PROYECTO FIN DE GRADO

TÍTULO: Sistema para la detección automática del movimiento de truchas en un experimento *Net Test*.

AUTOR: Diego Aceituno Seoane

TITULACIÓN: Ingeniería Telemática

TUTORA: Juana María Gutiérrez Arriola

DEPARTAMENTO: Departamento de Ingeniería Telemática y Electrónica

VºBº TUTORA

Miembros del Tribunal Calificador:

PRESIDENTA: Margarita Millán Valenzuela

TUTORA: Juana María Gutiérrez Arriola

SECRETARIO: Pedro Castillejo Parrilla

Fecha de lectura: 23 de julio de 2024

Calificación:

El Secretario,

Gracias a mi familia, especialmente a mi madre y mi hermano por apoyarme durante estos años.

Gracias a Juana por su labor como tutora en este trabajo y por todas las oportunidades que me han permitido acercarme al mundo de la investigación.

También, agradecer a todos los miembros del grupo GAMMA por su dedicación al alumnado y por sus conocimientos.

A mis compañeros del grupo GAMMA, entre ellos, Carlos, Rocío, Cristina, Yago, Lucía, Fernando y Héctor que me han acompañado y aconsejado sobre este trabajo, muchas gracias.

Gracias a mis amigos de la carrera por hacer esta etapa una de las mejores de toda mi vida académica, especialmente a Félix, Hugo, David y Estela por esforzarse conmigo.

Por último, gracias a todos mis amigos de fuera de la carrera, especialmente a Raúl por apoyarme en momentos en los que me sentí perdido.

RESUMEN

En los últimos años la visión por ordenador ha tomado un papel muy importante en la vida cotidiana. Dentro de este campo encontramos técnicas como el análisis de flujo óptico o las redes neuronales de tipo `CNN`, siendo `YOLO` una arquitectura muy popular.

Con estas nuevas técnicas se busca automatizar procesos en la industria de la acuicultura, tanto de producción como experimentales. En el estado del arte se presentan usos actuales recientes en piscifactorías para controlar bancos de peces, detectar especímenes y contabilización de los mismos. Algunas de las ventajas del uso de estas líneas de investigación es la mejora de las condiciones de los peces en la acuicultura.

En este trabajo se trata la automatización del experimento *NetTest*, que se utiliza para analizar la cantidad de estrés que genera en los animales el cambio de condiciones en los entornos en los que viven. Este experimento se basa en contar el número de movimientos realizados por un pez en una red fuera del agua, y, actualmente es realizado de forma manual.

Para conseguir esto, se ha desarrollado una aplicación de detección automática y contabilización del número de movimientos realizados por truchas en videos del experimento *NetTest*. Con este objetivo, se ha formado un conjunto de datos y se ha entrenado un modelo de `YOLOv8`, el cual procesa el video y devuelve los resultados de las `Bounding Boxes` asociadas a cada trucha.

Una vez localizada la trucha se ha diseñado un algoritmo que utiliza el área de la `bounding box` y elementos como el desplazamiento del centroide para marcar en qué fotograma del video se ha producido un movimiento de escape. Finalmente, para poder facilitar el uso de esta herramienta, se ha desarrollado una interfaz gráfica para el análisis de los resultados en los experimentos *NetTest*.

Para implementar todas las funcionalidades se han utilizado herramientas como la ejecución paralela y la inferencia de forma transparente al `HardWare` a través del uso de múltiples formatos de modelo de red neuronal.

Como resultados, el modelo `YOLOv8n` se ha entrenado hasta reducir el error asociado a la `Bounding Box` y maximizar la puntuación `F1`. Finalmente, la comparación de resultados con una serie de datos etiquetados en 64 videos ha demostrado una tasa de error de $\pm 20\,\%$. Estos mismos resultados que antes se tardaban en obtener semanas, con esta herramienta se han podido obtener en menos de una hora.

ABSTRACT

In recent years, the development of computer vision tools has become very important in our lives. In this field there are techniques such as optical flow analysis or convolutional neural networks, being YOLO one of the most popular architectures.

The use of these new technologies aims to automate different processes in the aquaculture industry, both in the production and in the experimental phases. The research in the state of the art in this field presents recent case use scenarios like fish schools monitoring, detect different specimens and count the number of fishes in an image.

This document deals with the automation of the NetTest experiment, which is being used to analize the amount of stress that changes in the environment can generate in a fish. In this experiment, the number of movements made by a trout in a net are counted, while being out of the water. Currently, this experiment is performed manually.

To achieve this, an application for automate detection and counting of the number of movements made by a trout in the NetTest experiment is presented. To accomplish this, a data set was made and a `YOLOv8` model was trained. The model task is to process the video and return data related to the `Bounding Boxes` detected for each trout.

When the trout is located, an algorithm has been designed that takes elements such as the area of the bounding box and it's centre to indicate the frames of the video in which a movement has occurred. Finally, to facilitate the use of this tool, a graphical user interface has been developed that helps to analize the results obtained in the videos processed.

To implement all these functionalities, different tools were used such as multiprocessing and `HardWare` independent inference with different formats of the same model.

In the validation of the results, the `YOLOv8n` model was trained till the `Box Loss` error was minimized and the `F1` metric maximized. Finally, the movement results were compared to labels obtain from scientist of the *NetTest*. The labels and the results were compared in 64 videos, obtaining an error of $\pm 20\,\%$. These results, which previously took weeks to obtain, can now be obtained in less than an hour with this tool.

Índice

Índice de figuras

Índice de tablas

1. LISTA DE ACRÓNIMOS

CITSEM Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad 25

CNN Convolutional Neural Networks 20, 24, 25, 40

COCO Common Objects in Context 24

CPU Central Processing Unit 29, 32, 49

CSV Comma Separated Values 9, 61

CVAT Computer Vision Annotation Tool 42

ETSIAAB Escuela Técnica Superior de Ingeniería Agronómica, Alimentaria y de Biosistemas 34

FAO Food and Agriculture Organization of the United Nations 13

FIFO First In First Out 58

FPS Frames Per Second 10, 24, 35, 36, 43

GAMMA Grupo de Aplicaciones Multimedia y Acústica 42

GIL Global Interpreter Lock 58

GPU Graphics Processing Unit 29, 43, 47, 49

GUI Graphical User Interface 53, 57, 91

I+D Investigación y Desarrollo 13

JSON JavaScript Object Notation 9, 61

mAP Mean Average Precision 24, 44, 45, 90

MLP MultiLayer Perceptron 19, 20, 22

MOV QuickTime File Format 36

MTS MPEG Transport Stream 35

NAS Network Attached Storage 42

OBB Oriented Bounding Boxes 44, 47, 52

OCR Optical Character Recognition 17

ODS Objetivos de Desarrollo Sostenible 15, 75

PRAN Grupo de Investigación de Producción Animal 34, 35, 37

RAM Random Access Memory 59

ReLU Rectified Linear Unit 21

ROI Region Of Interest 24

SIMD Single Instruction/Multiple Data 49

TPU Tensor Processing Unit 49

YOLO You Only Look Once 24

2. INTRODUCCIÓN

2.1. Marco y motivación del proyecto

Las crisis alimenticias afectan a diversas partes del globo, en mayor o menor medida. Estas crisis pueden producirse por factores naturales o factores humanos como, por ejemplo, fallos en industrias, guerras, o, la que directamente se relaciona con este proyecto, agotamiento de los recursos naturales por la sobreexplotación.

Ante estas situaciones globales que pasan por revisar los valores de obtención de alimento y el cómo se gestionan los recursos, la pesca marítima está viendo como insostenible la compatibilidad de su labor con la generación de suficiente alimento vivo para suplir las necesidades exponenciales del ser humano.

El caso que más afecta a este trabajo es la acuicultura, un método alternativo que nació para obtener control sobre el desarrollo de especies muy difíciles de capturar en entornos libres. Esto se consigue a través de crianza controlada en entornos limitados, como piscifactorías o jaulas masivas en costas.

Este método de generación de alimento ha ido desarrollándose y mejorando a través de la introducción de nuevas tecnologías. Esta mejora ha permitido utilizar la acuicultura como principal método de crianza de ciertas especies, que; aun encontrándose en libertad, es más viable económicamente trabajar con ellas en estos entornos controlados. Esto permite liberar estrés de los ecosistemas marítimos.

Con estos ecosistemas tan afectados por las pescas de arrastre y, con la presión añadida por políticas limitantes en este ámbito, la pesca tradicional ha visto como la cantidad de alimento generado no crece a lo largo de los años. Esto se ve reflejado a través de los informes realizados por la *FAO (Food and Agriculture Organization of the United Nations)*, siendo el último en 2022, que marcaba aún más la tendencia del impulso hacia sistemas basados en acuicultura, observable en la Figura 1.

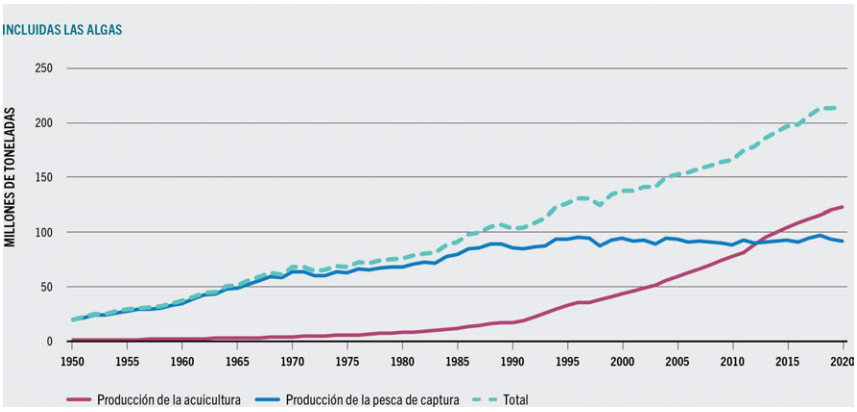


Figura 1: Evolución de la producción en la industria pesquera 1950-2020[1]

La acuicultura está representando un cambio de paradigma para proteger nuestros ecosistemas y producir con menos perdidas y de manera más ética. Con esto en mente, se están promoviendo muchos planes de transformación digital en estas industrias, con la inversión en I+D (Investigación y Desarrollo) que eso conlleva.

Algunas de las investigaciones que se realizan en este campo tienen relación con el estudio de métodos de crianza que generen menos estrés en el pez. Mejoras en este campo podrían permitir un crecimiento más rápido del alimento vivo aumentando su bienestar y un mejor marco ético en la crianza del animal.

En concreto, para realizar un análisis y parametrizado del estrés del animal, se llevan a cabo diferentes experimentos con el objetivo final de valorar los cambios en el entorno que afectan más a las truchas. El *NetTest*[2] es uno de estos experimentos.

Este experimento consiste en someter al pez a una situación de estrés sacándolo del agua durante un tiempo corto y analizando, posteriormente, cuántos movimientos de escape ha realizado.

El principal objetivo de este experimento es parametrizar la proactividad de un pez a través de los movimientos que realiza. Junto con este y otros datos del pez como podría ser su color o tamaño, intentar encontrar una relación entre los cambios de entorno y su nivel de estrés. Si bien esta prueba no está estandarizada (la definición de movimiento puede variar entre diferentes trabajos académicos), en este trabajo se tratará el caso específico en el que los movimientos contados, son arpeos del espécimen.

En la actualidad esta prueba se realiza de forma manual, siendo necesario que el investigador utilice herramientas de análisis de videos para apuntar cuando se realizan los movimientos, avanzando fotograma a fotograma. Un ejemplo de este tipo de herramientas es BORIS[3].

La existencia de este tipo de herramientas de código abierto es muy positiva para el mundo de la investigación, pero el hecho de tener un funcionamiento completamente manual conlleva un gasto de tiempo y recursos en obtener estos informes de movimientos para cada video. Además, este experimento es propenso a errores humanos, ya que el criterio para contar movimientos puede variar entre evaluadores e, incluso, entre dos evaluaciones del mismo experto que se realicen separadas en el tiempo.

Con esto en mente es interesante crear una alternativa que automatice el proceso de conteo de movimientos para los investigadores que utilizan el *NetTest* como experimento principal.

Con los avances en campos como la visión por ordenador y las redes neuronales en los últimos años, la automatización de este experimento se vuelve viable.

2.2. Objetivos técnicos y académicos

Los objetivos de este trabajo consisten en:

- Plantear una alternativa capaz de automatizar el *NetTest*.
- Estudiar el uso de tecnologías basadas en procesado y análisis de imagen y técnicas de `Deep Learning` para implementar esta alternativa.
- Desarrollar una aplicación completa a través de la alternativa planteada.

Los objetivos académicos de este proyecto son:

- Desarrollar una mejor capacidad de análisis de requisitos técnicos a la hora de diseño de aplicaciones.
- Profundizar en el entendimiento de técnicas basadas en `Deep Learning` como las redes neuronales.
- Mejorar las aptitudes relacionadas con el análisis de artículos académicos con el objetivo de realizar un estudio del arte suficientemente profundo, cohesionado y coherente con la problemática explicada.

2.3. Estructura del resto de la memoria

En el resto del documento, se hablará primeramente del marco tecnológico sobre el que se desarrolla la solución. A continuación se detallarán los requisitos y restricciones para el correcto diseño de la solución a desarrollar.

Posteriormente se presentará la solución y los datos con los que se ha trabajado. Con la solución descrita, se comenzará a describir el desarrollo que se ha seguido en el proyecto para cumplir los objetivos del mismo, incluyendo la validación de resultados relacionada con el correcto funcionamiento de la herramienta generada.

Finalmente, como apartados previos a la conclusión y las líneas de investigación futuras se indicará el presupuesto tomado para este proyecto, el manual de usuario de la aplicación y el impacto social que tiene siguiendo los ODS (Objetivos de Desarrollo Sostenible).

3. MARCO TECNOLÓGICO

El problema que se busca solucionar en este trabajo tiene relación directa con el estudio de la visión por ordenador. Este campo de la Ingeniería busca, a través de sistemas de captación y procesamiento de imágenes, generar información y automatizar procesos con ordenadores.
Dentro de este área se han desarrollado tecnologías que usamos día a día [4]:

- ***OCR (Optical Character Recognition)*** para realizar reconocimiento de texto en imágenes y documentos.
- **Detección de fallos en maquinaria** en procesos de fabricación.
- **Fotogrametría** para mapear imágenes a modelos `3D`.
- **Imagen médica** para uso en operaciones a tiempo real[5].
- **Detección de objetos** en imágenes o videos.
- **Detección de movimiento** y seguimiento de objetos en escenas.

Las aplicaciones que usan herramientas basadas en este tipo de soluciones suelen seguir una estructura similar a la siguiente:

1. Capturar información a través de cámaras o flujos de video IP de diferentes características.
2. (A veces) Realizar un preprocesamiento de las imágenes para adaptarlas al sistema.
3. Procesar las imágenes para extraer información relevante para el sistema.
4. Transformar la información extraída a través de algoritmos inteligentes para construir una base de datos que sea útil para el usuario de la aplicación.

En el caso de este trabajo, se tratan problemas de detección de objetos y análisis de cambio. Este tipo de tareas se han llevado a cabo antiguamente a través de algoritmos complejos y deterministas, pero en los últimos años, con el auge del aprendizaje automático, se ha creado un nuevo paradigma.

En los siguientes puntos se van a describir de forma breve las técnicas de aprendizaje automático, los sistemas de detección de objetos, seguimiento de cambio y los entornos de desarrollo y despliegue de estas técnicas de forma transparente al `Hardware` disponible así como proyectos relacionados con este trabajo.

3.1. Aprendizaje Automático y `Deep Learning`

3.1.1. Definición y tipos de tareas

El campo del aprendizaje automático busca desarrollar soluciones a través de algoritmos y sistemas que permitan a los ordenadores tratar los datos entrantes como el cerebro humano, siendo capaz de generalizar las propiedades que les definen y encontrar patrones.

Este aprendizaje automático se realiza sobre un conjunto de datos que aporta el usuario con el objetivo de abstraer diferentes informaciones específicas o agrupar los datos según patrones. Este aprendizaje habitualmente se realiza a través de un bucle de tres fases (ver Figura 2) en las cuales se configura el sistema para aprender, se valida el sistema resultante y se reconfiguran los parámetros para acercarse a un sistema que aporte mejores resultados.

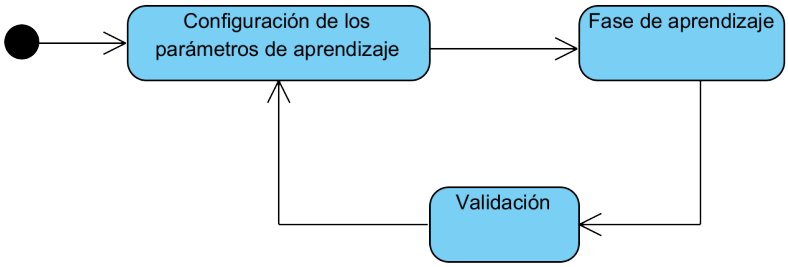


Figura 2: Bucle de fases de aprendizaje en un sistema de `Machine Learning`

Dentro del campo existen diferentes ramas dependientes del objetivo del aprendizaje que se busca y los datos con los que vamos a alimentar el sistema:

- **Entrenamiento supervisado**: es un tipo de entrenamiento en el que se conoce la relación entre los datos de entrada y los datos esperados.
- **Entrenamiento no supervisado**: el objetivo de este aprendizaje es aprender las relaciones de los datos de entrada entre sí.
- **Entrenamiento reforzado**: es una variación del entrenamiento supervisado en el que el sistema recibe feedback sobre los resultados como parte de la fase de aprendizaje. Se utiliza mucho en el ámbito de la robótica.

Entre todas las técnicas que implementan este tipo de tareas, uno de los sistemas modernos más conocidos son las redes neuronales.

3.1.2. Redes Neuronales

Este campo tiene su origen en una publicación de Warren McCulloch y Walter Pitts en 1943[6] en el que presentaban la idea de neurona como una unidad lógica básica implementable cuyo estado se puede definir como un "todo o nada", lo cual se representa en el campo de las telecomunicaciones con una función escalón como se ve en la Figura 3. Este trabajo y diseño de neurona sería conocido más tarde como perceptron.

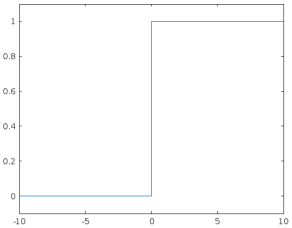


Figura 3: Figura de activación escalón de las primeras neuronas

La estructura básica del perceptron se puede observar en la Figura 4 y consiste en un nodo con un número $\mathcal{N}$ de conexiones entrantes, con un peso asociado $\mathcal{W}$ a cada entrada. El nodo calcula el sumatorio de cada conexión con su respectivo peso. Al resultado de este cálculo se le añade un factor de `bias` y por último, mediante una función de activación como la vista anteriormente, se decide el resultado de salida de la neurona.

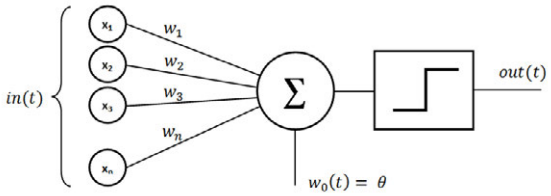
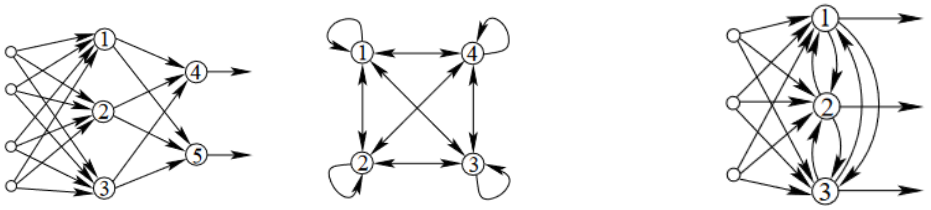


Figura 4: Diagrama general de un perceptron[7]

Más tarde, se conseguiría implementar un diseño electrónico de esta neurona, e interconectar más neuronas entre sí. Estas interconexiones dan lugar a la creación del concepto de **redes neuronales**, siendo una de las primeras redes complejas funcionales de tipo `MLP (MultiLayer Perceptron)`. Estos sistemas serían clave para la evolución del aprendizaje automático, apareciendo multitud de variaciones en la arquitectura de la red según las necesidades (ver Figura 5).



(a) Red FeedForward (b) Red recursiva (c) Red FeedForward con inhibidores laterales

Figura 5: Diferentes ejemplos de arquitecturas simples de redes neuronales[8]

El avance progresivo de la capacidad de cómputo de los ordenadores ha permitido el uso de redes neuronales de mayor tamaño, que permiten mayor capacidad de abstracción. Es en este marco donde aparece el término `Deep Learning`, que no indica nada más que el uso de redes neuronales con una cantidad masiva de capas intermedias. Además, son sistemas que trabajan muy bien cuando se implementan con ejecución paralela, aunque esto se explicará más adelante.

Este tipo de sistemas han demostrado su gran capacidad de abstracción y rapidez, es por esto mismo que las redes neuronales profundas son una de las principales herramientas que se utilizan hoy en día en visión por ordenador.

3.2. Detección de Objetos con redes neuronales

La detección de objetos como tarea surge de la necesidad de analizar la existencia de objetos en imágenes y; opcionalmente, localizarlos en las mismas. Para realizar este tipo de tareas se han ido desarrollando técnicas tradicionales basadas en algoritmos deterministas, pero, con el auge del aprendizaje automático, a partir de 2012[9] el paradigma pasó a centrarse en la creación de sistemas detectores basados en `Deep Learning` (Sistemas de Redes Neuronales con una gran cantidad de capas ocultas).

Ambas técnicas se basan en la extracción y el análisis de características de las imágenes, esto es, la caracterización de imágenes o zonas de interés a través de algoritmos o filtros. Podemos entender la caracterización de una imagen como la parametrización de los elementos que la definen. Dentro de la vista humana existen muchos elementos que se pueden usar para definir una imagen, algunos ejemplos son:

- **Nitidez**.
- **Color**.
- **Presencia de ciertas formas geométricas**.
- **Brillo**.
- **Ruido**

Se debe tener en cuenta que estos ejemplos son características que da un humano para una imagen, de forma contraria, las redes neuronales trabajan con características de bajo nivel (P. ej.: existencia de bordes en una zona específica, colores en un área, patrones de puntos, etc.) que permitan analizar zonas de la imagen. Sin embargo, las redes neuronales tipo `MLP` no fueron pensadas para este tipo de tareas. Esto es debido a que el análisis de características en una imagen es una tarea cuya dificultad computacional aumenta de forma exponencial con el aumento de la resolución, ya que, para cualquier imagen, debería haber una neurona de entrada por cada pixel y canal de la imagen, sobre cuyos valores la red `MLP` aplicaría los cálculos.

Por estos motivos, se buscó una forma de añadir independencia espacial a la entrada de las redes neuronales. Esto se consigue mediante el uso de diferentes arquitecturas, siendo la más importante las redes neuronales convolucionales.

3.2.1. Arquitecturas de redes neuronales tipo `CNN`

Las `CNN (Convolutional Neural Networks)` definen una arquitectura basada en dos pasos: la extracción de las características de las imágenes y la clasificación de las imágenes según las características. Su estructura básica se puede ver Figura 6 y se compone de dos subredes que cumplen las dos funciones diferentes para los datos de entrada.

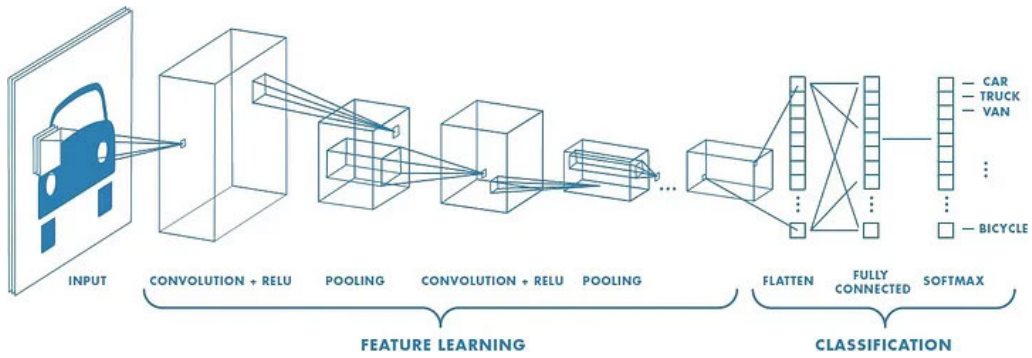


Figura 6: Arquitectura básica de una `CNN`[10]

La fase de caracterización se compone principalmente de capas neuronales convolucionales y, habitualmente capas de `pooling`:

- Capas convolucionales: tienen como objetivo la extracción de características a través de neuronas que implementan operaciones de convolución sobre las imágenes. Al ser las imágenes tensores, se les aplican filtros convolucionales implementados a través de `Kernels`. Dependiendo de la característica a buscar, existen diferentes filtros convolucionales y en las fases de entrenamiento se va conformando el `Kernel` que caracteriza mejor esa zona de la imagen para los datos esperados.

  Un ejemplo de `Kernel` es un filtro laplaciano gaussiano como el de la Figura 7, que, a través del gradiente, sirve para darle más énfasis a los bordes de la imagen, como se puede ver en la Figura 8.

$$\begin{vmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

Figura 7: Aspecto de un `kernel` de Laplace de tamaño `3x3`

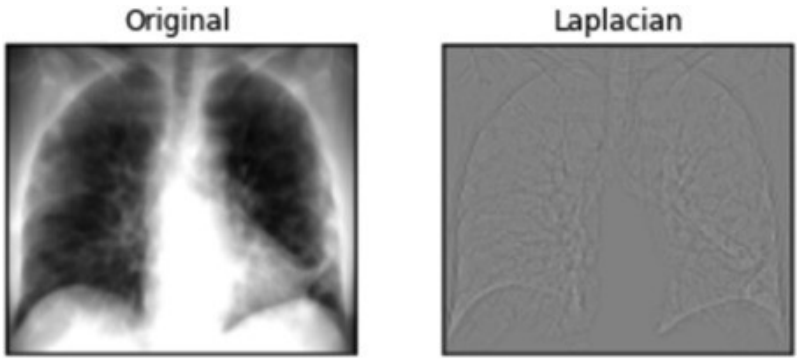


Figura 8: Aplicación de un filtro gaussiano de Laplace a una imagen médica[11]

Estas capas convolucionales van obteniendo la abstracción de las características transformando los volúmenes de entrada a volúmenes de salida de mayor profundidad pero de menor resolución vertical y horizontal.
Habitualmente en la parte convolucional se utilizan funciones de activación como la ReLU (Rectified Linear Unit) de la Figura 9.

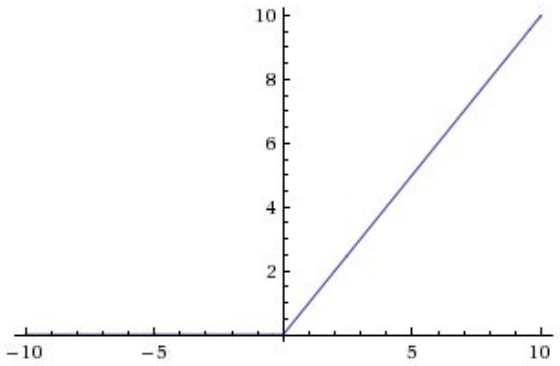


Figura 9: Función de activación `ReLU`

- Capas de `pooling`: Su objetivo es ir haciendo sub-muestreo de las capas convolucionales superiores para conformar el volumen de entrada de la siguiente capa convolucional, reduciendo la carga computacional necesaria. Se puede ver su funcionamiento en la Figura 10, donde sub-muestrean un volumen de resolución `28x28` a una resolución `14x14`, pero habitualmente se aumenta la profundidad del tensor.

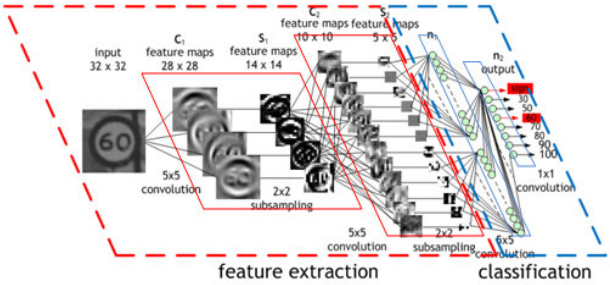


Figura 10: `CNN` con detalle de la caracterización y el sub-muestreo[12]

Este sistema se va repitiendo para cada capa de convolución, hasta que en la última capa de `pooling`, la salida es un vector horizontal cuyo tamaño es el miso que neuronas de entrada tenga la etapa de clasificación.

La fase de clasificación habitualmente se compone de una capa conectada completamente como se ha podido ver en la anterior sección, similar a la idea de una `MLP`. Esta red toma como entrada un vector aplanado final de la parte de extracción de características y realiza la clasificación. Habitualmente utiliza funciones de activación que son capaces de tener $\mathcal{N}$ salidas una de estas funciones habituales es la `SoftMax` de la Figura 11, que mapea un vector de entrada a un vector de salida de probabilidades que escalan según los valores de entrada.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Figura 11: Función de activación `SoftMax`

El uso de estas redes neuronales convolucionales para solucionar un problema está asociado al tipo de trabajo a realizar. Cuando se define un problema en el que se va a aplicar esta técnica, normalmente cae en tres tipos de categorías presentadas en la Figura 12. Además, esta selección tiene unas implicaciones en la complejidad computacional de la tarea, ya que la capa de clasificación debe ser cambiada para soportarla.

