PROYECTO FIN DE GRADO

TÍTULO: Aplicación para la segmentación y estimación automática del número de rodaballos en tanques de cría

AUTOR: Iván Viedma González

TITULACIÓN: Grado en Ingeniería Electrónica de Comunicaciones

TUTOR: Nicolás Sáenz Lechón

DEPARTAMENTO: INGENIERÍA AUDIOVISUAL Y COMUNICACIONES

VºBº TUTOR

Miembros del Tribunal Calificador:

PRESIDENTE: Francisco Martínez Moreno

TUTOR: Nicolás Sáenz Lechón

SECRETARIO: Víctor José Osma Ruiz

Fecha de lectura: 23 de Julio de 2024

Calificación:

El Secretario,

Agradecimientos

Tras un largo camino que llega a su fin, quiero expresar mi agradecimiento a todas las personas que han sido parte de esta etapa tan importante de mi vida.

A todo el profesorado de la universidad, por su guía y por contribuir a mi formación, haciendo de mí la persona que soy hoy. Su dedicación y enseñanza han sido fundamentales en este recorrido.

A mis compañeros y amigos, gracias por su compañía y por hacer que estos años hayan sido más amenos y llevaderos. Su apoyo y amistad han sido invaluables.

A mis tutores, Nico y Juana, muchas gracias por los consejos y hacer que el desarrollo del proyecto fuera más llevadero.

Sobre todo, a mi familia: Mamá, Papá, Jorge, Alex y Elena. Gracias por estar siempre a mi lado, por su apoyo incondicional en los momentos difíciles y por compartir mis éxitos. Su amor y aliento han sido mi mayor fortaleza.

Muchas gracias.

Resumen

Actualmente, la sobrepesca y la degradación de los ecosistemas marinos han disminuido las poblaciones de peces a nivel global, amenazando la seguridad alimentaria, la economía y el equilibrio ambiental. La Organización Mundial de la Salud (OMS) reconoce la acuicultura sostenible como una práctica esencial para abordar estos desafíos.

La acuicultura es la práctica de criar organismos acuáticos como peces, mariscos y plantas acuáticas en entornos controlados para producir alimentos, restaurar poblaciones silvestres y generar beneficios económicos. Esta actividad se puede realizar en agua dulce y salada, y busca ser sostenible minimizando el impacto ambiental. En el proceso de acuicultura intervienen diferentes etapas como el acondicionamiento de los estanques o tanques para la especie a cultivar, control de entornos para promover una alimentación adecuada y balanceada o la supervisión del crecimiento de las especies.

Tener una estimación precisa del número de especies presentes en un tanque puede generar ahorros económicos y mejoras en la producción acuícola. Esto se logra mediante la optimización del racionamiento de alimento, una preparación más eficiente de las instalaciones para las siguientes etapas del ciclo productivo, y un mejor entendimiento y aprendizaje de los propios procesos productivos.

En este proyecto de fin de grado se propone desarrollar una aplicación intuitiva y de fácil uso mediante la integración de un elemento gráfico interactivos (*widgets*) sobre el visor *Napari.* Esto permitirá crear una interfaz para que los investigadores estimen el número de peces presentes en una imagen. La aplicación se basa en un modelo preentrenado de segmentación semántica utilizando redes neuronales convolucionales (CNN) y transformadores de visión (*ViT*), específicamente la arquitectura *Segment Anything Model* (SAM), que ha demostrado ser efectiva en la precisa segmentación de objetos en imágenes. Los resultados finales consistirán en comparar la salida de cuantificación de rodaballos de nuestro modelo con un banco de imágenes previamente etiquetado manualmente, donde se conoce con precisión el número de rodaballos para distintos niveles de zoom.

Las imágenes para las cuales se ha adaptado el modelo son de tanques de rodaballos en etapa larvaria. Se han logrado resultados con menos de un 2% de error y un máximo del 30% en el conteo de rodaballos.

Abstract

Currently, overfishing and the degradation of marine ecosystems have reduced fish populations globally, threatening food security, the economy, and environmental balance. The World Health Organization (WHO) recognizes sustainable aquaculture as an essential practice to address these challenges.

Aquaculture is the practice of raising aquatic organisms such as fish, shellfish, and aquatic plants in controlled environments to produce food, restore wild populations, and generate economic benefits. This activity can be carried out in both freshwater and saltwater and aims to be sustainable by minimizing environmental impact. The aquaculture process involves different stages, such as conditioning ponds or tanks for the species to be cultivated, controlling environments to promote adequate and balanced feeding, and monitoring the growth of the species.

Having an accurate estimate of the number of species present in a tank can generate economic savings and improvements in aquaculture production. This is achieved by optimizing feed rationing, more efficient preparation of facilities for the next stages of the production cycle, and better understanding and learning of the production processes.

In this final degree project, it is proposed to develop an intuitive and easy-to-use application by integrating interactive graphical elements (widgets) on the Napari viewer. This will create an interface for researchers to estimate the number of fish present in an image. The application is based on a pre-trained semantic segmentation model using convolutional neural networks (CNN) and vision transformers (ViT), specifically the Segment Anything Model (SAM) architecture, which has proven effective in precise object segmentation in images. The results will consist of comparing the turbot quantification output of our model with a manually labeled image bank, where the number of turbots is precisely known for different zoom levels.

The images for which the model has been adapted are from turbot tanks in the larval stage. Results have been achieved with less than 2% error and a maximum of 30% in turbot counting.

Índice de figuras

Índice de tablas

Lista de acrónimos

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

CNN: *Convolutional Neural Network*

FAO: Organización de las Naciones Unidas para la Alimentación y la Agricultura

IoU: Intersection over the Union

NMS: Non-Maximum Suppression

OMS: Organización Mundial de la Salud

PLN: Procesamiento del lenguaje natural

SAM: *Segment Anything Model*

ViT: *Vision Transformer*

Índice de contenidos

1. Introducción

1.1 Marco y motivación del proyecto

En los últimos años la comunidad científica ha estado advirtiendo sobre el desastre que implica la sobrepesca en los océanos, la cual se refiere a la extracción de especies marinas a una velocidad excesiva, impidiendo su capacidad de regeneración. [1]. Las consecuencias de la sobrepesca llevan a una pérdida de la biodiversidad y el desequilibrio de los ecosistemas ya que puede llegar a transformar ecosistemas marinos eficientes con alta biomasa en otros inestables e ineficientes, que son de escasa biomasa [2].

La acuicultura es la práctica de cultivar y cosechar organismos acuáticos, como peces y mariscos, en ambientes controlados de agua dulce o salada, destinada al consumo humano y la conservación de especies. [3]. En este contexto, la acuicultura se presenta como una solución viable y sostenible para mitigar los efectos negativos de la sobrepesca, contribuyendo a la conservación de los ecosistemas marinos y a la recuperación de las poblaciones de peces. Según la Organización para la Alimentación y la Agricultura (FAO), "la acuicultura puede satisfacer la creciente demanda mundial de alimentos acuáticos ya que por primera vez superó a la pesca de captura en producción de animales acuáticos, con 94,4 millones de toneladas, lo que representa el 51 % del total mundial y un récord del 57 % de la producción destinada al consumo humano" [4]. Esta afirmación se refleja en la Figura 1, una estadística que compara la producción acuícola y de pesca en los últimos años donde se aprecia que la acuicultura ha sido crucial para satisfacer la creciente demanda mundial de productos pesqueros sostenibles, reduciendo la presión sobre los peces silvestres y contribuyendo a la seguridad alimentaria y económica global.

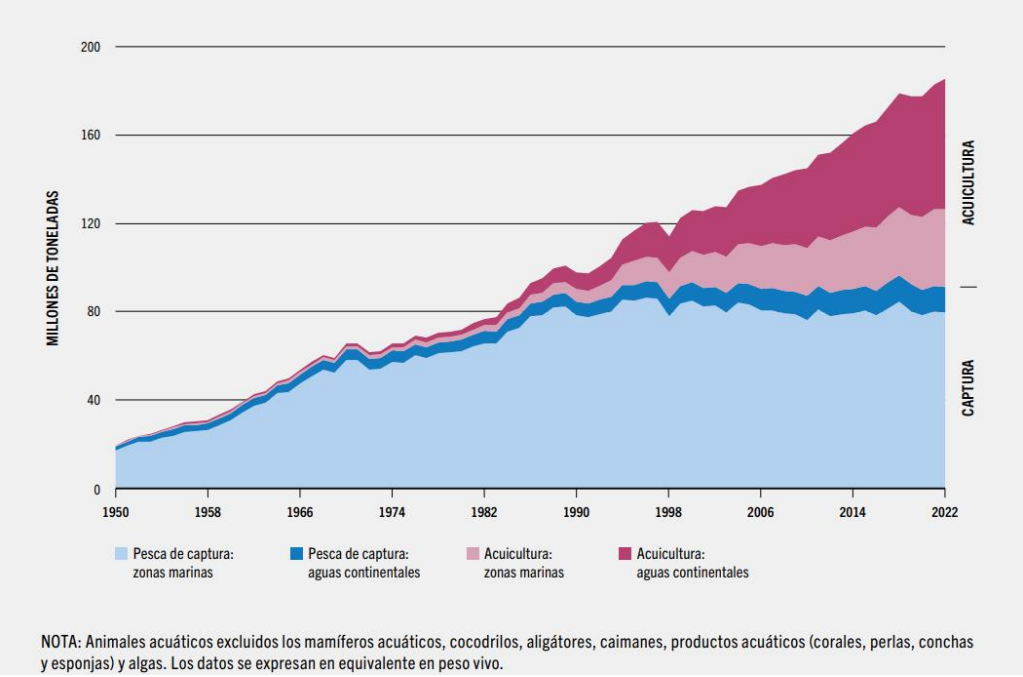

Figura 1 – Producción pesquera y acuícola de animales acuáticos (1950 – 2022) [4]

En la acuicultura se definen tres fases principales: reproducción y cría, preengorde y engorde [5]. La mejora continua en todas las etapas de producción acuícola que ocurren en ellas es fundamental para promover una acuicultura sostenible destacando el acondicionamiento de los tanques de cultivo y la supervisión del crecimiento de las especies.

Tener una estimación precisa del número de especímenes presentes en los tanques es crucial para mantener una densidad de siembra adecuada y optimizar los procesos de alimentación. Esta información facilita el control del stock, el monitoreo del crecimiento y desarrollo de los organismos, y mejora la eficiencia en el manejo sanitario. Además, permite una planificación precisa de la cosecha y la comercialización, contribuyendo así a una operación acuícola más eficiente, sostenible y rentable.

En este contexto, la detección de objetos en el procesamiento de imágenes desempeña un papel crucial. En los últimos años, ha habido avances significativos en esta área de investigación, dando lugar a técnicas cada vez más sofisticadas basadas en procesamiento de imágenes y aprendizaje automático. Estas técnicas permiten identificar y delimitar objetos de interés en imágenes, como peces en tanques acuícolas, de manera precisa.

En este proyecto, se llevará a cabo la estimación del número de rodaballos que se encuentran en etapas tempranas de desarrollo, con una edad máxima de 26 días. Los rodaballos están presentes en dos tipos de imágenes distintas según su nivel de zoom: imágenes con zoom e imágenes sin zoom. Estas imágenes se obtienen de videos proporcionados por el CITSEM, los cuales fueron grabados por cámaras cenitales instaladas en tanques de cría en plantas de acuicultura de la empresa Nueva Pescanova. Los tanques de cultivo son infraestructuras esenciales en la acuicultura moderna que permiten criar peces de manera controlada y sostenible. El diseño y la gestión efectiva de estos tanques juegan un papel crucial en el éxito de las operaciones acuícolas al garantizar condiciones óptimas para el crecimiento y la salud de las especies cultivadas.

La cuantificación automática de los rodaballos proporcionará rápidamente información sobre el número estimado de larvas en el tanque, permitiendo un mejor control de la alimentación y una planificación más eficiente de la transición a otras etapas según su tamaño.

1.2 Objetivos técnicos

El propósito principal de este proyecto es desarrollar una aplicación en Python que utilice métodos avanzados de procesamiento de imágenes para realizar el conteo automático de rodaballos en imágenes capturadas por una cámara cenital en tanques de cultivo. La aplicación no solo contará el número de objetos segmentados en la imagen de entrada, sino que también presentará los resultados de manera intuitiva y accesible, facilitando su interpretación y uso por parte de los usuarios.

Para alcanzar el objetivo planteado se ha dividido este en 4 secundarios:

O.1. Recopilación y preparación de datos: Recolectar imágenes de tanques de cultivo de rodaballos y etiquetarlas para ajustar los parámetros del modelo.

O.2. Implementación y optimización del modelo: Utilizar un modelo preentrenado de segmentación y ajustar sus parámetros para mejorar la segmentación de los tanques de cultivo.

O.3. Análisis y mejora de resultados: Evaluar el rendimiento del modelo, identificar áreas de mejora y aplicar funciones de postprocesamiento para refinar las máscaras segmentadas.

O.4. Desarrollo de una interfaz intuitiva: Crear una aplicación fácil de usar que permita visualizar los resultados de segmentación en formato de "pila de capas" para una mejor comprensión y análisis de los resultados.

1.3 Estructura del resto de la memoria

La presente memoria está dividida en estos capítulos:

- Marco tecnológico: En esta sección se proporciona una visión general de los conocimientos y tecnologías relevantes para el proyecto, ofreciendo contexto sobre las herramientas y enfoques utilizados. Se definirán conceptos técnicos fundamentales como redes neuronales convolucionales (CNN), transformadores de visión (ViT), mecanismos de atención, segmentación semántica y el lenguaje de programación Python. Estos términos proporcionarán una base clara para comprender el desarrollo posterior.

- Requisitos y limitaciones de diseño: Aquí se establecen las condiciones y especificaciones que el sistema debe cumplir, brindando un marco de referencia para el desarrollo.

- Solución propuesta: En este apartado se detalla la solución desarrollada para cumplir con los objetivos del proyecto, describiendo el modelo, los procesos y la interfaz de usuario implementados. El desarrollo se divide en varias etapas clave. En la primera fase, se lleva a cabo el etiquetado manual de imágenes de tanques de rodaballos para los distintos niveles de zoom. Posteriormente, se desarrollan funciones de preprocesamiento de imágenes diseñadas para optimizar la entrada del modelo. Luego, se analiza y selecciona el modelo de segmentación más adecuado. Se procede a la evaluación y ajuste de los parámetros del modelo para alcanzar una configuración óptima adaptada a las imágenes específicas del proyecto. Además, se aplican funciones de postprocesamiento para mejorar y refinar los resultados obtenidos. Finalmente, se implementa la interfaz de la aplicación utilizando el visor de imágenes Napari, mostrando las diferentes fases del proceso de segmentación en formato de "pila de capas", facilitando la visualización y análisis de cada etapa del procesamiento de imágenes.

- Resultados obtenidos: Se muestran los resultados de las pruebas realizadas contrastadas contra el banco de imágenes previamente etiquetado.

- Presupuesto: Se presenta el costo financiero necesario para llevar a cabo el proyecto.

- Impacto del proyecto: Aquí se analizan las posibles repercusiones del proyecto en diferentes áreas y cómo contribuye a los Objetivos de Desarrollo Sostenible (ODS), destacando su importancia y beneficios.

- Conclusiones: Después de presentar los resultados, se exponen las conclusiones obtenidas considerando los objetivos y limitaciones del proyecto, además de sugerir mejoras y posibles líneas futuras de trabajo.

- Referencias y bibliografía: Se incluye una lista de las fuentes consultadas en la memoria, garantizando la fiabilidad y fundamentación del trabajo realizado.
- ANEXO I: Incluye una lista con las funciones desarrolladas más relevantes del proyecto.
- ANEXO II: Incluye un manual de usuario de instalación de la aplicación.

2. Marco tecnológico

2.1 Deep learning

El *deep learning* (aprendizaje profundo) es una subdisciplina del machine learning (aprendizaje automático) dentro del campo más amplio de la inteligencia artificial (IA) (ver Figura 2). Antes de dar una definición de *deep learning*, debemos conocer los principios básicos y las propiedades esenciales del machine learning, que sientan las bases para entender cómo funcionan las técnicas más avanzadas.

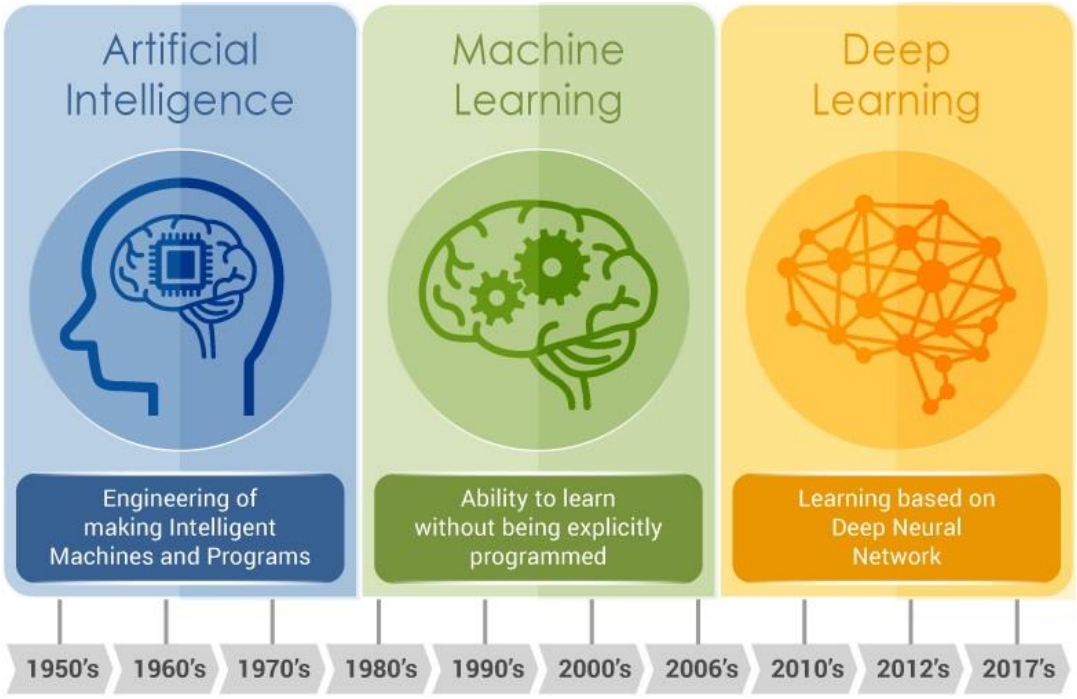

Figura 2 – Cronología de los distintos tipos de algoritmos de inteligencia artificial [6]

El aprendizaje automático es un método de análisis de datos que permite la creación automática de modelos. Es una subdisciplina de la inteligencia artificial que se fundamenta en la premisa de que los sistemas pueden adquirir conocimientos a partir de datos, reconocer patrones y tomar decisiones con una mínima intervención humana. [7]. Cualquier cosa que se pueda almacenar digitalmente puede servir como dato para el aprendizaje automático. Al detectar patrones en estos datos, los algoritmos mejoran su rendimiento en la ejecución de tareas específicas. Aprenden de forma autónoma a realizar estas tareas o hacer predicciones, y mejoran su desempeño con el tiempo. [8].

El aprendizaje en *machine learning* es el proceso mediante el cual un algoritmo o modelo mejora su rendimiento automáticamente a medida que se le proporcionan más datos.

Atendiendo a la forma de aprendizaje hay diferentes tipos [9]:

1) **Aprendizaje supervisado:** El aprendizaje supervisado es una técnica en la que el modelo se entrena utilizando un conjunto de datos con etiquetas. Cada entrada tiene una etiqueta de salida conocida, y el propósito es que el modelo aprenda a predecir la etiqueta correcta para nuevos datos no vistos. Este tipo de aprendizaje se divide en dos categorías principales:

   - Clasificación: El modelo aprende a asignar una etiqueta a cada entrada basada en sus características. Por ejemplo, reconocimiento facial para identificación de personas.
   - Regresión: El modelo predice un valor numérico continúo basado en los datos de entrada. Por ejemplo, estimar el precio de una casa según sus características.

2) **Aprendizaje no supervisado:** En el aprendizaje no supervisado, el modelo intenta encontrar patrones y estructuras ocultas en datos sin etiquetar. Este enfoque es útil cuando no se dispone de etiquetas de datos y se desea descubrir relaciones intrínsecas en los datos. Los dos métodos más comunes son:

   - *Clustering*: Agrupa los datos en diferentes clústeres o grupos basados en su similitud. Por ejemplo, el algoritmo *k-means* divide los datos en k clústeres.
   - Reducción de dimensionalidad: Simplifica los datos al reducir el número de variables bajo consideración. Técnicas como el Análisis de Componentes Principales (PCA) ayudan a encontrar las direcciones principales de variación en los datos.

