmo para determinar la posición espacial de la trucha mediante el seguimiento de esta.
- Desarrollar una técnica para optimizar la eficiencia y precisión del modelo
- Obtener resultados precisos con la mejor tasa de detección y eficiencia posible para mostrar información de manera compacta y realizar análisis cuantitativos.

1.2 Estructura de la memoria

La memoria está formada por las siguientes partes:

- Marco tecnológico: Esta sección explica las tecnologías y metodologías utilizadas en el proyecto, proporcionando un contexto teórico y técnico.
- Especificaciones y restricciones del diseño: Se detallan las especificaciones técnicas y las restricciones a las que está sujeta el proyecto.
- Descripción de la solución propuesta: En esta parte se explica todo el desarrollo del proyecto, tratando los problemas que han aparecido y sus posibles soluciones. Se explica en detalle el diseño del sistema y los algoritmos involucrados.
- Resultados: Se muestran y estudian los diferentes resultados obtenidos, evaluando los parámetros más importantes.
- Presupuesto: Esta parte proporciona una estimación detallada de los costes del proyecto.
- Impacto del proyecto: En esta sección se detalla que implicaciones sociales, de salud, seguridad, ambientales, económicas, tecnológicas e industriales ha tratado el proyecto, así como la contribución a los ODS.
- Conclusiones: se resumen los resultados más importantes del proyecto y se proponen futuras líneas de trabajo.
- Referencias: Esta parte incluye una lista completa de todas las fuentes bibliográficas y páginas webs han sido consultadas para el desarrollo del proyecto.

2. Marco tecnológico

En este capítulo se muestran las diferentes tecnologías que se han utilizado para el desarrollo. Se comienza con una introducción a la inteligencia artificial (apartado 2.1), luego se particulariza para el caso de detección de objetos (apartado 2.2), y se concluye con unas nociones de procesado digital de imágenes (apartado 2.3).

2.1 Inteligencia Artificial

La inteligencia artificial (IA) es un campo de la ciencia que ha irrumpido recientemente en nuestras vidas, despertando un gran interés y debate debido a su gran potencial para transformar y mejorar múltiples aspectos cotidianos. El propósito fundamental de la IA es emular el conocimiento humano no analítico, buscando que las máquinas puedan realizar tareas que usualmente requieren inteligencia humana [2].

En primer lugar, la IA es un área de estudio dentro de la ciencia computacional, que se enfoca en el desarrollo de los ordenadores capaces de realizar operaciones y procesos parecidos a los de los seres humanos, como el aprendizaje, el razonamiento y la autocorrección. Con esto se busca que estos puedan tomar decisiones y resolver problemas de manera autónoma, sin depender de una ayuda humana.

Además, la IA se puede entender como una extensión de la inteligencia humana, a través de un uso complementario de ella. La inteligencia artificial puede complementar las capacidades cognitivas de los humanos, facilitando tareas complejas o mejorando la eficiencia y precisión en distintas áreas.

Por último, la IA abarca el estudio de técnicas para realizar un uso de las computadoras más eficiente, mediante la mejora de técnicas de programación. Este enfoque se centra en optimizar el desarrollo de algoritmos y el uso de los recursos computacionales, para lograr unos objetivos de manera más rápida y eficiente [3].

La conceptualización de la inteligencia artificial se remonta a 1950, cuando Alan Turing propuso el famoso Test de Turing. Este experimento/prueba, que evalúa la capacidad de las máquinas para exhibir un comportamiento inteligente similar al humano, a través de la comunicación, consiste en una interacción en la que participan un interrogador humano y dos interrogados, uno humano y otra máquina. Durante la prueba, el interrogador formula preguntas a ambos sin saber quién es quién. Si al finalizar la prueba el interrogador no puede distinguir entre el humano y la máquina, se considera que la máquina ha superado el Test de Turing, demostrando así su capacidad para imitar la inteligencia humana de manera convincente [4], [5].

Finalmente, para comprender la arquitectura que forma la inteligencia artificial, es fundamental entender sus diferentes ramas (Figura 1). En este marco, el *Deep Learning* es una técnica específica que se encuentra dentro del *Machine Learning*, y a su vez, el *Machine Learning* es un conjunto de técnicas dentro del campo más amplio que es la inteligencia artificial [6].

Figura 1. Arquitectura de la Inteligencia Artificial. Adaptada de [7].

2.1.1 Aprendizaje Automático (*Machine learning*)

El aprendizaje automático o "*machine learning*" (ML) es la ciencia que se centra en el desarrollo de algoritmos y modelos estadísticos con el fin de poder realizar tareas específicas sin ser programados explícitamente para cada una de ellas. La premisa fundamental es encontrar la existencia de una relación entre los datos de entrada y los de salida. Este modelo no conoce de antemano esta relación, pero puede descubrirla mediante el análisis de grandes cantidades de datos.

Los distintos tipos de aprendizaje de ML proporcionan un marco diverso para abordar distintos problemas. Estos se podrían definir como: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

En el **aprendizaje supervisado**, el algoritmo escogido trabaja con datos "etiquetados" (*labeled data*), donde cada valor de entrada (input data) tiene asignada una etiqueta como valor de salida. El algoritmo se entrena con un conjunto de datos históricos, aprendiendo de estos ejemplos con el fin de que posteriormente pueda predecir las etiquetas de salida para nuevos valores de entrada. Este aprendizaje se suele usar para problemas de clasificación, como la identificación de dígitos en una imagen o clasificar imágenes de objetos, y regresión, como predicciones meteorológicas o de crecimiento. Este aprendizaje está en auge debido a los avances de tecnología y computación para procesamientos más rápidos, así como por el aumento de la disponibilidad de grandes volúmenes de datos que se pueden encontrar en la actualidad.

El **aprendizaje no supervisado** se utiliza cuando no se dispone de datos etiquetados para el entrenamiento. Este enfoque tiene un carácter exploratorio porque conocemos los datos de entrada, pero no existen las etiquetas de salida que teníamos en el aprendizaje supervisado. El objetivo principal es buscar y explorar la estructura inherente de datos e intentar encontrar patrones que simplifiquen el análisis. Esto se puede realizar a través de las tareas de *clustering* o agrupamiento, en las que se busca agrupar los datos en conjuntos basados en similitudes entre ellos.

El **aprendizaje por refuerzo** se basa en mejorar la respuesta del modelo a través de *feedback* o retroalimentación. El algoritmo aprende interactuando con el entorno, tomando decisiones y recibiendo una recompensa o penalización por sus acciones. Esta tiene como fin que aprenda a tomar las decisiones que maximicen su recompensa a largo plazo. Este proceso implica un ensayo constante de prueba y error, permitiendo al modelo aprender con la experiencia [8].

2.1.2 Aprendizaje Profundo (*Deep Learning*)

El aprendizaje profundo o "*Deep Learning*" (DL) es un algoritmo particular de aprendizaje automático, que enseña a los ordenadores a procesar datos de manera similar a como lo hace el cerebro humano mediante el uso de muchas capas de procesamiento. Mediante el uso de modelos de aprendizaje profundo, es posible identificar patrones complejos en imágenes, textos, sonidos y otros datos, con el fin de generar a la salida una información o predicción relevante. Estos modelos son archivos de computadora que se han entrenado a través de algoritmo o cierto número de pasos para ser utilizada en la automatización de actividades humanas.

Para comprender cómo funciona el DL es esencial conocer la estructura y funcionamiento de las redes neuronales

2.1.2.1 Elemento básico red neuronal (neurona)

2.1.2.1.1 Neurona artificial de McCulloch-Pitts

La neurona artificial de McCulloch-Pitts, también conocida como neurona MCP, es uno de los primeros modelos que aparecieron sobre las neuronas artificiales. Consiste en imitar el funcionamiento de una neurona biológica de una manera más simple. Esta recibe dos tipos de entradas: variables excitadoras $x_i \in \{0, 1\}$ y variables inhibidoras $x^*_j \in \{0, 1\}$. Ambas solo pueden tomar valores binarios, es decir 0 o 1, por lo que su principal uso es la implementación de funciones booleanas [9].

La neurona suma todas estas **entradas excitadoras** para determinar si debe activarse o no.

$$S = \sum_i x_i$$

Una vez realizada la suma (S), la neurona compara esta operación con un valor de umbral (θ). Si se supera este umbral y las entradas inhibidoras no están activas, la neurona se activa, es decir, produce a su salida un 1. Sin embargo, si no se supera este umbral, la neurona queda inactiva y produce un 0 a su salida (y).

Las **entradas inhibidoras** tienen mayor impacto en la decisión de la activación de la neurona, debido a que si alguna de estas entradas esta activa, el valor a la salida de la neurona será 0, independientemente de los valores de las entradas excitadoras.

Se podría expresar el comportamiento de la neurona de McCulloch y Walter Pitts mediante la siguiente fórmula:

$$y = \begin{cases} 1, & si\ (\sum_i x_i \geq \theta)\ \&\ (x^*{}_j\ = 0\ \forall j) \\ 0, & resto\ de\ casos \end{cases}$$

Si se observa detenidamente, se puede apreciar que esta neurona imita el comportamiento de una neurona biológica (Figura 2). Las entradas son las dendritas, el cuerpo de la neurona es el soma, y la salida es el axón de la neurona [10].

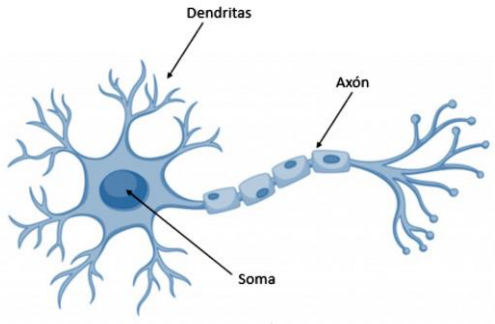

Figura 2. Partes de una neurona biológica **[11]**.

Entrando más en el ámbito geométrico, se puede interpretar el funcionamiento de estas neuronas desde una perspectiva geométrica. Al activar o desactivar la neurona mediante el cumplimiento de un umbral, realmente se está haciendo uso de un hiperplano de separación entre los dos casos. Este hiperplano viene definido como:

$$\sum_{i=1}^{n} x_i = \theta$$

Los puntos por encima del hiperplano provocan la activación de la neurona, y los valores por debajo de este provocan su no activación. En la figura 3, se ilustra este concepto:

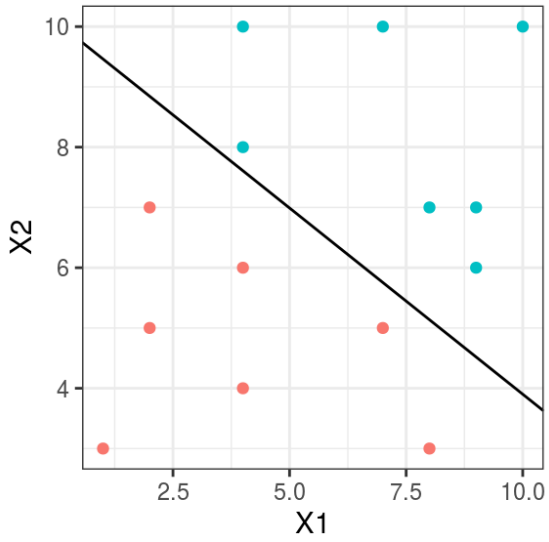

Figura 3. Hiperplano separando casos para activación o no activación de la neurona **[12]**.

Esta idea de neurona, formada por Warren McCulloch y Walter Pitts, presenta una serie de limitaciones. Entre ellas se encuentra la restricción a únicamente funciones booleanas, la necesidad de declarar el valor del umbral θ desde el principio y la imposibilidad de no poder asignar un mayor peso a distintas variables de entrada.

2.1.2.1.2 Perceptrón

Ante los problemas anteriormente mencionados, Frank Rosenblatt presentó el Perceptrón en 1958. Este modelo permite procesar cualquier valor real, lo que permite prescindir de las variables inhibidoras, ya que los valores negativos pueden simular su función. Esto se logra a través de la ponderación de las entradas con un peso específico en cada una de ellas (w), permitiendo controlar la importancia de cada entrada en la decisión final. A todo esto, se le añade un valor de sesgo o "bias" (b), que permite que el hiperplano no pase necesariamente por el origen. Tanto los valores de peso como de sesgo se suelen calcular durante la etapa de entrenamiento. La función del perceptrón se puede expresar como:

$$y = \begin{cases} 1, & si\ w^T * x + b > 0 \\ 0, & resto\ de\ casos \end{cases}$$

Donde:

- w : vector de los pesos de entrada. $w = [w_0, \dots, w_n]^T$
- x : vector de entradas. $x = [x_0, \dots, x_n]^T$
- b : valor de sesgo (bias)

En la figura 4, se ilustra el funcionamiento del perceptrón:

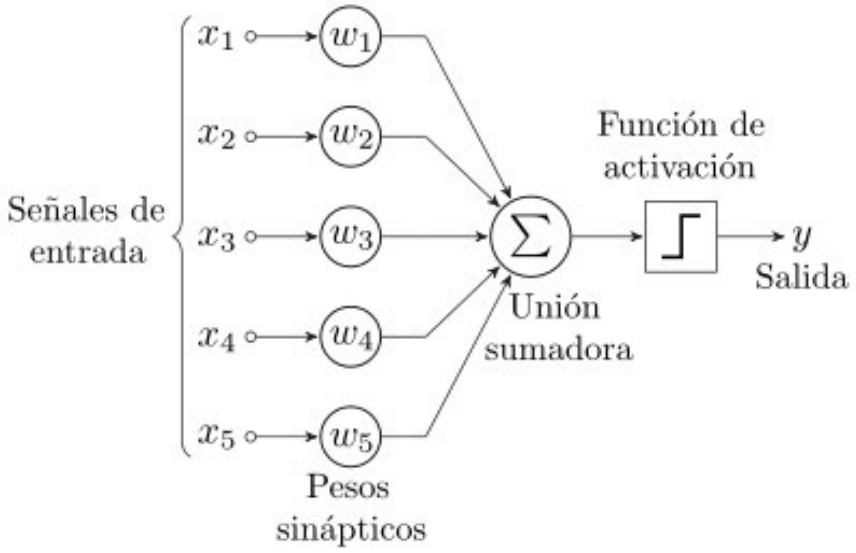

Figura 4. Funcionamiento del perceptrón **[13]**.

2.1.2.1.3 Función de activación

La función de activación es una función matemática que se aplica a la salida de una neurona o nodo antes de pasarla a la siguiente capa de una red neuronal. Su propósito es introducir una no linealidad en el modelo, permitiendo que la red trabaje y aprenda con casos más complejos. Los ejemplos más comunes de funciones de activación son: (Figura 5)

- Función escalonada: mayor simplicidad, limita los valores a 1 si son positivos y a 0 si son negativos. Se define como:

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

- Sigmoide: esta función tiende los valores de entrada entre un rango de 0 y 1, ideal para problemas de clasificación binaria. Se define como:

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

- Tangente hiperbólica (tanh): comportamiento similar a la sigmoide, pero con los valores tendiendo a -1 y 1. Se define como:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- RELU (Unidad lineal rectificada): es de las funciones más usadas. Se define como: [14]

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

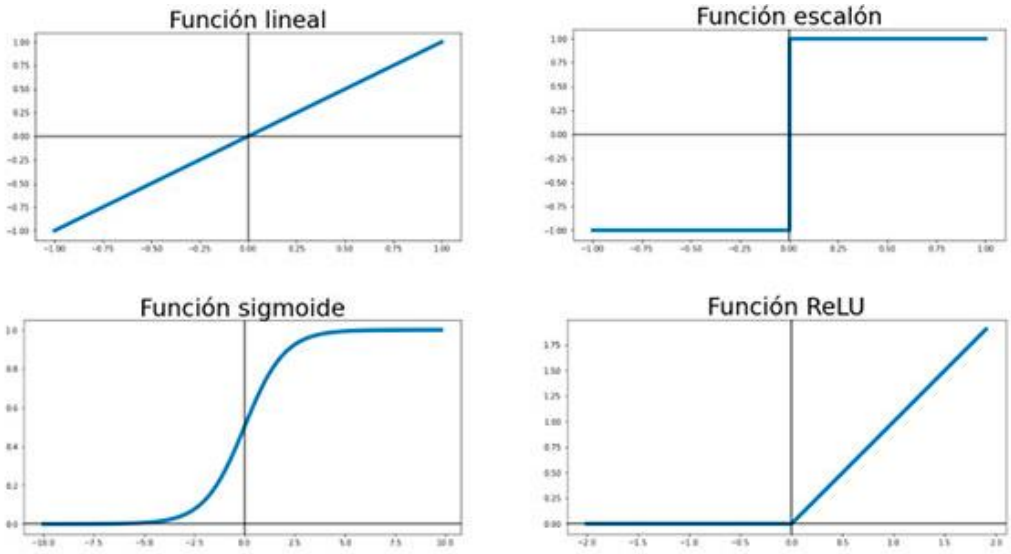

Figura 5. Funciones de activación más comunes **[15]**.

2.1.2.2 Redes neuronales

2.1.2.2.1 Definición y arquitectura

Es importante entender que el perceptrón es lo que se entiende hoy en día como neurona artificial, y no debe confundirse con la neurona de McCulloch-Pitts. La combinación de un grupo de perceptrones forma lo que se conoce como un perceptrón multicapa o, más comúnmente, una red neuronal artificial.

El primer componente de una red neuronal es la capa, la cual está formada por todas las neuronas que reciben la misma entrada. Para facilitar la comprensión, es común que las operaciones se denoten en formato vectorial. Las entradas a una capa se agrupan en un único vector $x = [x_o, \dots, x_n]^T$ siendo n el número de entradas, y x_o el valor de sesgo. Las salidas de la capa se agrupan en un único vector $y = [y_1, \dots, y_m]^T$, donde m el número de neuronas de esa capa, ya que cada capa puede tener un número distinto de neuronas.

Por último, la matriz de pesos, formada por los vectores de los pesos de cada neurona, se denota como $w^i = [{w_0}^i, \dots, {w_n}^i]^T$, donde i indica la neurona correspondiente. Cada vector w constituye una columna de la matriz de pesos W, la cual tiene n + 1 (correspondientes a n entradas más el sesgo) filas y m columnas. La expresión final de la operación sería:

$$y = W^T x$$

Por regla general, las neuronas de una misma capa comparten las mismas conexiones y funciones de activación. Dado que la función de activación $f()$ se aplica a la salida de cada neurona, podemos afirmar que la función de activación se puede aplicar al vector $W^T x$. De este modo, obtenemos finalmente el vector de salida de la capa:

$$y = f(W^T x)$$

En la figura 6, se presenta una ilustración para facilitar la comprensión de este concepto.

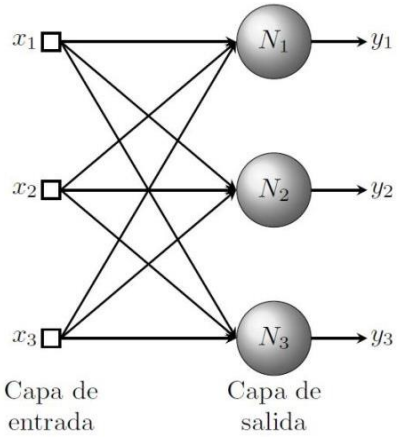

Figura 6. Representación de una capa de una red neuronal **[16]**.

Las capas juegan un papel crucial en el funcionamiento de la red neuronal y las podemos clasificar en tres clases distintas.

La **capa de entrada** es conocida como la capa 0. A diferencia del resto de capas, esta no está formada por neuronas, sino por los datos de entrada del modelo. Su función principal es recibir estos datos y seguidamente pasarlos a la siguiente capa sin realizar ningún procesamiento adicional.

Las **capas ocultas** y la **capa de salida** son las capas que están formadas por el conjunto de perceptrones. Cada perceptrón de esta capa cuenta con un peso que se encarga de ajustar la influencia de las entradas y aplica una función de activación para generar una salida. La diferencia entre estas capas es que las capas ocultas pasan este valor de salida a la siguiente capa, y la capa de salida es la última de la red generando el resultado final deseado.

La expresión para indicar el resultado final sería:

$$y^l = f^l\left(W^{l^T} x^l\right) = f^l\left(W^{l^T} f^{l-1}\left(W^{l-1^T} x^{l-1}\right)\right) = \ldots$$

donde

- y^l es el resultado final de la última capa
- f^i es la función de activación de la capa i-ésima
- W^i es la matriz de pesos de la capa i-ésima
- x^i es la entrada de la capa i-ésima

Se reemplazan los valores de entrada de la capa (x^i) con los valores de la salida de la capa anterior (y^{i-1}), dado que son los mismos datos. Esta expresión se extiende hasta alcanzar la primera capa. La organización de estas capas se puede apreciar en la figura 7 [9].

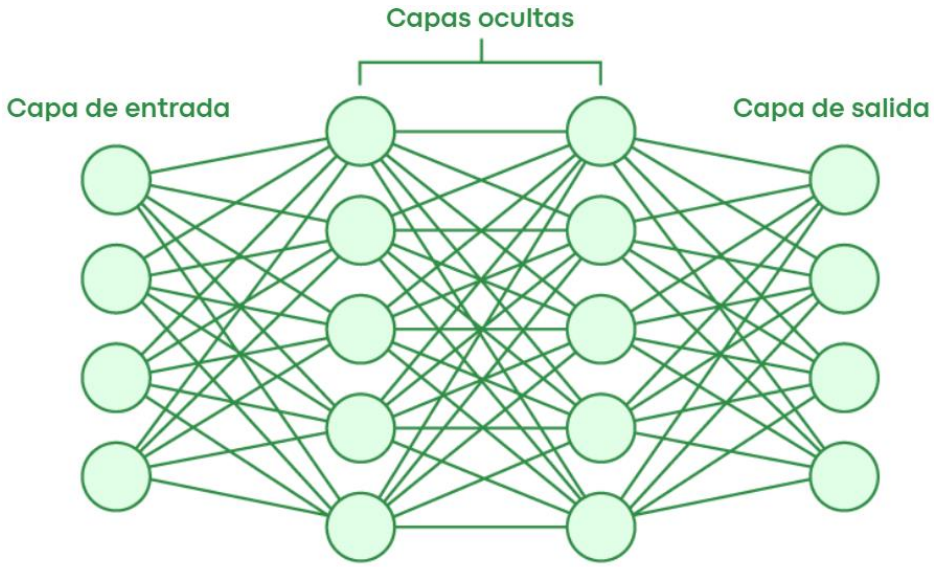

Figura 7. Capa de entrada, capas ocultas y capa de salida de una red neuronal. Adaptada de **[10]**.

2.1.2.2.2 Entrenamiento de la red

Las redes neuronales se basan en el entrenamiento de la red a través de un *dataset* para que aprenda y mejore su precisión en cada iteración. Un *dataset* es el conjunto de datos imprescindible para entrenar una red neuronal, y su calidad y cantidad afecta directamente en el rendimiento del modelo. El proceso de entrenamiento consiste en modificar iterativamente los pesos de las conexiones neuronales con el objetivo de minimizar el error en la salida. Este ajuste se realiza mediante distintos algoritmos, como el algoritmo de retropropagación (*backpropagation*) que se presentará más adelante. Una vez entrenadas, estas redes se convierten en herramientas potentes capaces de resolver de forma precisa, problemas y cálculos para los que han sido preparados. Se entiende que una red ha finalizado su aprendizaje cuando los pesos apenas se modifican de una iteración a otra.

El proceso de entrenamiento de una red neuronal está formado por grupos (*batches*) y épocas (*epochs*). Este entrenamiento se compone de épocas y cada época trabaja con un subconjunto de datos. En cada época del entrenamiento se trata con un subconjunto del conjunto de datos, llamado "*batches*". Esto sirve, para no trabajar con todo el conjunto de datos en cada época, permitiendo un procesamiento más veloz.

