PROYECTO FIN DE GRADO

TÍTULO: Automatización del Net Test de truchas con aprendizaje automático

AUTOR: Pablo Martínez Lafarga

TITULACIÓN: Grado en Ingeniería Electrónica de Comunicaciones

TUTOR: Nicolás Saénz Lechón

DEPARTAMENTO: Ingeniería Audiovisual y Comunicaciones

VºBº TUTOR

Miembros del Tribunal Calificador:

PRESIDENTE: Amador Miguel González Crespo

TUTOR: Nicolás Saénz Lechón

SECRETARIO: Jorge de Ponga del Pozo

Fecha de lectura: 22 de Julio de 2024

Calificación:

El Secretario,

Resumen

En el presente proyecto se aborda la automatización del *Net Test*, un método clave en la evaluación del bienestar y comportamiento de la trucha arcoíris (*Oncorhynchus mykiss*) en acuicultura. Esta prueba consiste en extraer al pez del agua y observar sus reacciones de escape, que tradicionalmente se analizan manualmente, un proceso laborioso y propenso a errores. La propuesta de automatización utiliza técnicas de aprendizaje automático y visión por computadora para mejorar la precisión y eficiencia del análisis.

El objetivo principal del proyecto es desarrollar un sistema automatizado que capture, etiquete y analice los movimientos de las truchas durante el *Net Test*, para que más tarde los científicos correspondientes puedan categorizar su comportamiento en proactivo, reactivo o neutral. Para ello, se implementa el modelo YOLOv8 (*You Only Look Once*, versión 8), conocido por su alta eficiencia en la detección de objetos en tiempo real, y se utilizan herramientas como el CVAT (*Computer Vision Annotation Tool*) para el etiquetado de datos.

El proceso de desarrollo del sistema se estructura en varias fases: recopilación y preprocesamiento de datos, desarrollo y entrenamiento del modelo, y validación y evaluación de resultados. Durante estas etapas, se considera la normativa vigente en bienestar animal, garantizando que el procedimiento sea lo menos invasivo y estresante posible para los peces.

Los resultados obtenidos con el sistema automatizado se comparan con los análisis manuales tradicionales, utilizando métricas de precisión, sensibilidad y consistencia para evaluar su eficacia. Los hallazgos demuestran que la automatización no solo reduce significativamente el tiempo de análisis, sino que también mejora la precisión y consistencia en la categorización del comportamiento de las truchas.

En conclusión, la automatización del *Net Test* mediante técnicas de aprendizaje automático representa una solución viable y eficiente para la evaluación del comportamiento de peces en acuicultura. Este proyecto no solo mejora la eficiencia operativa y termina ayudando a mejorar el bienestar animal, sino que también aporta una herramienta valiosa para la investigación en este campo. Además, promueve el desarrollo de competencias avanzadas en visión por computadora y aprendizaje automático, subrayando la importancia de la innovación tecnológica en la ingeniería y la acuicultura sostenible.

Abstract

This project deals with the automation of the Net Test, a key experimental method in the evaluation of the welfare and behavior of rainbow trout (Oncorhynchus mykiss) in aquaculture. This test consists of removing the fish from the water and observing its escape reactions. This escape movements are traditionally analyzed manually, which is a laborious and error-prone process. The automation proposal uses machine learning and computer vision techniques to improve the accuracy and efficiency of the analysis.

The main objective of the project is to develop an automated system that captures, tags and analyzes trout movements during the Net Test, so that later the corresponding scientists can categorize their behavior as proactive, reactive or neutral. For this purpose, the YOLOv8 (You Only Look Once, version 8) model, known for its high efficiency in detecting objects in real time, is implemented, and tools such as CVAT (Computer Vision Annotation Tool) are used for data labeling.

The results obtained with the automated system are compared to traditional manual analyses, using metrics of accuracy, sensitivity and consistency to evaluate its effectiveness. The results demonstrate that automation not only significantly reduces analysis time, but also improves accuracy and consistency in categorizing trout behavior.

In conclusion, automation of the Net Test using machine learning techniques represents a viable and efficient solution for the evaluation of fish behavior in aquaculture. This project not only improves operational efficiency and ultimately helps to improve animal welfare, but also provides a valuable tool for research in this field. In addition, it promotes the development of advanced competencies in computer vision and machine learning, highlighting the importance of technological innovation in engineering and sustainable aquaculture.

Índice de figuras

Lista de acrónimos

AdaGrad: *Adaptative Gradient Algorithm* (Algoritmo de Gradiente Adaptativo)

AI: *Artificial Intelligence* / IA: Inteligencia Artificial

AP: *Average Precision* (Precisión Media) [1]

CEIGRAM: Centro de investigación en Gestión de Riesgos Agrarios y Medioambientales

CITSEM: Centro de Investigación en Tecnologías Software y Sistemas Multimedia

CNN: *Convolutional Neural Networks* (Redes Neuronales Convolucionales)

COCO: *Common Objects in Context*

CVAT: *Computer Vision Annotation Tool* (Herramienta Anotación de Visión por Ordenador)

FFNN: *Feedforward Neural Network* (Redes Neuronales de Avance)

GAN: *Generative Adversarial Networks* (Redes Generativas Adversariales)

GPA: Grupo de Producción Animal

GPU: *Graphics Processing Unit* (Unidad de Procesamiento Gráfico o Tarjeta gráfica)

JSON: *JavaScript Object Notation* (Formato de fichero para almacenar datos)

MAE: *Mean Absolute Error* (Error Medio Absoluto)

mAP: *Mean Average Precision* (Precisión Media) [1]

ML: Machine Learning (Aprendizaje Automático)

MLP: *Multilayer Perceptron* (Perceptores Multicapa)

MSE: *Mean Squared Error* (Error Cuadrático Medio)

PFG: Proyecto Fin de Grado

ReLU: *Rectified Linear Unit* (Función Unidad Rectificada Uniforme)

RNN: Recurrent Neural Networks (Redes Neuronales Recurrentes)

ROC: *Receiver Operating Characteristic* (Característica Operativa del Receptor)

SGD: Stochastic Gradient Descent (Descenso de Gradiente Estocástico)

YOLO: You Only Look Once

[1] En su traducción al español estos dos términos significan lo mismo, sin embargo en inglés existen matices que los diferencian, por eso se emplearán estas palabras en inglés.

Glosario

Debido a la naturaleza del tema tratado, aunque algunos términos y expresiones tienen traducción al español, son ampliamente conocidos en inglés dentro del campo de conocimiento. Por esta razón, es posible que al escribirlos en español podría dificultarse el entendimiento. Es por ello por lo que durante la memoria se empleará terminología en inglés. En esta sección se muestra una posible traducción de estos términos .

Accuracy: Exactitud

Annotation: Anotación, etiqueta

Backpropagation: Retropropagación

Bounding Box: Caja Delimitadora

Clustering: Agrupación de objetos en un mismo grupo

Data Augmentation: Ampliación de Datos

Dataset: Conjunto de Datos

Deep Learning: Aprendizaje Profundo

Epoch: Época, iteración

Forward Pass: Pase Hacia Adelante

Inference: Inferencia

Keypoints: Puntos Clave

Leaky ReLU: Función Unidad Rectificada Uniforme con Pendiente

Pooling: Técnica de reducción de dimensionalidad

Precision: Precisión

Recall: Sensibilidad

Vanishing Gradient: Gradiente que se desvanece

Índice de contenidos

1. Introducción

1.1 Marco y motivación del proyecto

La acuicultura, la cría de organismos acuáticos en condiciones controladas, se ha convertido en un componente esencial en la producción alimentaria global. Proporciona una fuente sostenible de proteínas de alta calidad, contribuyendo significativamente a la seguridad alimentaria y la economía mundial. La evaluación del bienestar de los peces en los sistemas de acuicultura es crucial para garantizar prácticas sostenibles y éticas, así como para optimizar la productividad. En este contexto, herramientas que nos permitan evaluar, monitorear y mejorar las condiciones de cría son esenciales para poder mejorar este medio productivo.

Una de estas herramientas es el denominado *Net Test*, o test de la red. Este experimento es fundamental en el campo de la acuicultura y la investigación de la conducta animal. La prueba es utilizada principalmente para evaluar el estrés y el bienestar en peces, o en nuestro caso concreto, las truchas. Esta prueba consiste en, mediante una red de pesca, sacar del agua al pez al que se va a estudiar y ver cómo reacciona al estímulo de ser expuesto a una condición mortal. Por tanto, durante la realización del *Net Test*, es fundamental poder cuantificar la actividad motora del pez, es decir, evaluar la cantidad, amplitud y velocidad de sus movimientos.

La relevancia del *Net Test* radica en su capacidad para proporcionar datos objetivos sobre la respuesta de los peces a diferentes condiciones ambientales y de manejo. Sin embargo, el proceso tradicional de análisis de esta prueba implica la revisión manual de los vídeos grabados durante las diferentes repeticiones del experimento del *Net Test*. Actualmente, utilizando la aplicación BORIS (*Behavioral Observation Research Interactive Software*)[2], se visualiza uno a uno, cada fotograma de los vídeos, y de manera manual, una persona identifica cada movimiento. Este método es laborioso, sujeto a errores humanos y puede resultar en inconsistencias debido a la subjetividad.

Una de las líneas de investigación que se sigue en el Grupo de Producción Animal (GPA) de la E.T.S. de Ingenieros de Montes, se enfoca en la búsqueda de estandarización y normalización del *Net Test*. Dicha investigación dirigida por Dr. Morris Villarroel Robinson en colaboración con Álvaro de la Llave Propín se desarrolla en numerosas publicaciones y PFGs [1] [2]. Este proyecto se centra en la automatización del análisis de vídeos del *Net Test* mediante la implementación de técnicas de aprendizaje automático, específicamente utilizando modelos de visión por ordenador. La automatización de este proceso no solo aumentará la productividad y reducirá el tiempo necesario para el análisis, sino que también minimizará el error humano y mejorará significativamente la repetibilidad de los resultados.

[2] O. Friard y M. Gamba, "BORIS: a free, versatile open-source event-logging software for video/audio coding and live observations," *Methods in Ecology and Evolution*, vol. 7, no. 11, pp. 1325–1330, 2016. [En línea]. Disponible: https://www.boris.unito.it. [Accedido: 24-Jun-2024].

1.2 Objetivos técnicos y académicos

El objetivo principal de este PFG es automatizar el análisis de vídeos del *Net Test*. La automatización debe permitir una evaluación eficiente y precisa del movimiento realizado por los especímenes, reduciendo la necesidad de intervención manual y mejorando la rapidez del proceso. Este desarrollo se divide en otros objetivos más específicos como:

- Creación de un Conjunto de Datos Propio: Desarrollar un conjunto de datos específico que contenga imágenes y vídeos etiquetados con *keypoints* relevantes. Este conjunto de datos será utilizado para entrenar y validar el modelo de estimación de pose, asegurando que las etiquetas sean precisas y adecuadas para los objetivos del proyecto.
- Configuración, Entrenamiento y Validación del Modelo: Configurar el modelo YOLOv8 especializado en estimación de pose, llevar a cabo el entrenamiento con los datos etiquetados, y validar su desempeño. Esto incluye ajustar hiperparámetros y realizar múltiples iteraciones de entrenamiento para optimizar el modelo.
- Realizar una aplicación que integre el modelo entrenado y que permita la inferencia sobre nuevos vídeos del *Net Test*. Debe ser capaz de generar salidas claras y útiles, mostrando, entre otras cosas, los *keypoints* detectados y el nivel de confianza asociado a cada uno, facilitando así la evaluación automatizada del movimiento de los especímenes.

Cabe destacar que no es objetivo de este PFG replicar el análisis manual que se realiza actualmente en los experimentos del *Net Test*. Dicho análisis y sus resultados están diseñados teniendo en cuenta las limitaciones de personal y tiempo disponibles para cada análisis. Por lo tanto, existe el objetivo transversal de comprender correctamente la finalidad del *Net Test* para ofrecer resultados significativos a los investigadores

Desde el punto de vista académico, este proyecto permitirá al estudiante adquirir competencias en el uso de técnicas avanzadas de visión por computadora, procesamiento de imágenes y desarrollo de modelos de aprendizaje automático. Además, se obtendrá experiencia práctica en la implementación de soluciones tecnológicas para problemas reales en el campo de la ingeniería. Las habilidades desarrolladas incluyen la capacidad para trabajar con grandes volúmenes de datos, la comprensión profunda de los algoritmos de visión por computadora y la habilidad para integrar y validar sistemas complejos.

En cuanto a las competencias académicas y habilidades blandas, el estudiante mejorará su capacidad de análisis crítico y resolución de problemas, desarrollará habilidades de gestión de proyectos y trabajo en equipo, y fortalecerá sus habilidades de comunicación escrita y oral a través de la redacción de informes técnicos y la presentación de resultados. Además, se fomentará el autoaprendizaje y la adaptación a nuevas tecnologías, competencias esenciales en el dinámico campo de la inteligencia artificial y la ingeniería.

1.3 Estructura del resto de la memoria

El documento se estructura de la siguiente manera:

Capítulo 2: Marco tecnológico: Se ofrece una visión general de las tecnologías y herramientas utilizadas en el proyecto. Se detallan las características del modelo YOLO v8 para la detección de objetos y puntos clave, y se describe la aplicación BORIS utilizada en el análisis manual tradicional, proporcionando una base comparativa para los métodos automatizados.

Capítulo 3: Especificaciones y restricciones de diseño: Se describen los requisitos técnicos y las limitaciones que influyen en el diseño del sistema automatizado, incluyendo especificaciones de hardware y software, restricciones del entorno de trabajo y criterios de desempeño. También se abordarán consideraciones éticas y de bienestar animal.

Capítulo 4: Descripción de la solución propuesta: Se explica en detalle el desarrollo del sistema de automatización, abarcando la configuración y el entrenamiento del modelo YOLOv8, el desarrollo del pipeline de procesamiento de vídeo y la integración del sistema completo. Se incluyen diagramas de flujo y bloques para ilustrar los procesos implementados.

Capítulo 5: Resultados: Se presentan los resultados obtenidos del sistema automatizado, comparándolos con los resultados del análisis manual tradicional. Se utilizarán métricas de evaluación específicas, tablas y gráficos para ilustrar la precisión, eficiencia y consistencia del sistema, y se discutirán los hallazgos y posibles discrepancias.

Capítulo 6: Conclusiones: Se resumen las principales conclusiones del proyecto, destacando los logros y aprendizajes obtenidos. Se discutirá la validez del sistema automatizado como solución al problema planteado y se identificarán áreas para futuras investigaciones. Se propondrán posibles mejoras y expansiones del sistema.

Anexo A: Presupuesto: Se ofrece un análisis detallado de los costos asociados al desarrollo e implementación del proyecto, desglosando los gastos en hardware, software, mano de obra y otros recursos necesarios. Se proporciona una estimación del presupuesto total y una versión reducida para contextos con recursos limitados.

Anexo B: Impacto del proyecto: Se discuten las implicaciones sociales, ambientales y económicas del proyecto, evaluando cómo la automatización del *Net Test* puede contribuir a prácticas de acuicultura más sostenibles y eficientes. Además, se analizará la alineación del proyecto con los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas.

2. Marco tecnológico

2.1 Inteligencia artificial

La inteligencia artificial (IA) es un campo de la informática que busca crear sistemas capaces de realizar tareas que normalmente requieren de la intervención de un humano, ya sea debido a la inteligencia , el razonamiento, la percepción, el aprendizaje o la toma de decisiones del individuo.

Desde su categorización como disciplina académica en 1956 [3], la inteligencia artificial ha atravesado numerosos altibajos. Durante las décadas de 1960 y 1970, la IA vivió un auge inicial gracias a la creación de programas como ELIZA y SHRDLU, que demostraban la capacidad de las máquinas para procesar lenguaje natural y comprender instrucciones. Sin embargo, en los años 80, la falta de avances significativos y el sobreoptimismo llevaron a un período conocido como el "invierno de la IA", caracterizado por una disminución en la financiación y el interés. Este ciclo se repitió en la década de 1990. No fue sino hasta 2012 que la IA resurgió con fuerza gracias a los avances en el *deep learning*, impulsados por la disponibilidad de grandes conjuntos de datos y el aumento de la capacidad de procesamiento de las tarjetas gráficas o *GPUs* (por sus siglas en inglés: Unidad de Procesamiento Gráfico). Este renacimiento permitió avances notables en áreas como el reconocimiento de imágenes y el procesamiento del lenguaje natural. Aun así, este resurgimiento fue solo a nivel de nicho, es decir, su impacto estaba mayormente concentrado en aplicaciones específicas de la computación y no había penetrado ampliamente en la vida cotidiana o en una gran variedad de industrias. Hoy en día, en la década de 2020, se ha sucedido que, gracias al minado de datos masivo, al enorme potencial computacional disponible y a su salto al público general y no especializado con modelos como GPT, DALL-E o SORA, la inteligencia artificial ha alcanzado un impacto sin precedentes en diversas áreas de la vida cotidiana y la industria, transformando radicalmente cómo interactuamos con la tecnología y mejorando la eficiencia en numerosos sectores.

Las aplicaciones de la IA son diversas y abarcan numerosos sectores. En la salud, la IA se utiliza para el diagnóstico y la predicción de enfermedades; en la educación, mejora la personalización del aprendizaje; en las finanzas, optimiza la detección de fraudes y la gestión de riesgos. Además, tecnologías como los asistentes virtuales (Siri, Alexa) y los vehículos autónomos son ejemplos prominentes de IA en la vida cotidiana.

2.2 Aprendizaje automático

El aprendizaje automático (*Machine Learning*, ML) es una subdisciplina de la IA que se centra en desarrollar algoritmos que permiten a las máquinas aprender a partir de datos. A diferencia de los sistemas tradicionales programados explícitamente, los algoritmos de ML identifican patrones en los datos y hacen predicciones basadas en esos patrones. Muchas veces, y a causa de ser patrones que para la IA son producidos por causas multifactoriales, estos patrones no son fácilmente traducibles a lenguaje humano, por lo que, en algunos casos, su comportamiento puede resultar impredecible. Los principales enfoques del aprendizaje automático incluyen:

- El aprendizaje supervisado es una técnica de aprendizaje automático en la que un modelo se entrena utilizando un conjunto de datos previamente etiquetados. Esto significa que para cada dato de entrenamiento, el modelo conoce no solo la entrada sino también la salida que debería generar. El objetivo del modelo es aprender a mapear las entradas a las salidas correspondientes de manera precisa, permitiéndole predecir la salida correcta para nuevas entradas no vistas. Este enfoque es comúnmente utilizado en problemas como la clasificación y la regresión.
- El aprendizaje no supervisado, trabaja con datos que no están etiquetados. En este enfoque, el modelo intenta encontrar estructuras o patrones inherentes en los datos sin ninguna guía explícita sobre cuál debería ser la salida. Los métodos de aprendizaje no supervisado son útiles para tareas como la detección de anomalías, la agrupación de objetos similares (*clustering*) y reducción de dimensionalidad. Este tipo de aprendizaje es útil cuando las etiquetas no están disponibles o son extremadamente costosas de obtener.
- En el aprendizaje por refuerzo, un agente aprende a tomar decisiones mediante la interacción con un entorno. El agente realiza acciones y recibe retroalimentación en forma de recompensas o castigos, lo que le permite aprender qué acciones maximizarán la recompensa acumulada a lo largo del tiempo. A diferencia del aprendizaje supervisado, donde el modelo se entrena con un conjunto de datos etiquetados, en el aprendizaje por refuerzo, el agente debe explorar y explotar las acciones para descubrir las mejores estrategias o políticas. Este enfoque es comúnmente usado en el juego, la robótica y la optimización de sistemas complejos.

2.3 Aprendizaje Profundo

El *deep learning*, o aprendizaje profundo, es una subdisciplina del ML que se centra en el uso de redes neuronales artificiales con muchas capas (de ahí el término "profundo") para modelar y entender datos complejos. A diferencia de los enfoques tradicionales de aprendizaje automático que suelen requerir características diseñadas manualmente para la interpretación de datos, el *deep learning* puede automáticamente descubrir representaciones y características relevantes directamente de los datos brutos.

El *deep learning* ha ganado una inmensa popularidad en la última década debido a su capacidad para superar en rendimiento a otros métodos, en una variedad de tareas. Esta capacidad ha sido potenciada por el aumento en la disponibilidad de grandes volúmenes de datos y la mejora significativa en el poder computacional.

2.4 Redes neuronales

Las redes neuronales son la base del *deep learning*, y se trata de un modelo computacional de aprendizaje automático inspirado en la estructura y funcionamiento del cerebro humano aunque de forma simplificada.

2.4.1 Estructura de una red neuronal

La unidad más simple de estas redes es en efecto, una neurona o nodo, las cuales se agrupan en diferentes capas, y se interconectan con las siguientes capas mediante una especie de sinapsis. El número de neuronas que compone una capa puede variar entre las diferentes capas. Una red neuronal típica consta de una capa de entrada, una o más capas ocultas y una capa de salida. La capa de entrada recibe las señales del entorno externo, las capas ocultas procesan la información mediante combinaciones lineales y funciones de activación, y la capa de salida genera la respuesta final del modelo. Las conexiones entre las neuronas se realizan a través de estos pesos sinápticos, que son modificados durante el proceso de aprendizaje para optimizar el rendimiento de la red. La Figura 1 muestra una representación gráfica de una red neuronal artificial con una capa de entrada, múltiples capas ocultas y una capa de salida.

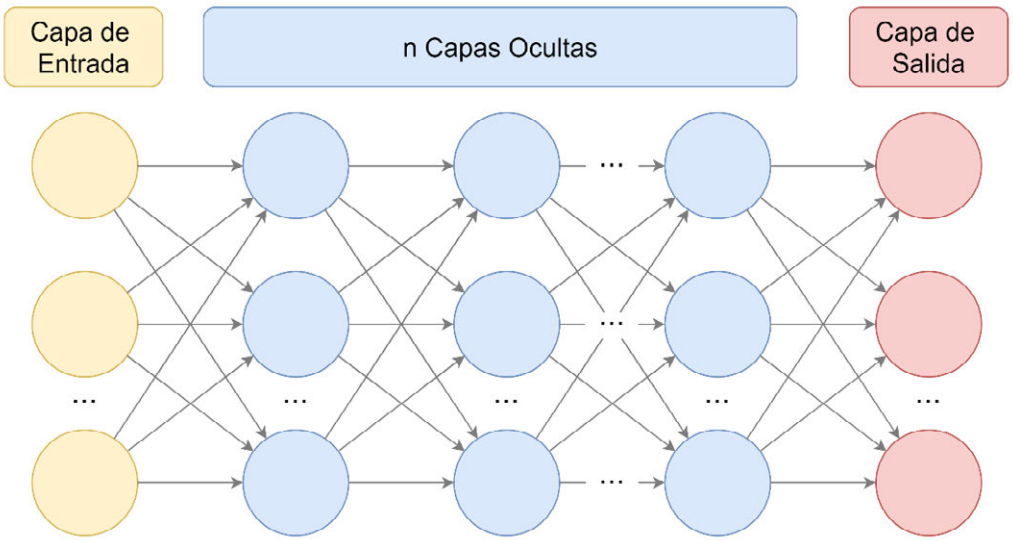

Figura 1: Estructura de una Red Neuronal Artificial Multicapa

2.4.2 Estructura de una neurona

Cada neurona recibe una o más entradas, realiza una operación (habitualmente una suma ponderada) sobre estas entradas, y luego aplica una función de activación para determinar su salida. Matemáticamente, el proceso puede ser representado de la siguiente manera:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$y = \sigma(z)$$

Siendo:

- x_i las entradas de la neurona
- w_i el peso asociado a cada entrada
- b el sesgo asociado
- z suma ponderada de las entradas
- σ la función de activación
- y la salida de la neurona

En la Figura 2 se muestra una representación gráfica del funcionamiento de una neurona en una red neuronal artificial, mostrando el flujo de información desde las entradas, pasando por los pesos, la suma ponderada y la función de activación, hasta llegar a la salida.