3) **Aprendizaje "semisupervisado":** Este tipo de aprendizaje se sitúa entre el supervisado y el no supervisado, ofreciendo un equilibrio entre ambos enfoques. Durante el proceso de entrenamiento, se utiliza un conjunto pequeño de datos etiquetados para guiar la clasificación y la extracción de características de un conjunto mucho más grande de datos no etiquetados. Este método resulta útil cuando no hay suficientes datos etiquetados disponibles para entrenar un algoritmo de manera completamente supervisada.

4) **Aprendizaje por refuerzo:** El aprendizaje por refuerzo se centra en cómo los agentes deben tomar acciones en un entorno para maximizar alguna noción de recompensa acumulada. A diferencia del aprendizaje supervisado y no supervisado, el aprendizaje por refuerzo se basa en la interacción con el entorno y la obtención de retroalimentación en forma de recompensas o castigos. Este tipo de aprendizaje es común en robótica y juegos.

5) **Aprendizaje *zero-shot***: es una técnica de aprendizaje automático donde un modelo puede realizar tareas o hacer predicciones sobre clases que no fueron explícitamente parte de su entrenamiento inicial. Esto se logra utilizando información sobre las relaciones entre clases o conceptos dentro de los datos disponibles. En lugar de entrenar con ejemplos específicos de cada clase, el modelo generaliza y aplica el conocimiento aprendido para hacer predicciones sobre nuevas clases o escenarios, basándose en la comprensión de las relaciones subyacentes entre las clases conocidas. Este enfoque es especialmente útil cuando la obtención de datos etiquetados para cada clase sería difícil o si el número de datos etiquetados fuera muy reducido [10].

Los algoritmos de machine learning son métodos matemáticos y estadísticos diseñados para identificar patrones en los datos. Los más comunes son [11]:

1) **Algoritmos de regresión lineal:** Es un método de aprendizaje supervisado que facilita la comprensión de las relaciones entre los datos. La regresión lineal se emplea para estimar el valor de una variable dependiente basándose en el valor de una variable independiente (ver Figura 3)

2) **Algoritmos de regresión logística:** Es un método de aprendizaje supervisado aplicado en la clasificación de datos. La regresión logística se utiliza principalmente para problemas de clasificación binaria, donde el objetivo es estimar la probabilidad de que una observación pertenezca a una de dos clases (ver Figura 4).

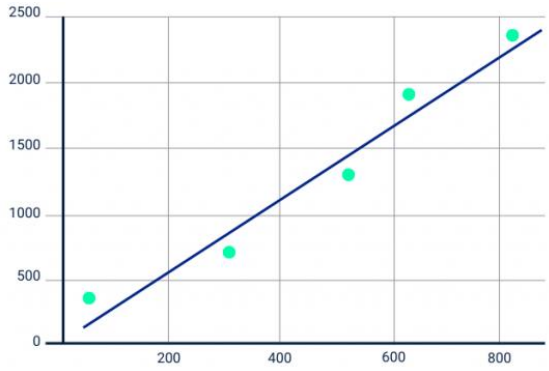

Figura 3 – Regresión lineal [11]

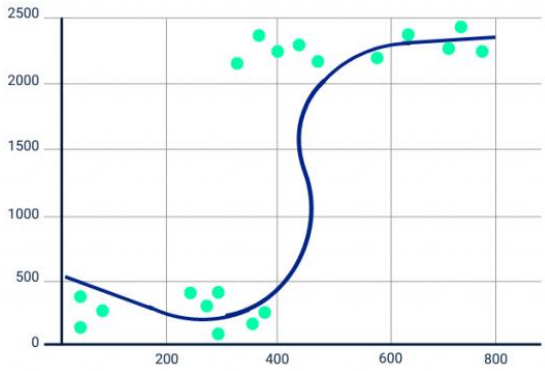

Figura 4 – Regresión logística [11]

3) **Arboles de decisión**: Los árboles de decisión se emplean para abordar problemas tanto de regresión como de clasificación. Su estructura jerárquica en forma de árbol consta de un nodo raíz en la parte superior, que sirve como punto inicial, seguido de ramas o divisiones que se conectan con otros nodos (hojas), finalizando en un nodo terminal.

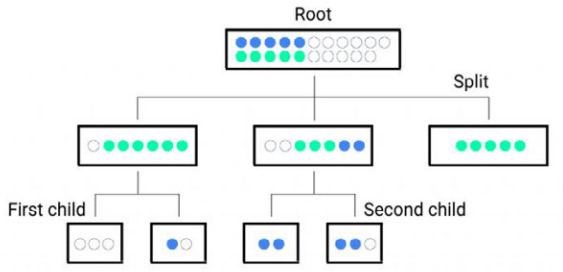

Figura 5 – Árbol de decisión [11]

4) **Redes neuronales:** Son modelos algorítmicos con estructuras en forma de red que incluyen varias capas. La primera capta datos de entrada, capas ocultas procesan estos datos para extraer conclusiones, y la capa final asigna probabilidades a cada una de estas conclusiones.

El *deep learning* es una versión mejorada del *machine learning*. Utiliza una técnica que le da mayor capacidad para detectar los patrones incluso los más sutiles. Esta técnica se llama red de neuronas profundas, se les llamada “profundas” debido al gran número de capas de nodos de cálculo que constituyen estas redes y que trabajan conjuntamente para tratar los datos y hacer predicciones. Estas redes de neuronas se inspiran directamente en el funcionamiento del cerebro humano, los nodos de cálculo se pueden comparar a las neuronas, y la red misma se parece al cerebro.

El objetivo del aprendizaje profundo es aprender automáticamente representaciones de datos de alto nivel a partir de los datos de entrada, sin necesidad de una programación explícita de características. Esto permite una mayor capacidad para comprender y generalizar a partir de datos complejos y de gran escala. Sus aplicaciones son diversas y van desde el reconocimiento de imágenes, procesamiento de lenguaje natural, hasta juegos y robótica, entre otros campos.

2.2 Redes neuronales

Como se ha comentado en el apartado anterior las redes neuronales son uno de los métodos de *machine learning* utilizados para identificar patrones con el objetivo de que nuestro sistema aprenda.

El objetivo principal de las redes neuronales es emular el funcionamiento del cerebro humano en términos de procesamiento de información y aprendizaje. Estas estructuras computacionales están diseñadas para aprender de datos mediante la identificación de patrones y relaciones complejas. Esto permite a las redes neuronales realizar tareas como reconocimiento de patrones en imágenes, procesamiento de lenguaje natural, predicción de series temporales y muchas otras aplicaciones. El objetivo es lograr que la red neuronal aprenda y generalice a partir de datos de entrenamiento, de manera que pueda realizar predicciones precisas y tomar decisiones informadas en nuevos datos no vistos durante el entrenamiento.

Una red de neuronas "profunda" está compuesta por múltiples capas ocultas que permiten afinar los resultados de la anterior (ver Figura 6). Es la que se utiliza en el campo del aprendizaje profundo. Cada nodo (neurona) en la red neuronal está interconectado con otros nodos mediante conexiones. Cada enlace de estos nodos tiene asignado un valor numérico (peso) que determina la intensidad de la conexión, y durante el proceso de aprendizaje, estos valores se modifican para optimizar la exactitud y el desempeño de las predicciones del modelo neuronal.

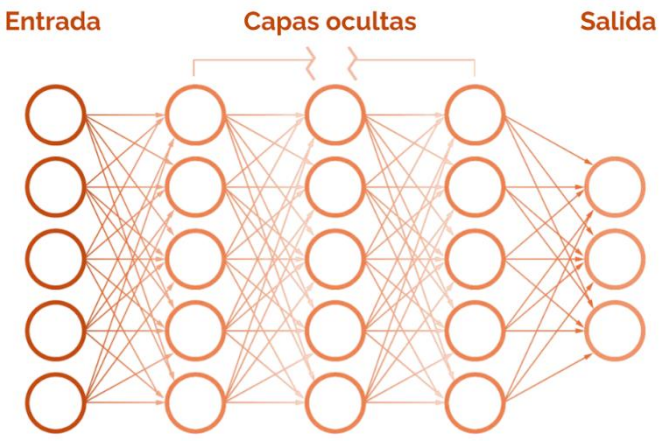

Figura 6 – Arquitectura red neuronal profunda [12]

Se encuentran diferentes tipos de redes neuronales en función del uso del tratamiento de los datos [13] :

- **Redes neuronales *feedforward***: La información fluye en una dirección, desde la entrada hasta la salida, sin ciclos.
- **Redes neuronales recurrentes:** Permiten conexiones retroalimentadas, lo que les permite procesar secuencias de datos y tener memoria.
- **Redes neuronales convolucionales (*CNNs*):** Especializadas en datos estructurados como imágenes, aplicando filtros convolucionales para capturar patrones espaciales.
- **Redes neuronales generativas (*GANs*):** Utilizadas para la generación de datos nuevos que parecen provenir de un conjunto de datos original.

2.2.1 Redes neuronales convolucionales (CNNs)

Las redes neuronales convolucionales (*CNNs*) son un tipo avanzado de algoritmo de aprendizaje automático, especialmente diseñado para procesar y comprender imágenes. Están inspiradas en la forma en que funciona el sistema visual humano, organizando capas de neuronas que detectan características como bordes, formas y patrones en diferentes partes de una imagen. Utiliza filtros o "núcleos" para escanear la imagen y extraer automáticamente estas características, luego pasa por capas que agrupan y resumen la información antes de hacer predicciones finales. Las CNNs son fundamentales en aplicaciones como clasificación de imágenes, detección de objetos y reconocimiento facial debido a su capacidad para aprender y adaptarse a partir de grandes volúmenes de datos visuales.

La arquitectura de una red neuronal convolucional (CNN) se compone de varias capas interconectadas diseñadas para procesar y extraer características de imágenes de manera eficiente (ver Figura 7). A continuación, se describen los componentes principales de su arquitectura [14]:

- **Capas de Convolución:** Utilizan filtros para detectar características como bordes, texturas o formas en la imagen.
- **Funciones de Activación:** Introducen no linealidad para capturar relaciones complejas entre las características.
- **Capas de Agrupamiento (*Pooling*):** Reducen la dimensionalidad de la representación de características, conservando lo más relevante.
- **Capas Totalmente Conectadas:** Transforman los datos para hacer predicciones finales, aprendiendo relaciones entre características.
- **Capa de Salida:** Produce la predicción final, utilizando funciones de activación.
- **Regularización y Optimización:** Técnicas como *dropout* y regularización L2 se utilizan para mejorar el rendimiento y evitar el sobreajuste.

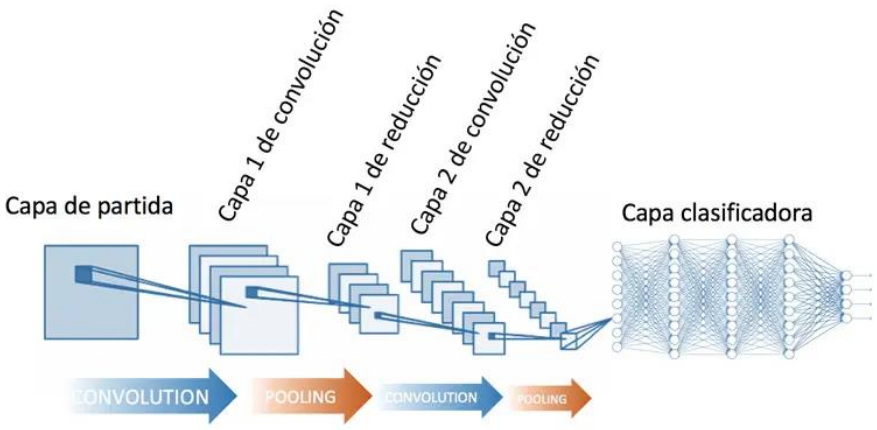

Figura 7 – Arquitectura de red neuronal convolucional (*CNN*) [15]

2.2.2 Redes neuronales generativas (GANs)

Las redes neuronales generativas o también conocidas como *Generative Neural Networks* (*GANs*) son un tipo especial de modelo de inteligencia artificial compuesto por dos redes neuronales: el generador y el discriminador (ver Figura 8). El generador crea datos nuevos que se parecen a los datos de entrenamiento original, mientras que el discriminador intenta distinguir entre los datos reales y los generados. Ambas redes se entrenan de manera competitiva, mejorando iterativamente su desempeño. Este proceso permite que el generador aprenda a producir muestras que son indistinguibles de las reales para el discriminador [16].

Las *GANs* han demostrado ser efectivas en una amplia gama de aplicaciones, como la generación de imágenes, la síntesis de videos, la mejora de imágenes, y más recientemente, en el procesamiento de lenguaje natural. Por el contrario, su arquitectura presenta una principal desventaja llamada "Colapso de modo" que consiste en que durante el proceso de entrenamiento sincronizado de ambas redes el generador puede inclinarse hacia la reproducción repetitiva de un solo patrón específico, el cual puede engañar al discriminador.

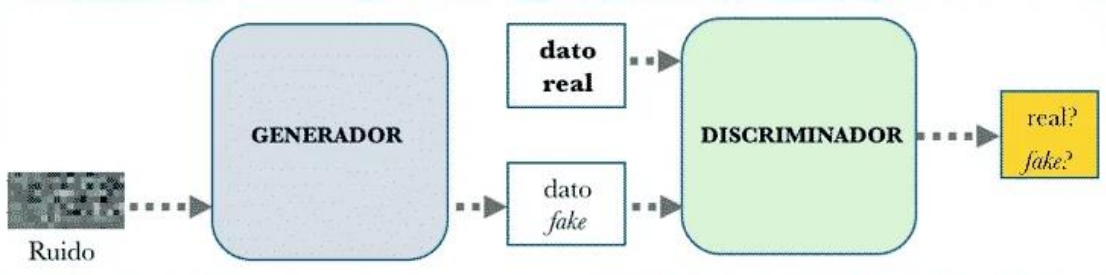

Figura 8 – Arquitectura de red neuronal generativa (*GAN*) [17]

2.3 Transformadores de visión (ViT)

ViT (Vision Transformer) es una arquitectura avanzada de aprendizaje profundo desarrollada por Google en 2020 para aplicaciones en visión por computadora. Basada en los principios de los transformadores, *ViT* revoluciona la manera en que se procesan y comprenden las imágenes al utilizar mecanismos de atención personalizados [18].

La arquitectura de transformadores es un tipo de modelo de aprendizaje automático que se ha destacado especialmente en tareas de procesamiento de lenguaje natural (PNL) y posteriormente en visión por computadora. A diferencia de las redes neuronales convolucionales (*CNNs*) tradicionales, que se basan en operaciones locales como convoluciones y *pooling*, los transformadores se centran en el mecanismo de atención *(self-attention*) para capturar características a diferentes escalas y niveles de abstracción, es por ello por lo que son una herramienta potente para la comprensión de imágenes [19].

La estructura de *ViT* se compone principalmente de dos componentes clave: la red de visión y la red de transformadores [20] (ver Figura 9).

- **Red de visión:** La red de visión de ViT enfrenta la complejidad de las imágenes dividiéndolas en pequeños parches, lo cual evita el cálculo de autoatención sobre la imagen completa en la siguiente etapa. Este enfoque reduce drásticamente la carga computacional al considerar los parches como la unidad de análisis en lugar de los píxeles individuales. Cada parche se transforma en un vector unidimensional para representar la imagen como vectores de características, optimizando así el procesamiento y permitiendo una gestión más eficiente de la información visual.

- **Red de Transformadores (*Transformer Encoder*):** El bloque de transformador en ViT consiste en múltiples capas de autoatención y redes neuronales totalmente conectadas. Esta estructura permite al modelo capturar relaciones a largo plazo entre los parches de la imagen, lo que resulta fundamental para entender contextos amplios y relaciones complejas dentro de la escena visual. Este enfoque facilita la integración de información, optimizando así la capacidad del modelo para procesar y comprender imágenes de manera más efectiva.

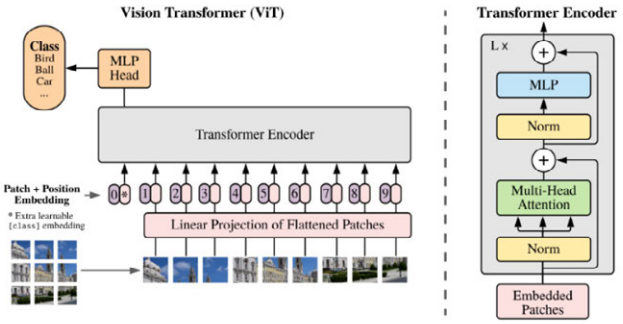

Figura 9 – Arquitectura *ViT* [21]

Este enfoque difiere de las CNN tradicionales al reemplazar convoluciones locales con atención global entre todos los parches, mejorando la capacidad del modelo para manejar características visuales complejas y aprender representaciones significativas de las imágenes.

2.3.1 Mecanismos de atención

Los mecanismos de atención son componentes fundamentales en los modelos de inteligencia artificial basados en redes neuronales, como los transformadores. Estos mecanismos permiten que el modelo seleccione de manera ponderada qué partes de la entrada deben recibir más atención o importancia durante el proceso de aprendizaje o inferencia. La atención permite al modelo enfocarse en características específicas de los datos de entrada, facilitando la captura de relaciones importantes y mejorando la capacidad de generalización del modelo. Esto es especialmente útil en tareas que requieren comprensión de contexto o relaciones a largo plazo entre elementos de la entrada, como en el procesamiento del lenguaje natural y el análisis de imágenes [22].

2.4 Segmentación Semántica

La segmentación de imágenes es el proceso de dividir una imagen en múltiples segmentos o regiones, identificando y agrupando píxeles que comparten características similares. El objetivo es simplificar la representación de la imagen para facilitar su análisis, permitiendo la identificación precisa de objetos y sus límites.

La segmentación semántica es una técnica de visión por computadora que clasifica cada píxel de una imagen en una clase específica (etiqueta), como personas, árboles o automóviles (ver Figura 10). El objetivo es identificar y delinear objetos y regiones en la imagen, creando un mapa detallado donde cada píxel se asocia con una etiqueta semántica. Esto permite una comprensión profunda del contexto visual y se utiliza en aplicaciones como la conducción autónoma, la interpretación de imágenes médicas y la realidad aumentada.

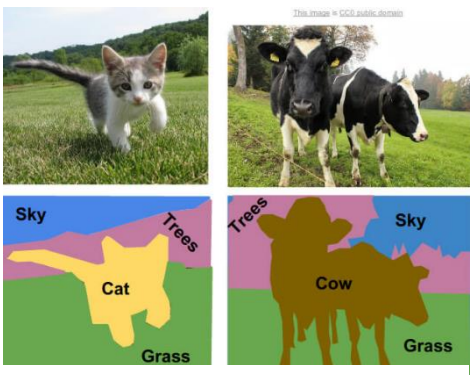

Figura 10 – Segmentación semántica [23]

2.5 Lenguaje de programación Python

Python es un lenguaje de programación de alto nivel, ampliamente utilizado por su simplicidad y legibilidad. Es conocido por su sintaxis clara, lo que facilita su aprendizaje y uso. Python es versátil y se aplica en diversas áreas como desarrollo web, ciencia de datos, inteligencia artificial, automatización, y más. Su amplia biblioteca de módulos y su comunidad activa lo convierten en una herramienta poderosa para desarrolladores y científicos que lo ha posicionado como uno de los lenguajes de programación de referencia [24] (ver Figura 11).

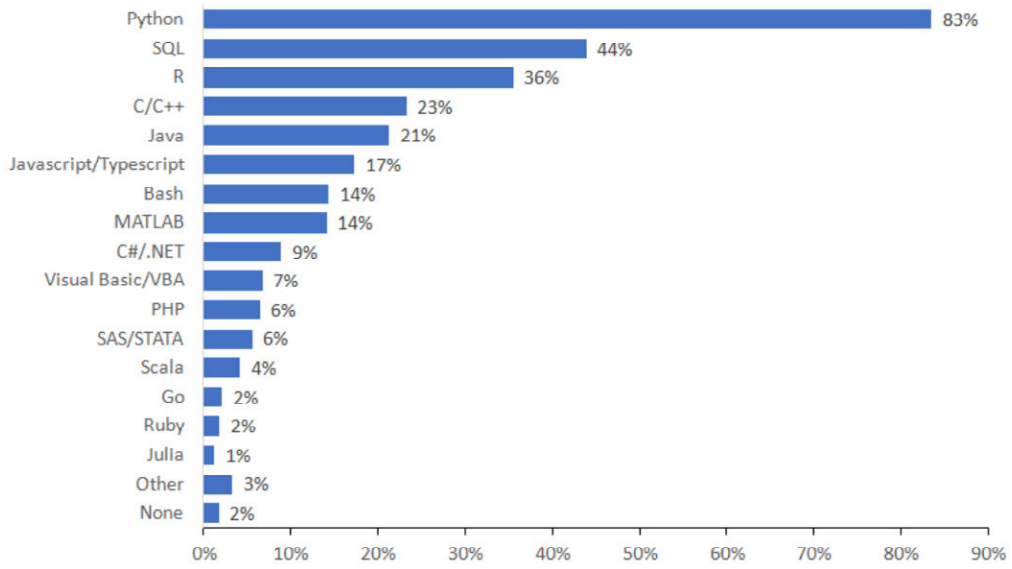

Figura 11 – Usabilidad de los lenguajes de programación [25]

La instalación de bibliotecas en Python proporciona acceso instantáneo a una amplia gama de funcionalidades especializadas sin necesidad de desarrollar desde cero. Esto permite a los desarrolladores aprovechar soluciones probadas y optimizadas para diversas tareas, como análisis de datos, machine learning, visualización, procesamiento de imágenes, entre otras.