- Clasificación: las imágenes son marcadas dependiendo de si existe o no un elemento específico dentro de ella.
- Detección de objetos: regiones locales de la imagen son clasificadas y marcadas dependiendo de la existencia del elemento en cuestión.
- Segmentación semántica: se realiza una clasificación por cada pixel para delimitar todos los píxeles que conforman la instancia del objeto.

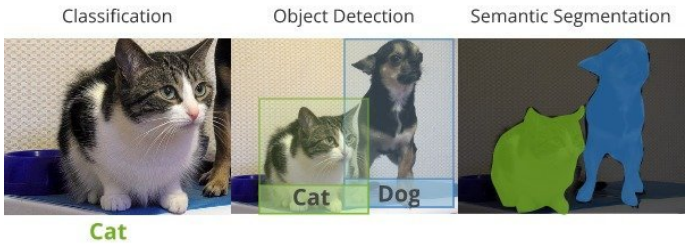


Figura 12: Tareas típicas de una CNN[13]

Toda red neuronal sobre la que se realice entrenamiento supervisado debe ser entrenada con un conjunto de datos etiquetados de forma coherente a la tarea que se quiera realizar. Por ejemplo, una CNN entrenada para clasificar imágenes deberá ser alimentada con etiquetas adecuadas, no con coordenadas sobre la posición del elemento detectado.

Estos conjuntos de datos que se preparan para las redes neuronales se dividen habitualmente en conjuntos con diferentes propósitos en el entrenamiento de la red neuronal:

- Conjunto de datos de entrenamiento: es el conjunto de datos sobre el que la red va a aprender, aplicando de forma constante su algoritmo. Pasar por este conjunto entero una vez suele llamarse época de entrenamiento.
- Conjunto de datos de prueba: es un conjunto de datos completamente separado que nunca se aplicará a la red en la fase de entrenamiento. Su objetivo es evaluar al final del entrenamiento el rendimiento general de la red en un conjunto de datos que nunca haya visto.
- Conjunto de datos de validación: estos datos se utilizan como conjunto de referencia entre épocas, su existencia y uso depende de la arquitectura de red neuronal, ya que existen algunos algoritmos que modifican parámetros del entrenamiento de la red entre épocas.
  Cuando se habla de los parámetros de la red, habitualmente se refieren a cosas como los pesos entre neuronas, la función de activación, el número de capas ocultas, etc... Sin embargo, los parámetros que se definen para el entrenamiento como el número de épocas, parámetros del algoritmo de aprendizaje, tamaño de los datos introducidos a la vez, etc., son llamados hiperparámetros.

Normalmente se dispone de un conjunto de datos que se divide en entrenamiento, validación y prueba. Es habitual una separación de 70 % para el conjunto de entrenamiento, 15 % para el conjunto de validación y 15 % para el conjunto de prueba.

La incorrecta preparación del conjunto de datos puede afectar severamente a los resultados obtenidos. Por lo tanto es habitual intentar evitar redundancia de datos y alimentar a la red con datos que no sean muy similares entre sí para que sea capaz de generalizar correctamente en la mayor cantidad de situaciones posibles.

Para solucionar este problema, podemos encontrar técnicas de aumento de datos, que generan artificialmente datos extra haciendo transformaciones en las imágenes, cambiando el color o incluso aplicando distorsión. Esto permite solucionar situaciones en las que se dispongan de pocos datos o los datos sean demasiado parecidos entre sí como para poder asegurar una buena generalización a otros tipos de situaciones.

La evolución de estas CNN ha sido exponencial y han aparecido arquitecturas que han ido añadiendo mejoras de eficiencia, calidad de resultados y estándares a la hora de crear nuevos sistemas. Algunos ejemplos de esto son:

- R-CNN (Region-CNN)(2014): introdujo el análisis solo en regiones con datos interesantes.
- Fast R-CNN (2015): mejora R-CNN introduciendo búsqueda selectiva, definiendo un número máximo de ROI (Region Of Interest) a tratar.
- Faster R-CNN (2015).
- YOLO (You Only Look Once)[14]: arquitectura extremadamente rápida perteneciente al estado del arte moderno en redes neuronales, que ha ido mejorándose en forma de versiones.

Cada arquitectura ha traído consigo mejoras, a veces, a cambio de perdida de rendimiento. Esto es un problema que debe tenerse en cuenta y existen diversos trabajos que tratan simplemente las diferencias existentes entre diferentes arquitecturas.

Uno de estos trabajos[15] compara diferentes arquitecturas contra el conjunto de datos de la universidad de Oxford "Pascal2012"[16], que contiene 20 clases de objetos.

A través del estudio del estado del arte realizado en el artículo, se observa que YOLOv2 es muy competente a la hora de obtener rendimiento y velocidad de procesamiento, aún sin ser la red con los mejores resultados, es capaz de superar a arquitecturas ya establecidas como SSD512, obteniendo un mejor mAP (Mean Average Precision)(parámetro de caracterización del error medio típico sobre todo el conjunto de datos) con el doble de rendimiento. Se puede observar un resumen de los resultados de este artículo en la Tabla 1.

	mAP	FPS
Faster R-CNN	76.4	5
SSD300	74.3	46
SSD512	76.8	19
YOLO	63.4	45
YOLOv2 $544 \cdot 544$	78.6	40

Tabla 1: Detalle de rendimiento en FPS en la comparativa de arquitecturas de CNN[15]

Estos resultados comparativos, a diferencia de otros artículos[17], son útiles, ya que utilizan reglas tales como la comparación bajo el mismo conjunto de datos e indican el tamaño de la imagen de entrada a la red neuronal. Esto permite realizar una comparación realista y en las mismas condiciones.

En el estudio, también se observa que YOLOv2 tiene ciertos problemas al tratar con objetos pequeños. Algunos estudios[18] ya han observado que es un problema bastante típico de YOLO y es una de las situaciones en las que peor funciona.

Estos problemas han ido solucionándose con el tiempo, a través de la mejora y el aumento de rendimiento y puntuación mAP. Esto se puede ver en la Figura 13, donde se comparan diferentes versiones posteriores a la versión 5 sobre el conjunto de datos COCO (Common Objects in Context) (conjunto de datos con más de 200.000 imágenes etiquetadas que se usa como estándar para validar redes neuronales en objetos comunes).

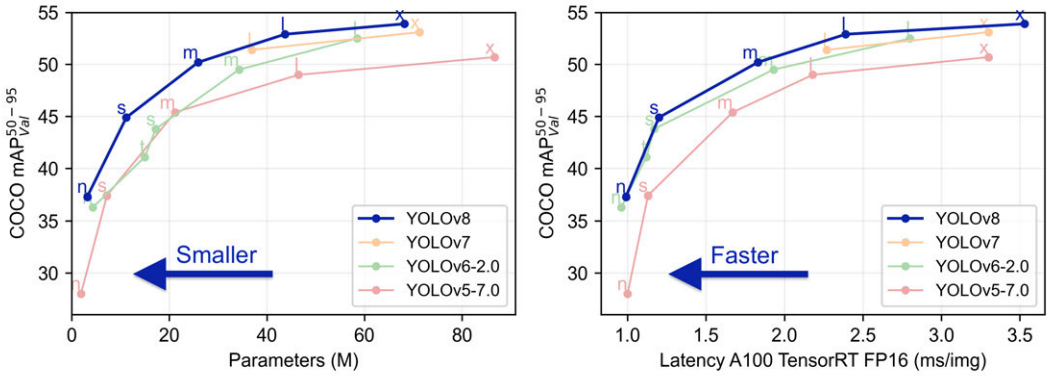


Figura 13: Rendimiento de diferentes versiones de `YOLO` superiores a la 5 en el conjunto de datos `COCO`[19]

Esto ha convertido a `YOLO` en un estándar de facto en la implementación rápida de soluciones basadas en `Deep Learning` para muchos investigadores por sus resultados en la detección en tiempo real.

Actualmente `YOLO` es un proyecto llevado por la empresa `ultralytics`, siendo el último modelo `YOLOv8`. Este modelo está conformado por una capa convolucional personalizada basada en la arquitectura `CSPDarknet53`, que conecta diferentes fases convolucionales para obtener un entrenamiento mucho más rápido. Esta capa convolucional se conecta al cuello de `YOLO`, cuya función es utilizar varias salidas de algunas de las secciones de la capa convolucional para tener información de diferentes escalas (cada capa convolucional va transformando la anterior entrada en un vector más y más horizontal).

Lo anterior es común a todos los modelos de `YOLOv8` independientemente de la tarea que realizan, lo que cambia es la parte final de la red, que, aprovechando la presencia de datos de diferentes escalas, utiliza redes neuronales completamente conectadas para diferentes tareas, una para las localizaciones, otra para las probabilidades de clase de cada elemento y otra para trabajar con regiones de interés.

Para facilitar la lectura del documento, en el anexo A se realiza una explicación detallada de los parámetros e hiperparámetros habituales de los modelos de `YOLOv8`.

3.2.2. Uso de redes neuronales en acuicultura

La acuicultura es un trabajo que engloba desde la producción de algas hasta la crianza de marisco en cautiverio. Además, dentro de cada rama podemos tener especies que sean similares y que deban ser diferenciadas.

En la literatura de este ámbito se lleva implementando tecnologías basadas en aprendizaje automático para diferentes tareas, en el caso del uso de `CNN` podemos ver sistemas para:

- Sistemas de conteo de peces[20].
- Sistemas de análisis de densidad.
- Sistemas de seguimiento[21].
- Sistemas de diferenciación y detección de especies en entornos abiertos[22][23]: aprovechando la existencia de conjuntos de datos [24] para experimentos con diferentes peces.

Existen ejemplos del uso de estas tecnologías en proyectos como el del CITSEM (Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad) "Acuicultura 4.0"[25] y diversos trabajos de fin de grado y máster relacionados, pero no para el conteo de movimientos a través de YOLO.

3.3. Técnicas para el análisis de movimiento en la visión por ordenador

Estas técnicas son muy importantes en el campo de la vigilancia, para poder percibir el cambio de elementos en la imagen y realizar su seguimiento. Según los autores M.Sonka y V.Hlavac[26], los problemas que trata se pueden dividir en tres grupos:

- Detección de movimiento general, clasificando una imagen con o sin movimiento a través de un descriptor global.
- Detección y localización de objetos en movimiento, donde la escena es estática y el objeto se mueve sobre ella.
- Uso del movimiento para la reconstrucción 3D de una escena.

De los anteriores puntos, el más relevante en este trabajo es la detección y localización de objetos en movimiento (conteo de movimiento de un pez).

Debemos entender que el análisis de movimiento tiene en cuenta el eje temporal entre las imágenes, que puede tener muestras equiespaciadas o no. Es en este eje sobre el que queremos realizar el análisis, pero cabe destacar que hay muy poco análisis realizado sobre el uso de aprendizaje automático para estimación de movimiento en video. Los principales sistemas se basan en análisis a través de algoritmos.

- **Análisis diferencial**: un ejemplo de esta técnica es la resta de una imagen número n con la imagen número $n-1$, para obtener una diferencia en ciertos pixeles y determinar a través de un límite la existencia de movimiento. Uno de los tipos de algoritmos más interesantes en este campo es el análisis de flujo óptico.
- **Correspondencia de puntos de interés**: se basa en la detección y seguimiento a través del eje temporal de ciertos puntos claves, para detectar el movimiento de elementos. Existen operadores para este objetivo, pero no se adaptan bien a cambios demasiado rápidos.
- **Detección de patrones de movimiento**: uno de los pocos sistemas que se basa en entrenar un clasificador capaz de detectar patrones de movimiento, sin embargo son herramientas muy complejas de implementar y tienen un alto coste computacional, ya que normalmente realizan las tres tareas: detección, clasificación y segmentación.

La mayoría de herramientas desarrolladas en este campo tienen necesidades muy específicas y no se adaptan bien a cambios demasiado bruscos, la única que puede manejarlos es una herramienta como el análisis de flujo óptico. Esto es debido a que se puede usar para analizar cambios entre todos los píxeles de las imágenes.

3.3.1. Definición y uso del análisis de flujo óptico

El análisis de flujo óptico u `OpticalFlow` apareció por primera vez en 1981[27] y se basa en asumir:

1. El brillo observado en un punto del objeto es constante en el eje temporal.
2. Puntos cercanos se mueven de la misma manera.

El problema es que en las imágenes reales las reglas sobre el brillo y la suavidez del cambio es habitual que no sean respetadas. Esto puede ser por diversos motivos como poco `framerate` o movimientos muy cercanos a la cámara que generen cambios bruscos en los pixeles.

Es una técnica que se usa mucho en las carreteras, por la existencia de entornos muy estáticos. Se puede ver un ejemplo de su aplicación en la Figura 14.

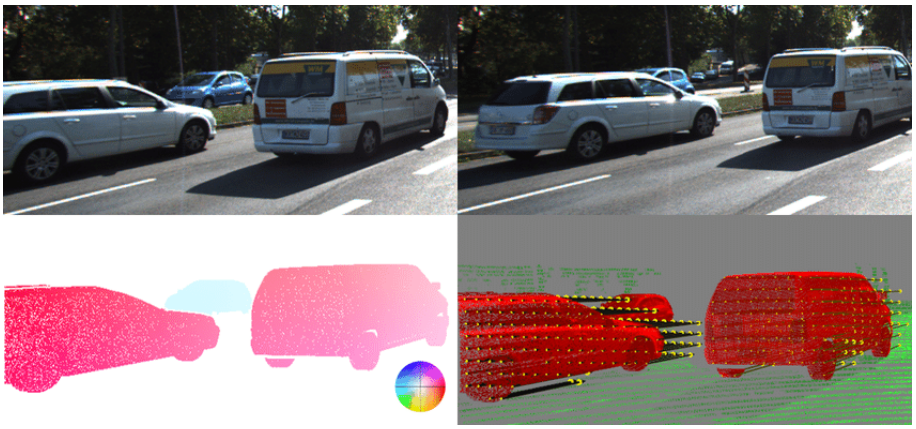


Figura 14: Análisis del flujo óptico entre 2 fotogramas de una escena[28]

Actualmente, existen tres algoritmos principales para calcular el flujo óptico de una escena:

- Método Horn-Schunck: devuelve una estimación global.
- Método de Lucas-Kanade: asume que el flujo va a ser constante en los vecinos, y con esto, realiza una serie de estimaciones locales.
- Método combinado local-global: calcula el flujo óptico en zonas rectangulares de la imagen.

3.3.2. Uso del análisis del flujo óptico en entornos de acuicultura

El análisis de flujo óptico se ha usado en acuicultura de forma diversa:

- Analizar comportamientos de grupo, por ejemplo, para determinar la actividad de un conjunto de peces de forma global[29].
- Realizar una detección, conteo y seguimiento de peces en un tanque en la India, mezclando `YOLOv5` con `OpticalFlow`[30]. Este método dio buenos resultados porque la distancia del pez a la cámara era suficiente como para que no hubiese movimientos demasiado grandes como se puede ver en la Figura 15.

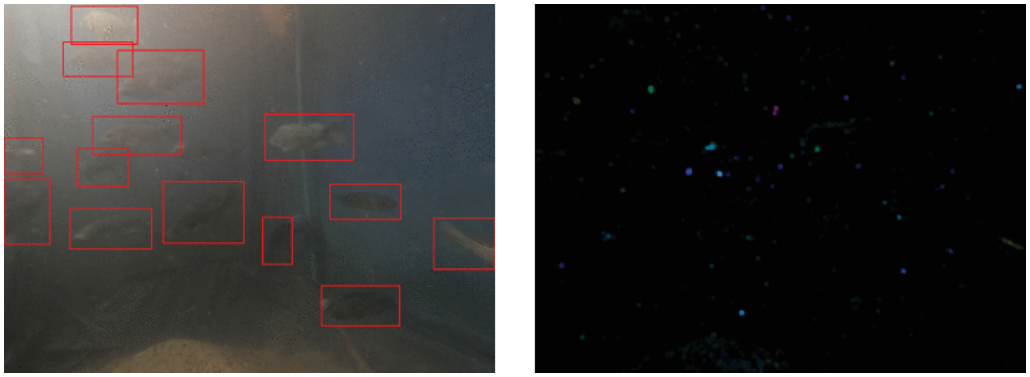


Figura 15: Datos obtenidos por OpticalFlow respetando las reglas de distancia sobre peces[30]

3.4. Herramientas de desarrollo y despliegue en `HardWare` transparente

- **Librerías de desarrollo para Redes Neuronales**: Para trabajar en la creación de redes neuronales, se usan principalmente dos librerías: `Keras` y `PyTorch`. `Keras` es una librería de alto nivel, mientras que `PyTorch` es de bajo nivel y tiene mejores herramientas de manejo de errores y de control del rendimiento de la red creada. Aparte de esto, `PyTorch` está pensado para su uso directamente en `Python` y soporta entrenamiento en sistemas distribuidos.

  En el caso de `YOLO`, la empresa `Ultralytics` retomó el proyecto y creo las versiones a partir de `YOLOv5`. El objetivo de esto ha sido crear una librería para que el usuario no tenga que interactuar con las librerías de bajo nivel. Esto nos permite entrenar, validar y desplegar modelos a través de modelos base.

- **Transfer Learning y Fine Tunning**: Entrenar una red neuronal desde 0 es demasiado complejo, por ejemplo, una red `YOLOv8` de tamaño pequeño tiene `11.2` millones de pesos que deben ser entrenados y ajustados. Estos pesos incluyen las capas convolucionales y la capa de clasificación.
  Pero lo habitual es obtener un modelo ya entrenado, cuyas capas convolucionales sean capaces de extraer características típicas de cualquier imagen y, solo entrenar la parte de clasificación. Esto reduce mucho el coste computacional de entrenamiento y por lo tanto, el tiempo requerido para llegar al objetivo deseado con la red. Esta técnica se conoce como `Transfer Learning` y se puede ver en la Figura 16.

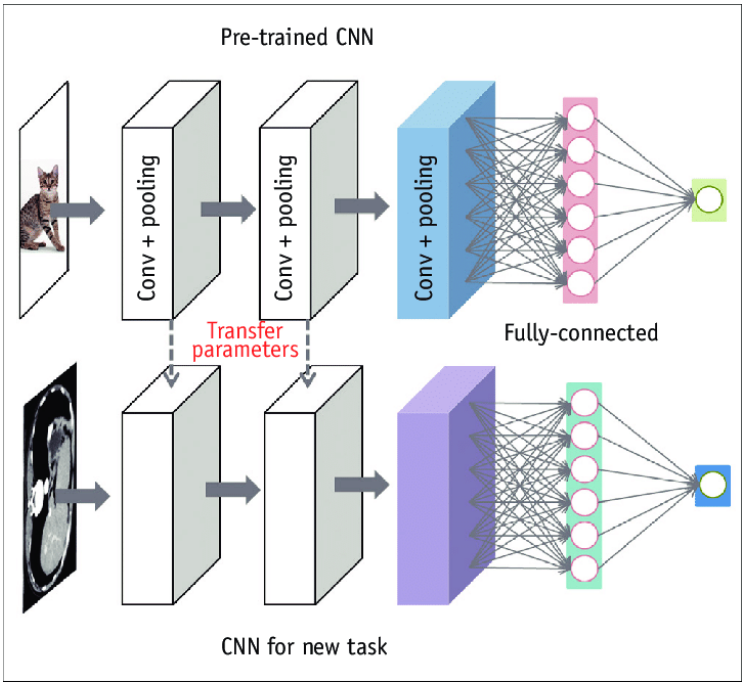


Figura 16: Uso de parámetros realizando `Transfer Learning`[31]

Posteriormente, cuando se alcanza el rendimiento esperado, se pueden desbloquear los pesos de las capas de convolución y hacer entrenamiento de toda la red. Esta fase se llama `Fine Tunning`. Su objetivo es ajustar la capa de convolución también para solo analizar las características de la imagen más relevantes para la tarea.

- **Herramientas comunes para el análisis de flujo óptico**: En la mayoría de lenguajes existen librerías que implementan los algoritmos de análisis de flujo óptico, pero, al ser herramientas de caracterización de imagen, existen muchas facilidades en entornos matemáticos como MATLAB o de procesado de datos como el lenguaje de programación `Python`, más concretamente la librería de `OpenCV`.

- **Abstracción de la red neuronal al `HardWare`**: A la hora de desplegar una red neuronal, se debe diseñar una estrategia para poder aprovechar todos los recursos del sistema que va a ejecutar la red. Esto es debido a que una red neuronal se aprovecha de la computación en paralelo, ya que principalmente resuelve sumas y multiplicaciones con matrices.

  En este sentido, los despliegues más eficientes son los que son capaces de aprovechar elementos como las `GPU (Graphics Processing Unit)`, que se centran en este tipo de operaciones. Al contrario que una `CPU (Central Processing Unit)`, la `GPU (Graphics Processing Unit)` cuenta con más núcleos de cómputo de menor potencia como se puede ver en la Figura 17.

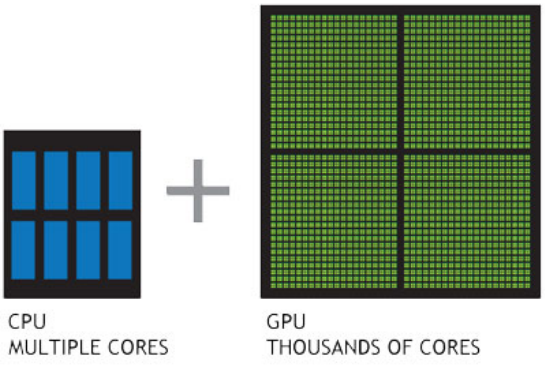


Figura 17: Diferencia de cantidad de núcleos en una `CPU` y una `GPU`[32]

En este sentido, según la plataforma de desarrollo de red neuronal que se utilice, el despliegue podrá aprovechar la gráfica dependiendo de factores como:

- Marca: `NVidia`, `AMD` o `Intel`.
- Generación del `HardWare`.
- Sistema Operativo en el que se despliega el sistema.

Por ejemplo, una red neuronal desarrollada en PyTorch solo soporta gráficas de `AMD` si el sistema operativo es `Linux`. Es por este motivo que han surgido librerías abiertas que permiten el uso de una aceleración por `HardWare` común a muchos más dispositivos. Ejemplo de esto son los modelos desarrollados en el entorno `ONNX`. De hecho, algunas empresas como `Intel` desarrollan sus propias librerías de herramientas para permitir compatibilidad con sus sistemas de aceleración. Estas librerías suelen contener además, herramientas de transformación entre modelos para aportar más transparencia al usuario final y evitar problemas de compatibilidad.

4. REQUISITOS Y RESTRICCIONES DE DISEÑO

Como proyecto de ingeniería, el análisis previo de requisitos y restricciones del sistema se ha realizado para modelar el comportamiento esperado resultante del desarrollo de la solución propuesta

Dentro del análisis de requisitos, se han diferenciado entre funcionales(definiciones del comportamiento del sistema) y no funcionales(definiciones de atributos de control de calidad).

4.1. Análisis de requisitos

Req. ID	Descripción de requisito funcional	Prioridad
0	Los cálculos automáticos deben ser transparentes a los conocimientos de los usuarios	Alta
1	El usuario deberá poder ver un manual del sistema	Baja
2	Si el video seleccionado dura menos de 50 fotogramas, la inferencia debe cancelarse y se informará al usuario	Alta
3	La inferencia debe hacerse posterior a la selección, verificación y confirmación del video	Alta
4	El usuario deberá poder seleccionar diferentes formatos de video sobre los que inferir de forma transparente	Media
5	Si ocurre un error en la selección de fichero, se informará por un pop-up	Alta
6	La inferencia se realizará de forma transparente al usuario en una pantalla de carga	Alta
7	La inferencia se podrá cancelar para evitar perdidas de tiempo	Media
8	En la pantalla de carga se informará de cuanto queda para acabar la inferencia	Alta
9	Si ocurre un error en la inferencia se deberá informar con un pop-up y volver a la pantalla inicial	Alta
10	Se deberá hacer la inferencia en paralelo para no colgar la aplicación	Alta
11	Si se completa la inferencia, deberá pasarse a la pantalla de resultados	Alta
12	En la pantalla de resultados el usuario deberá poder elegir guardarlos en `CSV`	Alta
13	En la pantalla de resultados el usuario debe poder verificar los resultados obtenidos	Alta
14	En la pantalla de resultados el usuario debe poder añadir o eliminar movimientos de los resultados obtenidos	Media
15	En la pantalla de resultados el usuario debe poder volver a la pantalla inicial	Media
16	La aplicación se abrirá desde un ejecutable para facilitar el uso	Media
17	El usuario debe poder elegir la carpeta de guardado del `CSV`	Media

Tabla 2: Análisis de requisitos funcionales

Req. ID	Descripción de requisito no funcional	Prioridad
18	El sistema automático deberá permitir a los científicos realizar su tarea en menos tiempo que manualmente	Alta
19	El sistema debe poder ser manejado por cualquier usuario que sepa realizar tareas de ofimática	Alta
20	El sistema debe poder compilarse en un ejecutable para que el usuario no necesite manejar dependencias	Alta
21	Los errores no deben bloquear la aplicación, deben ser manejados y controlados correctamente en lo posible	Media
22	Las entradas de datos si fuesen necesarias se harán de forma simple para el usuario, ya sea a través de darle a botones o usar el ratón	Media
23	El sistema debe aprovechar al máximo los recursos `HardWare` a ser posible, seleccionando el mejor modelo para el ordenador en el que se lanza la aplicación	Alta
24	La comunicación entre procesos debe manejar problemas de concurrencia	Alta
25	La inferencia se realizará fotograma por fotograma para limitar el uso de memoria RAM del sistema al mínimo necesario	Alta
26	Se utilizará una librería para la interfaz de usuario que permita el renderizado sobre OpenGL o similares para descargar la `CPU`	Alta
27	La capa de inferencia debe dar unos resultados transparentes independientemente del modelo utilizado	Baja
28	El modelo debe poder cambiar en tamaño para ajustarse a diferentes requerimientos en un futuro	Media
29	El fichero de datos de usuario debe poder ser portátil para que el ejecutable lo detecte y cargue los datos de usuario si existen	Baja
30	El usuario no tiene por qué especificar el número de truchas que contiene el video, se debe estimar solo	Media
31	La pantalla de muestra de datos ocultará gráficas en función del número de peces que existan	Alta
32	Los elementos de la interfaz deben ser en su mayoría redimensionables para que el usuario pueda ajustarlos para ver mejor	Media
33	Se intentará usar letras claras para mejorar la accesibilidad de la aplicación	Baja
34	En el `CSV` guardado, se almacenará el nombre del video, los números de movimiento de cada lado y los fotogramas en los que ocurren	Baja

Tabla 3: Análisis de requisitos no funcionales

4.2. Análisis de restricciones

Rest. ID	Descripción de restricción
0	La aplicación debe poder correr en un ordenador con `Windows`
1	La aplicación se implementará usando el lenguaje de programación `Python`
2	Se usará en la medida de lo posible, librerías y herramientas de código abierto y/o gratis
3	La aplicación se desarrollará usando `Git` para realizar un control de versiones
4	Los datos de la aplicación sobre los videos se guardarán en claro, no tienen un riesgo de seguridad
5	Los videos de entrada no podrán contener menos de 50 fotogramas
6	Por la naturaleza del *NetTest*, el sistema está pensado para videos de truchas

Tabla 4: Análisis de restricciones

5. DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

Se busca desarrollar una aplicación completa que sirva como sistema en gran medida automatizado para obtener una estimación de los movimientos realizados por truchas en videos de experimentos *NetTest*. Para realizar esta tarea, se diseñará una arquitectura dividida en tres sistemas específicos como se puede ver en la Figura 18.

- Interfaz gráfica de usuario: permitirá al usuario seleccionar el video a analizar, observar los resultados y guardarlos.
- Capa de análisis del video: a través de técnicas como el `Machine Learning` o el análisis de flujo óptico, parametrizará los movimientos de las truchas que aparezcan en el video.
- Capa de procesado de datos: obtendrá la caracterización realizada por la capa de análisis y a través de la transformación y procesado de los datos, devolverá los datos que se deban mostrar en la interfaz gráfica. Además realizará la estimación final del número de movimientos por trucha.