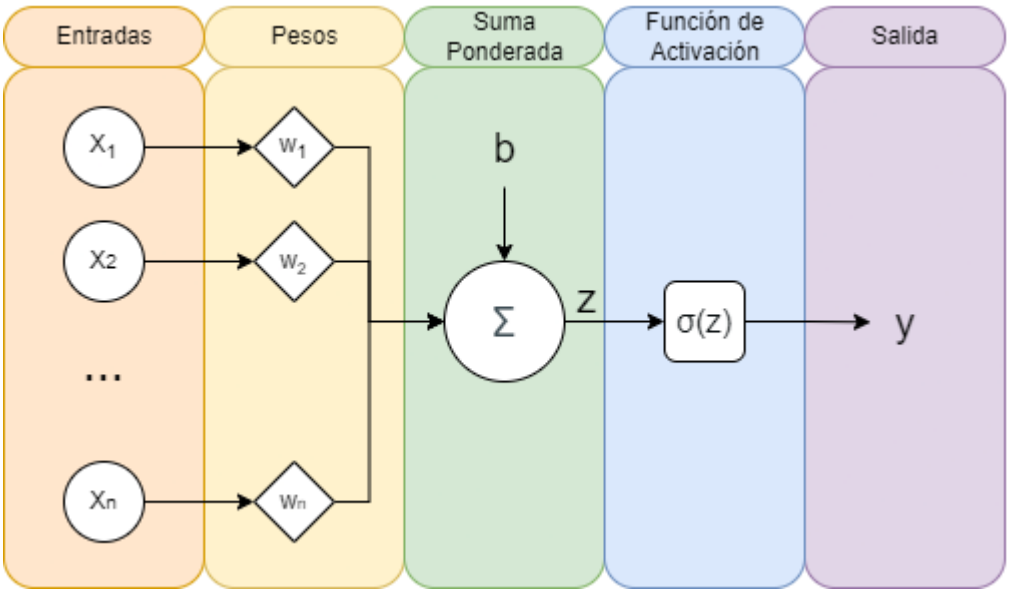

Figura 2: Esquema de Cálculo en una Neurona Artificial

2.4.2.1 Término de sesgo

En el contexto de las redes neuronales, el término de sesgo b es un valor adicional que se suma a la suma ponderada de las entradas de una neurona antes de aplicar la función de activación. Este término es crucial para que el modelo pueda ajustar y desplazar las funciones de activación, lo que le permite a la red neuronal:

- Adaptarse mejor a los datos, ya que sin el sesgo, la salida de una neurona estaría estrictamente relacionada con una combinación lineal de las entradas. El sesgo permite que la neurona produzca una salida distinta de cero incluso cuando todas las entradas son cero, lo que puede ser esencial para ciertas tareas de aprendizaje.
- Al permitir que las funciones de activación se desplacen, el sesgo ayuda a que el modelo pueda aprender relaciones más complejas y por tanto mejorar la flexibilidad.
- Si no existiera el sesgo, una neurona solo podría aprender funciones que pasan por el origen (0,0) en un espacio bidimensional. El sesgo permite que la neurona aprenda funciones que no necesariamente pasan por el origen, por lo que evita la dependencia exclusiva de las entradas.

Para entender mejor la importancia crucial del sesgo se plantea un ejemplo trivial: Supongamos que vamos a predecir el precio de una casa basándonos en el tamaño en metros cuadrados de la misma. Sin el término de sesgo, nuestro modelo puede predecir que una casa de 0 metros cuadrados (lo cual es absurdo) tiene un precio de 0. Sin embargo, en la vida real, incluso la casa más pequeña tendrá un precio mínimo debido a factores como la ubicación, el terreno y otras características.

El término de sesgo permite a nuestro modelo ajustar esta predicción básica. Si introducimos un sesgo positivo en el modelo, éste puede capturar el precio base de una casa, independientemente de su tamaño. Así, incluso si el tamaño es cero, el sesgo asegura que la predicción de precio no sea cero sino un valor que represente este precio base mínimo.

Por ejemplo, si todas las casas en una cierta ciudad tienen un precio base de 50k EUR debido a la ubicación y otros factores, el término de sesgo en nuestro modelo permitirá que incluso para un tamaño de 0 metros cuadrados, la predicción del modelo sea 50k EUR, ajustándose así más a la realidad del mercado inmobiliario.

2.4.2.2 Funciones de activación

Las funciones de activación son un componente esencial en las redes neuronales artificiales. Su principal propósito es introducir no linealidad en el modelo, permitiendo que la red neuronal pueda aprender y modelar comportamientos complejos. Sin la no linealidad proporcionada por las funciones de activación, la red neuronal sería simplemente una combinación lineal de sus entradas, lo que limitaría severamente su capacidad de representar relaciones complejas en los datos.

Existen varias funciones de activación que se utilizan comúnmente en las redes neuronales, cada una con sus propias características y usos específicos:

- La función **sigmoide**: convierte cualquier valor de entrada en un valor entre 0 y 1. Es especialmente útil en las últimas capas de una red neuronal para problemas de clasificación binaria. Sin embargo, su uso en capas ocultas ha disminuido debido a problemas como el *vanishing gradient*. En la Figura 3 se muestra la definición de la función, y su representación gráfica en la cual el eje horizontal es el valor de entrada z y el eje vertical el valor de salida $\sigma(z)$.

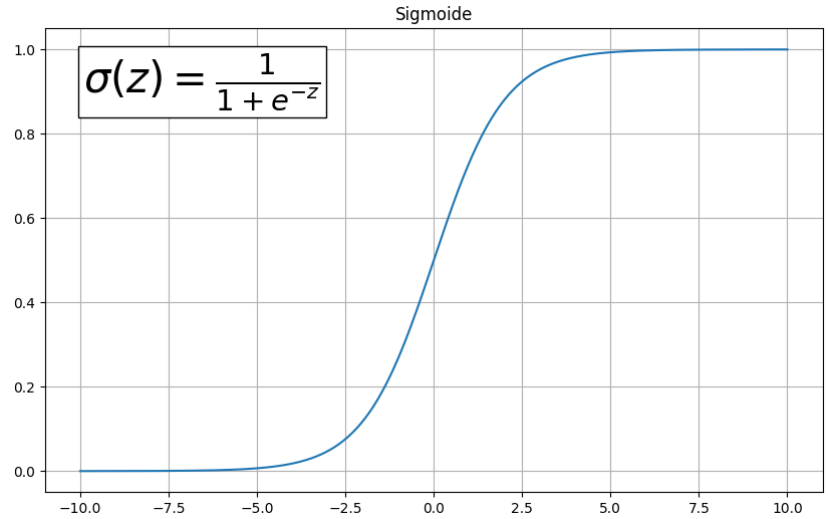

Figura 3: Función de Activación Sigmoide

- La función **tangente hiperbólica** es similar a la sigmoide, pero con sus salidas son normalizadas entre -1 y 1. Esto puede hacer que las redes neuronales entrenen más rápido o que las salidas sean más centrales en torno a cero. Sin embargo, al igual que la sigmoide, también puede sufrir del problema de gradiente que se desvanece. En la Figura 4 se muestra la definición de la función, y su representación gráfica en la cual el eje horizontal es el valor de entrada z y el eje vertical el valor de salida $\sigma(z)$.

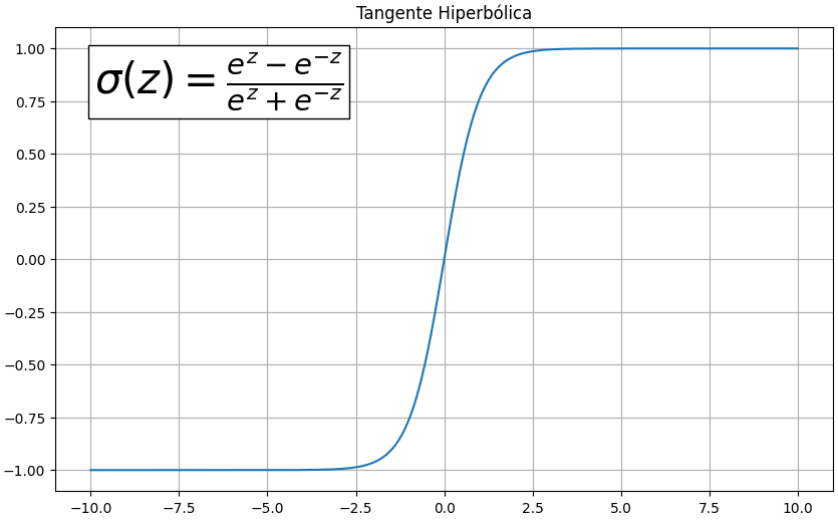

Figura 4: Función de Activación Tangente Hiperbólica

- La función **Unidad Lineal Rectificada (ReLU)** se ha vuelto muy popular en las redes neuronales profundas debido a su simplicidad y eficiencia. ReLU convierte cualquier valor negativo en cero y deja los valores positivos sin cambios. Esto ayuda a mitigar el *vanishing gradient,* además permite que la red entrene más rápidamente. En la Figura 5 se muestra la definición de la función, y su representación gráfica en la cual el eje horizontal es el valor de entrada z y el eje vertical el valor de salida $\sigma(z)$.

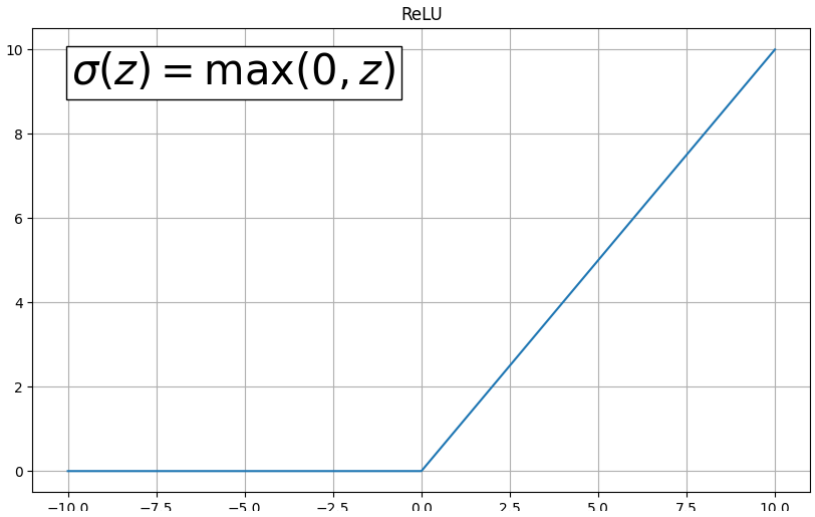

Figura 5: Función de Activación ReLU

- ***Leaky* ReLU** es una variante de ReLU que permite valores negativos pequeños (α es un número positivo muy pequeño). En lugar de tener una pendiente cero para valores negativos, tiene una pendiente pequeña. Esto significa que la función no se detiene completamente en valores negativos, lo que puede ayudar a evitar la saturación de la neurona. En la Figura 6 se muestra la definición de la función, y su representación gráfica en la cual el eje horizontal es el valor de entrada z y el eje vertical el valor de salida $\sigma(z)$.

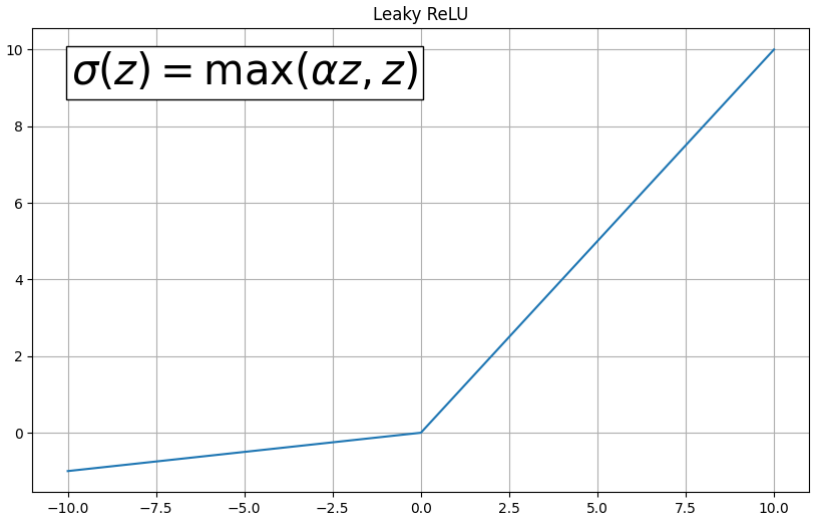

Figura 6: Función de Activación *Leaky* ReLU

- La función ***softmax*** convierte un vector de valores en un vector de probabilidades, donde la suma de todas las probabilidades es 1. Es especialmente útil en la capa de salida para problemas de clasificación multiclase. En la Figura 7 se muestra la definición de la función, y su representación gráfica en la cual el eje horizontal es el valor de entrada z y el eje vertical el valor de salida $\sigma(z)$.

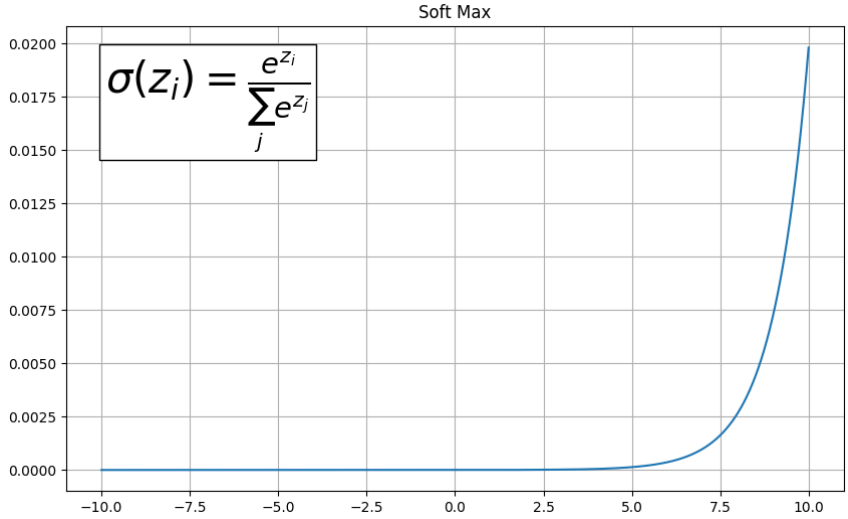

Figura 7: Función de Activación *Softmax*

2.4.3 Arquitecturas de redes neuronales

2.4.3.1 Redes Neuronales *Feedforward* (FFNN)

Las Redes Neuronales *Feedforward*, también conocidas como *Multi-Layer Perceptrons* (MLP), son la forma más simple de red neuronal artificial. En estas redes, la información se procesa en una sola dirección: desde la capa de entrada, a través de las capas ocultas, hasta la capa de salida, sin ciclos ni retroalimentación. Cada neurona en una capa está conectada a todas las neuronas de la siguiente capa, y cada conexión tiene un peso asociado que se ajusta durante el entrenamiento.

Las FFNN son utilizadas para resolver problemas de clasificación y regresión, ya que pueden aproximar funciones complejas y modelar relaciones no lineales entre las variables de entrada y salida. Su simplicidad y estructura directa las hacen una de las arquitecturas más básicas y ampliamente utilizadas en el aprendizaje supervisado.

2.4.3.2 Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales (CNN) están especialmente diseñadas para procesar datos con una estructura de cuadrícula, como las imágenes. A diferencia de las redes neuronales tradicionales, las CNN introducen capas convolucionales que utilizan filtros para detectar características locales, como bordes, texturas y formas, mediante operaciones de

convolución. Estos filtros se aplican a pequeñas regiones de la entrada y permiten que la red capture patrones espaciales jerárquicos, es decir, ordenados de mayor a menor relevancia.

Las CNN suelen incluir capas de *pooling*, que reducen la dimensionalidad de los datos y ayudan a que la detección de características sea robusta frente las variaciones de posición y escala. Esta arquitectura es altamente efectiva para tareas de visión por computadora, como la clasificación de imágenes, detección de objetos y segmentación de imágenes. La capacidad de las CNN para capturar características espaciales y patrones hace que sean especialmente efectivas en tareas de reconocimiento de imágenes y visión por ordenador.

2.4.3.3 Redes Neuronales Recurrentes (RNN)

Las Redes Neuronales Recurrentes (RNN) están diseñadas específicamente para procesar secuencias de datos, como series temporales o texto, donde el orden de los elementos es crucial. A diferencia de las FFNN, las RNN tienen conexiones cíclicas que permiten que la información persista y se propague a través de múltiples pasos de tiempo, creando una memoria interna. Esta característica les permite capturar dependencias temporales y contextuales en los datos.

Sin embargo, las RNN tradicionales pueden enfrentar problemas de *vanishing gradient*, lo que dificulta el aprendizaje de dependencias a largo plazo. Para solucionar esto, se utilizan variantes como las *Long Short-Term Memory* (LSTM) y las *Gated Recurrent Unit (*GRU). Estas variantes incluyen mecanismos que mejoran el manejo de la memoria a largo plazo y aumentan la capacidad de aprendizaje en secuencias largas. Las RNN, son ampliamente utilizadas en tareas como el procesamiento de lenguaje natural, traducción automática, generación de texto, reconocimiento de voz, y análisis de series temporales.

2.4.3.4 Redes Neuronales Generativas Adversas (GAN)

Las *Generative Adversarial Networks* (GAN) o Redes Generativas Adversariales consisten en dos redes neuronales que compiten entre sí: una generadora y una discriminadora. La red generadora crea datos sintéticos que imitan el conjunto de datos reales, mientras que la red discriminadora evalúa si los datos provienen del conjunto real o son generados. Durante el entrenamiento, la generadora mejora en la producción de datos más realistas, mientras que la discriminadora se vuelve más precisa en distinguir entre datos reales y sintéticos.

Este proceso permite que las GAN generen datos altamente realistas y se utilicen en diversas aplicaciones como la creación de imágenes, la generación de música y la mejora de la resolución de imágenes. Las GAN son particularmente innovadoras por su capacidad para generar nuevos datos a partir de ruido aleatorio, expandiendo significativamente las posibilidades en la generación de contenido.

2.4.4 Entrenamiento de redes neuronales

El proceso de entrenamiento de una red neuronal implica ajustar los pesos de todas las entradas de cada una de las neuronas y el valor de los sesgos de estas, con el objetivo de minimizar el error entre las predicciones de la red y las etiquetas reales. Este proceso se realiza mediante el uso del algoritmo de retropropagación (*backpropagation*) y optimizadores como el descenso de gradiente estocástico (SGD).

2.4.4.1 Retropropagación

La retropropagación es un algoritmo de optimización que utiliza el cálculo del gradiente para ajustar los pesos de la red. Involucra dos pasos principales: *forward pass* y b*ackward pass*. Durante el *forward pass*, cada capa de la red procesa la entrada y pasa su salida a la siguiente capa hasta que se obtiene la predicción final. En el *backward pass*, la diferencia entre la predicción y la etiqueta real (error) se propaga de vuelta desde la salida hasta la entrada de la red, calculando los gradientes que indican cómo deben ajustarse los pesos.

2.4.4.2 Métricas clásicas de pérdida

Las métricas de pérdida cuantifican qué tan bien o mal está funcionando el modelo al comparar sus predicciones con los valores reales. Algunas métricas de pérdida comunes incluyen:

- El **Error Cuadrático Medio (MSE)**, calculado según la (1), es una métrica comúnmente usada en problemas de regresión, que mide el promedio de los errores al cuadrado entre las predicciones y los valores reales. Penaliza mucho los errores grandes debido al cuadrado del término.

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

- El **Error Absoluto Medio (MAE)**, calculado según la (2), es similar al MSE, pero toma el valor absoluto de las diferencias entre las predicciones y los valores reales. Es menos sensible a grandes errores en comparación con el MSE.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}| \quad (2)$$

- La **pérdida de entropía cruzada (CEL)**, se calcula con la (3) y es utilizada principalmente en problemas de clasificación, mide la diferencia entre dos distribuciones de probabilidad: las etiquetas reales y las predicciones del modelo. Penaliza los errores en los que la probabilidad predicha para la clase correcta es baja.

$$CEL = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{3}$$

2.4.4.3 Métricas clásicas de rendimiento

Antes de profundizar en las métricas específicas, es fundamental entender los conceptos de verdaderos y falsos en el contexto de las predicciones del modelo. En la Figura 8 se muestra la Matriz de Confusión, siendo esta, una herramienta esencial para evaluar la precisión de un modelo de clasificación. Esta matriz compara las predicciones del modelo con los resultados reales, dividiéndose en cuatro categorías: Verdaderos Positivos (TP), donde el modelo predice correctamente la presencia de una condición; Falsos Negativos (FN), donde el modelo no detecta una condición que está presente; Falsos Positivos (FP), donde el modelo predice una condición que no está presente; y Verdaderos Negativos (TN), donde el modelo predice correctamente la ausencia de una condición. Comprender esta matriz es crucial para evaluar el rendimiento y la precisión de un modelo de clasificación.