2.5.1 Librería Pytorch: torch y torchvisión

Torch y *torchvision* son dos bibliotecas importantes dentro del ecosistema de *PyTorch*, una popular plataforma de código abierto para el desarrollo de aplicaciones en inteligencia artificial y aprendizaje profundo.

Torch es un marco de trabajo de aprendizaje profundo desarrollado principalmente por *Facebook's AI Research lab (FAIR)*. *Torch* proporciona una interfaz flexible y fácil de usar para la creación y entrenamiento de modelos de aprendizaje profundo. Se destaca por su capacidad para crear gráficos computacionales dinámicos, lo que facilita la construcción y la depuración de modelos, así como su integración fluida con Python.

Torchvision es una biblioteca complementaria de *Torch* diseñada específicamente para el procesamiento y la manipulación de imágenes y vídeos. Proporciona herramientas y utilidades para cargar datos de conjuntos de datos estándar, preprocesar imágenes para su uso en modelos de aprendizaje profundo, aplicar transformaciones de datos, y más. Además, *Torchvision* incluye modelos de redes neuronales preentrenados para tareas comunes de visión por computadora, como clasificación de imágenes, detección de objetos, segmentación semántica, entre otras [26].

PyTorch es ampliamente utilizado en investigación y producción para una variedad de aplicaciones de aprendizaje automático, incluyendo visión por computadora, procesamiento de lenguaje natural, reconocimiento de voz, y más.

PyTorch está diseñado para ser compatible con CUDA, una plataforma de computación paralela de NVIDIA que permite ejecutar operaciones intensivas en paralelo en *GPU* (*Graphics Processing Unit*). Cuando se instala PyTorch, si se dispone de un entorno compatible con CUDA (una *GPU* NVIDIA y los controladores correspondientes), PyTorch utilizará automáticamente CUDA para acelerar operaciones como el entrenamiento de redes neuronales, el cálculo de gradientes y la evaluación de modelos.

En resumen, Pytorch proporciona herramientas poderosas y eficientes para el desarrollo y despliegue de modelos de aprendizaje profundo, especialmente en aplicaciones que requieren procesamiento avanzado de imágenes y vídeo gracias a su integración con CUDA lo que permite hacer uso de la memoria GPU.

3. Especificaciones y restricciones de diseño

El objetivo principal del proyecto es desarrollar una aplicación dedicada a estimar el número de rodaballos presentes en tanques de acuicultura a partir de imágenes tomadas con distintos niveles de zoom haciendo uso de modelos de segmentación.

3.1 Especificaciones de diseño

E.1. Interfaz de usuario intuitiva: La aplicación cuenta con una interfaz de usuario intuitiva y fácil de usar desarrollada a partir del visor *Napari*, que permita a los usuarios cargar imágenes, visualizar los resultados y realizar ajustes si es necesario antes de iniciar el proceso de segmentación de imágenes.

E.2. Precisión del conteo: El modelo es capaz de estimar con precisión el número de rodaballos presentes en la imagen ofrecida.

E.3. Eficiencia: La aplicación es eficiente en términos de recursos computacionales, especialmente en cuanto a tiempo de procesamiento.

E.4. Robustez ante la variabilidad de tamaños y posiciones: El sistema es robusto ante la variabilidad en los tamaños y posiciones de los rodaballos en la imagen. Es capaz de segmentar los objetos presentes y contar los peces independientemente de su tamaño, orientación o posición en el tanque de cultivo.

E.5. Capacidad de interpolación: El sistema es capaz de estimar el número de rodaballos presentes en imágenes que no ha visto antes.

E.6. Lenguaje de programación que implementará el modelo: El sistema emplea Python 3.8 Se emplearán las siguientes librerías principales:

a) ***QtPy*:** Una biblioteca que proporciona una capa de compatibilidad para seleccionar entre PyQt5/PySide2 de manera transparente en aplicaciones Qt en Python.
b) ***PyQt5*:** Proporciona funcionalidades extensas para crear aplicaciones de escritorio multiplataforma.
c) ***OpenCV*:** Una biblioteca de código abierto que incluye una amplia gama de herramientas para procesamiento de imágenes y visión por computadora.
d) ***Matplotlib***: Una biblioteca para la creación de gráficos en 2D en Python
e) scikit-image: Una colección de algoritmos para el procesamiento de imágenes en Python.
f) ***Napari:*** Una herramienta interactiva de visualización de imágenes en Python.
g) timm: Una biblioteca de modelos de redes neuronales preentrenados y arquitecturas de redes neuronales en PyTorch.
h) ***Torch, torchvision*:** PyTorch es un marco de aprendizaje profundo que facilita la implementación y experimentación con redes neuronales. torchvision proporciona conjuntos de datos, transformaciones y modelos específicos de visión por computadora para PyTorch.

3.2 Restricciones de diseño

R.1. La base de datos de imágenes etiquetadas manualmente y que se usaran para validar la solución del diseño estará limitada a 30 imágenes con zoom y 30 imágenes sin zoom. Las imágenes serán las proporcionadas por el Grupo de Aplicaciones Multimedia y Acústica (GAMMA).

R.2. El etiquetado de las imágenes no ha sido realizado por profesionales, por lo tanto, pueden contener errores.

R.3. El PC utilizado para el desarrollo del diseño, ya que sus recursos hardware y especialmente su *GPU (Graphics Processing Unit)*, limitarán el rendimiento en términos de velocidad de procesamiento:

- 13th Intel(R) Core(TM) i7-13700F 2.10 GHz
- 16,0 GB RAM
- NVIDIA GeForce RTX 306
- Sistema Operativo Windows 11

4. Descripción de la solución propuesta

Este proyecto introduce una aplicación capaz de estimar el número de rodaballos en imágenes con diferentes niveles de zoom. Los resultados se muestran en una interfaz intuitiva que facilita la visualización y el análisis de cada etapa del procesamiento.

La solución propuesta se divide en diferentes fases:

1. **Etiquetado de imágenes:** En primer lugar, es necesario disponer de un banco de imágenes en el que se conozca con precisión el número de rodaballos presentes en ellas para posteriormente comprobar la fiabilidad de nuestro sistema.

2. **Selección del modelo:** Se detallarán las causas que han llevado a la elección final del modelo de segmentación y se hará un estudio de este.

3. **Etapa de preprocesamiento:** Se tratará la imagen de entrada para mejorar el proceso de segmentación en el modelo SAM.

4. **Implementación del modelo:** A partir de las imágenes de entrada, se analizarán las características del modelo de segmentación seleccionado y se procederá a su implementación en Python. Después de seleccionar el modelo, se ajustarán los parámetros de segmentación para optimizar los resultados según el nivel de zoom de las imágenes de entrada.

5. **Etapa de postprocesamiento:** Se desarrollarán funciones para filtrar y refinar los resultados generados por el modelo, con el fin de estimar con precisión el número total de rodaballos en la imagen.

6. **Implementación de la interfaz:** Se implementará una interfaz intuitiva que permita al usuario ajustar manualmente los parámetros, visualizar los procesos de segmentación y exportar los resultados.

4.1 Etiquetado de imágenes

Las imágenes para las cuales se ha desarrollado la solución de cuantificar el número de rodaballos provienen de la empresa pesquera Nueva Pescanova con sede en Vigo.

En estas imágenes encontramos rodaballos en etapa larvaria, es decir, rodaballos de hasta 26 días de vida. Las larvas de rodaballo recién eclosionadas miden unos 3 milímetros de largo y pesan de 0,1 a 0,2 miligramos [27]. En la Figura 12 podemos observar un ejemplo de una larva de rodaballo de 4 días.

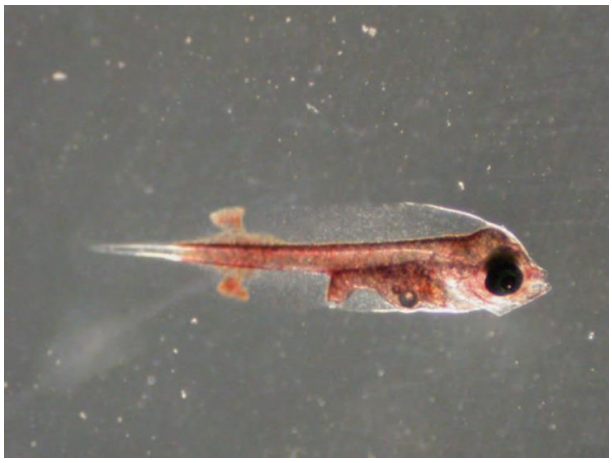

Figura 12 – Larva de rodaballo de 4 días [27]

En el día 15, la larva alcanza aproximadamente 7 milímetros de longitud y empieza su metamorfosis, durante la cual adquiere una forma plana y el ojo derecho se desplaza hacia el lado izquierdo [27]. En las Figura 13 y 14 se aprecian las diferencias entre una larva de 10 días y otra de 16 días.

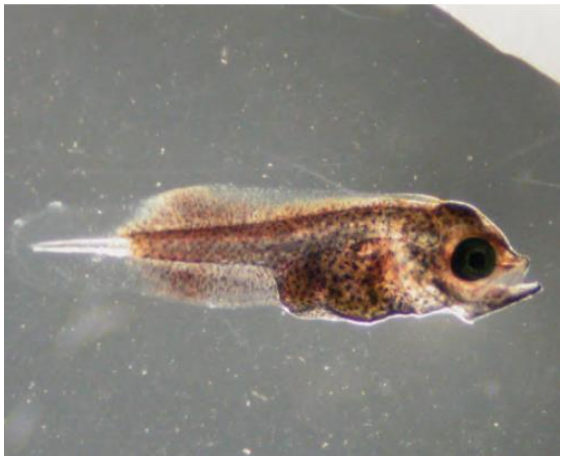

Figura 13 – Larva de rodaballo de 10 días [27]

Figura 14 – Larva de rodaballo de 16 días [27]

Hacia el final de la metamorfosis que tiene lugar alrededor del día 40 y 50 de vida las postlarvas miden de 20 a 30 milímetros y pesan de 0,1 a 0,15 gramos [27]. Aunque, como se ha dicho anteriormente, nuestras imágenes de entrada sólo contienen rodaballos en etapa larvaria de hasta 26 días, la Figura 15 presenta un rodaballo de 40 días que nos ofrece una imagen de un rodaballo hacia el final de su etapa larvaria.

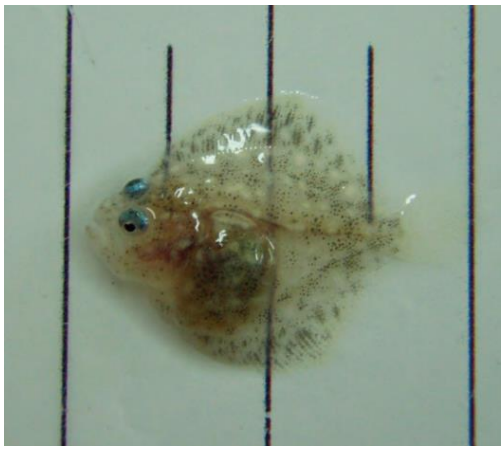

Figura 15 – Larva de rodaballo de 40 días [27]

Las imágenes que se han utilizado provienen de diferentes tanques de cultivo, de una de las plantas de acuicultura, en los cuales se han colocado cámaras cenitales que graban 24 horas al día. En función del nivel de zoom aplicado sobre los tanques, se ha diferenciado el proceso de etiquetado en dos partes: etiquetado de imágenes con zoom y etiquetado de imágenes sin zoom.

4.1.1 Etiquetado de imágenes con zoom

El etiquetado de las imágenes con zoom se ha llevado a cabo siguiendo la aplicación *Turbot Classify Application* desarrollada por Silvia González Barquín en Matlab [28].

La aplicación ofrece una interfaz donde a partir de un video de entrada seleccionamos el fotograma a etiquetar. A partir de este fotograma, se encarga de segmentar todos los objetos que interpreta y a continuación, se muestran los objetos segmentados de mayor a menor. El usuario debe seleccionar si se trata de un rodaballo o un grupo de rodaballos y el grado de certeza; en el caso de que sea un grupo, el usuario debe de indicar el centro de cada uno de ellos.

En la Figura 16 podemos observar un ejemplo de imágenes de los tanques con zoom.

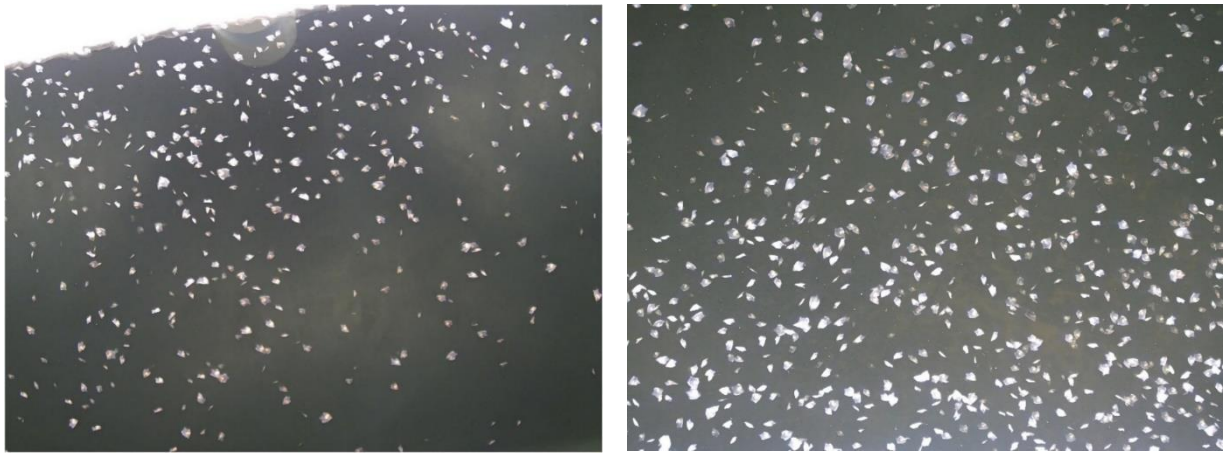

Figura 16 – Imágenes del tanque con zoom (1) y (2)

En la Figura 17 se muestra la interfaz de la aplicación:

Figura 17 – Interfaz *Turbot Classify Application.*

Una vez terminada la selección de todos los objetos se genera un archivo *.mat*, que podemos visualizar en la Figura 18. Este fichero contiene diferentes variables de interés como son: matriz que almacena las posiciones *x,y* de los centros de los rodaballos, una matriz de las mismas dimensiones que las de la imagen en la cual todos los pixeles de un objeto tienen el mismo valor o una matriz que contiene las características morfológicas de los objetos.

Import	Name	Size	Bytes	Class
☑	BS	864x1	1302944	cell
☑	LS	1920x2560	39321600	double
☑	countObjectS	1x1	8	double
☑	iFrameS	1x1	8	double
☑	mrPropsS	11x864	76032	double
☑	nS	1x1	8	double
☑	turbotCentroidS	812x3	19488	double

Figura 18 – Fichero de salida .mat

En nuestro caso además del fichero de salida, guardamos la imagen con los centros aplicados (ver Figura 19) y el número de objetos etiquetados que contiene (este valor lo obtenemos de la variable *turbotCentroidS*).

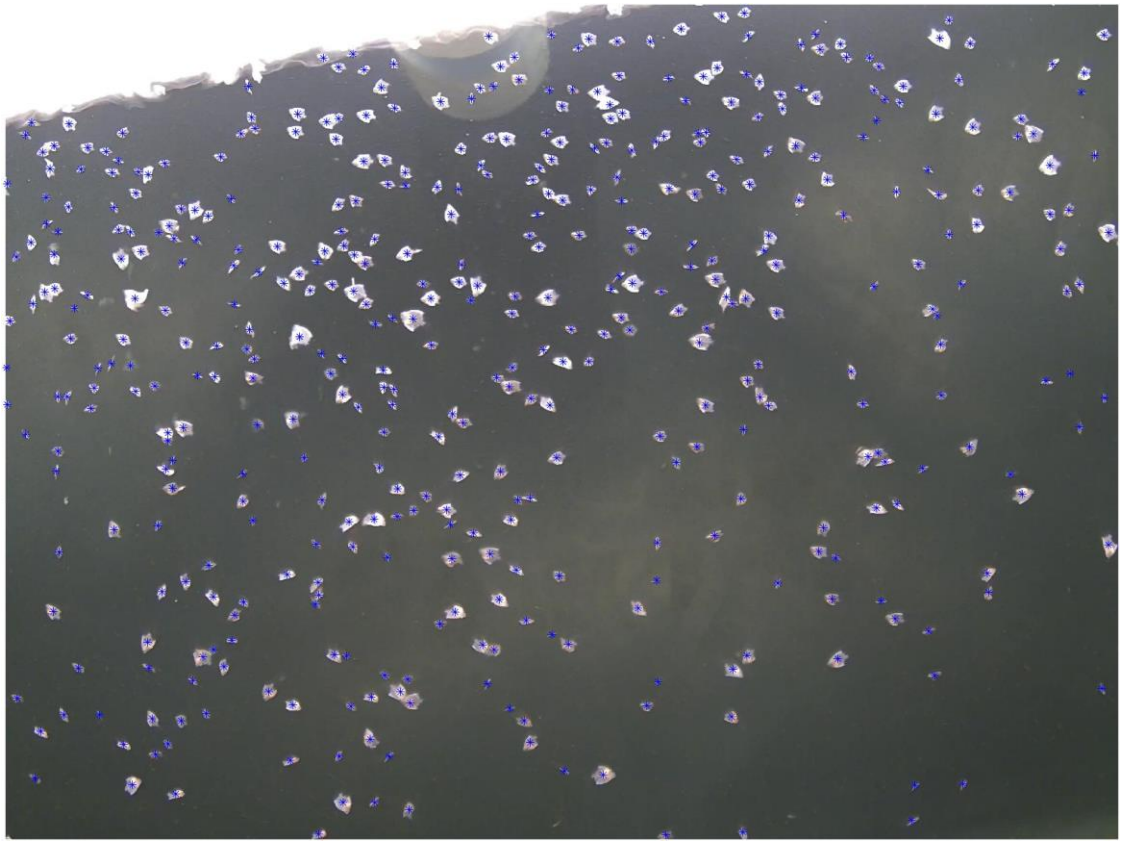

Figura 19 – Imagen con zoom (1) etiquetada

Se han etiquetado 5 nuevas imágenes con zoom, las cuales se han añadido a un conjunto de imágenes previamente etiquetadas. En total, se ha creado un banco de prueba compuesto por 30 imágenes con zoom etiquetadas, en las cuales se conoce el número exacto de rodaballos presentes. Estas imágenes se utilizarán para validar nuestro sistema.

4.1.2 Etiquetado de imágenes sin zoom

El etiquetado de las imágenes sin zoom se ha llevado a cabo con una aplicación desarrollada en Python por el CITSEM llamada *Marcador de Puntos* que permite cargar un vídeo, seleccionar un fotograma específico y marcar puntos de interés en el mismo [29].

En las Figuras 20 y 21 podemos observar un ejemplo de imágenes de los tanques sin zoom.

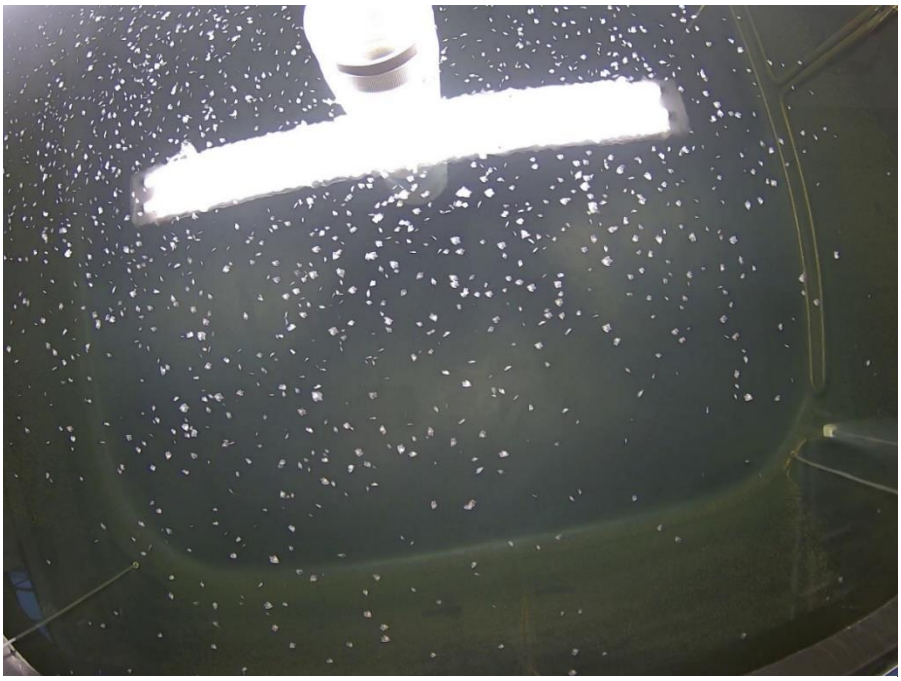

Figura 20 – Imagen del tanque sin zoom (1)

Figura 21 – Imagen del tanque sin zoom (2)

En las Figura 22 y 23 se muestra la interfaz de la aplicación, la cual permite hacer zoom en cada cuadrante para una mayor facilidad a la hora de establecer el centro en cada uno de los rodaballos.