Si el entrenamiento de una red neuronal estuviera compuesto de pocas épocas, la red no se habría entrenado lo suficiente, provocando un infra ajuste (*underfitting*). Por otro lado, si se realizan demasiadas épocas, la red sufriría de un sobre ajuste (*overfitting*). Este es un problema común en el entrenamiento de redes neuronales, donde el modelo se ajusta demasiado bien a los de entrada, pero falla al generalizar en datos nuevos no vistos durante esta etapa [17].

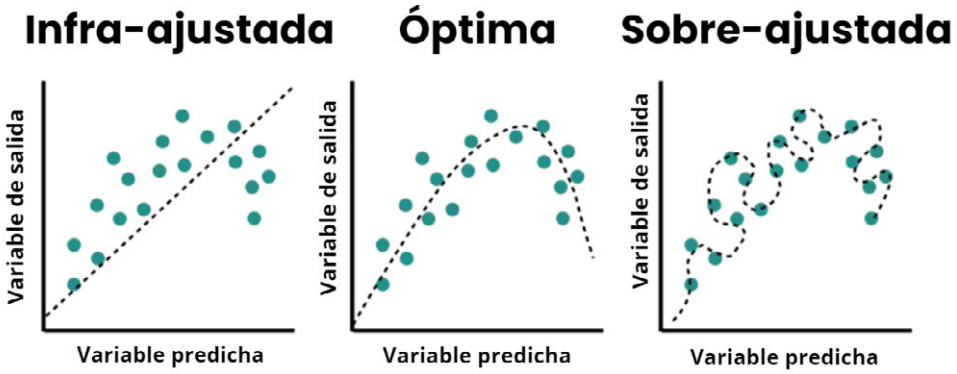

Figura 8. Ejemplo de *underfitting* y *overfitting*. Adaptada de **[18]**

Como se puede observar en la figura 8, se muestra el ejemplo de una red infra-ajustada, la cual no se adecua a los valores de entrada, una red entrenada adecuadamente, por lo que se adecua a la tendencia en la media de los datos y, finalmente, una red sobre-ajustada, en la que la red aprende tal nivel de detalle, que ante cualquier anomalía mínima se obtiene un resultado incorrecto.

2.1.2.2.3 Descenso del gradiente

El descenso del gradiente es un algoritmo de optimización que se utiliza para entrenar las redes neuronales o modelos de *machine learning*. Este algoritmo se basa en la búsqueda de la reducción máxima de pérdidas de la red.

El objetivo principal es descender lo máximo posible en la función convexa. El punto de partida es aleatorio y para obtener la inclinación de la pendiente, se va obteniendo la derivada de ese punto. A partir del reconocimiento de la pendiente se va desplazando el punto hacia el eje negativo del eje y, hasta llegar al punto con derivada igual a cero. Este punto es conocido como punto de convergencia, y al tener derivada nula significa que se ha llegado al punto más bajo de la función convexa.

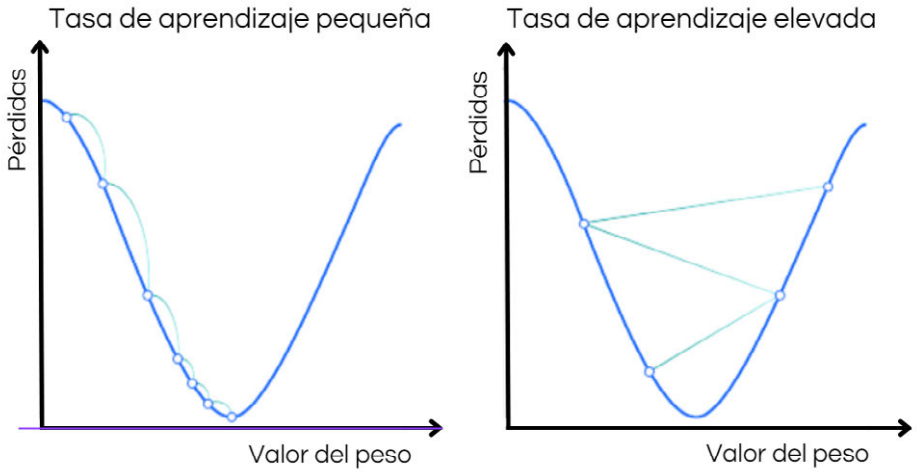

Figura 9. Función convexa con tasa de aprendizaje baja y alta. Adaptada de **[19]**.

Los dos factores involucrados en este algoritmo son la tasa de aprendizaje y la función de pérdida. La tasa de aprendizaje marca el tamaño de los pasos: si este es muy pequeño se necesitarán muchas iteraciones para alcanzar el punto de convergencia. En cambio, si la tasa de aprendizaje es muy elevada, se corre el riesgo de sobrepasar el mínimo e incluso llegar a un bucle infinito en el que nunca se obtendrá el punto de convergencia. La función de pérdida mide el error entre el descenso real y el descenso pronosticado, proporcionando retroalimentación al modelo para que pueda ajustar sus parámetros y reducir el error [19].

En la figura 9, se pueden observar los dos casos de la variación de la tasa de aprendizaje.

2.1.2.3 Redes neuronales Convolucionales

Las redes neuronales convolucionales son un modelo de red neuronal basado en el funcionamiento del neocórtex, especialmente en como la corteza visual procesa la información. El neocórtex es la parte del cerebro que se encarga de realizar las funciones más complejas de los seres humanos.

En 1959, dos profesores llamados David Hubel y Torsten Wiesel realizaron un experimento analizando el comportamiento de la activación de las neuronas de un gato al observar dibujos. Con este experimento, se descubrieron dos tipos de neuronas:

- Células simples: se activan con líneas de orientación específica.
- Células complejas: se activan con líneas de orientación específica que además se desplazan en una dirección.

Estas neuronas están organizadas jerárquicamente. Las células simples detectan características básicas como líneas o bordes, y luego transmiten esta información a las células complejas que integran estas características para detectar patrones más complejos.

En 1980 se desarrolló uno de los primeros modelos de red neuronal llamado Neocognitron. Este modelo procesa imágenes mediante capas secuenciales que imitan el comportamiento de las neuronas anteriormente mencionadas. Cada capa está formada por dos tipos de células diferentes que son:

- Célula S: Extraen características de bajo nivel, imitando el comportamiento de las neuronas simples.
- Célula C: Extraen características de alto nivel, imitando el comportamiento de las células complejas.

Estas células se distribuyen en planos dentro de una cuadrícula y cada célula en un plano comparte los mismos pesos y analiza diferentes regiones de entrada.

A diferencia de una red neuronal tradicional, este modelo respeta la estructura de los datos y no convierte la entrada en vector. Las células procesan pequeñas partes de la imagen mediante convoluciones, comparando la región actual con los pesos de la característica

que detecta la célula. Las células C utilizan una operación de agrupamiento (*pooling*), para combinar las salidas de diferentes células S.

Finalmente, en el año 1989, Yann LeCun y su equipo presentaron la primera red neuronal convolucional (CNN) de la historia, siendo entrenada con el algoritmo de *backpropagation.*

La arquitectura de una red neuronal convolucional tiene varios aspectos distintos a una red neuronal tradicional. En una CNN, se preserva la estructura espacial de los datos siempre que se pueda para facilitar la extracción de características. Esto se logra mediante el uso de las capas de convolución y *pooling*, que aprovechan la relación espacial de los datos para extraer características relevantes.

Una característica distintiva de las CNN es su capacidad para trabajar con datos de diferentes dimensiones. Un escalar, es un tensor de orden 0, un vector es un tensor de orden 1, una matriz es un tensor de orden 2, y de forma general, un dato con dimensión n es un tensor de orden n [20]. En la figura 10, se ilustra este concepto.

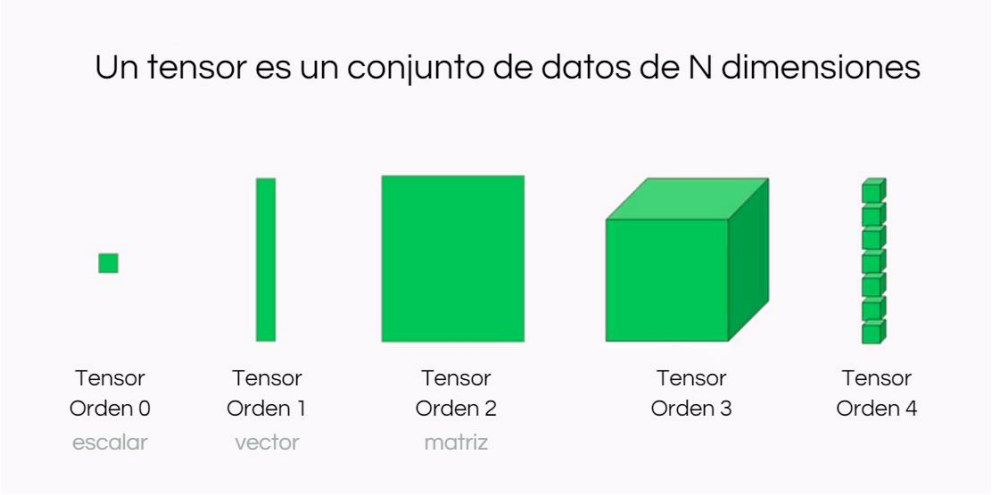

Figura 10. Órdenes de un tensor. Adaptada de **[21]**.

Para extraer características, las CNNs utilizan sucesiones de capas de convolución y *pooling*. Estas capas se encargan de la etapa de extracción de características de los datos de entrada. Posteriormente, en la etapa de transformación, las redes neuronales convoluciones emplean redes neuronales tradicionales, conocidas como capas completamente conectadas. Estas, como indica el nombre de su etapa, se encargan de transformar los datos obtenidos tras pasar por las capas de convolución y *pooling*. Esta estructura permite que las CNNs sean flexibles y adaptables a una gran cantidad de tipos de tareas o de datos (Figura 11). En las siguientes secciones, se explica en detalle las capas mencionadas y su funcionamiento.

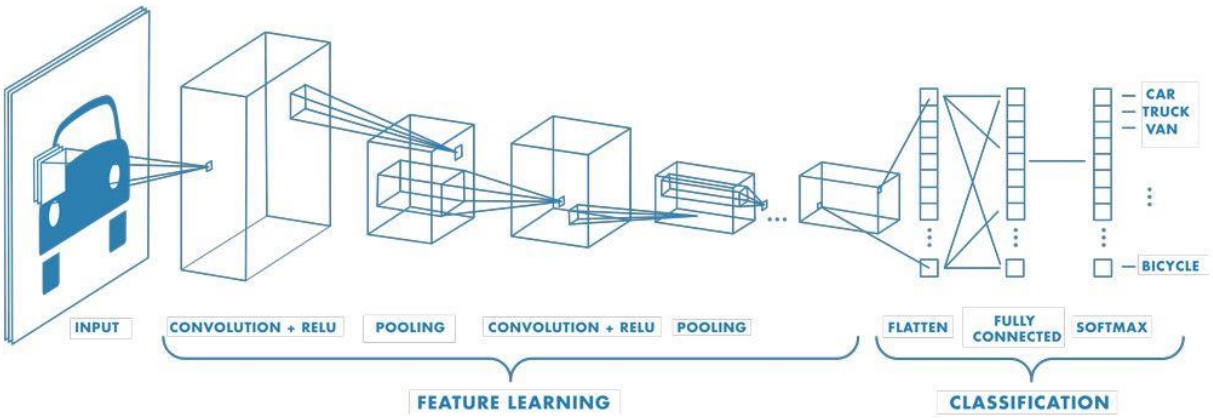

Figura 11. Capas de una red neuronal convolucional **[22]**.

a) Capa de convolución

La capa de convolución se encarga de extraer características de los datos de entrada. Esta capa aplica una serie de convoluciones a los datos de entrada, estos pueden ser los datos de entrada de la propia red neuronal o los datos de salida de una capa anterior. La entrada y salida de la capa se denomina mapa de características de entrada o de salida, respectivamente.

La **convolución** es un operador matemático que realiza la transformación de dos funciones f y g mediante la integración del producto de las dos funciones, con una de ellas invertidas y desplazada cierta distancia; esta última recibe el nombre de *kernel*. Su función matemática discreta es:

$$(f * g)(x) = \sum_i f(i) * g(x - i)$$

Durante el entrenamiento, los pesos del *kernel* se ajustan iterativamente mediante algoritmos como la retropropagación para minimizar el error de salida de la red. Aunque los pesos se comparten a lo largo de la imagen (lo que se denomina compartición de parámetros), estos pesos se actualizan en cada iteración de entrenamiento.

Tres hiperparámetros influyen en el tamaño del volumen de salida de la capa:

- El número de *kernels* determina la cantidad de mapas de características generados. Aumentar este número supone más mapas de características diferentes.
- El *stride* o paso, indica la distancia o número de datos que el *kernel* se desplaza en cada paso. Un mayor desplazamiento resulta en una salida de menor tamaño
- El *zero-padding* se utiliza cuando el *kernel* no se ajusta perfectamente a los datos de entrada, dado que puede sobresalir por los bordes. Este añade ceros para permitir poder realizar la convolución correctamente.

Finalmente, después de cada operación de convolución, la CNN aplica la función de activación ReLU, introduciendo no linealidad al modelo [23]. En la figura 12, se ilustra la convolución aplicada en una matriz.

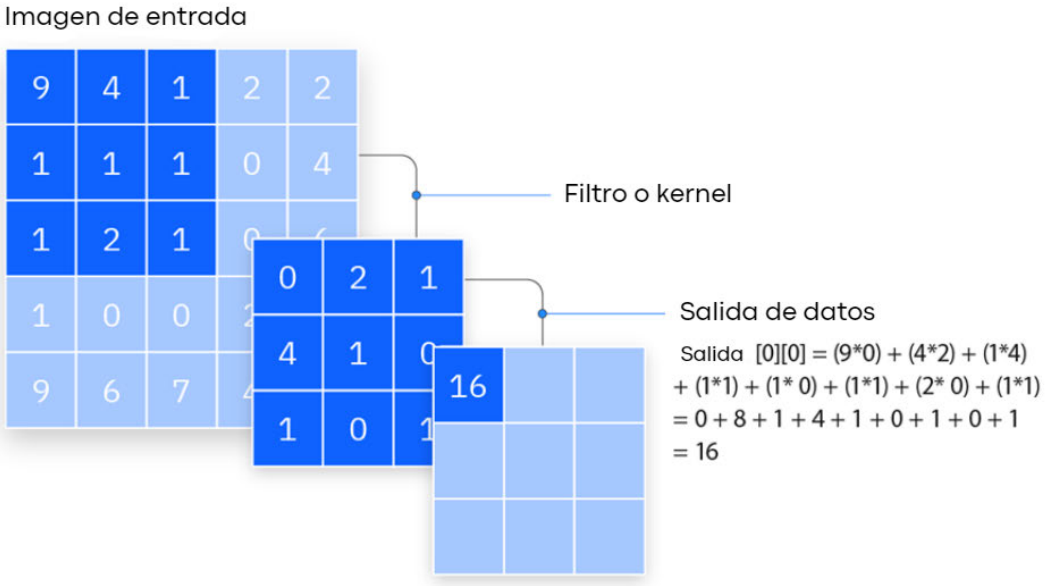

Figura 12. Operación de convolución aplicado en una matriz. Adaptada de **[23]**.

b) Capa de *pooling*

La operación de *pooling* o agrupación desempeña otra función fundamental en las redes neuronales convolucionales. Su función principal es reducir la dimensión de los datos de entrada, disminuyendo el número de parámetros. A diferencia con las capas de convolución, donde los *kernels* tenían pesos variantes durante el entrenamiento, en las capas de *pooling*, las ventanas no tienen pesos asociados. En su lugar, se realiza una función de agregación, como máximo o media, a los valores de dentro de la ventana. Existen dos tipos principales de agrupación:

- *Max-pooling*: Esta es la técnica más utilizada. A medida que la ventana se desplaza sobre los datos de entrada, selecciona el valor más alto de todos los valores y lo asigna como salida.
- *Average-pooling*: En este enfoque, se calcula el valor medio de todos los valores dentro de la ventana mientras se desplaza sobre los datos de entrada, y este valor medio lo asigna a la salida.

Estas operaciones de agrupación ayudan a reducir la cantidad de información, la complejidad del modelo y mejorar la eficiencia. En la figura 13, se pueden observar estos dos tipos de agrupaciones.

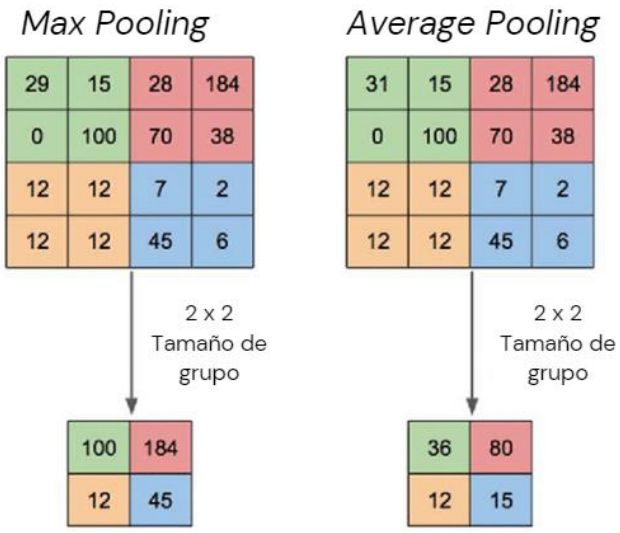

Figura 13. Técnicas de *Max pooling* y *Average Pooling*. Adaptada de**[24]**.

2.2 Seguimiento de Objetos

El seguimiento de objetos u "*object tracking*" es una técnica fundamental en visión por computadoras que involucra una continua monitorización sobre la posición de un objeto en un vídeo. Por otro lado, la detección de objetos es una técnica empleada en visión por computadoras para identificar, clasificar y localizar un objeto en una imagen.

Ambas técnicas están intrínsecamente relacionadas, ya que el seguimiento de objetos es, esencialmente, una detección de objetos continua en una sucesión de fotogramas. Por esta razón, se suelen implementar de manera complementaria en muchas aplicaciones.

El procedimiento que siguen estás técnicas es clasificar el objeto detectado, es decir, decir qué es, y delimitar su localización con una caja delimitadora o *bounding box*. El objetivo del seguimiento de objetos es monitorizar y mantener la localización del objeto de la manera más precisa posible a lo largo del tiempo o secuencia de vídeo, sin importar los cambios en el tamaño, posición, luminosidad o condiciones de entorno.

Esta técnica utiliza aprendizaje supervisado y requiere de un conjunto de datos etiquetados para aprender a detectar el objeto en imágenes futuras. Entre sus aplicaciones más comunes se encuentran:

- Vigilancia y seguridad en sistemas de cámaras de seguridad
- Conducción autónoma en vehículos
- Interfaces de usuario basadas en gestos humanos
- Robots móviles
- Seguimiento de objetos en realidad aumentada
- Monitorización del tráfico

El objeto detectado va a estar delimitado por la *bounding box*. Este contiene al menos cuatro valores con los que indica la posición y puede contener más para indicar diferentes parámetros. Hay diferentes formatos para su expresión como YOLO, PASCAL VOC y COCO. En este proyecto, se ha utilizado el formato YOLO, que se compone de un identificador de clase y cuatro coordenadas normalizadas[25]:

- Identificador de clase: es un número que indica que objeto es
- $x_{central}$ e $y_{central}$ son las coordenadas normalizadas del centro de la caja
- *width*: es el ancho de la caja normalizado
- *height*: es la altura de la caja normalizada
- Formato: id $x_{central}$ $y_{central}$ width height

Para la normalización de coordenadas, se utilizan las siguientes fórmulas:

$$x_{central} = \frac{centro\ bounding\ box\ en\ el\ eje\ x}{ancho\ del\ fotograma} \qquad y_{central} = \frac{centro\ bounding\ box\ en\ el\ eje\ y}{alto\ del\ fotograma}$$

$$width = \frac{ancho\ de\ la\ bounding}{ancho\ del\ fotograma} \qquad height = \frac{alto\ de\ la\ bounding\ box}{alto\ del\ fotograma}$$

Estos cuadros pueden solaparse y suelen llevar asociados un identificador que indique que objeto es y la confianza que tiene el modelo en cada objeto. En la figura 14 se observa lo mencionado:

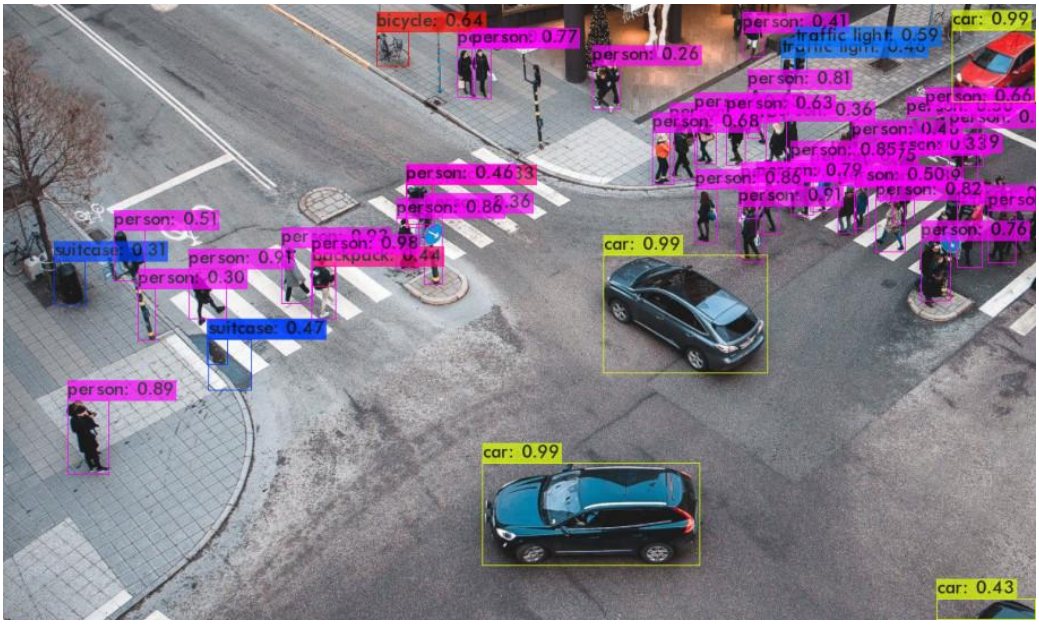

Figura 14. Detección de objetos en una imagen con bounding boxes **[26]**.

Además de definir y utilizar la *boundig box* es importante entender y utilizar ciertos parámetros que ayudan a mejorar su precisión. Uno de los parámetros más importante es el IoU (*Intersection over Union*).

El IoU es una métrica que se utiliza para evaluar la precisión de la detección de un objeto, midiendo el grado de solapamiento entre la *bounding box* predicha y la real.

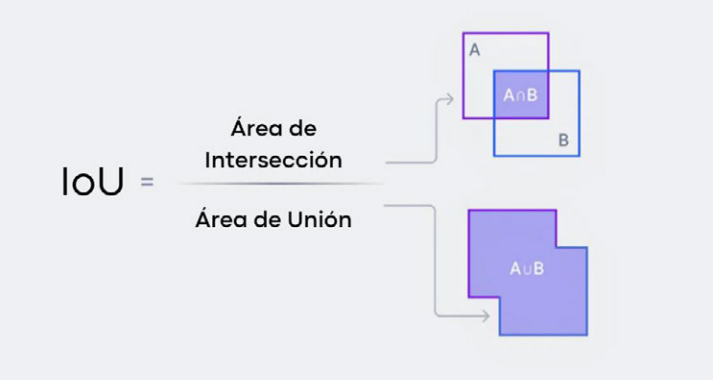

Figura 15. Explicación gráfica de la métrica IoU. Adaptada de **[27]**.