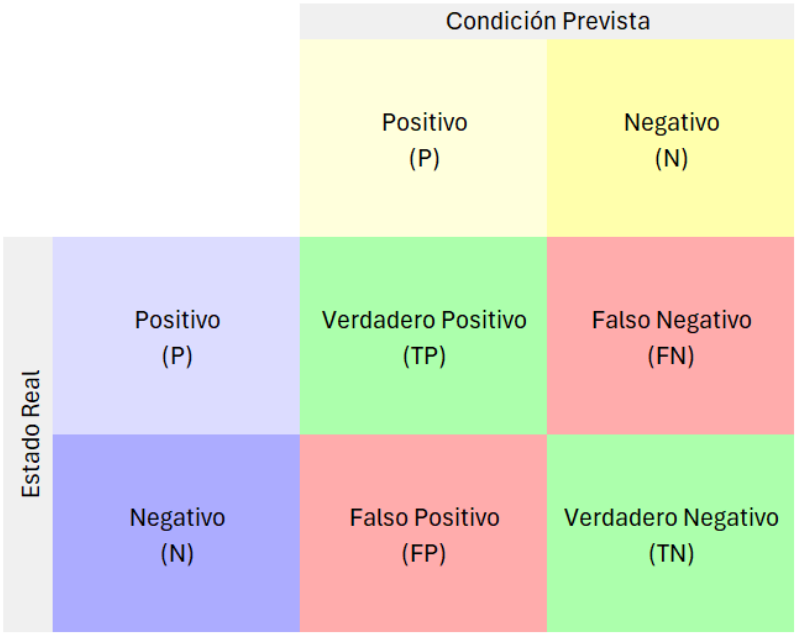

Figura 8: Matriz de Confusión en Evaluación de Modelos de Clasificación

Además de las métricas de pérdida, es importante evaluar el rendimiento general del modelo utilizando métricas adicionales que puedan proporcionar una visión más completa de su efectividad. Algunas métricas de rendimiento comunes incluyen:

- La **exactitud (*accuracy*)**, mostrada en la (4), es la proporción de predicciones correctas realizadas por el modelo en relación con el total de predicciones. Es importante considerar que la exactitud puede no ser la mejor métrica en todos los contextos, especialmente en casos de datos desbalanceados donde una clase puede ser mucho más frecuente que otra. En tales situaciones, otras métricas pueden proporcionar una visión más completa del rendimiento del modelo.

$$Exactitud = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

- La **precisión (*precision*)** mide la proporción de verdaderas instancias positivas entre todas las instancias clasificadas como positivas. Se calcula según la (5) y es especialmente útil en situaciones donde el costo de falsos positivos es alto.

$$Precisión = \frac{TP}{TP + FP} \tag{5}$$

- La **sensibilidad** (***recall)***, mostrada en la (6), mide la proporción de verdaderas instancias positivas que fueron identificadas correctamente por el modelo. Es crucial en contextos donde es importante capturar todos los verdaderos positivos, como en la detección de enfermedades. Se calcula como:

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

- **F1 Score** es la media armónica de la *precision* y el *recall*, proporcionando un equilibrio entre ambos. Se calcula según la (7), y es particularmente útil en problemas de clasificación con clases desbalanceadas.

$$F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{7}$$

- **AUC-ROC (Área bajo la curva - Curva de características operativas 'del receptor)** mide la capacidad del modelo para distinguir entre clases. Un valor de 1 indica un modelo perfecto, mientras que un valor de 0.5 indica un modelo cuya capacidad de discriminación no es mejor que el azar. Esta métrica es útil para comparar modelos y elegir el más adecuado para problemas específicos de clasificación.

Cada métrica de rendimiento proporciona una perspectiva única sobre el comportamiento del modelo, y la elección de las métricas adecuadas es esencial para abordar el problema específico en cuestión. La selección de métricas debe considerar los costos asociados a los errores de predicción, como falsos positivos y falsos negativos. Por ejemplo, en un sistema de detección de fraudes, minimizar los falsos negativos es crucial para capturar todos los casos fraudulentos, mientras que en una campaña de marketing es más importante minimizar los falsos positivos para evitar desperdiciar recursos en clientes no interesados.

2.4.5 Optimización

La optimización es una parte fundamental del entrenamiento de redes neuronales, ya que se encarga de ajustar los pesos de la red para minimizar la función de pérdida. La función de pérdida mide cuán lejos están las predicciones de la red de los valores reales, y el objetivo del proceso de optimización es reducir este error lo máximo posible.

Si ponemos un ejemplo sencillo para entenderlo: imaginemos que tratamos de encontrar el punto más bajo en un paisaje montañoso; los optimizadores actúan como guías que te indican en qué dirección moverte para bajar lo más rápido posible. Algunos de los optimizadores más comunes incluyen:

- El **Descenso de Gradiente Estocástico (SGD)** es uno de los métodos más simples y comunes. En lugar de usar todos los datos de entrenamiento de una vez, SGD toma un subconjunto (*mini-batch*) de los datos en cada iteración y ajusta los pesos de la red en función de ese pequeño conjunto. Esto hace que el proceso sea más rápido y eficiente, aunque introduce algo de variabilidad en los ajustes de los pesos.
- **Adagrad** ajusta la tasa de aprendizaje en función de la frecuencia de actualización de los parámetros. Los parámetros que se actualizan con frecuencia reciben ajustes más pequeños, mientras que los parámetros que se actualizan con menos frecuencia reciben ajustes mayores. Esto es útil para características esporádicas pero importantes.
- **RMSprop** es otro optimizador que adapta la tasa de aprendizaje para cada parámetro. Este método es especialmente útil para problemas en los que los gradientes pueden variar mucho en tamaño, ayudando a estabilizar el proceso de entrenamiento y acelerando la convergencia.
- **Adam (*Adaptive Moment Estimation*)** es un optimizador más avanzado que combina las ideas de dos otros métodos: AdaGrad y RMSProp. Ajusta las tasas de aprendizaje de forma individual para cada parámetro, lo que ayuda a que el proceso de optimización sea más eficiente y estable. Adam es popular porque funciona bien en la práctica para una amplia gama de problemas.

2.4.6 Proceso de entrenamiento

1. En el primer paso, los pesos de la red neuronal se inicializan. Este proceso suele hacerse de manera aleatoria para romper la simetría y asegurar que la red empiece con una diversidad de valores que permitan un aprendizaje efectivo.
2. Las entradas del conjunto de datos se envían a través de la red neuronal. Durante este paso, cada capa de la red transforma sus entradas en salidas hasta que se generan las predicciones finales. Esta secuencia de transformaciones se realiza mediante funciones de activación aplicadas a combinaciones lineales de los pesos y las entradas.

3. Una vez obtenidas las predicciones, se calcula la pérdida o error del modelo comparando las predicciones con los valores reales del conjunto de datos. Para ello, se emplea una función de pérdida adecuada, que puede variar según el tipo de problema (p. ej., la entropía cruzada para clasificación).
4. Con la pérdida calculada, se procede a la retropropagación de errores. Este paso implica calcular los gradientes de la función de pérdida con respecto a cada peso de la red, usando el algoritmo de retropropagación. Los gradientes indican la dirección y magnitud en la que cada peso debe ajustarse para minimizar la pérdida.
5. Los pesos de la red se ajustan utilizando un optimizador, que aplica los gradientes calculados en el paso anterior. Estos optimizadores utilizan los gradientes para actualizar los pesos de manera iterativa, buscando minimizar la función de pérdida.
6. El proceso de entrenamiento se repite durante múltiples épocas. Una época consiste en pasar por todo el conjunto de datos una vez. Durante cada época, los pasos de *forward pass* (paso 2), cálculo de la pérdida (paso 3), *backward pass* (paso 4) y actualización de pesos (paso 5) se realizan de manera iterativa. El entrenamiento continúa hasta que la pérdida se minimiza suficientemente, el modelo converge o se alcanza un número máximo de épocas predeterminado.

Este ciclo continuo de ajustes y evaluaciones permite que la red neuronal mejore su capacidad de aprender patrones complejos a partir de los datos de entrenamiento.

2.5 Visión por ordenador

La visión por ordenador, también conocida como visión artificial, es una técnica de inteligencia artificial dedicada a permitir que las máquinas interpreten y comprendan la información visual del mundo. Esto implica el análisis de imágenes y vídeos para medir distancias, detectar movimientos, encontrar diferencias entre imágenes, extraer información cuantitativa referente a patrones, etc.

2.5.1 Antecedentes

Las aplicaciones de la visión por ordenador abarcan una amplia gama de áreas, desde vehículos autónomos que perciben y navegan por su entorno, hasta sistemas de reconocimiento facial utilizados para seguridad y herramientas de análisis de imágenes que detectan defectos en procesos de manufactura. En medicina, se utiliza para el análisis de imágenes hiperespectrales médicas, permitiendo la detección de tejidos patógenos [4]. En el ámbito del fitness, ayuda a detectar la postura correcta durante la realización de ejercicios [5]. En seguridad, las cámaras de vigilancia equipadas con visión por ordenador pueden registrar todos los accesos de los vehículos en una zona determinada [6]. En acuicultura, ya se utiliza para monitorear el comportamiento y la salud de los peces, detectar enfermedades y optimizar las condiciones de cría, mejorando así la eficiencia y sostenibilidad del proceso productivo en experimentos de *Open Field* [7].

2.5.2 Detección de Objetos

La detección de objetos es el proceso de identificar y localizar instancias de objetos de interés dentro de una imagen o vídeo. Un ejemplo de esta tecnología puede verse en la Figura 9. Utiliza distintos modelos, destacando recientemente las redes neuronales convolucionales para identificar patrones característicos de los objetos en los datos de entrada. Esta técnica es fundamental en aplicaciones como la vigilancia, donde es crucial identificar y seguir objetos específicos en tiempo real, así como en la automatización de fábricas donde se necesita reconocer y manipular componentes.

Figura 9: Ejemplo de Detección de Objetos

[Tomado de Ultralytics: https://docs.ultralytics.com/es/tasks/detect/]

2.5.3 Segmentación de Instancias

La segmentación de instancias no solo identifica objetos en una imagen, sino que también los separa en diferentes segmentos, permitiendo una comprensión más detallada de la estructura de la escena. Esta técnica es esencial en aplicaciones que requieren una precisión detallada, como la cirugía asistida por robots, donde es crucial distinguir entre diferentes tejidos y órganos. La Figura 10 muestra un ejemplo de uso de esta tecnología.

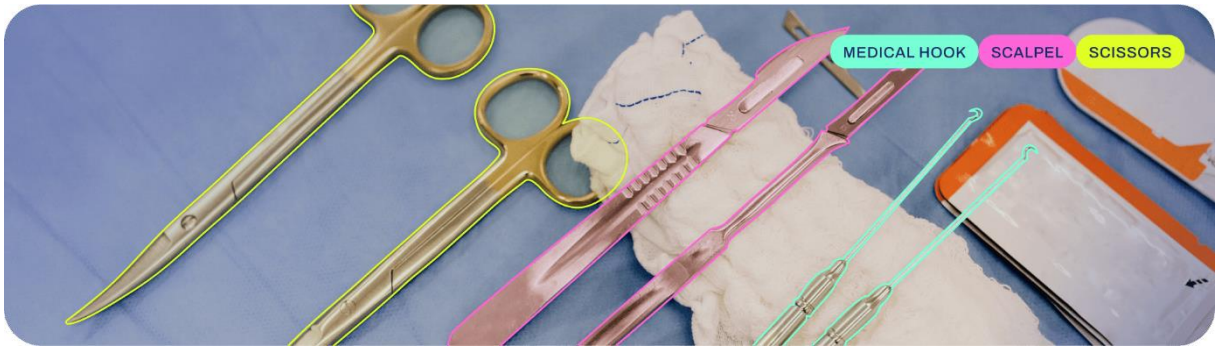

Figura 10: Ejemplo de Segmentación de Instancias

[Tomado de Ultralytics: https://docs.ultralytics.com/es/tasks/segment/]

2.5.4 Clasificación de Imágenes

La clasificación de imágenes es la tarea de categorizar una imagen completa en una de varias categorías predefinidas, un ejemplo de esto puede verse en la Figura 11. Esta técnica se utiliza en aplicaciones como la detección de enfermedades a partir de imágenes médicas y la clasificación de productos en plataformas de comercio electrónico. Los modelos de clasificación de imágenes aprenden a reconocer patrones y características distintivas de las diferentes clases, mejorando la precisión en la identificación.

Figura 11: Ejemplo de Clasificación de Imágenes

[Tomado de Ultralytics: https://docs.ultralytics.com/es/tasks/classify/]

2.5.5 Estimación de Pose

La estimación de pose implica la detección de la postura de un objeto o persona en una imagen, identificando la posición y orientación de sus partes clave. Es crucial en aplicaciones de realidad aumentada y análisis de movimiento. Esta técnica se utiliza para crear experiencias de usuario más interactivas y precisas en aplicaciones de entretenimiento y deportivas. Un ejemplo de uso es el mostrado en la Figura 12.

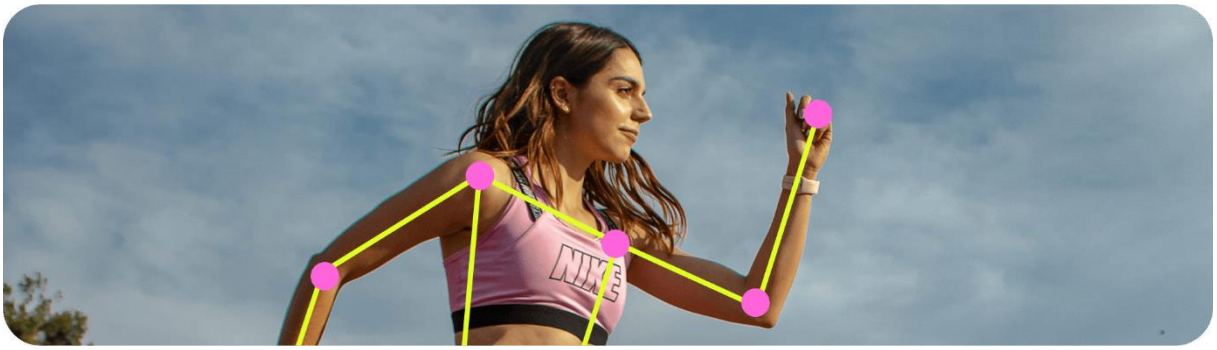

Figura 12: Ejemplo de Estimación de Pose

[Tomado de Ultralytics: https://docs.ultralytics.com/es/tasks/pose/]

2.6 YOLO

YOLO (*You Only Look Once*) es una popular arquitectura de red neuronal y un algoritmo de detección de objetos en tiempo real, desarrollado por Joseph Redmon. A diferencia de los métodos tradicionales que dividen la detección de objetos en múltiples etapas, YOLO unifica estas etapas en un solo paso, permitiendo procesar imágenes completas en una única ejecución de la red.

Actualmente, Ultralytics es la empresa propietaria de la red neuronal. Aunque la red se distribuye bajo la licencia AGPL-03 de código abierto, lo que permite su uso gratuito siempre y cuando no sea con fines comerciales. Ultralytics desempeña un papel fundamental en el desarrollo y mejora continua de YOLO, proporcionando implementaciones avanzadas y herramientas accesibles que facilitan el uso del algoritmo en diversas aplicaciones prácticas.

2.6.1 Principios Fundamentales de YOLO

El funcionamiento de YOLO se puede desglosar en los siguientes principios clave:

- Cuadrícula de Divisiones: YOLO divide la imagen de entrada en una cuadrícula de celdas. Cada celda es responsable de predecir un número fijo de cajas delimitadoras (*bounding boxes*) y las probabilidades de clase para esos recuadros si el centro del objeto cae dentro de la celda.
- Predicciones por Celda: Cada celda de la cuadrícula predice varias cajas delimitadoras y una puntuación de confianza para cada caja. Esta puntuación de confianza refleja tanto la precisión de la caja como la probabilidad de que dicha caja contenga un objeto.
- Cajas Delimitadoras: Cada predicción de caja delimitadora incluye coordenadas (x, y) para el centro de la caja, el ancho y alto de la caja, y la confianza de la predicción. Las coordenadas están normalizadas con respecto a las dimensiones de la celda.
- Clasificación de Objetos: Además de las coordenadas de la caja y la puntuación de confianza, cada celda predice las probabilidades de clase para los objetos. Estas probabilidades están condicionadas a que la celda contenga un objeto.

2.6.2 Versiones de YOLO

Desde su introducción en 2015, YOLO ha evolucionado a través de varias versiones, cada una mejorando aspectos clave del modelo anterior. Las principales variantes incluyen:

- YOLOv1: La versión original de YOLO introdujo el enfoque de detección en tiempo real y demostró su eficacia en tareas de detección de objetos.
- YOLOv2 (YOLO9000): Mejoró la precisión y permitió la detección de una mayor cantidad de clases. Introdujo técnicas como el anclaje de cajas y una mejor representación de las características.

- YOLOv3: Aumentó aún más la precisión y la velocidad, incorporando un diseño de red más profundo y mejorado, así como la detección a múltiples escalas.
- YOLOv4: Introdujo mejoras adicionales en la eficiencia y precisión, utilizando técnicas avanzadas de preprocesamiento de imágenes y optimización de redes neuronales.
- YOLOv5: Aunque no fue desarrollado por los creadores originales, YOLOv5 ha ganado popularidad en la comunidad debido a mejoras de rendimiento y su facilidad de uso.
- YOLOv8: La versión más reciente que incorpora avances significativos en velocidad y precisión, haciendo uso de técnicas de vanguardia en *deep learning* y optimización.

2.6.3 Implementación de YOLO

Implementar YOLO para la detección de objetos implica los siguientes pasos generales:

- Preparación del Conjunto de Datos: Recopilación y etiquetado de un conjunto de datos de imágenes que contengan los objetos de interés. Las etiquetas deben incluir las coordenadas de las cajas delimitadoras y las clases de los objetos.
- Entrenamiento del Modelo: Utilización de entornos de trabajo de *deep learning* como TensorFlow para configurar y entrenar el modelo YOLO en el conjunto de datos preparado. Este proceso implica ajustar los hiperparámetros del modelo y monitorear la pérdida durante el entrenamiento para asegurar la convergencia.
- Evaluación del Modelo: Después del entrenamiento, se evalúa el modelo en un conjunto de datos de prueba para medir su precisión y capacidad de generalización. Las métricas comunes incluyen la media de precisión promedio (mAP) y la curva de características operativas del receptor (ROC).
- Implementación en Producción: Una vez que el modelo ha sido entrenado y evaluado, se puede implementar en un entorno de producción para realizar detecciones en tiempo real. Esto puede implicar la integración del modelo en aplicaciones de software, dispositivos móviles, sistemas de vigilancia, vehículos autónomos, entre otros.

2.6.4 Resultados/Estadísticas

Los resultados del entrenamiento de YOLOv8 se evalúan mediante diferentes métricas divididas en métricas de pérdida y métricas de evaluación de rendimiento.

2.6.4.1 Métricas de Pérdida

- *box_loss*: Mide la precisión con la que el modelo predice las coordenadas de los cuadros delimitadores. Es crucial para asegurar que las cajas predichas se alineen correctamente con los objetos en la imagen.

- *pose_loss*: Evalúa la exactitud de la estimación de pose. Esta métrica es particularmente importante en aplicaciones donde la orientación y la postura del objeto son críticas.
- *kobj_loss*: Mide la pérdida asociada con la identificación de objetos clave. Esto ayuda a evaluar qué tan bien el modelo puede detectar objetos específicos que son de particular interés en la aplicación.
- *cls_loss*: Evalúa la precisión de la clasificación de objetos. Asegura que los objetos detectados sean correctamente clasificados en sus respectivas categorías.
- *dfl_loss*: Mide la pérdida de distribución focal. Esta métrica evalúa la capacidad del modelo para enfocarse en las partes más relevantes de la imagen, mejorando así la precisión en la detección de objetos.

2.6.4.2 Métricas de Evaluación de Rendimiento

- *Precision*: Mide la proporción de verdaderos positivos entre todas las predicciones positivas. Esta métrica es crucial para evaluar la exactitud del modelo en identificar correctamente los objetos de interés.
- *Recall*: Mide la proporción de verdaderos positivos entre todas las instancias positivas en los datos. Evalúa qué tan bien el modelo puede identificar todos los objetos relevantes sin dejar de detectar algunos.
- *AP50*: Área bajo la curva de *precision-recall* a un umbral de 0.5. Intuitivamente, esta métrica nos dice qué tan bien el modelo puede detectar objetos con una tolerancia de error moderada (donde se permite un 50% de superposición).
- *AP50-95*: Promedio de AP en múltiples umbrales, proporcionando una evaluación más robusta de la precisión del modelo. Intuitivamente, nos dice qué tan bien el modelo puede detectar objetos con diferentes niveles de tolerancia, desde una superposición moderada (50%) hasta una superposición muy precisa (95%).

2.6.5 Ventajas de YOLO

Una de las principales ventajas de YOLO es su capacidad para procesar imágenes en tiempo real. Esto lo hace ideal para aplicaciones donde la rapidez es crítica, como la vigilancia en tiempo real, la conducción autónoma y la interacción en aplicaciones móviles.

Además la arquitectura de YOLO es más sencilla y directa en comparación con otros métodos de detección de objetos que requieren múltiples etapas. Esto facilita su implementación y ajuste, lo que permite a los desarrolladores integrarlo rápidamente en sus proyectos. Por otra parte, YOLO ofrece una buena precisión en la detección de objetos en una variedad de contextos y tamaños. Aunque puede tener limitaciones en la detección de objetos muy pequeños, su rendimiento general en términos de *precision* y *recall* es competitivo.

2.6.6 Limitaciones de YOLO

- Detección de Objetos Pequeños: YOLO puede tener dificultades para detectar objetos muy pequeños dentro de una imagen. Debido a su división de la imagen en una cuadrícula, los objetos pequeños pueden no ser capturados con precisión suficiente si caen dentro de una celda que los diluye.
- Superposición de Objetos: En escenas complejas con muchos objetos superpuestos, la precisión de YOLO puede disminuir. Esto se debe a que las celdas de la cuadrícula pueden no manejar bien las intersecciones y superposiciones densas, lo que puede llevar a errores en la detección.
- Dependencia del Tamaño de la Celda: La precisión de las predicciones de YOLO depende de la resolución de la cuadrícula utilizada para dividir la imagen. Si la cuadrícula es demasiado grande, puede perder detalles importantes; si es demasiado pequeña, puede aumentar la complejidad computacional y el tiempo de procesamiento.
- Balance entre Velocidad y Precisión: Aunque YOLO es extremadamente rápido, este rendimiento puede venir a costa de una ligera disminución en la precisión en comparación con métodos más lentos y detallados. Es crucial encontrar un equilibrio adecuado según los requisitos específicos de la aplicación.

2.7 Conjuntos de Datos (*Datasets*)

En el desarrollo y entrenamiento de modelos de visión por computador, como YOLO (*You Only Look Once*), los conjuntos de datos juegan un papel crucial. Los conjuntos de datos son la base sobre la cual se construyen los modelos de aprendizaje automático. La calidad y diversidad de los datos determinan la precisión y la capacidad de generalización del modelo. En la visión por computador, los conjuntos de datos contienen imágenes y, a menudo, anotaciones que describen el contenido de cada imagen, lo que permite al modelo aprender a reconocer patrones visuales.

2.7.1 Características de un Buen *Dataset*

2.7.1.1 Diversidad

Un buen conjunto de datos debe incluir una amplia variedad de imágenes que representen todas las posibles variaciones del objeto o escena que se desea detectar. Esto incluye variaciones en términos de iluminación, ángulos, escalas, fondos y condiciones ambientales. Además, es importante considerar la diversidad de perspectivas, como vistas aéreas, laterales, frontales y traseras. Una mayor diversidad en el conjunto de datos permite al modelo aprender a manejar una gama más amplia de situaciones del mundo real, lo que mejora su robustez y capacidad de generalización. No tener un nivel de variedad suficiente puede llevar al modelo a sobre especializarse en las condiciones de entrenamiento y por

ende no poder abstraer información. Por ejemplo si un modelo pretende diferenciar entre coches y motos, si todos los coches del *dataset* son rojos, es posible, que al presentarle al modelo un coche blanco, no sepa clasificarlo correctamente ya que ha asociado el color como una característica clave de los coches.