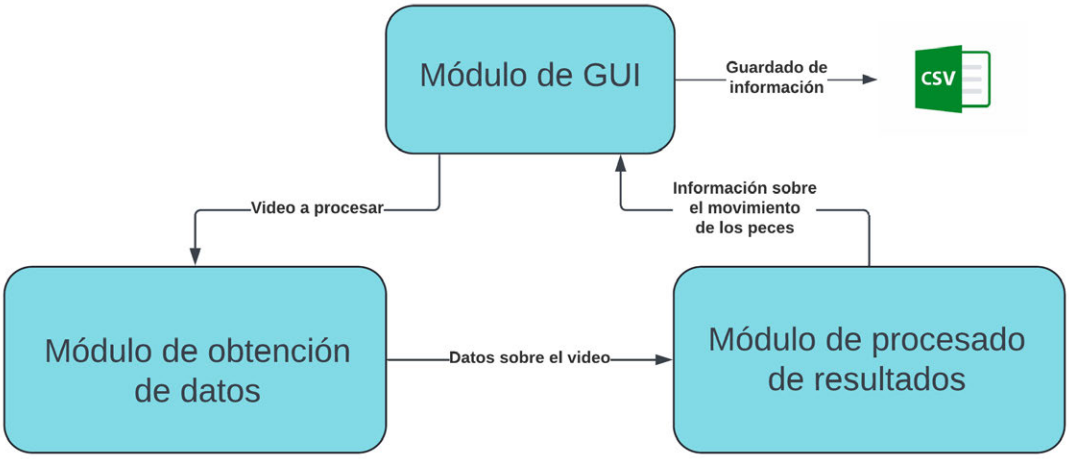


Figura 18: Arquitectura básica de la solución propuesta

El bucle de uso principal consistirá en la selección de un video a procesar, el análisis del mismo, la transformación de los datos resultantes, la estimación de los movimientos y la devolución de los datos al usuario a través de la interfaz gráfica.

El usuario podrá elegir verificar los datos o guardar directamente un archivo con los resultados. Realizado esto, el usuario podrá volver a la pantalla inicial para seleccionar otro video para continuar con el proceso.

La solución se implementará de manera que se puedan aprovechar al máximo los recursos del sistema, permitiendo una reducción en el tiempo global que toma procesar un experimento.

En los siguientes puntos, se explicarán los datos con los que se va a trabajar y se detallará los pasos seguidos para el desarrollo de la solución propuesta.

Para explicar la implementación de la propuesta, se empezará por los componentes internos del módulo de obtención de datos, luego se detallaran los procesos de transformación de datos y finalmente el desarrollo de la aplicación completa con interfaz.

5.1. Descripción de los datos

Como se ha comentado en la introducción de este trabajo, la búsqueda de mejoras en las condiciones animales en los procesos de crianza ha fomentado la investigación en diferentes marcadores biológicos relacionados con el estrés.

El PRAN (Grupo de Investigación de Producción Animal)[33], perteneciente a la ETSIAAB (Escuela Técnica Superior de Ingeniería Agronómica, Alimentaria y de Biosistemas), realiza investigaciones en estas líneas, siendo una principal el bienestar animal en peces, usando experimentos como el *NetTest*.

A partir de estas labores anteriores de investigación, se han podido obtener materiales para este trabajo, siendo principalmente videos del experimento junto a sus etiquetas. Para obtener estos datos, en sucesivos trabajos del PRAN se ha utilizado la herramienta *BORIS*[3], que ha permitido anotar diferentes momentos en los que ha sucedido un movimiento.

En concreto, el grabado y etiquetado manual se llevó a cabo por los técnicos Juan Enrique Barrios Sánchez[2] y Almudena Gallego Fernández, bajo la supervisión de los investigadores Álvaro De la LLave-Propín y Morris Villarroel Robinson.

Por lo tanto, el etiquetado manual no siempre se ha llevado a cabo por la misma persona, con lo que puede existir un factor de subjetividad que añade error humano a la medición de los movimientos de los peces.

5.1.1. Conjunto de datos disponibles: videos

Los videos se pueden separar en 2 subconjuntos dependiendo del momento en el que se realizó la grabación, lo cual se ve reflejado en la calidad de la misma y la facilidad para identificar el movimiento del pez a lo largo de los fotogramas.

Videos con truchas jóvenes

Para este tipo de videos se cuenta en total con un conjunto de 6 videos completos de un experimento *NetTest* sobre todos los peces de un grupo, pero no se tiene más información relevante sobre los especímenes que aparecen en los videos aparte de que son relativamente jóvenes.

Se tratan de videos en formato `16:9`, con resolución `1920x1080`, con la cámara posicionada fija verticalmente encima de una mesa donde se lleva a cabo el experimento. Esto se puede ver en la Figura 19. En este caso, el experimento se lleva a cabo con redes negras, lo cual puede ser problemático teniendo en cuenta que los peces tienen tonalidades similares al marrón y gris.

Figura 19: Aspecto de los videos con truchas jóvenes

Las especificaciones de la cámara en el momento de grabación son desconocidas, pero a través de los detalles de los archivos se observa que son archivos con extensión `MTS (MPEG Transport Stream)`, grabados a 25 FPS y con una tasa de datos por segundo de `16.96 Mbps`.

En estos videos, se realizan el experimento 5 veces con 2 peces a la vez para maximizar el tiempo. Por lo tanto, puede ser necesario segmentarlo manualmente.

Como último punto importante, en estos videos se puede observar que se realiza la variación del *NetTest* que dura 1 minuto aproximadamente.

Videos con truchas adultas

Este es el conjunto de datos más amplio y detallado. Cuenta en total con 4 experimentos *NetTest*, en los que se realiza el experimento a 10 jaulas en total (10 peces por jaula en parejas). Cada jaula consta de un video que muestra el experimento entero y luego una serie de subvideos que los investigadores de PRAN han sacado recortando el original para eliminar momentos en los que no hay ningún experimento en pantalla. Esta estructura final se puede ver en la Figura 20.

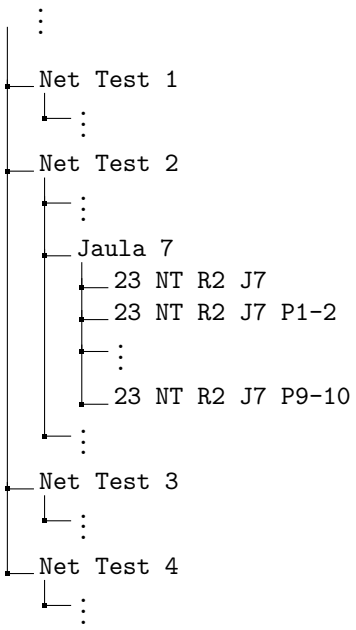


Figura 20: Estructura de los videos de truchas adultas

Los videos mantienen las características de formato en `16:9`, con resolución `1920x1080` y cámara apuntando verticalmente a la mesa del experimento (Figura 21), sin embargo, podemos ver que se ha pasado a usar una red verde que aporta cierto contraste respecto al cuerpo del pez. Aparte de esto, el marco de la red también ha cambiado y tiene un color plateado. En otros experimentos como los del *NetTest*3 también se observa el uso de redes negras.

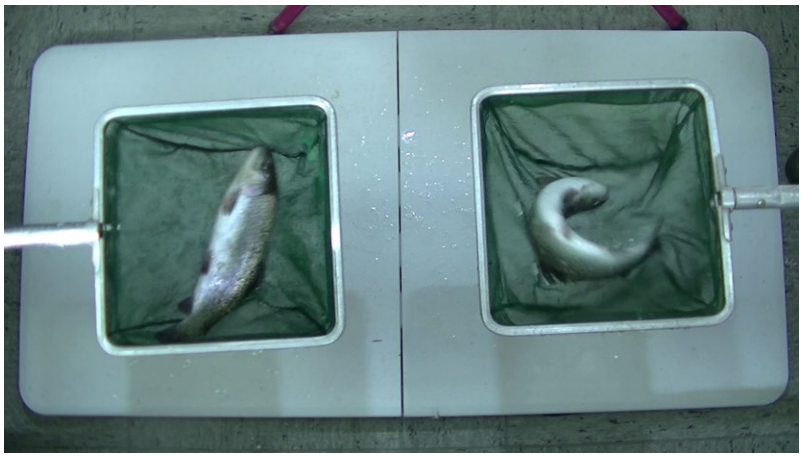


Figura 21: Aspecto de los videos de truchas adultas

Las especificaciones de la cámara para estos videos siguen siendo desconocidas, pero se puede asumir que los parámetros han cambiado por el aumento de la luminosidad de la imagen. Además, se ha movido un poco la colocación de la mesa y la línea negra queda justo en el medio entre las dos truchas con las que se realiza el experimento.

Respecto a datos técnicos del video, el contenedor ha cambiado a `MOV (QuickTime File Format)`, se mantienen los 25 FPS y la tasa de datos aumenta hasta los `21 Mbps`.

Los videos obtenidos en este caso también contienen videos de experimentos en grupo, es decir, varios experimentos a varias parejas de truchas. Sin embargo, a diferencia de los videos de truchas jóvenes, en este caso los experimentos *NetTest* duran 15 segundos aproximadamente.

5.1.2. Conjunto de datos disponibles: etiquetas

Por motivos ajenos a este trabajo, no se cuentan con las etiquetas manuales correspondientes a los videos de truchas jóvenes. Esto implica que no se tiene un marco de referencia para contar los movimientos de los videos más antiguos.

Videos de truchas adultas

Existen dos archivos `Excel`, el primero de ellos contiene toda la información sobre los experimentos:

- Chip de cada individuo del experimento por grupo, pez y *NetTest*.
- Pez de cada grupo y *NetTest*.
- Todos los fotogramas en los que cada pez ha realizado un movimiento, tomando como referencia el fotograma 0 del video no recortado.
- Momento en el tiempo del video en el que sucede cada movimiento, tomando como referencia el tiempo del video no recortado.

De todos los datos, los útiles van a ser los referentes a los movimientos, que se pueden ver en la Figura 22. Esto se debe a que los datos sobre el chip no nos aportan información.

NET TEST 1 GRUPO 1			NET TEST 1 GRUPO 2			NET TEST 1 GRUPO 3		
Subject	Time	Image index	Subject	Time	Image index	Subject	Time	Image index
Pez 1	23080	578	Pez 1	22240	557	Pez 1	23240	582
Pez 1	23680	593	Pez 1	22800	571	Pez 1	23480	588
Pez 1	23960	600	Pez 1	23120	579	Pez 1	24000	601
Pez 1	24240	607	Pez 1	23400	586	Pez 1	24240	607
Pez 1	24520	614	Pez 1	25240	632	Pez 1	24560	615
Pez 1	24760	620	Pez 1	26600	666	Pez 1	24880	623
Pez 1	25120	629	Pez 1	26840	672	Pez 1	25160	630
Pez 1	25480	638	Pez 1	28320	709	Pez 1	27640	693
Pez 1	28400	711	Pez 1	28760	720	Pez 1	28000	701
Pez 1	28600	716	Pez 1	32360	810	Pez 1	28320	709
Pez 1	29400	736	Pez 1	32600	816	Pez 1	28720	719
Pez 1	29680	743	Pez 1	32960	825	Pez 1	28960	725

Figura 22: Columnas del `Excel` con los datos de los fotogramas en los que sucede un movimiento

El segundo archivo `Excel` nos proporciona información ya procesada por los investigadores del PRAN, en los cuales aparecen los movimientos finales totales para cada pez según grupo y número de experimento *NetTest*. Un ejemplo de una sección del archivo se puede ver en la Figura 23.

GRUPO	CHIP	INDIVIDUO NT1	AYUNO NT1	MOVIMIENTOS TOTALES NT1	INDIVIDUO NT2	AYUNO NT2	MOVIMIENTOS TOT
1	7B8BE17	Pez 1	1	18	Pez 10	1	8
1	79475DF	Pez 2	1	18	Pez 5	1	18
1	7B8C3F7	Pez 3	1	19	Pez 2	1	20
1	7B8BF9D	Pez 4	1	12	Pez 1	1	19
1	7944FAD	Pez 5	1	20	Pez 4	1	17
1	7946643	Pez 6	1	15	Pez 7	1	17
1	7947A29	Pez 7	1	10	Pez 3	1	16
1	79464D4	Pez 8	1	18	Pez 8	1	16
1	7B9C003	Pez 9	1	23	Pez 9	1	19
1	79447B2	Pez 10	1	11	Pez 6	1	18
2	7946EAE	Pez 1	1	14	Pez 8	1	14
2	7B8C35C	Pez 2	1	20	Pez 10	1	14
2	7826B2F	Pez 3	1	20	Pez 9	1	6

Figura 23: Sección del archivo de movimientos finales para los videos de truchas adultas

Finalmente, es necesario indicar que se han realizado grandes esfuerzos para entender la estructura y el significado de los datos etiquetados proporcionados para permitir su uso en la validación de resultados más adelante en el documento.

6. DESARROLLO DE LA SOLUCIÓN

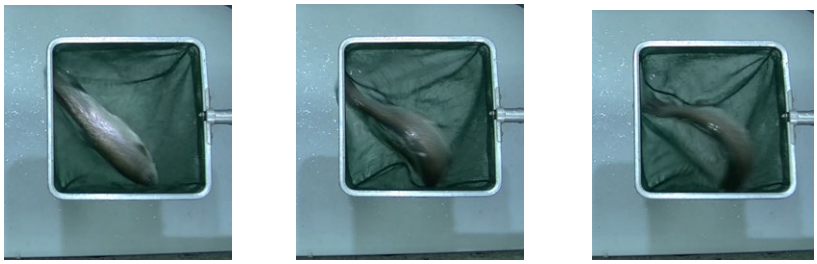
6.1. Pruebas de concepto: `OpticalFlow` y `YOLO`

El desarrollo del módulo de procesado del video afectaba al tratamiento de datos que se iba a realizar posteriormente, por esto mismo se decidió tratarlo como el primer problema a solucionar.

Dentro de este módulo se busca automatizar la obtención de datos y parametrización del movimiento de las truchas dentro del video.

En este sentido se plantearon dos alternativas para su desarrollo usando diferentes tecnologías:

1. **Solución basada en *OpticalFlow***: La idea general de este método es caracterizar el movimiento de la trucha respecto al fondo de la imagen.
   Si tenemos en cuenta que el análisis por `OpticalFlow` nos da como resultado una serie de vectores que representan el cambio en los píxeles de la imagen, cabe la posibilidad de obtener una serie de vectores globales que represen el movimiento de los bordes de la trucha entre fotogramas. La representación de esta idea se puede ver en la Figura 24.



(a) Fotogramas de entrada

(b) Fotogramas con el fondo eliminado

(c) Representación de los vectores que se buscan obtener

Figura 24: Idea general basada en `OpticalFlow` para el módulo de procesamiento del video

   Como se puede observar, la idea es obtener un conjunto de vectores en el contorno del pez que nos permitan saber diferentes datos:

   - Si se ha movido o no: a través de la suma de todos los vectores y la obtención del módulo del vector global. Si este valor es mayor que cierto umbral, se podría indicar que está sucediendo un movimiento entre los dos fotogramas.

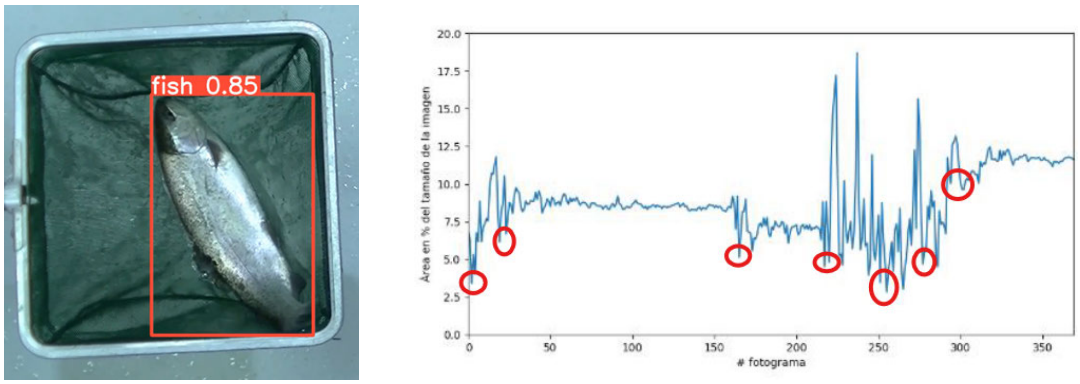
- Hacia donde se ha movido: a través del análisis de la dirección del vector global se puede indicar el sentido del movimiento que está ocurriendo.

2. **Solución basada en el uso de *YOLO* como herramienta de análisis**: A través del uso de una `CNN` se procesa el video, obteniendo información sobre los objetos detectados como truchas.
   Para realizar esto, hay que definir la tarea que realizaría la red:

   - Clasificación: no es útil, ya que no aporta información sobre la posición de los objetos detectados en la imagen.
   - Segmentación: puede ser útil, pero para entrenar la red en esta tarea, los datos etiquetados deben ser segmentos de la imagen. Esto puede llevar demasiado tiempo y; como se verá más tarde, no es la única tarea que puede aportar información para la automatización.
   - Detección: es la más interesante, ya que nos permite conseguir unos tiempos de procesado por imagen muy bajos a la vez que nos da datos relacionados con la posición y el tamaño aproximado de una caja rectangular que envuelve al pez.

   Teniendo en cuenta lo anterior, lo más viable es entrenar un modelo `YOLO` en tarea de detección de truchas. Esto es realizable a través de la creación de un conjunto de datos de imágenes de truchas y sus respectivas etiquetas, que en este caso son dos esquinas de la caja (`Bounding Box`) que engloba el objeto que se quiere detectar.
   La red neuronal entrenada podra ser capaz de devolver datos sobre las truchas detectadas y las posiciones y tamaños de las cajas que las engloban.

   A través de estas `Bounding Boxes`, podemos parametrizar un movimiento como la reducción del área o movimientos de la posición de la `Bounding Boxes`.



(a) Imagen procesada por `YOLO`

(b) Idea de estimación de movimiento a través de los resultados de las `Bounging Boxes`, marcando en rojo los movimientos

Figura 25: Idea general basada en `YOLO` para el módulo de procesamiento del video

Hay que tener en cuenta que sería necesario hacer una validación de los entrenamientos realizados para conseguir cumplir con las necesidades. Esto es debido a que, al contrario que el `OpticalFlow`, las redes neuronales no son deterministas y dependiendo del fotograma sobre el que se realice inferencia, podemos obtener resultados alejados de los esperados.

Para decidir qué método podía dar mejores resultados, se probaron las diferentes tecnologías aplicadas en el video `23_NT_R1_J1_P7_8.mp4`. Este video dura 15 segundos y contiene dos peces, uno a la izquierda y otro a la derecha.

6.1.1. Pruebas de concepto con `OpticalFlow`

Para realizar las pruebas se utilizó el entorno de `MATLAB` en la versión `2023B`. En esta aplicación se disponen de 2 funciones principales que implementan análisis de flujo óptico:

- Método de Horn-Schunck: método de análisis denso (aplicado para todos los píxeles de la imagen).
- Método de Lucas-Kanade: método de análisis local (aplicado a áreas de píxeles que se asumen que tienen el mismo movimiento).

Ambas pruebas demostraron que los videos con los que se estaba trabajando tenían una tasa de fotogramas demasiado baja para la cantidad de movimiento que podía ocurrir entre fotogramas. Esto se puede observar en el flujo óptico percibido cuando suceden cambios bruscos del pez como en la Figura 26 para el método Horn-Schunck y en la Figura 27 para el método Lucas-Kanade.

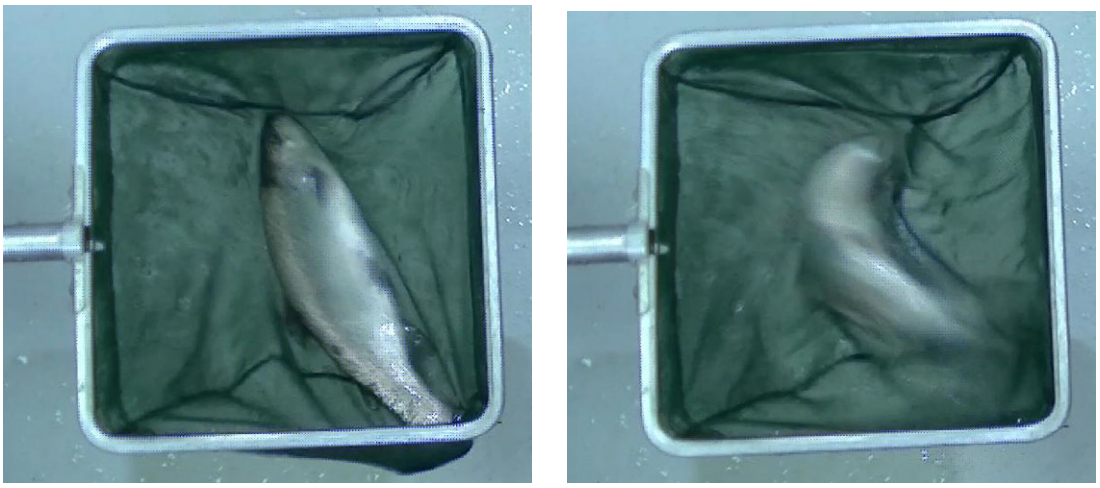


Figura 26: Flujo óptico detectado entre dos fotogramas por el método Horn-Schunck en los videos del *NetText*

Figura 27: Flujo óptico detectado entre dos fotogramas por el método Lucas-Kanade en los videos del *NetText*

El principal problema del método Horn-Schunck es que se ve muy afectado por el ruido, en el caso de este experimento, ese ruido aparece por la poca `tasa de fotogramas` y la falta de `bitrate` que tiene el video en comparación con la velocidad de movimiento que tienen las truchas. Aparte de esto, es un método computacionalmente muy costoso. En el caso de este video de 15 segundos tardó

cerca de 10 minutos en realizar el análisis completo del flujo óptico, lo cual incumple el requisito 18.

Vemos cierta mejoría usando el método de Lucas-Kanade, pero al realizar el análisis de forma local en zonas con similitudes, en cuanto la trucha hace un movimiento, el análisis deja de funcionar sobre la parte de la imagen en la que se sitúa la trucha. Aún siendo bastante rápido (sin llegar a ser tiempo real), esta situación elimina la posibilidad de utilizar este método para la parametrización del movimiento de la trucha.

En ambos métodos se ha comprobado que mejorarían los resultados aplicando umbralizaciones a la imagen. También sería necesario un preprocesado para delimitar la sección de imagen sobre la que es de interés realizar el análisis de flujo óptico (la zona de la red que contiene la trucha). Aún con lo comentado anteriormente, el único método que puede aportar algún tipo de información de manera consistente para todo el video es el método de Horn-Schunck.

6.1.2. Pruebas de concepto con `YOLO`

Para realizar esta prueba, creo un conjunto de datos con el que entrenar a través de fotogramas del video `23_NT_R1_J1_P1_2.mp4`. Esto se realizó a través de la herramienta `CVAT (Computer Vision Annotation Tool)`. Esta herramienta de código abierto[34] es mantenida por los creadores de la librería `OpenCV`. Dispone de una versión online limitada al número de archivos que se pueden subir a `500 MB` y en otros aspectos, pero también dispone de una versión desplegable por `Docker`. Uno de sus puntos fuertes y el motivo de su uso en este trabajo es la compatibilidad con muchos formatos de exportación de etiquetas.

Como parte de este proyecto y como sistema que sirviese de soporte para guardar todos los conjuntos de datos necesarios, se desplegó como contenedor en el servidor NAS (Network Attached Storage) del GAMMA (Grupo de Aplicaciones Multimedia y Acústica).

Con esta aplicación se realizó el etiquetado de 24 imágenes para realizar un primer entrenamiento, el conjunto de datos se dividió de la siguiente manera:

- 16 imágenes para el conjunto de entrenamiento.
- 4 imágenes para el conjunto de validación.
- 4 imágenes para el conjunto de pruebas.

Posteriormente se realizó el entrenamiento de la red neuronal `YOLOv8` versión `nano`, que es la más pequeña y rápida de entrenar. Los detalles en profundidad de resultados de este entrenamiento se pueden observar en el anexo B.

Se consiguió unos buenos resultados y al inferir sobre el video `23_NT_R1_J1_P7_8.mp4`, se observó que el seguimiento de la trucha se conseguía incluso en situaciones críticas donde la trucha realizaba el movimiento como en la Figura 28. Sin embargo, situaciones como las de la Figura 29,en donde la trucha tenía formas suficientemente extrañas o se ocultaba detrás del marco de la red resaltaron la falta de entrenamiento y la necesidad de expandir el conjunto de datos.

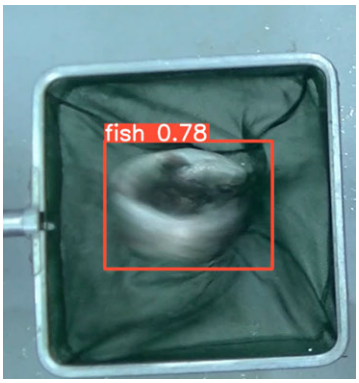
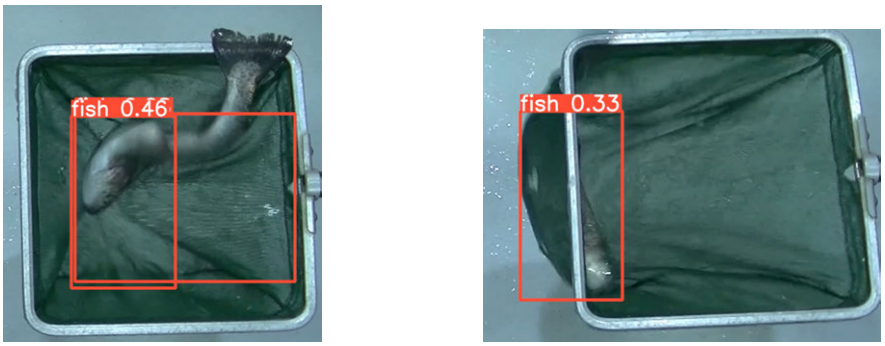


Figura 28: Correcta detección en la prueba de concepto de `YOLO` en situaciones de movimiento



(a) Detección de poca confianza y de baja calidad en el primer entrenamiento

(b) Detección de poca confianza por ocultación de la trucha

Figura 29: Situaciones negativas en la prueba de concepto de `YOLO`

Aún con los problemas varios que se pudieron observar, se observó que en una `GPU GTX 1060 6GB` de `NVidia` el procesado del video y la obtención de los resultados solo tardaba `10 ms` por fotograma de video. Esto permitiría un procesado menor al tiempo del video hasta con 100 FPS.

Esta prueba de concepto demuestra la viabilidad del uso de `YOLO` como herramienta para obtener datos sobre las truchas existentes en el video de forma rápida.

Teniendo en cuenta que la mayoría de los movimientos observables de las truchas se basan en coletazos, podemos determinar que la trucha cuando busca realizar un movimiento, tendrá que comprimir su cuerpo en un menor área y después volver a estirar el cuerpo. Esto es directamente caracterizable con los datos de las `Bounding Boxes` obtenidos usando `YOLO`.

6.1.3. Análisis de las pruebas de concepto

En los anteriores puntos se presentaron dos alternativas:

- **OpticalFlow**: aunque es una solución que idealmente debería funcionar, es incapaz de tener la suficiente consistencia con los videos usados en este trabajo, que tienen demasiado movimiento entre fotogramas. Aparte de esto, un punto negativo severo es que el único algoritmo que consigue sacar algo de información vectorial sobre el flujo en las zonas del pez es de tipo denso, tardando una relación de 40 veces la duración del video en procesar el resultado. Esto incumple el requisito de acelerar el experimento para los investigadores del *NetTest*.
- **YOLO**: aún con situaciones con problemas de detección, pueden ser en mayor grado solucionado con mayor entrenamiento y expansión del conjunto de datos. Como la tarea que realiza son solo de detección, alcanza unas velocidades que permiten la viabilidad del proyecto.

Por lo anterior, se ha decidido finalmente descartar la solución basada en `OpticalFlow` y utilizar un modelo `YOLO` como base para el análisis y obtención de datos del video.

Gracias a estas pruebas de concepto, se realizaron unas gráficas representativas sobre el cambio de área y centroide para demostrar la viabilidad de contar movimientos con este sistema. Esto se utilizó para el aporte en un artículo de conferencia sobre el mismo tema llevado a cabo por el investigador Álvaro de la LLave-Propín[35].

6.2. Entrenamiento progresivo de los diferentes modelos de YOLO

En la librería de ultralytics, existen dos modelos de detección que se busca soportar en el desarrollo de la solución para el procesado de los videos:

- Detección tradicional con Bounding Boxes con ejes paralelos a los ejes de la imagen.
- Detección con OBB (Oriented Bounding Boxes), que es capaz de minimizar el área que define al pez, pero tarda más en realizar el procesamiento y tiene resultados en distinto formato.

Habitualmente, la detección tradicional, al no tener que lidiar con rotación, es capaz de ser mucho más rápida en el tiempo de inferencia, pero la posibilidad de desarrollar el sistema soportando redes de tipo OBB puede permitir manejar mejores precisiones en un futuro.

Al usar la herramienta de etiquetado CVAT se observó que la exportación a etiquetas para un modelo YOLO OBB no eran funcionales. Estas etiquetas requieren para cada imagen un archivo .txt que contenga todas las instancias de los objetos en la imagen y los cuatro puntos que definen el cuadro que lo delimita.
Para poder entrenar la red OBB, se usó una herramienta disponible en un repositorio de GitHub[36] que transformaba etiquetas en formato COCO (formato estándar que se estructura en un XML) a las etiquetas necesarias.

Antes de seguir con el desarrollo, es necesario explicar los estadísticos que se obtienen de realizar un entrenamiento con un modelo YOLO. En el caso de este trabajo, se realizará una explicación de los resultados que devuelve un modelo de detección.