En la figura 15, se puede observar que está formada por 2 áreas que son:

Área de Intersección o *Area of Overlap* es el área donde se solapan la caja predicha y la real. Podemos calcularla mediante las siguientes fórmulas:

- $x_{min} = \max\left(x_{min}^{pred}, x_{min}^{true}\right)$
- $y_{min} = \max\left(y_{min}^{pred}, y_{min}^{true}\right)$
- $x_{max} = \min\left(x_{max}^{pred}, x_{max}^{true}\right)$
- $y_{max} = \min\left(y_{max}^{pred}, y_{max}^{true}\right)$

$$Área\ de\ Intersección = \max(0, x_{max} - x_{min}) \times \max(0, y_{max} - y_{min})$$

Área de Unión o *Area of Union* es el área que forma la unión de estas dos cajas delimitadoras.

$$Área\ de\ Unión = Área\ de\ la\ predicha + Área\ de\ la\ real - Área\ de\ Intersección$$

Para un valor de IoU superior o igual a 0.5 la predicción se considera un verdadero positivo; si no se, considera un falso positivo [28]. Este concepto se ilustra en la figura 16.

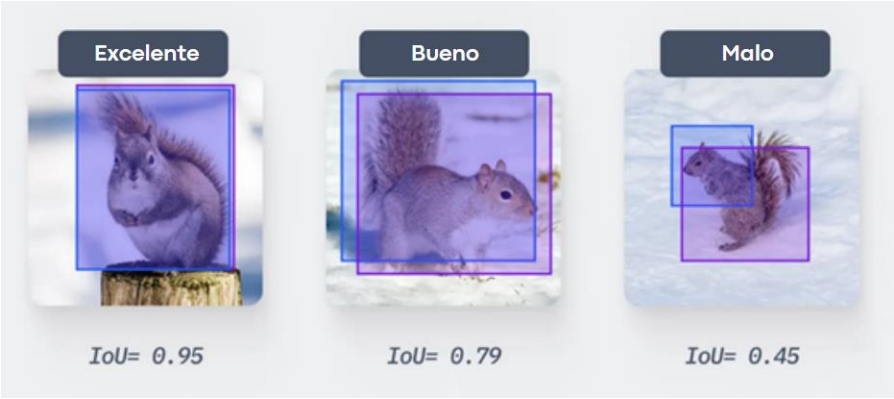

Figura 16. Casos de IoU. Adaptada de **[27]**.

2.2.1 YOLO

2.2.1.1 Introducción

YOLO, que significa "*You Only Look Once*", es un algoritmo de detección de objetos en tiempo real desarrollado por Joseph Redmon y Ali Farhadi en 2015. El objetivo principal de YOLO es realizar la detección en una imagen en una sola pasada a través de la red neuronal, lo que mejora la rapidez y lo hace adecuado para aplicaciones en tiempo real. A lo largo de los años, YOLO ha evolucionado con versiones cada vez más eficientes y precisas, incluyendo YOLOv2, YOLOv3, YOLOv4 hasta llegar a YOLOv8. Actualmente se está desarrollando YOLOv9, que promete continuar con esta tendencia de mejora continua.

2.2.1.2 Arquitectura

Red convolucional: YOLO utiliza una red neuronal convolucional profunda. Las primeras versiones estaban formadas por 24 capas convolucionales seguida de 2 capas completamente conectadas. Estas estructuras han ido evolucionando hasta llegar a arquitecturas muy complejas con mayor profundidad y robustez. YOLOv8 usa una arquitectura de espina dorsal (*Backbone*) y cuello (*Neck*) de última generación. El *backbone* consiste en una serie de capas convolucionales que se encargan de extraer características de bajo nivel y alto nivel de la imagen de entrada. El *neck* se encuentra entre el *backbone* y la cabeza de detección. Su principal objetivo es fusionar y refinar las características obtenidas gracias al *backbone* para luego pasarlo a la cabeza de detección (figura 17) [29], [30].

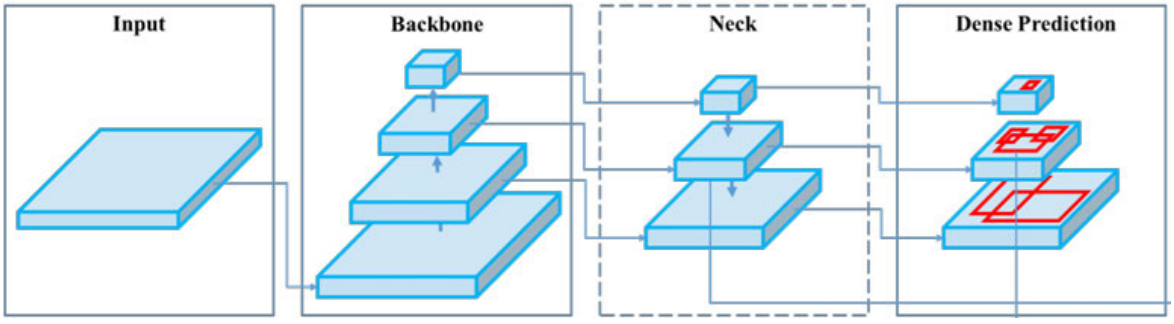

Figura 17. Detector de objetos con YOLOv4 **[31]**.

División de la Imagen: Yolo divide la imagen de entrada en una cuadrícula S x S. Para cada celda YOLO predice un número de *bounding boxes*. Cada *bounding box* contiene las coordenadas del centro, la anchura, la altura, la confianza y su probabilidad de clasificación. La confianza refleja con qué certeza el modelo está seguro de que el objeto se encuentra dentro de la caja delimitadora.

Parámetros de la Predicción: Los parámetros predichos, que permiten al modelo identificar y localizar, para cada *bounding box* son sus coordenadas y dimensiones (x, y, w, h), confianza y clase (Figura 18).

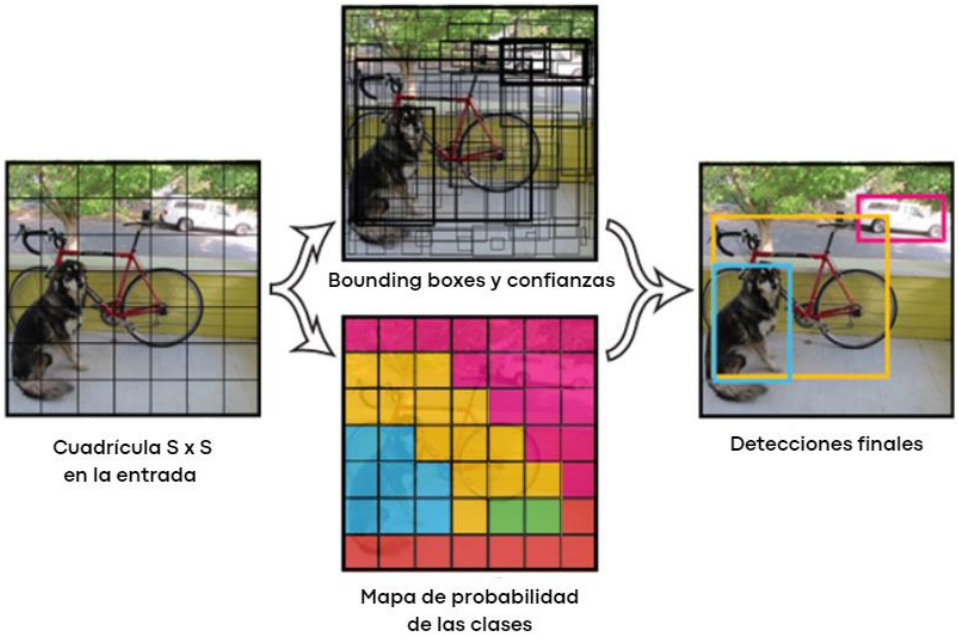

Figura 18. Funcionamiento de la detección de objetos con YOLO. Adaptada de **[29]**.

Finalmente, Yolo destaca por su velocidad, permitiendo procesar imágenes en tiempo real, ideal para aplicaciones de vigilancia o conducción autónoma. Además, ofrece una alta precisión en sus detecciones, convirtiéndolo en un algoritmo muy potente y eficaz.

2.3 Tratamiento de imágenes

2.3.1 Definición

El tratamiento de imágenes es el proceso de manipular y editar una imagen mediante técnicas computacionales. Este campo abarca diversas tareas como la mejora de la calidad, segmentación, reconocimiento de patrones y extracción de características. Estas últimas muy relacionadas con el mundo de las redes neuronales. El principal objetivo es transformar una imagen en una forma que sea más adecuada o para extraer información de ella.

2.3.2 OpenCV

OpenCV (*Open Source Computer Vision Library*) es una biblioteca de código abierto que incluye varios cientos de algoritmos de visión por ordenador. Fue desarrollado en 1999 por Intel, lanzado bajo una licencia de código abierto. Desde entonces ha ido evolucionando y se ha desarrollado con contribuciones de la comunidad.

OpenCV se puede ejecutar en diferentes plataformas y su biblioteca incluye más de 2500 algoritmos optimizados para el procesamiento de imágenes y visión por computadora. Además, se integra bien con bibliotecas relacionadas con la inteligencia artificial como PyTorch y TensorFlow [32].

2.3.3 Data augmentation

Data augmentation es una técnica utilizada para aumentar la cantidad y diversidad de los datos disponibles en un *dataset*, mediante la creación de nuevas versiones de imágenes existentes. Esto se logra modificando parámetros de las imágenes originales.

La diversidad de datos de entrenamiento es fundamental para un correcto funcionamiento de algoritmos precisos y robustos. En muchos casos, puede ser complicado o costoso obtener un conjunto de imágenes grande y etiquetado para el entrenamiento de la red neuronal. Data augmentation ofrece una solución a este problema, aumentando la cantidad de datos a partir de nuevas muestras de datos. Con ello los modelos aprenden a tratar con situaciones y condiciones más complejas y variables y previenen el *overfitting.* Las transformaciones más comunes son:

- Rotación: girar la imagen de manera aleatoria.
- Traslación: desplazar el fotograma vertical o horizontalmente, para cambiar la posición de dentro del marco
- Escalado: cambiar el tamaño de escala, tanto agrandar como encoger.
- Volteo: Invertir la imagen vertical o horizontalmente
- Brillo y contraste: variar estos parámetros en rangos lógicos.
- Aplicación de ruido: añadir ruido a la imagen
- Recorte aleatorio: recortar la imagen aleatoriamente
- Transformaciones de color: variar los colores de la imagen con la saturación, balance de blancos, etc.

Todas estas transformaciones pueden aplicarse una sobre otras, para así incrementar, aún más, el conjunto de imágenes. En la figura 19, se pueden observar estas transformaciones [33], [34].

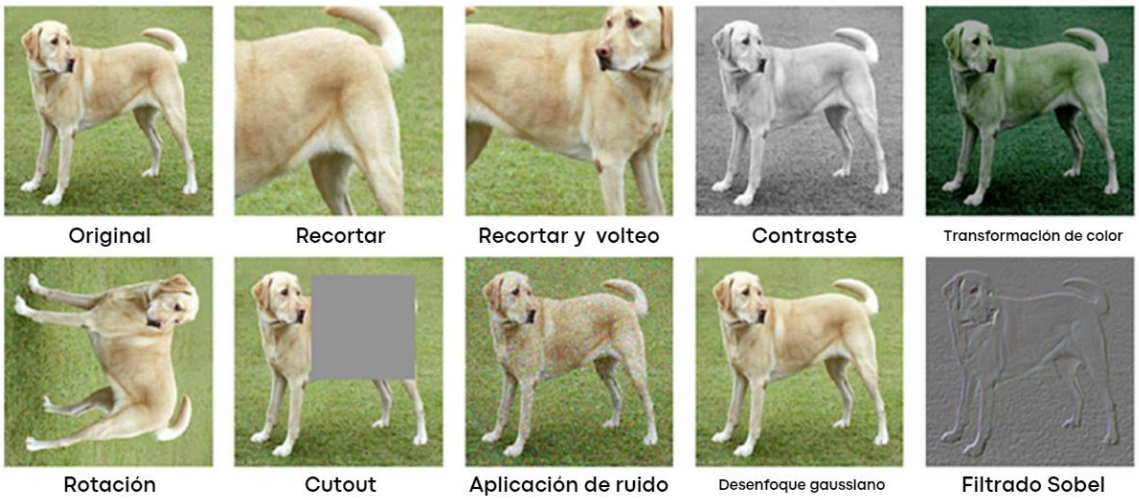

Figura 19. Transformaciones de imágenes. Adaptada de **[35]**.

3. Especificaciones y restricciones de diseño

Como ya se ha comentado en el apartado 1.1, el objetivo principal de este proyecto es el desarrollo de un sistema, basado en aprendizaje profundo, capaz de realizar un seguimiento continuo de un objeto en un entorno cerrado, en concreto, el de una trucha en un tanque. Para conseguir ese objetivo, se fijan una serie de especificaciones mínimas que debe alcanzar el sistema. También se listan unas restricciones de partida que limitan el alcance del proyecto y que son conocidas antes del comienzo del mismo.

3.1 Especificaciones de diseño

- **Precisión de seguimiento:** El sistema deberá de ser capaz de detectar y seguir con precisión el objeto durante la mayor parte de la duración de todo el vídeo.
- **Eficiencia computacional:** Como el seguimiento implica un procesamiento imagen a imagen, el sistema deberá de ser eficiente en cuanto al consumo de recursos, para garantizar un procesamiento del vídeo completo en una duración de tiempo relativamente corta.
- **Adaptabilidad:** el sistema deberá de ser capaz de reconocer el objeto en diferentes condiciones de la pecera, como la nitidez del agua, el tamaño de la trucha o el cambio de su fondo.
- **Exclusividad:** el sistema será entrenado para el seguimiento de un pez en una pecera.
- **Interfaz de usuario:** para facilitar su uso, se diseñará una interfaz gráfica para poder realizar el seguimiento sin tener conocimientos de programación, redes neuronales, inteligencia artificial, etc.

3.2 Restricciones de diseño

- **Número de Clips**: El conjunto de vídeos disponible con seguimiento del pez manualmente, está formado por 60 vídeos, del Clip 1 al Clip 30 y del Clip 61 al Clip 90. Estos 30 últimos clips son con los que se trabajan para el resultado final, porque tienen resultados manuales relevantes para futuras comparaciones.
- **Base de datos**: La base de datos se obtendrá de los Clips 1, 2, 3, 4, 5, 29 y 30 y se repartirán entre imágenes de entrenamiento e imágenes de validación.
- **Etiquetas**: Todas las imágenes han sido etiquetadas por el autor manualmente, por lo que el etiquetado no es 100% preciso.
- **Librerías**: El proyecto se ha desarrollado con librerías de código abierto.
- **Entrenamiento red neuronal**: el entrenamiento ha sido realizado en el entorno de programación *Google Colab* con el lenguaje de programación Python.

- **Diseño interfaz gráfica**: el diseño se ha realizado con *Visual Studio Code* con el lenguaje de programación de Python.
- **Dificultades del vídeo**: El fondo de la pecera es parecido a la trucha, por lo que, en condiciones muy desfavorables, como el agua muy sucia/turbia o partes del pez tapadas, su detección es complicada hasta para el ojo humano.

4. Diseño del sistema

En este capítulo, se aborda el diseño del sistema para el seguimiento de truchas en un tanque mediante el uso de redes neuronales convolucionales. Este capítulo detalla las etapas del diseño, desde la recopilación de imágenes hasta la implementación del algoritmo de seguimiento, y el desarrollo de la interfaz gráfica. Se describen las decisiones técnicas, los desafíos encontrados y las soluciones adoptadas para garantizar un sistema robusto y eficiente. Además, se presentan las estrategias utilizadas para optimizar el rendimiento del sistema y la precisión del modelo.

4.1 Recopilación de imágenes

La recopilación de imágenes es un paso muy importante en el entrenamiento de una red neuronal, debido a que su rendimiento dependerá en gran medida de la calidad y nitidez de las imágenes utilizadas. Inicialmente, se obtuvieron imágenes de truchas de diversas fuentes en línea. Sin embargo, rápidamente se observó que con este conjunto de imágenes no se alcanzaban los resultados esperados durante el entrenamiento de la red neuronal. La variabilidad del tamaño del objeto, la iluminación y el contraste era muy amplia, lo que permitía entrenar una red para muchos entornos y casos, pero poco eficientes en cada uno de ellos.

El principal motivo que desembocó el descarte de esta idea fue la limitada diversidad en las posturas de las truchas. La gran mayoría de las imágenes mostraban la trucha de perfil, mientras había un número muy escaso de imágenes tomadas desde ángulos frontales o traseros. Esta falta de diversidad dificultaba el entrenamiento óptimo de la red neuronal para reconocer la trucha en diferentes orientaciones.

Para abordar este problema, se decidió extraer imágenes directamente de los propios vídeos disponibles. Este enfoque aseguraba que las imágenes extraídas estuvieran en el mismo contexto y condiciones que los vídeos, minimizando así los posibles errores.

Como se explicó en el apartado de restricciones de diseño (apartado 3.2), se dispone de un total de sesenta vídeos denominados Clip 1, Clip 2, Clip 3, y así sucesivamente hasta Clip 30, y luego Clip 61, Clip 62, Clip 63, hasta el Clip 90. Los clips entre el 61 y 90 están destinados a los resultados, por lo que la extracción de imágenes solo se podrá ser aplicada en los 30 primeros vídeos.

Todos los vídeos disponibles tienen una duración aproximada de 5 minutos y un tamaño alrededor de 680 MB. Funcionan a 25 fotogramas por segundo (FPS) y está en formato de archivo MTS. En cuanto al contenido, cada vídeo muestra una trucha nadando libremente sobre un fondo que simula una pared natural. Al principio y al final de los vídeos se

observan reflejos de una puerta. La pantalla se divide en 6 cuadrantes, y en ocasiones, el marco del tanque puede obstruir la vista de la trucha.

Para la obtención de las imágenes se utilizó el software VLC [36], capturando un fotograma cada cierto número de fotogramas para obtener posiciones del sujeto completamente aleatorias. Se utilizaron los vídeos Clip 1, Clip2, Clip3, Clip 4 y Clip 5 obteniendo un total de 496 imágenes.

Una vez obtenidas los fotogramas, se procedió a etiquetarlas con el programa CVAT [37]. Para cada imagen se dibujó un rectángulo que delimitará lo más ajustado posible, la localización del pez. Este rectángulo, conocido como "*bounding box*", define los límites del objeto dentro de una imagen (Figura 20) [38].

Esta tarea de etiquetado se realizó meticulosamente y fue esencial para asegurar que la red neuronal pudiera detectar de manera precisa el individuo en diferentes condiciones. Finalmente se exportaron las etiquetas en formato YOLO, un formato ampliamente utilizado debido a su eficiencia y precisión.

Esta metodología de recopilación y etiquetado de imágenes constituyó el primer paso hacia la obtención de una red neuronal convolucional entrenada de manera óptima.

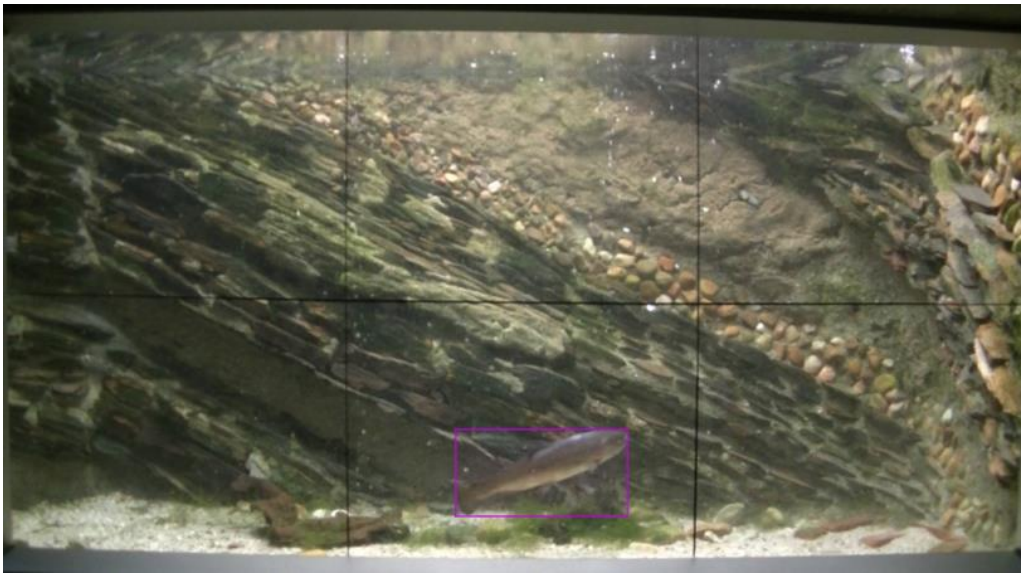

Figura 20. Imagen etiquetada con CVAT

4.2 Entrenamiento del primer modelo de red neuronal

Una vez recopiladas y etiquetadas este primer paquete de imágenes, se procedió al entrenamiento de la red neuronal con YOLO en Google Colab. Google Colab es una plataforma de cuadernos Jupyter que no requiere configuración y ofrece acceso a recursos de computación, como GPUs y TPUs. Está especialmente diseñada para aprendizaje automático y profundo debido a su capacidad para manejar grandes volúmenes de datos y realizar cálculos intensivos [39].

Para el entrenamiento se generó un archivo YAML que especificaba las direcciones de los directorios de las imágenes a utilizar (Figura 21). Estos fotogramas se dividieron en dos carpetas: "*train*" y "*val*". Las imágenes de la carpeta "*train*" se utilizan para que la CNN aprenda a reconocer características y patrones de las truchas. Durante el entrenamiento, el modelo va autoajustando los parámetros, buscando una minimización del error en la predicción. Por otro lado, las imágenes pertenecientes a la carpeta "val" se utilizan para evaluar el rendimiento del modelo. Esta validación permite medir el rendimiento del modelo en imágenes no vistas previamente [40].

Adicionalmente, el archivo YAML contiene las clases que pueden aparecer en las imágenes, que en nuestro caso es "*trout*".

Para gestionar la gran cantidad de imágenes, etiquetas y carpetas que se iban a emplear, se diseñó un script que permitía mover los archivos automáticamente a sus carpetas correspondientes, asegurando que un subconjunto de imágenes fuera destinado a validación (carpeta val). Este tiempo destinado a la generación del código era necesario para asegurar una claridad y un orden en los datos de entrada de la red neuronal, así como tener la posibilidad de incrementar los datos en el futuro.

```yaml
path: /content/drive/MyDrive/TFG
train: 00_data_video_trout/images/train
val: 00_data_video_trout/images/val

# Classes
names:
  0: trout
```

Figura 21. Fichero YAML con sus parámetros

4.3 Primera prueba del primer modelo de red neuronal

Una vez entrenada la red neuronal, se pasó a realizar pruebas utilizando los vídeos disponibles para estudiar su comportamiento. Tras varias pruebas con distintos vídeos, se observó que en aquellos vídeos donde el agua estaba limpia, el desempeño del modelo era aceptable, aunque no perfecto. Sin embargo, el problema ocurría en los vídeos donde el agua estaba sucia o turbia, debido a que la detección de la trucha era deficiente.

Ante esta mala detección en vídeos con agua sucia, se exploraron diferentes soluciones para mejorar el rendimiento del modelo. Se experimentó con la edición del vídeo, ajustando diferentes niveles de brillo y contraste buscando una mayor visibilidad de la trucha. Además, se probó a visualizar el vídeo en diferentes canales en el espacio RGB, con la esperanza de que en alguno de estos hubiera un cambio notorio en la apreciación de la trucha.