2.7.1.2 Etiquetado Preciso

Las anotaciones de los conjuntos de datos deben ser precisas y consistentes. En el caso de YOLO, cada imagen necesita estar etiquetada con las coordenadas de los cuadros delimitadores (*bounding boxes*) que contienen los objetos de interés, junto con las respectivas clases de los objetos, y si el modelo es especifico de ello, junto con los puntos clave (*keypoints*). Inconsistencias en el etiquetado pueden llevar a un entrenamiento defectuoso, donde el modelo aprende patrones incorrectos o imprecisos, disminuyendo su efectividad en la detección de objetos. Por ejemplo, si se está entrenando un modelo para poder reconocer ojos, y el centro del ojo no es precioso, es posible que, por ejemplo, confunda el lagrimal con el iris.

2.7.1.3 Tamaño del Conjunto de Datos

El tamaño del conjunto de datos es un factor crucial para el entrenamiento exitoso de modelos de visión por computador. Los modelos de YOLO requieren grandes cantidades de datos para entrenarse efectivamente. Un conjunto de datos grande y variado ayuda a asegurar que el modelo pueda generalizar bien a nuevas imágenes que no haya visto antes. Por ejemplo, un conjunto de datos para detección de rostros que incluye millones de imágenes de personas en diferentes edades, géneros y etnias, permitirá al modelo aprender una representación robusta de los rostros humanos. Además, un gran volumen de datos también ayuda a suavizar los errores que pueden surgir debido a anotaciones imperfectas, ya que el modelo puede aprender a partir de la tendencia general de los datos en lugar de sobre ajustarse a errores específicos.

2.7.1.4 Equilibrio de Clases

Es importante que el conjunto de datos esté equilibrado, es decir, que contenga una cantidad similar de ejemplos para cada clase que el modelo necesita aprender (una clase es a lo que se conoce como a un objeto que se quiere detectar). Un conjunto de datos desequilibrado puede llevar a un modelo que favorezca las clases más representadas, resultando en un rendimiento deficiente para las clases menos comunes.

2.7.2 Fuentes Públicas de *Datasets*

Existen numerosos conjuntos de *datasets* disponibles para entrenar modelos de visión por computador, cada uno con sus propias características y aplicaciones. Por ejemplo, Imagenet[3] cuenta con más de 14 millones de imágenes y 20,000 categorías, es uno de los conjuntos de datos más grandes disponibles. Imagenet ha sido crucial para el avance de los modelos de clasificación de imágenes, proporcionando una base sólida para la investigación y el desarrollo de nuevas arquitecturas de redes neuronales.

2.7.3 Creación de Conjuntos de Datos Propios

Para aplicaciones específicas, puede ser necesario crear un conjunto de datos propio. Este proceso implica varios pasos importantes, desde la recolección de imágenes hasta su anotación y validación.

2.7.3.1 Etiquetado de Imágenes

Es fundamental asegurar que las imágenes recolectadas sean representativas de las condiciones en las que se utilizará el modelo. Una vez recolectadas suficientes imágenes, se pasa a la anotación o el etiquetado, que puede ser un proceso laborioso pero crucial para la precisión del modelo. En el contexto de la visión por ordenador, el etiquetado puede incluir la identificación de objetos en las imágenes, la segmentación de regiones específicas y la anotación de *keypoints*. Herramientas como CVAT (*Computer Vision Annotation Tool*) se utilizan para facilitar este proceso. Esta herramienta permite la anotación detallada de puntos clave en las imágenes, proporcionando varias propiedades útiles para cada *keypoint*.

- **Visible**: Se asigna cuando el *keypoint* es claramente identificable en la imagen.
- **Oculto**: Se utiliza para *keypoints* que, aunque no se vean directamente debido a la postura de la trucha o a obstrucciones parciales causadas por la red, se pueden deducir razonablemente su posición.
- **Fuera de límites**: Se aplica a *keypoints* que se encuentran más allá del borde de la imagen y, por lo tanto, no se pueden anotar con precisión.

Los fotogramas se han clasificado en diferentes tipos según la claridad de los *keypoints* y la calidad de la imagen, es decir, según las propiedades de los *keypoints* y la borrosidad de la imagen. Las categorías de fotogramas utilizadas son las siguientes:

- **Imágenes triviales**: Representan el caso ideal donde la trucha se encuentra de lado y todos los *keypoints* son normalmente **visibles** y fácilmente identificables.
- **Imágenes ambiguas**: Aunque la trucha se encuentra perfectamente nítida, la orientación hace difícil determinar todos los *keypoints*. En estos fotogramas, algunos *keypoints* pueden estar **ocultos**, pero su posición puede deducirse razonablemente.

[3] "ImageNet," Imagenet.org. [En línea]. Disponible en: http://www.image-net.org. [Accedido: 24-jun-2024].

- **Imágenes semi-borrosas**: A pesar de no tener buena calidad, es posible identificar los puntos de interés. En estos casos, la mayoría de los *keypoints* son **visibles**, aunque con menor claridad.
- **Imágenes borrosas**: Algunos *keypoints* se pueden intuir y otros son interpretables. En estos fotogramas, algunos *keypoints* pueden estar **fuera de límites** o estar **ocultos**, pero se puede hacer una estimación razonable de su ubicación.
- **Imágenes imposibles**: Sin el contexto de los fotogramas de los instantes anteriores o posteriores, es prácticamente imposible reconocer la postura de la trucha o identificar muchos de los *keypoints*. En estos casos, los *keypoints* pueden estar **fuera de límites** o esta0072 **ocultos** además de no tener una referencia clara. Si un ser humano no puede determinar correctamente los *keypoints* en este tipo de imágenes, es muy difícil que el modelo lo logre.

Es importante realizar revisiones periódicas de las anotaciones para asegurar la consistencia y corrección de los datos, lo que a su vez mejora la calidad del entrenamiento del modelo. En proyectos colaborativos, la revisión cruzada entre diferentes personas puede ser muy útil para mantener la coherencia y la precisión de las anotaciones.

2.7.3.2 Formatos de anotación de puntos

En el mundo de la visión por ordenador existen muchas formas diferentes de denominar un mismo punto dentro de una imagen dada. La anotación se puede dividir en dos secciones principales:

- **Datos**: En esta parte se incluyen las coordenadas que describen la posición exacta de los puntos de interés dentro de la imagen. Estas coordenadas pueden ser absolutas o normalizadas según el formato de anotación utilizado. En el caso de la detección de objetos y *keypoints*, las coordenadas típicas incluyen:
    - ***Bounding Boxes***: Se dan los límites del rectángulo que encierra el objeto.
    - ***Keypoints***: Para cada punto de interés, se registran las coordenadas (x, y) y la visibilidad (v), que puede tomar los valores descritos en el 2.7.3.1

- **Metadatos**: Aquí se registra información adicional sobre cómo se conectan los puntos y otras características contextuales. Estos metadatos pueden incluir:
    - **Esqueletos**: En la estimación de pose, los esqueletos definen cómo se conectan los *keypoints* para representar articulaciones y segmentos del cuerpo.
    - **Nombre de etiquetas**: Cada objeto o punto anotado está asociado con una clase específica, indicando qué tipo de objeto o punto es. Por ejemplo, en un *dataset* de estimación de pose humana, las clases podrían incluir "codo", "rodilla", "muñeca", etc.

2.7.3.2.1 YOLO

El formato YOLO es conocido por su simplicidad y eficiencia, especialmente diseñado para la detección de objetos en tiempo real. Las anotaciones en YOLO se representan en un archivo de texto separado para cada imagen, y ambos archivos deben tener el mismo nombre.

El archivo de anotaciones está organizado de tal manera que cada instancia de un objeto presente en la imagen ocupa una línea distinta en el archivo de etiquetas. Cada entrada o línea de anotación sigue el formato de la Figura 13. En este formato, las coordenadas están normalizadas con respecto al tamaño de la imagen.

- `class_id`**:** Identificador numérico de la clase del objeto.
- `x_center, y_center`: Coordenadas del centro de la *bounding box*
- `width, height`: Ancho y alto de la *bounding box*
- `x1, y1, v1, x2, y2, v2, ..., xN, yN, vN`: Coordenadas y visibilidad de cada *keypoint*. La visibilidad indica si el punto es visible (2), oculto (1) o fuera de limites (0).

Los metadatos adicionales, como las etiquetas de clase y la relación entre los *keypoints*, se definen en el archivo de configuración del modelo cuando se vaya a entrenar.

```
class_id x_center y_center width height x1 y1 v1 x2 y2 v2 ... xN yN vN
```

Figura 13: Formato de Anotación YOLO para *Keypoints*

2.7.3.2.2 COCO

El formato COCO (*Common Objects in Context*) es uno de los estándares más completos y ampliamente utilizados para la anotación de imágenes en visión por ordenador. Este formato es particularmente popular debido a su capacidad para manejar anotaciones complejas y detalladas, como la segmentación de objetos, la detección de *keypoints* y las relaciones espaciales entre objetos.

COCO se basa en archivos JSON que estructuran las anotaciones en varias categorías, incluyendo imágenes, anotaciones y categorías. Todos los datos y metadatos de cada anotación se encuentran en un mismo fichero. Un archivo de anotación COCO típicamente sigue el formato mostrado en la Figura 14. En este formato, las coordenadas son absolutas al tamaño de la imagen.

- **Categorías**: Define las clases de objetos y, en el caso de la estimación de pose, los *keypoints* y sus conexiones (esqueleto)
  - `id`: Identificador único de la categoría.
  - `name`: Nombre de la categoría.
  - `keypoints`: Lista de nombres de los *keypoints*.
  - `skeleton`: Lista de índices que definen las conexiones entre los *keypoints*

- **Imágenes**: Contiene información sobre cada imagen en el conjunto de datos.
  - `id`: Identificador único de la imagen.
  - `width`: Ancho de la imagen.
  - `height`: Alto de la imagen.
  - `file_name`: Nombre del archivo de la imagen.
- **Anotaciones**: Contiene las anotaciones específicas para cada objeto detectado en las imágenes.
  - `id`: Identificador único de la anotación.
  - `image_id`: Identificador de la imagen a la que pertenece la anotación.
  - `category_id`: Identificador de la categoría del objeto.
  - `bbox`: Coordenadas de la *bounding box*, especificadas como [x, y, ancho, alto], donde (x, y) representa la esquina superior izquierda de la caja.
  - `keypoints`: Coordenadas de los *keypoints* y su visibilidad
  - `num_keypoints`: Número total de *keypoints* anotados en la imagen.

```
{
    "categories":[
        {   "id": 1, "name": "Trucha",
            "keypoints": ["Nariz", "Branquia", "Aleta_Pectoral",
                          "Aleta_Dorsal", "G", "Aleta_Pelvica",
                          "Cola_Sup", "Cola_Inf", "Cola_Mid"],
            "skeleton": [[2,1], [6,5], [5,4],
                         [2,3], [9,5], [8,9],
                         [9,7], [7,8], [5,2]]
        }, . . .
    ],
    "images": [
        {"id":1,"width":960,"height":1080,"file_name":"L_001.jpg"},
      {"id":2,"width":960,"height":1080,"file_name":"L_003.jpg"}, . . .
    ],
    "annotations": [
        {       "id":1, "image_id":1, "category_id":1,
                "bbox":[X, Y, width, height],
                "keypoints":[X1,Y1,V1, X2,Y2,V2, . . ., X9,Y9,V9],
                "num_keypoints":9
        },
        {      "id":2, "image_id":2, "category_id":1,
                "bbox":[X, Y, width, height],
                "keypoints":[X1,Y1,V1, X2,Y2,V2, . . ., X9,Y9,V9],
                "num_keypoints":9
        }, . . .
    ]
}
```

Figura 14: Ejemplo de Formato de Anotación COCO para *Keypoints*

2.7.3.3 Aumento de Datos (*Data Augmentation*)

Como se ha mencionado anteriormente, es crucial que un *dataset* posea una suficiente variedad de colores, formas y perspectivas. Si el conjunto original de imágenes carece de alguna de estas cualidades o si la obtención de nuevas imágenes etiquetadas implica un coste elevado de recursos, se pueden aplicar técnicas de *data augmentation* para aumentar la diversidad del conjunto de datos.

El *data augmentation* puede incluir diversas transformaciones geométricas como rotaciones, traslaciones y escalados, así como transformaciones de color, ajustes de brillo y contraste, o añadir ruido como la pérdida de píxeles y borrosidad. Estas modificaciones permiten que el modelo vea variaciones más amplias de las imágenes durante el entrenamiento, mejorando su capacidad de generalización y reduciendo el riesgo de sobreajuste. Además, como se ha dicho, esta técnica es especialmente útil en situaciones donde la recolección de datos adicionales es costosa o impracticable, proporcionando una solución efectiva para mejorar la robustez del modelo con los datos disponibles.

La Figura 15 muestra ejemplos de técnicas de *data augmentation*, incluyendo volteo horizontal, recorte, desenfoque, ajuste de contraste, modificación de matiz, saturación y valor, y ajuste de gamma. Estas transformaciones demuestran cómo es posible diversificar un conjunto de datos a partir de una única imagen original.

Lo realmente interesante es que, si se hace correctamente con imágenes ya etiquetadas, junto con las nuevas imágenes se generarán las correspondientes etiquetas, lo que facilita enormemente el proceso de anotación y asegura la consistencia de los datos generados.

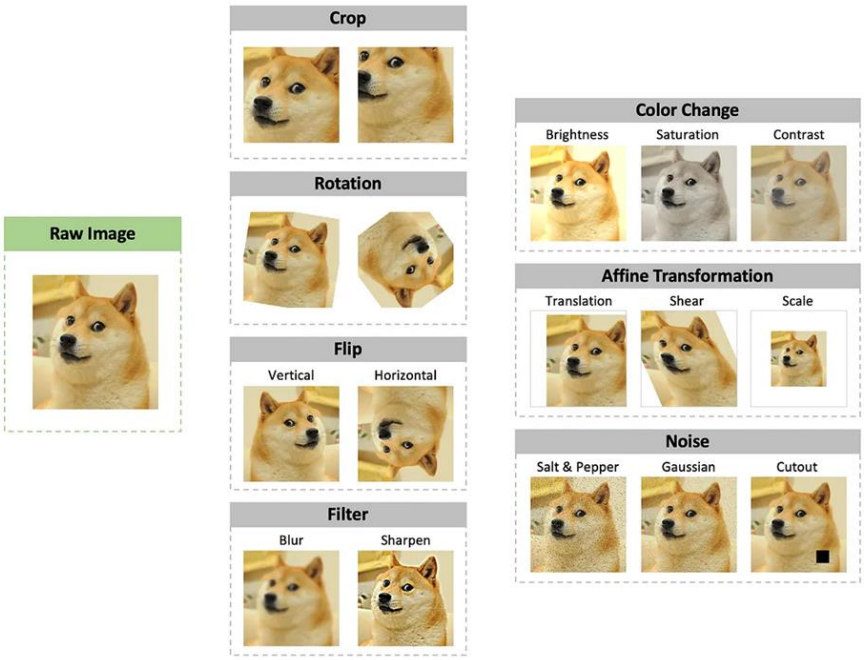

Figura 15: Ejemplos de Técnicas de *Data Augmentation*
[Tomado de: https://towardsdatascience.com/complete-guide-to-data-augmentation-for-computer-vision-1abe4063ad07]

2.7.4 División del Conjunto de Datos

Finalmente, para entrenar y evaluar un modelo de manera efectiva, es común dividir el conjunto de datos en tres subconjuntos principales:

- Entrenamiento (*Training Set*): Este subconjunto se utiliza para entrenar el modelo. Debe contener la mayor parte de las imágenes para asegurar que el modelo tenga suficientes ejemplos para aprender. Generalmente, se recomienda asignar aproximadamente el 70-80% del conjunto de datos total a este subconjunto.
- Validación (*Validation Set*): Este subconjunto se utiliza para ajustar los hiperparámetros del modelo y para evitar el sobreajuste. Durante el entrenamiento, el rendimiento del modelo en el conjunto de validación se supervisa para tomar decisiones sobre ajustes en la arquitectura del modelo o en los hiperparámetros. Aproximadamente el 10-15% del conjunto de datos total se debe asignar a este subconjunto.
- Prueba (*Test Set*): Este subconjunto se utiliza para evaluar el rendimiento final del modelo después de que se ha completado el entrenamiento. No se utiliza en absoluto durante el entrenamiento ni en la validación, asegurando así una evaluación imparcial del rendimiento del modelo. Aproximadamente el 10-15% del conjunto de datos total se debe asignar a este subconjunto.

Crear un conjunto de datos propio y asegurarse de que esté bien anotado y dividido adecuadamente puede ser un desafío, pero es esencial para el desarrollo de modelos precisos y robustos de visión por computador.

3. Especificaciones y restricciones de diseño

En el desarrollo de la aplicación final de detección y análisis de movimiento de una trucha, se tendrán en cuenta las siguientes especificaciones:

- Todo el código que se desarrolle deberá ser escrito en Python.
- La resolución de todos los vídeos a usar debe ser programable, nunca pudiendo ser inferior a 720p, además el formato de estos será en mp4.
- Se partirá de una serie de vídeos grabados por el Grupo de Producción Animal (GAP) del Centro de Estudios e Investigación para la Gestión de Riesgos Agrarios y Medioambientales (CEIGRAM) en los que aparecen simultáneamente dos truchas en vista cenital.
- La interfaz de la aplicación final debe ser intuitiva y amigable para un usuario no familiarizado con la programación ni el aprendizaje automático.
- El código debe estar correctamente documentado para su posterior revisión, mejora y posible ampliación por parte de otros desarrolladores.
- Las especificaciones mínimas de la aplicación final deberán ser lo más bajas posible para que cualquier ordenador medio de un laboratorio sea capaz de ejecutarla sin problemas.

También se deberán tener en cuenta que el proyecto se debe desarrollar bajo el marco impuesto por las siguientes restricciones:

- La potencia de computación disponible (tarjetas gráficas) para el entrenamiento del modelo es limitada.
- La cantidad de vídeos disponibles es escasa no ampliable. Se pueden llegar a proponer mejoras para futuras grabaciones.

4. Descripción de la solución propuesta

En este capítulo se presenta la solución propuesta para el desarrollo y entrenamiento de una red neuronal. Se ha utilizado un modelo ya preentrenado, especializado para aprender a detectar *keypoints*, el modelo en concreto es YOLOv8-n. Además, para hacer el entrenamiento, se va a emplear un conjunto de datos propio.

Debido a la metodología de entrenamiento de redes neuronales, previamente explicada en el 2.4.6, algunos de los pasos del proceso se han tenido que realizar varias veces, modificando algunos parámetros, para poder obtener un modelo significativamente mejor. Para mantener la claridad y coherencia, las iteraciones con mayor relevancia de un mismo paso (normalmente la primera y última iteración) serán las que se expliquen en cada sección, detallando las diferentes pruebas y ajustes realizados.

4.1 Análisis de vídeos pregrabados

Como se ha detallado en el Especificaciones y restricciones de diseño, la cantidad de vídeos disponibles es limitada y no puede ser ampliada. Por esta razón, la gestión adecuada de este recurso ha sido de gran importancia.

La grabación del *Net Test* se llevó a cabo por los técnicos Juan Enrique Barrios Sánchez y Almudena Gallego Fernández, bajo supervisión de los investigadores Álvaro De la Llave-Propín y Morris Villarroel Robinson, los cuales terminaron realizando el experimento en cuatro ocasiones diferentes [8]. En cada repetición, se disponía de nueve jaulas, cada una con diez truchas. Todos los especímenes de una misma jaula se grabaron en un único vídeo de forma secuencial, lo cual significa que, primero se graba simultáneamente la pareja de peces 1-2, luego la 3-4, y así sucesivamente. La duración del experimento para cada pareja es siempre constante, siendo de 15 segundos aproximados.

Para un manejo más adecuado de las grabaciones, se extrajeron de los vídeos originales de cada jaula, un vídeo de cada pareja, obteniendo así un vídeo individual para cada experimento. Teniendo en cuenta que hay 4 versiones del experimento, con 9 jaulas y 10 peces en cada una, se obtuvieron un total de 180 vídeos. Una sección de uno de los experimentos se puede observar en la Figura 16: Fotogramas de un Segundo del *Net Test*. Ordenadas de izquierda a derecha y de arriba abajo, se muestran veinticinco imágenes, las cuales, debido a la velocidad de grabación de la cámara de 25 fotogramas por segundo, equivalen a un segundo de grabación del experimento. Se puede observar cómo las truchas, en algunos instantes, se mueven a velocidades elevadísimas, de forma casi aleatoria y adoptando posiciones muy variadas, o directamente no se mueven.

4.2 Selección de fotogramas de interés

Se dispone de 180 vídeos, cada uno con una duración media de aproximadamente 15 segundos, grabados a una velocidad de 25 fotogramas por segundo. En cada instante, los vídeos muestran dos truchas diferentes, lo que resulta en un total de 135 000 imágenes de truchas individuales. Dado el gran volumen de imágenes disponibles, es necesario desarrollar una estrategia para reducir este número a una cantidad más manejable, permitiendo un etiquetado manual con un número realista de imágenes.

De las cuatro repeticiones del experimento, se seleccionó la que presentaba las mejores condiciones visuales. Esto significa que se eligió la repetición con la iluminación más homogénea y la menor cantidad de peces saliendo del encuadre de la cámara. Las demás repeticiones se utilizarán más adelante, para validar el modelo final. Esta reducción inicial disminuyó el número de imágenes de 135000 a 33750 aproximadamente. Adicionalmente se descartaron los vídeos que no se encontraban en formato mp4 debido a las restricciones de diseño impuestas en el Especificaciones y restricciones de diseño.