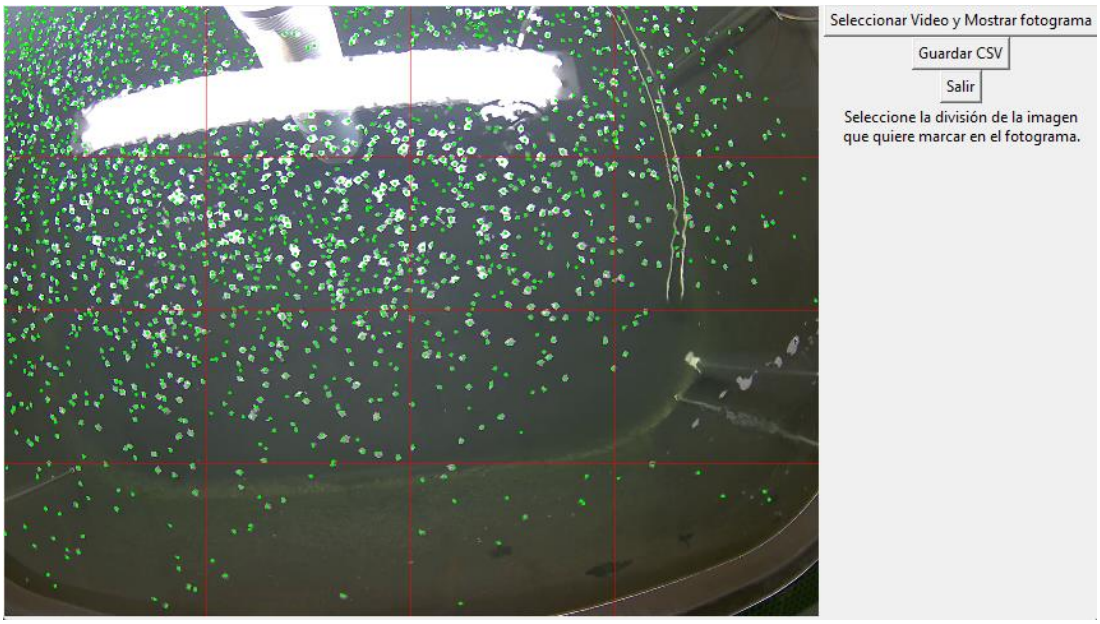

Figura 22 – Interfaz *Marcador de puntos*

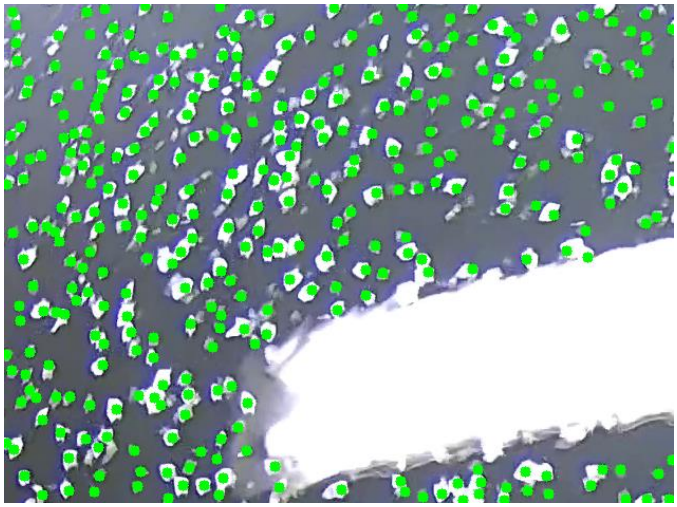

Figura 23 – Cuadrante *Marcador de puntos*

Una vez terminado el etiquetado manual en todos los objetos que se considere, se almacenará un archivo CSV que contiene las posiciones *x,y* de los centros de los rodaballos establecidos.

Se han etiquetado 5 nuevas imágenes sin zoom que se han añadido a un conjunto de imágenes previamente etiquetadas. En total, tenemos un banco de 15 imágenes sin zoom etiquetadas de las que conocemos el número presente de rodaballos en ellas y que usaremos para validar nuestro sistema.

4.2 Selección del modelo

El objetivo del proyecto es la estimación del número de rodaballos presentes en los tanques considerando diferentes niveles de zoom. Por lo tanto, es crucial disponer de un modelo capaz de identificar y segmentar los diferentes objetos en las imágenes de entrada.

Inicialmente, se propuso utilizar un modelo basado en el entrenamiento de un conjunto de datos específico, en este caso de rodaballos, como *U-Net*. *U-Net* es una arquitectura de red neuronal convolucional diseñada para realizar segmentación semántica en imágenes, donde su objetivo es asignar una etiqueta a cada píxel de la imagen basándose en el contenido visual. Fue desarrollada originalmente para segmentar imágenes biomédicas [30]. Este modelo debía abordar ciertos desafíos:

- **Cantidad limitada de datos etiquetados:** Es fundamental recolectar una cantidad significativa de imágenes que contengan los objetos de interés y generar máscaras de segmentación para cada imagen, para posteriormente, proceder con el entrenamiento del modelo. En estas máscaras, cada píxel debe estar etiquetado para indicar si pertenece al objeto de interés o al fondo. Se dispone de 30 imágenes etiquetadas de tanques con zoom y 30 imágenes etiquetadas sin zoom. Esto puede limitar la capacidad del modelo para generalizar correctamente debido a la falta de variabilidad en los datos.

- **Tamaño de los objetos de interés reducido:** En las imágenes sin zoom, los rodaballos se aprecian muy pequeños, y en las imágenes con zoom, pueden aparecer larvas de edad temprana en el fondo, lo cual complica su identificación precisa.

- **Superposición y agrupación de objetos:** Los rodaballos a menudo aparecen agrupados o superpuestos, lo que dificulta que la red neuronal distinga correctamente entre objetos individuales y grupos. En la Figura 24 podemos observar un ejemplo de estas agrupaciones.

Figura 24 – Superposición de rodaballos

- **Orientación variada de los objetos:** Los rodaballos aparecen en todo tipo de orientaciones, como se puede observar en la Figura 25, dificultando su identificación.

Figura 25 - Rodaballos con distinto tipo de orientación

Dada la diversidad en las características de los rodaballos en las imágenes, como tamaño, superposición, densidad y orientación, junto con las diferencias de iluminación entre las imágenes, se concluyó en que el rendimiento del modelo podría no alcanzar los estándares deseados.

4.2.1 Segment Anything Model (SAM)

Segment Anything Model (SAM) es el modelo elegido para dar solución al desarrollo del proyecto. SAM es un modelo de inteligencia artificial desarrollado por *Meta AI* para abordar el desafío de la segmentación de imágenes. Sus desarrolladores se basaron en trasladar la idea de los modelos de lenguaje natural (PNL) a la segmentación y por ello, está diseñado para realizar segmentación semántica, es decir, identificar y etiquetar cada píxel de una imagen según el objeto o la región a la que pertenece. Por lo general, los modelos de segmentación requieren ser entrenados desde el inicio con un *dataset* etiquetado que contenga los objetos relevantes, para lo cual es necesario tener un *dataset* amplio que aporte la máxima información del objeto. SAM tiene como objetivo ofrecer un modelo base capaz de segmentar cualquier objeto basado en aprendizaje *zero-shot* (apartado 2.1), permitiendo su aplicación en distintos dominios sin la necesidad de entrenamiento adicional.

A continuación, se detallarán las características más destacadas de SAM [31]:

1) SAM fue concebido como un modelo de segmentación basado en *prompts*, permitiéndole generar máscaras de segmentación válidas a partir de cualquier tipo de indicación, ya sean espaciales como puntos o cajas delimitadoras o textuales que identifiquen el objeto a segmentar (ver Figura 26).

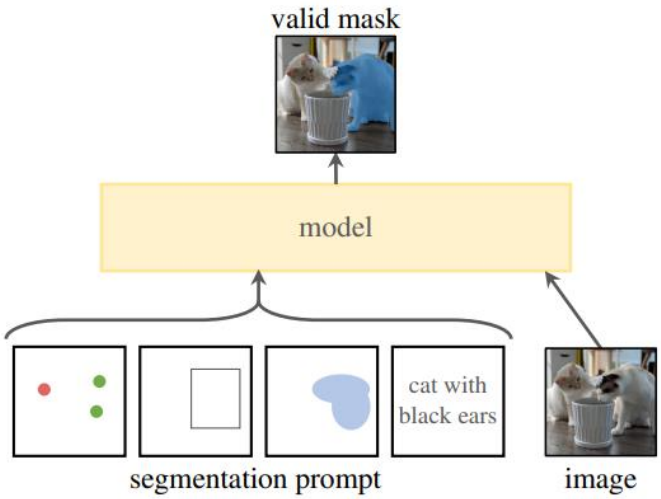

Figura 26 – *Prompts* de segmentación [31]

2) *Segment Anything Model* (SAM) es una arquitectura de segmentación eficiente diseñado para aplicaciones interactivas en visión por computadora. Esta arquitectura tiene tres componentes fundamentales representados en la Figura 27:

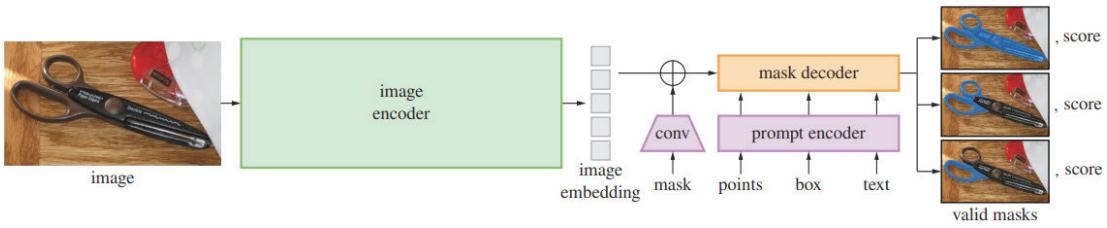

Figura 27 – Arquitectura *SAM* [31]

- ***Image encoder*:** Utiliza *Mask Auto-Encoder* (MAE) *pre-trained Vision Transformer*, el cual es una adaptación del ViT, que como se explicó anteriormente, es una arquitectura basada en *Transformers* adaptada para procesar imágenes mediante el uso de atención sobre parches de píxeles en lugar de convoluciones (apartado 2.3). Esta parte del modelo codifica la información visual de la imagen de entrada y entrega a su salida *image embedding*, la cual es una representación de la imagen de entrada reducida 16 veces en forma de vectores que capturan información relevante sobre la estructura, el contenido y los detalles de la imagen que es más fácil de procesar. Esta información es crucial para que el modelo realice tareas específicas como clasificación, detección de objetos o segmentación.

- ***Prompt encoder*:** Se diseña para manejar diferentes tipos de entradas que ofrecen al modelo información sobre lo que se debe segmentar en una imagen representado por *prompts embedding*. Se especifican 2 tipos de prompts:

    - ***Sparse prompts*:** Se refiere a los puntos, cajas o texto que el usuario puede establecer como entrada en la imagen para indicar qué desea segmentar (ver Figura 28). Los puntos y las cajas se representan usando codificaciones posicionales de manera que el modelo entienda su posición relativa en relación con otros puntos o elementos en la imagen. El texto libre se maneja utilizando un codificador de texto preentrenado como el de CLIP, un modelo desarrollado por *OpenAI* que combina aprendizaje de lenguaje y visión de manera contrastiva que permite entender y generar descripciones sobre imágenes y texto [32], adaptado para integrarse con el modelo de segmentación de SAM.

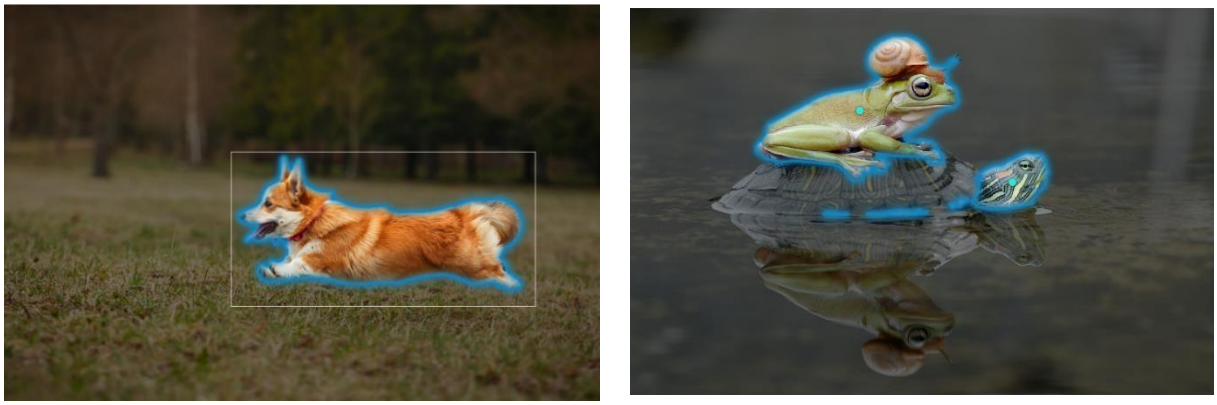

Figura 28 – *Prompts* de caja y puntos de segmentación [33]

    - ***Dense prompts*:** Se refieren a entradas más detalladas que especifican directamente la forma completa de la región u objeto que se desea segmentar en la imagen. Estas máscaras son matrices bidimensionales que cubren toda el área del objeto de interés, donde cada píxel en la máscara indica si pertenece al objeto (1) o no (0). Para manejar estas máscaras, el modelo utiliza convoluciones para integrar la información eficientemente, sumándola con la información de la imagen original. Esto permite al modelo comprender y generar segmentaciones detalladas según las máscaras proporcionadas como entrada.

- ***Mask decoder*:** Esta etapa utiliza mecanismos de atención para procesar y combinar información relevante para generar máscaras detalladas a partir de la información de la imagen y los *prompts*. Las características principales del decodificador son:

  - **Mapeo:** Implica la integración de tres tipos de entradas principales: i*mage embedding* (que contiene información visual detallada), *prompts embedding* (que indican qué objetos o áreas segmentar), y un token de salida (que guía la generación de la máscara)

  - **Arquitectura:** Adapta la arquitectura de un bloque decodificador *Transformer* agregando una capa adicional especializada en predecir las máscaras de segmentación. Usa mecanismos de atención para enfocarse automáticamente en partes específicas de la imagen y en la información proporcionada por los prompts. La atención cruzada permite al modelo combinar esta información de manera efectiva para generar máscaras detalladas y precisas que identifican las áreas que se deben segmentar en la imagen.

  - **Predicción de máscara:** Después de procesar la información mediante dos bloques decodificadores, SAM incrementa la resolución de *image embedding*. Luego, utiliza un perceptrón multicapa (MLP), una red neuronal encargada de transformar la información del token de salida (que es una representación numérica) en un formato adecuado para realizar una clasificación lineal. Por último, se emplea este clasificador para calcular la probabilidad de que cada píxel de la imagen pertenezca a la máscara de segmentación.

Figura 29 – Generación de máscaras para *prompts* ambiguos [31]

De esta manera el modelo proporciona a su salida las máscaras de segmentación basadas en las probabilidades calculadas. El modelo puede producir varias máscaras válidas para un mismo *prompt* provocando ambigüedad. Por esta razón, se estableció un límite de 3 máscaras posibles para un *prompt* dado, donde cada máscara tiene su propia puntuación de confianza (ver Figura 29).

3) *Meta AI* ha desarrollado un conjunto de datos denominado *SA-1B*, que consta de 11 millones de imágenes de alta resolución y más de mil millones de máscaras de segmentación, han sido validadas por evaluaciones humanas y experimentos, demostrando alta calidad y diversidad.

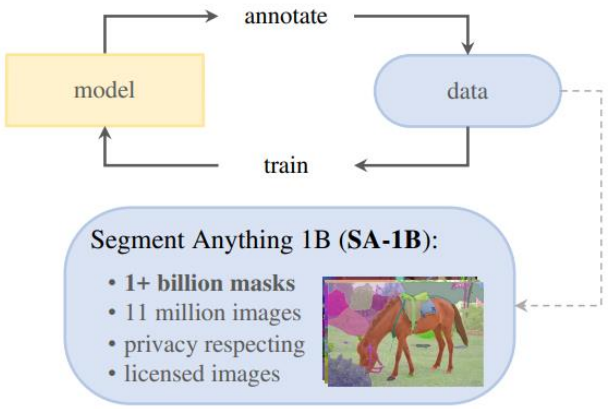

Figura 30 – Esquema de preentrenamiento de *SAM* [31]

Este conjunto de datos con el que ha sido preentrenado (ver Figura 30) proporciona a SAM una sólida base para aprender a identificar y segmentar objetos con gran precisión, dando lugar a máscaras de referencia, incluso en casos donde no haya visto esos objetos previamente [34]. En la Figura 31 se muestra un ejemplo de una de las imágenes pertenecientes a *SA-1B*.

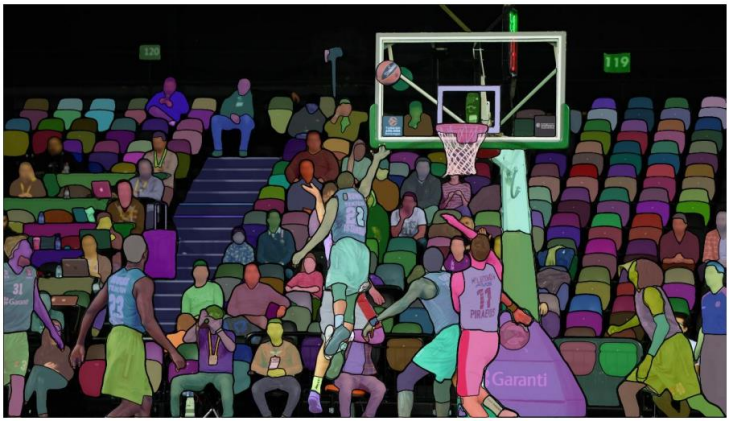

Figura 31 – Imagen *SA-1B* [31]

4) SAM nació como un modelo fundacional, es por ello por lo que se basa en la necesidad de segmentar imágenes con precisión sin una formación específica previa, una tarea que tradicionalmente requiere modelos personalizados. Es por ello por lo que se evaluó la capacidad de aprendizaje *zero-shot* de SAM, que como se explicó previamente se refiere a la habilidad del modelo de ejecutar tareas para las cuales no fue entrenado, obteniendo un excelente rendimiento. Esto permite la adaptación rápida y versátil de SAM en múltiples aplicaciones de ingeniería.

En resumen, SAM desarrollado por *Meta AI*, es un modelo preentrenado de segmentación semántica. Fue preentrenado con el conjunto de datos *SA-1B*, que contiene 11 millones de imágenes y más de mil millones de máscaras de segmentación validadas, permitiendo a SAM segmentar objetos con precisión sin necesidad de entrenamiento adicional, facilitando su aplicación en diversos dominios. Es por esto, por lo que se eligió SAM como solución ante la problemática del número limitado de imágenes etiquetadas de tanques de rodaballos.

4.2.2 Mobile SAM

MobileSAM es una versión optimizada y ligera de *Segment Anything Model* (SAM), diseñada por *Kyung Hee University* específicamente para aplicaciones con recursos limitados, como teléfonos móviles. Este modelo fue desarrollado para abordar la necesidad de ejecutar tareas de segmentación de manera eficiente y rápida en entornos con capacidad computacional reducida [35].

Como se ha explicado previamente, SAM consta de dos partes diferenciadas en lo que a su arquitectura se refiere: el codificador basado en ViT que obtiene información detallada de la imagen y el decodificador que genera máscaras de segmentación basadas en las indicaciones proporcionadas (ver Figura 32).