Las métricas que se tienen en cuenta durante el entrenamiento son:

- **Box Loss sobre el conjunto de entrenamiento**: representa el error del tamaño y posición de las Bounding Boxes de las etiquetas sobre las predichas en cada época. Cuando menor sea, mayor es la cercanía entre las cajas predichas en el conjunto de entrenamiento respecto a sus etiquetas reales.
  Para calcularlo se utilizan métricas como la intersección entre unión, parámetro que indica cuanta parte del área coincide entre bounding box real y predicha. Su funcionamiento se puede observar en la Figura 30.

Figura 30: Factor IoU para calcular la perdida de cajas en una CNN de detección

  Aparte de esto, se utilizan otras métricas como la distancia, el ángulo de libertad respecto a la bounding box esperada y una medición de perdida en la relación alto/ancho de la caja.
- **Cls Loss sobre el conjunto de entrenamiento**: representa el error de clasificación de cada objeto detectado en la imagen respecto al de la etiqueta real en el conjunto de entrenamiento.
- **Dfl Loss sobre el conjunto de entrenamiento**: es un error sobre la capacidad del modelo YOLO para soportar deformaciones de los objetos detectados.
- **Precisión (B)**: indica la precisión general sobre los objetos detectados, para realizar este cálculo se tienen en cuenta todas las métricas anteriores como una media con pesos y condiciones. Idealmente es 1 como máximo.
- **Recall (B)**: indica la capacidad que se ha tenido para detectar todas las instancias, se utiliza después para saber cuantos elementos detecta con una precisión determinada. Idealmente es 1.
- **Métrica mAP50 (B)**: es una métrica sobre la precisión media en elementos con una intersección entre unión de 0.5. Esta métrica se puede entender como la capacidad de detección

en elementos fáciles de forma correcta. Es bastante representativa en el entrenamiento para saber si algo malo esta sucediendo.

- **Métrica mAP50-95 (B)**: igual que antes, representa una media ponderada de precisión, pero en este caso de los elementos con una intersección entre unión en el rango de `0.5` a `0.95`. Podemos considerar esto como las detecciones dificiles y ajustadas a los resultados que esperamos.
- **Box Loss sobre el conjunto de validación**: igual que el `Box Loss` sobre el conjunto de entrenamiento, pero en este caso, al no variar el conjunto de validación, se va tomando como referencia entre épocas para saber si el entrenamiento esta siendo correcto.
- **Cls Loss sobre el conjunto de validación**: igual que en la situación de entrenamiento pero como métrica entre épocas en un conjunto de datos invariante.
- **Dfl Loss sobre el conjunto de validación**: igual que en la situación de entrenamiento pero como métrica entre épocas en un conjunto de datos invariante para la precisión en deformación.

Estas métricas idealmente van acercándose a valores aceptables según pasan las épocas. Esto puede ser guardado al finalizar un entrenamiento de forma automática por parte de `YOLO` como se puede ver en el ejemplo de la Figura 31 para 100 épocas.

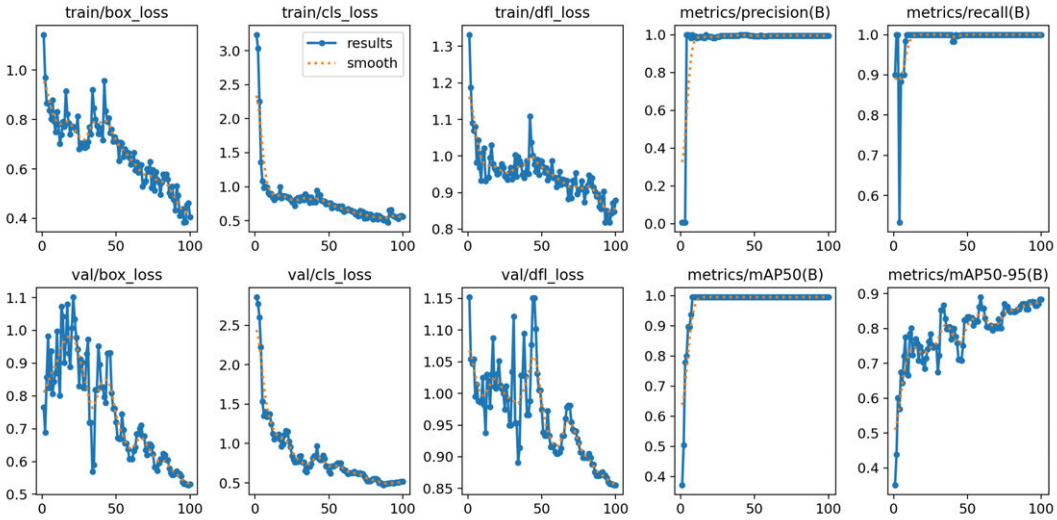


Figura 31: Ejemplo de imagen de resultados resumidos por parte de `YOLO`

Adicionalmente, `YOLO` produce resultados adicionales, que dependiendo del objetivo de la red, se buscará maximizar uno u otro:

- **Matriz de confusión**: es una matriz como la de la Figura 32,que muestra el número de objetos detectados y su clasificación, si el objeto está correctamente clasificado será un verdadero positivo, si se piensa que hay objeto, pero no hay, será falso positivo y si no se detecta, pero existe será falso negativo. Finalmente se tiene la situación de un verdadero negativo si no hay y no se detecta un objeto. Al ser un estadístico de clasificación por naturaleza, no es el más apropiado para el análisis de resultados en detección de objetos, y menos cuando solo hay 1 clase de objetos.

Figura 32: Ejemplo de matriz de confusión en la medicina

- **Curva de puntuación F1**: representa la puntuación F1, que se calcula a través de

$$\text{Puntuación F1} = \frac{2 * Precision * Recall}{Precision + Recall}$$

siendo el recall calculado como:

$$\text{Recall} = \frac{TP}{TP + FN}$$

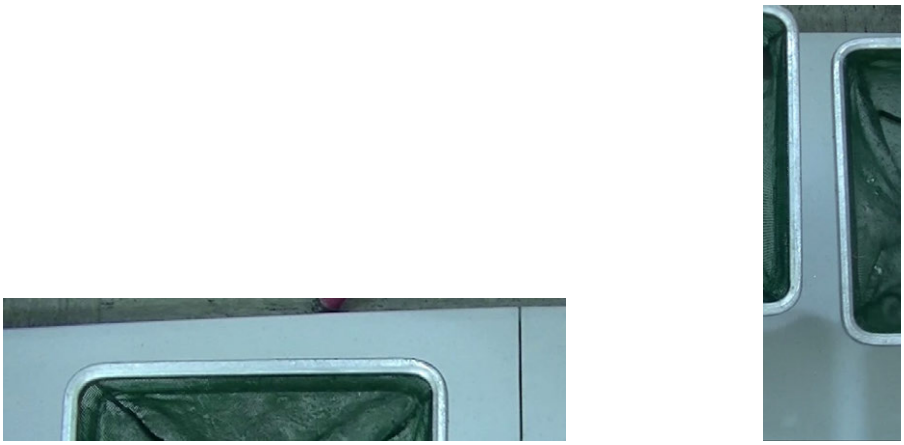
Cuanto mayor área tenga bajo la curva mejor, porque significará que hay más verdaderos positivos en precisiones altas respecto a falsos.

- **Curva de precisión contra recall**: muestra la relación entre precisión y recall, mostrando una relación entre los valores seleccionados para seleccionar umbrales.
- **Curva de precisión**: refleja para cada valor de confianza de clasificación de elemento la precisión media de esa asignación. Cuanto más plana y cercana al 1 sea, mejor. Idealmente sería un escalón.
- **Curva de recall**: muestra para cada valor de confianza, el recall medio en ese valor. Idealmente es una línea horizontal en `y=1`.

En este proyecto, se busca obtener buenos resultados en el entrenamiento para ciertos estadísticos. Siendo el más importante la precisión de la `Bounding Box`, ya que queremos que se ajuste lo máximo posible al mínimo tamaño necesario. También nos interesa mucho las métricas de precisión y obtener un área máxima bajo la curva `F1`, ya que tiene en cuenta la precisión y, si conseguimos buenos resultados en la parte alta para la confianza, implicará que tenemos un buen número de verdaderos positivos respecto a falsos positivos con buena precisión. Al trabajar con una sola clase, los estadísticos relacionados con el ámbito de clasificación no son tan relevantes mientras se cumpla un mínimo de clasificación en las detecciones.

Estas métricas van a ser muy importantes para las siguientes páginas donde se detallan el entrenamiento progresivo del modelo.

Lo primero que se realizó fue un pequeño aumento de datos. Como se buscaba que la red generalizase mejor el concepto de trucha, se añadieron imágenes sin truchas como las de la Figura 33, en la que apareciesen la mesa vacía y partes de la red vacías.



(a) Imagen de esquina vacía del segundo conjunto

(b) Imagen del área entre las truchas añadida al segundo conjunto

Figura 33: Imágenes extra para el segundo conjunto sin truchas

Esto permitió al modelo entrenado tener mejor capacidad de generalización de las truchas, además de tener menos falsos positivos (ver anexo B: entrenamiento 2). En otro entrenamiento posterior aumentando las épocas hasta 300, se observó que se podía seguir reduciendo el error asociado al ajustado de las `Bounding Boxes` (ver anexo B: entrenamiento 3), por lo tanto para los siguientes entrenamientos se decidió trabajar con un número de épocas mayor o igual a 300.

Posteriormente, se decidió usar el conjunto de videos con truchas jóvenes del *NetTest* para expandir el conjunto de datos. Esto se debía a 3 razones principales:

1. **Aumento de datos y creación de un modelo más robusto**: el usar imágenes como las de los videos con truchas jóvenes permitiría una capacidad alta al modelo de detectar truchas bajo diferentes condiciones de luz y de diferentes tamaños (las de esos videos son mucho más pequeñas). Además de esto, los movimientos de los peces en estos videos eran más difíciles de analizar por su falta de detalle, mejorando la cantidad de datos que representaban movimiento.
2. **Evitar Overfitting para futuros entrenamientos**: el `overfitting` es una situación conocida en el entrenamiento de redes neuronales en las que las redes son entrenadas de forma excesiva, aprendiendo el ruido del conjunto de datos. Esto se ve representado por la incapacidad del modelo de predecir correctamente sobre datos que no haya visto en ningún momento del entrenamiento. No es una situación que deba suceder, pero cuanto más variado sea el conjunto de datos, menos posibilidades hay de llegar a esta situación.

En total se añadieron 34 imágenes de los videos con truchas jóvenes, dando como resultado un conjunto de datos de 84 imágenes que se dividieron en subconjuntos entrenamiento, validación y prueba con la relación 70-15-15.

Con este conjunto de datos se realizaron 2 entrenamientos(ver anexo B), uno con un modelo `YOLO` de detección tradicional y otro a un modelo `OBB`, el cual se tuvo que realizar en una herramienta llamada `Kaggle` (se decidió usar esta plataforma porque da flexibilidad y potencia de GPU).
Ambos modelos dieron buenos resultados en detectar todas las situaciones de la trucha completamente. El modelo OBB se observó que tenía menos confianza en las predicciones de manera general, lo cual necesitaría mejorarse con mejores datos, etiquetas y ajuste de hiperparámetros.

Más tarde se descubrió que no se había tenido en cuenta un factor importante: el color de la red. Este factor se vio cuando se intentó realizar inferencia sobre un video del *NetTest* número 2, donde la red era completamente negra y había fotogramas como el de la Figura 34 donde generalizar el pez era demasiado complicado para el modelo entrenado.

Figura 34: Fotograma de ejemplo en videos con red negra

Para solucionar este problema y como entrenamiento final, se decidió hacer un último aumento del conjunto de datos. Para esto se tomaron imágenes aleatorias de los videos `23_NT_R2_J1_P1_P2`, `23_NT_R3_J1_P1_P2` y `23_NT_R3_J2_P1_P2`. En total el conjunto de datos fue aumentado hasta los 203 fotogramas. Esto sumado a expandir el entrenamiento hasta las 600 épocas, permitió crear un modelo entrenado muy robusto para todo tipo de iluminaciones, cambios de redes y de variaciones de fondo.

Para poder aprovechar los diferentes recursos del sistema, se trabajó con 3 formatos de redes neuronales en los que YOLO es capaz de trabajar:

- **PyTorch**: formato muy estandarizado y con soporte muy amplio. Es capaz de trabajar en CPU, GPU e incluso en dispositivos especializados para multiplicaciones de matrices como las TPU (Tensor Processing Unit). Su principal problema es que no está optimizado realmente para funcionar en CPU. Obteniendo muy malos tiempos por fotograma. Su uso ideal es sobre las GPU de NVidia, aprovechando el uso de operaciones CUDA para hacer cálculos con matrices.
- **ONNX**: plataforma de traducción abierta entre modelos, ofrece mucho mejor soporte a distintos HardWare y ofrece mejor rendimiento si se quiere inferir sobre cpu.
- **OpenVino**: plataforma propietaria de Intel, que optimiza redes neuronales para todo el HardWare de su marca. Las redes neuronales exportadas en este formato son capaces de aprovechar operaciones SIMD (Single Instruction/Multiple Data), consiguiendo aprovechar GPUs de Intel y mejorar mucho el rendimiento.

En un ordenador con un i7-8700 y una GTX 1060 6GB se han visto los resultados para cada modelo mostrados en la Tabla 5.

	tiempo (ms) por fotograma
PyTorch-CPU	60
PyTorch-CUDA	10
ONNX	40
OpenVino	13

Tabla 5: Diferencia de tiempo de inferencia por formato de modelo de YOLO

Estos formatos de exportación de modelo se integrarán en la aplicación para conseguir acelerar el proceso de inferencia en el ordenador del usuario dependiendo de sus recursos HardWare.

6.3. Análisis del número de movimientos en el video

Se buscó desarrollar un sistema secuencial como el representado en la Figura 35, donde se utiliza `YOLO` para procesar los videos y detectar en cada fotograma una `Bounding Box` de la trucha. Posteriormente estos datos son procesados y, a través de la aplicación de un algoritmo, se detecta el número de movimientos que han ocurrido y cuando han sucedido.

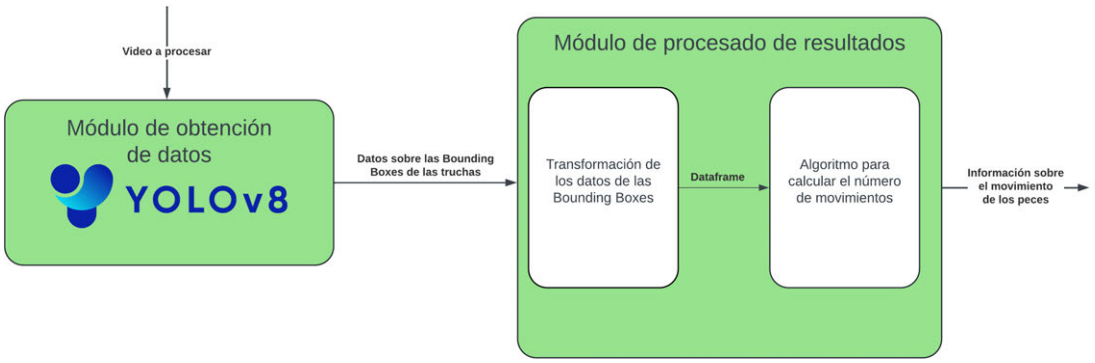


Figura 35: Diagrama del primer sistema secuencial desarrollado

Para plantear el procesamiento, primero se deben entender que datos es capaz de devolver un modelo `YOLO`, su formato y seleccionar los mas relevantes. Esto depende en gran medida del tipo de modelo que se esté usando.

6.3.1. Obtención del número de truchas en el video

Los modelos de detección habitual de `YOLO` devuelven un objeto `ultralytics.engine.results`, cuyo contenido relevante para este trabajo ha sido:

- **Boxes**: objeto con información sobre los elementos detectados, tamaño, posición y confianza de clasificación.
- **OBB**: en este caso es `None` por que no se realiza este tipo de tarea.
- **Imagen original**: objeto que contiene la imagen original de ese fotograma.
- **Tamaño de la imagen**: diccionario de dos elementos con el tamaño de la imagen de tipo `(ancho, altura)` expresado en píxeles.

Sin embargo, los modelos que se basan en detectar `Bounding Boxes` orientadas devuelven:

- **Boxes**: en este caso `None`
- **OBB**: objeto con la información sobre las detecciones, igual que `Boxes` en contenido pero estructurado de forma diferente.
- **Imagen original**.
- **Tamaño de la imagen**.

De los elementos anteriores que contienen la información de las `Bounding Boxes`, se deciden utilizar ciertos elementos respectivamente:

- **Boxes**: Este objeto contiene un elemento llamado `xywh` conteniendo la posición `x` e `y` del centro de cada `Bounding Boxes` en la imagen respecto a la esquina superior izquierda. Además contiene el ancho `w` y el alto `h` de cada una. Es de tipo `Tensor` internamente.
- **OBB**: Este objeto contiene un elemento llamado `xywhr` igual que el elemento anterior, pero al ser cajas orientadas, se añade un elemento extra que representa la rotación de la `Bounding Box`.

Una vez se obtienen todos estos elementos de un video, es necesario saber el pez al que pertenece para `Bounding Box` detectada. Esto es importante, ya que estos sistemas pueden detectar varias instancias de un objeto en la misma posición de forma superpuesta. Un ejemplo de esto se ve en la Figura 36. En este caso es esperable obtener 1 datos por fotograma, por lo tanto se decide la mejor `Bounding Box`.

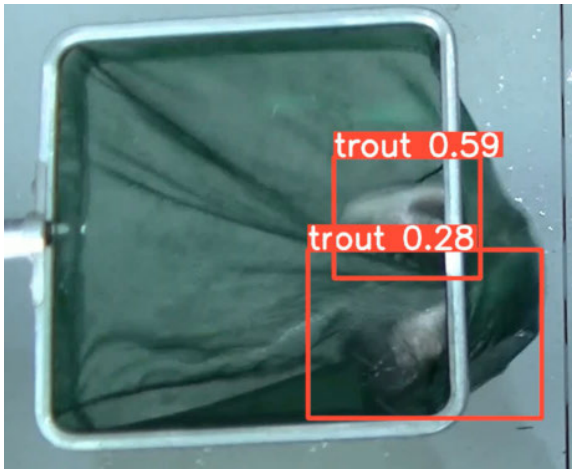


Figura 36: Presencia de varias `Bounding Boxes` superpuestas para una trucha en el experimento

Para saber cuantos peces aparecían en el video, se utilizó la media de la posición `x` de las primeras 50 muestras para analizar el punto medio de la imagen entre los datos. Posteriormente se calcula la desviación de las muestras para analizar si la desviación es mayor a un límite según la resolución del video. Este sistema permite saber si hay 1 o 2 truchas en el experimento, añadiendo flexibilidad al sistema.

Posterior a esto, se realiza una transformación de datos para construir un objeto `DataFrame` de la librería `Pandas`, usada en la ciencia de datos por su capacidad de manejar muchos datos de forma extremadamente rápida, ya que internamente utiliza el lenguaje de programación `C`.
Para cada fotograma, se selecciona la `Bounding Box` que ha tenido mayor confianza por la red para cada zona en la que existe una trucha. Cada índice de este `dataframe` representa el fotograma y el contenido de cada columna será un diccionario de `Python` con la información relevante. Esto se puede ver en la Figura 37.

	left	right
0	{'area': 4.711131154755015, 'centroideX': 546....	{'area': 9.103529188368055, 'centroideX': 1455...
1	{'area': 4.265890615957755, 'centroideX': 647....	{'area': 9.082215862509644, 'centroideX': 1455...
2	{'area': 6.3515949013792445, 'centroideX': 724...	{'area': 9.18430507330247, 'centroideX': 1455....
3	{'area': 5.046920964747299, 'centroideX': 721....	{'area': 9.392859188126929, 'centroideX': 1458...
4	{'area': 5.258187376422647, 'centroideX': 705....	{'area': 9.394917052469136, 'centroideX': 1449...
..	...	...
385	{'area': 6.180908203125, 'centroideX': 591.0, ...	{'area': 5.243815104166667, 'centroideX': 1399...
386	{'area': 6.150444878472222, 'centroideX': 592....	{'area': 5.901150173611111, 'centroideX': 1436...
387	{'area': 5.980278862847222, 'centroideX': 594....	{'area': 5.5546875, 'centroideX': 1440.0, 'cen...
388	{'area': 5.997395833333333, 'centroideX': 593....	{'area': 8.189643012152779, 'centroideX': 1447...
389	{'area': 6.020345052083333, 'centroideX': 593....	{'area': 3.891072591145833, 'centroideX': 1317...

[390 rows x 2 columns]

Figura 37: Datos estructurados procesados por fotograma y por trucha para las `Bounding Boxes`

De forma más concreta, por cada trucha y por cada fotograma se guarda:

- Área de la `Bounding Box` en porcentaje respecto a la resolución de la imagen.
- Posición `X` del centroide de la `Bounding Box`.
- Posición `Y` del centroide de la `Bounding Box`.
- Relación del ancho entre la altura de la `Bounding Box`.
- Ángulo de rotación (Si no es detección OBB, será 0).
- Valor del `Blur`: Se guarda la varianza del filtro laplaciano sobre la sección donde se encuentra la trucha. Esto es interesante porque un movimiento de una trucha implica una imagen más borrosa. A través del filtro laplaciano podemos buscar bordes, y cuantos menos aparezcan en la imagen, menor será la varianza.

6.3.2. Obtención de los movimientos de cada trucha

El `DataFrame` con todos los datos se procesa de la siguiente manera:

1. Para cada columna de trucha, se realiza un nuevo `DataFrame` que almacene en columnas separadas los valores de área, centroides, `blur`, etc. Esto es para no tratar con diccionarios.
2. Se calcula el gradiente de la columna del área. Esto nos permite ver el cambio entre fotogramas de área y es uno de los puntos importantes a la hora de buscar movimientos.
3. Se calcula el cambio absoluto de la posición del centroide de las `Bounding Boxes` entre fotogramas.

Con el procesamiento de los datos realizados, se puede aplicar el algoritmo para buscar movimientos, que utiliza el siguiente pseudocodigo:

Algorithm 1 Estimación de movimiento

```
GradienteTotalSubconjunto ← suma de los gradientes del subconjunto
if gradienteTotalSubconjunto ≥ −1 then
    return no movimiento
else if todos los valores del gradiente del subconjunto ≥ −0,8 then
    return posibilidad de movimiento
else if 0,3 ≤ Todos los valores de la relacion x/y del subconjunto ≤ 2,7 then
    if El mayor área del subconjunto ≥ mediana de los datos · 0,9 then
        return posibilidad de movimiento
    else
        resultado ← Fotograma del subconjunto con menor varianza del Blur
        return resultado
else
    return posibilidad de movimiento
```

Al algoritmo se le pasan todos los subconjuntos de fotogramas en los que el gradiente ha sido menor que 0, siendo necesario que sean subconjuntos mayores a 1 elemento y menores a 9. Esto ayuda a eliminar situaciones en las que la `Bounding Box` cambia mucho entre fotogramas sin ningún movimiento de la trucha. Posteriormente, dentro del algoritmo se verifican los datos para eliminar situaciones que pueden ser errores o pueden no ser representativas de un movimiento.

6.4. Desarrollo de aplicación completa

El siguiente objetivo del trabajo fue desarrollar una aplicación completa que permitiese a los usuarios obtener los resultados de forma fácil. Esto fue decidido principalmente por dos razones:

- Era necesario cumplir el requisito que afecta al nivel de conocimiento necesario para manejar la herramienta. Con una aplicación que tuviese interfaz gráfica, se podría manejar mucho mejor que obligar al usuario a usar una consola de comandos o a conocer como funcionar las diferentes dependencias de las librerías usadas.
- Servía como oportunidad para mejorar los servicios que era capaz de proporcionar el sistema, además de poder diseñar la aplicación para poder aprovechar de forma eficiente los recursos del `HardWare`.

Para desarrollar esta aplicación primero se seleccionó la librería de GUI (Graphical User Interface). Este paso fue importante porque existen muchas posibles librerías en el entorno de `Python`, pero cada una aporta diferentes utilidades y abstracciones, siendo las más comunes:

- `TKinter`: es el `framework` más usado en `Python`. Es una librería simple y que no se suele usar para manejar datos multimedia, ya que es muy simple.
- `PyQT`: librería que internamente usa el `framework QT`. Es mucho más potente y capaz de manejar videos y elementos complejos.
- `Kivy`: librería pensada para el desarrollo de aplicaciones móviles.
- `PySimpleGUI`: es una librería `wrapper` sobre `Tkinter` y similares, por lo tanto es muy simple, pero limita mucho.
- `Remi`: se usa para crear interfaces web.
- `DearPyGUI`[37]: librería muy reciente que se centra en dar al usuario la máxima eficiencia posible, usando aceleración `HardWare` a través de `OpenGL`.

Al estar trabajando con redes neuronales, queremos que la interfaz gráfica utilice la menor cantidad de recursos posible y de la manera más eficiente. Además, como se verá en los siguientes puntos, se implementaron partes multimedia en la aplicación, por lo tanto fue necesaria una librería centrada en la eficiencia, lo cual dejaba como posibles opciones `PyQT` y `DearPyGUI`, sin embargo por motivos personales del autor, la librería seleccionada fue `DearPyGUI`.

La selección se debe a la experiencia del autor en otros trabajos con la misma librería para realizar aplicaciones de manejo de cámaras IP. Al conocer ya la librería, se reducirían los posibles problemas y la necesidad de mirar documentación. Además de esta experiencia pasada, quedó patente que `DearPyGUI` es capaz de manejar los recursos del ordenador de forma muy eficiente.

6.4.1. Diseño de interfaz para la aplicación

Para realizar el diseño de la aplicación, se diseñó a través de la herramienta `DrawIO`, un esquema general del flujo y ventanas de la aplicación (completo en el anexo c).
Usando este esquema como idea general, se fue desarrollando la aplicación poco a poco, desde la ventana inicial hasta la ventana de datos

- **Ventana inicial de la interfaz**: esta ventana tiene como objetivo que el usuario seleccione el video sobre el que quiere realizar la detección automática de movimiento, su aspecto se puede ver en la Figura 38.

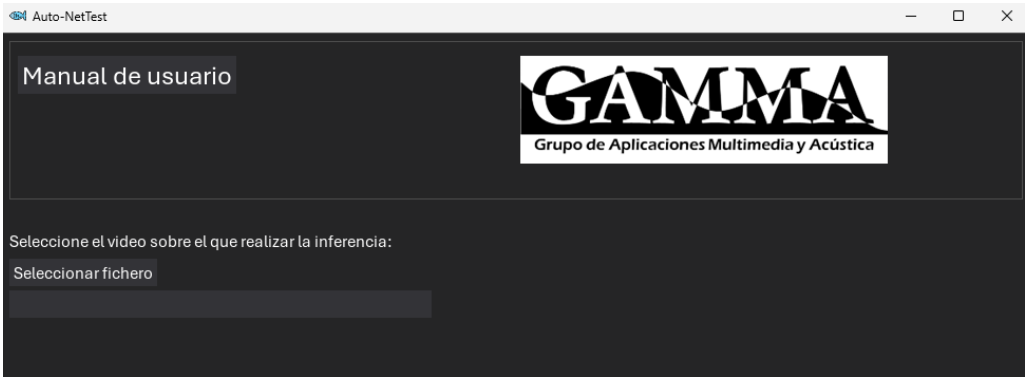


Figura 38: Pantalla inicial sin video seleccionado

Esta ventana contiene un botón que permite abril una ventana modal que contiene un manual de usuario. Esto se implementó para que las personas que usen esta aplicación no necesiten tener el documento de este trabajo a mano. El aspecto de la ventana del manual se puede observar en la Figura 39.