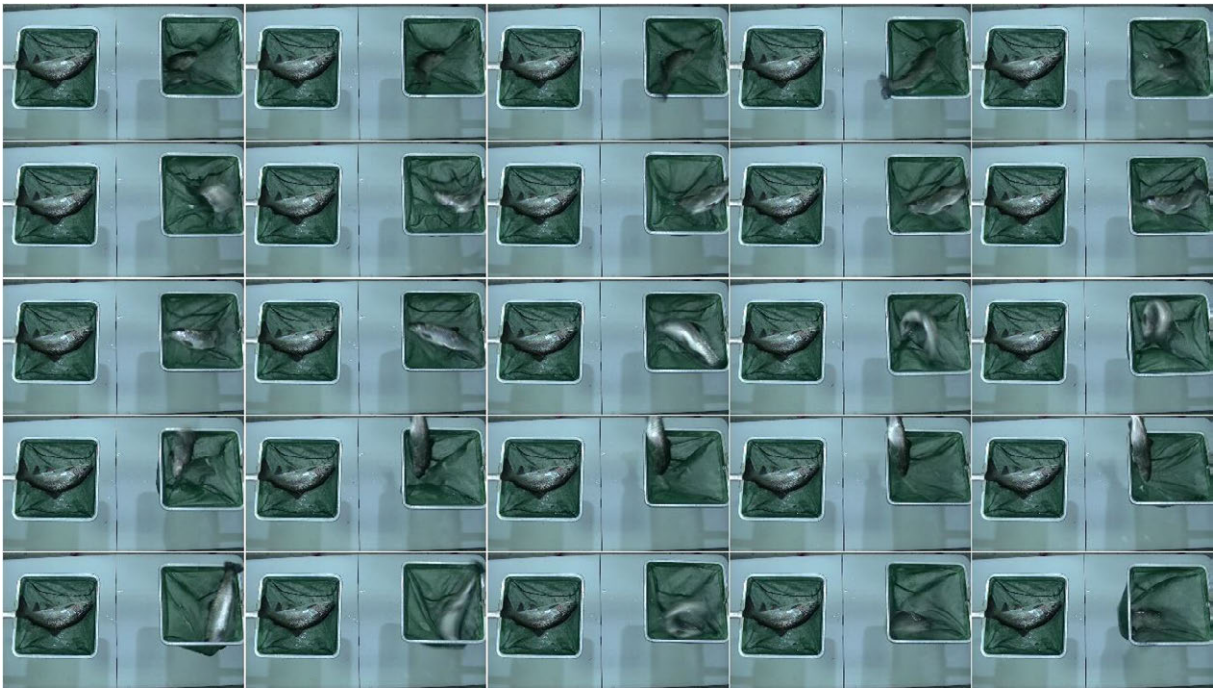

Figura 16: Fotogramas de un Segundo del *Net Test*

A continuación, se intentó mejorar la calidad de los vídeos reduciendo los borrones causados por el rápido movimiento de las truchas. Se desarrollaron varios scripts para preprocesar los vídeos, probando métodos como filtros gaussianos inversos, filtros de suavizado adaptativo y filtros de Wiener y Richardson-Lucy. Sin embargo, estos enfoques no produjeron mejoras significativas, por lo que se descartó la idea de realizar un preprocesamiento adicional.

Para crear un buen *dataset*, siguiendo las pautas descritas en el 2.7.1, se seleccionaron fotogramas que ofrecieran información nueva, clara y que no presentaran altos niveles de borrosidad, lo que dificultaría el etiquetado manual. Se desarrolló un script que, a partir de un vídeo, divide cada fotograma en dos, separando así los dos especímenes en fotografías distintas. El script también detecta si un fotograma está borroso y, en caso contrario, verifica si hay un cambio significativo en comparación con el fotograma anterior. Si ambas condiciones se cumplen, el medio fotograma se guarda para su posterior etiquetado. La finalidad de este script es que por ejemplo en el caso de la Figura 16: Fotogramas de un Segundo del *Net Test*, no se almacenará más de un fotograma de la trucha de la izquierda, ya que tener la misma información repetida, no aportaría información adicional.

Para evaluar el nivel de borrosidad, se aplicó un filtro diferencial de segundo orden mediante el operador laplaciano a cada mitad de fotograma, lo que permite detectar bordes. Luego, se calcula la varianza del operador, que actúa como un indicador del nivel de borrosidad. Una imagen bien enfocada tendrá más bordes definidos y, por lo tanto, una mayor varianza en la respuesta del operador laplaciano. Si la varianza es mayor que un umbral predefinido, el medio fotograma se considera válido para el etiquetado debido a que es realista encontrar los *keypoints* en dicha imagen.

Para detectar cambios significativos entre fotogramas, se utiliza la diferencia absoluta entre el cuadro actual y el anterior. Esto implica comparar cada píxel en términos de su código RGB con su estado anterior. Si la media de estas diferencias supera un umbral predefinido, se considera que ha habido un cambio significativo y por ende el medio fotograma es válido para el etiquetado ya que aporta información nueva significativa.

Después de aplicar este script a las 33 750 imágenes en bruto que teníamos, se terminó dejando en 732 imágenes de calidad suficiente para el etiquetado.

4.3 Creación de una etiqueta propia

La creación de una etiqueta propia es un paso fundamental en el desarrollo de cualquier proyecto de visión por ordenador, ya que las etiquetas proporcionan la referencia necesaria para que el modelo pueda aprender a identificar y clasificar objetos de interés.

Teniendo en cuenta el objetivo final del modelo: cuantificar el movimiento, se probaron diferentes definiciones de los *keypoints.* Inicialmente, se definieron un total de 12 *keypoints* distribuidos de la manera mostrada en la Figura 17: Etiqueta Preliminar de una Trucha. Al crear el esqueleto uniendo dichos puntos, se podía detectar si las líneas se intersecaban, indicando que la trucha se había retorcido, permitiendo contabilizar los coletazos. También se pretendía determinar la orientación de la trucha calculando el área del esqueleto. Sin embargo, esta idea fue descartada tras diversas pruebas, ya que era difícil identificar puntos como el ojo en situaciones no ideales, es decir, cuando el pez no estaba paralelo al plano de grabación.

Además, este enfoque era similar a segmentar el pez, haciendo innecesario el uso de un modelo de estimación de pose, que es mucho más complejo que uno de segmentación.

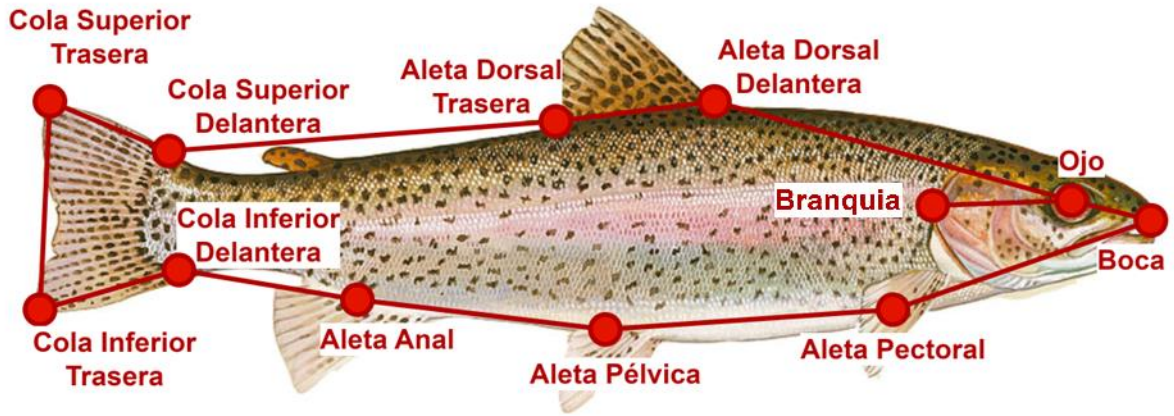

Figura 17: Etiqueta Preliminar de una Trucha

Se fueron perfeccionando las etiquetas hasta llegar a la versión mostrada en la Figura 18: Etiqueta Definitiva de una Trucha. Para alcanzar este resultado, se identificaron los puntos que eran más fácilmente reconocibles en diversas situaciones, independientemente del plano en el que se encontrara el espécimen respecto a la cámara. Los puntos de interés clave para cuantificar el movimiento son: Cola Media, Branquia y punto G (centro de gravedad). Los otros puntos son auxiliares y ayudan al modelo a posicionar correctamente los puntos de verdadero interés.

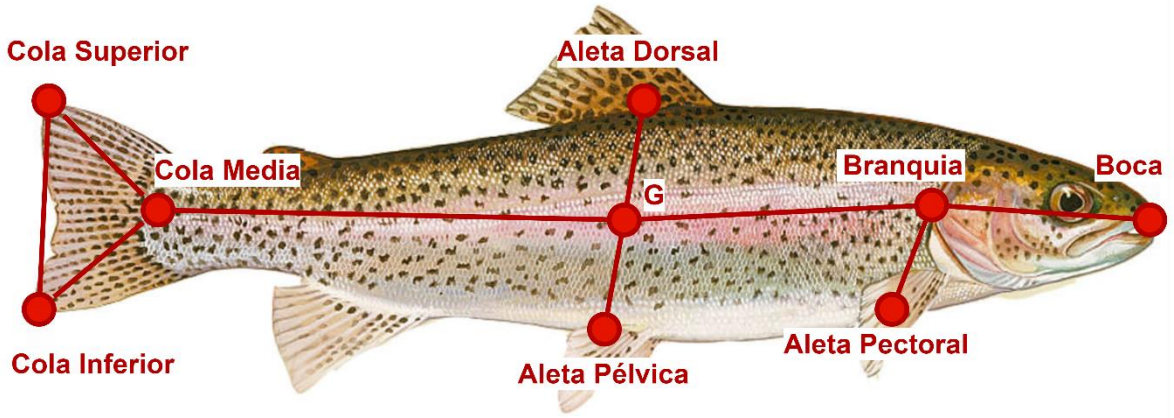

Figura 18: Etiqueta Definitiva de una Trucha

Es importante destacar que tanto el punto G, como el punto Cola Media y el punto Branquia no tienen una ubicación exacta fija. El punto G se aproxima como el punto medio entre la línea que une la Aleta Dorsal con la Aleta Pélvica, el punto Cola Media se define como el punto medio entre el final del tronco del animal y el inicio de la cola, por último, el punto Branquia se ubica en el punto central de la branquia.

4.4 Etiquetado de imágenes

En las primeras iteraciones del proyecto se etiquetaron 150 imágenes, aunque finalmente se terminaron por etiquetar las 732 imágenes. En iteraciones posteriores, con el fin de eliminar sesgos subjetivos a la hora de realizar las estimaciones en los *keypoints* ocultos, todas las etiquetas fueron revisadas por 4 personas adicionales. En dichas revisiones se comprobaron y mejoraron algunos *keypoints*, además se eliminaron imágenes borrosas con *keypoints* difícilmente estimables e imágenes imposibles que habían pasado el filtro del script descrito en el 4.2. Con estas revisiones finales, se terminó reduciendo el *dataset* a 526 imágenes.

Una vez se etiquetan todas las imágenes, se debe exportar dicha información para más tarde usarla en el proceso de entrenamiento. La herramienta de CVAT exportaba el etiquetado de todas las imágenes en un único fichero de tipo JSON siguiendo el formato de COCO. Para poder ser entrenado, el modelo de YOLOv8 requiere que las anotaciones estén en su formato propio, es por ello por lo que se tuvo que desarrollar un script que transformara de un formato a otro, siguiendo todas las características descritas en el 2.7.3.2. Más tarde se encontró una versión más eficiente de este script de traducción, disponible en github. Aun así, no se usó y fue de gran utilidad el desarrollar el script ya que esto obligó a entender perfectamente el funcionamiento de los *datasets* y los formatos disponibles.

4.5 *Data Augmentation*

Como se ha visto en el 2.7.3.3, la utilidad de la *data augmentation* radica en su capacidad para generar nuevas imágenes a partir de una imagen original mediante la aplicación de una o varias técnicas, como se muestra en la Figura 15. Este enfoque permite ampliar virtualmente el *dataset*, mejorando la capacidad del modelo para generalizar. Además, existen muchas librerías que no solo modifican las imágenes, sino que también crean las anotaciones correspondientes a las nuevas imágenes, eliminando así la necesidad de volver a etiquetarlas.

En este caso, se ha utilizado la librería de Python llamada Albumentations[4], la cual está altamente documentada en su página oficial. Aplicando un conjunto de transformaciones que ofrece esta librería, se crearon siete nuevas imágenes por cada imagen original de trucha. Cada composición incluye diferentes modificaciones con el objetivo de que las nuevas imágenes no sean iguales para todas las imágenes originales. Se añadieron factores de aleatoriedad mediante probabilidades o factores específicos para asegurar la diversidad en las imágenes generadas. La Figura 19: Estrategias de *Data Augmentation* Aplicadas muestra un collage de diferentes imágenes originales y sus modificaciones generadas.

[4] "albumentations - albumentations 1.1.0 documentation," [en línea]. Disponible en: https://albumentations.readthedocs.io/en/latest/. [Accedido: 24-jun-2024].

Estas composiciones se realizaron siguiendo una estrategia concreta basada en observaciones durante diferentes entrenamientos. Por ejemplo, se notó que, cuando la trucha estaba en posición vertical, mostrando su parte oscura, y la red era de color oscuro, el modelo tendía a confundirse. Para abordar este problema y otros más que se encontraron, se amplió el *dataset* siguiendo las siguientes estrategias: Para simular diferentes condiciones de iluminación, se crearon las composiciones uno y dos. Las composiciones tres y cuatro se centraron en proporcionar más perspectivas de lo que es una trucha, enriqueciendo así el conjunto de datos con diversas vistas y con diferentes colores de los entornos de grabación. Las composiciones cinco y seis se centraron en generar imágenes en blanco y negro, con poca iluminación y mucha borrosidad, enseñando así al modelo que el color no es un factor determinante. Para hacer el modelo robusto frente a ruido, como los reflejos, se diseñó la composición siete. En total se terminó aumentando el *dataset* a 4.200 imágenes.

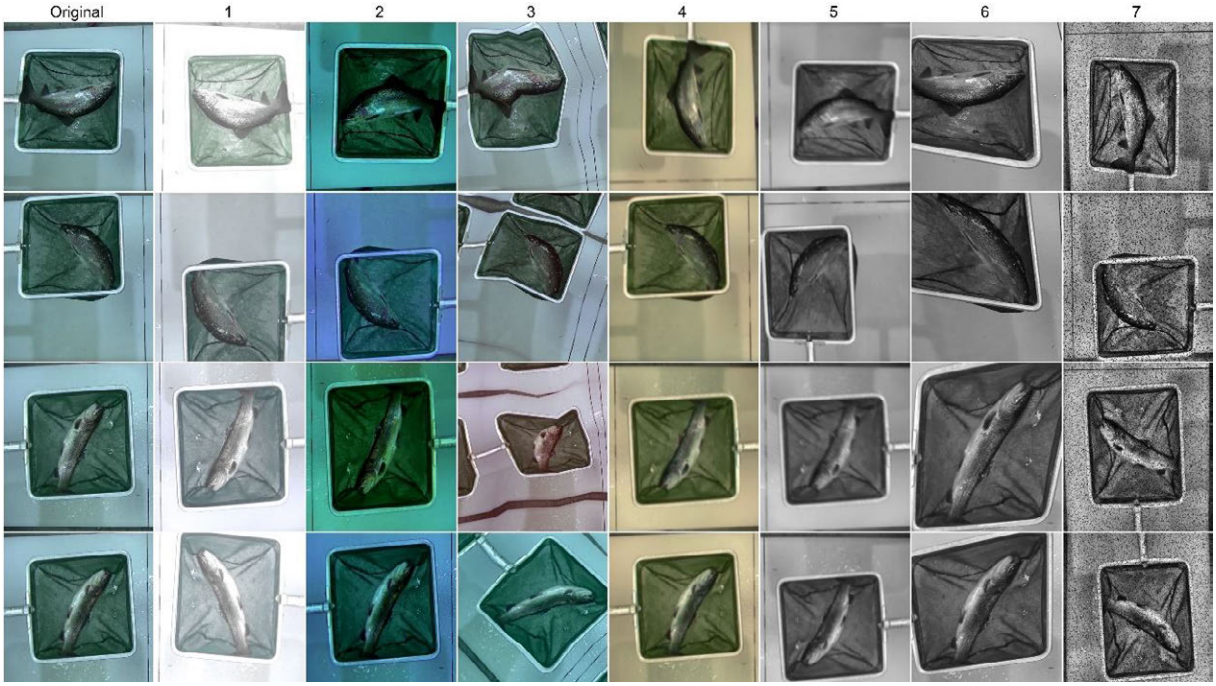

Figura 19: Estrategias de *Data Augmentation* Aplicadas

4.6 Separación de *Datasets*

En los primeros compases del desarrollo del proyecto se siguieron estrictamente los porcentajes recomendados mencionados en el 2.7.4. Sin embargo, a medida que aumentó el número total de imágenes etiquetadas, se redujo el porcentaje de imágenes destinadas al conjunto de prueba. Esta decisión se tomó porque para evaluar correctamente el funcionamiento de un modelo no es necesario comprobarlo en 400 imágenes diferentes, que sería el tamaño del conjunto de prueba de haber seguido los porcentajes iniciales. Se determinó que 50 imágenes son suficientes para este propósito, garantizando su diversidad.

4.7 Entrenamiento del modelo

Se llevaron a cabo un total de siete entrenamientos, incluyendo tanto los ensayos preliminares como los entrenamientos que resultaron realmente útiles. Son destacables y por tanto son los que se van a analizar, tres de ellos:

4.7.1 Entrenamiento Conceptual

Este fue el primer entrenamiento realizado para comprobar la viabilidad de cuantificar el movimiento con el enfoque elegido, utilizando un modelo de visión por ordenador centrado en la estimación de pose. En este entrenamiento, no se hicieron ajustes manuales más allá de definir el número de épocas, fijado en 250, utilizando la configuración por defecto que ofrece Ultralytics para el modelo YOLOv8n-pose.

Se utilizó un *dataset* básico de 30 imágenes, divididas en 25 imágenes para el conjunto de entrenamiento y 5 para el conjunto de validación. En este momento, no se destinó un conjunto de datos de prueba, ya que el objetivo principal era evaluar la viabilidad del enfoque y no la mejora de eficiencia del modelo.

El análisis de los resultados, mostrados en la Figura 20: Métricas del Primer Entrenamiento del Modelo, demostró que el enfoque elegido es viable para cuantificar el movimiento mediante la detección y estimación de pose. Las pérdidas de entrenamiento y validación disminuyeron, indicando una mejora en la tarea específica del modelo. Las métricas de *recall y precision* mostraron valores altos, sugiriendo que el modelo YOLOv8n-pose es capaz de detectar y clasificar *keypoints* con buena precisión.

Analizar los resultados en mayor profundidad carece de sentido, debido a la naturaleza preliminar y simplificada de este experimento. La limitada cantidad de datos y la falta de optimización detallada impiden obtener conclusiones definitivas sobre el rendimiento y la robustez del modelo.

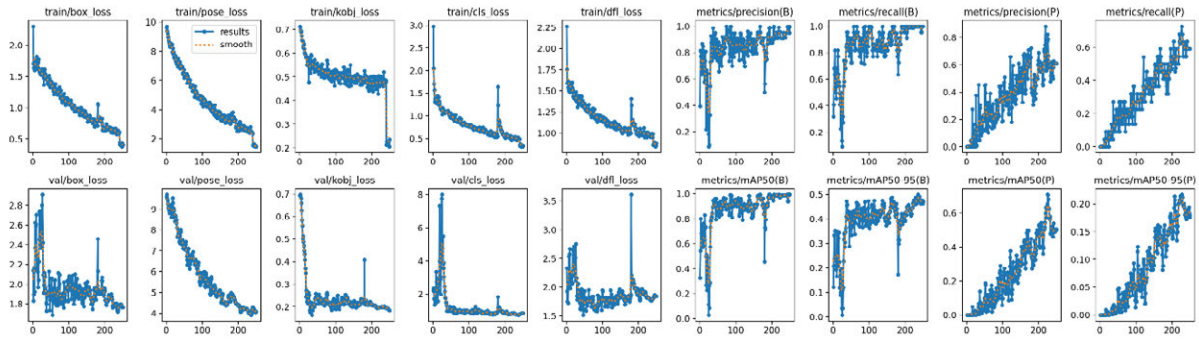

Figura 20: Métricas del Primer Entrenamiento del Modelo

4.7.2 Entrenamiento básico

4.7.2.1 Estrategia

El segundo entrenamiento se diseñó para evaluar y optimizar el rendimiento del modelo YOLOv8n-pose utilizando un *dataset* ampliado, ajustando parámetros críticos para mejorar la precisión y la consistencia en la estimación de poses. La configuración del entrenamiento incluyó la implementación interna de *data augmentation* para aumentar la diversidad y calidad del conjunto de datos, lo que permitió al modelo manejar un mayor volumen de datos y mejorar su capacidad de generalización. Con estas mejoras, se buscó maximizar la precisión y consistencia en la detección y estimación de poses.

La arquitectura empleada fue el modelo YOLOv8n-pose, configurado para entrenar durante 300 épocas. La tasa de aprendizaje fue ajustada y se definió un tamaño de *batch* de 16, que se refiere al número de muestras procesadas antes de actualizar los parámetros del modelo. Este ajuste es crucial para la eficiencia del entrenamiento y la estabilidad del aprendizaje, es decir para evitar esa variación tan extrema que se apreciaba en las gráficas del primer entrenamiento.

Se utilizó un *dataset* más amplio y balanceado en cuanto a orientación y disposición de las truchas. Las imágenes etiquetadas con el que contaba el *dataset* era de 150 en total, de las cuales 105 se destinaron al entrenamiento, 30 a la validación y 15 a las pruebas. En este caso todas las imágenes que componían el *dataset* eran originales y ninguna de ellas había sido generada por *data augmentation.*

4.7.2.2 Resultados

Como se puede observar en la Figura 21: Métricas de Perdida, el entrenamiento seguía una buena tendencia de aprendizaje. Se puede incluso observar que entorno a la época 250, hay una mejora repentina en todas las estadísticas de pérdidas y métricas.