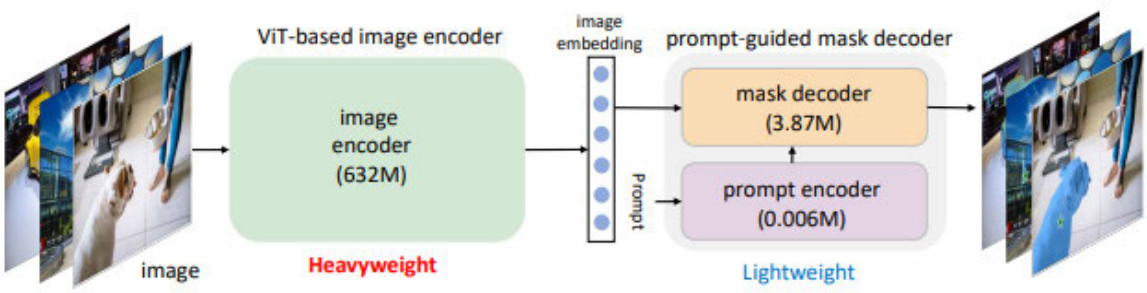

Figura 32 – Arquitectura de *SAM* resumida [35]

La computación pesada se encuentra en la etapa de codificación, en ViT encontramos 3 modelos principales diferenciados principalmente por su tamaño y capacidad [36]:

- ***ViT-Huge (Vit-H):*** Es el modelo más grande con alrededor de 632 millones de parámetros y ofrece el mejor rendimiento debido a su mayor capacidad para capturar patrones complejos en los datos, aunque es el que requiere mayores recursos computacionales.
- ***ViT-Large (Vit-L):*** Es el modelo intermedio con alrededor de 307 millones de parámetros y que mantiene un equilibrio entre rendimiento y eficiencia computacional.
- ***ViT-Base (ViT-B):*** Es el modelo más pequeño con alrededor de 86 millones de parámetros que sigue ofreciendo un buen rendimiento, aunque no tan alto como los dos anteriores y es más eficiente en términos de recursos computacionales.

También encontramos otros modelos basados en ViT:

- ***Class-Attention in Image Transformers (CaiT):*** Este modelo presta atención a diferentes partes de la imagen en función de las clases de objetos que está tratando de identificar. Está optimizado para tareas de clasificación de imágenes y puede manejar múltiples clases de objetos en una imagen con eficiencia.
- ***TinyViT (ViT-T):*** Es una versión muy ligera de ViT con alrededor de 8 millones de parámetros lo que lo hace adecuado para implementaciones en entornos con restricciones de hardware.

El objetivo principal es reemplazar el codificador pesado por uno más liviano como Vit-T, asegurando de que, a pesar de haber reducido el tamaño, *MobileSAM* mantenga la capacidad de generar *images embedding* precisas para las tareas de segmentación. Para lograr esto, se ejecutó una *destilación de conocimiento*, que es un proceso de compresión en el cual se transfiere el conocimiento aprendido por un modelo más grande y complejo (*ViT-H*) a un modelo más pequeño y ligero (*ViT-T*) [37]. Con ello, no solo se espera reducir el tamaño del modelo, sino que también puede mejorar la eficiencia computacional al optimizar el uso de recursos durante la inferencia. En la inferencia, el modelo entrenado aplica el conocimiento adquirido previamente para hacer predicciones o generar resultados basados en nuevos datos que no fueron utilizados durante su entrenamiento [38].

Figura 33 – Comparación de segmentación de los modelos [35]

Como resultado se obtuvo *MobileSAM*, un modelo 60 veces más pequeño que SAM, pero con un rendimiento comparable (ver Figura 33), además se realizó una comparación con otro modelo similar llamado *FastSAM* donde como se observa en la Tabla 1, *MobileSAM* es 4 veces más rápido y 7 veces más pequeño.

	OriginalSAM	FastSAM	MobileSAM
Size	611 M	68 M	9.66 M
Speed	452 ms	64 ms	12 ms

Tabla 1 – Comparativa *MobileSAM* y *FastSAM* [35]

Dado que *MobileSAM* mantiene todo el flujo de trabajo del SAM original y solo reemplaza el codificador de imágenes, puede integrarse fácilmente en proyectos existentes basados en SAM de una forma muy sencilla.

Debido a las características mencionadas, se ha optado por implementar *MobileSAM* debido a su capacidad para reducir significativamente los tiempos de procesamiento y su menor demanda de recursos CPU/GPU. Esto no solo mejora la eficiencia del modelo, sino que también amplía la compatibilidad de la solución propuesta con más sistemas.

4.3 Etapa de preprocesamiento

Durante esta etapa, se pretende tratar la imagen de entrada del modelo para garantizar que el proceso de segmentación se realice de manera óptima. En este contexto, el uso de imágenes en escala de grises en la segmentación de imágenes tiene varias ventajas que pueden ser útiles dependiendo del tipo de objetos que se desean segmentar [39]:

- El uso de imágenes en escala de grises elimina las variaciones de color, simplificando la tarea del modelo de segmentación. Esto es especialmente útil si el color no aporta información relevante para identificar los objetos de interés. En nuestro caso, los rodaballos tienen un color uniforme, como se observa en las imágenes anteriores, por lo que las variaciones de color no son necesarias para la segmentación.

- Las imágenes en escala de grises permiten que el modelo se enfoque en las diferencias de intensidad de píxeles, lo cual suele ser suficiente para distinguir entre distintas regiones de una imagen. Conociendo intensidad como el nivel de brillo de los píxeles en la imagen. En una imagen en escala de grises, la intensidad de un píxel se mide en una escala que va del negro (valor de intensidad bajo, generalmente 0) al blanco (valor de intensidad alto, generalmente 255). Esto es especialmente importante en nuestro caso, ya que la iluminación del tanque provoca reflejos en los rodaballos, haciendo que estos objetos tengan una alta intensidad y por lo tanto poder diferenciarlos de otros posibles objetos segmentados.

De este proceso se encarga la función *convertRGB* (ver Anexo A.1.1*)*. Esta función realiza las siguientes tareas:

a) Convierte la imagen de entrada RGB en escala de grises
b) Obtiene las dimensiones de la imagen
c) Genera una imagen RGB vacía con las mismas dimensiones que la original
d) Asigna el valor del canal de la escala de grises a cada canal de la imagen RGB
e) Devuelve una imagen RGB donde cada canal tiene el mismo valor que la versión en escala de grises de la imagen original (ver Figura 34).

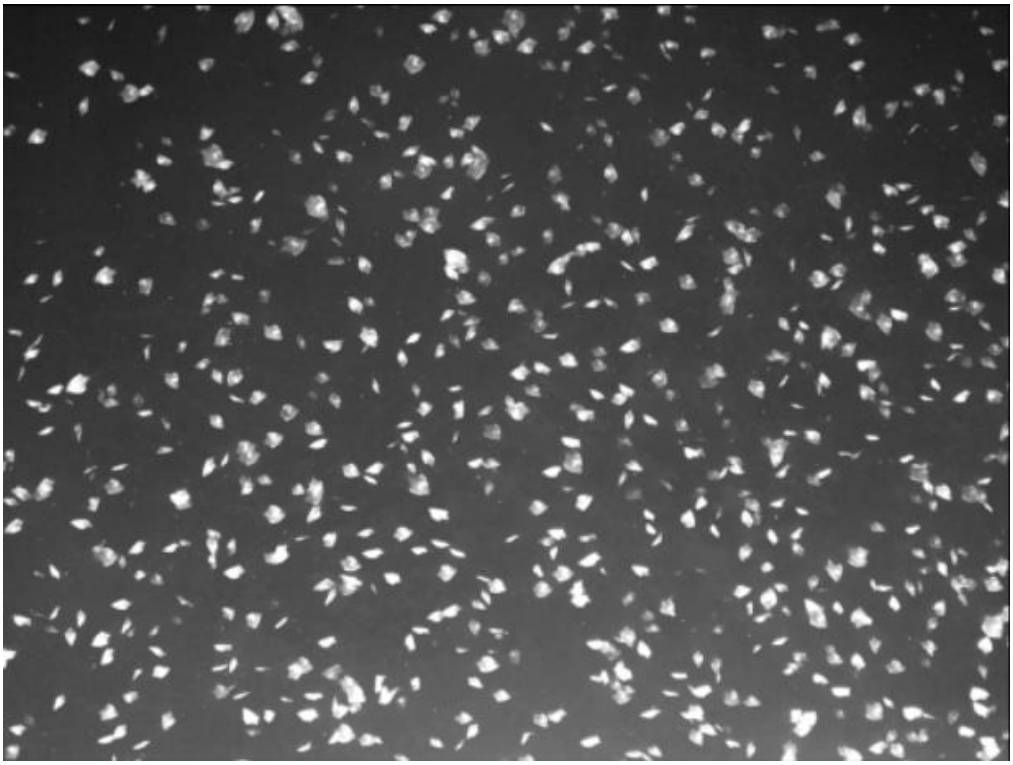

Figura 34 – Imagen de salida de la función *convertRGB*

Este proceso se realiza principalmente por razones de compatibilidad con modelos y métodos que esperan una imagen en formato RGB. Algunos modelos esperan entradas con tres canales y pueden no funcionar correctamente si se les proporciona una imagen con solo un canal de intensidad (escala de grises), además de que muchas bibliotecas y funciones de procesamiento de imágenes están optimizadas para trabajar con imágenes RGB. De esta forma, se mejora la compatibilidad de la aplicación ante integraciones futuras.

A partir de aquí, se entenderá que todas las imágenes de entrada utilizadas para la implementación del modelo han pasado por este proceso.

4.4 Implementación del modelo en Python

La integración del modelo en *Python* es sencilla, ya que los repositorios oficiales de SAM [40] y *MobileSAM* [41] proporcionan información detallada sobre el proceso de instalación, configuración del modelo y ejemplos de uso. Esto facilita a los desarrolladores la implementación rápida y eficiente de estos modelos en sus proyectos.

4.4.1 Instalación y configuración del modelo

En cuanto a la instalación tal y como indica el repositorio oficial de *MobileSAM* primero hay que descargarse el archivo *mobileSAM.pt* que contiene el modelo preentrenado *MobileSAM* que como vimos anteriormente está formado por *TinyViT* como codificador de imágenes.

Posteriormente, dentro de un entorno virtual Python >= 3.8, se ejecutará el comando "*pip install git+https://github.com/ChaoningZhang/MobileSAM.git*" que instalará las librerías necesarias, aunque es necesario instalar dependencias adicionales requeridas ya que es imprescindible tener instaladas las librerías PyTorch >= 1.7 y TorchVision >= 0.8. Tal como se mencionó en el apartado de 2.8, instalaremos estas librerías con soporte CUDA. Esto permite que los cálculos y procesos se realicen en la GPU en lugar de la CPU, lo que resulta en una aceleración significativa del rendimiento. En caso de que no se detecte la instalación de las librerías con soporte CUDA, se utilizará la CPU para el procesamiento, lo cual resultará en tiempos de ejecución más elevados. Finalizados estos pasos estaremos en disposición de configurar el modelo.

Una vez que registramos el modelo a partir del archivo *mobileSAM.pt* (ver Anexo A.1.3 línea 11), se nos ofrece la opción de elegir el modo de segmentación según la instancia de clase seleccionada:

a) ***SamAutomaticMaskGenerator:*** Esta clase permite generar máscaras automáticamente utilizando el modelo SAM. Funciona tomando puntos de entrada individuales muestreados en una cuadrícula sobre la imagen. Desde cada uno de estos puntos, SAM puede predecir múltiples máscaras potenciales. Luego, las máscaras se filtran para seleccionar las de mejor calidad. Esto facilita la integración del modelo en aplicaciones que requieren segmentación automática de objetos en imágenes. Además, existen parámetros adicionales para mejorar la calidad y la cantidad de las máscaras generadas [41].

b) ***SamPredictor:*** Esta clase actúa como una interfaz simplificada para interactuar con el modelo *MobileSAM*, específicamente diseñada para la predicción de máscaras de segmentación en imágenes. Su función principal es facilitar el proceso de configuración de imágenes y la generación de máscaras predictivas utilizando diferentes tipos de *prompts* vistos anteriormente como puntos o cajas de segmentación [41].

Dado que el objetivo del proyecto es estimar el número de objetos (rodaballos) en imágenes con diferentes niveles de zoom, se optó por utilizar el método de segmentación automática. Este método facilita la segmentación de todos los objetos en la imagen, permitiendo ajustar los parámetros de segmentación según los niveles de zoom específicos para lograr resultados óptimos.

4.4.2 Parámetros de segmentación

Al instanciar la clase *SamAutomaticMaskGenerator* ésta tiene como argumentos unos parámetros de segmentación establecidos por defecto:

- points_per_side: int | None = 32
- points_per_batch: int = 64
- pred_iou_thresh: float = 0.88
- stability_score_thresh: float = 0.95
- stability_score_offset: float = 1
- box_nms_thresh: float = 0.7
- crop_n_layers: int = 0
- crop_nms_thresh: float = 0.7
- crop_overlap_ratio: float = 512 / 1500
- crop_n_points_downscale_factor: int = 1
- point_grids: List[ndarray] | None = None
- min_mask_region_area: int = 0
- output_mode: str = "binary_mask"

Estos parámetros sirven para controlar diferentes aspectos del proceso de generación automática de máscaras utilizando el modelo SAM. A continuación, se detalla la definición de cada uno de ellos y la respuesta del modelo ante las variaciones de los valores de los parámetros más significativos llevando a la elección final de éstos.

Hay que recordar que los tiempos de segmentación que se mencionan son el resultado de la ejecución de la aplicación en un PC con los requisitos especificados en el apartado 3, además hay que tener en cuenta el tamaño de las imágenes utilizadas para la demostración ya que el peso de estas influye en la carga de la etapa de codificación vista anteriormente. La imagen con zoom representada en la Figura 35 tiene un peso de 343 KB mientras que la imagen sin zoom representada en la Figura 36 tiene un peso de 1.12 MB.

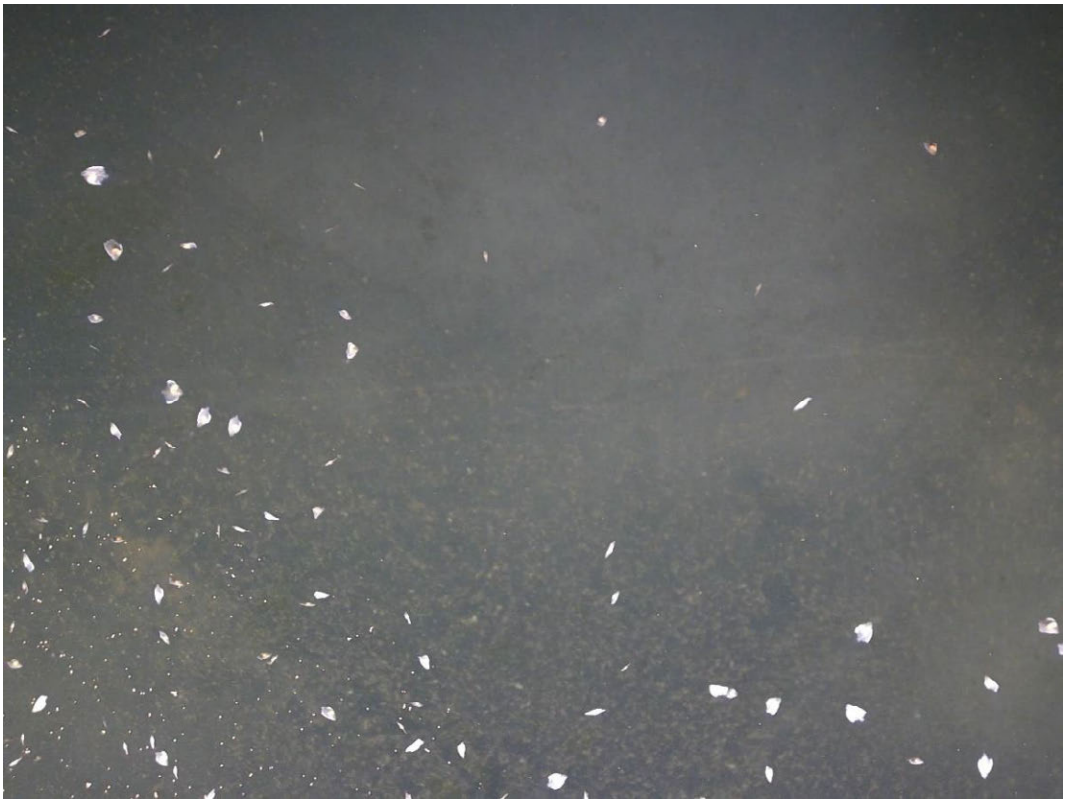

Figura 35 – Imagen del tanque con zoom (3)

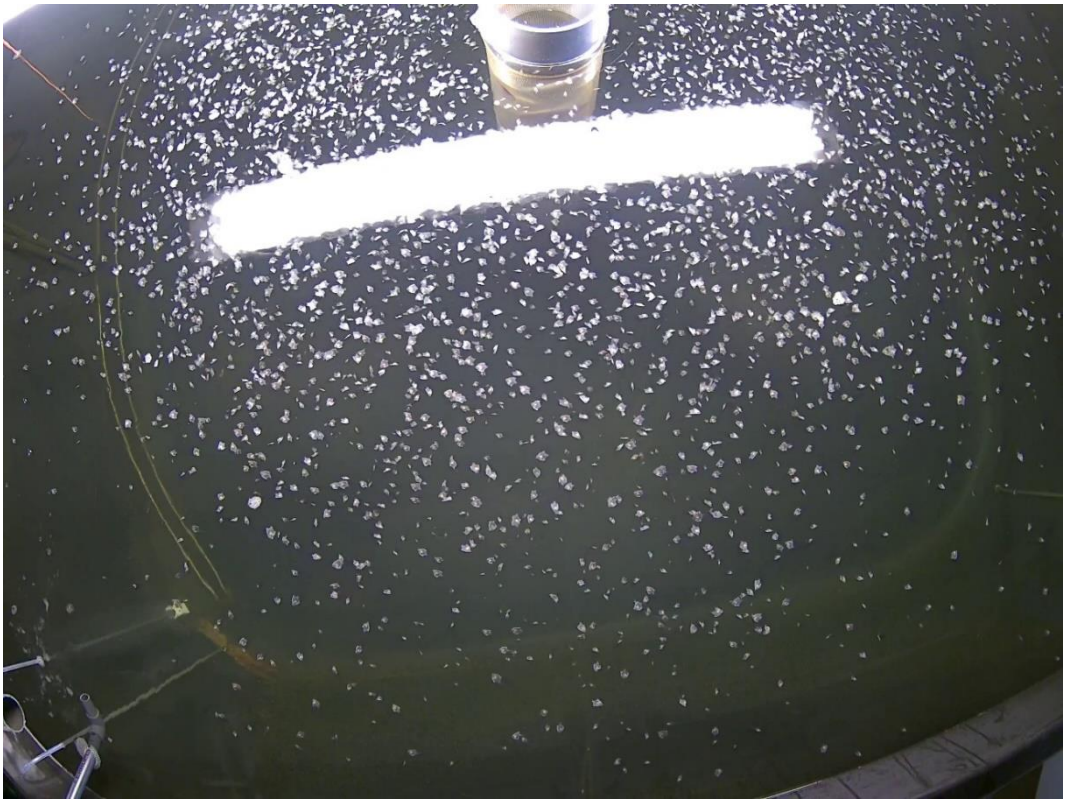

Figura 36 – Imagen de tanque sin zoom (3)

a) points_per_side

Este parámetro hace referencia a la cantidad de puntos de entrada individuales que se muestrean en una cuadrícula uniforme sobre la imagen para generar las máscaras de segmentación. Estos puntos de entrada son puntos de referencia sobre los cuales SAM predice posibles máscaras de segmentación. Si una imagen tiene un tamaño de 1024x1024 píxeles y se establece *points_per_side* en 32, los puntos de entrada se distribuirán uniformemente cada 32 píxeles a lo largo del ancho y el alto de la imagen, generando una cuadrícula de puntos de 32x32 (ver Figura 37).