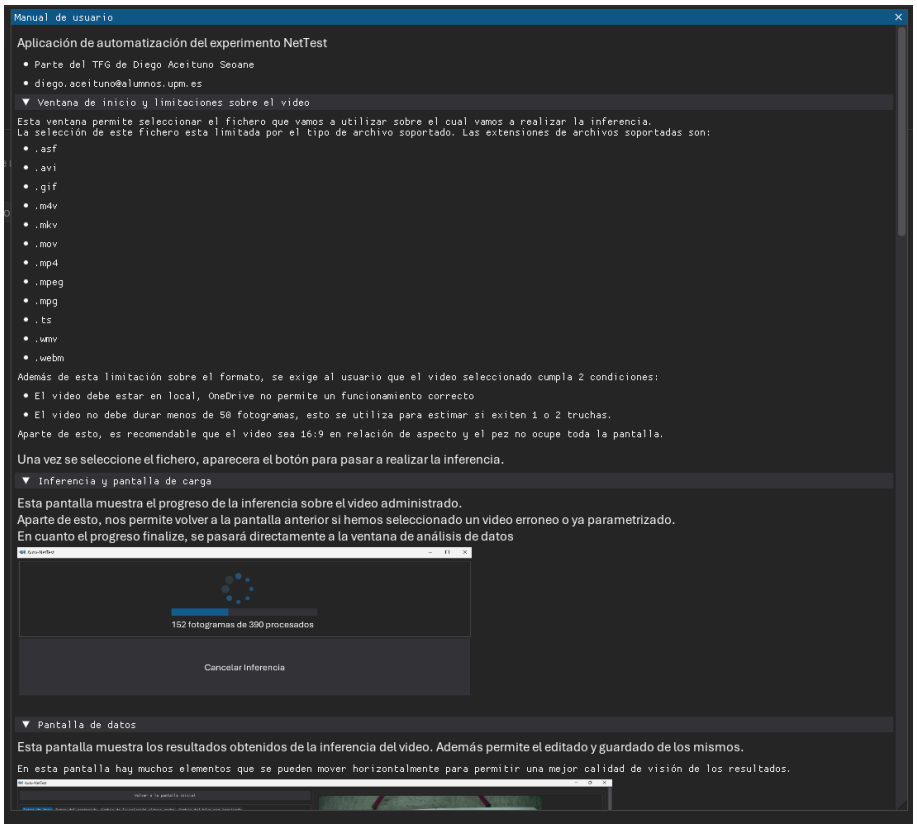
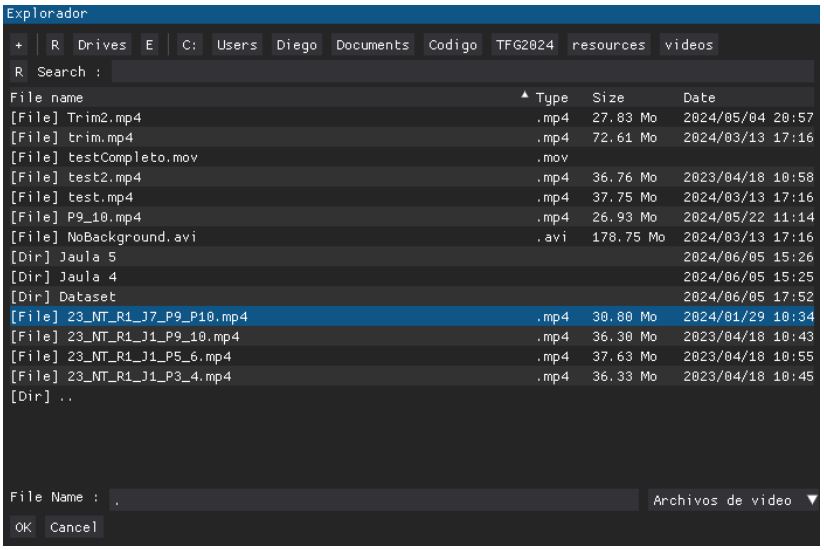


Figura 39: Aspecto del manual contenido en la pantalla inicial de la aplicación

El usuario puede selecciona a través de un explorador de archivos, el fichero (debe ser un fichero soportado, lo cual se indica en el manual) sobre el que quiere realizar la inferencia. También puede utilizar el explorador para cancelar la selección.

En el momento que el fichero se selecciona, aparece un botón de inferencia como se ve en la Figura 40. Cuando se pulsa el botón, se realizan una series de comprobaciones sobre el video para ver sí es compatible, en caso contrario la inferencia no se realizará y se volverá al estado inicial.



(a) Aspecto del explorador de archivos para seleccionar video

(b) En cuanto se selecciona el fichero, aparece el botón para iniciar el procesado

Figura 40: Flujo de selección de video

- **Ventana de inferencia**: esta ventana sirve como pantalla de carga mientras se procesa el video, como se ve en la Figura 41 se implementó junto con un botón para cancelar por dos motivos:
  - Permitir al usuario conocer el estado del procesamiento del video y que la aplicación no se quede estática.
  - Permitir al usuario cancelar el proceso si se ha equivocado de video seleccionado.

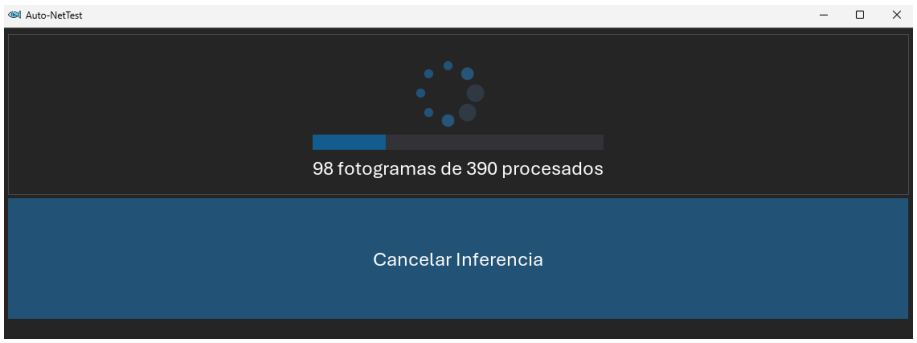


Figura 41: Aspecto de la pantalla de carga de la aplicación

- **Ventana de datos**: esta ventana tiene tres propósitos:
  - Mostrar un análisis por gráficas sobre las `bounding boxes` de las truchas de cada lado, eliminando o añadiendo elementos de la GUI dependiendo de sí hay 1 o 2 truchas en el video. En concreto se ha diseñado para mostrar 2 conjuntos de gráficas, el primer conjunto contiene gráficas para cada uno de los datos como el área, centroide, relación de tamaño y `blur` para las 2 truchas. El segundo conjunto tiene una gráfica para cada trucha donde se representan la gráfica del área, centroide y `blur` al mismo tiempo.
  - Mostrar el número de movimientos para cada trucha.
  - Validar los resultados: se quería permitir que los usuarios pudiesen verificar los movimientos que han sucedido, para esto se han creado dos líneas de tiempo, una para cada trucha. A través de ellas y un reproductor del video integrado en la aplicación, se pueden verificar los movimientos que han sucedido, pudiendo añadir o quitar movimientos.
    Esta funcionalidad se realiza añadiendo la capacidad de que el usuario pueda pinchar en las líneas de tiempo, sirviendo como referencia del fotograma donde se quiere añadir el movimiento. Para que el usuario sepa donde está, se marca con una línea vertical.

  Los datos para rellenar las gráficas se obtienen cuando finaliza la inferencia previa y su transformación de datos, como se explicó en una sección anterior.
  Elementos como las líneas de tiempo se han creado a base de unir gráficas y crear series discretas según los datos obtenidos. Estas series y gráficas se actualizan en tiempo de fotograma, además de ser ajustadas en altura para aprovechar al máximo la pantalla del usuario y para no tener que depender de una aplicación a pantalla completa.

  El aspecto de esta ventana se puede verificar en la Figura 42

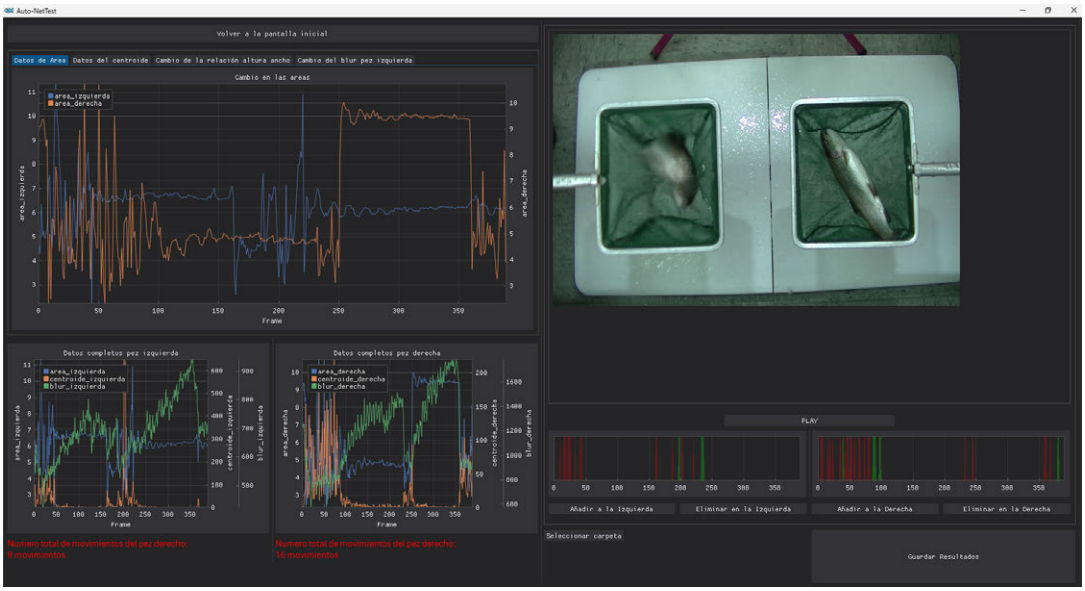


Figura 42: Aspecto de la ventana de datos

6.4.2. Implementación de las funcionalidades a través de procesos

Para poder implementar todas las funcionalidades de forma correcta, se necesitaba realizar de forma paralela diferentes tareas. Para esto se utilizó la librería `Multiprocessing` de `Python`.

El uso de `Multiprocessing` es debido a que la implementación de hilos en `Python` a través de la librería `Threading` no es como se espera. Esto es debido a que `Python` es un lenguaje interpretado, lo que implica que hay una aplicación que va analizando los ficheros de código y ejecutando operaciones.
Para realizar esas tareas de forma segura respecto a la aplicación padre, `Python` implementa un `GIL (Global Interpreter Lock)`, el cual evita que haya más de un hilo en ejecución a la vez.

Las implicaciones del GIL son en resumen:

- El uso de hilos en `Python` se limita a situaciones especiales donde haya mucho `I/O`, ya que no son capaces de funcionar en paralelo.
- Para poder obtener concurrencia real, se necesita usar librerías que manejen varios intérpretes de `Python` a través de un lenguaje con mayor control como `C` o `C++`. La librería `Multiprocessing` realiza esto, manejando los cambios de contexto y la relación entre procesos.

El uso de `multiprocessing`, al contrario que el uso de `threading` implica que no hay memoria compartida entre procesos, por lo tanto hay que pensar la forma para comunicar información entre procesos. Es un punto muy importante, ya que pueden aparecer problemas de concurrencia por falta de operaciones atómicas (1 línea de código de alto nivel no se ejecuta de golpe en el procesador).

Para la comunicación entre procesos existen muchas técnicas y sistemas ya existentes. En este trabajo se ha decidido utilizar técnicas basadas en colas. Aún sin ser el mecanismo que podría ser más eficiente en sobrecarga de memoria, es extremadamente fácil de usar.

Una cola es un mecanismo de bajo nivel que consiste en un `buffer` de tipo `FIFO (First In First Out)` que comunica diferentes procesos como se puede ver en la Figura 43. Permite operaciones de varios procesos productores y varios procesos consumidores. Además de esto, maneja los problemas de concurrencia de manera que el acceso para introducir y adquirir elementos se controle para todos los procesos, teniendo llamadas bloqueantes y métodos de `polling` sobre la cola.

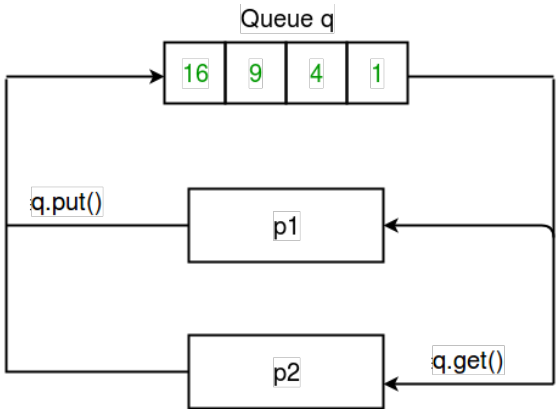


Figura 43: Funcionamiento de una cola para datos de procesos

En la librería de `multiprocessing`, este mecanismo está completamente implementado de dos formas:

- `Queue`: la cola que se ha explicado anteriormente, con varios productores y consumidores.
- `Pipe`: una cola que solo gestiona 1 productor y 1 consumidor. Es hasta 3 veces más rápida que una `Queue`, y por las tareas en las que se va a usar, es el mecanismo perfecto para comunicar procesos en este trabajo. Cada objeto de esta clase cuenta con dos `enpoints`, uno para entrada de datos y otro para salida (si esto no se respeta saltan excepciones).

En este trabajo se utilizó `multiprocessing` para las siguientes tareas:

1. **Realizar la inferencia**:
   Cuando se creó una solución secuencial de prueba en anteriores secciones, se observó que procesar el `blur` de la imagen al final de procesar el video obligaba a acumular todas las imágenes descomprimidas del video. Esto efectivamente creaba un pico en el uso de memoria RAM (Random Access Memory) (para un video de 15 s, era capaz de utilizar `3GB`), lo cual era extremadamente problemático.

   Para poder realizar la inferencia sin que la aplicación se quedase colgada y poder crear una pantalla de carga se implementó un sistema, en el que al momento de procesar un video, se crean dos procesos en el programa principal a la vez que se crean dos objetos `Pipe`, comunicados como se ve en la Figura 44:

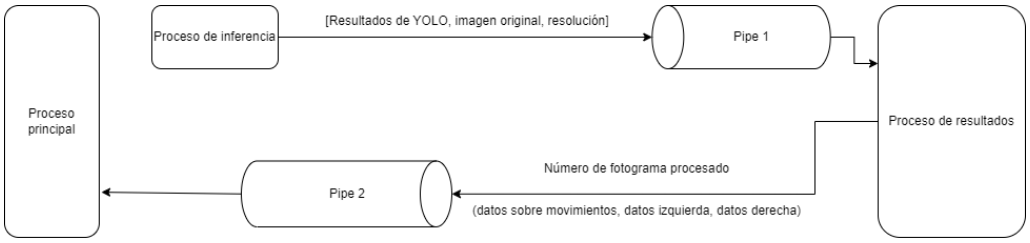


Figura 44: Flujo de datos entre procesos para la inferencia

   - Proceso de inferencia: va realizando la inferencia sobre el video fotograma a fotograma en modo streaming con el modelo `YOLO` entrenado en secciones anteriores. Antes de ser arrancado, a este proceso se le pasa el `endpoint` que le permite introducir datos hacia el proceso para resultados. Para cada fotograma introduce en el `endpoint` un `array` que contiene: [resultados del `YOLO`, imagen original, resolución de la imagen]. Para poder serializar correctamente los resultados de `YOLO` se utilizó la librería `Pickle` de `Python`.
   - Proceso para resultados: este proceso va eliminando los mensajes que obtiene por el `endpoint` de salida en el que el proceso de inferencia va mandando los resultados y los acumula en una variable propia. Cuando hay 50 mensajes, realiza la estimación del número de truchas que hay en el video, y a partir de aquí va procesando los resultados (incluyendo el cálculo del `blur`). Esto permite que las imágenes solo se guarden en memoria `RAM` el tiempo que estén en la `Pipe`. A la par que se realiza esto, a través del `endpoint` de entrada hacia el proceso principal, se va indicando el fotograma por el que va el proceso de inferencia, lo cual permite rellenar la barra de carga para que el usuario pueda verlo. Cuando todos los resultados han sido procesados, en el `endpoint` que se comunica con el proceso principal se manda un diccionario con el resultado del cálculo de movimientos y los datos de las truchas.

   Una vez se ha acabado de procesar el video, se cambia de pantalla y se matan los anteriores procesos hijos creados.

   En el anexo c se puede encontrar un diagrama de secuencia con más detalle de estas operaciones.

2. **Mostrar el video en la pestaña de datos**:
   Para permitir que el usuario pudiese editar los movimientos que han sucedido, se implementó una utilidad de reproducción de video. Esto se consigue añadiendo una textura a la interfaz y actualizándola en cada bucle de renderizado. Para conseguir esto se implementó un proceso que controla la reproducción del video internamente manejando peticiones desde el hilo principal. Para esto se utilizan dos `Pipe`, uno del proceso principal al controlador de video, a través del cual se mandan indicaciones para comenzar la reproducción, parar y saltar a otra posición del video.
   Estas comunicaciones se pueden ver en la Figura 45.

   Antes de entrar en la ventana de datos, se usa la primera imagen del video (esta se guardó al verificar los 50 fotogramas mínimos de duración antes de la pantalla de carga) para cargar en la textura.

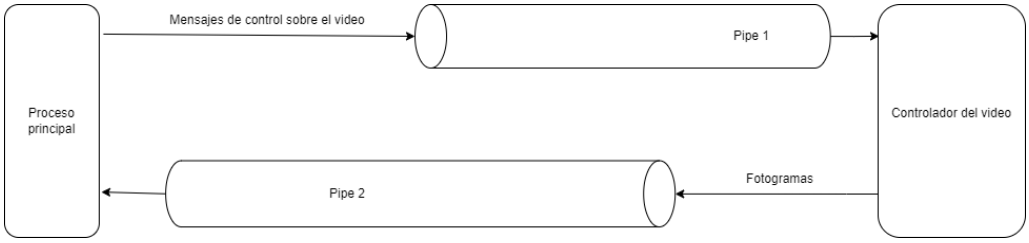


Figura 45: Flujo de información entre el proceso principal y el controlador de video

   El proceso principal puede mandar como señales:

   - True: indica que se debe reproducir el video para obtener fotogramas.
   - False: indica que se debe pausar el video
   - Un valor entero positivo: indica una posición en fotograma al que se debe saltar en el reproductor. Esta es la parte que permite al usuario verificar los diferentes movimientos.

   El proceso controlador de video tomará como referencia temporal $t = 0{,}8/fps$ y, sí se encuentra en modo reproducción, irá obteniendo, procesando y mandando los fotogramas en ese ritmo. Para todo esto se aprovecha de la librería `OpenCV`.

   En el anexo c se pueden encontrar los diagramas de secuencia que se asocian a estas situaciones.

Aparte de todo lo anterior, se pueden encontrar todas las referencias de métodos, clases y módulos en el diagrama de clases general de la aplicación, el cual se encuentra en el anexo c de este documento.

6.4.3. Guardado de los datos y despliegue de la aplicación

Como parte del sistema global, se espera poder almacenar la información detectada en un fichero. Para realizar esto se añadió un selector de carpeta y un botón de guardado de resultados, como se puede ver en la Figura 46.

Figura 46: Widgets relacionados con el guardado

Para permitir flexibilidad en el manejo de los datos, se guardan a la vez dos archivos:

- Archivo CSV (Comma Separated Values): este tipo de archivos se pueden abrir directamente con editores de texto plano y son compatibles con Excel.
  Manejan datos con un conjunto de separaciones por comas en horizontal, y saltos de líneas en vertical. En este caso se guarda la estructura de la Figura 47, no siendo obligatorio los datos de la trucha de la derecha (esto ocurre cuando hay solo una trucha).

```
23_NT_R1_J7_P9_P10.mp4                          <---Nombre del video
9                                               <---Número de movimientos en la trucha izquierda
9,19,23,33,41,162,188,202,221                   <---Fotogramas con movimiento en la trucha izquierda
16                                              <---Número de movimientos en la trucha derecha
8,17,20,24,34,42,45,54,57,67,74,80,233,246,361,368 <---Fotogramas con movimiento en la trucha derecha
```

Figura 47: Formato de los resultados guardados en CSV

- Archivo JSON (JavaScript Object Notation): este tipo de archivos son semi-estructurados y funcionan muy bien, ya que se usan en todos los lenguajes de programación. En este caso la estructura que se guarda es la de la Figura 48.

```
{
    "video_name": "23_NT_R1_J7_P9_P10.mp4",
    "left": {
        "movement_number": 9,
        "movement_frames": [
            9,
            19,
            23,
            33,
            41,
            162,
            188,
            202,
            221
        ]
    },
    "right": {
        "movement_number": 16,
        "movement_frames": [
            8,
            17,
            20,
            24,
            34,
            42,
            45,
            54,
            57,
            67,
            74,
            80,
            233,
            246,
            361,
            368
        ]
    }
}
```

Figura 48: Formato de los resultados guardados en JSON

Si no se selecciona una carpeta con el selector, se guardará por defecto en la carpeta raíz que contenga el módulo principal o el ejecutable.
Para añadir velocidad, se ha asumido que cada nombre de video es único (esto se indica en el manual de la aplicación), esto nos permite que el archivo guardado sea simplemente añadir la extensión al nombre original. Por ejemplo, si tenemos el video `video1.mp4`, los archivos de guardado seran `video1.mp4.csv` y `video1.mp4.json`.

Finalmente, para desplegar la aplicación se utilizó la herramienta `PyInstaller`. Esta herramienta permite crear ejecutables que consiguen encapsular la aplicación de manera que el usuario no necesite manejar las dependencias de la aplicación. Además de esto, dentro de la aplicación inserta el intérprete de `Python`.

Su manera de funcionamiento es, para cada vez que se abra la aplicación, crear un entorno virtual. Y se puede crear un ejecutable creando:

- Un ejecutable: contiene todo dentro, por lo tanto también realiza compresión. Suele pesar mucho menos, pero tarda en abrirse, ya que crea en una carpeta temporal del usuario el entorno de `Python`.
- Una carpeta con un ejecutable dentro: funciona de la misma manera que lo anterior, pero las dependencias vienen ya descomprimidas, por lo tanto no le hace falta crear una carpeta temporal y se abre mucho más rápido, sin embargo pesa casi el doble habitualmente.

En el caso se han creado las dos versiones, con los siguientes comandos, el primero para crear un ejecutable completo, y el segundo para una carpeta (cambiando –onefile por –onedir):

```
pyinstaller --onefile --add-data="C:/Users/Diego/Documents/Codigo/
    TFG2024/.venv/Lib/site-packages/ultralytics":"ultralytics/" --
    add-binary="C:/Users/Diego/Documents/Codigo/TFG2024/.venv/Lib/
    site-packages/openvino/libs":"." --hidden-import openvino --
    collect-submodules openvino --collect-binaries openvino --
    collect-data openvino --hidden-import onnx --hidden-import
    onnxruntime .\main.py

pyinstaller --onedir --add-data="C:/Users/Diego/Documents/Codigo/
    TFG2024/.venv/Lib/site-packages/ultralytics":"ultralytics/" --
    add-binary="C:/Users/Diego/Documents/Codigo/TFG2024/.venv/Lib/
    site-packages/openvino/libs":"." --hidden-import openvino --
    collect-submodules openvino --collect-binaries openvino --
    collect-data openvino --hidden-import onnx --hidden-import
    onnxruntime .\main.py
```

Ambas instrucciones cargan los runtimes necesarios (como el de `CUDA`, `ONNX` y `OpenVino`) para que la aplicación pueda aprovechar al máximo el `HardWare` y detectar cuál formato de modelo de red neuronal utilizar.

7. VALIDACIÓN DE RESULTADOS

Como proyecto de ingeniería, se debe llevar a cabo un análisis de cumplimiento de los requerimientos del sistema desarrollado a través de métricas. Para la validación se van a evaluar las diferentes partes del trabajo.

Primeramente, a través del entrenamiento de la red neuronal, se ha buscado minimizar el error asociado a las `Bounding Boxes`, ya que el sistema usa esto como base para calcular movimientos.

En este sentido, con el último conjunto de datos y 600 épocas, se han conseguido alcanzar un error de perdida de la `bounding box` de `0.2` y `0.9` en el conjunto de datos de entrenamiento y validación respectivamente, siendo este resultado uno de los mejores de todos los entrenamientos y con un conjunto de validación mucho más realista. Los resultados de la validación se pueden ver en la Figura 49.

(a) Etiquetas del conjunto de validación final

(b) Predicciones sobre el conjunto de validación final

Figura 49: Resultados finales sobre el conjunto de validación

Aparte de los resultados de validación, como ya se ha dicho en el trabajo y se puede ver en las gráficas de resultados del anexo c, se buscaba maximizar otras métricas como son el `recall` a niveles altos para obtener una cantidad elevada de verdaderos positivos. Finalmente, también se ha conseguido alcanzar una precisión `mAP50-95` muy elevada (0.82) teniendo en cuenta la variabilidad del conjunto de datos, lo cual nos indica una precisión general de la red muy buena.

Seguidamente, se realizó la validación del número de movimientos utilizando las etiquetas proporcionadas por los investigadores como ya se comentó en el análisis de los datos. Para esto, se utilizó la aplicación en el conjunto de videos del *NetText*1 al *NetText*4, a través de las jaulas 1 a la 4, excluyendo los videos que contenían el primer y el segundo pez (ya que para entrenamiento se han usado los videos `23_NT_R1_J1_P1_2`, `23_NT_R2_J1_P1_P2`, `23_NT_R3_J1_P1_P2` y `23_NT_R3_J2_P1_P2`).

En total se han procesado 64 videos, los cuales eran completamente nuevos para la red neuronal. Este procesamiento sin la aplicación conllevaría el gasto de tiempo por parte de los investigadores de varias semanas anotando manualmente movimientos. A través de la aplicación los videos se han conseguido procesar en menos de una hora.

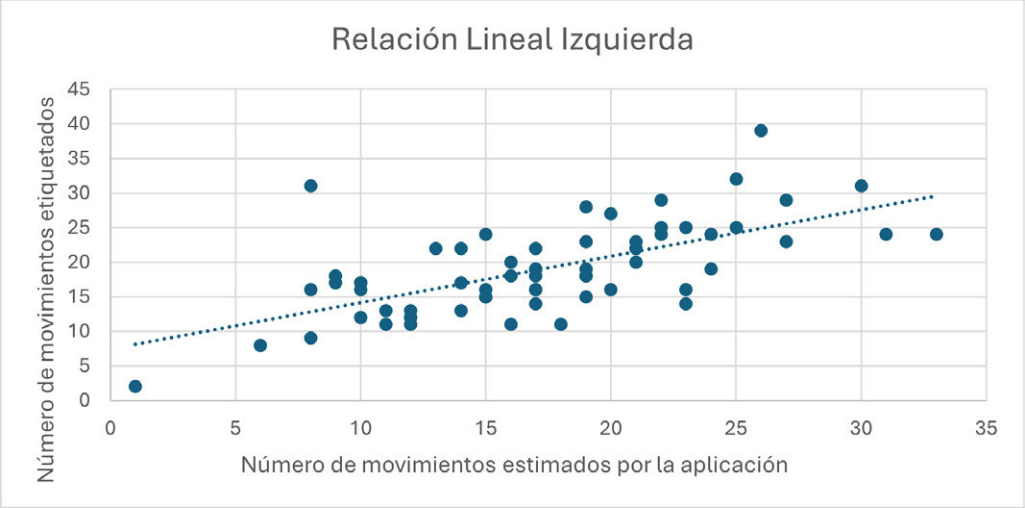
Los resultados por video y por pez se pueden observar en la Figura 50.

Nombre del video	izquierda Aplicación	Izquierda etiqueta	Derecha Aplicación	Derecha etiqueta
23_NT_R1_J1_P3_4	12	12	17	19
23_NT_R1_J1_P5_6	19	15	24	20
23_NT_R1_J1_P7_8	19	18	11	10
23_NT_R1_J1_P9_10	12	11	24	23
23_NT_R1_J2_P3_4	19	19	19	20
23_NT_R1_J2_P5_6	25	25	26	24
23_NT_R1_J2_P7_8	16	20	22	19
23_NT_R1_J2_P9_10	21	22	18	12
23_NT_R1_J3_P3_4	9	18	26	21
23_NT_R1_J3_P5_6	14	17	21	32
23_NT_R1_J3_P7_8	15	15	23	24
23_NT_R1_J3_P9_10	13	22	9	9
23_NT_R1_J4_P3_4	31	24	21	17
23_NT_R1_J4_P5_6	17	16	17	17
23_NT_R1_J4_P7_8	22	25	14	20
23_NT_R1_J4_P9_10	21	20	16	17
23_NT_R2_J1_P3_4	9	17	15	16
23_NT_R2_J1_P5_6	17	18	14	18
23_NT_R2_J1_P7_8	15	16	15	17
23_NT_R2_J1_P9_10	6	8	16	19
23_NT_R2_J2_P3_4	8	9	17	21
23_NT_R2_J2_P5_6	21	23	20	17
23_NT_R2_J2_P7_8	23	14	11	15
23_NT_R2_J2_P9_10	17	14	10	6
23_NT_R2_J3_P3_4	10	17	22	30
23_NT_R2_J3_P5_6	18	11	28	14
23_NT_R2_J3_P7_8	33	24	21	26
23_NT_R2_J3_P9_10	10	17	24	23
23_NT_R2_J4_P3_4	23	25	14	18
23_NT_R2_J4_P5_6	24	19	29	23
23_NT_R2_J4_P7_8	27	29	17	13
23_NT_R2_J4_P9_10	23	16	12	20
23_NT_R3_J1_P3_4	17	16	14	28
23_NT_R3_J1_P5_6	8	31	21	16
23_NT_R3_J1_P7_8	10	16	25	29
23_NT_R3_J1_P9_10	19	28	10	17
23_NT_R3_J2_P3_4	8	16	10	15
23_NT_R3_J2_P5_6	15	15	13	14
23_NT_R3_J2_P7_8	17	19	15	12
23_NT_R3_J2_P9_10	24	24	31	32
23_NT_R3_J3_P3_4	22	24	23	22
23_NT_R3_J3_P5_6	25	32	25	26
23_NT_R3_J3_P7_8	20	27	17	27
23_NT_R3_J3_P9_10	17	22	18	25
23_NT_R3_J4_P3_4	14	13	21	19
23_NT_R3_J4_P5_6	12	13	3	3
23_NT_R3_J4_P7_8	12	12	17	20
23_NT_R3_J4_P9_10	11	11	9	7
23_NT_R4_J1_P3_4	1	2	18	10
23_NT_R4_J1_P5_6	15	24	15	21
23_NT_R4_J1_P7_8	14	22	20	24
23_NT_R4_J1_P9_10	22	29	22	23
23_NT_R4_J2_P3_4	27	23	24	28
23_NT_R4_J2_P5_6	20	16	21	19
23_NT_R4_J2_P7_8	19	23	26	23
23_NT_R4_J2_P9_10	30	31	13	17
23_NT_R4_J3_P3_4	16	18	19	24
23_NT_R4_J3_P5_6	17	19	25	24
23_NT_R4_J3_P7_8	26	39	18	15
23_NT_R4_J3_P9_10	11	13	29	31

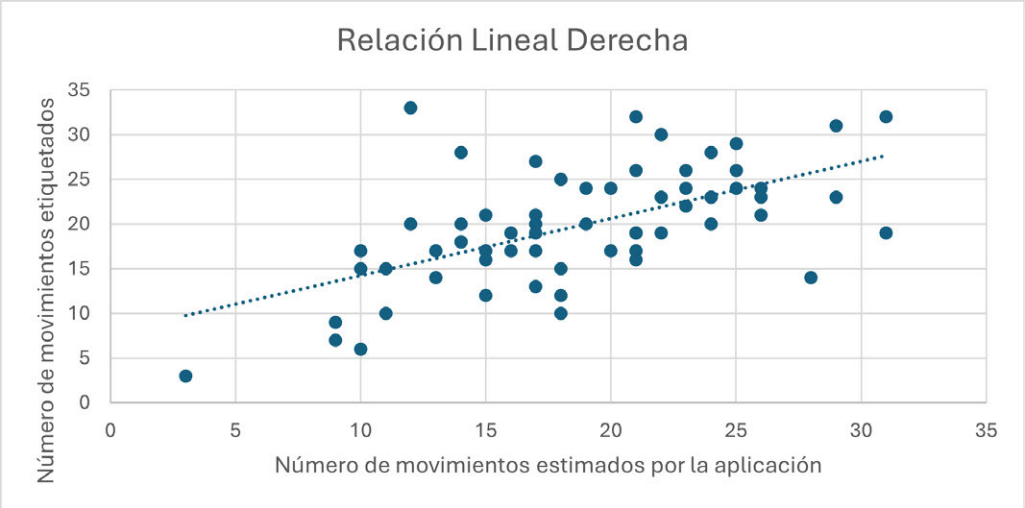
Figura 50: Tabla comparativa entre los movimientos estimados y los movimientos etiquetados

Analizando los anteriores resultados, se obtienen las gráficas de la Figura 51, donde se puede observar que la correlación de la aplicación y el algoritmo desarrollado frente a las etiquetas disponibles demuestra buenos resultados.