Las pérdidas de entrenamiento y validación observadas en la Figura 21: Métricas de Perdida muestran una tendencia decreciente continua. No hay indicaciones claras de que las pérdidas se hayan estabilizado completamente, lo que sugiere que el modelo aún podría beneficiarse de más entrenamiento. Además se puede observar también que la cercanía entre las pérdidas de entrenamiento y validación sugiere que no hay un sobreajuste significativo, lo cual significa que el modelo sigue generalizando bien y todavía tiene espacio para mejorar antes de llegar a la sobre especialización del modelo para este *dataset.*

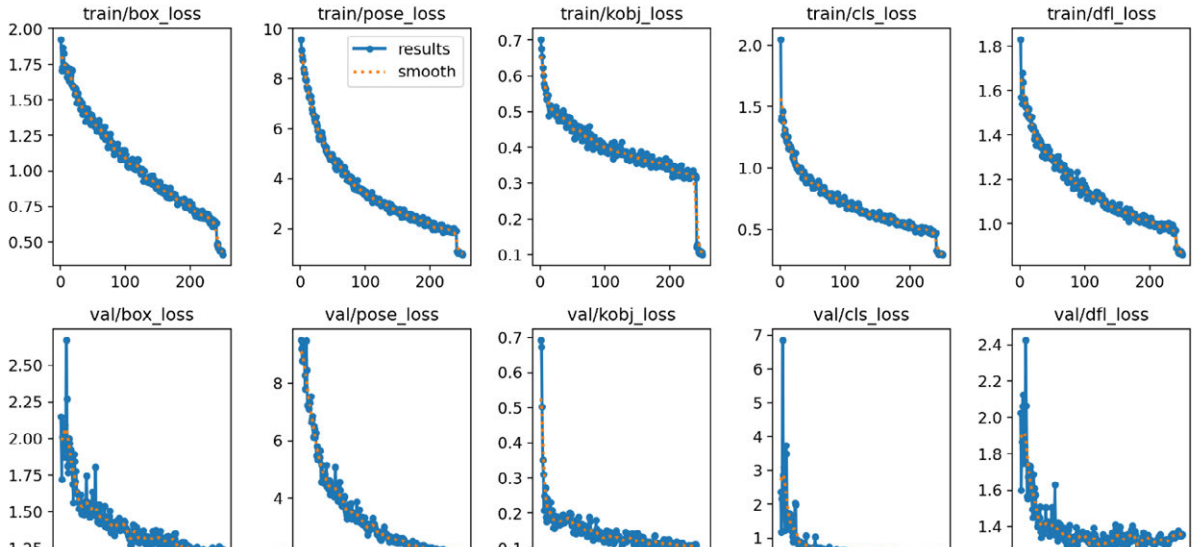

Figura 21: Métricas de Perdida Entrenamiento Básico

Las métricas de la Figura 22 muestran una mejora constante con margen de mejora. Las estadísticas de las bounding boxes son generalmente mejores que las de la pose. Esto indica que, aunque el modelo puede detectar con precisión la presencia y posición de una trucha, tiene más dificultades para identificar los *keypoints* exactos de la pose. En cuanto a las bounding boxes, el margen de mejora es principalmente en la estadística de `AP50-95`, sugiriendo que el modelo detecta bien la trucha y su ubicación general, pero tiene problemas con las predicciones más precisas en diferentes escalas y aspectos.

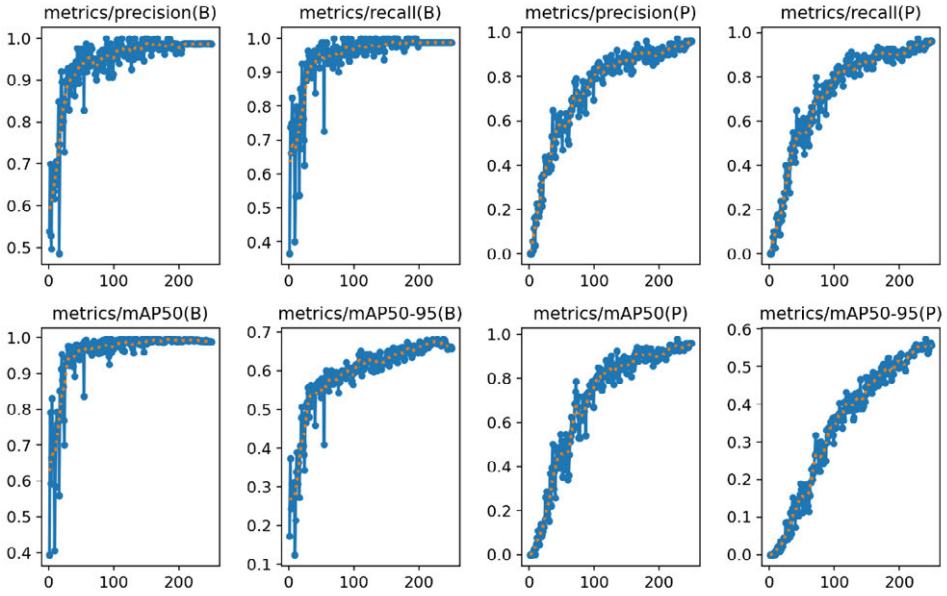

Figura 22: Métricas de Rendimiento Entrenamiento Básico

En referencia a las poses, se observa un potencial de mejora muy significativo en varias métricas, particularmente en `mAP50-95(P)`. Esto sugiere que el modelo tiene dificultades para detectar con alta precisión los puntos clave de la postura de la trucha en diferentes escalas y niveles de detalle. Aunque las métricas de *recall y precision* para las poses muestran una mejora constante y un resultado muy aceptable, no alcanzan valores tan buenos como con las *bounding boxes*. Por lo tanto, hay oportunidades significativas para mejorar la precisión en la detección de poses mediante ajustes en el modelo y técnicas de entrenamiento avanzadas.

4.7.2.3 Verificación del modelo

En la Figura 23 se muestra la inferencia realizada con el modelo entrenado sobre un vídeo de un experimento del *Net Test* que el modelo nunca había visto. Los fotogramas de este vídeo no están presentes en el conjunto de datos utilizado para el entrenamiento. Como en ocasiones anteriores, los fotogramas se presentan en orden de izquierda a derecha y de arriba hacia abajo. Además, el nivel de confianza con el que se detecta cada punto está representado por el color del keypoint: si es de color puramente rojo, el nivel de confianza es del 50%, y si es de color puramente verde, el nivel de confianza es del 100%.

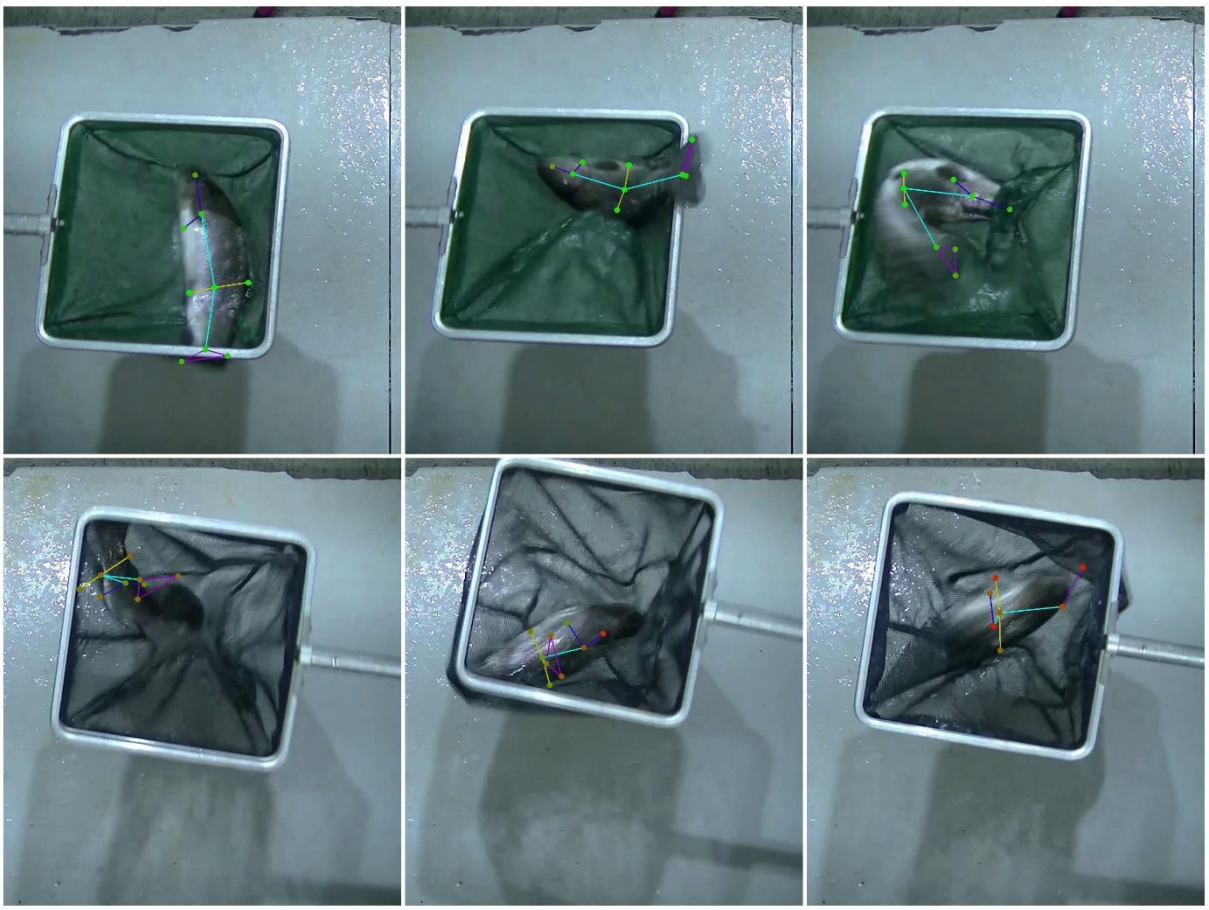

Figura 23: Fotogramas de Vídeo Inferido Entrenamiento Básico

Como se puede observar, por lo general, el modelo funciona adecuadamente, aunque presenta grandes dificultades a la hora de detectar una trucha cuando esta está sobre una red de color oscuro. Es por esto mismo por lo que se desarrolló la estrategia de *data augmentation* descrita en el 4.5.

4.7.3 Entrenamiento Avanzado

4.7.3.1 Estrategia

Este entrenamiento se diseñó con un enfoque orientado a maximizar el rendimiento del modelo, siempre teniendo en cuenta las limitaciones de la capacidad computacional disponible y el tiempo. Se utilizó un *dataset* ampliado que incluía tanto imágenes originales como imágenes generadas mediante técnicas de *data augmentation*, como se muestra en el 4.5. Este enfoque permitió al modelo manejar una gama más amplia de variaciones y mejorar su capacidad de generalización.

No se reentrenó el modelo del 4.7.2. En su lugar, se partió nuevamente de la fase preentrenada facilitada por Ultralytics, utilizando el modelo yolov8n-pose.py. Esta decisión se tomó debido a que el modelo entrenado anteriormente presentaba problemas significativos en las detecciones en determinadas situaciones, lo cual afectaba negativamente su rendimiento y precisión. Si se hubiera continuado con dicho modelo, no se podría asegurar que el proceso de entrenamiento fuera suficiente para corregir estos problemas críticos. Reiniciar el entrenamiento desde la fase preentrenada garantizó una base más sólida y confiable para las nuevas iteraciones de entrenamiento.

El modelo fue configurado para entrenar durante 950 épocas, el número máximo de épocas posible dentro del periodo delimitado para el desarrollo de este TFG. El entrenamiento se realizó de manera ininterrumpida durante 35 días. Se estableció una tasa de aprendizaje dinámica que se ajustaba en función del progreso del entrenamiento. El tamaño del *batch* se mantuvo en 16 para asegurar la estabilidad del aprendizaje y optimizar el uso de recursos computacionales. Este ajuste permitió una mayor exploración del espacio de parámetros y una mejor convergencia del modelo. El *dataset* utilizado en este entrenamiento constaba de 4200 imágenes, de las cuales 3400 se destinaron al entrenamiento, 750 a la validación y 50 a las pruebas.

4.7.3.2 Resultados

La Figura 24: Métricas de Pérdida Entrenamiento Avanzado muestra las pérdidas de entrenamiento y validación a lo largo de las épocas. Se observa una clara tendencia decreciente en las pérdidas del entrenamiento, con una notable estabilidad alcanzada después de aproximadamente 800 épocas. Si bien es cierto que según la línea de tendencia indica, es posible seguir mejorando, los valores de estas métricas son muy positivos.

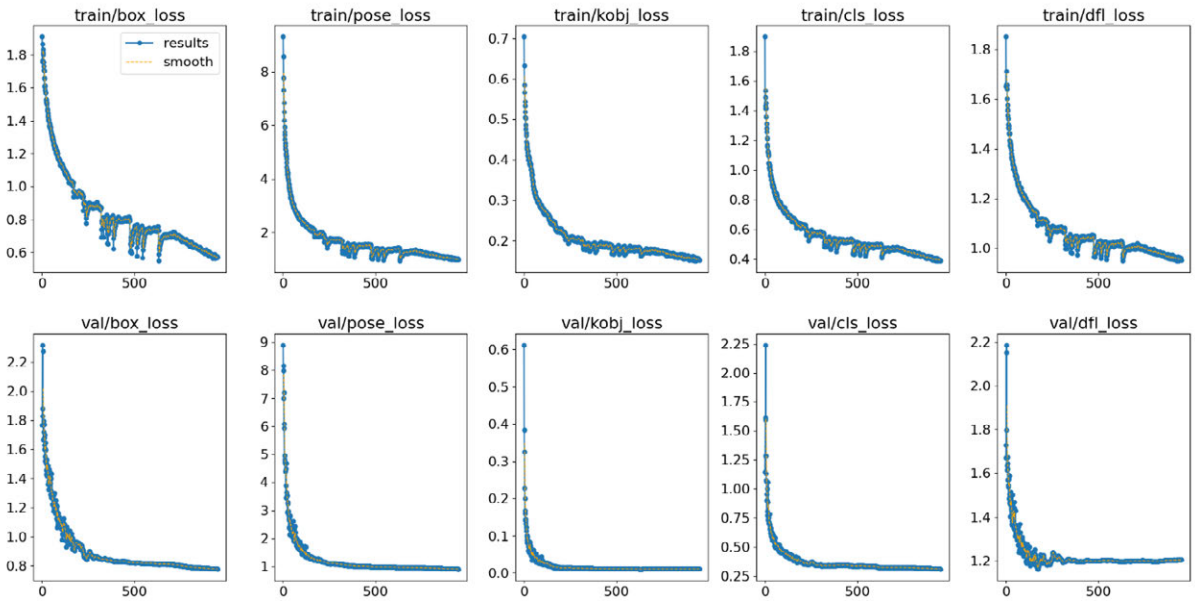

Figura 24: Métricas de Pérdida Entrenamiento Avanzado

Las métricas de rendimiento presentadas en la Figura 25 indican una mejora continua en *recall* y *precision* tanto para las bounding boxes como para los *keypoints* de poses. Las métricas de *recall y precision* para las poses, aunque mejoradas, todavía muestran un margen de mejora en comparación con las métricas de las bounding boxes, aunque este valor menor es normal debido a que la detección de una *bounding box* siempre será más compleja que en la detección de todos los *keypoints*.

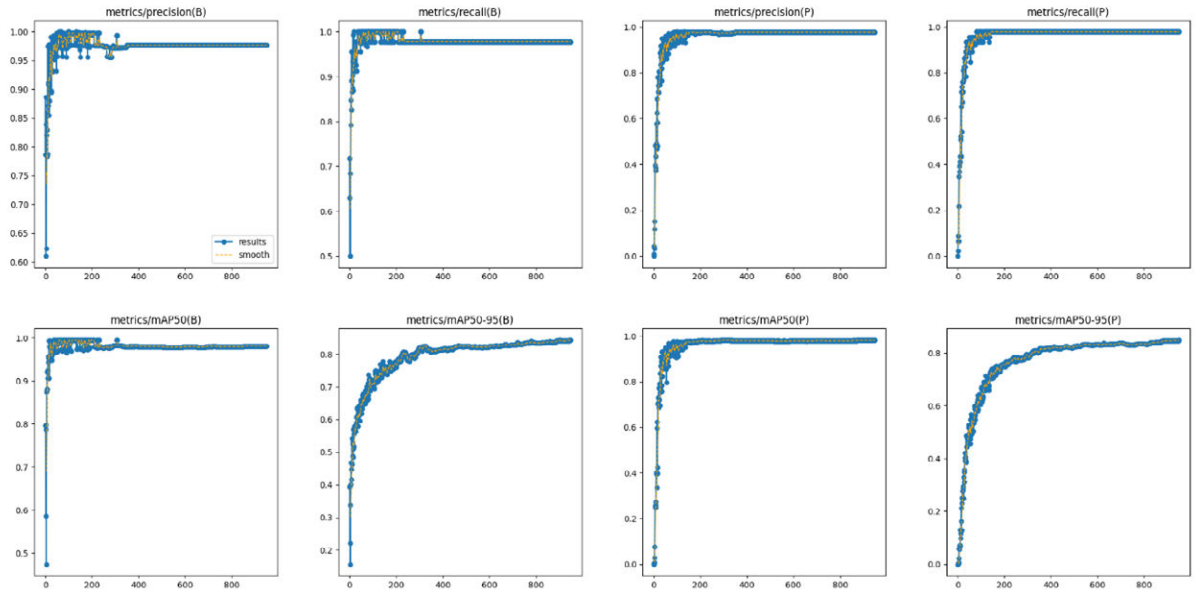

Figura 25: Métricas de Rendimiento Entrenamiento Avanzado

4.7.3.3 Verificación del modelo

La verificación del modelo, como se muestra en la Figura 26, se realizó un video cuyos fotogramas no habían sido utilizados para ningún *dataset* de entrenamiento. Concretamente los fotogramas mostrados en la Figura 26 son los mismos mostrados en la Figura 23, para poder facilitar la comparación de inferencias, entre diferentes modelos. Los resultados de inferencia demuestran que el modelo ha mejorado en la detección de truchas y sus poses, incluso en condiciones variadas de iluminación y fondo. La implementación de *data augmentation* ha mitigado los problemas encontrados en el entrenamiento básico del 4.7.2 en su totalidad, quedando un modelo que, para la gran mayoría de casos es prácticamente perfecto.

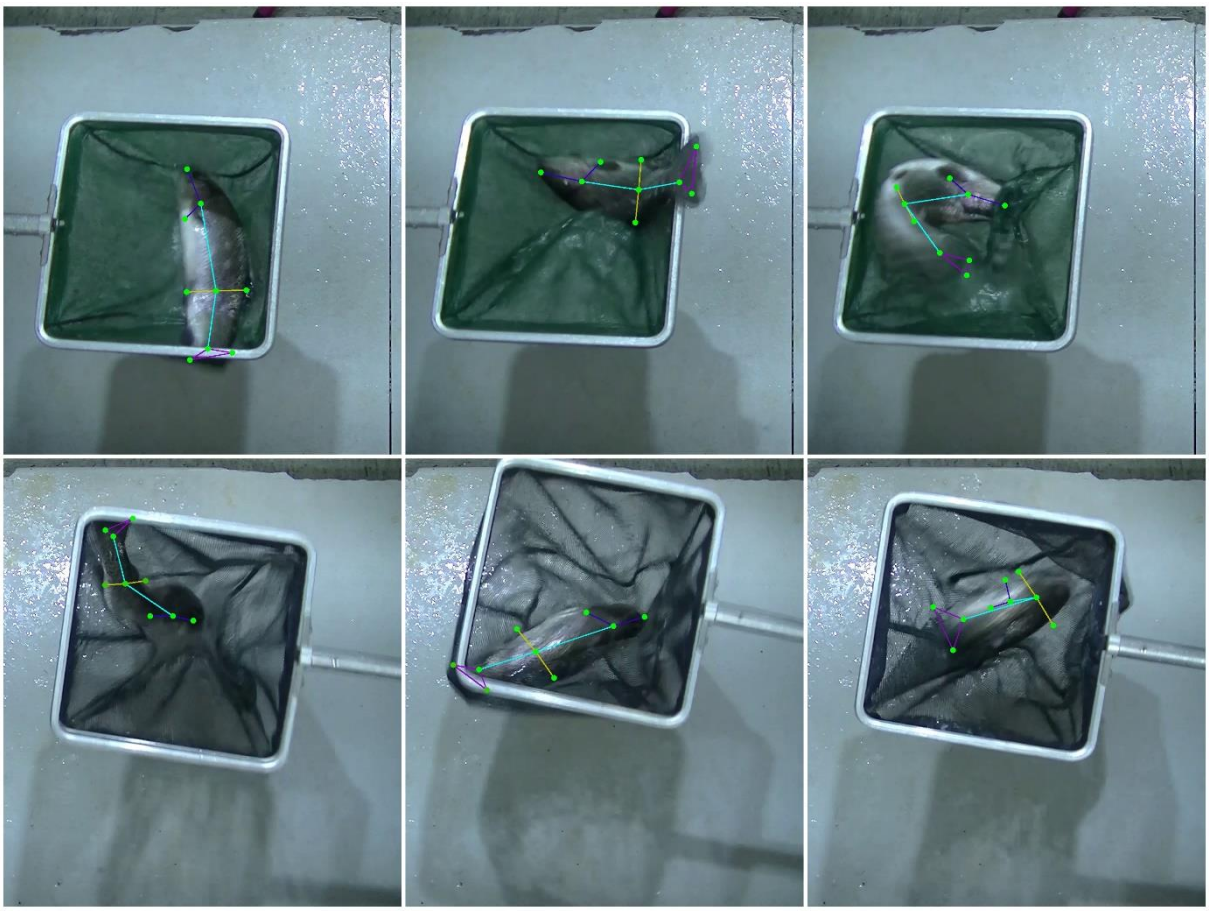

Figura 26: Fotogramas de Vídeo Inferido Entrenamiento Avanzado

En resumen, este entrenamiento ha mostrado mejoras muy significativas en la precisión y consistencia de la detección y estimación de poses, acercando e incluso superando, el desempeño previsto del modelo durante los objetivos del proyecto.

4.8 Generación de resultados

El objetivo primordial de este PFG es la cuantificación de movimiento durante el experimento del *Net Test*. Para poder medir este movimiento lo que se va a evaluar es el ángulo formado por las dos secciones de la línea principal del modelo presentado en la segunda etiqueta mostrada en el 4.3. Es decir, el ángulo formado por la sección frontal de la trucha (segmento que une los puntos “Branquia” y “G”) con la sección caudal (segmento que une los puntos “G” y “Cola Media”). El estado ideal de reposo de la trucha cuando esta se encuentra totalmente estirada es de 180 grados.

Adicionalmente, una restricción de diseño es que el público objetivo que interpretará los datos de salida está compuesto principalmente por técnicos especializados en producción animal, quienes no tienen por qué poseer conocimientos de inteligencia artificial. Teniendo esto en cuenta, los datos de salida se han diseñado para que sean fácilmente interpretables sin necesidad de tener un conocimiento detallado de IA. Por ello, se han suprimido las métricas de rendimiento en favor de mostrar lo realmente importante para el proyecto en el que se engloba este PFG, el movimiento.

Todos los resultados generados para cada vídeo de entrada se guardan en un único archivo comprimido en formato zip, cuyo nombre coincide con el del vídeo original. Este método se utiliza para optimizar el uso del almacenamiento, asegurando que los datos ocupen menos espacio en el disco y permitiendo una gestión más eficiente de los archivos. Además, comprimir todos los resultados relacionados con un vídeo en un solo archivo facilita su distribución posterior, ya que se puede transferir, compartir o archivar de manera más sencilla sin perder la organización de los datos, ya que todo lo relacionado con un mismo experimento se encuentra en un solo archivo comprimido, evitando la dispersión de archivos y posibles pérdidas de datos.