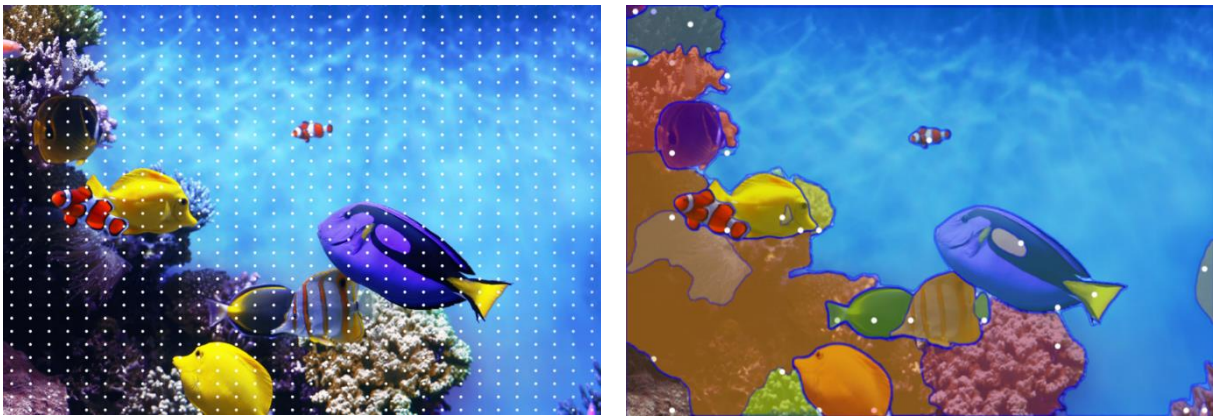

Figura 37 – *Points per side* de 32 [33]

Incrementar el valor de *points_per_side* produce una cuadrícula más densa, lo que puede mejorar la precisión de la segmentación al proporcionar más puntos de entrada para el modelo. Sin embargo, también aumenta el costo computacional y el tiempo de procesamiento. Se procede a detallar la salida del proceso de segmentación para las imágenes con distintos tipos de zoom ante diferentes valores de entrada para este parámetro:

Imágenes con zoom: En la Figuras 38 se puede observar la diferencia en la respuesta de segmentación ante una entrada de 32 puntos y otra de 64 puntos. Se ha obtenido una mejor respuesta de segmentación para la entrada de 64, ya que es capaz de localizar muchas más máscaras, y una diferencia de tiempo de segmentación de 3 minutos entre ellos.

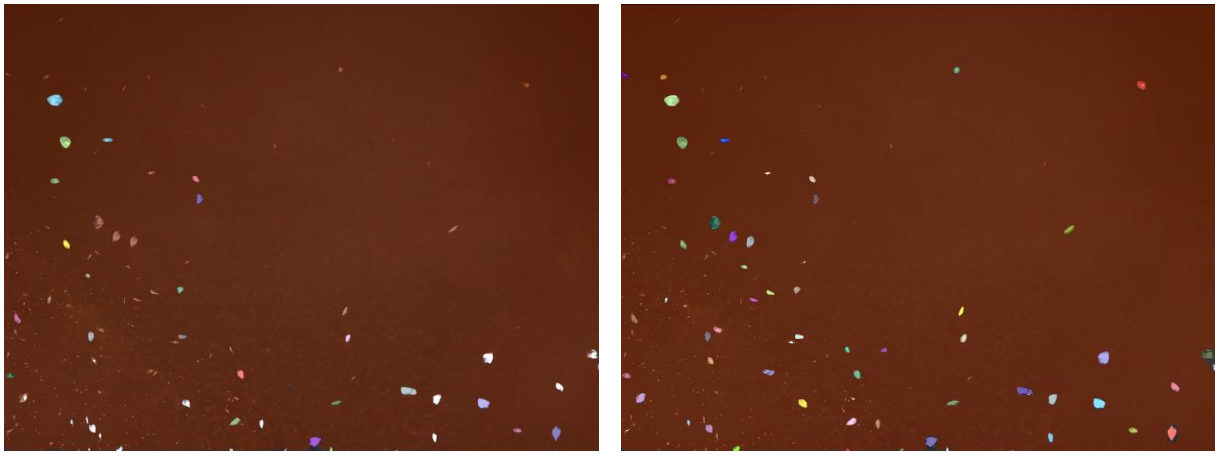

Figura 38 – Salidas de imagen con zoom (3) para 32 y 64 puntos

Imágenes sin zoom: En este caso, se observa que usar 64 puntos de entrada no es efectivo, ya que no logra identificar los objetos debido a su tamaño, además de que el tiempo de segmentación aumenta hasta los 7 minutos. Inicialmente, se realizaron pruebas con 128 puntos de entrada (ver Figura 39), lo que mejoró los resultados un poco; sin embargo, el tiempo de segmentación se elevó a 20 minutos. Por ello, se buscó una manera más eficiente de segmentar estas imágenes.

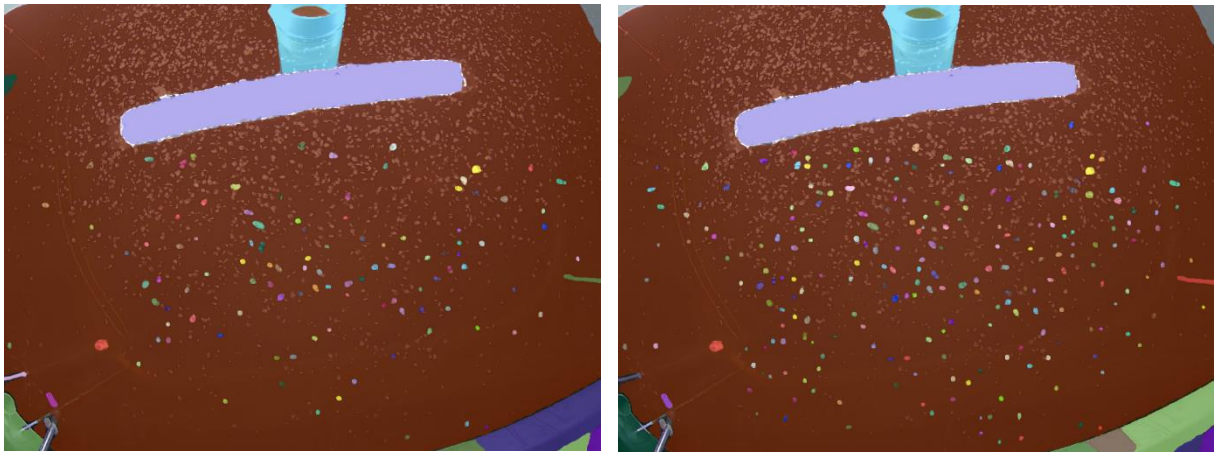

Figura 39 – Salida de imagen sin zoom (3) para 64 y 128 puntos

La solución fue aplicar segmentación en bloques: primeramente, se selecciona el número de cuadrantes en función de la densidad de objetos que se aprecia en la imagen, cada cuadrante se introduce como entrada a SAM para que ejecute el proceso de segmentación y finalmente se vuelven a unir los cuadrantes formando una imagen. En la salida del proceso se observaron 3 puntos a destacar:

- Estableciendo 64 puntos de entrada la respuesta del modelo era buena para cada cuadrante.
- Cada cuadrante tiene un peso inferior al de la imagen original lo que provoca que los tiempos de segmentación de cada uno sean bajos.
- Cuanto más se divida la imagen en cuadrantes, más profunda será la segmentación. Por lo tanto, es importante ajustar el número de cuadrantes según la densidad de rodaballos percibida.
- Algunos objetos se encuentran en los bordes de los cuadrantes y, al combinarlos en una sola imagen, pueden interpretarse como dos objetos separados en lugar de uno solo (ver Figura 41). Sin embargo, el error asociado a este problema es muy bajo en comparación con la cantidad total de objetos en imágenes de este tipo, que suelen superar los 1500.

A continuación, se analizarán los pasos que han facilitado el proceso de segmentación por bloques. Después de cada proceso de segmentación, SAM devuelve una lista de diccionarios donde cada diccionario almacena atributos de cada máscara, estos atributos contienen información detallada de cada máscara. Se usarán para obtener toda la información necesaria de la posición de cada máscara.

- ***segmentation*:** Es un array, donde cada valor representa la máscara binaria de la segmentación. Los píxeles que pertenecen al objeto de interés tienen un valor 1 y los que no, un valor 0.
- ***area***: El área de la región segmentada en píxeles.
- ***bbox*:** Las coordenadas de la caja delimitadora (*Bounding Box*) que rodea la región segmentada, en formato [coordenada 'x' del borde izquierdo de la caja delimitadora, coordenada 'y' del borde superior de la caja delimitadora, ancho, alto].
- ***predicted_iou*:** La Intersección sobre la Unión (IoU) predicho por el modelo para evaluar la calidad de la segmentación.
- ***point_coords:*** Las coordenadas de los puntos utilizados para generar la máscara, que pueden ser útiles para entender la ubicación de los puntos de entrada en la imagen original.
- ***stability_score***: Un puntaje que indica la estabilidad de la máscara. Valores más altos indican máscaras más estables.
- ***crop_box:*** Las coordenadas de la caja delimitadora, que indica la región de la imagen original de donde se generó la máscara.

En el proceso de segmentación de bloques se han desarrollado las siguientes funciones que se muestran en el Anexo 1:

- ***recortarCuadrantes* (ver Anexo A.1.2):** Esta función divide una imagen en un número especificado de cuadrantes de tamaño uniforme y devuelve una lista de estos cuadrantes.
- ***generarMascarasPorCuadrante* (ver Anexo A.1.4):** Esta función se encarga de iniciar el proceso de segmentación por cuadrante y almacenar la información de salida de las máscaras (que son los atributos especificados anteriormente) en una lista, donde cada posición corresponde a la información de cada cuadrante.
- ***superponerMascaras* (ver Anexo A.1.5):** Esta función itera sobre cada cuadrante para recoger la información de las máscaras, ajusta las coordenadas de la segmentación para que coincidan con la posición correcta en la imagen original y devuelve una nueva lista de diccionarios con la información de las máscaras actualizada.

Se aplicó una división en 16 cuadrantes; como resultado, en la Figura 40 se aprecia una salida con alrededor de 2800 máscaras y un tiempo de segmentación total de 5 minutos.

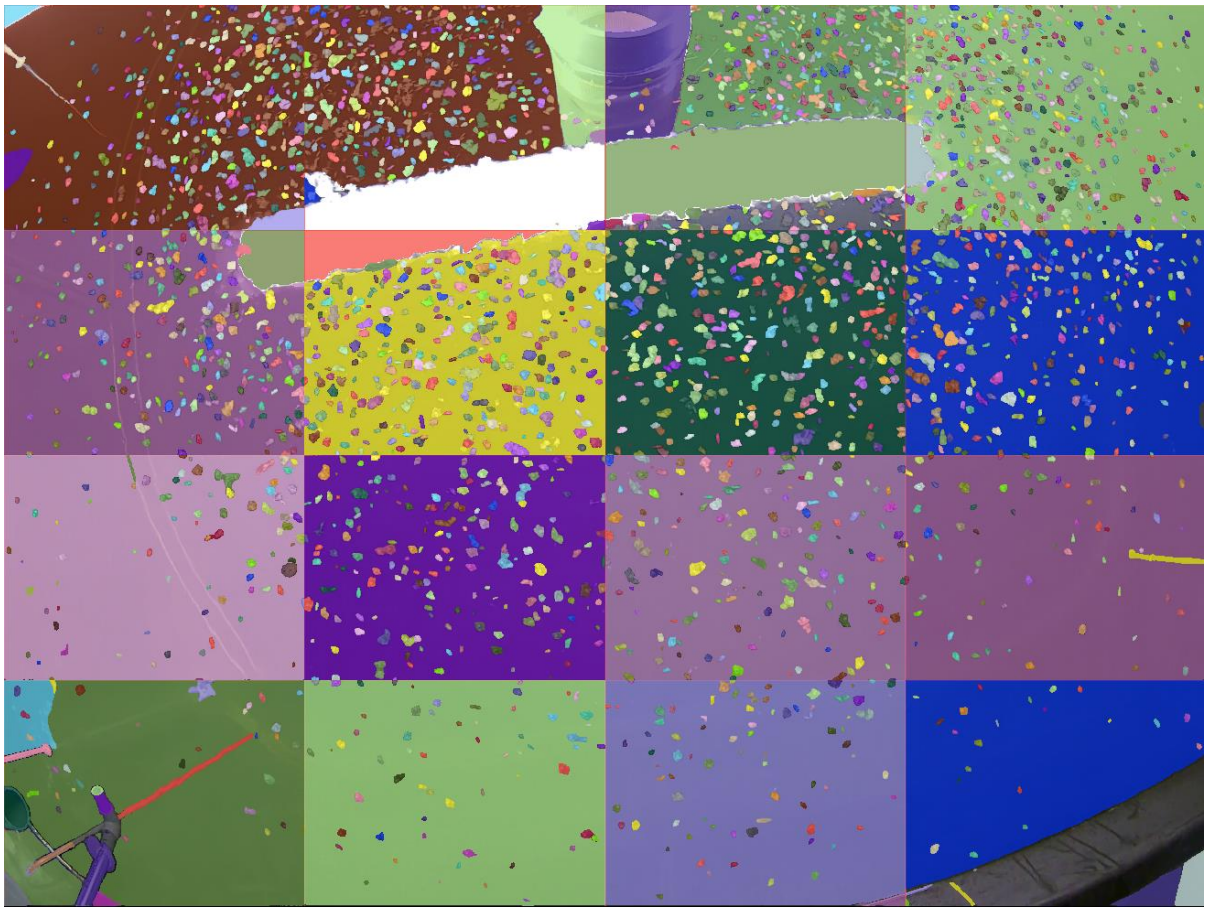

Figura 40 – Salida de imagen sin zoom (3) aplicando segmentación por bloques

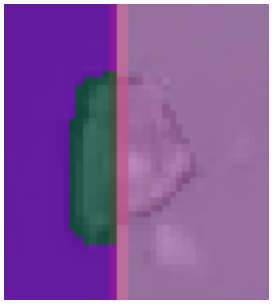

Figura 41 – Objeto limitante con los cuadrantes .

b) points_per_batch

Un lote o mejor conocido como *batch,* es un grupo de puntos de datos que se procesan simultáneamente en una única ejecución del modelo. En el caso de la segmentación de imágenes, un lote se refiere a un conjunto de puntos de entrada que el modelo analiza de una vez para generar las máscaras de segmentación correspondientes.

Este parámetro define cuántos puntos de entrada se procesan simultáneamente en un solo lote. Un valor más elevado de este parámetro reduce el tiempo de segmentación, pero requiere un mayor uso de memoria GPU, por ello se decidió establecer el valor por defecto de 64 puntos.

c) pred_iou_thresh

El índice de Jaccard o Intersección sobre la Unión (IoU) es una métrica que mide la superposición entre dos áreas, en este caso, entre la máscara predicha (máscara de referencia) y la máscara real. Se calcula como el área de la intersección entre las dos máscaras dividida por el área de su unión.

$$IoU = \frac{\text{Area de Intersección}}{\text{Area de la Unión}}$$

Este parámetro define el umbral de Intersección sobre la Unión (IoU) utilizado para determinar la calidad de las máscaras de segmentación predichas por el modelo. Un valor más alto de *pred_iou_thresh* indica que solo se considerarán de alta calidad las máscaras cuya superposición con la máscara predicha sea muy alta. Esto quiere decir que si se establece en 0.88, el modelo solo considerará las máscaras como válidas si tienen al menos un 88% de superposición con la máscara de referencia. Por el contrario, un valor bajo permite más predicciones con menor precisión, lo que puede aumentar la cantidad de máscaras generadas, pero también puede incluir más errores.

Se notó que al reducir el umbral por debajo de 0.8 se generaban muchas máscaras innecesarias, mientras que superar el valor de 0.9 resultaba en que muchos objetos no fueran segmentados. Por esta razón, se optó por dejar el umbral predeterminado de 0.88, con el cual obtuvimos los mejores resultados en la segmentación.

d) stability_score_thresh

Cuando el modelo SAM genera múltiples máscaras potenciales para un objeto en la imagen, cada una de estas máscaras tiene asociado un puntaje de estabilidad. Este puntaje indica qué tan robusta es la máscara en términos de calidad en relación con las otras máscaras generadas para el mismo objeto, tal y como se comentó en el punto "Predicción de máscara" del apartado 4.2.1.

El parámetro *stability_score_thresh* en SAM determina qué tan estricto será el modelo al elegir máscaras de segmentación. Un umbral alto asegura precisión al descartar máscaras menos estables, mientras que uno bajo puede ampliar la cobertura, aunque pueda incluir máscaras menos confiables. Se obtuvieron buenos resultados al establecer el valor por defecto 0.95.

e) stability_score_offset

Este parámetro es un ajuste adicional aplicado al puntaje de estabilidad vinculado al parámetro anterior para refinar la selección de máscaras. Este campo es interesante para segmentar objetos complejos que estén compuestos por varias máscaras, en nuestro caso no aplica por lo que se decidió a dejar su valor por defecto en 1.

f) box_nms_thresh

El NMS (*Non-Maximum Suppression*) es una técnica utilizada comúnmente en tareas de detección y segmentación de objetos para eliminar detecciones redundantes o superpuestas. Este parámetro controla cuán solapadas deben estar dos máscaras para que se considere que están superpuestas. Si el IoU entre dos máscaras es mayor que *box_nms_thresh*, una de ellas se suprime durante el proceso de NMS.

Un valor bajo de *box_nms_thresh* permitirá una supresión más agresiva de máscaras superpuestas, lo que puede resultar en menos máscaras finales, pero potencialmente más precisas. Por otro lado, un valor alto permitirá mantener más máscaras superpuestas, lo que podría aumentar la cobertura, pero a costa de posiblemente introducir máscaras redundantes.

El ajuste de este parámetro ha sido crucial para mejorar la segmentación de rodaballos. Se observa que, con un valor alto en este campo, se generaban múltiples máscaras para un mismo objeto (ver Figura 42). Por otro lado, al establecer un valor bajo, las agrupaciones de rodaballos interpretadas como un único objeto eran más frecuentes (ver Figura 43). Finalmente se estableció el valor en 0.4, este valor específico se seleccionó por su capacidad para lograr un equilibrio óptimo entre la segmentación precisa de objetos agrupados y la minimización de máscaras redundantes para un solo objeto.

Figura 43 – Varias máscaras de segmentación en un único rodaballo

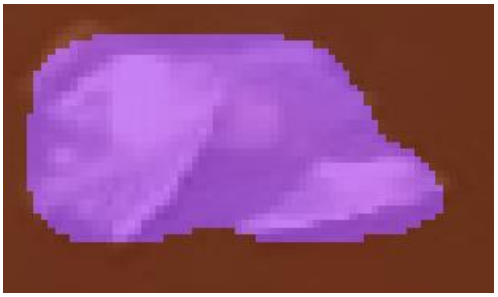

Figura 42 – Una única máscara de segmentación para varios rodaballos

g) crop_n_layers

Cuando SAM segmenta una imagen grande, puede dividirla en múltiples partes más pequeñas (*crops*) para facilitar la tarea de segmentación. El parámetro *crop_n_layers* determina cuántas capas de recorte se aplicarán. Cada capa de recorte puede dividir la imagen en partes más pequeñas, lo que permite que SAM se centre en segmentar regiones más manejables y detalladas de la imagen en lugar de tratar de manejar todo el contenido en una sola vez.

Al aumentar el valor de este parámetro se multiplicó exponencialmente el tiempo de segmentación y no se observaron mejorías apreciables por lo que se estableció en 0.

Existen parámetros de configuración para las capas de recorte equivalentes a los vistos anteriormente:

- ***crop_nms_thresh***: Umbral de supresión no máxima para recorte.
- ***crop_overlap_ratio****:* Proporción de superposición de recorte
- ***crop_n_points_downscale_factor***: Factor de reducción de puntos de recorte.

Como no hemos establecido capas de recorte estos parámetros no aplican para el modelo y por lo tanto dejamos sus valores por defecto.

h) point_grids

Se refiere a una lista de matrices que representan las cuadrículas de puntos de entrada sobre las cuales SAM realiza la segmentación automática. Cada matriz en la lista define un conjunto específico de puntos de entrada distribuidos uniformemente sobre la imagen para generar las máscaras de segmentación. En nuestro modelo este parámetro no aplica.

i) min_mask_region_area

Se refiere al área mínima que debe tener una máscara para ser considerada válida durante el proceso de segmentación automática. Este parámetro sí es interesante para poder filtrar objetos que debido a su tamaño tenemos la certeza de que no son rodaballos y podría ser comida distribuida en el tanque.

En las pruebas realizadas, se notó que los objetos deseados no estaban siendo filtrados correctamente, lo cual motivó la creación de una función de postprocesamiento dedicada a este propósito, descrita en detalle en la sección 4.5.1.

Este parámetro se estableció en 0.

j) output_mode

Especifica el formato en el que se devuelven las máscaras generadas por el modelo SAM. Las opciones de salida son:

- ***binary_mask***: Este modo devuelve las máscaras como imágenes binarias, donde cada píxel tiene un valor de 0 o 1, indicando la pertenencia del píxel al objeto segmentado.
- ***uncompressed_rle***: En este modo, las máscaras se devuelven en formato de codificación de longitud de ejecución (RLE) sin comprimir. RLE es un formato eficiente para representar máscaras segmentadas, donde se codifica la longitud de las secuencias de píxeles contiguos.
- ***coco_rle****:* Este modo devuelve las máscaras en el formato de codificación de longitud de ejecución (RLE) utilizado por el conjunto de datos COCO. Este formato está optimizado para reducir el espacio de almacenamiento y la transferencia de datos al trabajar con grandes conjuntos de datos de imágenes.

En nuestro caso al estar tratando con imágenes individuales es interesante la opción *binary_mask*.