La gran aproximación lineal que se observa demuestra la correcta automatización para videos disponibles.



(a) Relación lineal de los movimientos estimados y etiquetados en la izquierda(Coeficiente R = 0,539)

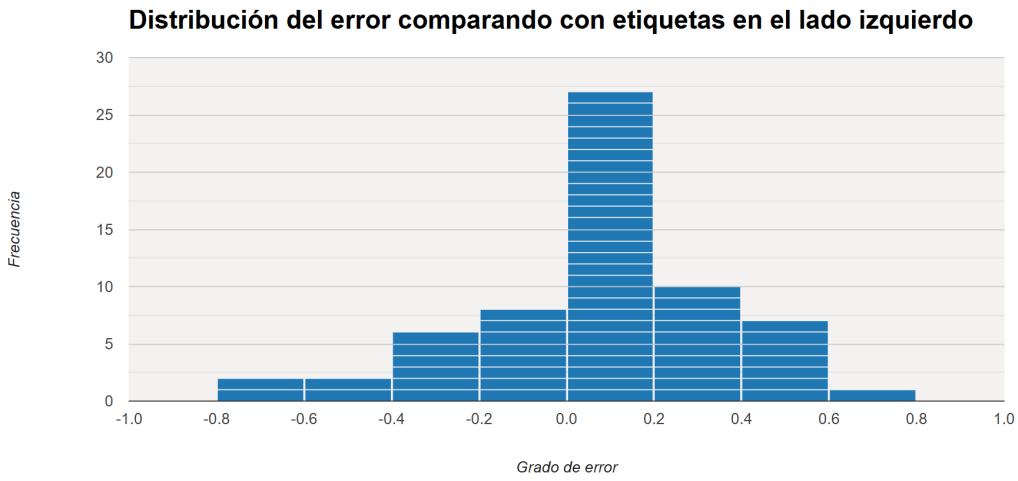


(b) Relación lineal de los movimientos estimados y etiquetados en la derecha(Coeficiente R = 0,584)

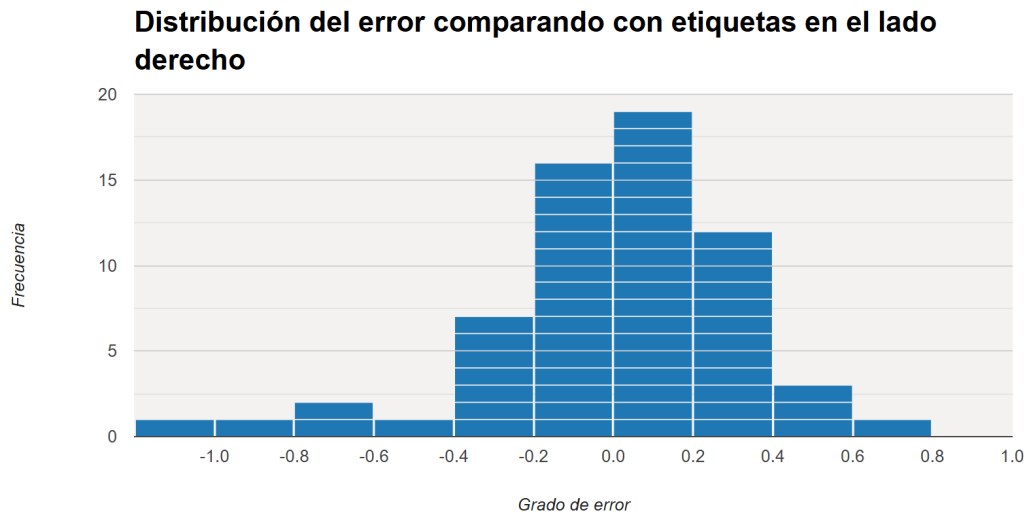
Figura 51: Relación de los datos estimados con los etiquetados

Finalmente, se ha calculado los respectivos histogramas de la Figura 52, en donde se puede observar la distribución del error en la validación. Se puede observar que la mayor parte del error está entre -20 % y el 30 %:

- Para la comparación de las etiquetas del lado izquierdo se ha obtenido una media del error de `0.0709` con una desviación estándar con valor `0.273`.
- Para la comparación de las etiquetas del lado derecho se ha obtenido una media del error de `0.0048` con una desviación estándar con valor `0.302`.



(a) Distribución del error en las truchas de la izquierda en validación de resultados



(b) Distribución del error en las truchas de la derecha en validación de resultados

Figura 52: Distribución del error respecto a las etiquetas de los investigadores

Como resumen, la detección y el conteo de movimiento se ha hecho correctamente, si bien hay errores que pueden provenir de diversos factores como la subjetividad del etiquetado manual y falta de ajuste del algoritmo de conteo de movimientos de la aplicación. Sin embargo, como primera aproximación al problema permite ver fácilmente el comportamiento de las truchas de manera comparativa en el *NetTest*.

La última validación de resultados que falta es la relacionada con el rendimiento de la aplicación y el uso del `HardWare`. En este sentido:

- Se ha conseguido una aplicación que consigue seleccionar el mejor formato de red neuronal según el ordenador en el que se ejecute, permitiendo minimizar el tiempo necesario para procesar un video. Esto se ve reflejado en los anteriores resultados, en los que se han procesado 64 videos de 15 segundos cada uno en solo 1 hora en total.
- A través de `DearPyGUI`, se consigue una aplicación fluida que mantiene una tasa de fotogramas sin caídas, lo que provee una experiencia muy buena al usuario.
- A través del uso de procesos, se ha conseguido descargar el hilo principal, minimizar tiempos de proceso e implementar funciones multimedia sin que afecte al rendimiento. Esto se ve reflejado en la carga que tiene la aplicación sobre la memoria `RAM`, que en las pruebas no ha superado el umbral de `1.5GB` en ningún momento para videos de resolución `1080P`.

8. PRESUPUESTO

En este apartado se describen los costes de desarrollo que conlleva este trabajo, detallándose el coste de los recursos de forma individual. Estos pueden ser materiales, mano de obra u otros.

En el proyecto ha habido 4 fases:

1. Estudio y análisis del estado del arte (45h)
2. Pruebas de diseño y concepto (85h)
3. Implementación de la solución (160h)
4. Documentación del trabajo (60h)

El punto 1 y 2 se han considerado investigación, siendo el punto 3 y 4 partes de la fase de desarrollo.

En la Tabla 6 se presenta el presupuesto general:

Nombre	***Precio unitario (€)***	***Unidades***	***Precio total(€)***
Personal de investigación	20	130	2400
Personal de desarrollo	20	220	4400
Ordenador portátil Asus F1704VA	150	1	150
Torre MSI	30	1	30
Equipo NAS para los videos	50	1	50
		Total=	7030

Tabla 6: Presupuesto general del proyecto

En este caso, algunos de los equipos ya habían sido proporcionados previamente, por lo tanto se aplicó una amortización respecto al valor del producto:

- Ordenador portátil Asus F1704VA (Valor original: 600€) amortización de 150€.
- Torre MSI (Valor original: 1000€) amortización de 30€.
- Equipo NAS (Valor original: 1200€) amortización de 50€.

Por como se ha planteado el proyecto, los programas usados durante el desarrollo son de libre uso o gratis, por lo tanto no ha sido necesario incluirlos en el presupuesto.

9. MANUAL DE USUARIO

En esta sección se va a describir como utilizar la aplicación generada en este trabajo.

Primero, al haber sido exportada a un ejecutable solo es necesario abrir el ejecutable, pero para que todo funcione es necesario que en la misma carpeta en la que se encuentra el ejecutable haya una carpeta con los recursos de la aplicación, que son:

- Icono de la aplicación.
- Modelos de la red neuronal.
- Imágenes extra.

Estos elementos deberían venir en el `.zip` que contenga el ejecutable, si no están la aplicación no se abrirá.

Una vez abierta la aplicación, el usuario se encontrará con la ventana inicial de la Figura 53.

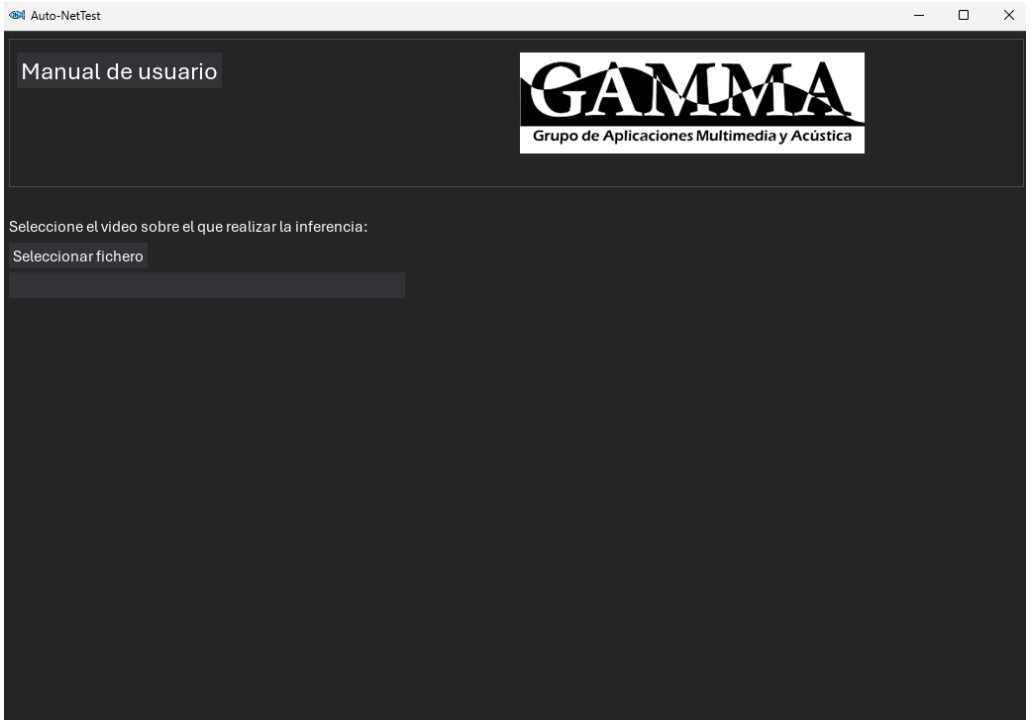


Figura 53: Ventana inicial de la aplicación

En esta pantalla el usuario puede:

- Abrir un manual propio en la aplicación que explica de forma resumida el funcionamiento de la misma.
- Seleccionar un video a través de un selector como se ve en la Figura 54. Este selector está diseñado para que filtre los videos que tiene formato compatible con la aplicación, eso quiere decir que sí el video no aparece, no es compatible y debe exportarse a otro formato.

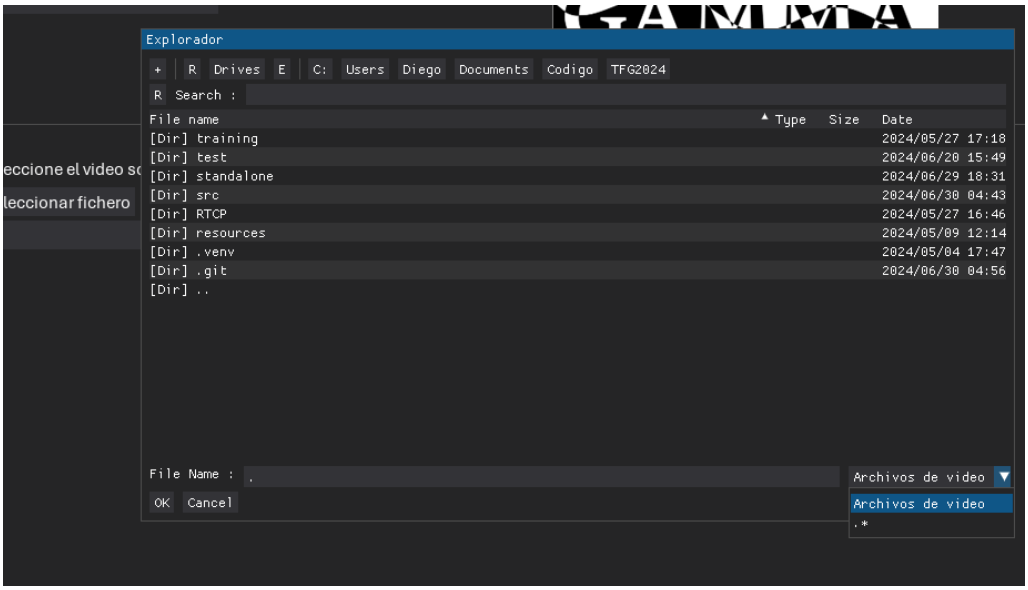


Figura 54: Vista del explorador para seleccionar video

- Inferir: una vez seleccionado un video aparece un botón como el de la Figura 55. Una vez pulsado se hacen una serie de comprobaciones para que no haya ningún error y se pasa a la pestaña de carga.

Figura 55: Botón de inferir de la ventana inicial de la aplicación

Una vez en la pantalla de carga, la cual se puede ver en la Figura 56, el usuario puede observar el progreso en tiempo real del procesado del video. Además de esto, el usuario puede cancelar la inferencia con un botón que le devuelve a la ventana principal.

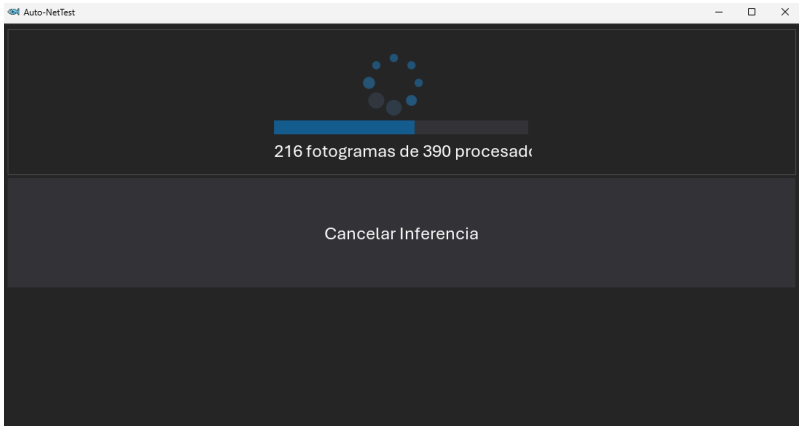


Figura 56: Pantalla de carga en medio de un video con la barra de progreso

Finalizado el procesado del video, se pasa a la pantalla de guardado y análisis de resultados, cuyo aspecto y secciones se indican en la Figura 57.

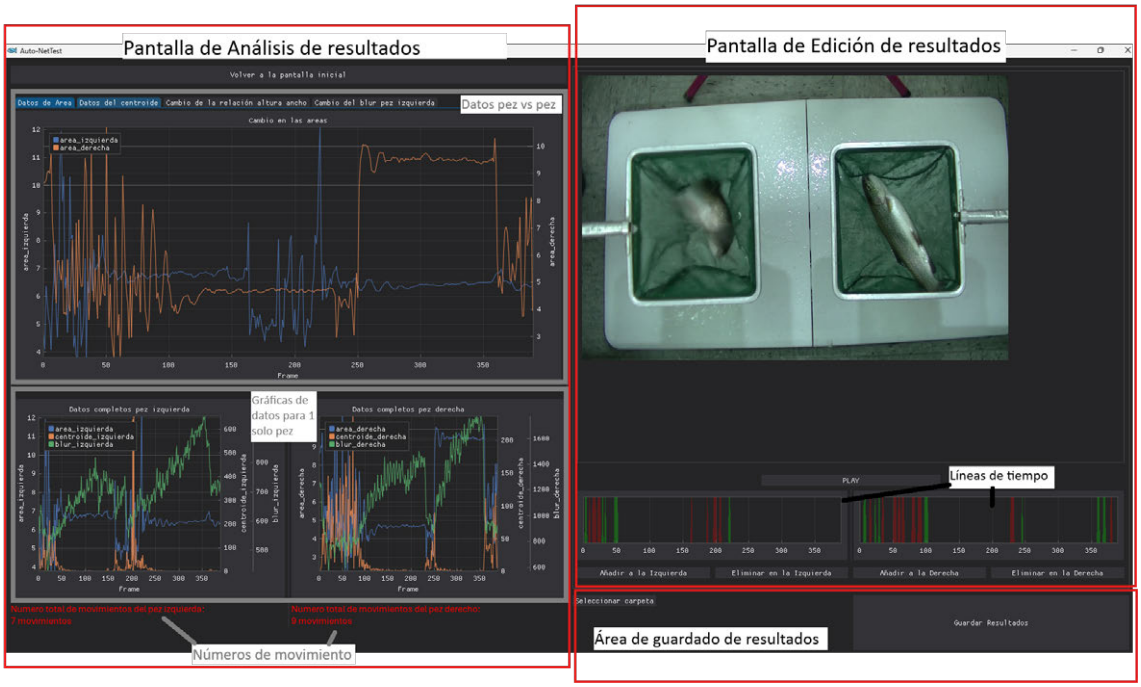


Figura 57: Pantalla de datos y sus zonas

Esta pantalla se divide en secciones, ya que cumple las funciones de muestra de datos obtenidos y de revisión/edición de los movimientos resultantes:

- Sección de análisis de resultados: muestra los datos del área, centroide, relación de aspecto de la trucha y el `blur`, además del número de movimientos. En la parte vertical muestra para cada dato las dos gráficas de las truchas para analizar comparativamente. En la parte inferior se muestran todos los datos para cada pez, de manera que se pueda hacer una correlación visual.
- Sección de edición y revisión de resultados: contiene un reproductor de video con una serie de controles y una pareja de líneas de tiempo. El reproductor funciona como se espera y la

utilidad de las líneas de tiempo se basa en la revisión y edición del número de movimientos que han ocurrido.

En las líneas de tiempo, se marcan con rojo las situaciones en las que han ocurrido movimientos y en verde las situaciones en las que posiblemente hayan ocurrido movimientos, pero por ciertas reglas, no se han tenido en cuenta y merecen ser revisadas manualmente (opcional).

Las líneas de tiempo son seleccionables y, cuando se realiza un clic dentro de alguna de ellas el video se para y se deja una marca de la última posición en la que se ha clicado. Este comportamiento se puede observar en la Figura 58, donde aparece un marcador azul.

Figura 58: Selección sobre una línea de tiempo y marcado

Cuando se realiza este clic, el video se pausa y, la siguiente vez que se reproduzca, cambiará a esa posición. Además de esto, en esta posición podemos añadir o eliminar movimientos que hayan ocurrido como se ve en la Figura 59, donde se añade un movimiento manualmente. Esto se verá reflejado en el número de movimientos que se marcan en la sección de análisis de resultados.

Figura 59: Movimiento añadido manualmente a través de la línea de tiempo

- Sección de guardado: contiene un simple explorador de carpetas para seleccionar un lugar donde guardar los archivos. Si no se selecciona ninguna carpeta, se guardan en el directorio donde se encuentre el ejecutable.
  Se guardan los resultados en formato `CSV` y `JSON` para permitir flexibilidad e integrarlo en un procesado automático.

Finalmente, desde la pantalla de análisis de datos se puede volver a la pantalla inicial usando el botón correspondiente, que está situado en la parte superior izquierda.

10. ANÁLISIS DE IMPACTO DEL PROYECTO

Como se discutió en la introducción de este trabajo, el cambio climático cada vez está tomando más importancia en la sociedad. Dentro de todas las herramientas que existen para reducir su efecto, está la investigación para la reducción de huella de carbono en la generación de alimento. Especialmente, se están realizando muchas investigaciones con la acuicultura, ya que es un método muy seguro y controlado para cubrir las necesidades de proteína de origen marítimo y evitar la pérdida de los ecosistemas marítimos.

Este trabajo tiene un impacto social, siendo este no sobre el ser humano, sino sobre los animales. La herramienta que se ha generado permite a investigadores parametrizar de forma experimental las consecuencias que tiene un tratamiento u otro. Esto tiene como objetivo determinar técnicas que aumenten el bienestar animal en las piscifactorías.

En el análisis de impacto económico, este trabajo mejora sistemas que antes tomarían demasiado tiempo y serían propensos a errores humanos, lo cual se transforma en una reducción del dinero perdido en analizar recursos como videos.

Dentro del análisis de impacto también aparece la eliminación de la brecha digital a la hora de gestionar resultados de experimentos con áreas que no son la ingeniería.

Objetivos de Desarrollo Sostenible

Este trabajo busca alinearse con los ODS definidos por la Unión Europea. Dentro de los objetivos individuales que se ven afectados por este trabajo tenemos:

- **Trabajo decente y crecimiento económico**: los investigadores que pierden más tiempo analizando videos que realizando experimentos y obteniendo conclusiones van a ver esta herramienta para automatizar procesos llenos de errores humanos.
- **Industria, innovación e infraestructura**: este trabajo busca dar herramientas para mejorar las condiciones animales en la industria de la acuicultura.
- **Producción y consumo responsables**: si se mejora la calidad animal en la acuicultura, la producción aumentará y será menos necesario utilizar los entornos marítimos para realizar pesca masiva.
- **Acción por el clima**: mejorar las condiciones de producción en la acuicultura reduce los beneficios que produce la pesca de alta mar.
- **Vida submarina**: la obtención de proteína a través de la acuicultura consigue aliviar los entornos marinos.

11. CONCLUSIONES Y TRABAJOS FUTUROS

Los experimentos actuales en la ciencia cada vez manejan mayores cantidades de datos. La capacidad de automatizar la obtención de los datos, su procesado y su análisis es uno de los campos más importantes de la tecnología moderna. Dentro de esto encontramos la visión por ordenador como campo que engloba diversas tecnologías para procesar imagen.

Dentro la visión por ordenador, uno de los paradigmas más eficientes en la actualidad para conseguir analizar videos son las arquitecturas de redes neuronales `CNN` que aprovechan un planteamiento de operaciones por matrices para poder manejar grandes cantidades de datos a la vez. Una de las `CNN` más populares actualmente es `YOLOv8` por su rendimiento y eficiencia.

A través de pruebas comparativas, se ha observado que respecto a métodos tradicionales como el análisis de flujo óptico, las arquitecturas `CNN`; más específicamente `YOLO`, se pueden usar para caracterizar el movimiento de truchas de forma extremadamente rápida. Estos sistemas son capaces de reconocer correctamente truchas en movimiento y con una tasa de error muy baja con poco tiempo de entrenamiento.

Esto ha dado lugar a la participación en un artículo de congreso sobre la viabilidad de automatizar procesos experimentales en acuicultura a través de la obtención de datos por aprendizaje profundo.

En este trabajo se ha usado el modelo `YOLO` entrenado para la detección de truchas para construir una aplicación que cuente el número de movimientos. Para conseguir esto se han aprovechado técnicas como los sistemas basados en ejecución paralela y la correcta elección del modelo dependiendo del `HardWWare` disponible del ordenador.

Con esto se ha conseguido obtener resultados como el procesamiento de 64 videos en 1 hora, obteniendo una tasa de error respecto a los movimientos detectados sobre movimientos previamente etiquetados en su mayoría de entre -20 % y 20 %.
Estos resultados son muy buenos y cumplen con los objetivos marcados en el proyecto.

Con todo esto se ha conseguido construir una aplicación capaz de funcionar a tiempo real, que no necesita conocimientos de tecnologías sobre las redes neuronales, lenguajes de programación o uso de una consola de comandos para ser usada. Esto permite potenciar la automatización de procesos experimentales en campos donde el uso de este tipo de herramientas va retrasado.

A través de este trabajo, se ha podido estudiar el desarrollo de soluciones aplicando técnicas de ingeniería (Planteamiento → Prueba de concepto → Desarrollo → Validación de resultados → Entrega final), entendiendo los requisitos que puede tener una persona ajena al mundo de la ingeniería y sabiendo definir las limitaciones que puede tener la puesta a prueba de un concepto en un entorno real donde hay variables que no se pueden controlar.

Las futuras líneas que se plantean en este trabajo son las siguientes:

- Mejora de la interfaz de usuario: principalmente añadir funcionalidades centradas en la accesibilidad.
- Mejora de la red neuronal: hacer un `fine-tunning` de hiperparámetros y realizar un entrenamiento lo mejor posible teniendo un conjunto de datos mucho más amplio. Esto permitiría ajustar completamente la `Bounding Box` de la trucha. También sería interesante entrenar la red neuronal para trabajar con peces similares.
- Despliegue sobre dispositivos IoT y creación de un cliente ligero: esto permitiría distribuir el sistema y que un usuario pudiese usarlo sin tener un ordenador de altas prestaciones.
- Mejorar el algoritmo de control de movimiento: plantear una alternativa mucho más flexible e inteligente, como un árbol de decisión.

12. BIBLIOGRAFÍA

[1] *El estado mundial de la pesca y la acuicultura 2022*. DOI: 10.4060/cc0461es. dirección: https://www.fao.org/3/cc0461es/online/cc0461es.html (visitado 27-02-2024).

[2] J. E. Barrios Sánchez, *La prueba de la red: evaluando su repetibilidad y el efecto del ayuno en la trucha arcoiris (Oncorhynchus mykiss)*, info:eu-repo/semantics/bachelorThesis, Madrid, nov. de 2023. dirección: https://oa.upm.es/77142/ (visitado 04-04-2024).

[3] 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, n.º 11, págs. 1325-1330, 2016, ISSN: 2041-210X. DOI: 10.1111/2041-210X.12584. dirección: https://onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.12584 (visitado 04-04-2024).

[4] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer Nature, 2022. dirección: https://books.google.es/books?hl=es&lr=&id=QptXEAAAQBAJ&oi=fnd&pg=PR9&dq=computer+vision+algorithms+and+applications+2nd+ed&ots=BNvitZVCsk&sig=ecxTFEpxbEYtL2yzTq0MHzbPPpI (visitado 09-06-2024).

[5] *NEMESIS-3D-CM*. dirección: http://nemesis3d.citsem.upm.es/inicio/ (visitado 09-06-2024).

[6] W. S. Mcculloch y W. Pitts, «A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY»,

[7] *Jg-Fisher/Perceptron: Perceptron Model Built from Scratch*. dirección: https://github.com/jg-fisher/perceptron?tab=readme-ov-file (visitado 12-06-2024).

[8] K.-L. Du y M. Swamy, *Neural Networks and Statistical Learning*. oct. de 2013, ISBN: 978-1-4471-5570-6. DOI: 10.1007/978-1-4471-5571-3.

[9] Z. Zou, K. Chen, Z. Shi, Y. Guo y J. Ye, «Object Detection in 20 Years: A Survey», *Proceedings of the IEEE*, vol. 111, n.º 3, págs. 257-276, 2023. dirección: https://ieeexplore.ieee.org/abstract/document/10028728/ (visitado 09-06-2024).

[10] S. Saha, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*, nov. de 2022. dirección: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visitado 12-06-2024).

[11] *Laplacian Filter - an Overview — ScienceDirect Topics*. dirección: https://www.sciencedirect.com/topics/engineering/laplacian-filter (visitado 12-06-2024).