Los cambios realizados no lograron buenos resultados, por lo que estas ideas fueron descartadas. La solución adoptada fue realizar una nueva recopilación de fotogramas extraídos únicamente de vídeos donde el agua no era nítida, en concreto los clips 29 y 30. Esta decisión se basó en la necesidad de mejorar la detección del modelo en circunstancias más complejas. Se extrajeron y etiquetaron de la misma manera que se hizo con las imágenes anteriores, asegurando que hubiera una mayor diversidad de imágenes representativas en agua sucia.

Además, se optó por entrenar el modelo con imágenes con un nivel de zoom mayor, enfocando al pez dentro de esta nueva imagen. Esta estrategia buscaba variar las condiciones del entorno, perspectivas, detalles y el fondo de la imagen, debido a que, hasta ese momento, todas las capturas tenían el mismo fondo.

La extracción de estos fotogramas se realizó de manera distinta, ya no se utilizó la aplicación VLC. Se empleó un enfoque más eficiente para ahorrar tiempo y optimizar el proceso. Se utilizó el propio modelo previamente entrenado para seleccionar un fotograma cada cierto número de fotogramas en los que se detectaba la presencia del objeto. Una vez identificados se procedió a recortar el fotograma alrededor de la trucha, buscando que esta ocupara más de un 30% de la imagen. Para aumentar la variabilidad y la robustez del modelo en cada recorte se aplicaban márgenes distintos.

Este nuevo paquete de imágenes formado por las imágenes con agua sucia y zoom (Figura 22), aportaron un nuevo enfoque que enriqueció el conjunto de datos de entrada de la red neuronal con 627 imágenes nuevas, mejorando su capacidad de precisión y generalización en la detección de nuestro objetivo.

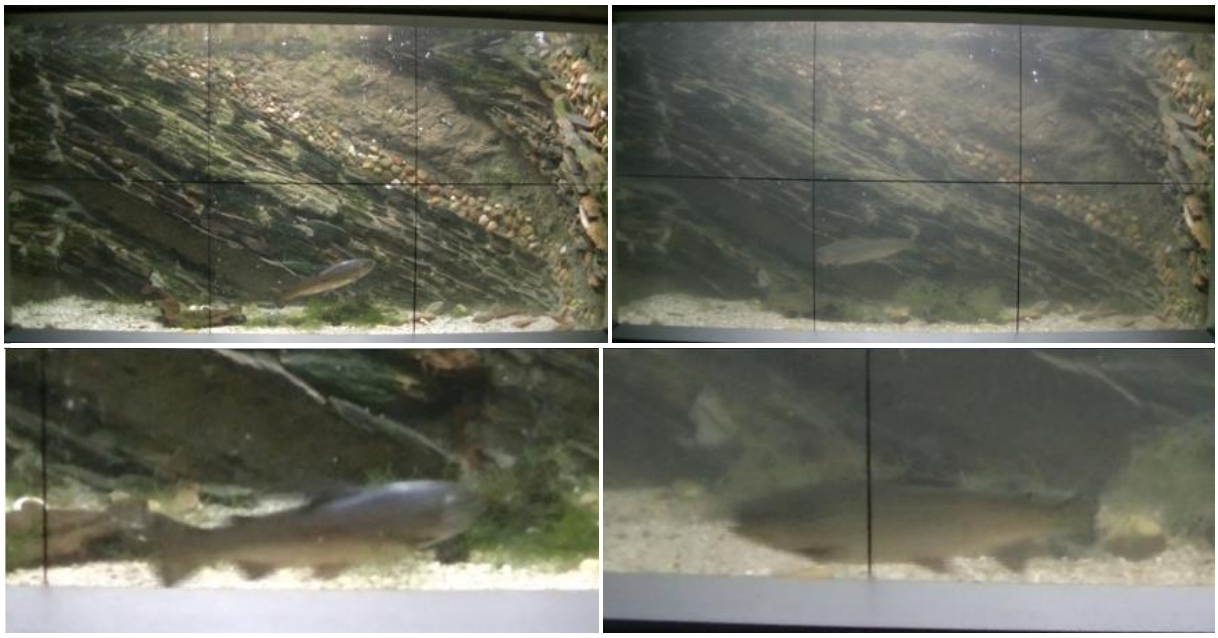

Figura 22. Imágenes con agua nítida y turbia con zoom y sin zoom.

4.4 Data augmentation

Con las medidas tomadas anteriormente, el funcionamiento del modelo era correcto y realizaba las detecciones de manera aceptable en prácticamente todos los vídeos. Sin embargo, para mejorar aún más el rendimiento del modelo, se optó por utilizar técnicas de data augmentation (ver apartado 2.2.3).

Existen diferentes maneras de realizar esta técnica. Durante el entrenamiento de la red, YOLO realiza por defecto un ajuste de aumento e hiperparámetros. Estos valores se pueden variar para conseguir mayores variaciones, pero no siempre proporciona los resultados esperados debido a la falta de control del usuario sobre cada transformación [41].

Otra posible opción es añadir al archivo YAML una configuración de los hiperparámetros para aplicar data Augmentation. Es una opción muy válida, pero, como en la opción anterior, el usuario tiene una falta de control sobre el proceso que está ocurriendo en cada transformación [42].

Para obtener un mayor control y flexibilidad en la transformación de las imágenes, se decidió utilizar la librería de Albumentations. Albumentations es una librería de Python diseñada para realizar data Augmentation de manera eficiente, ofreciendo diferentes funciones para variar los parámetros de una imagen. Con esta herramienta, se diseñó un código con Python en Google Colab que generaba seis imágenes nuevas a partir de cada imagen original. Cada imagen nueva se obtenía variando un parámetro específico: brillo, contraste, filtro gaussiano, ruido, cutout (colocar cuadrados negros en la imagen) y una combinación de todas estas transformaciones (Figura 23) [25].

El principal inconveniente que presentaba esta opción en comparación con las otras era el considerable incremento en la cantidad imágenes y etiquetas que debían ser almacenadas. Sin embargo, esto no supuso un problema significativo porque todas estas imágenes se almacenaron en Google Drive.

Finalmente, se obtuvo un total de 7861 imágenes etiquetadas con 6601 imágenes destinadas al entrenamiento y 1260 imágenes destinadas a validación. Es un conjunto de imágenes suficientemente grande para asegurar un correcto funcionamiento del modelo y es el número total de imágenes final con el que se entrenan las redes neuronales que se hacen uso en este proyecto.

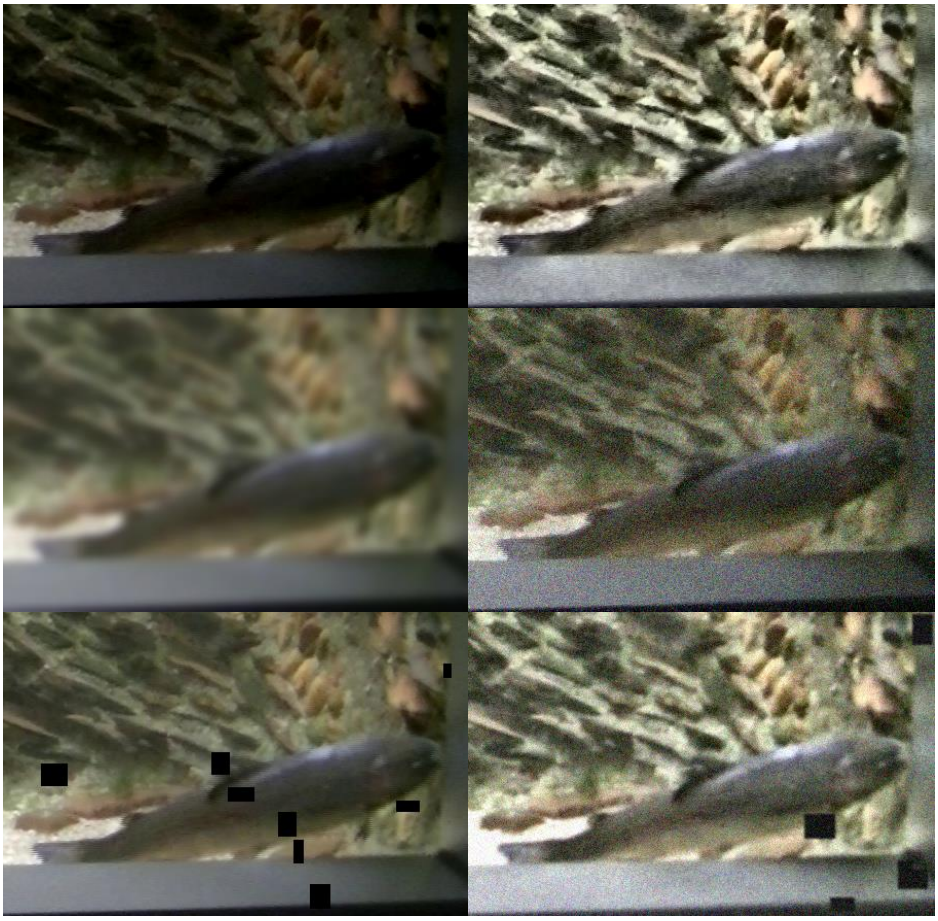

Figura 23. Imágenes con las transformaciones aplicadas (brillo, contraste, filtro gaussiano, ruido, cutout y todas juntas)

4.5 Mejora de la eficiencia con zoom

Inicialmente, el proceso de seguimiento del objeto en vídeo se realizaba con un modelo entrenado con todas las imágenes mencionadas en el punto anterior. Las predicciones realizadas con YOLO tienen multitud de ajustes para encontrar la predicción lo más perfecta posible. El parámetro clave con el que se ha ajustado el modelo para obtener una mejor eficiencia es el “imgsz” que significa “*image size*” (tamaño de imagen). Este parámetro se

puede aplicar tanto en el entrenamiento como en la detección y se establece como un valor entero que determina el redimensionamiento de la imagen. Este primer modelo realizaba detecciones con un imgsz de 640, lo que significa que redimensionaba la imagen a 640x640. Un tamaño de imagen mayor permite una mejor detección del objeto, pero también implica un mayor tiempo de procesamiento.

Con el método convencional, el seguimiento del objeto era relativamente lento, debido a que, dependiendo de la velocidad de procesamiento del ordenador, el tiempo necesario para detectar la trucha en la imagen podía llegar a alcanzar los 300 milisegundos por fotograma, resultando en un tiempo de procesamiento total del vídeo de casi 1 hora. Esta situación planteó un reto significativo en términos de eficiencia.

Para abordar este problema, se desarrolló un algoritmo adaptativo, al que se ha llamado *zoom dinámico*. Esta estrategia consistía en que, una vez detectado el pez en un fotograma, en el siguiente fotograma el modelo solo buscaría cerca de la ubicación anterior del pez. En otras palabras, la detección se haría sobre un fotograma considerablemente más pequeño del fotograma original, centrado en la ubicación previa del pez. Este recorte permitió poder reducir el parámetro de redimensionamiento de la imagen a 160, es decir, una resolución de 160 x 160. Si la detección con este recorte no era exitosa, se volvía a buscar el pez en el fotograma original.

Este nuevo enfoque permitió reducir considerablemente el tiempo de procesamiento del vídeo completo. En un ordenador donde el procesamiento de la imagen con imgsz de 640 tomaba 300 ms, ahora con imgsz de 160 solo tomaba 30 ms, logrando una mejora de velocidad de 10 veces en los fotogramas donde se aplicaba, permitiendo un procesamiento mucho más eficiente y rápido. Aunque estimar con exactitud la reducción del tiempo era complejo, se observó que cuando el pez estaba presente la mayor parte del tiempo, la eficiencia del método aumentaba significantemente. En cambio, cuando el pez no se detectaba, porque por ejemplo estaba escondido, la ventaja en términos de eficiencia se reducía.

Se consideraron otras estrategias, como la no detección intencionada en ciertos fotogramas, que consistía en no realizar la detección durante los siguientes 20 fotogramas si no se había conseguido detectar durante los 5 fotogramas anteriores. Esta estrategia buscaba reducir aún más el tiempo de procesamiento, dado que los momentos más costosos ocurrían cuando el pez estaba escondido y no se podía aplicar el método de recorte. Sin embargo, esta idea no se implementó, porque el objetivo principal era maximizar el número de detecciones exitosas, y aplicar esta estrategia podría resultar en la pérdida de detecciones en algunos fotogramas.

El desarrollo de este algoritmo es extenso y complejo y por ello se ha dedicado un apartado completo de este proyecto exclusivamente a su explicación. Todos los detalles se abordarán en el capítulo 5.

4.6 Interfaz gráfica

Se ha desarrollado una interfaz gráfica de usuario (GUI) para facilitar la visualización de resultados sobre el seguimiento de objetos en vídeo y permitir al usuario realizar distintas funciones. Esta herramienta permite al usuario tratar con los datos necesarios de forma más sencilla e intuitiva, sin necesidad de que este tenga conocimientos sobre programación o inteligencia artificial. Proporciona un acceso sencillo a las funciones del sistema y a los resultados del seguimiento del objeto.

4.6.1 Objetivo de la interfaz gráfica

El principal objetivo de esta interfaz gráfica es simplificar lo máximo posible las acciones a realizar por el usuario, para poder realizar un seguimiento de la trucha de la manera más eficiente y rápida. Mediante esta herramienta, los usuarios pueden realizar las siguientes acciones:

- Cargar y visualizar vídeos de seguimiento del objeto
- Analizar los movimientos de la trucha por la pecera
- Extraer distintos resultados sobre el seguimiento

4.6.2 Tecnologías Utilizadas

Para la creación y desarrollo de la interfaz, se han utilizado varias tecnologías y bibliotecas que permiten realizar aplicaciones de manera robusta, eficiente y sencilla. Estas son:

- Python: Lenguaje de programación
- Tkinter: Librería especializada en el diseño de interfaces gráficas
- OpenCV: Librería utilizada para el procesamiento y visualización de los vídeos resultantes.
- Pandas: Librería para tratar archivos y extraer o cargar datos de él.
- Plotly: Librería para realizar gráficas con los resultados obtenidos

4.6.3 Estructura interfaz

Antes de explicar cada funcionalidad de la aplicación, se expondrán las distintas ventanas que forman la interfaz. Estas ventanas son clases de Python que heredan de tk.Frame que son módulos de Tkinter [43]. De esta manera, las clases se convierten en una especie de plantillas y pueden utilizar todas las funciones y métodos de este módulo, permitiendo la colocación de diferentes widgets en cada una de ellas.

La aplicación está formada básicamente por 3 ventanas principales. La **ventana principal** contiene, principalmente, dos botones que permiten acceder al resto de las ventanas para realizar diferentes funciones (Figura 24).

En **la ventana de reproducción de vídeo** presenta una serie de botones que permiten parar, continuar y cargar el vídeo para un correcto análisis. Estas opciones facilitan la interacción del usuario con el contenido multimedia (Figura 25).

La **ventana de procesamiento de vídeo** es la ventana con mayor número de acciones posibles. Al entrar en esta ventana, se debe cargar el vídeo del que se desea realizar un seguimiento de la trucha. Al realizar esta acción, aparecerá una miniatura del vídeo, y se solicitará al usuario que marque dos puntos que forman las intersecciones entre los seis cuadrantes o zonas. Si este paso no se realiza, el sistema utilizará dos puntos predeterminados donde suelen estar estas intersecciones.

Además, el usuario tiene la opción de proporcionar un archivo Excel con anotaciones realizadas manualmente sobre ese vídeo, para comparar resultados.

Una vez completadas estas acciones, o si se decide omitirlas, se puede proceder a realizar el seguimiento del objeto. Aparecerá una pantalla de carga con un pequeño lienzo donde se podrá observar el vídeo a la velocidad de procesamiento, permitiendo verificar que todo se está llevando a cabo correctamente.

Al finalizar este proceso, se crearán todos los ficheros resultantes con la información necesaria permitiendo repetir nuevamente el proceso de esta ventana (Figura 26).

Cada ventana incluye un botón para retornar a la página principal o navegar a otra ventana. Además, se ha integrado una imagen generada por inteligencia artificial, concretamente con la tecnología DALL-E, en cada ventana [44]. A continuación, se muestran las tres pantallas mencionadas.

Figura 24. Ventana principal de la Interfaz gráfica

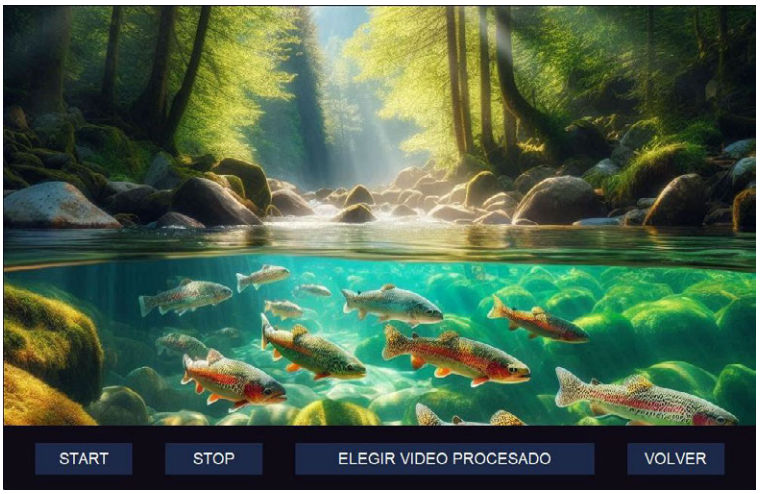

Figura 25. Ventana de reproducción de vídeos de la Interfaz gráfica.

Figura 26. Ventana de procesamiento de vídeos de la Interfaz gráfica.

4.6.4 Resultados obtenidos de la interfaz

Tras realizar el procesamiento del seguimiento del objeto, se generará automáticamente una carpeta, en la que se almacenarán todos los resultados. Esta carpeta contendrá los siguientes ficheros.

4.6.4.1 Mapas de calor

Se generarán dos mapas de calor: uno que considera toda la *bounding box* (Figura 27) y otro que se enfoca únicamente en el centro de la trucha (Figura 28). Para su creación, se utilizará un gradiente de colores, marcando con colores fríos las zonas con presencia nula o muy baja de la trucha, y con colores cálidos, las zonas con mayor presencia.

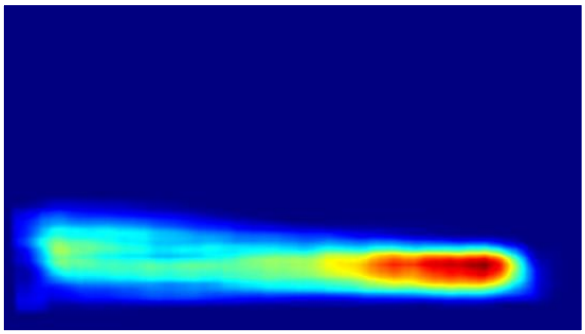

Figura 27. Mapa de calor con bounding box

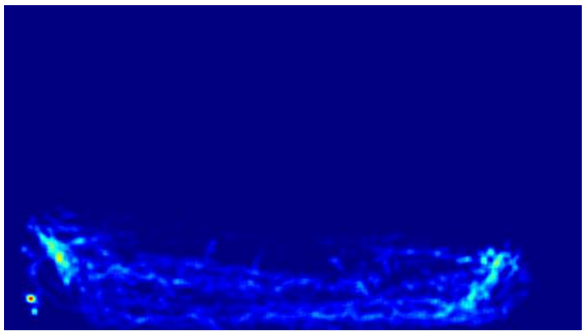

Figura 28. Mapa de calor centro bounding box

4.6.4.2 Gráfica interactiva

Se generará una gráfica interactiva en formato HTML que muestra las zonas en las que ha estado la trucha durante todo el vídeo. En el eje x se representa el fotograma, y en el eje y, la zona en la que se encuentra el individuo. Si se añade una detección manual a partir del archivo Excel mencionado anteriormente en el apartado 4.6.3, la gráfica comparará los resultados obtenidos manualmente con los resultados automáticos, informando de la coincidencia entre ambos. Si no se proporciona este archivo Excel, la gráfica mostrará únicamente los resultados obtenidos automáticamente. Además, a la derecha del gráfico, se incluirán todos los segmentos para posibles análisis (Figura 29).

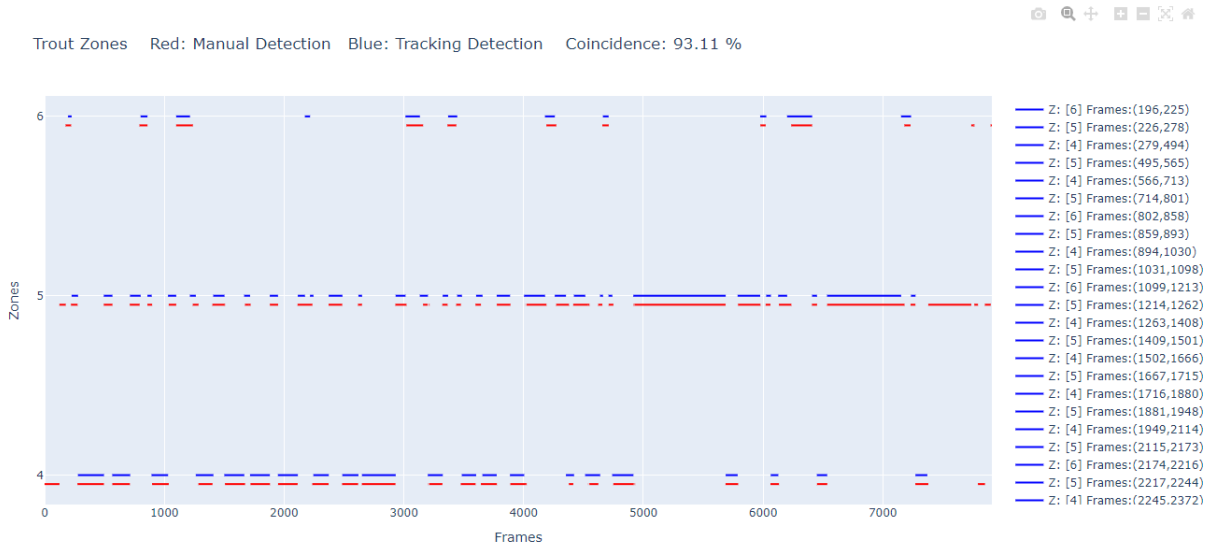

Figura 29. Gráfica HTML ilustrando la zona en la que se encuentra la trucha de manera manual y automática.

4.6.4.3 Vídeo resultante

Se generará un vídeo con anotaciones de interés que incluyen: el fotograma del video, la zona en la que se encuentra la trucha, la *bounding box* indicando su localización, la confianza y la clase sobre la caja delimitadora, y la traza de por donde ha pasado la trucha en los últimos cien fotogramas.

4.6.4.4 Parámetros relevantes

Para evaluar correctamente el modelo y comparar distintos modelos entre sí, se generará un archivo Excel con varios parámetros relevantes. Estos parámetros son: la confianza media de todas las detecciones, el número de detecciones realizadas en el fotograma original, el número de detecciones realizadas en el fotograma recortado, el porcentaje de detecciones en fotogramas recortados frente los originales, el tiempo ejecución, el modelo de red neuronal utilizado para las detecciones en el fotograma original, el modelo de red neuronal para las detecciones en el fotograma recortado y la coincidencia entre los valores obtenidos con este algoritmo con los resultados manuales.