Los resultados que se incluyen en dicho archivo comprimido son:

- **Tres vídeos:** El vídeo original y dos vídeos en los que las truchas han sido analizadas por el modelo. Uno de estos vídeos se presenta a velocidad normal, es decir, a 25 fotogramas por segundo, y el otro a una velocidad reducida de 2 fotogramas por segundo. Esta variación permite una comparación rápida para verificar la precisión del análisis y una supervisión detallada casi fotograma a fotograma. En la Figura 27 se muestra un fotograma de uno de los vídeos de salida. El esqueleto de la trucha, explicado en el capítulo 4.3, se dibuja con diferentes colores para denotar las distintas partes: azul oscuro para la cabeza, amarillo para la parte central, y morado para la cola. La sección del esqueleto que une estas tres partes se pinta de azul claro. La confianza de cada keypoint se indica con un degradado de color desde verde (100% de confianza) hasta rojo (50% o menos). Los *keypoints* importantes para medir el movimiento, "Branquia", "G", y "Cola Media", tienen su valor de confianza mínima mostrado en la parte inferior de la imagen, junto con el ángulo de la línea del esqueleto que une estos puntos. También se indica el fotograma actual del vídeo.

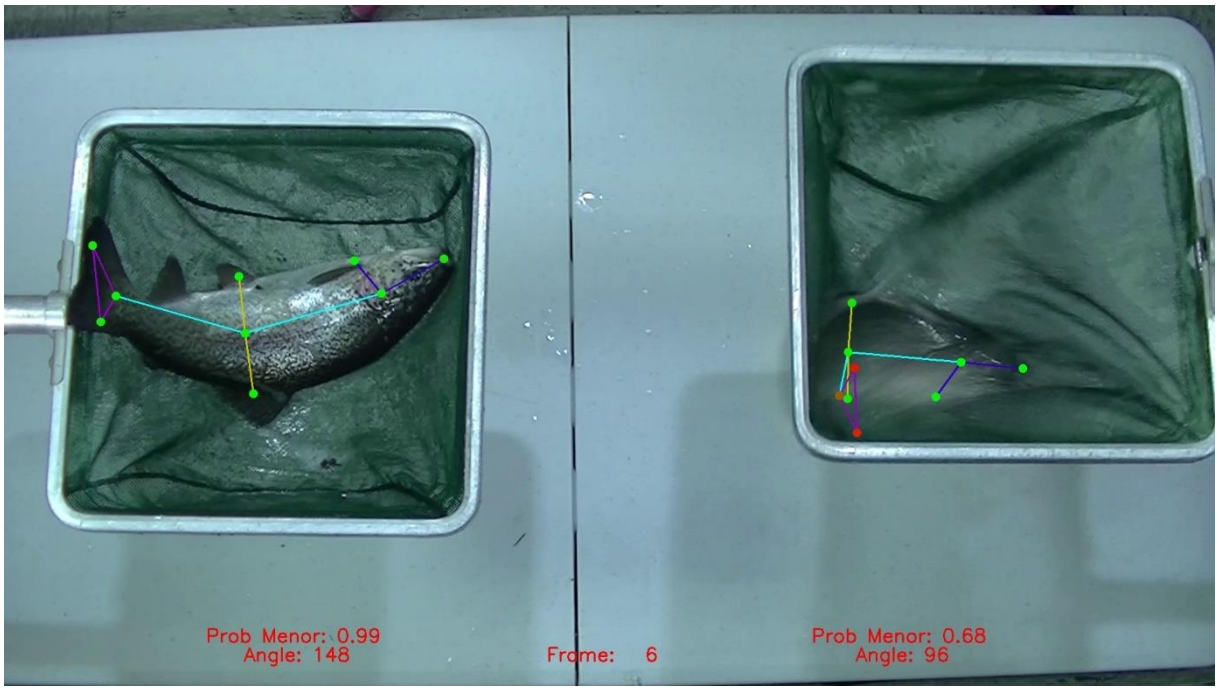

Figura 27: Fotograma de un Video de Salida del Sistema

- **Un fichero de texto** que contiene los datos estadísticos de amplitud y confianza en un formato amigable para la lectura humana. Este fichero permite verificar rápidamente todos los números importantes en un solo lugar. Una pequeña parte de este fichero puede verse en la Figura 28.

Frame	Izquierda		Derecha	
No	Prob	Ang	Prob	Ang
1	0.99	148	0.97	174
2	0.99	147	0.97	180
3	0.99	148	0.99	175
4	0.99	148	1.00	158
5	0.99	148	0.99	48
6	0.99	148	0.68	96
7	0.99	149	0.94	45
8	0.99	149	0.99	38
9	0.99	149	0.98	39
10	0.99	149	0.98	69
11	0.99	149	0.98	127
12	0.99	149	0.99	153

Figura 28: Fichero Legible de Salida del Sistema

- **Un fichero npy**[5] que contiene los datos estadísticos de amplitud y confianza en un formato orientado a la programación. Este fichero es útil para realizar nuevos análisis sin tener que volver a inferir los vídeos, ahorrando tiempo y recursos.

[5] Este formato es específico de la librería de python NumPy y está diseñado para almacenar arrays permitiendo la lectura y escritura rápida, así como la conservación de la estructura de los datos.

- **Dos imágenes**, una para la trucha izquierda y otra para la de la derecha. Estas imágenes representan la salida más visual y rápida de analizar en la que, finalmente, se muestra la cuantificación de movimiento. Como se puede ver la Figura 29, estas imágenes muestran un gráfico representando la evolución de dichos valores de amplitud de las truchas a lo largo del vídeo y la derivada de estos valores. El gráfico de la derivada es especialmente importante porque los picos en este gráfico indican un coletazo. Por otro lado, los gráficos de amplitud absoluta son menos relevantes debido al posible ruido causado por la posición de descanso del pez y el movimiento de la red. Ambos gráficos incluyen un gradiente de colores que va del rojo al verde, indicando el valor mínimo de confianza de los tres *keypoints* más importantes, de la misma manera que se hace en los vídeos inferidos explicados anteriormente.

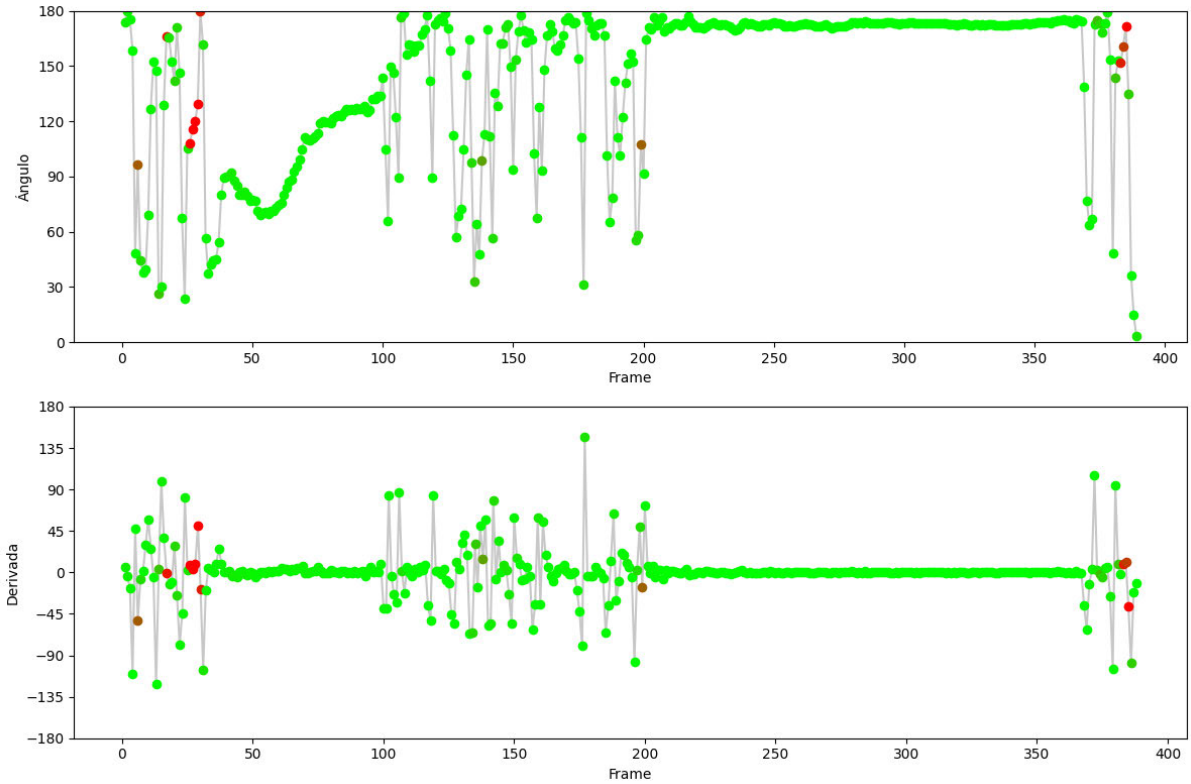

Figura 29: Gráficos de Salida del Sistema

5. Resultados

Dado que el objetivo de este PFG no es replicar los resultados del análisis manual de los investigadores, no tiene sentido hacer una comparación directa uno a uno con los datos obtenidos manualmente. Sin embargo, ya se adelanta que existe un alto grado de correlación entre los momentos en los que los investigadores del GAP anotaban la presencia de movimiento en los especímenes y los instantes en los que el modelo detecta dicho movimiento.

5.1 Rendimiento del modelo

Para validar el correcto funcionamiento del modelo, se hará un análisis similar a los descritos en el 4.7.2 y el 4.7.3, utilizando el conjunto de datos de prueba. Como se ha explicado anteriormente, estas imágenes y sus etiquetas no han sido vistas por el modelo durante el entrenamiento. Además, al separar los distintos conjuntos de datos, se consideró el origen del vídeo de cada imagen, garantizando que el modelo no ha visto fotogramas de los vídeos utilizados para la comprobación de resultados discutidos en este apartado.

A lo largo de los entrenamientos, se ha observado que las métricas de *pose* o *keypoints* son consistentemente menores que las de *bounding boxes*. Por esta razón, en este análisis final nos centraremos principalmente en las primeras de ellas. Como se detalla en el 2.4.4.3 y en el 2.6.4.2, las métricas más efectivas para evaluar el rendimiento en la detección de pose son las mostradas en las siguientes figuras:

La Figura 30 representa la relación entre el valor de confianza de las predicciones y la *precision*. Se muestra que la *precision* se mantiene cerca de 1 para un amplio rango de valores de confianza (0.2 a 1.0), alcanzando el 100% de precisión a un nivel de confianza de 0.944. Esto sugiere que el modelo tiene una alta capacidad para evitar falsos positivos cuando se seleccionan niveles de confianza más altos, y una precisión robusta incluso a niveles de confianza moderados.

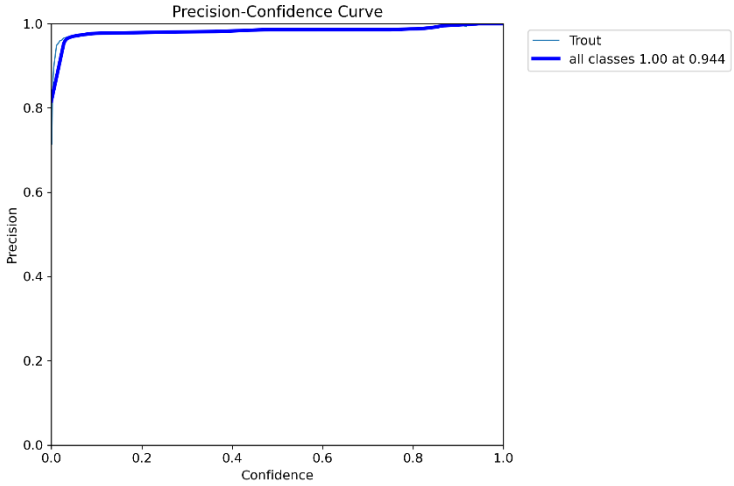

Figura 30: Gráfica *Precision-Confidence*

La Figura 31 muestra la relación entre el valor de confianza de las predicciones y el *recall*. Indica que el *recall* se mantiene cerca de 1 para un amplio rango de valores de confianza (0.0 a 0.8), alcanzando el 99% de *recall* a un nivel de confianza de 0.000. A medida que el valor de confianza aumenta más allá de 0.8, el *recall* disminuye drásticamente, indicando que el modelo se vuelve más selectivo y omite más verdaderos positivos. Esto sugiere que el modelo es muy efectivo en capturar la mayoría de los verdaderos positivos cuando se seleccionan niveles de confianza más bajos.

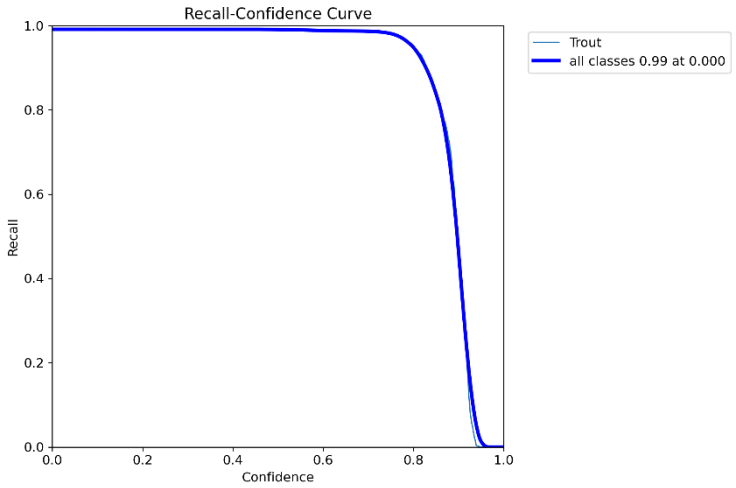

Figura 31: Gráfica *Recall-Confidence*

La Figura 32 muestra la relación entre la *precision* y el *recall* del modelo. Se indica que la precisión se mantiene alta mientras que el *recall* crece, alcanzando un *mAP* de 0.992 a un umbral de 0.5. Esta curva casi forma un rectángulo, lo que indica que el modelo mantiene un alto valor simultáneo de *recall y precision*. Un alto *mAP* sugiere que el modelo es muy efectivo en la detección y clasificación de *keypoints*, manteniendo un excelente equilibrio entre evitar falsos positivos y no omitir verdaderos positivos.

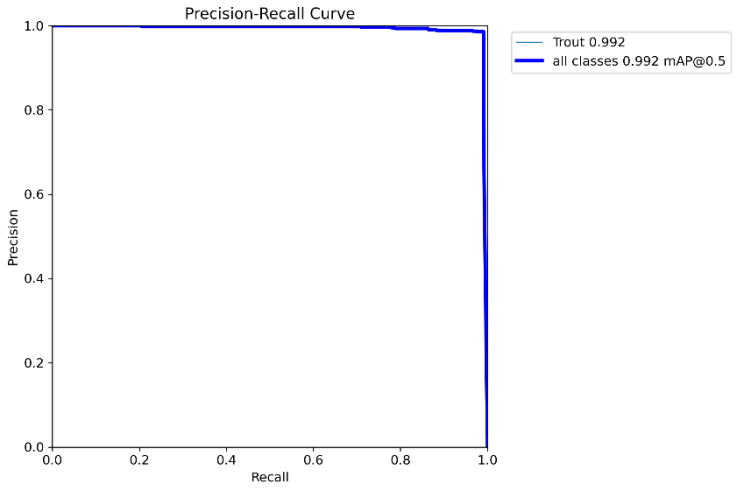

Figura 32: Gráfica *Precision-Recall*

La Figura 33 muestra la relación entre el valor de confianza de las predicciones y la métrica F1. Indica que el modelo alcanza un valor F1 cercano a 1 cuando el nivel de confianza está entre 0.4 y 0.8, sugiriendo un equilibrio óptimo entre *recall y precision* en este rango. A medida que el valor de confianza aumenta más allá de 0.8, el F1 score disminuye rápidamente, indicando que las predicciones se vuelven demasiado conservadoras, resultando en menos verdaderos positivos. El F1 score es de 0.99 a un nivel de confianza de 0.473, demostrando un rendimiento muy alto del modelo en general.

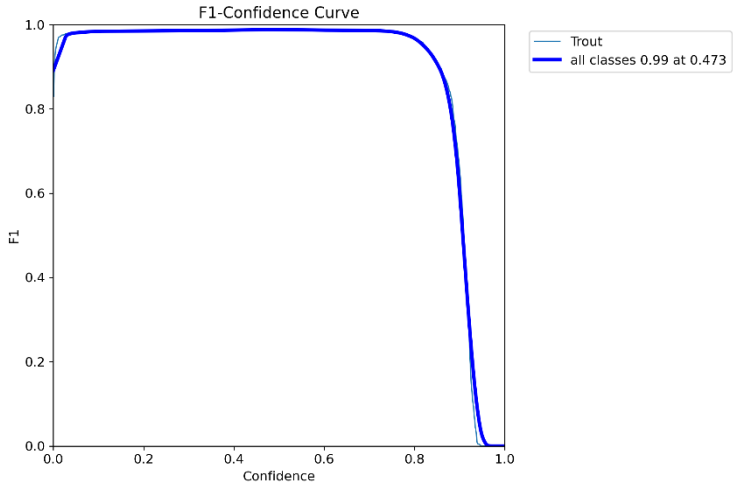

Figura 33: Gráfica *F1-Confidence*

Por último en la Figura 34 se muestran todas las gráficas anteriores, pero en relación a las *bounding boxes*, y como se adelantaba en la introducción son ligeramente superiores a las métricas de *keypoints*. La curva *Precision-Confidence* revela que la precisión alcanza el 100% a un nivel de confianza de 0.869, mientras que la curva *Recall-Confidence* muestra un *recall* constante de 1 hasta niveles de confianza muy altos. La curva *Precision-Recall* indica un *mAP* de 0.994 a un umbral de 0.5, y la curva *F1-Confidence* demuestra un F1 score cercano a 1 en un amplio rango de confianza. Esto refleja que el modelo es altamente preciso y balanceado, siendo efectivo en la detección de objetos con mínimos falsos positivos y negativos.

El modelo tarda aproximadamente 110 ms en realizar la inferencia de cada imagen de una trucha cuando el ordenador no está trabajando en ninguna otra tarea, y 190 ms con trabajos en paralelo. Debido a la estrategia adoptada en el script de automatización, cada fotograma de los videos se separa en dos mitades y se infiere individualmente. Esto se debe a que el modelo ha sido entrenado con imágenes aisladas de truchas y, aunque es capaz de realizar detección múltiple, su rendimiento en *recall* y *precision* de los *keypoints* es algo inferior. En resumen, cada fotograma del *Net Test* tarda en promedio 250 ms.

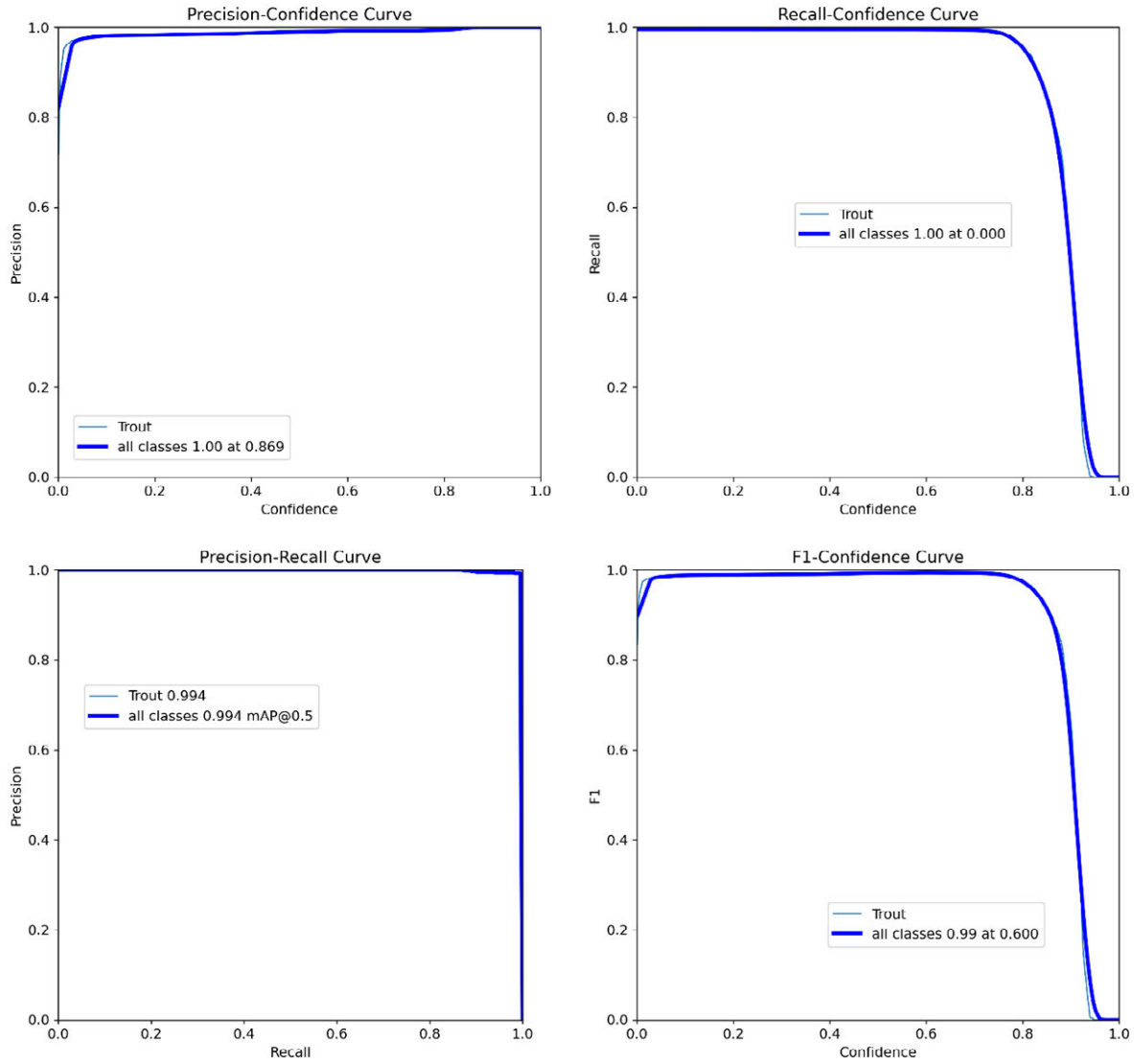

Figura 34: Gráficas de Rendimiento en *Bounding Boxes*

5.2 Comprobación manual

A pesar de los buenos resultados obtenidos en el punto anterior, se ha realizado un análisis manual del rendimiento del modelo. Este análisis se ha efectuado sobre vídeos no antes vistos por el modelo, los cuales no están etiquetados. Este proceso es similar a la validación que realizarán los técnicos del GAP cuando se les presente la automatización.

En general, la mayoría de los fotogramas son inferidos correctamente, es decir, la estimación de pose del modelo ha sido precisa. Algunos ejemplos se pueden ver en la Figura 35. La fila A muestra los casos ideales, donde no hay movimiento y la trucha está perfectamente posicionada. En la fila B, se presentan casos en los que, a pesar de que el individuo esté borroso, la inferencia sigue siendo correcta. Finalmente, en la última fila, se encuentran algunos casos que merecen ser comentados.