Una vez analizados todos los parámetros de segmentación, llegamos a la conclusión de la utilización de los siguientes valores para ambos niveles de zoom teniendo en cuenta la correcta segmentación por bloques en el caso de las imágenes sin zoom.

- points_per_side = 64
- points_per_batch = 64
- pred_iou_thresh = 0.88
- stability_score_thresh = 0.95
- stability_score_offset = 1
- box_nms_thresh = 0.2
- crop_n_layers = 0
- min_mask_region_area = 0
- output_mode = "binary_mask"

4.5 Etapa de postprocesamiento

4.5.1 Filtro de máscaras

El objetivo principal de esta etapa es filtrar las máscaras generadas por el modelo para obtener el menor error de cuantificación de los rodaballos posible. Se desarrolló la función *procesarMáscaras* (ver Anexo A.1.8), adaptada de una función del repositorio público de GitHub *tnia-python* [42]. Esta función, basada en la lista de información de las máscaras, da solución a los siguientes dos puntos:

- Como se puede ver en las Figuras 38 y 40 que pertenecen a la salida del proceso de segmentación de imágenes con zoom y sin zoom respectivamente, se generan máscaras que corresponden al fondo de la imagen o de los cuadrantes, además de otras máscaras pertenecientes al foco de luz, bordes del tanque o comida.
- Capacidad de filtrar máscaras según la intensidad de los píxeles.

Para ello a esta función se le pasan los siguientes parámetros:

min_size: Tamaño mínimo de máscara, medido como píxeles contenidos dentro de la máscara. Tras diversas pruebas, se concluyó que, para imágenes con zoom, el tamaño promedio de los rodaballos más pequeños es de 100 píxeles; disminuir este valor por debajo de 70 píxeles permite que objetos más pequeños, como la comida en el tanque, pasen el filtro. Para imágenes sin zoom, el tamaño mínimo se estableció en 50 píxeles, ya que los rodaballos aparecen más pequeños.

max_size: Tamaño máximo de máscara, definido como el cociente entre el número de pixeles contenidos dentro de la máscara y el número total de píxeles de la imagen. Esto permite estimar fácilmente el tamaño máximo de máscara deseado. Por ejemplo, si se desea que el tamaño máximo de máscara sea un cuarto de la imagen, basta con establecer este parámetro en 0.25. Para una imagen de 800 x 600 píxeles (480,000 píxeles), el tamaño máximo de la máscara sería 120,000 píxeles. Durante las pruebas, se estableció un valor de 0.00150 para imágenes con zoom y 0.00050 para imágenes sin zoom.

min_intensity: La intensidad mínima requerida para que un píxel sea considerado parte de una máscara. Ayuda a eliminar regiones que no cumplen con el umbral de intensidad mínimo.

Esta etapa también implica un consumo significativo de recursos computacionales y tiempo, dado que cada máscara generada debe ser procesada individualmente. Por lo tanto, el tiempo de procesamiento aumenta proporcionalmente según el número de máscaras generadas. A continuación, se presentan las imágenes de segmentación resultantes después de aplicar el filtro de máscaras. La Figura 44 muestra la segmentación de la imagen con zoom donde el tiempo de procesamiento de esta etapa ha sido de 3 minutos, mientras que la Figura 45 muestra la segmentación de una imagen sin zoom y el tiempo de procesamiento ha sido de 10 minutos.

Figura 44 – Salida de imagen con zoom (3) de la etapa de filtrado

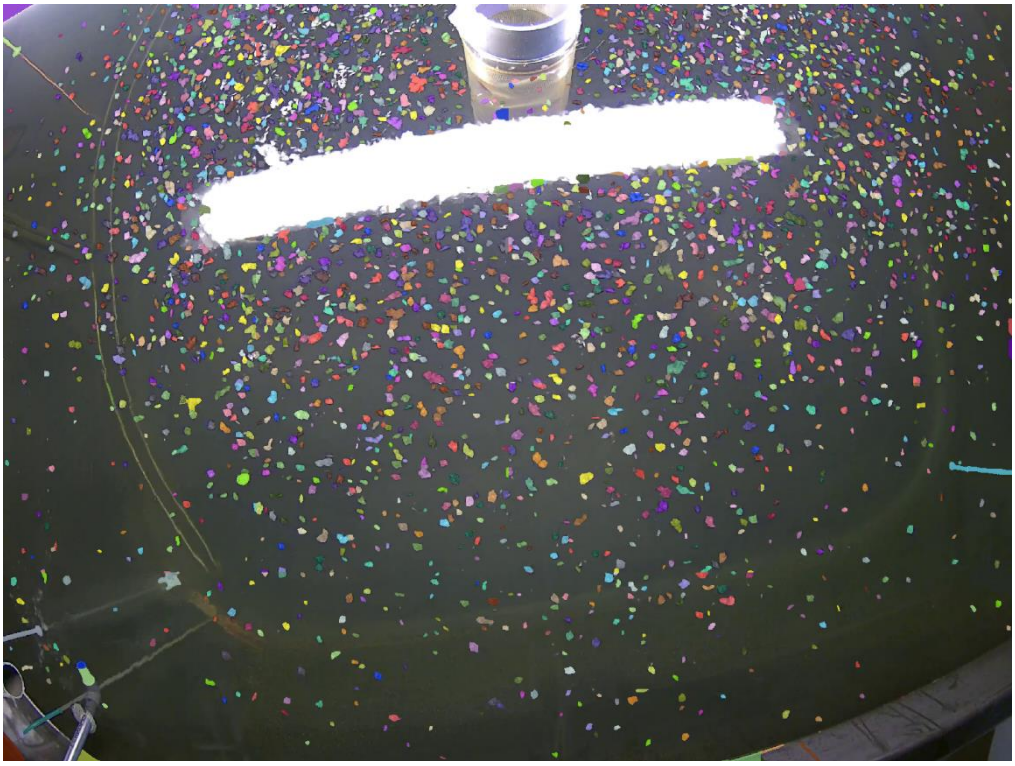

Figura 45 – Salida de imagen sin zoom (3) de la etapa de filtrado

4.5.2 Centroides de objetos

Durante esta etapa, se busca marcar el centro de los objetos segmentados, ya que esto proporciona una información rápida sobre si el modelo ha segmentado correctamente los objetos de interés.

Para lograrlo, se desarrolló la función *pintarCentroidesMascaras (ver Anexo A.1.6)*, que utiliza la librería *SciPy* para calcular el centroide de cada máscara seleccionada y pintar un punto en la posición correspondiente.

En la Figura 46, se muestra el resultado de esta etapa para un grupo de rodaballos:

Figura 46 – Centroides sobre los rodaballos

4.6 Implementación de la interfaz

Uno de los objetivos de este proyecto era que la aplicación fuera intuitiva y de fácil manejo para los usuarios, es por ello, que se decidió implementar el desarrollo haciendo uso de la biblioteca *Napari*.

Napari fue diseñada para la visualización interactiva de datos científicos, especialmente útil para imágenes y datos volumétricos. Proporciona una interfaz gráfica de usuario (GUI) que permite a los investigadores explorar datos de manera eficiente y realizar análisis interactivos [43]. Sus características principales son

- Permite cargar y visualizar datos complejos, como imágenes multidimensionales, volúmenes y datos de puntos.
- Permite la integración de complementos y extensiones para adaptarse a diferentes tipos de datos.
- Ofrece una interfaz de usuario amigable (ver Figura 47) que facilita la navegación, el ajuste de parámetros y la realización de análisis directamente desde la visualización al organizar las imágenes como una "pila de capas". Esta estructura permite una gestión eficiente de datos multidimensionales y la aplicación de herramientas de análisis específicas a cada capa de manera intuitiva.

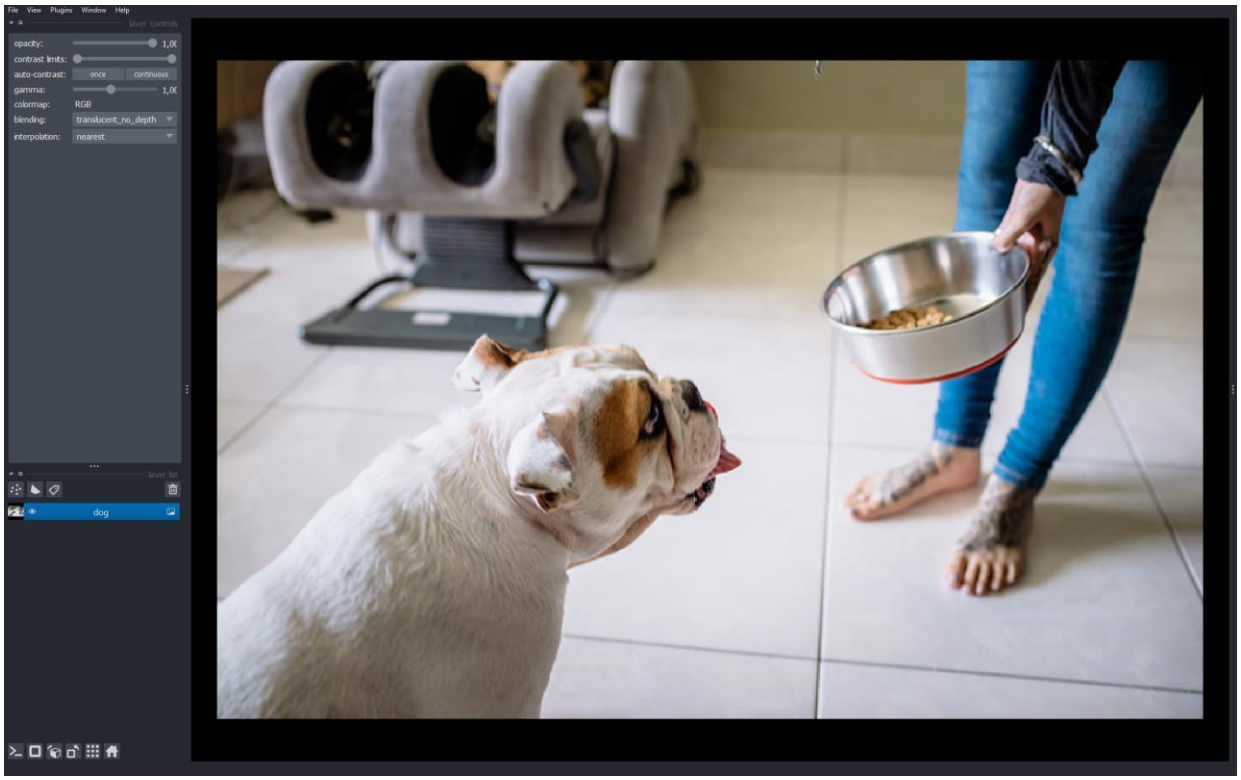

Figura 47 – Visor *Napari*

Para integrar completamente el desarrollo en el visor de *Napari*, se han tenido en cuenta una serie de aspectos clave:

a) La biblioteca *Napari* ofrece una variedad de *widgets* que permiten la interacción con el usuario, como *checkboxes*, listas, botones y campos de texto. Estos *widgets* se organizan en un bloque dividido en secciones para cargar la imagen de entrada, configurar los parámetros del modelo, iniciar el proceso de segmentación, exportar resultados y mostrar información al usuario mediante un cuadro de salida.

b) Para superponer las máscaras generadas por el modelo sobre la imagen original, es necesario almacenarlas en una capa de *labels* (etiquetas). Estas etiquetas representan una imagen segmentada donde cada píxel está asignado a una clase o categoría específica mediante un valor entero. Antes de ser visualizadas en el visor, la lista de máscaras pasa por la función *mostrarLabels (*ver Anexo A.1.7*)*, que las transforma en una matriz de etiquetas. Esta matriz refleja las regiones segmentadas de la imagen, permitiendo una visualización clara en la interfaz de *Napari*.

c) Las máscaras generadas por el modelo *MobileSAM*, las resultantes del postprocesamiento, y los centroides, se representan como una "pila de capas" (ver Figura 48). Esta estructura permite al usuario elegir cuáles visualizar en cada momento o si visualizar todas superponiéndose estas con la imagen cargada. Para gestionar la incorporación de estas capas al visor, se utiliza un temporizador que, cada minuto, llama a la función *__cargarMascaras (*ver Anexo A.1.10*)*. Esta función verifica la existencia de nuevas capas y las añade al visor, asegurando que la visualización esté siempre actualizada.

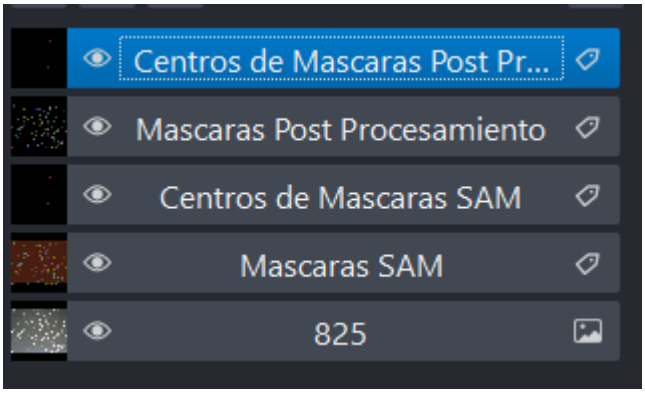

Figura 48 – Pila de capas

La integración de estos aspectos, junto con la orquestación de todas las funciones mencionadas en los puntos anteriores, se gestiona a través de la clase *NapariSAM*. Esto inicializa la interfaz *Napari* y gestiona los procesos iniciados por el usuario sobre el bloque de *widgets*.

En la Figura 49, se muestra el bloque con los *widgets* necesarios para el funcionamiento de la aplicación y en la Figura 50 el resultado final de la interfaz.

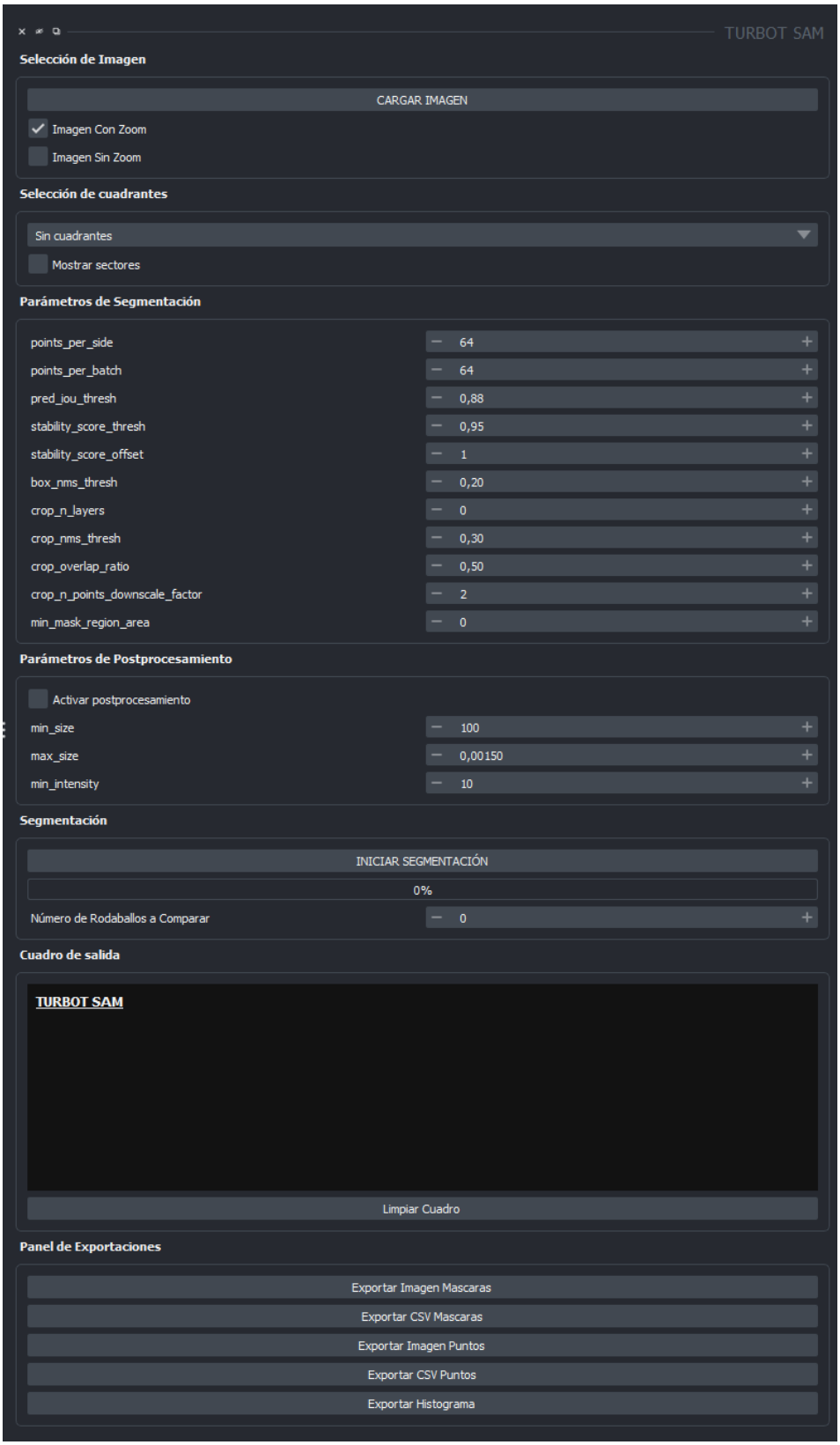

Figura 49 – Bloque de *widgets*.

Figura 50 – Interfaz de la aplicación

El bloque está formado por varias secciones:

1. Selección de imagen

Permite al usuario cargar la imagen a partir de un cuadro de diálogo que permite seleccionar la imagen deseada en cualquier ruta. Tras ello la imagen se carga en el visor y el usuario debe seleccionar si se trata de una imagen con zoom o sin zoom.

2. Selección de cuadrantes

En esta sección, se ofrece al usuario la opción de ejecutar un proceso de segmentación por bloques o una segmentación normal, permitiéndole elegir el número de cuadrantes. Se puede seleccionar entre 4 cuadrantes, que es óptimo para algunas imágenes con zoom, hasta 32 cuadrantes, recomendables para imágenes sin zoom con una gran cantidad de rodaballos. Las líneas divisoras de los cuadrantes se almacenan como una imagen de etiquetas para representarlas sobre la imagen original, además, se puede activar la opción “Mostrar sectores” para una mejor visualización del área de los cuadrantes (ver Figura 51).

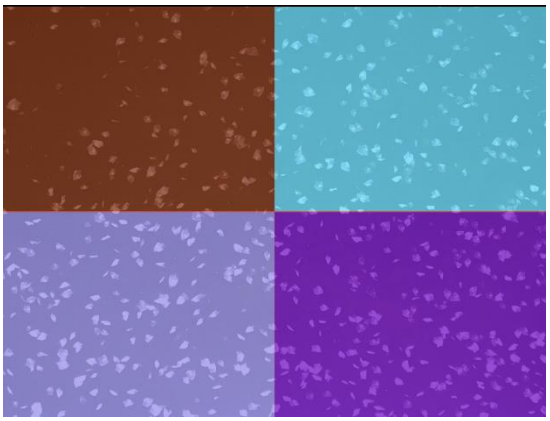

Figura 51 – Cuadrantes con sectores

3. Parámetros de segmentación

Una lista de los parámetros de segmentación del modelo SAM. El usuario tendrá la opción de configurarlos a medida o dejar los valores por defecto analizados en el punto 4.4.2.

4. Parámetros de postprocesamiento

Si se activa esta opción, tras iniciar el proceso de segmentación en el visor aparecerán las siguientes imágenes de etiquetas:

- **Mascaras SAM:** Máscaras generadas por el modelo de segmentación.
- **Centros de Mascaras SAM**: Centroides de las máscaras generadas por el modelo
- **Mascaras PostProcesamiento:** Máscaras de salida de la función de filtrado del apartado 4.5.1
- **Centros de Mascaras PostProcesamiento**: Centroides de las máscaras filtradas en la etapa de postprocesamiento.

De lo contrario, únicamente se generarán “Mascaras SAM” y “Centros de Mascaras SAM”. Al igual que en el apartado anterior, se da al usuario la opción de establecer los parámetros de postprocesamiento de forma manual o bien dejar sus valores por defecto en función del nivel de zoom seleccionado como se mostró en el apartado 4.5.1.

5. Segmentación

Al iniciar el proceso de segmentación, se ejecuta la función *iniciarSegmentacionThread (ver Anexo A.1.9)*, la cual inicia el proceso de segmentación en un nuevo hilo utilizando la librería *threading* de *Python*. Esto permite que, durante el tiempo de procesamiento, el usuario pueda analizar la imagen de entrada y las máscaras que se vayan generando en tiempo real. Además, el usuario puede utilizar las herramientas que ofrece *Napari*, como ajuste de contraste, brillo, y opacidad, sin interrupciones en la interfaz. Por lo tanto, es posible iniciar varios procesos de segmentación en paralelo, teniendo en cuenta la capacidad computacional del PC donde se ejecute la aplicación.