4.7 Arquitectura del sistema

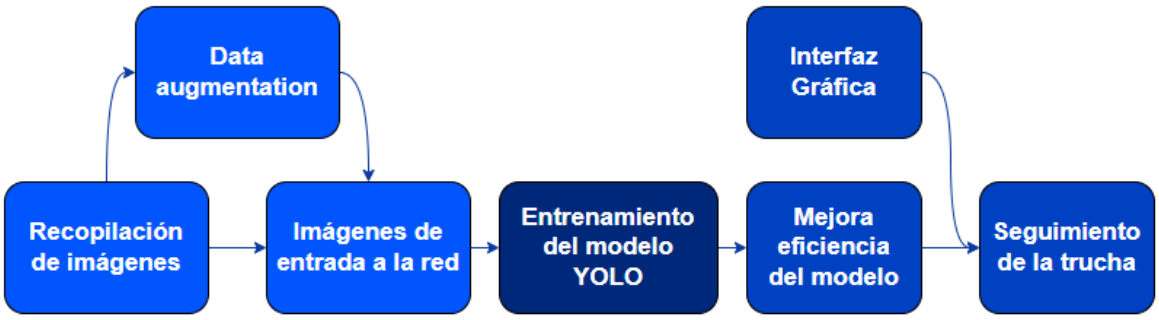

Figura 30. Diagrama de bloques del sistema

Este diagrama de la figura 30, ilustra todo el procedimiento realizado hasta lograr el objetivo principal: seguir la trucha durante el vídeo. En primer lugar, se obtienen las imágenes y se aplica la técnica de data augmentation. Esto nos permite generar un conjunto de imágenes ampliado para entrenar el modelo. Posteriormente, se aplica el método de zoom dinámico para aumentar la eficiencia del seguimiento. Finalmente, con el uso de la interfaz gráfica obtenemos todos los resultados previamente mencionados, incluido el vídeo con el seguimiento detallado de la trucha.

5. Desarrollo del algoritmo y modelos

En el capítulo anterior, se ha descrito el desarrollo del sistema de seguimiento de truchas, proporcionando una visión general de los métodos y tecnologías utilizadas. Sin embargo, no se ha profundizado en detalle en el desarrollo de los modelos utilizados ni la selección de los modelos definitivos. Además, no se ha explicado específicamente qué modelo se ha utilizado para cada parte de la técnica de zoom dinámico, la cual puede requerir de dos modelos: uno para la detección en el fotograma completo y otro para la detección con zoom.

En este capítulo, se profundizará en el desarrollo de los distintos modelos entrenados y su integración en la técnica de zoom dinámico. Se detallarán los conjuntos de datos utilizados, las configuraciones de los modelos y el proceso de entrenamiento.

5.1 Mejora de la eficiencia

Para los tiempos de procesamiento mencionados en el apartado 4.5, se utilizó un ASUS VivoBook 14 X420F equipado con 8 GB de RAM y un procesador i5-8265U CPU @ 1.60GHz 1.80 GHz [45]. En este equipo, el tiempo de procesamiento para la detección en fotogramas sin zoom fue de aproximadamente 300 milisegundos por fotograma, mientras que, para la detección en fotogramas con zoom, el tiempo de procesamiento se redujo a unos 30 ms por fotograma.

Para acelerar la recolección de resultados y realizar un mayor número de pruebas, los vídeos fueron procesados con un ordenador más potente, equipado con 16 GB de RAM y un procesador AMD Ryzen 5 3600 6-Core 3.59 GHz [46].

Este ordenador posee una alta velocidad de procesamiento, lo que permite que, al utilizar la técnica de zoom dinámico, la detección se procese a la misma velocidad que el video original, es decir, 25 fotogramas por segundo. En este dispositivo, los tiempos se reducen a 100 ms y 15 ms (tiempo mínimo, porque corresponde a 25 FPS) respectivamente. Para todos los resultados expuestos a continuación, los vídeos fueron recortados a 2500 fotogramas, permitiendo generar un mayor número de resultados y comparaciones.

Tras aplicar esta estrategia de zoom dinámico, los tiempos de procesamiento de los vídeos se redujeron considerablemente. Pasando de 385 segundos de media a 204 segundos de media, llegando a reducir a la mitad el tiempo de procesamiento en la mayoría de los vídeos. En la figura 31 se puede observar la comparación entre los tiempos de procesamiento. En esta figura, el eje x representa los diferentes clips, mientras que el eje y indica el tiempo de procesamiento correspondiente a cada clip.

Figura 31. Comparación de tiempos de ejecución para método sin eficiencia y con eficiencia.

Para destacar como ha sido detectado el objeto, en el vídeo se utiliza una caja delimitadora de color distinto, junto con sus parámetros, proporcionando una visión clara de como actúa el modelo. En la figura 32, se pueden observar los dos casos: una detección en fotograma normal y una detección en un fotograma con zoom, respectivamente.

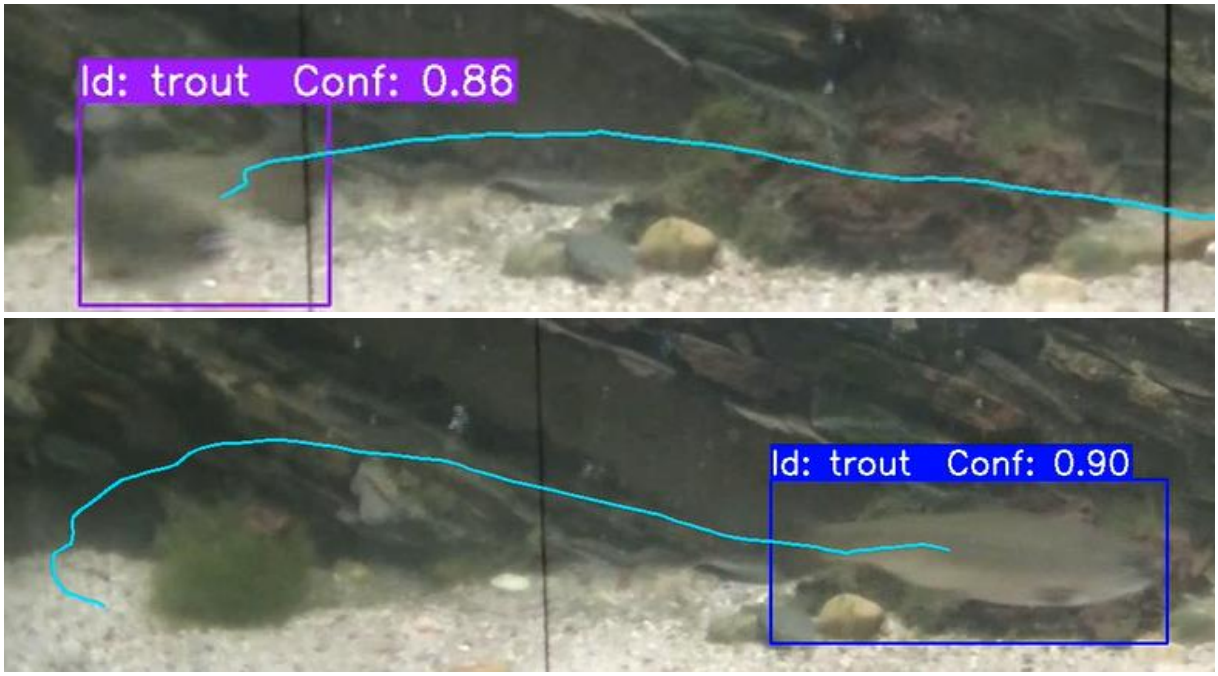

Figura 32. Detección sin zoom (Morado) y detección con zoom (Azul) de fotogramas recortados.

5.2 Redes entrenadas con sus características

Tras optimizar el proceso, aparecen varias cuestiones sobre cómo debería ser entrenada la red. Al utilizar esta estrategia, se puede realizar el seguimiento de la trucha con dos

modelos: uno para detectar en el fotograma entero y otro para detectar en el fotograma con zoom. A partir de esta idea aparecen diferentes modelos entrenados con diferentes conjuntos de imágenes. Los distintos directorios que se disponen para entrenar estos modelos son:

1. Imágenes del fotograma completo tanto con agua nítida o turbia.
2. Imágenes con zoom cerca del objeto con agua nítida.
3. Imágenes con zoom cerca del objeto con agua turbia.
4. Imágenes generadas con data augmentation del primer directorio
5. Imágenes generadas con data augmentation del segundo directorio
6. Imágenes generadas con data augmentation del tercer directorio

En la figura 33 se muestra un mosaico con un ejemplo de cada imagen de estos directorios respectivamente:

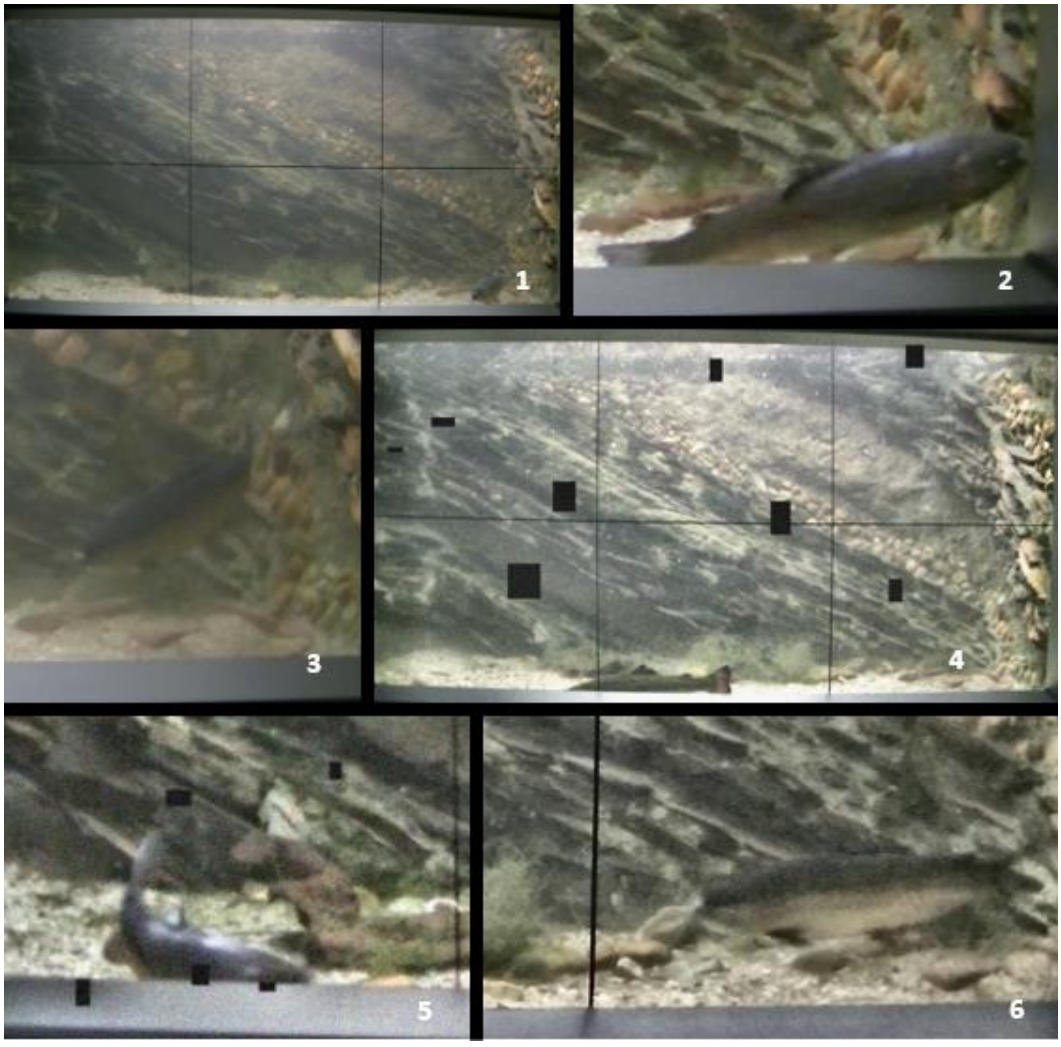

Figura 33. Ejemplos de cada conjunto de imágenes. 1) sin zoom; 2) zoom agua clara; 3) zoom agua turbia; 4) variación de 1; 5) Variación de 2; 6) Variación de 3.

Con todos estos conjuntos de datos, surgieron varios modelos entrenados con un nombre específico asignado para evitar repetir constantemente el conjunto de imágenes que los forman. Estos modelos son:

- Modelo ALL: Entrenado con todos los conjuntos de imágenes
- Modelo OZ (*Only Zoom*): Entrenado solo con los directorios que contienen imágenes con zoom, que serían el segundo, tercero, quinto y sexto.
- Modelo WZ (*Without Zoom Augmentation*): Entrenado con los cuatro primeros directorios, excluyendo los conjuntos de datos formados por imágenes generadas con data augmentation con Zoom.
- Modelo N (*Normal*): Entrenado solo con las imágenes de los 3 primeros directorios, es decir, las imágenes sin aplicar data augmentation
- Modelo OZN (*Only zoom Normal*): Entrenado solo con imágenes con zoom sin data augmentation, es decir, los directorios 2 y 3.
- Modelo S (*Simple*): Entrenado solo con imágenes del primer directorio. Imágenes sin zoom ni data augmentation.

5.2.1 Parámetros relevantes en la calidad del modelo

Para entender adecuadamente los resultados de los modelos, es fundamental entender correctamente los parámetros involucrados.

5.2.1.1 Pérdidas (Losses)

Estas pérdidas ocurren tanto en la etapa de entrenamiento como en la etapa de validación, los parámetros tienen el mismo significado para cada etapa.

- **box_loss (Pérdida Bounding Box):** Representa qué tan bien se ajustan las *bounding boxes* predichas a las verdaderas. La disminución de este parámetro indica una mejora de la precisión de la localización de los objetos.
- **cls_loss (Pérdida clasificación):** Representa la precisión con la que el modelo clasifica los objetos. Con la disminución de este valor se mejora la clasificación de objetos.
- **dfl_loss (Pérdida de desviación focal):** Representa la precisión de las predicciones para los bordes de la *bounding box*. Si disminuye esta pérdida obtenemos una mejor predicción en los bordes predichos.

5.2.1.2 Métricas

Estas métricas proporcionan una evaluación cuantitativa del modelo, permitiendo identificar las áreas de mejora.

- **Precision (Precisión):** Mide la proporción de verdaderos positivos entre todas las predicciones positivas realizadas por el modelo. Una alta precisión indica que el modelo tiene una alta probabilidad de que las detecciones realizadas sean correctas. Puede tomar valores desde el 0 hasta el 1. Su fórmula sería:

$$Precisión = \frac{Verdaderos\ Positivos\ (TP)}{Verdaderos\ Positivos\ (TP) + Falsos\ Positivos\ (FN)}$$

- **Recall (Sensibilidad):** Mide la proporción de verdaderos positivos entre todos los objetos presentes. Una alta sensibilidad significa que el modelo tiene una alta probabilidad de detectar todos los objetos de la imagen. Su rango de valores varía de 0 a 1.

$$Sensibilidad = \frac{Verdaderos\ Positivos\ (TP)}{Verdaderos\ Positivos\ (TP) + Falsos\ Negativos\ (FP)}$$

- **mAP50 (mean Average Precision at IoU = 0.5):** Indica la precisión promedio a la hora de detectar objetos con un valor de IoU del 50%. Los valores típicos suelen rondar entre 0.7 y 0.9
- **mAP50:95 (mean Average Precision at IoU = 0.5:0.95):** Mide la precisión promedio en diferentes umbrales de IoU de 50% hasta el 95%. Sus valores típicos suelen rondar entre 0.5 y 0.75 [47].

5.2.2 Parámetros resultants

A continuación, se presentan los resultados obtenidos con todos los conjuntos de datos para los modelos ALL, OZ, WZ, N, OZN y S. Los parámetros evaluados incluyen pérdidas de entrenamiento y validación, así como métricas de precisión, sensibilidad y mAP.

	ALL	OZ	WZ	N	OZN	S
Train box loss	0,536	0,416	0,447	0,775	0,516	0,702
Train cls loss	0,354	0,302	0,299	0,520	0,384	0,393
Train dfl loss	0,893	0,890	0,842	1,169	1,051	0,964
Precision:	0,958	0,892	0,967	0,956	0,919	0,987
Recall:	0,962	0,951	0,982	0,963	0,962	0,989
mAP0.5	0,977	0,961	0,992	0,985	0,982	0,993
mAP0.5:0.95	0,765	0,805	0,748	0,723	0,839	0,708
Val box loss	0,913	0,825	0,992	1,043	0,722	1,149
Val cls loss	0,425	0,478	0,435	0,535	0,435	0,469
Val dfl loss	1,005	1,052	1,041	1,411	1,180	1,332

Tabla 1. Resultados de los modelos ALL, OZ, WZ, N, OZN y S.

MODELO ALL: Este modelo presenta un rendimiento alto en todas las métricas, beneficiándose de la diversidad y tamaño del conjunto de imágenes. Por lo que es muy prometedor para detecciones de imágenes nuevas en diferentes entornos y situaciones.

Modelo OZ: Este modelo tiene peor precisión y sensibilidad comparado con los modelos ALL y WZ, pero muestra un buen mAP0.5:0.95. Esto implica que el modelo puede detectar objetos pequeños con mayor precisión, pero la gran falta de diversidad reduce la precisión de sus detecciones

Modelo WZ: Al excluir el conjunto de imágenes de data augmentation con zoom, el modelo presenta un valor mAP0.5:0.95 menor que otros modelos, pero en general, es el que presenta mejores resultados.

Modelo N: Este modelo tiene un rendimiento equilibrado, pero la falta de imágenes generadas con data augmentation afecta a su capacidad en comparación con otros modelos que si lo tienen implementado. Esto se detecta por sus valores de entrenamiento y validación, siendo insuficientes.

Modelo OZN: Este modelo es parecido al modelo OZ, pero con un conjunto de imágenes menor.

Modelo S: Muestra una alta precisión y sensibilidad en mAP0.5, pero menor en mAP0.5:095. La falta de diversidad de imágenes limita la capacidad del modelo.

Los tres últimos modelos, N, OZN y S fueron los primeros descartados debido a sus resultados insuficientes.

En la figura 34 se presenta el desempeño de los tres modelos: WZ, OZ y ALL con un tamaño de imagen (imgsz) de 640 con gráficas divididas en tres filas.

La primera fila contiene las gráficas sobre las pérdidas durante el entrenamiento (*train/box_loss, train/cls_loss* y *train/dfl_loss*). El eje X representa las épocas de entrenamiento, mientras que el eje Y representa la magnitud de las pérdidas.

La segunda fila presenta las métricas mencionadas en el apartado 5.2.1.2 (*precision(B), recall(B), mAP50(B)* y *mAP50-95(B)*). Esta letra "B" afirma que es una tarea de detección (si fuera segmentación sería una "M"). El eje X representa las épocas del entrenamiento, y el eje Y representa el valor de la métrica correspondiente.

La tercera fila muestra las gráficas con los resultados de validación (*val/box_loss, val/cls_loss* y *val/dfl_loss*). El eje X representa las épocas de validación, y el eje Y representa la magnitud de las pérdidas.

Es importante destacar que, aunque las gráficas de la primera fila y la tercera fila muestran pérdidas similares (*box_loss, cls_loss* y *dfl_loss*), las escalas en el eje Y difieren. Esta

diferencia en las escalas se utiliza intencionadamente para apreciar mejor la disminución de las pérdidas y permitir una visualización más detallada de los resultados.

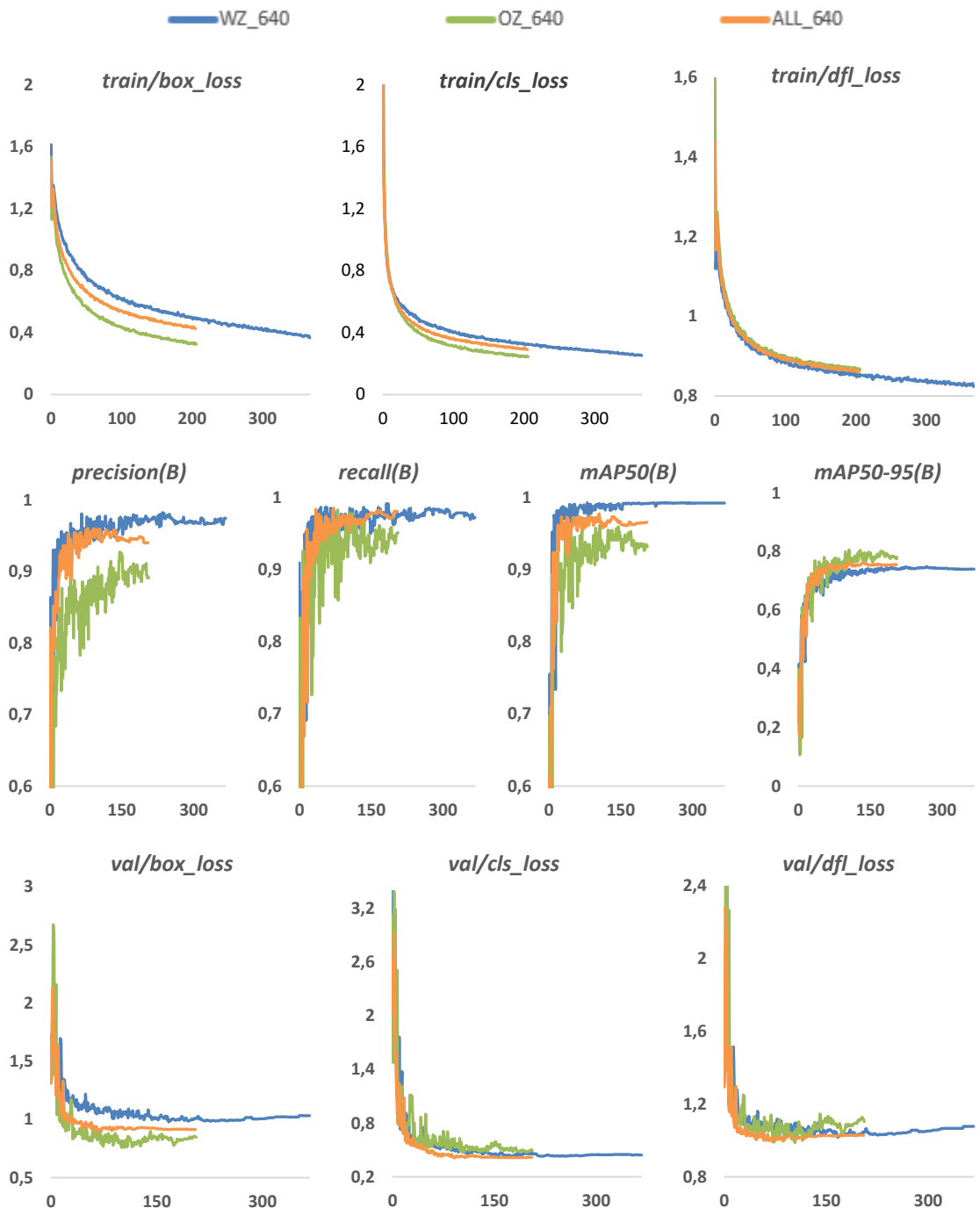

Figura 34. Parámetros de los modelos WZ, OZ y ALL entrenados con imgsz de 640

Estos modelos se pusieron a prueba mediante su utilización en el seguimiento de la trucha. Después de analizar las métricas de precisión, sensibilidad y mAP50(B) y realizar pruebas en vídeos se descartó el modelo OZ. A pesar de que este modelo, formado solo por imágenes con zoom, presentaba pérdidas más bajas, sus valores en las métricas clave era inferiores comparados con los otros modelos. Además, en pruebas prácticas de

seguimiento, el modelo OZ funcionaba de manera insuficiente, detectando el objeto con un valor de IoU bajo o no detectándolo.