Por ejemplo, en el fotograma C1, aunque la cabeza del animal está completamente tapada por el cuerpo, el modelo sigue estimándola correctamente. Sin embargo, esta estimación no siempre es precisa en estas situaciones, como se observa en la imagen C4. Hay casos en los que, aunque los puntos auxiliares no se estimen correctamente, el ángulo de la línea principal (el dato relevante) sigue estando perfectamente estimado; un ejemplo de esto es el fotograma C2. Por otro lado, en la imagen C3, se puede ver que aunque no se han estimado correctamente todos los puntos centrales del animal, el ángulo de la línea principal, aun no siendo exacto, es una muy buena aproximación.

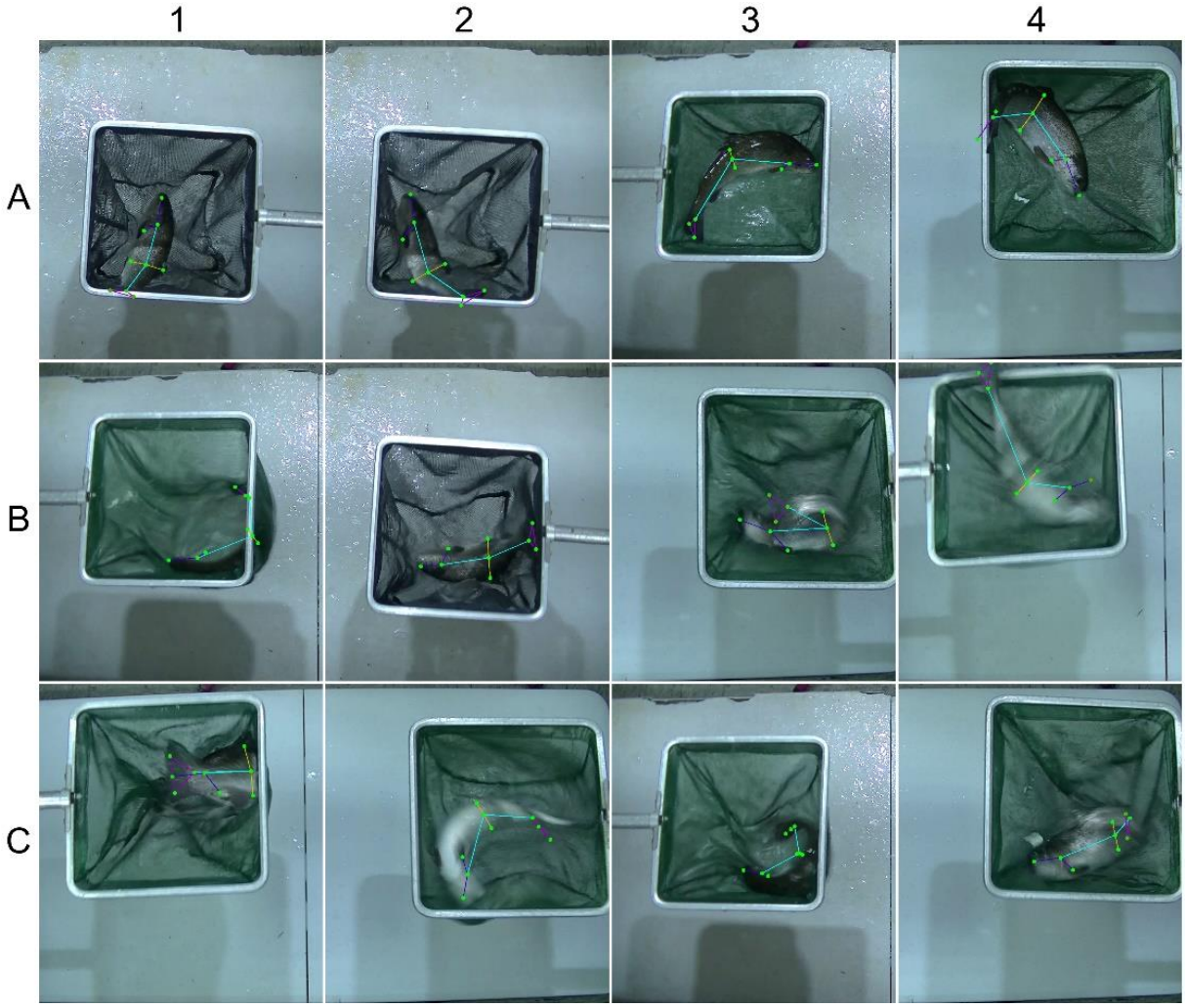

Figura 35: Fotogramas Correctamente Inferidos Resultado Final

En la Figura 36 se muestran inferencias que podemos clasificar como incorrectas. En la imagen A1, a pesar de tener alta confianza en los puntos, la inferencia de la cola es incorrecta y causa un error en la medida. En las imágenes A2 y B2, que son fotogramas contiguos de un vídeo, se observa cómo el error en la inferencia de los puntos causa una importante variación en el ángulo de la línea principal. En ambos casos, el modelo infiere los puntos de la cola con un alto nivel de confianza, aunque no está completamente seguro, como se puede apreciar en las tonalidades de los puntos. Las imágenes B1 y B3 muestran casos en los que el modelo infiere algunos puntos con aún menor confianza, indicándolo con tonalidades más suaves. En el caso de la imagen B4, aunque la inferencia de la cola es incorrecta, la borrosidad es tal que no se considera un fallo del modelo. Adicionalmente, no siempre se infieren algunos o todos los puntos, como puede verse en los casos A3 y A4.

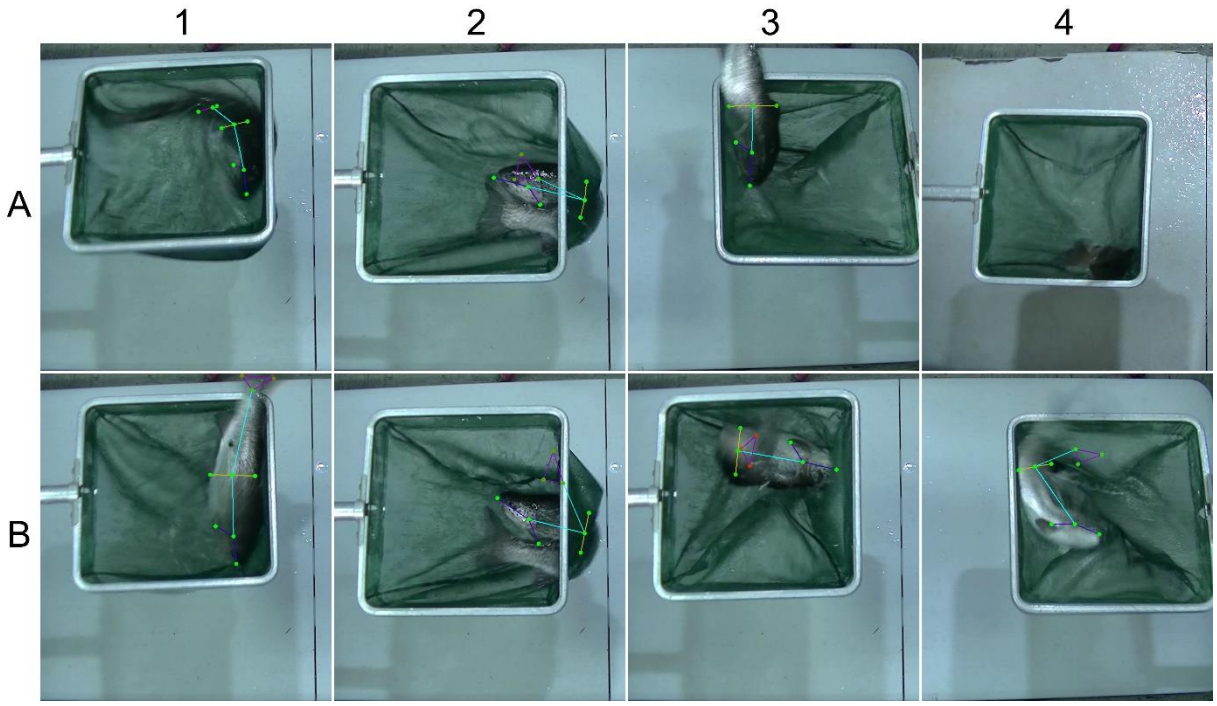

Figura 36: Fotogramas Dudosamente Inferidos Resultado Final

6. Comentario final

El proyecto puede considerarse concluido con resultados muy satisfactorios. A pesar de los errores mencionados anteriormente, la cantidad de errores en el conjunto total de vídeos inferidos es muy pequeña. Aunque siempre hay espacio para la mejora, en este caso consideramos que es mínimo, ya que al ser la inteligencia artificial esencialmente un modelo estadístico, siempre existirán pequeñas variaciones e incorrecciones.

Analizando los objetivos de este PFG, todos se han cumplido con resultados mucho mejores de lo esperado inicialmente. Se ha creado un conjunto de datos propio que se ha utilizado para entrenar un modelo de estimación de pose, con el objetivo final de medir cuantitativamente el movimiento de las truchas durante el experimento del *Net Test*. La salida de este análisis automático, aunque no es exactamente la misma que obtenían los investigadores del GAP, se considera mucho más significativa para realizar los estudios pertinentes. Aunque esta es una opinión personal, queda a evaluación del GAP determinar si los datos estadísticos son suficientemente claros y autoexplicativos. Independientemente de esto, el modelo entrenado cumple completamente con los objetivos, y en el caso de disconformidad con los estadísticos presentados, sería suficiente con utilizar el fichero de datos npy para realizar el estudio deseado, sin necesidad de volver a inferir los vídeos.

El tiempo de procesado de los videos se ha reducido drásticamente en un 86,67%, pasando de un tiempo de procesado manual de 15 minutos por video a solo 2 minutos de media. Con solo los vídeos iniciales, es difícil determinar si bajo alguna condición no prevista el modelo de inferencia podría tener un desempeño peor de lo esperado. En caso de que esto ocurra, bastaría con realizar alguna iteración adicional de entrenamiento con imágenes de dicha condición.

Para mejorar aún más el proyecto, se han identificado algunas mejoras a la hora de grabar: incrementar la velocidad de adquisición de fotogramas de la cámara, capturar un único pez por experimento, ya que, de esta manera podría ocupar todo el encuadre de la cámara lo que derivaría en una mayor precisión.

Futuras líneas de desarrollo incluyen la posibilidad de exportar el modelo a un entorno IoT, para desarrollar un producto de análisis de *Net Tests* en tiempo real, o hacer que la aplicación final ya desarrollada, sea más rápida.

Personalmente, quedo muy satisfecho con el desempeño/resultado final de este proyecto.

7. Referencias

[1] A. M. Villalba, A. de la Llave-Propín, J. De la Fuente, C. Pérez, G. de Chávarri, M. T. Díaz, A. Cabezas, R. González-Garoz, F. Torrent, M. Villarroel, y R. Bermejo-Poza, "Using underwater currents as an occupational enrichment method to improve the stress status in rainbow trout," *Fish Physiology and Biochemistry*, vol. 50, no. 2, pp. 463-475, 2024.

[2] S. Russell y P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.

[3] A. De la Llave-Propín, A. M. Villalba, M. Villarroel, C. Pérez, G. de Chávarri, M. T. Díaz, A. Cabezas, R. G. Garoz, J. De la Fuente, y R. Bermejo-Poza, "Environmental enrichment improves growth and fillet quality in rainbow trout", *Journal of the Science of Food and Agriculture*, vol. 104, no. 6, pp. 3487-3497, 2024.

[4] P. L. Cebrián Castel, "Desarrollo de un sistema de monitorización para cultivo en acuicultura basado en redes neuronales", Trabajo Fin de Grado, Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, 2020.

[5] A. Vidal Alegría, "Diseño e implementación de un sistema de reconocimiento de gestos mediante visión por computadora", Proyecto Fin de Carrera, Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros de Telecomunicación, 2021.

[6] I. Ortega Lobo, "Implementación de la red neuronal YOLOv3 para la detección de matrículas de vehículos", Proyecto Fin de Carrera, Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros de Telecomunicación, 2020.

[7] Chen Fu Hao Fen, "Automatización del experimento *openfield* con detección y seguimiento de truchas", Proyecto Fin de Carrera, Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, 2024.

[8] J. E. Barrios Sánchez, "La prueba de la red: Evaluando su repetibilidad y el efecto del ayuno en la trucha arcoíris (*Oncorhynchus mykiss*)", Proyecto Fin de Carrera, Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingeniería Agronómica, Alimentaria y de Biosistemas, 2023.

[9] Bovik, Ed., Handbook of Image and Video Processing, 2ª ed., Academic Press, 2005, ISBN: 978-0121197926.

8. Bibliografía

En este capítulo se recogen las referencias bibliográficas consultadas durante la realización del PFG y que han servido para guiar el desarrollo del mismo y redactar la memoria. El criterio de clasificación seguido es: División entre naturaleza del recurso citado, ordenación alfabética por el apellido del primer autor y más tarde orden cronológico.

8.1 Libros y Artículos Científicos:

M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," arXiv preprint arXiv:1603.04467, 2016.

D. Bahdanau, K. Cho, y Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," arXiv preprint arXiv:1409.0473, 2014.

Y. Bengio, I. Goodfellow, y A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," en Proceedings of COMPSTAT'2010, 2010, pp. 177-186.

F. Chollet, Deep Learning with Python. Shelter Island, NY, USA: Manning Publications, 2018.

B. J. Copeland, "Artificial intelligence (AI) | Definition, Examples, Types, Applications, Companies, & Facts," Encyclopædia Britannica. Available: https://www.britannica.com/technology/artificial-intelligence. [Accessed: 10-Jun-2024].

A. Dosovitskiy et al., "ImageGPT: Generative Pretraining from Pixels," en Proceedings of the 37th International Conference on Machine Learning (ICML), 2020.

G. E. Hinton et al., "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 82-97, Nov. 2012.

G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, y N. Jaitly, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 82-97, 2012.

S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

K. He, X. Zhang, S. Ren, y J. Sun, "Deep Residual Learning for Image Recognition," en Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.

I. Goodfellow, et al., "Generative adversarial nets," in Advances in Neural Information Processing Systems, vol. 27, 2014.

I. Goodfellow, Y. Bengio, y A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

P. Isola, J. Zhu, T. Zhou, y A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," en Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1125-1134.

D. P. Kingma y M. Welling, "Auto-Encoding Variational Bayes," en Proceedings of the 2nd International Conference on Learning Representations (ICLR), 2014.

D. P. Kingma y J. Ba, "Adam: A Method for Stochastic Optimization," arXiv preprint arXiv:1412.6980, 2014.

Y. Lecun, L. Bottou, Y. Bengio, y P. Haffner, "Gradient-based Learning Applied to Document Recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, 1998.

A. Radford, L. Metz, y S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," arXiv preprint arXiv:1511.06434, 2015.

A. Radford et al., "Learning Transferable Visual Models From Natural Language Supervision," en Proceedings of the International Conference on Machine Learning (ICML), 2021.

S. Ren, K. He, R. Girshick, y J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 2017.

J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," en Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788.

O. Ronneberger, P. Fischer, y T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," en Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2015, pp. 234-241.

J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," Neural Networks, vol. 61, pp. 85-117, 2015.

D. Silver et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," Nature, vol. 529, pp. 484-489, 2016.

K. Simonyan y A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," en International Conference on Learning Representations (ICLR), 2015.

I. Sutskever, O. Vinyals, y Q. V. Le, "Sequence to Sequence Learning with Neural Networks," en Advances in Neural Information Processing Systems (NeurIPS), 2014, pp. 3104-3112.

C. Szegedy et al., "Going Deeper with Convolutions," en Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1-9.

A. Vaswani et al., "Attention is All You Need," en Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 5998-6008.

J. Yosinski, J. Clune, Y. Bengio, y H. Lipson, "How Transferable are Features in Deep Neural Networks?," en Advances in Neural Information Processing Systems (NeurIPS), 2014, pp. 3320-3328.

H. Zhao et al., "Pyramid Scene Parsing Network," en Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2881-2890.

M. D. Zeiler y R. Fergus, "Visualizing and Understanding Convolutional Networks," en European Conference on Computer Vision (ECCV), 2014, pp. 818-833.

8.2 Recursos en Línea:

"AI Index Report 2024 – Artificial Intelligence Index," Stanford University. Available: https://aiindex.stanford.edu

"Artificial Intelligence: Overview, Recent Advances, and Considerations for the 118th Congress," Congressional Research Service, 2023. Available: https://crsreports.congress.gov/product/pdf/R/R47644.

"Artificial intelligence - Wikipedia," Wikipedia. Available: https://en.wikipedia.org/wiki/Artificial_intelligence. [Accessed: 10-Jun-2024].

"What Is Artificial Intelligence (AI)? | Built In," Built In. Available: https://builtin.com/artificial-intelligence. [Accessed: 10-Jun-2024].

"Artificial Intelligence Techniques: A Complete Overview," The Knowledge Academy. Available: https://www.theknowledgeacademy.com/blog/artificial-intelligence-techniques/. [Accessed: 10-Jun-2024].

"What is Artificial Intelligence (AI)? | IBM," IBM. Available: https://www.ibm.com/artificial-intelligence. [Accessed: 10-Jun-2024].

Anexo A: Presupuesto Económico

En este apartado se estiman los costes requeridos para llevar a cabo el proyecto en el hipotético caso de que existiera un cliente final, aunque en realidad se trata de un proyecto de investigación. En el estudio se ha hecho una distinción entre costes materiales, que incluyen software y hardware. Cabe destacar que, gracias al respaldo del CITSEM, no ha sido necesario gastar dinero para adquirir material. Además, al tratarse de un Proyecto Fin de Grado (PFG), el coste de recursos humanos ha sido absorbido por el proyectante y el tutor.

El precio por hora que aparece en el estudio no refleja el sueldo bruto de los ingenieros, sino el coste empresarial total que implicarían las horas de cada tipo de ingeniero. Este coste incluye el sueldo de los ingenieros y otros gastos asociados.

Aunque el desarrollo de este proyecto no está enfocado en un uso comercial, se ha utilizado la licencia AGPL-3.0 de Ultralytics. Sin embargo, para realizar el estudio del presupuesto económico de un proyecto con fines lucrativos, esa licencia no podría ser utilizada y habría que adquirir una licencia empresarial. El coste de dicha licencia ha sido estimado al máximo coste posible, aunque es probable que el coste real sea algo inferior.

Las siguientes tablas recogen las diferentes partidas contempladas en el PGF.

COSTES DIRECTOS (CD)				
MANO DE OBRA	**Nº Personas**	**Horas**	**Precio/Hora**	**Total**
Ingeniero Junior	1	250	45 €	11.250 €
Ingeniero Senior	1	25	56 €	1.400 €
Total				**12.650 €**
GASTOS MATERIALES	**Precio compra**	**Tiempo uso**	**Amortización**	**Total**
Ordenador Personal	800 €	6	60	80 €
GPU Externa (Opt)	500 €	4	60	33 €
Licencia Ultralytics	10.000 €	6	6	10.000 €
Total				**10.113 €**

GASTOS GENERALES	% Sobre		Total
GASTOS TOTALES CD			**22.763 €**
GASTOS TOTALES CI	15%	CD	**3.415 €**
Total			**26.178 €**
RESUMEN	**% Sobre**		**Total**
BENEFICIO INDUSTRIAL	20%	CD + CI	**5.236 €**
SUBTOTAL PRESUPUESTO			**54.177 €**
IVA APLICABLE	21%	Subtotal	**11.377 €**
Total			**70.789 €**

Anexo B: Impacto del proyecto

El proyecto desarrollado en este PFG impacta varios Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas[6]. A continuación, se detallan los ODS relevantes y cómo el proyecto contribuye a cada uno de ellos:

- **ODS 2: Hambre Cero:** La acuicultura es una fuente importante de proteínas de alta calidad. Mejorar la eficiencia y sostenibilidad en la cría de peces a través de la automatización del *Net Test* puede ayudar a incrementar la producción sostenible de alimentos acuáticos, contribuyendo a la seguridad alimentaria y nutricional.
- **ODS 3: Salud y Bienestar:** La evaluación del bienestar de los peces es crucial para garantizar prácticas acuícolas sostenibles y éticas. La automatización del análisis del *Net Test* permite una evaluación más precisa y consistente del bienestar de los peces, mejorando así las prácticas de manejo y reduciendo el estrés animal.
- **ODS 9: Industria, Innovación e Infraestructura:** El uso de tecnologías avanzadas como la inteligencia artificial y la visión por ordenador en la acuicultura representa una innovación significativa. La implementación de un sistema automatizado para el análisis del *Net Test* mejora la infraestructura tecnológica de la industria acuícola, promoviendo una mayor eficiencia y productividad.
- **ODS 12: Producción y Consumo Responsables:** Al mejorar la eficiencia en la cría de peces y reducir el desperdicio a través de prácticas mejoradas basadas en datos precisos y automatizados, el proyecto fomenta la producción responsable y sostenible. Esto contribuye a un uso más eficiente de los recursos y a la minimización de impactos ambientales negativos.

  ODS 14: Vida Submarina: La automatización del *Net Test* ayuda a garantizar prácticas de acuicultura más sostenibles y respetuosas con el medio ambiente. Al monitorear y mejorar las condiciones de cría, el proyecto contribuye a la conservación y uso sostenible de los océanos, mares y recursos marinos, promoviendo la salud de los ecosistemas acuáticos.

[6] M. J. Gamez, «Objetivos y metas de desarrollo sostenible», Desarrollo Sostenible. Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

Anexo C: Manual de usuario

En este apartado se explicará el proceso de instalación, y el uso de la aplicación final:

C.1 Instalación

Haga clic derecho en el archivo de instalación `Ejetruchar_v1_WIN_setup.exe` y seleccione "Ejecutar como administrador". Espere unos minutos para que el instalador arranque ya que este proceso puede ser lento. Siga las instrucciones que aparecerán durante la instalación.

NOTA: Tenga en cuenta que durante el proceso de instalación, el antivirus puede generar una alerta debido a que el software es de código abierto y no está firmado por Microsoft.

C.2 Aplicación

Tipo de procesado (1): Seleccione el tipo de entrada: "Video", "Carpeta" o "Todo". Esto define si se procesará un solo video, todos los videos de una carpeta específica, o todos los videos en todas las subcarpetas, de la carpeta seleccionada.

Entrada (2): Se selecciona el archivo de video o la carpeta de entrada según el tipo de procesado seleccionado. La ruta seleccionada se mostrará en el cuadro de texto.

Salida (3): Se selecciona la carpeta de salida donde se guardarán los resultados del procesamiento. La ruta seleccionada se mostrará en el cuadro de texto correspondiente.

Procesar (4): Una vez que ambas rutas (entrada y salida) estén seleccionadas, haga clic en este botón para iniciar el procesamiento. Se abrirá una ventana de progreso que mostrará el avance del procesado de cada uno de los videos. Este proceso no puede ser interrumpido.

Ayuda (5): Haga clic en el ícono de interrogación para ver una ayuda sobre el funcionamiento de la aplicación y cómo usar cada uno de los elementos de la interfaz.

NOTA: La aplicación es lenta a la hora de arrancar, puede demorar un par de minutos.