[12] *Convolutional Neural Network (CNN)*. dirección: https://developer.nvidia.com/discover/convolutional-neural-network (visitado 12-06-2024).

[13] B. Kallfelz Sirmacek, N. Botteghi y S. Sanchez, «SEQUENTIAL IMAGE PROCESSING METHODS FOR IMPROVING SEMANTIC VIDEO SEGMENTATION ALGORITHMS A PREPRINT», oct. de 2019.

[14] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection», en *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2016, págs. 779-788. DOI: 10.1109/CVPR.2016.91. dirección: https://ieeexplore.ieee.org/document/7780460 (visitado 06-03-2024).

[15] J. Han, D. Zhang, G. Cheng, N. Liu y D. Xu, «Advanced Deep-Learning Techniques for Salient and Category-Specific Object Detection: A Survey», *IEEE Signal Processing Magazine*, vol. 35, n.º 1, págs. 84-100, ene. de 2018, ISSN: 1558-0792. DOI: 10.1109/MSP.2017.2749125. dirección: https://ieeexplore.ieee.org/document/8253582 (visitado 14-02-2024).

[16] *The PASCAL Visual Object Classes Challenge 2012 (VOC2012)*. dirección: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/ (visitado 13-06-2024).

[17] P. C. Manojkumar, L. S. Kumar y B. Jayanthi, «Performance Comparison of Real Time Object Detection Techniques with YOLOv4», en *2023 International Conference on Signal Processing, Computation, Electronics, Power and Telecommunication (IConSCEPT)*, mayo de 2023, págs. 1-6. DOI: 10.1109/IConSCEPT57958.2023.10169970. dirección: https://ieeexplore.ieee.org/document/10169970 (visitado 14-02-2024).

[18] C. Bhagya y A. Shyna, «An Overview of Deep Learning Based Object Detection Techniques», en *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, abr. de 2019, págs. 1-6. DOI: 10.1109/ICIICT1.2019.8741359. dirección: https://ieeexplore.ieee.org/document/8741359 (visitado 14-02-2024).

[19] Ultralytics, *YOLOv8*. dirección: https://docs.ultralytics.com/es/models/yolov8 (visitado 13-06-2024).

[20] M. Castillo Moral, *Sistema de conteo de peces en piscifactorías mediante redes neuronales*, Madrid], 2022.

[21] S.-H. Lee, H.-C. Ku y R.-W. Huang, «Detection of Catfish Activity in Smart Fish Farming System», en *2024 IEEE International Conference on Consumer Electronics (ICCE)*, ene. de 2024, págs. 1-3. DOI: 10.1109/ICCE59016.2024.10444155. dirección: https://ieeexplore.ieee.org/document/10444155 (visitado 13-06-2024).

[22] C. Xia, L. Fu, H. Liu y L. Chen, «In Situ Sea Cucumber Detection Based on Deep Learning Approach», en *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*, mayo de 2018, págs. 1-4. DOI: 10.1109/OCEANSKOBE.2018.8559317. dirección: https://ieeexplore.ieee.org/document/8559317 (visitado 13-06-2024).

[23] R. Lade, A. Patil, K. Malu, S. Pal, N. Sable y P. Shelke, «Automated Fish Species Identification Using Computer Vision», en *2023 7th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*, ago. de 2023, págs. 1-8. DOI: 10.1109/ICCUBEA58933.2023.10392234. dirección: https://ieeexplore.ieee.org/document/10392234 (visitado 13-06-2024).

[24] O. Ulucan, D. Karakaya y M. Turkan, «A Large-Scale Dataset for Fish Segmentation and Classification», en *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, oct. de 2020, págs. 1-5. DOI: 10.1109/ASYU50717.2020.9259867. dirección: https://ieeexplore.ieee.org/document/9259867 (visitado 06-03-2024).

[25] *Acuicultura 4.0*. dirección: https://cetga.org/cuatropuntocero/ (visitado 13-06-2024).

[26] M. Sonka, V. Hlavac y R. Boyle, *Image Processing, Analysis and Machine Vision*. Springer, nov. de 2013, ISBN: 978-1-4899-3216-7.

[27] B. Horn y B. Schunck, «Determining Optical Flow», *Artificial Intelligence*, vol. 17, págs. 185-203, ago. de 1981. DOI: 10.1016/0004-3702(81)90024-2.

[28] R. Schuster, C. Bailer, O. Wasenmüller y D. Stricker, «Combining Stereo Disparity and Optical Flow for Basic Scene Flow», mar. de 2018.

[29] T. Kobayashi, Y. Tanaka, K. Fukae, T. Imai y K. Arai, «Aqua Colony for Fully Automated Aquaculture», en *2023 11th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, jul. de 2023, págs. 11-16. DOI: 10.1109/MobileCloud58788.2023.00008. dirección: https://ieeexplore.ieee.org/document/10229441 (visitado 13-06-2024).

[30] K. M. Pai, K. B. A. Shenoy y M. M. M. Pai, «A Computer Vision Based Behavioral Study and Fish Counting in a Controlled Environment», *IEEE Access*, vol. 10, págs. 87 778-87 786, 2022, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3197887. dirección: https://ieeexplore.ieee.org/document/9853540 (visitado 13-06-2024).

[31] S. Do, K. Song y J. Chung, «Basics of Deep Learning: A Radiologist's Guide to Understanding Published Radiology Articles on Deep Learning», *Korean journal of radiology*, vol. 21, págs. 33-41, ene. de 2020. DOI: 10.3348/kjr.2019.0312.

[32] *¿GPU vs. CPU? ¿Qué Es La Computación Por GPU? — NVIDIA*. dirección: https://www.nvidia.com/es-la/drivers/what-is-gpu-computing/ (visitado 14-06-2024).

[33] *Observatorio de I+D+i UPM*. dirección: https://www.upm.es/observatorio/vi/index.jsp?pageac=estructuras/grupo.jsp&idGrupo=177&h=1 (visitado 01-06-2024).

[34] CVAT.ai Corporation, *Computer Vision Annotation Tool (CVAT)*, nov. de 2023. dirección: https://github.com/cvat-ai/cvat (visitado 25-06-2024).

[35] Á. De la Llave-Propín, D. Aceituno Seoane, N. Sáenz Lechón et al., «Estilos de Afrontamiento: Hacia La Automatización de La Prueba de La Red o Net Test Con Truchas Mediante YOLOv8», en *XIX Congreso Nacional de Acuicultura 2024*, Las Palmas de Gran Canaria,Spain, jun. de 2024.

[36] D. Kolesnikov, *Koldim2001/COCO_to_YOLOv8*, jun. de 2024. dirección: https://github.com/Koldim2001/COCO_to_YOLOv8 (visitado 27-06-2024).

[37] *Hoffstadt/DearPyGui: Dear PyGui: A Fast and Powerful Graphical User Interface Toolkit for Python with Minimal Dependencies*. dirección: https://github.com/hoffstadt/DearPyGui (visitado 29-06-2024).

[38] L. doleron, *Deep Learning from Scratch in Modern C++: Gradient Descent*, ago. de 2023. dirección: https://pub.towardsai.net/deep-learning-from-scratch-in-modern-c-gradient-descent-670bc5889112 (visitado 26-06-2024).

[39] Ultralytics, *Configuration*. dirección: https://docs.ultralytics.com/usage/cfg (visitado 26-06-2024).

13. ANEXOS

ANEXO A: Explicación de los hiperparámetros de YOLO

Las redes neuronales como `YOLO` y en general todas las que utilizan entrenamiento supervisado, tienen como objetivo acercarse lo máximo posible al resultado esperado en las etiquetas de entrenamiento. A lo largo de este entrenamiento se van actualizando parámetros (valores que sí cambian en entrenamiento) como son los pesos, los `bias` y otros elementos. Esta búsqueda de aproximación se puede ver reflejada como la búsqueda de mínimos globales en una función, en la mayoría de casos es buscar los mínimos globales de la función de perdidas. Esto se puede ver en la Figura 60.

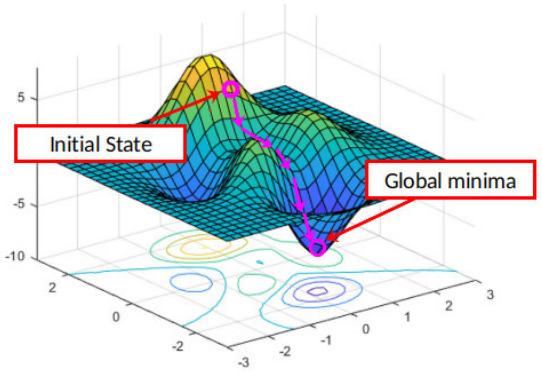


Figura 60: Búsqueda de mínimos globales por un algoritmo de aprendizaje automático[38]

Los modelos de `YOLO` tienen definidos una serie de hiperparámetros (valores que no cambian en entrenamiento) que sirven como ajustes fijos durante en el entrenamiento. Estos hiperparámetros ajustan la manera en la que el algoritmo decide el siguiente paso a tomar en su búsqueda de un mínimo global y pueden acelerar o ralentizar el tiempo necesario de entrenamiento para alcanzar unos valores de perdida adecuados. Algunos de los hiperparámetros más típicos[39] son:

- **Learning Rate inicial**: es el valor que afecta cuanto se cambian los pesos entre época. Por defecto es `0.01`, que puede parecer poco, pero es mejor dar pasos pequeños para llegar al mínimo global.
- **Learning Rate final**: es una fracción del valor inicial. Según se realiza el entrenamiento de la red es normal reducir el `Learning Rate` para ajustar de forma más fina. Por defecto es `0.01`.
- **Momentum**: es una variable que permite tener en cuenta que camino se ha recorrido en la anterior época, para no volver a situaciones anteriores y seguir progresando. Por defecto es `0.937`.
- **Weight Decay**: hiperparámetro para prevenir sobreajuste y penalizar los pesos que son extremadamente grandes en la red.
- **Épocas de calentamiento**: número de épocas que se utilizaran para ir poco a poco acercándose a los valores iniciales de `Learning Rate`. Por defecto su valor es 3.
- **Tamaño de lote**: número de imágenes que se utilizaran en una pasada. Si tenemos 50 imágenes y el tamaño del lote es 10, se pasaran 10 y se ajustaran los pesos, y en este caso una época se compondra de 5 pasadas de lote.
- **Número de épocas**
- **Número de capas ocultas**

Además, YOLO utiliza de base una herramienta de aumento de datos simple, por lo tanto tenemos muchos hiperparámetros para ajustar su funcionamiento:

- **Hsv componente h**: ajusta el Hue de la imagen por una fracción que por defecto es 0.015.
- **Hsv componente s**: ajusta la saturación de la imagen por una fracción que por defecto es 0.7, permite simular diferentes condiciones de iluminación.
- **Hsv componente v**: modifica el brilo de la imagen, por defecto a una fracción de 0.4.
- **Translate**: mueve la imagen horizontal y verticalmente por defecto un 10%.
- **Scale**: ajusta el tamaño de la imagen para simular diferentes distancias a la cámara. Por defecto 0.5.
- **Fliplr**: espeja la imagen horizontalmente con una probabilidad por defecto de 0.5.
- **Mosaic**: permite combinar 4 imágenes en una para crear imágenes compuestas que mejoran mucho la calidad del entrenamiento. Por defecto siempre está activo.
- **Erasing**: borra partes de la imagen de forma aleatoria para que el modelo sepa reconocer segmentos del elemento y que sea capaz de detectar características más específicas.
- **Crop fraction**: de forma aleatoria corta la imagen a fracciones de su tamaño original.

El resultado de este aumento de datos se puede ver en los lotes con los que se entrena la red como el de la Figura 61, donde vemos cambios de brillo, saturación, recortes de imagen y combinación de imágenes en diferentes tamaños.

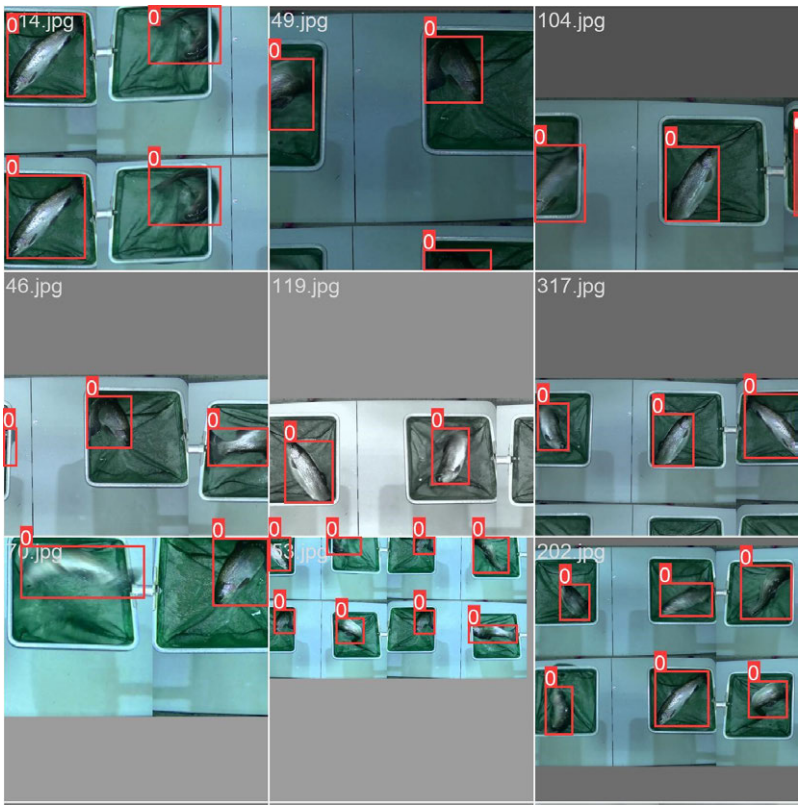


Figura 61: Ejemplo de aumento de datos automático de YOLO

ANEXO B: Datos de entrenamiento

Entrenamiento 1

- **Fecha**: 9 de abril de 2024 (Prueba de concepto).
- **Descripción del conjunto de datos**: 24 imágenes seleccionadas manualmente del video `23_NT_R1_J1_P1_2.mp4`. Conjunto dividido en 70-15-15 para los respectivos subconjuntos.
- **Resultados de entrenamiento**: En las siguientes figuras se observan unos resultados que aparentan buenos para ser un primer entrenamiento, pero debido al conjunto de datos con datos similares entre sí, en situaciones de posiciones de trucha nuevas, se observó problemas en la precisión de la red. Aparte de esto se puede ajustar más, ya que con 100 épocas vemos que se para cuando todavía hay una tendencia descendente en el error en el conjunto de validación.

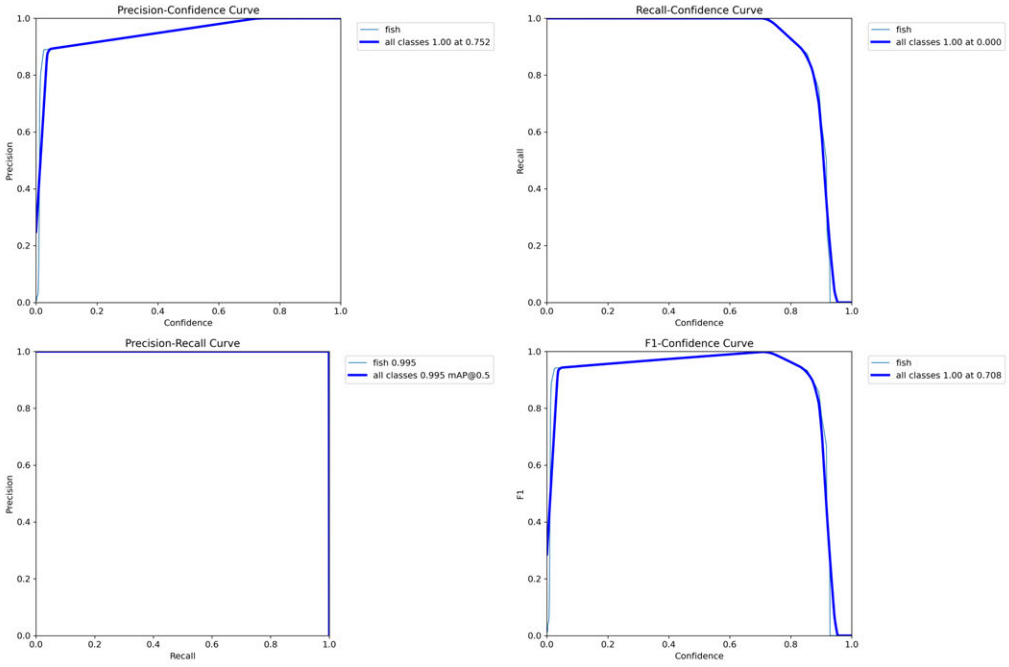


Figura 62: Gráficas de estadísticos finales del entrenamiento 1

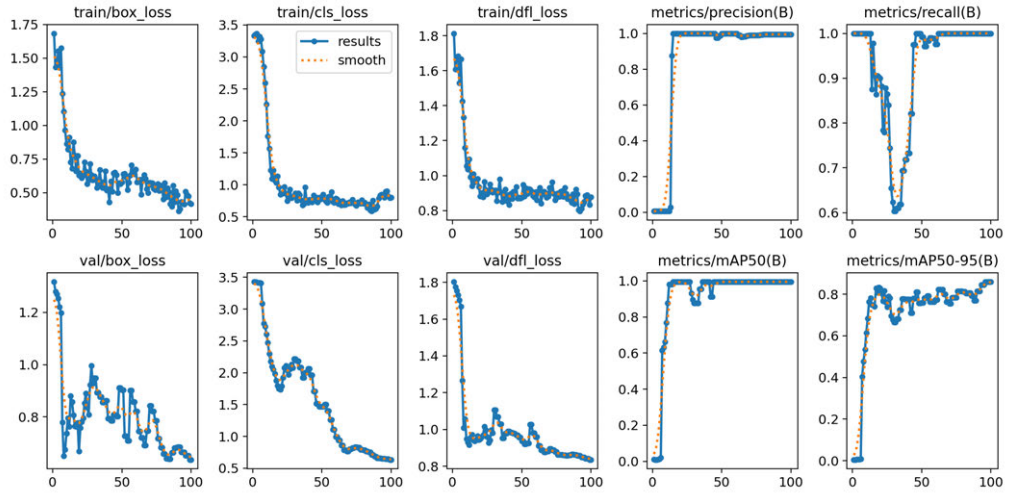


Figura 63: Gráficas de evolución en épocas del entrenamiento 1

Entrenamiento 2

- **Fecha**: 25 de abril de 2024.
- **Descripción del conjunto de datos**: al conjunto del entrenamiento de prueba se le añadieron imágenes extra y situaciones con redes vacías. En total 50 imágenes en división 70-15-15 para los subconjuntos.
- **Resultados del entrenamiento**: A través del aumento de datos del conjunto y con el mismo número de épocas, se puede observar en la Figura 64 que hemos mejorado mucho el `recall` teniendo menos falsos positivos, pero hemos perdido precisión en situaciones de poca confianza, esto puede ser debido al añadir imágenes de redes sin peces, pero no es nada problemático, ya que para `Bounding Boxes` con tan poca confianza, es mejor no procesarlas. Aparte se ve en la Figura 65 que hemos mejorado mucho la precisión en la perdida de las `Bounding Boxes`, esto era el objetivo. Vemos que el resto de parametros no han cambiado mucho, buena señal.

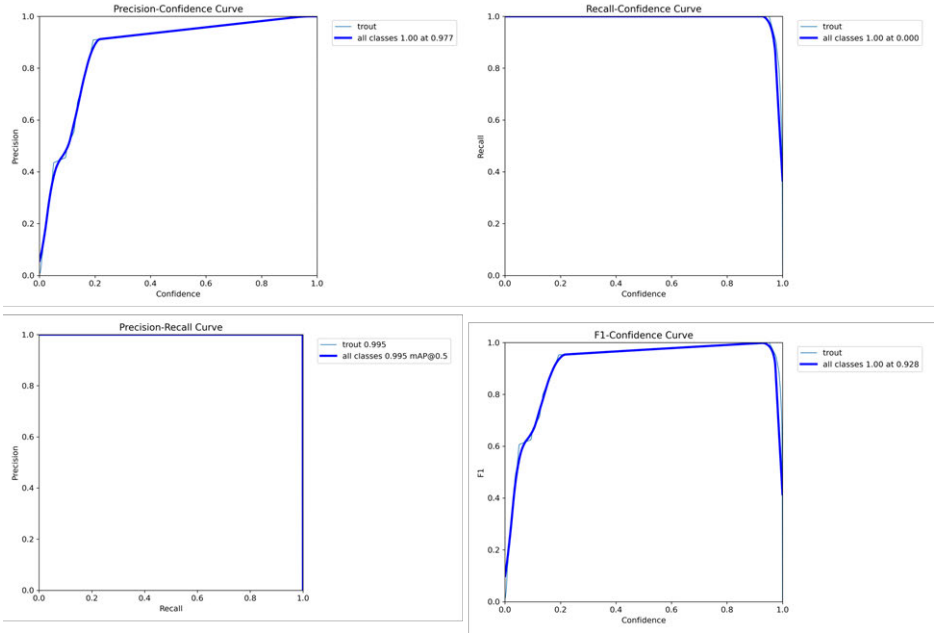


Figura 64: Gráficas de estadísticos finales del entrenamiento 2

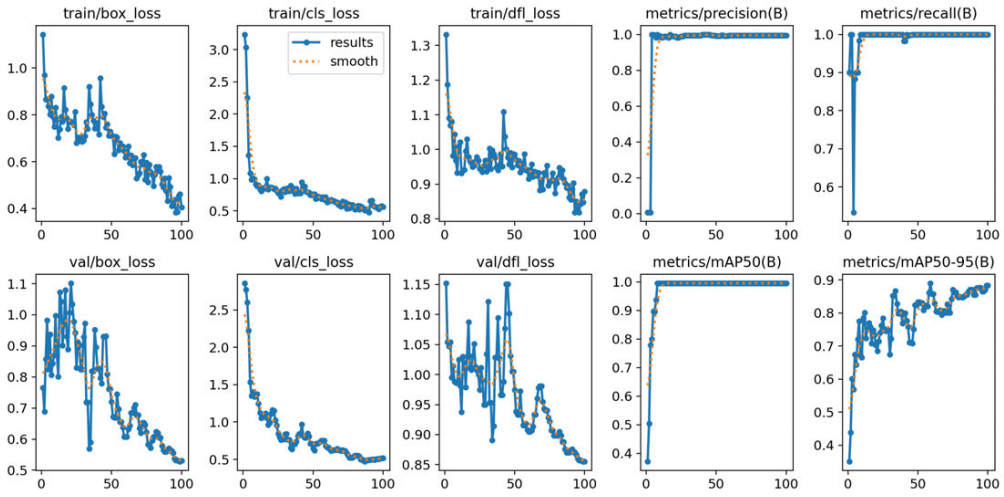


Figura 65: Gráficas de evolución en épocas del entrenamiento 2

Entrenamiento 3

- **Fecha**: 4 de mayo de 2024.
- **Descripción del conjunto de datos**: Mismo conjunto que el entrenamiento anterior.
- **Resultados del entrenamiento**: a través del aumento de las épocas hasta 300, se pudo observar como se podían alcanzar valores de error en las `Bounding Boxes` bastante más bajas. Este entrenamiento también dejo claro que el conjunto de datos de validación era demasiado reducido y no era suficientemente representativo, las imágenes eran muy similares entre sí.

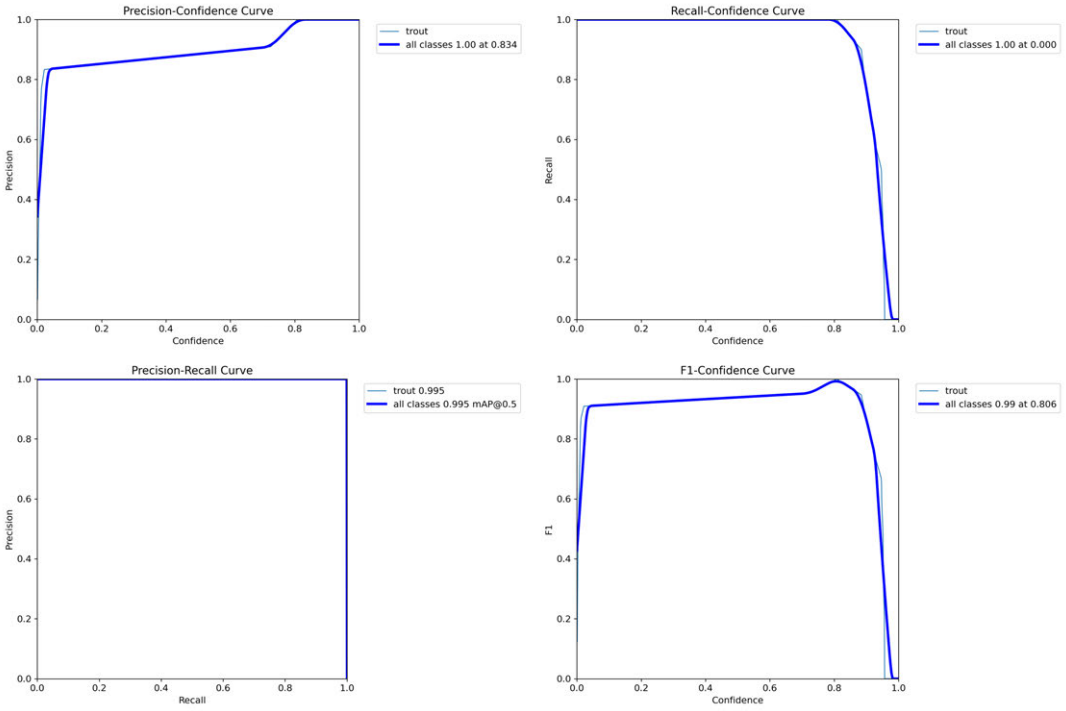


Figura 66: Gráficas de estadísticos finales del entrenamiento 3

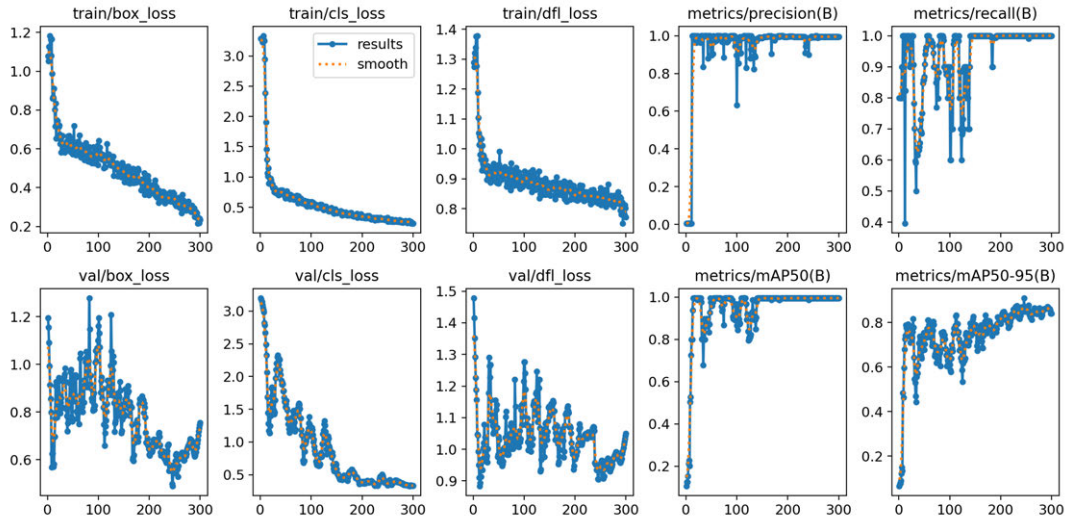


Figura 67: Gráficas de evolución en épocas del entrenamiento 3

Entrenamiento 4

- **Fecha**: 4 de mayo de 2024.
- **Descripción del conjunto de datos**: el conjunto de datos fue expandido con imágenes de los videos antiguos, obteniendo en total 84 imágenes representativas divididas en 70-15-15.
- **Resultados del entrenamiento**: por los nuevos tipos de imágenes añadidas, hemos perdido en la mayoría de métricas, pero esta red es mucho más representativa para los datos reales. Además de esto, el conjunto de validación que se usa de referencia ahora es mucho más realista y tiene imágenes difíciles que implican movimientos, ya que para imágenes de truchas estáticas ha estado funcionando correctamente durante todos los entrenamientos.