En la figura 35 se demuestra el funcionamiento insuficiente del modelo OZ en comparación con el modelo WZ. En la imagen de la derecha, se puede observar que la detección para el modelo OZ tiene una *bounding box* excesivamente grande, lo que delimita de manera poco correcta la localización de la trucha. Esto contrasta con la detección del modelo WZ, que muestra una mayor precisión y una *bounding box* más adecuada para la localización del pez.

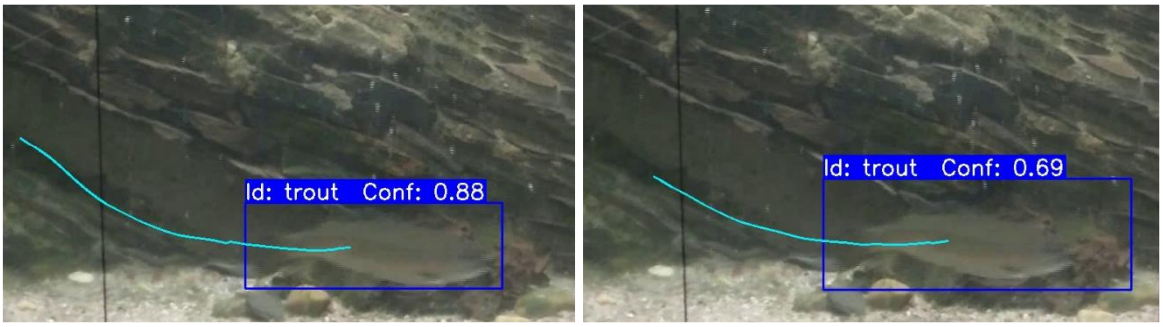

Figura 35. Comparación en la detección entre los modelos WZ y OZ, respectivamente.

5.3 Mejor tamaño de imagen para el entrenamiento

Llegado a este punto, los mejores modelos identificados son ALL y WZ. Estos modelos fueron sometidos a distintas pruebas y entrenados de distintas maneras, variando principalmente el parámetro *imgsz*.

Para las detecciones del fotograma entero se utilizó un redimensionamiento de imagen de 640, debido a que un tamaño más pequeño no permitiría una detección precisa del objeto. Esto se debe a que dicho objeto es pequeño, en proporción al tamaño del fotograma, y reducir demasiado el fotograma dificultaría su detección. Por otro lado, los fotogramas recortados con zoom sobre el objeto serán procesados con un redimensionamiento de imagen de 160, porque en este caso el objeto ocupa un porcentaje considerable del fotograma.

En la figura 36, se pueden observar los resultados obtenidos de entrenar el modelo WZ con tres redimensionamientos distintos: 160, 640 y 1280, permitiendo evaluar su rendimiento en diferentes escalas. Esta gráfica tiene la misma disposición que la Figura 34, con la primera fila mostrando las pérdidas de entrenamiento, la segunda fila presentando las métricas clave del modelo y la tercera fila exhibiendo las pérdidas de validación.

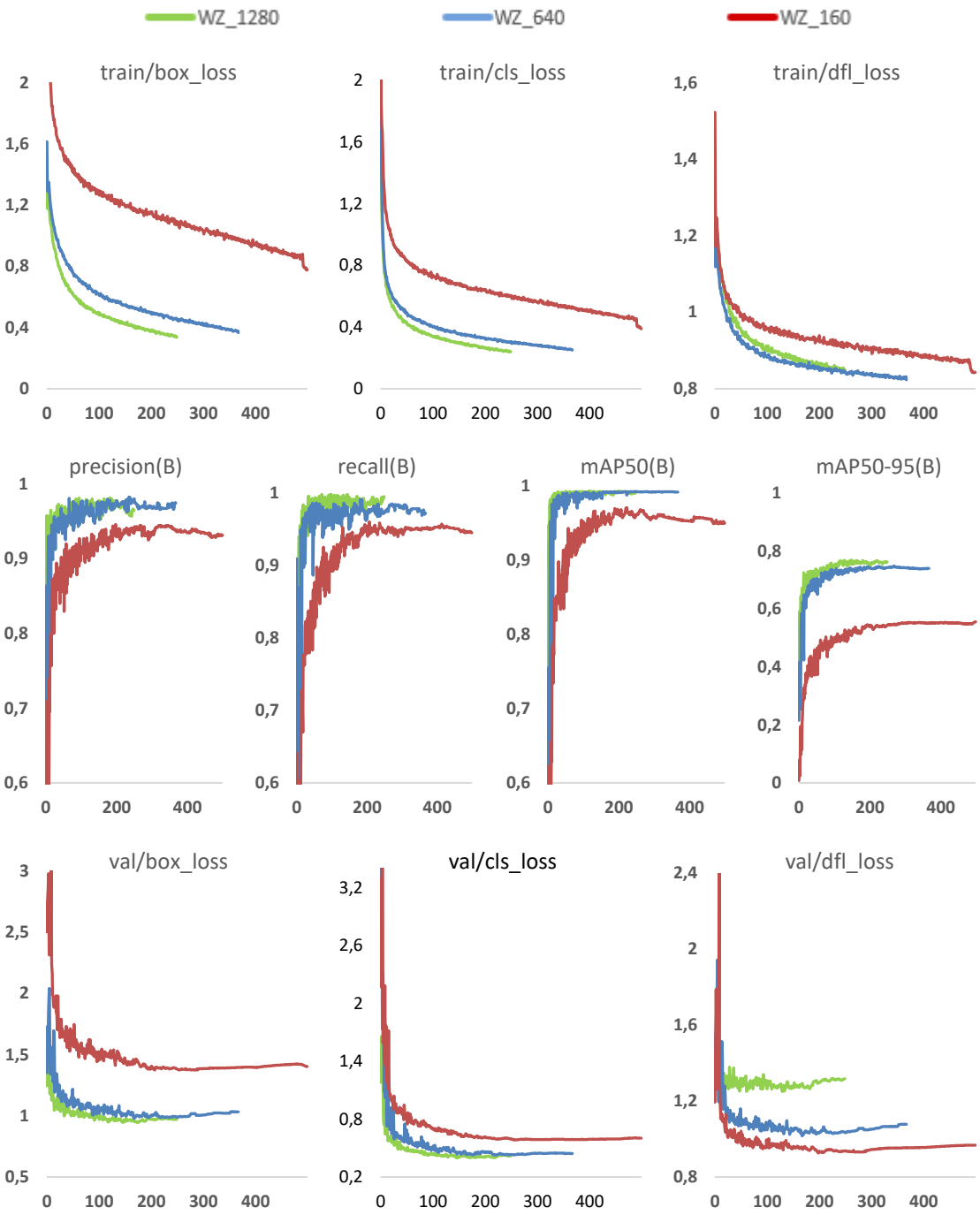

Figura 36. Parámetros del modelo WZ para imgsz de 1280, 640 y 160

Observando la figura 36, se puede afirmar que los valores de rendimiento para los modelos WZ_640 y WZ_1280 son satisfactorios y similares, mientras que WZ_160 presenta resultados insuficientes. Además, estos tres modelos fueron probados con vídeos, y el mejor modelo resultó ser WZ_640. Esto se debe a varias razones:

- 160: Las imágenes a esta resolución pueden ser demasiado pequeñas para poder detectar detalles con precisión, lo que resulta en una pérdida de información importante para la detección de objetos

- 1280: Esta resolución es la que permite detectar los detalles con mayor precisión. Esta precisión tan exacta, conlleva también a detectar el ruido y detalles tan específicos que no son detectados adecuadamente cuando se reducen a imgsz 160 durante la inferencia, perdiendo precisión.
- 640: ofrece un equilibrio entre la detección de detalles y proporcionar suficiente contexto sobre el contorno del objeto. Esto permite realizar detecciones robustas y precisas.

En conclusión, los modelos deben de generalizar bien para ser más efectivos en imágenes nuevas. Por ello, los dos modelos más consistentes han sido los modelos WZ y ALL entrenados con un redimensionamiento de imagen igual a 640.

5.4 Mejores modelos

Estos modelos son bastante precisos y realizan detecciones de manera óptima. Ante esta situación, se procedió a determinar qué modelo se utilizaría para detectar en fotogramas completos y cuál en fotogramas con zoom. Esta tarea se llevó a cabo probando las distintas combinaciones de modelos, que son las siguientes:

- 1:ALL/2:WZ: El modelo ALL detecta en fotogramas completos y el modelo WZ en fotogramas con zoom.
- 1:WZ/2:WZ: El modelo WZ detecta en ambos tipos de fotogramas.
- 1:WZ/2:ALL: El modelo WZ detecta en fotogramas completos y el modelo ALL en fotogramas con zoom.
- 1:ALL/2:ALL: El modelo ALL detecta en ambos tipos de fotogramas

En la figura 37, se puede observar la comparación de estas disposiciones, midiendo el porcentaje de fotogramas realizados con zoom respecto los fotogramas normales.

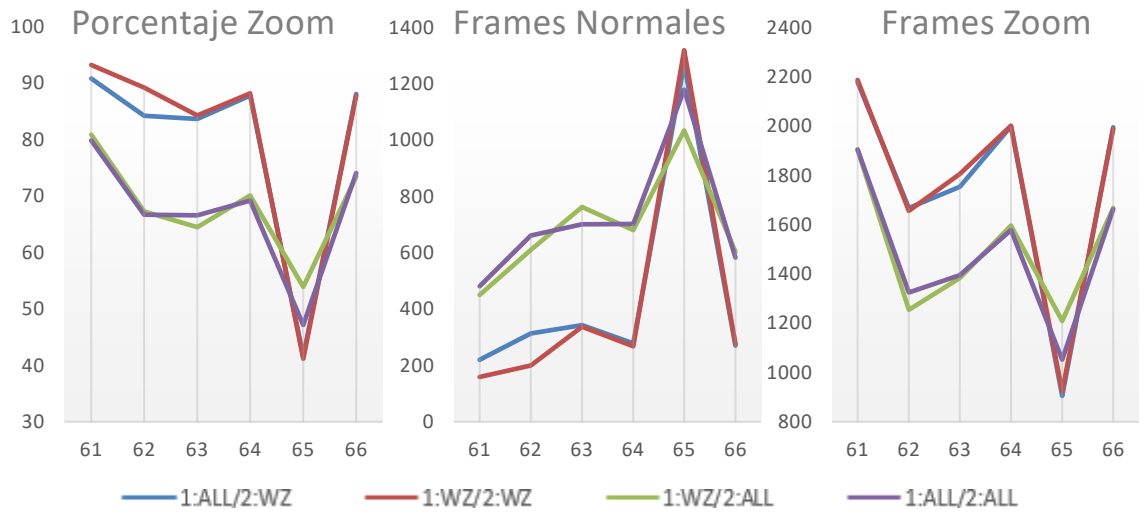

Figura 37. Comparación de fotogramas para las distintas distribuciones.

Las primeras disposiciones demostraron ser superiores a las otras, logrando un porcentaje significativamente más alto de detecciones en los fotogramas recortados con zoom. Después de evaluar varios modelos y configuraciones, se seleccionó la disposición más adecuada mediante pruebas en 22 vídeos distintos, con el objetivo de seleccionar la mejor opción para el proyecto.

Para facilitar la comprensión, se denominará “mixto” a la configuración que usa el modelo ALL para el fotograma entero y el modelo WZ para el fotograma con zoom, y “igual” a la disposición que usa el mismo modelo WZ para ambos casos.

En las dos gráficas de la figura 38, se puede observar que la confianza en las detecciones es ligeramente mayor para la opción “igual” (con una de media 0.843 frente 0,838) y que el tiempo de procesamiento es ligeramente menor con la opción “igual” (con una media de 194,08 ms frente a 198.07 ms).

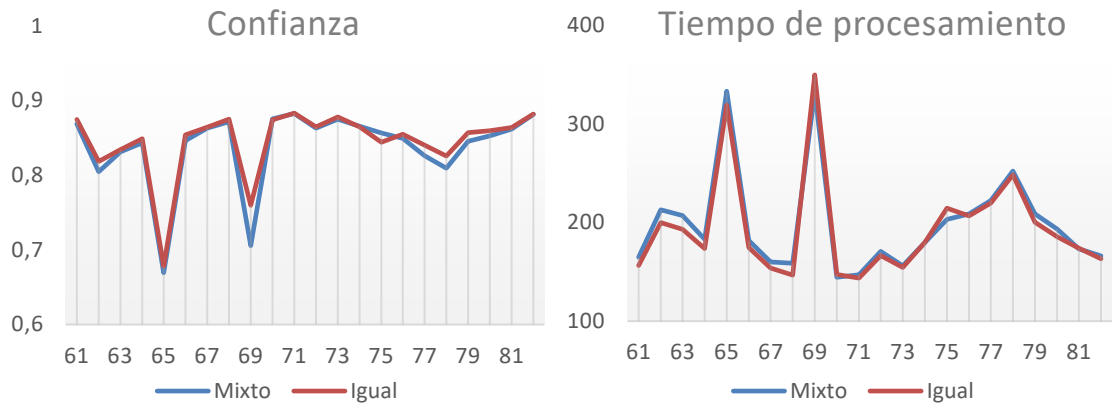

Figura 38. Comparación de la confianza de las detecciones y el tiempo de procesamiento de las mejores disposiciones.

En la figura 39, se compara el porcentaje de fotogramas otenidos con zoom respecto a los normales. Las dos opciones detectan aproximadamente el mismo número de *frames* con zoom, pero la opción “igual” detecta más frames completos (con una media de 342 frente a 325 *frames*).

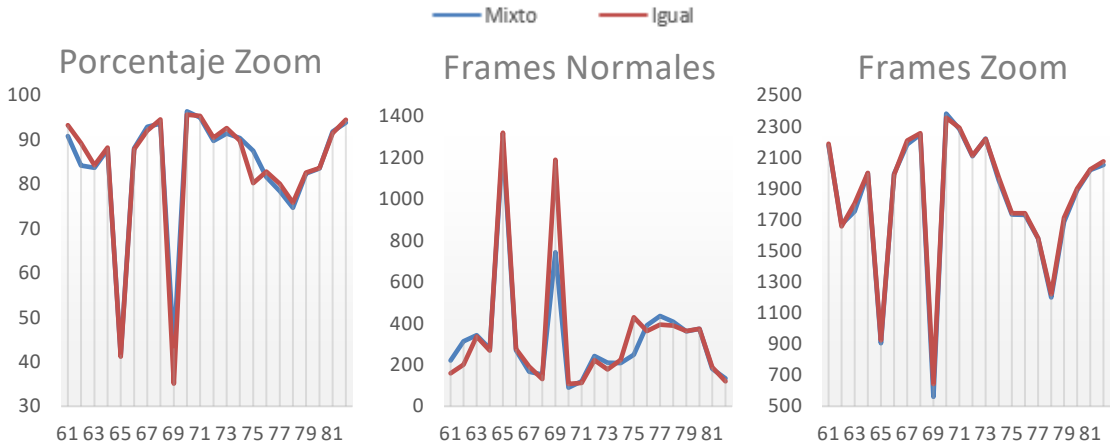

Figura 39. Comparación de fotogramas con y sin zoom para las mejores disposiciones.

Con estos resultados, parece ser que la mejor opción es la de los modelos WZ para ambos casos, es decir, la opción “igual”. Sin embargo, si analizamos la coincidencia entre los valores obtenidos y los valores extraídos manualmente sobre la ubicación donde se encuentra la trucha se verá que la opción “mixto” presenta una mayor coincidencia. Esto se puede observar en la figura 40.

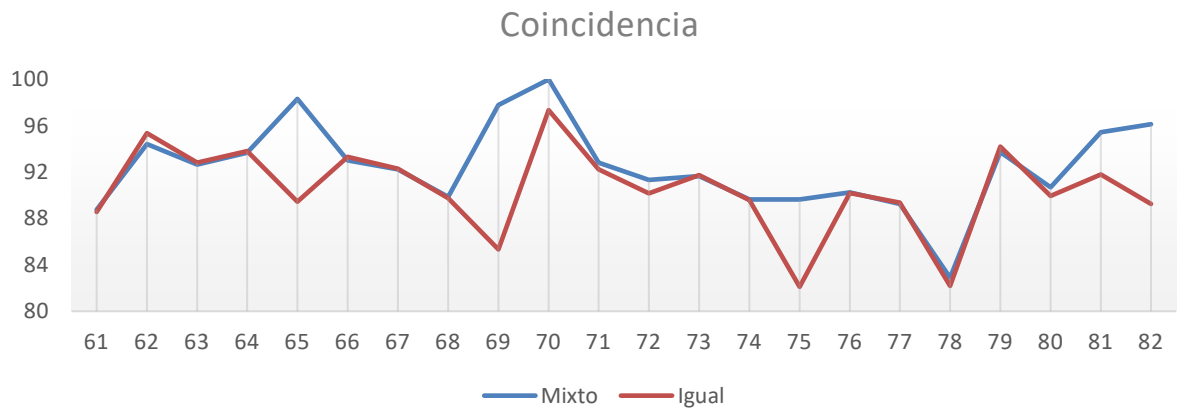

Figura 40. Comparación de las mejores opciones con la coincidencia entre los resultados manuales y automáticos.

Este fenómeno se debe a que el aumento de detecciones en los fotogramas normales no representa una mejora positiva, sino negativa. Muchas de estas detecciones son falsos positivos, el modelo cree haber encontrado el objeto, pero en realidad está realizando una detección errónea. Se llegó a esta conclusión, tras revisar los vídeos en los que había una diferencia bastante significativa, particularmente, en los clips 65 y 69 donde el algoritmo funciona presenta un rendimiento inferior. Gracias a la traza que va dibujando el objeto, se puede observar en la figura 41 como la opción “igual” ha detectado varios falsos positivos, mientras en la opción “mixto” lo detecta correctamente.

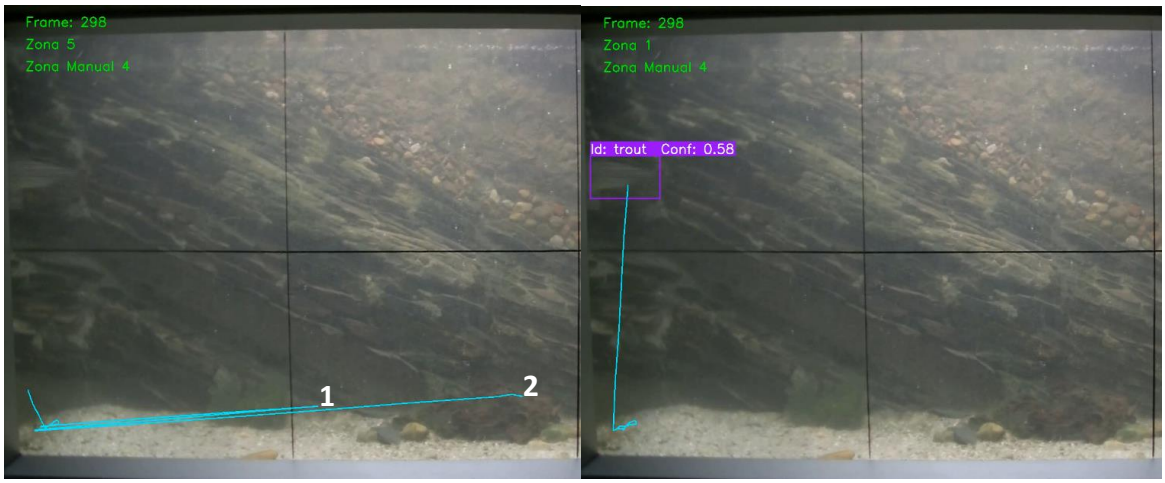

Figura 41. Comparación de falsos positivos en vídeo en Clip 65.

5.5 Resultados finales

Finalmente, queda demostrado, que la mejor opción para el seguimiento de la trucha en el tanque son los modelo WZ y ALL, para detecciones con zoom y detecciones normales, respectivamente. En la figura 42, aparecen todos los parámetros principales de estos dos modelos

	Train box loss	Train cls loss	Train dfl loss	Precision	Recall	mAP0.5	mAP 0.5:0.95	Val box loss	Val cls loss	Val dfl loss
ALL	0,536	0,3542	0,8936	0,9581	0,962	0,9779	0,7652	0,9138	0,4257	1,0056
WZ	0,4479	0,2993	0,8426	0,9673	0,9827	0,9924	0,7487	0,9924	0,4358	1,0417

Figura 42. Parámetros de los modelos ALL y WZ

Estos son resultados excelentes, ya que en ambos modelos la precisión, sensibilidad y mAP0.5 superan el 0.95. Además, los modelos tienen un mAP0.5:0.95 alrededor de 0.75, lo que asegura que las *bounding boxes* delimiten correctamente el objeto.

En el anexo se presentan los resultados de todos los modelos: ALL, WZ, OZ, N, OZN y S.

6. Resultados visuales

6.1 Funcionamiento del sistema

En esta sección se presentarán resultados visuales obtenidos con la implementación final, con el objetivo de ilustrar el funcionamiento del sistema.

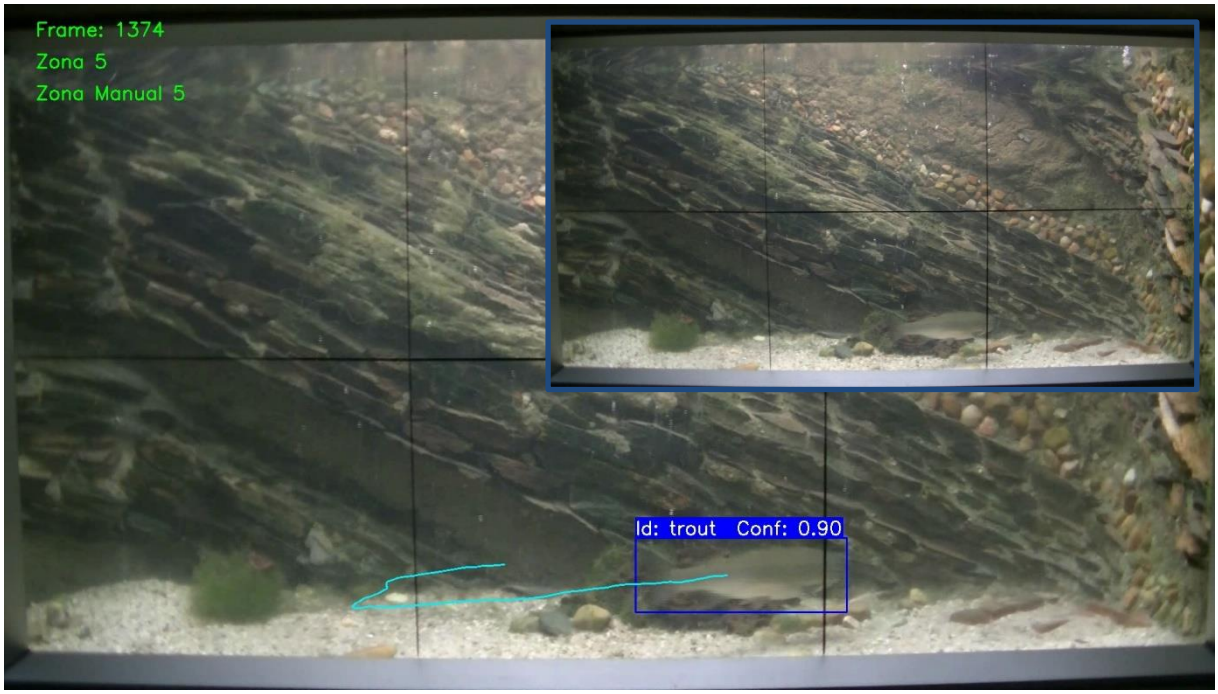

Figura 43. Mismo fotograma con detección del pez con la bounding box y la traza.

En la figura 43, se puede observar cómo es detectada la trucha, a partir de su delimitación con la *bounding box*. Se indica qué objeto es, la confianza de la detección y una traza del centro del objeto en los últimos 100 fotogramas para saber de dónde proviene. Además, en la parte superior izquierda se proporciona información sobre el fotograma actual, la zona donde se encuentra la trucha según el algoritmo y la zona donde se encuentra según la detección manual.

La figura 44 es una imagen editada con el objetivo de representar de forma visual cómo funciona el sistema y cómo realiza el seguimiento de la trucha. Está compuesta por seis imágenes correspondientes a los fotogramas mencionados en el título de la figura.

Figura 44. Explicación visual del funcionamiento del algoritmo con fotogramas 2151,2235,2291,2347,2403,2487.

6.2 Detecciones en condiciones difíciles

En este apartado se presentarán los resultados de detección en situaciones que presentan mayores desafíos para el sistema.

6.2.1 Oclusión

En la figura 45, se puede observar, a partir del fotograma extraído del Clip 62, una detección correcta con solo la parte superior visible.

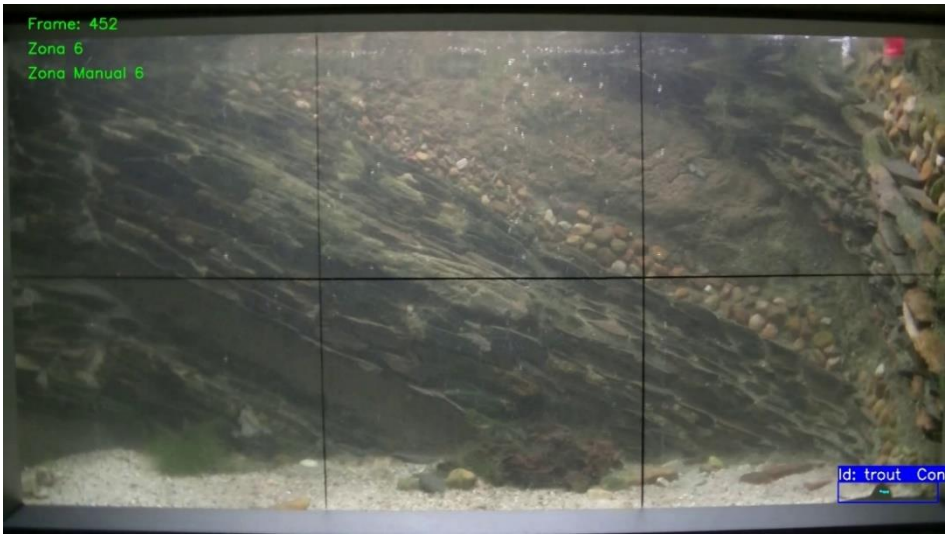

Figura 45. Detección del pez con solo la parte superior visible.

A partir de un fotograma del Clip 62, obtenemos la figura 46, en el que se puede observar como la detección del modelo es correcta aun teniendo solo visible la aleta.

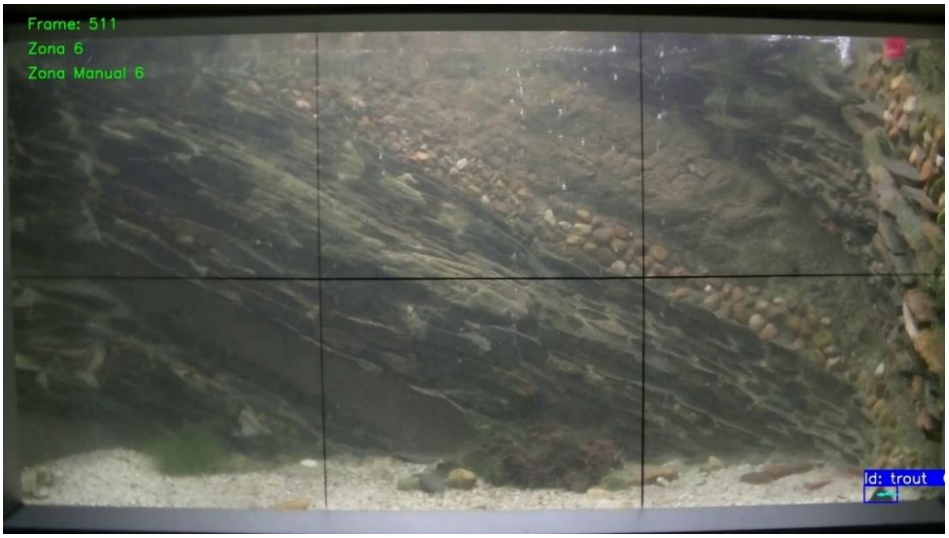

Figura 46. Detección del pez con solo la aleta superior visible.

En la figura 47, a partir de la extracción de un fotograma del Clip 63, se puede apreciar una detección correcta con solo aleta visible.

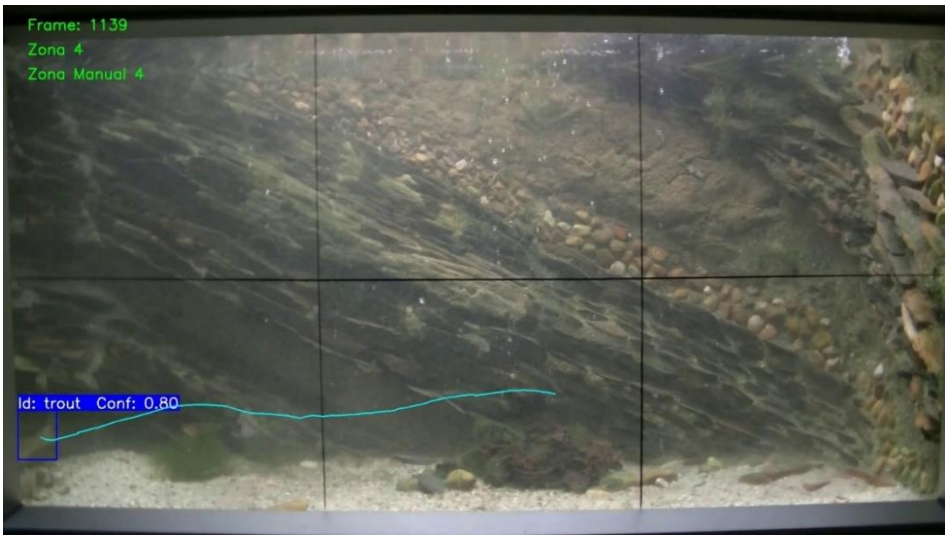

Figura 47. Detección del pez con solo la cola visible.

En la figura 48 se observa una detección correcta del pez con solo la cabeza visible. Este fotograma está extraído del Clip 63.

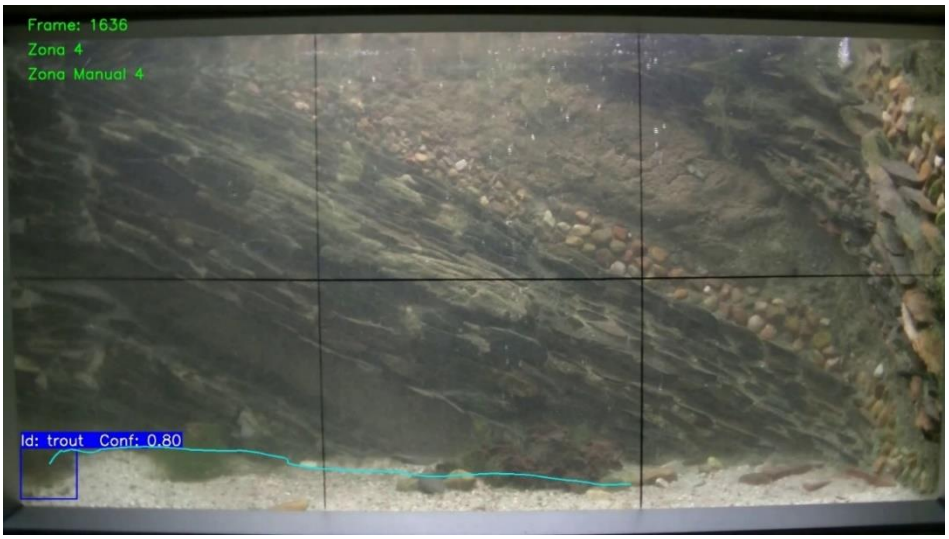

Figura 48. Detección del pez con solo la cabeza visible.

6.2.2 Circunstancias desfavorables

En la figura 49, se presentan dos imágenes del Clip 82 para ilustrar la efectividad del sistema de detección de truchas. La primera es una imagen sin editar y sin detección de objetos, mostrando el estado original del fotograma. La segunda imagen muestra la detección de la trucha delimitándola con la *bounding box*.

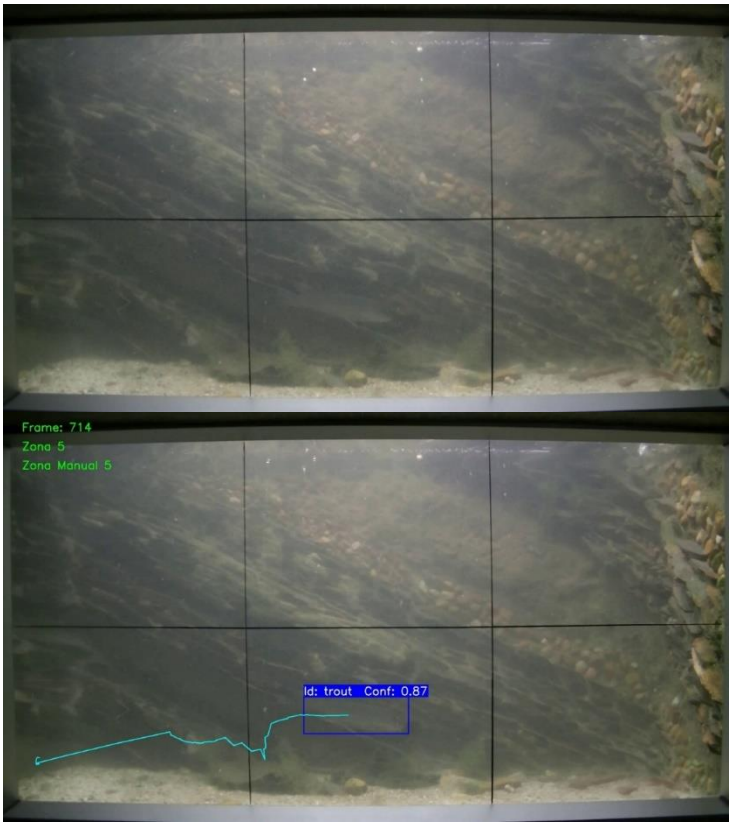

Figura 49. Fotograma antes y después de la detección (1).

En la figura 50, se presentan dos imágenes del Clip 84 con la misma finalidad que la figura anterior.

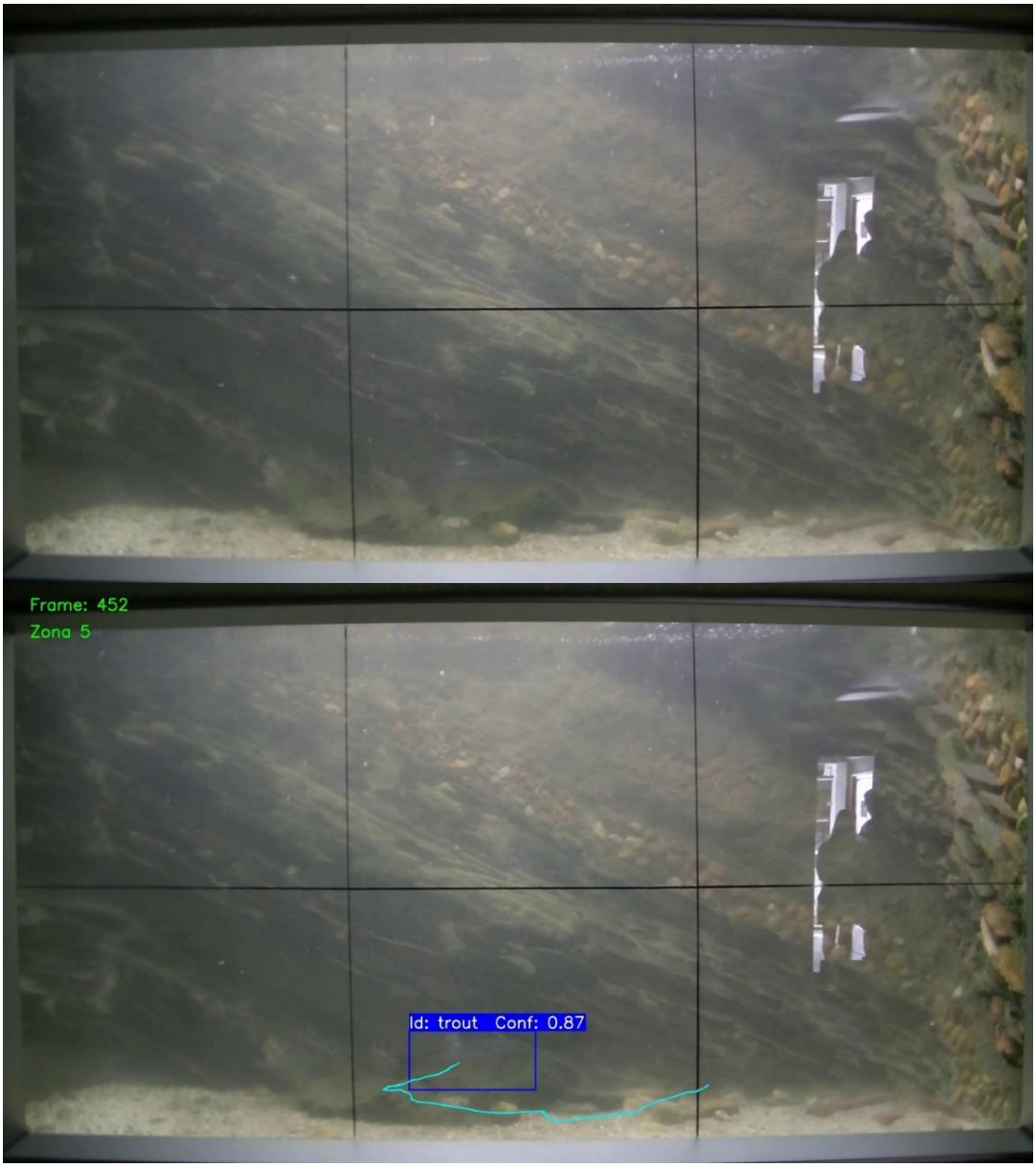

Figura 50. Fotograma antes y después de la detección (2).

En la figura 51, se presentan dos imágenes del Clip 85.

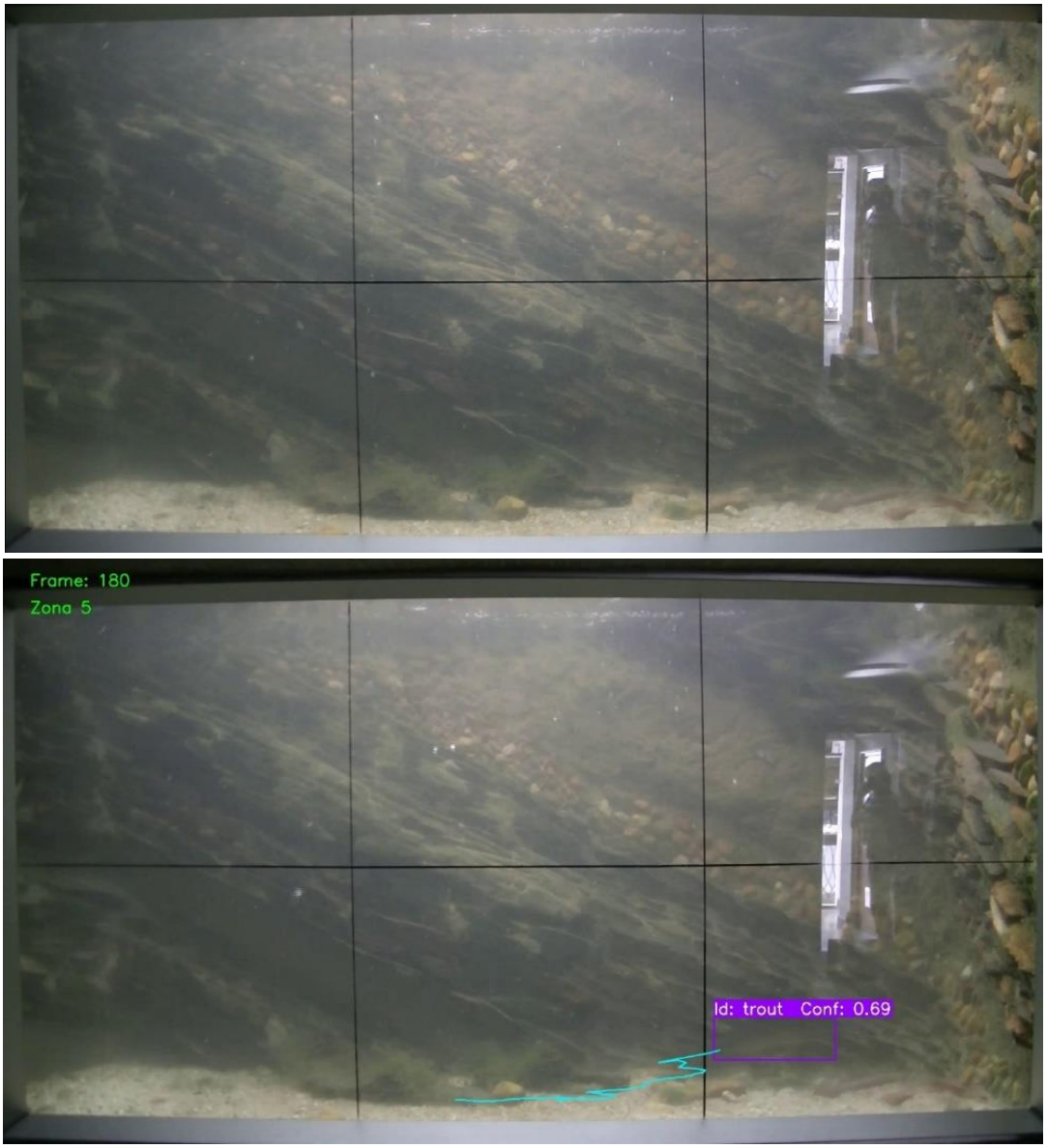

Figura 51. Fotograma antes y después de la detección (3).

En la figura 52, se presentan dos imágenes del Clip 88. Es el caso más extremo que se ha encontrado.

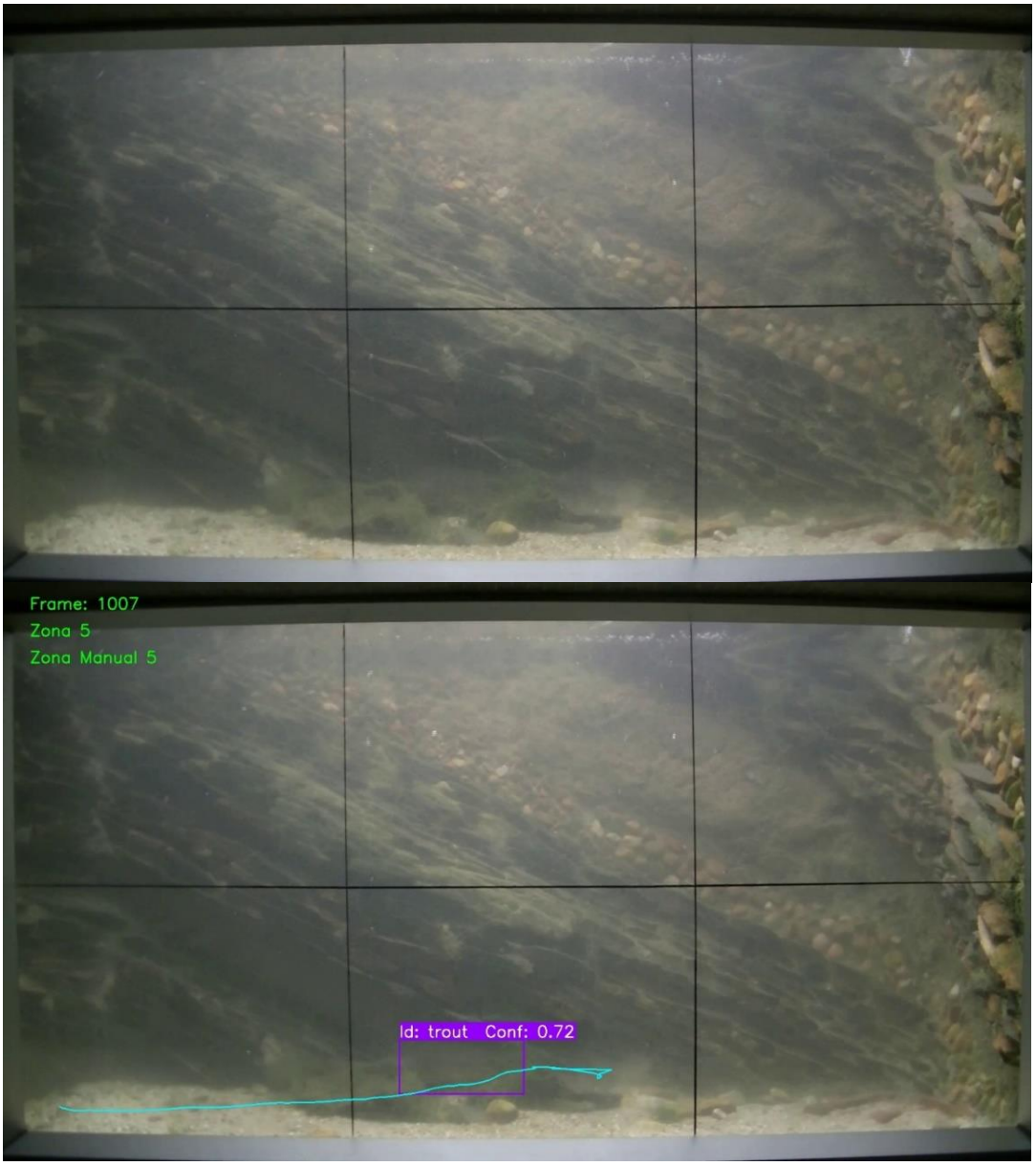

Figura 52. Fotograma antes y después de la detección (4).

6.3 Detecciones fallidas

A lo largo del desarrollo y pruebas del sistema de detección de truchas, se han identificado varios escenarios en los que el modelo no logra realizar detecciones precisas. Estas fallas en la detección son notables en ciertos fotogramas que presentan condiciones particularmente desafiantes.

Uno de los principales problemas se encuentra en los fotogramas donde la nitidez del agua es insuficiente. La turbidez del agua puede causar que el modelo confunda elementos del entorno con la trucha, resultando en falsos positivos o en la completa falta de detección del pez.

Las oclusiones también representan un desafío significativo. En situaciones donde la trucha se esconde detrás de objetos en el tanque, el modelo a menudo falla en mantener un seguimiento preciso.

Los movimientos rápidos del pez también pueden afectar negativamente la precisión del modelo. En fotogramas donde la trucha se mueve a gran velocidad, el desenfoque de movimiento puede hacer que el modelo pierda temporalmente la pista del objeto, resultando en detecciones erróneas.

Finalmente, ciertas posiciones y orientaciones del pez complican la detección. Cuando la trucha se encuentra en una postura inusual o se orienta de una manera que no es común en los datos de entrenamiento, el modelo puede tener dificultades para identificar correctamente la *bounding box* y la posición del pez.

7. Presupuesto

En este capítulo se presentan los presupuestos para cada elemento utilizado en el estudio y desarrollo de este proyecto, abarcando tanto el material como la mano de obra empleada.

Ordenador 1: ASUS VivoBook 14 X420F.