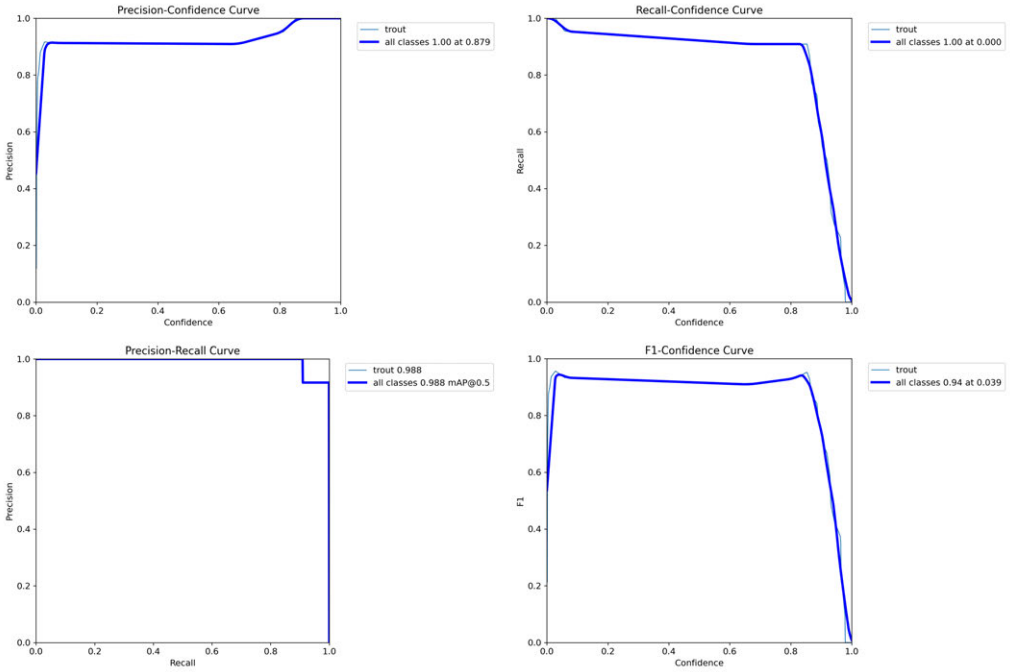


Figura 68: Gráficas de estadísticos finales del entrenamiento 4

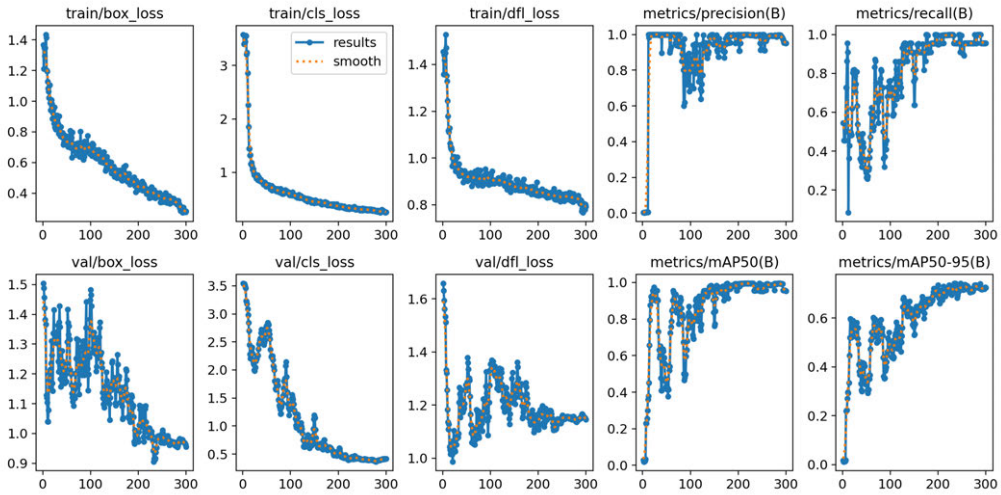


Figura 69: Gráficas de evolución en épocas del entrenamiento 4

Entrenamiento Final

- **Fecha**: 9 de junio de 2024.
- **Descripción del conjunto de datos**: 203 imágenes en total. Se añadieron fotogramas aleatorios de varios videos del experimento *NetTest* para poder manejar un conjunto más grande.
- **Resultados del entrenamiento**: resultados muy positivos, sobre todo en la Figura 71 en la confianza con la que se realizan las clasificaciones y la curva F1, que aún siendo mejorable, solo tiene problemas con falsos positivos en valores de confianza muy altos. Aparte de esto, como se puede apreciar en la Figura 70, el error de aproximación a la `Bounding Box` predicha se ha conseguido reducir mucho tanto en entrenamiento como en el conjunto de validación, que es principalmente lo que se necesitaba del modelo `YOLO` para este trabajo.

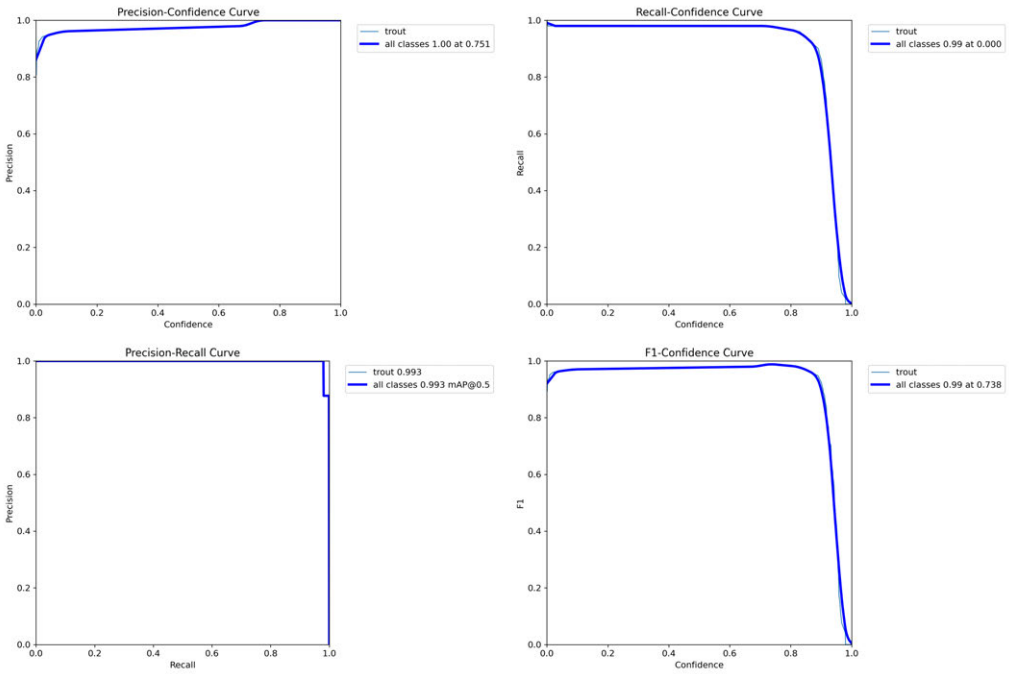


Figura 70: Gráficas de estadísticos del último entrenamiento

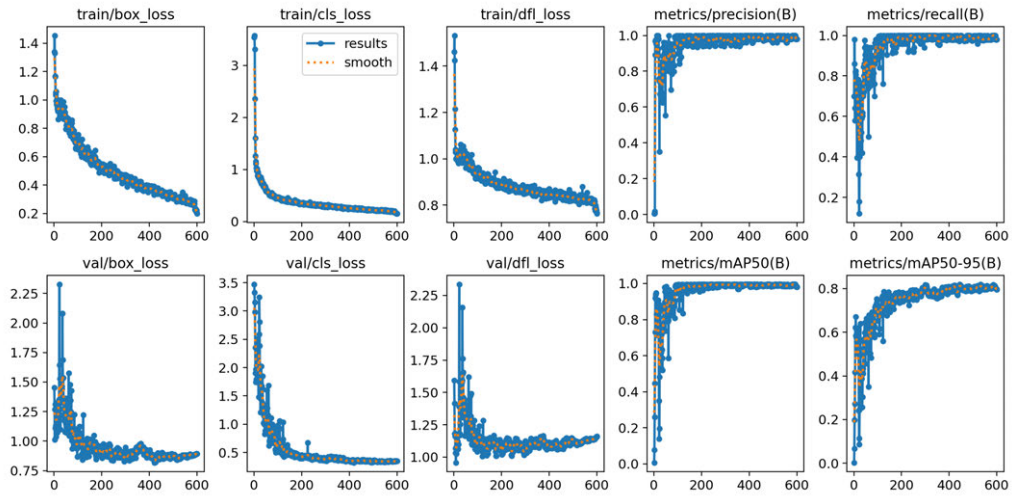


Figura 71: Gráficas de evolución en épocas del último entrenamiento

Entrenamiento OBB

- **Fecha**: 9 de mayo de 2024.
- **Descripción del conjunto de datos**: conjunto común al entrenamiento 4, 84 imágenes tanto de videos nuevos como de videos antiguos.
- **Resultados del entrenamiento**: se observan los problemas que conllevan trabajar con `Bounding Boxes` orientadas, se pierde mucho tanto en métricas como en rendimiento general. El punto positivo de este sistema es acotar lo máximo posible la `Bounding Box` al elemento, pero principalmente se observa una dificultad añadida y necesidad de mucho más entrenamiento y ajuste de hiperparámetros para conseguir sobre todo mejorar el mAP50-95.

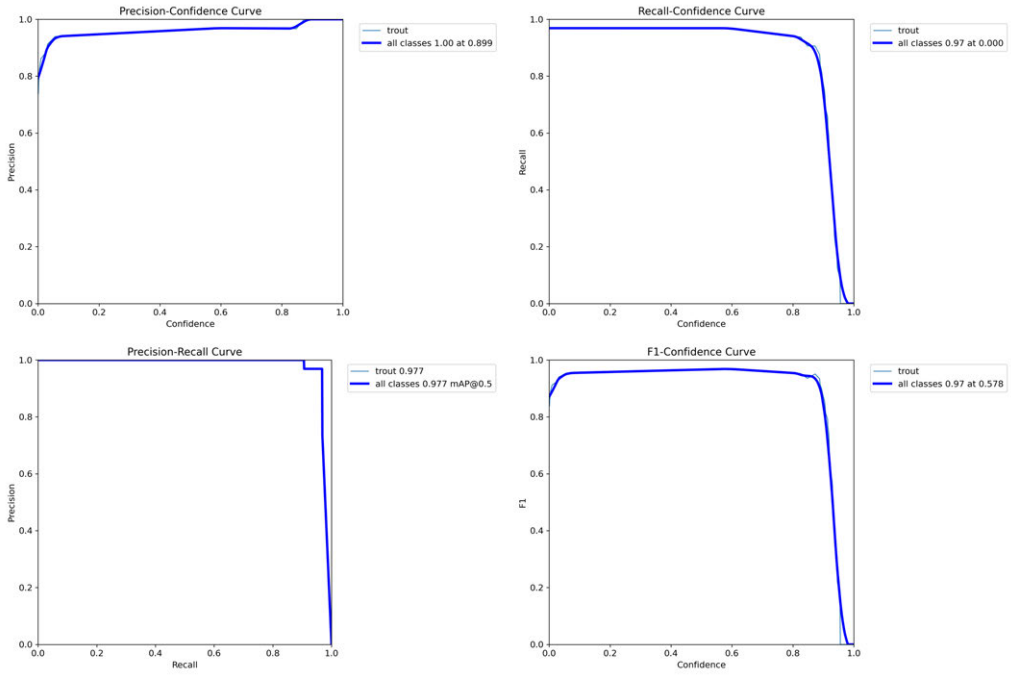


Figura 72: Gráficas de estadísticos finales del entrenamiento `OBB`

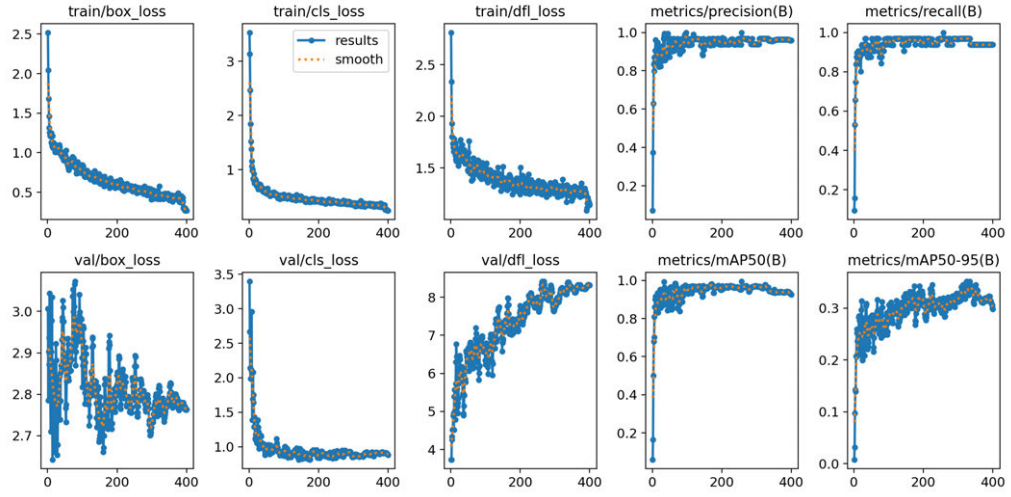


Figura 73: Gráficas de evolución en épocas del entrenamiento `OBB`

ANEXO C: Diagramas de diseño

Esquema de flujo y ventanas de GUI

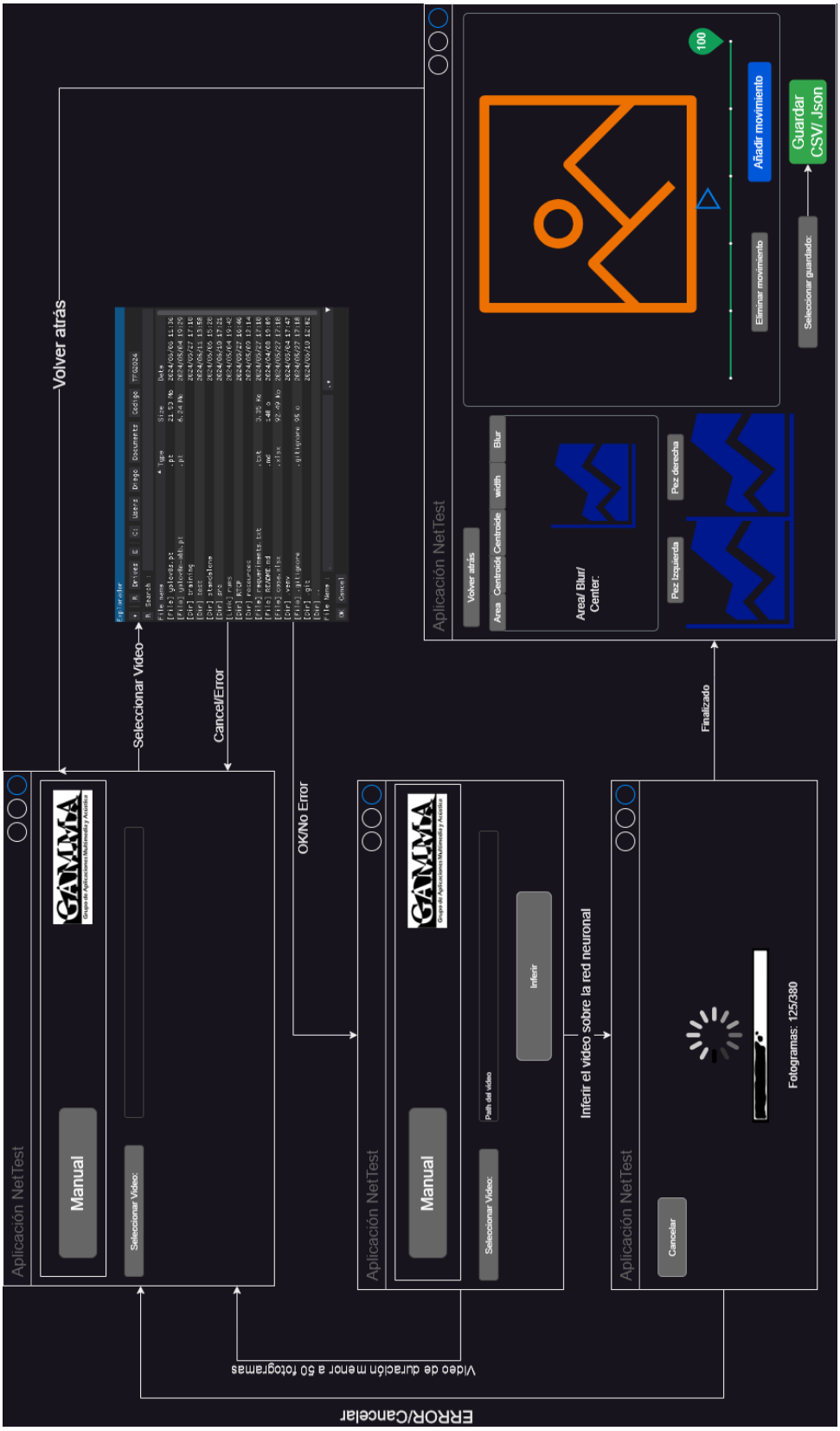


Figura 74: Esquema de flujo y ventanas de la aplicación

Diagrama de clases de la aplicación

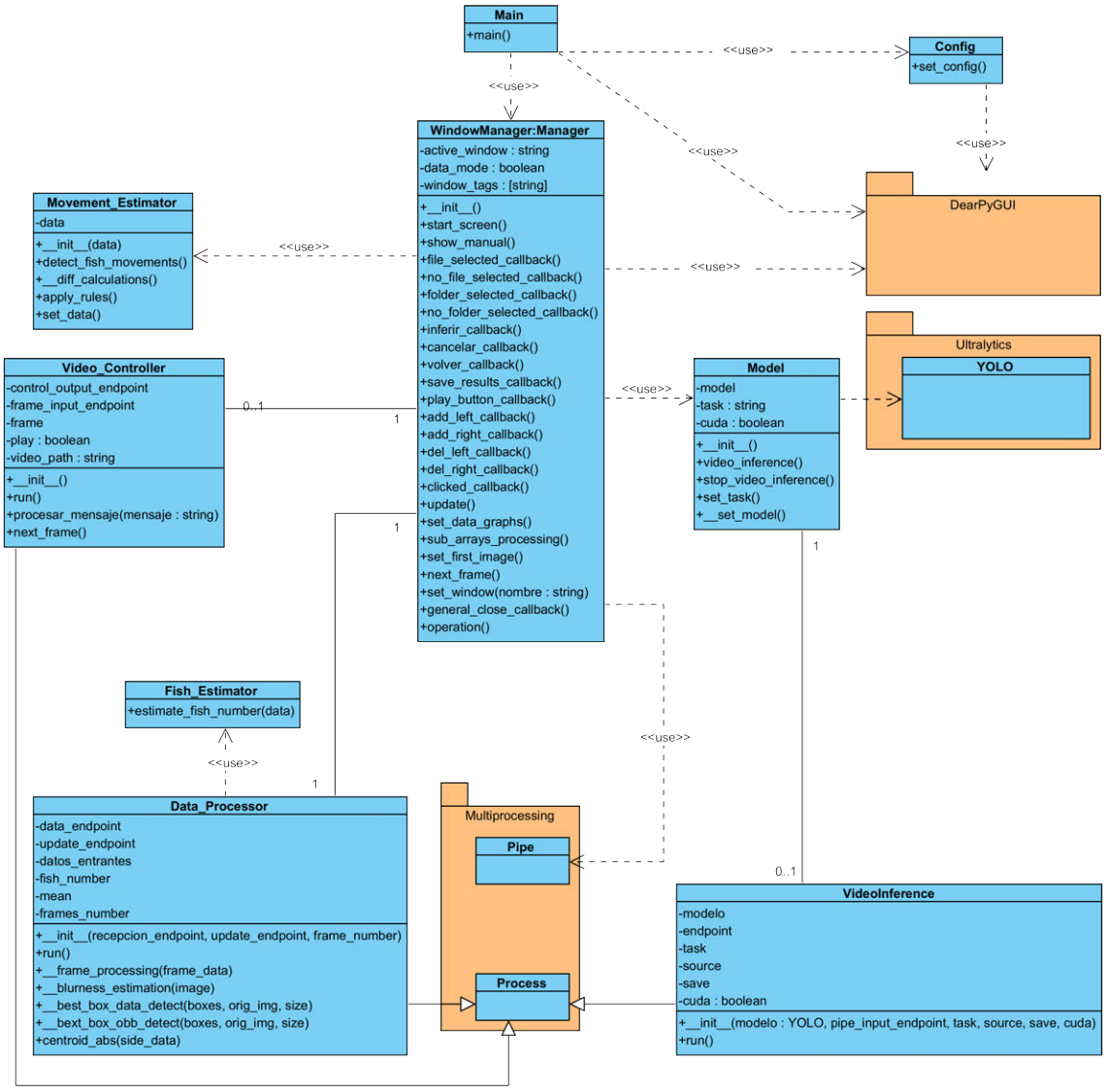


Figura 75: Diagrama de clases de la aplicación

Diagramas de secuencia

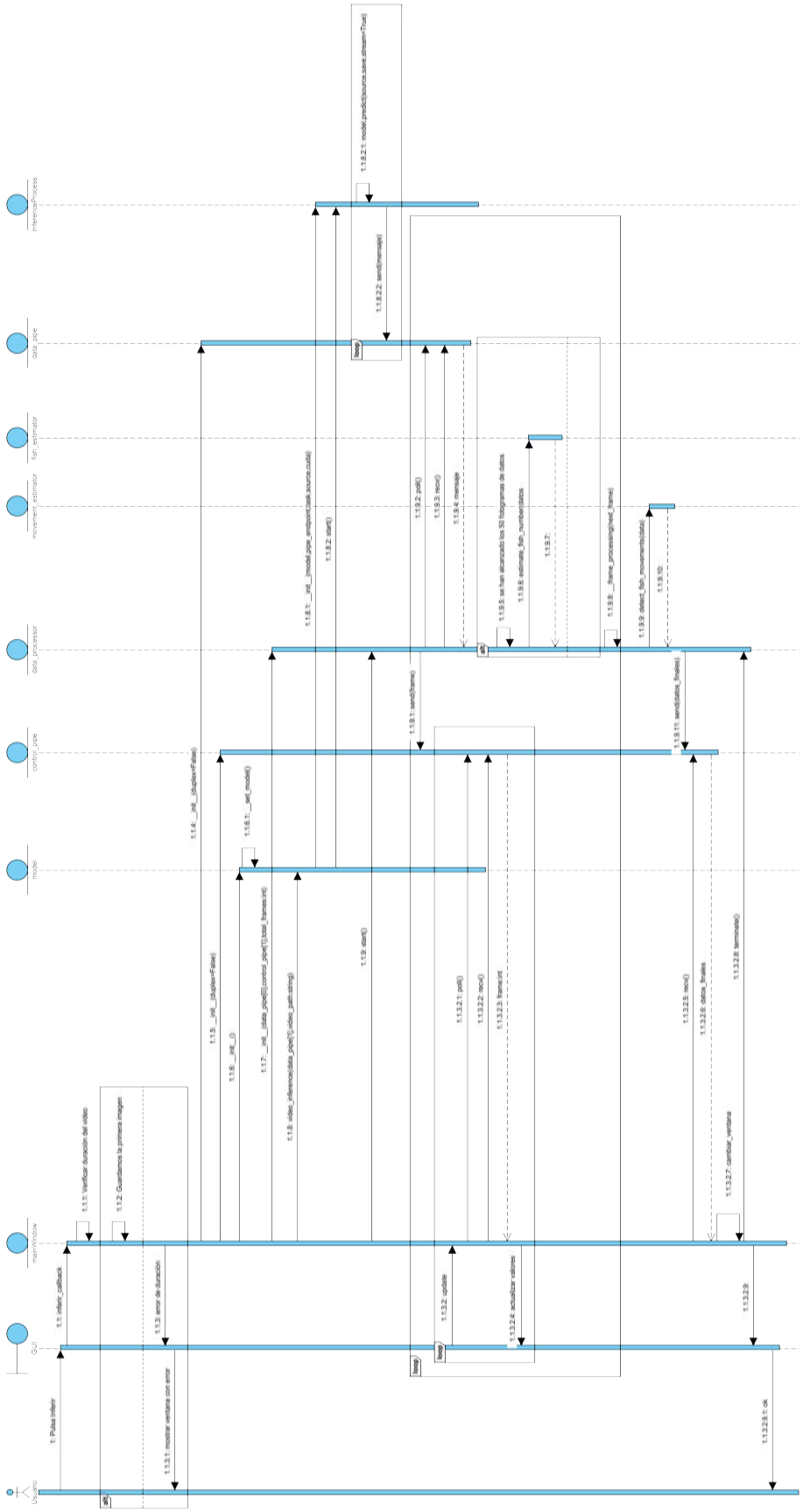


Figura 76: Diagrama de secuencia en el caso de uso de pulsar el botón de inferir

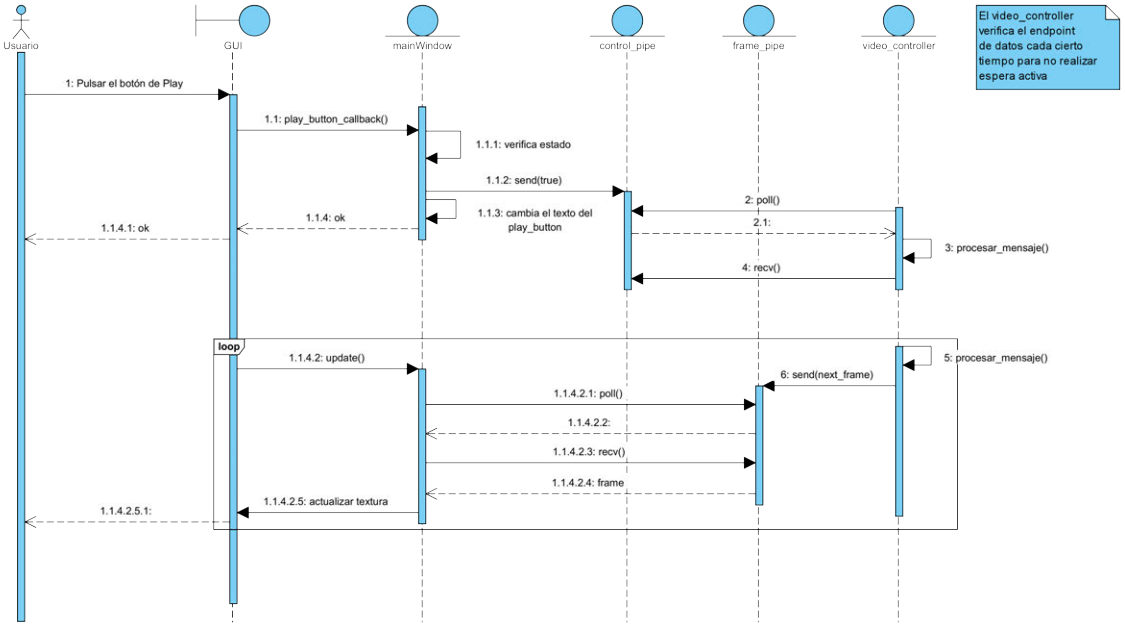


Figura 77: Diagrama de secuencia en el caso de uso de pulsar el botón de reproducción

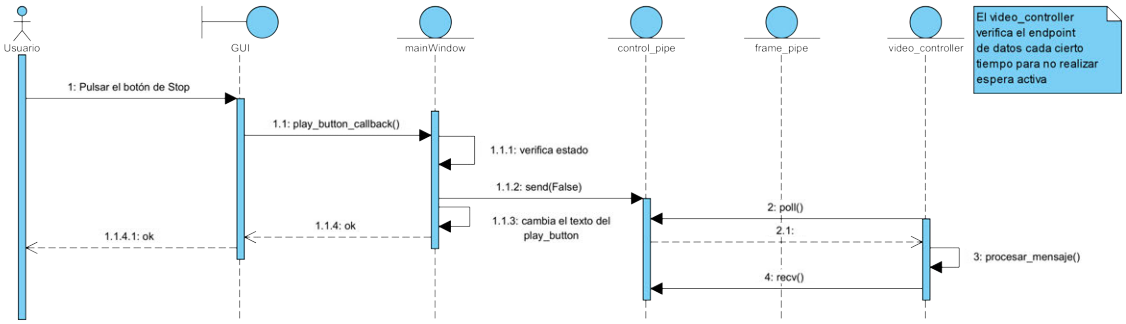


Figura 78: Diagrama de secuencia en el caso de uso de pulsar el botón de pausa

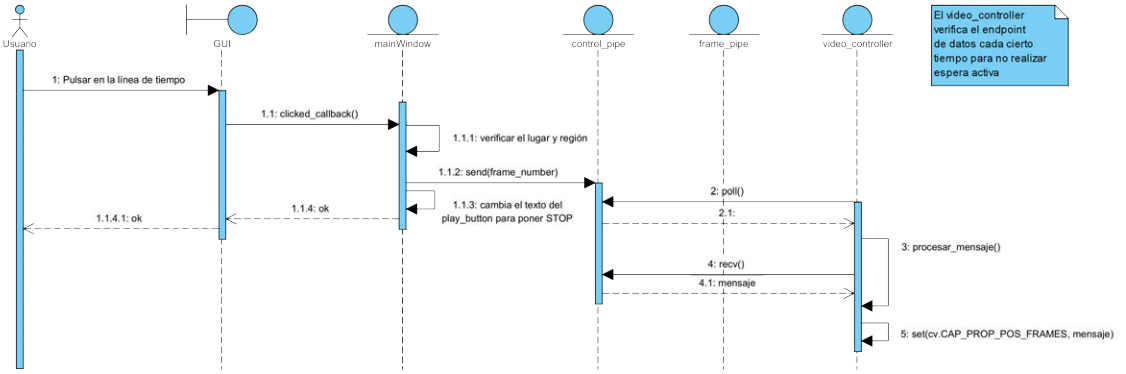


Figura 79: Diagrama de secuencia en el caso de uso de seleccionar algún fotograma en las líneas de tiempo