Ordenador 2: Ordenador de mesa con 16 GB de RAM y procesador AMD Ryzen 5 3600 6-Core 3.59 GHz.

Paquete Office: 0€, gracias a la Licencia Estudiante UPM.

Entorno de desarrollo software: (el software utilizado en este proyecto es de libre acceso, de acuerdo con las especificaciones planteadas en el capítulo 3).

- Espacio de trabajo: Visual Studio Code (0,00 €)
- Lenguaje de programación: Python (0,00 €)
- Librerías y bases de datos empleadas (0,00€)
- VLC y CVAT (0,00 €)

Personal de desarrollo e investigación: 14,60 € la hora, sueldo medio de un recién egresado [48]. El presupuesto total es de 6.799,98 €

DESCRIPCIÓN	PRECIO UNITARIO	UNIDADES	PRECIO TOTAL
Ordenador	650,00 €	1	650,00
Ordenador 2	870,00 €	1	870,00
Paquete Office	0,00 €	1	0,00€
Entorno de desarrollo software	0,00 €	1	0,00€
Google Colab / mensual	11,99 €	2	23,98 €
Personal de investigación / hora	14,60 €	100 horas	1.460€
Personal de desarrollo / hora	14,60 €	260 horas	3.796 €
TOTAL PRESUPUESTO			6.799,98

Tabla 2. Presupuesto total.

8. Impacto del proyecto

El presente proyecto sobre el seguimiento de truchas en un tanque mediante el uso de redes neuronales convolucionales tiene diversas implicaciones que abarcan aspectos sociales, de salud y seguridad, ambientales, económicos, tecnológicos e industriales.

El rastreo y análisis de los movimientos de las truchas en cautividad puede ayudar a mejorar la vida de estos animales. Entender mejor sus comportamientos y necesidades son los objetivos principales necesarios para poder diseñar entornos que mejoren su confort y salud, reduciendo así el estrés del animal y mejorar su calidad de vida. Esto no es solo un objetivo ético, sino que también ayuda a la creciente demanda social que pide unas prácticas más responsables y sostenibles en el mundo de la acuicultura.

El monitoreo del comportamiento de estos animales ayuda a prevenir enfermedades mediante la detección de movimientos anómalos. Permitiendo una intervención temprana y rápida, reduciendo el uso de fármacos y antibióticos necesarios para su recuperación. En consecuencia, se mejora la seguridad de los alimentos, al garantizar un cultivo de pescado seguro y saludable para el consumo humano.

La implementación de estas tecnologías puede aumentar la eficiencia en la producción acuícola, aumentando, así, la rentabilidad del producto. Al mejorar la salud y bienestar del animal, se mejora la calidad del producto, lo que se traduce en mejores precios de mercado. Además, con la prevención de enfermedades, se reducen las pérdidas económicas asociadas al tratamiento de peces enfermos o, en el peor de los casos, su muerte.

Este proyecto promueve la utilización de tecnologías avanzadas en el mundo acuícola, fomentando el desarrollo tecnológico. El uso de redes neuronales y técnicas de visión por computadora en la industria acuícola representa un avance tecnológico, demostrando viabilidad y rentabilidad de incorporar estas técnicas en el sector.

Con todo ello, se puede afirmar que este proyecto contribuye a varios Objetivos de Desarrollo Sostenible (ODS) marcados por la agenda 2030:

- ODS 2: Hambre cero: al mejorar la calidad y producción del pescado, se contribuye a la reducir la escasez de alimentos y mejorar la seguridad alimentaria.
- ODS 3: Salud y bienestar: se mejora la salud y bienestar de los peces y como consecuencia, la salud y bienestar de los consumidores humanos.
- ODS 8: Trabajo decente y crecimiento económico: estas tecnologías pueden producir nuevos puestos de trabajo en cuanto la implementación de este sistema en diferentes escenarios.

- ODS 9: Industria, innovación e infraestructura: fomentar la innovación tecnológica en la acuicultura promueve la creación constante de infraestructuras más óptimas para la vida marina.
- ODS 12: Producción y consumo responsables: se promueve la producción de manera más sostenible y eficiente.
- ODS 14: Vida submarina: es el principal objetivo. Mejorar las practicas acuícolas contribuyen a la conservación de los océanos y sus recursos [49], [50].

No obstante, el uso de redes neuronales convolucionales implica un uso masivo de ordenadores, lo que conlleva a un gasto energético considerable. Este aumento en el consumo energético podría tener un impacto ambiental negativo si no se gestionan de manera responsable las fuentes de energía empleadas. Por lo tanto, es fundamental considerar estrategias de eficiencia energética y el uso de las energías renovables para reducir estos efectos negativos.

9. Conclusiones y Trabajo Futuro

9.1 Conclusiones

El presente PFG ha logrado desarrollar un sistema para el seguimiento de truchas en una pecera mediante la utilización de técnicas de visión por computadora, redes neuronales convolucionales y una técnica de zoom dinámico. A lo largo de este proyecto, se han abordado desafíos técnicos para mejorar la precisión y eficiencia del seguimiento de peces en entorno controlado.

Una de las aportaciones más destacadas de este proyecto ha sido la implementación de una técnica de zoom dinámico. Esta ha optimizado el procesamiento del seguimiento al centrar la atención del modelo en la región donde se encuentra el pez, detectando peces que no eran detectados anteriormente y reduciendo la cantidad de falsos positivos. Además, esta técnica ha demostrado ser eficaz en la reducción del tamaño de los fotogramas, mejorando la eficiencia del modelo sin comprometer la calidad de las detecciones. Esta idea es especialmente útil para situaciones donde los recursos computacionales son limitados o para ordenadores con mayor potencia alcanzar una detección casi en tiempo real.

El uso de las redes neuronales convolucionales con el algoritmo YOLO ha permitido realizar detecciones de manera precisa y eficiente. Estas redes han demostrado reconocer patrones complejos de manera similar al cerebro humano.

La combinación de dos modelos entrenados con diferente conjunto de imágenes ha conseguido realizar el objeto principal de este proyecto con resultados mejores de los esperados.

Finalmente, este proyecto se compromete con varios Objetivos de Desarrollo Sostenible de las Naciones Unidas, promoviendo prácticas responsables en la producción de estos alimentos.

9.2 Trabajos futuros

A lo largo de este PFG se han conseguido resultados bastante satisfactorios. Sin embargo, el sistema ha mostrado fallos en algunos fotogramas particularmente complicados, donde las condiciones de nitidez, oclusiones, movimientos rápidos o ciertas posiciones del pez afectaban a la precisión del seguimiento. Esto abre diversas oportunidades para futuros trabajos.

9.2.1 Mejora de la robustez del sistema

- **Aumento de la cantidad y diversidad del conjunto de datos**: Una posible mejora es aumentar este conjunto de datos para el entrenamiento. Variando las condiciones de iluminación y nitidez, diferentes estados de oclusión o posiciones del objeto.
- **Implementación de modelos más avanzados**: Los modelos utilizados en este proyecto representan las tecnologías más recientes y avanzadas. Sin embargo, estas tecnologías están sufriendo una mejora constante en el día a día y probablemente, no en mucho tiempo, aparecerán modelos y sistemas más potentes y eficientes

9.2.2 Optimización del algoritmo del zoom dinámico

- **Mejora del zoom dinámico**: Optimizar esta técnica para adaptarse mejor a movimientos rápidos u oclusiones para realizar un procesamiento aún más preciso y rápido.
- **Algoritmos de postprocesamiento**: Desarrollar estos algoritmos que analicen las trayectorias de los peces y puedan rectificar detecciones erróneas. Una posible técnica que estudiar, para observar si pudiese ser útil en este sistema sería el filtro de Kalman, que se utiliza para estimar el estado de un sistema a partir de predicciones y ajustes y se complementa muy bien con sistemas de detecciones en imágenes.

10. Referencias

[1] F. Chen, X. Wang, Y. Zhao, S. Lv, y X. Niu, «Visual object tracking: A survey», *Comput. Vis. Image Underst.*, vol. 222, p. 103508, sep. 2022, doi: 10.1016/j.cviu.2022.103508.

[2] J. Mira, A. E. Delgado, J. González, y F. J. Díez, *Aspectos básicos de la inteligencia artificial*. Pinos Alta, 49 - 28029 Madrid: editorial sanz y torres, S.L., 2003.

[3] J. N. Kok, *artificial intelligence*. EOLSS Publications, 2009.

[4] K. Warwick y Y. H. Shah, «El futuro de la comunicación humano-máquina: el test de Turing».

[5] S. J. Russell, P. Norvig, y E. Davis, *Artificial intelligence: a modern approach*, 3rd ed. en Prentice Hall series in artificial intelligence. Upper Saddle River: Prentice Hall, 2010.

[6] «Diferencias entre IA, Machine Learning y Deep Learning», HardZone. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://hardzone.es/tutoriales/rendimiento/diferencias-ia-deep-machine-learning/

[7] «Figure 1: Artificial intelligence and its subsets», ResearchGate. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.researchgate.net/figure/Artificial-intelligence-and-its-subsets_fig1_370470867

[8] P. R. de los Santos, «Tipos de aprendizaje en Machine Learning: supervisado y no supervisado», Telefónica Tech. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://telefonicatech.com/blog/que-algoritmo-elegir-en-ml-aprendizaje

[9] I. Pérez Borrero y M. E. Gegúndez Arias, *DEEP LEARNING Fundamentos, teoría y aplicación*. Huelva, 2021.

[10] «Artificial Neural Networks and its Applications», GeeksforGeeks. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/

[11] admin@xeridia.com, «Redes Neuronales artificiales | Blog Xeridia», Xeridia. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i

[12] «Algoritmo Perceptrón». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://cienciadedatos.net/documentos/50_algoritmo_perceptron

[13] «Perceptrón», *Wikipedia, la enciclopedia libre*. 14 de noviembre de 2023. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Perceptr%C3%B3n&oldid=155349909

[14] B. AI, «Redes neuronales», Medium. Accedido: 24 de junio de 2024. [En línea]. Disponible en: https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

[15] «Redes Neuronales y Deep Learning. Capítulo 2: La neurona», Future Space S.A. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.futurespace.es/redes-neuronales-y-deep-learning-capitulo-2-la-neurona/

[16] inteligenciartificiallmca, «1.2.- Clasificación de las Redes Neuronales», Inteligencia Artificial. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://inteligenciartificiallmca.wordpress.com/2017/06/10/1-2-clasificacion-de-las-redes-neuronales/

[17] «Image Data Augmentation Approaches: A Comprehensive Survey and Future directions, ar5iv. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://ar5iv.labs.arxiv.org/html/2301.02830

[18] S. Bhandari, «Overfitting and Underfitting», The Correlation. Accedido: 24 de junio de 2024. [En línea]. Disponible en: https://thecorrelation.in/overfitting-and-underfitting/

[19] «¿Qué es el descenso de gradiente? | IBM». Accedido: 24 de junio de 2024. [En línea]. Disponible en: https://www.ibm.com/es-es/topics/gradient-descent

[20] R. R. Abril, «Redes Convolucionales • Un artículo de La Máquina Oráculo», La Máquina Oráculo. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://lamaquinaoraculo.com/deep-learning/redes-neuronales-convolucionales/

[21] «sobre los tensores: su interpretación conceptual», Marxist Philosophy of Science. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://marxistphilosophyofscience.com/2022/03/26/sobre-los-tensores-parte-i-conceptos/

[22] «¿Qué son las redes neuronales convolucionales? | 3 cosas que debe saber». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://es.mathworks.com/discovery/convolutional-neural-network.html

[23] «¿Qué son las redes neuronales convolucionales? | IBM». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.ibm.com/es-es/topics/convolutional-neural-networks

[24] «Figure 2. Illustration of Max Pooling and Average Pooling Figure 2...», ResearchGate. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451

[25] «Albumentations Documentation - Bounding boxes augmentation for object detection». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://albumentations.ai/docs/getting_started/bounding_boxes_augmentation/

[26] G. Alves, «Detecção de Objetos com YOLO - Uma abordagem moderna», IA Expert Academy. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://iaexpert.academy/2020/10/13/deteccao-de-objetos-com-yolo-uma-abordagem-moderna/

[27] «Intersection over Union (IoU): Definition, Calculation, Code». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.v7labs.com/blog/intersection-over-union-guide, https://www.v7labs.com/blog/intersection-over-union-guide

[28] A. Rosebrock, «Intersection over Union (IoU) for object detection», PyImageSearch. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[29] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection». arXiv, 9 de mayo de 2016. Accedido: 23 de junio de 2024. [En línea]. Disponible en: http://arxiv.org/abs/1506.02640

[30] Ultralytics, «YOLOv8». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/es/models/yolov8

[31] A. Bochkovskiy, C.-Y. Wang, y H.-Y. M. Liao, «YOLOv4: Optimal Speed and Accuracy of Object Detection». arXiv, 22 de abril de 2020. Accedido: 23 de junio de 2024. [En línea]. Disponible en: http://arxiv.org/abs/2004.10934

[32] «OpenCV: Introduction». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://docs.opencv.org/4.x/d1/dfb/intro.html
[33] C. Shorten y T. M. Khoshgoftaar, «A survey on Image Data Augmentation for Deep Learning», *J. Big Data*, vol. 6, n.º 1, p. 60, dic. 2019, doi: 10.1186/s40537-019-0197-0.
[34] «Albumentations Documentation - What is image augmentation». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://albumentations.ai/docs/introduction/image_augmentation/
[35] «Photometric data augmentation in projection radiography | by Thijs Kooi | Lunit Team Blog | Medium». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://medium.com/lunit/photometric-data-augmentation-in-projection-radiography-bed3ae9f55c3
[36] «VLC: Sitio oficial - ¡Soluciones multimedia libres para todos los sistemas operativos! - VideoLAN». Accedido: 1 de julio de 2024. [En línea]. Disponible en: https://www.videolan.org/index.es.html
[37] «CVAT». Accedido: 1 de julio de 2024. [En línea]. Disponible en: https://www.cvat.ai/
[38] R. Díaz, « ✅ Introducción a la detección de objetos 🔍 », The Machine Learners. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.themachinelearners.com/introduccion-deteccion-objetos/
[39] «Google Colab». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://research.google.com/colaboratory/intl/es/faq.html
[40] Ultralytics, «Val». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/es/modes/val
[41] Ultralytics, «Tren». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/es/modes/train
[42] Ultralytics, «Hyperparameter Tuning». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/guides/hyperparameter-tuning
[43] «TkDocs - Frame». Accedido: 24 de junio de 2024. [En línea]. Disponible en: https://tkdocs.com/pyref/frame.html
[44] Y. Fernández, «DALL-E 3: qué es, cómo funciona, sus novedades y qué puedes hacer con esta inteligencia artificial», Xataka. Accedido: 25 de junio de 2024. [En línea]. Disponible en: https://www.xataka.com/basics/dall-e-3-que-como-funciona-sus-novedades-que-puedes-hacer-esta-inteligencia-artificial
[45] «ASUS VivoBook 14 X420F», eBay. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.ebay.es/itm/124700123378
[46] «AMD Ryzen 5 3600 3.6GHz BOX | PcComponentes.com». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.pccomponentes.com/amd-ryzen-5-3600-36ghz-box
[47] Ultralytics, «YOLO Métricas de rendimiento». Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/es/guides/yolo-performance-metrics
[48] «Recién graduado salario en España - Comprueba recién graduado salario promedio en Jooble», Jooble. Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://es.jooble.org/salary/reci%C3%A9n-graduado/Espa%C3%B1a
[49] M. J. Gamez, «Objetivos y metas de desarrollo sostenible», Desarrollo Sostenible. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/
[50] F. Aller, «Guía para trabajar la Responsabilidad Social y Ambiental (GRSA)».

11. Anexo

La Tabla 1 presenta todos los parámetros obtenidos de los vídeos Clip 61 al Clip 90, recortados a 2500 fotogramas, es decir, recortados hasta el segundo 100. Los parámetros incluidos son: la confianza media de todas las *bounding boxes* de ese vídeo; el número de *frames* completos y con zoom detectados, junto con el porcentaje de fotogramas con zoom obtenidos respecto al total de fotogramas detectados; y el tiempo de ejecución requerido para procesar cada vídeo.

	Confianza	Frames Completos	Frames Zoom	Porcentaje Zoom	Tiempo ejecución (s)
Clip 61	0,869	220	2178	90,82	164,67
Clip 62	0,805	313	1668	84,20	212,81
Clip 63	0,831	343	1753	83,63	206,93
Clip 64	0,843	278	1999	87,79	182,91
Clip 65	0,669	1269	905	41,62	332,91
Clip 66	0,846	270	1995	88,07	181,60
Clip 67	0,863	167	2183	92,89	160,06
Clip 68	0,871	153	2247	93,62	158,63
Clip 69	0,706	742	561	43,05	331,98
Clip 70	0,876	90	2381	96,35	144,49
Clip 71	0,882	120	2282	95,00	146,93
Clip 72	0,863	242	2108	89,70	170,57
Clip 73	0,875	210	2221	91,36	156,02
Clip 74	0,865	208	1952	90,37	179,82
Clip 75	0,856	248	1732	87,47	202,95
Clip 76	0,849	391	1731	81,57	208,45
Clip 77	0,826	435	1579	78,40	222,30
Clip 78	0,809	407	1200	74,67	252,13
Clip 79	0,845	361	1686	82,36	208,66
Clip 80	0,852	373	1889	83,51	193,24
Clip 81	0,862	181	2018	91,76	173,36
Clip 82	0,881	135	2051	93,82	166,11
Clip 83	0,864	205	1536	88,22	215,39
Clip 84	0,843	323	1753	84,44	202,85
Clip 85	0,824	401	1531	79,24	224,07
Clip 86	0,890	7	2308	99,69	135,92
Clip 87	0,855	108	794	88,02	271,20
Clip 88	0,823	450	1114	71,22	264,06
Clip 89	0,850	288	1817	86,31	194,35
Clip 90	0,833	433	1185	73,23	254,57

Tabla 3. Parámetros principales obtenidos desde el Clip 61 al Clip 90.

La tabla 2, contiene todas las pérdidas en entrenamiento y validación y las métricas de los seis modelos mencionados en el apartado 5.2.

	ALL	OZ	WZ	N	OZN	S
Train box loss	0,536	0,41639	0,44792	0,77542	0,51689	0,7024
Train cls loss	0,35417	0,30225	0,2993	0,52037	0,38456	0,39351
Train dfl loss	0,89358	0,89034	0,84255	1,1699	1,0516	0,96441
Precision:	0,95808	0,89249	0,96734	0,95605	0,9191	0,98795
Recall:	0,96203	0,95157	0,98271	0,96379	0,9629	0,9899
mAP0.5	0,97786	0,96141	0,99237	0,98539	0,98229	0,99301
mAP0.5:0.95	0,76517	0,80581	0,74866	0,72325	0,83922	0,70817
Val box loss	0,91379	0,82591	0,9924	1,0436	0,72286	1,1494
Val cls loss	0,4257	0,4785	0,43577	0,53574	0,43586	0,46994
Val dfl loss	1,0056	1,0529	1,0417	1,4114	1,1806	1,3327

Tabla 4. Pérdidas y métricas de los modelos ALL, OZ, WZ, N, OZN y S.

12. Manual de usuario

En este capítulo, se proporcionará una guía sobre como ejecutar el programa desde el principio. Se abordarán todos los pasos necesarios, desde la configuración inicial hasta la ejecución completa.

12.1 Arranque de la GUI

La carpeta que contiene todos los ficheros se llama PFG_aplicacion. Esta carpeta deberá ser descargada y contiene todos los ficheros necesarios para la ejecución del programa.

Para la instalación de las librerías se recomienda instalar un entorno virtual. Para ello dentro de la carpeta PFG_aplicacion, debemos ejecutar el siguiente comando desde el cmd:

```
python –m venv nombre_del_entorno
```

El nombre más común para los entornos virtuales que se suele utilizar es ".venv". Los siguientes comandos se describirán como si el nombre del entorno virtual es ".venv".

Una vez creado, se debe activar el entorno virtual a través del siguiente comando, desde el mismo directorio anterior (PFG_aplicacion):

```
.\.venv\Scripts\activate
```

Ahora se debería observar a la izquierda de la línea de comando (.venv). Esto significa que está activado correctamente.

Para instalar todas las librerías necesarias se proporciona un documento de texto llamado "requirements.txt". Este contiene el nombre de todas estas librerías, y a partir del siguiente comando, se instalarán todas:

```
pip install -r requirements.txt
```

Una vez realizados todos estos pasos, se puede ejecutar el programa. Para ello, con el entorno virtual activado, ejecutaremos el siguiente comando:

```
python main.py
```

12.2 Funciones de la GUI

La interfaz gráfica se compone de 3 ventanas, como se menciona durante la memoria del proyecto.

12.2.1 Ventana Principal

Esta es la ventana que aparece cuando se ejecuta el programa, desde él se pueden tomar dos caminos:

- Pulsando el botón: “VER VÍDEOS PROCESADOS”: Se accede a la ventana de reproducción de vídeo.
- Pulsando el botón: “PROCESAR VÍDEO”: Se accede a la ventana de procesamiento de vídeo.

12.2.2 Ventana de reproducción de vídeo

Contiene cuatro botones. Los botones “START” y “STOP” no realizan ninguna función hasta que no se cargue un vídeo.

Con el botón “ELEGIR VÍDEO PROCESADO” podemos elegir un vídeo para visualizarlo en la interfaz gráfica. Con los botones “START” y “STOP”, podemos reanudar o parar el vídeo para analizar distintos casos.

Con el botón “VOLVER” se volvería a la Ventana Principal.

12.2.3 Ventana de reproducción de vídeo

Contiene tres botones: “ABRIR ARCHIVO”, “PROCESAR VÍDEO” y “VOLVER”. El botón “PROCESAR VÍDEO” queda inutilizado hasta que no se cargue un vídeo.

Para cargar un vídeo, se debe pulsar el botón “ABRIR ARCHIVO”. Se abrirá un explorador de archivos y desde ahí se escogerá el vídeo que se desea procesar.

Una vez elegido el vídeo aparecerá una miniatura del vídeo, y se podrán tomar dos caminos distintos:

- Pulsar el botón “PROCESAR VÍDEO” y comenzaría el seguimiento del objeto
- Añadir más parámetros

Esta segunda opción consiste en añadir más parámetros al seguimiento para obtener distintos resultados. Se pueden realizar dos acciones:

- Elegir un archivo Excel donde hay información relevante de en qué zona ha estado la trucha durante todo el vídeo. Esta información se obtiene manualmente observando el vídeo e ir tomando anotaciones de dónde se encuentra. Con la incorporación de este archivo, posteriormente se podrá realizar una comparación entre estos resultados manuales y los generados automáticamente.
- Ajustar las zonas. El tanque está formado por seis zonas: tres en la zona superior y tres en la zona inferior. Estas zonas ya están creadas de manera automática, pero el ángulo de la cámara desde donde se graba puede variar, por lo que si el usuario quiere puede delimitar las zonas manualmente pinchando en las dos intersecciones que forman las zonas. En la figura 53, el círculo rojo corresponde con la intersección de la izquierda y el círculo azul corresponde con la intersección de la derecha.

Figura 53. Ajustar zonas de la pecera en la interfaz gráfica.

Después de realizar estas acciones, se procede a pulsar el botón “PROCESAR VÍDEO”. Con ello, aparecerá una pantalla de carga, en la que se podrá ir observando cómo se procesa el vídeo.

Una vez acabado el procesamiento del vídeo, aparecerá un botón indicando el vídeo que se ha procesado con éxito. Si se pulsa este botón se reiniciará la pestaña para seguir procesando otros vídeos.

Todos los resultados generados se guardarán en la carpeta “*video results*”. Esta carpeta está contenida en el directorio PFG_aplicacion