Se añade una barra de progreso para indicar el porcentaje completado del proceso de segmentación y un campo de entrada para establecer el número de rodaballos en la imagen, si se conoce. Este dato se utilizará para calcular los errores absoluto y relativo en la cuantificación de los rodaballos, lo que permitirá ajustar los parámetros de segmentación de manera más precisa.

6. Cuadro de salida

En él se muestran tres tipos de información indicados bajo los siguientes indicativos (ver Figura 52):

- **[INFO]:** Información relacionada con las etapas de los procesos de segmentación, con la interacción de los *widgets* y los resultados finales de la cuantificación de rodaballos.
- **[WARNING]:** Avisa al usuario de que ha realizado una operación no permitida como por ejemplo iniciar un proceso de segmentación sin previamente cargar una imagen.
- **[ERROR]:** Muestra la salida de un error interno del programa

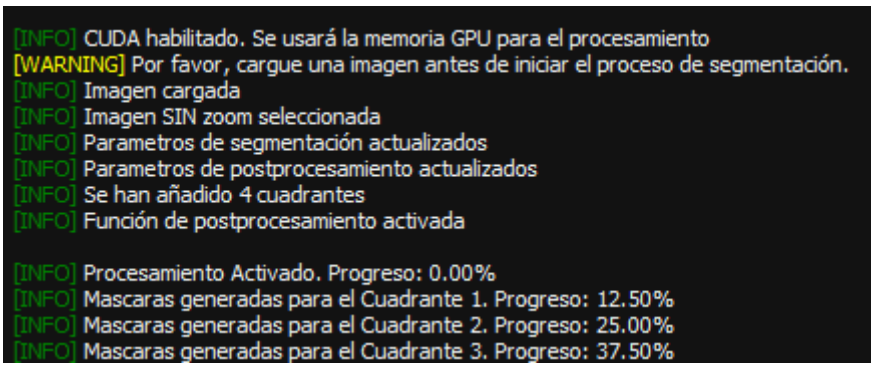

Figura 52 – Cuadro de salida

7. Panel de Exportaciones

Contiene varias opciones de exportación de los resultados del proceso de exportación permitiendo al usuario almacenar la información exportada en la ruta indicada:

a) Imagen de Máscaras

Es una imagen exportada en formato *.png* o *.jpg* que contiene la superposición de la imagen de entrada con las máscaras generadas en el proceso de segmentación o bien si se ha seleccionado la opción de postprocesamiento las máscaras resultantes de esta etapa. La Figura 45 sería un ejemplo de exportación de esta opción.

b) CSV Máscaras

Es un archivo CSV, donde cada fila hace referencia a cada máscara generada en el proceso de segmentación o bien a la etapa de postprocesamiento. Cada columna representa uno de los

atributos de las máscaras, que incluyen: *área*, *bbox*, *predicted_iou*, *point_coords*, *stability_score* y *crop_box*, tal como se mencionó en la página 41. En la Figura 53 hay un ejemplo de los datos que componen este archivo.

```
area,bbox,predicted_iou,point_coords,stability_score,crop_box
4862683,[0\, 0\, 2559\, 1916],1.0348519086837769,[[220.0\, 675.0]],0.9803174734115601,[0\, 0\, 2560\, 1920]
1695,[2034\, 1691\, 51\, 41],0.9580685496330261,[[2060.0\, 1695.0]],0.9859648942947388,[0\, 0\, 2560\, 1920]
2114,[2055\, 1489\, 47\, 54],0.9558513164520264,[[2100.0\, 1485.0]],0.9761793613433838,[0\, 0\, 2560\, 1920]
1336,[467\, 973\, 35\, 47],0.9553653597831726,[[500.0\, 975.0]],0.9778270721435547,[0\, 0\, 2560\, 1920]
1456,[1316\, 1856\, 45\, 39],0.9539629817008972,[[1340.0\, 1875.0]],0.9883959293365479,[0\, 0\, 2560\, 1920]
2368,[189\, 388\, 64\, 48],0.9538201689720154,[[220.0\, 435.0]],0.9861634969711304,[0\, 0\, 2560\, 1920]
1221,[1843\, 1673\, 34\, 42],0.9506134390830994,[[1860.0\, 1695.0]],0.9869918823242188,[0\, 0\, 2560\, 1920]
1294,[543\, 994\, 33\, 49],0.950200617313385,[[540.0\, 1005.0]],0.9869731664657593,[0\, 0\, 2560\, 1920]
1033,[766\, 1694\, 38\, 36],0.9487231969833374,[[780.0\, 1695.0]],0.9808428883552551,[0\, 0\, 2560\, 1920]
```

Figura 53 – CSV Máscaras

c) Imagen de Puntos

Es una imagen exportada en formato *.png* o *.jpg* que contiene la superposición de la imagen de entrada con los puntos de los centroides generados para las máscaras resultantes. La Figura 46 sería un ejemplo de exportación de esta opción.

d) CSV Puntos

Es un archivo CSV en el que cada fila hace referencia a las posiciones *x,y* de los centroides generados para cada máscara. La Figura 54 muestra un ejemplo del contenido de este archivo.

```
x,y
1278,952
2057,1711
2079,1516
484,997
1337,1874
222,411
1859,1693
558,1017
785,1711
```

Figura 54 – CSV Puntos

e) Histograma

Es una imagen exportada en formato *.png* que contiene un histograma donde el eje *x* representa cada una de las máscaras generadas y en el eje *y* el área de cada una de ellas representada como número de píxeles contenidos dentro de la máscara; adicionalmente, se representan los valores del atributo *stability_score* normalizados y se representa una línea de tendencia sobre las áreas.

El histograma nos ofrece información relativa a:

- Representación del tamaño de los objetos segmentados.
- Representación del índice Jaccard para evaluar la calidad de las máscaras generadas.
- Variaciones de la calidad de las máscaras en función de su área.

La exportación de la imagen con el histograma está condicionada a la selección previa de la opción de postprocesamiento. Esto se debe a que, de lo contrario, el fondo de la imagen se consideraría como una máscara adicional, lo cual complicaría la representación del histograma. En las Figuras 55 y 56 se puede observar un ejemplo de la generación del histograma para la imagen con zoom y sin zoom respectivamente.

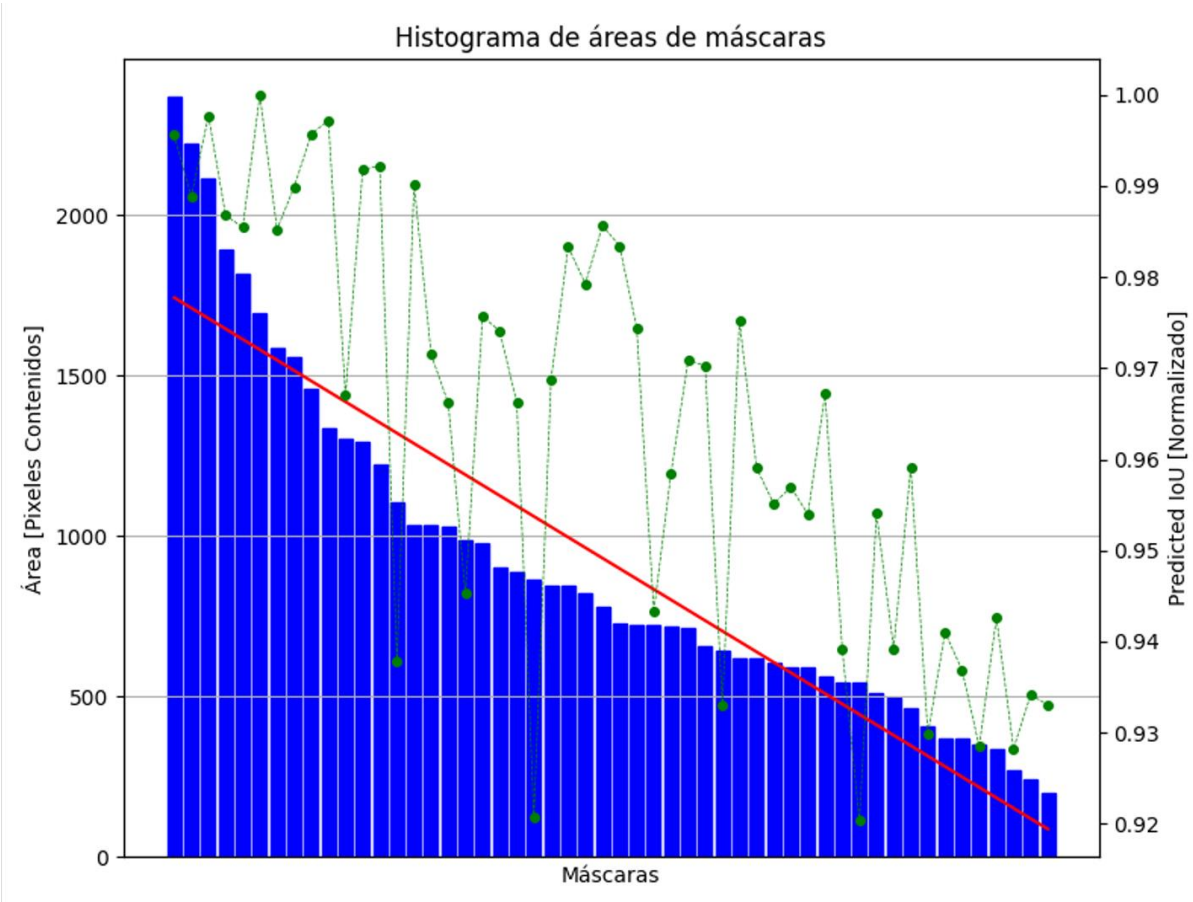

Figura 55 – Histograma imagen con zoom (3)

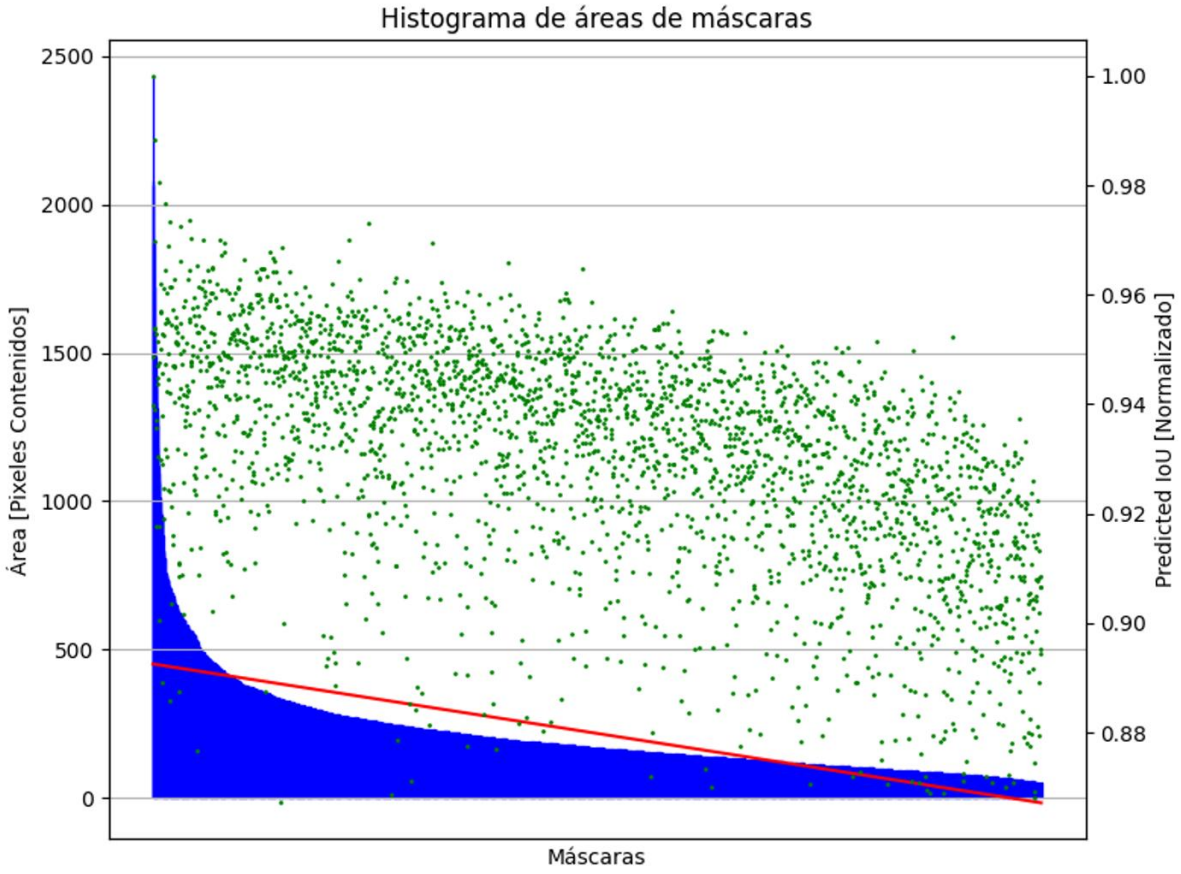

Figura 56 – Histograma imagen sin zoom (3)

5. Resultados

En este apartado se exponen los resultados finales de la interfaz de usuario desarrollada y los resultados de la cuantificación de los rodaballos para los distintos tipos de zoom haciendo uso de los bancos de imágenes comentados en el apartado 4.1 que fueron previamente etiquetadas.

5.1 Resultados de la interfaz de usuario

La aplicación desarrollada presenta una interfaz de usuario con las siguientes características:

Facilidad de uso: La interfaz permite a los usuarios interactuar con las imágenes y datos de manera eficiente, sin necesidad de un conocimiento técnico profundo. Esto se logra mediante el uso de *tooltips*, pequeñas ventanas emergentes que aparecen al colocar el cursor sobre elementos interactivos en la interfaz. Estos *tooltips* están presentes en cada panel, ofreciendo orientación y consejos para guiar al usuario en su uso (ver Figura 57).

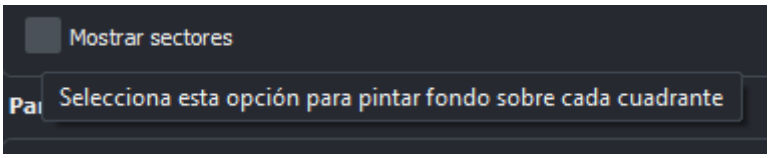

Figura 57 – *Tooltip*

Interacción dinámica: Ajuste manual de los parámetros de segmentación y postprocesamiento permitiendo a los usuarios una segmentación personalizable.

Gestión multicapa: Permite la organización de imágenes, máscaras y puntos como "pila de capas" lo que ayuda al usuario a la visualización y comparación simultanea de datos.

Visualización de resultados: El cuadro de salida informativo proporciona información clara sobre las etapas del proceso, advertencias y errores mediante categorización de los mensajes. En él se ven reflejados los resultados de cuantificación de los rodaballos y los errores cometidos sobre una imagen previamente cuantificada (ver Figura 58).

Ejecución simultánea de tareas: Permite la ejecución concurrente de varios procesos de segmentación, lo que habilita al usuario para analizar resultados previos mientras se procesa otro conjunto de datos.

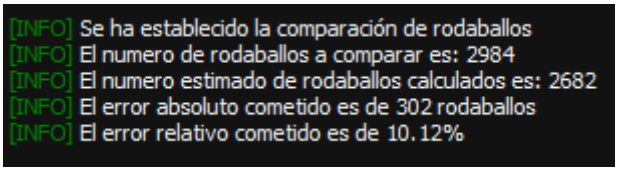

Figura 58 – Salida de cuantificación

En la Figura 59 se puede observar el resultado final de la interfaz de usuario.

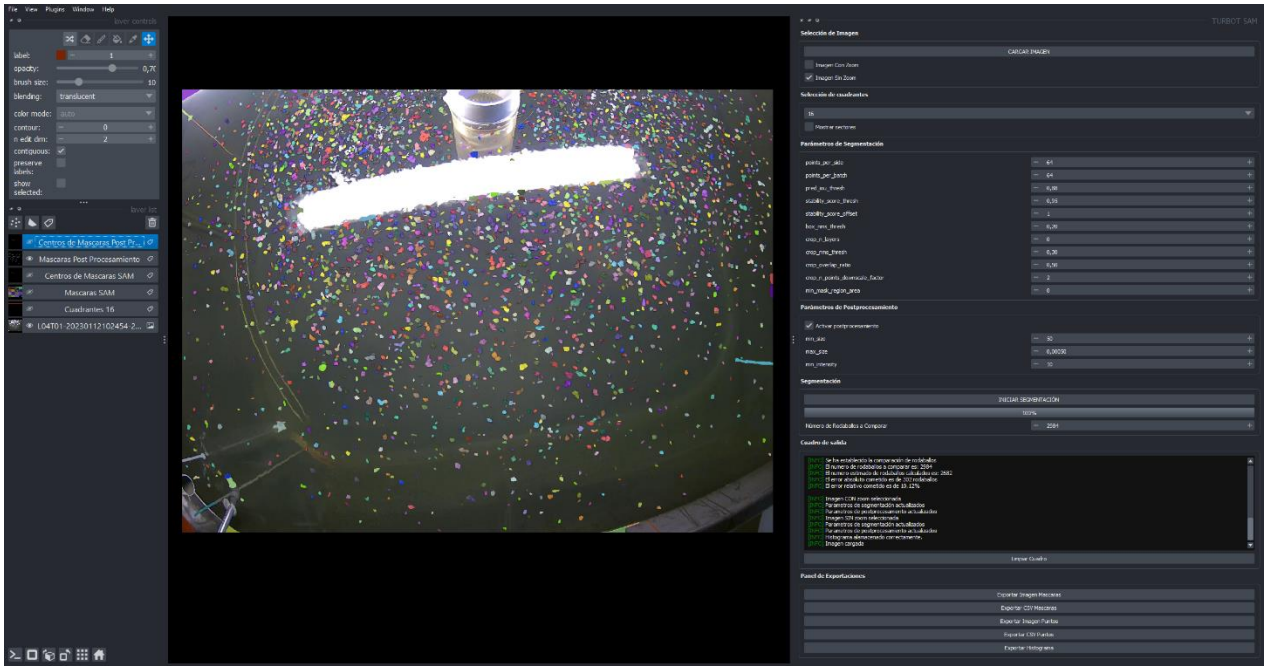

Figura 59 – Interfaz de usuario

5.2 Resultados de imágenes con zoom

A continuación, se mostrarán los resultados de la cuantificación de rodaballos para las 30 imágenes con zoom etiquetadas disponibles que se han seleccionado para verificar la aplicación.

En el repositorio *git* del desarrollo, en el directorio *imágenes/conZoom* se almacenan 30 carpetas (ordenadas según el número de rodaballos etiquetados manualmente) que hacen referencia a cada imagen de test y que contiene la siguiente información:

- Imagen original sin etiquetar.
- Archivo *.mat* generado tras el proceso de etiquetado manual descrito en el apartado 4.1.1.
- Imagen con los centros de máscaras generados por la aplicación.
- CSV de Puntos generados.
- Histograma generado.

En la Tabla 2 se detallan los resultados finales, donde se han aplicado los valores de los parámetros establecidos en los apartados 4.4.2 y 4.5.1. En cada imagen, se ha seleccionado segmentación de bloques o no, según la propia interpretación (basada en la densidad de los rodaballos en la imagen, la prominencia de los rodaballos en el fondo y el tamaño relativo de los rodaballos), para un mejor resultado y establecido el número de cuadrantes para la segmentación en tal caso.

Imagen	N.R	N.E	S.B	E.A	E.R
L4T8-20220224180650-20220224181150_2250	89	52	No	37	41.57 %
L4T8-20220224152634-20220224153135_2265	104	54	No	50	48.08 %
L4T2-20220109081551-20220109082051_300	145	165	4	20	13.79 %
L4T2-20220109074548-20220109075048_300	174	198	4	24	13.79 %
L4T2-20220109083052-20220109083552_300	199	217	4	18	9.05 %
L4T8-20220109100058-20220109100558_300	314	327	4	13	4.14 %
L4T8-20220109095057-20220109095557_300	316	340	4	24	7.59 %
L4T8-20220109075045-20220109075546_302	325	316	4	9	2.77 %
L4T8-20220109070040-20220109070541_302	329	333	4	6	1.77 %
L4T8-20220109094556-20220109095057_302	332	342	4	10	2.90 %
L4T8-20220109095557-20220109100058_302	334	348	4	14	4.19 %
L4T8-20220109073043-20220109073544_302	340	317	4	23	6.76 %
L4T8-20220109065039-20220109065540_302	342	358	4	16	4.68 %
L4T8-20220109072543-20220109073043_300	347	374	4	27	7.78 %
L4T8-20220109094056-20220109094556_300	350	368	4	18	5.14 %
L4T8-20220109064539-20220109065039_300	351	356	4	5	1.42 %
L4T8-20220109065540-20220109070040_300	359	360	4	1	0.28 %
L4T8-20220109070541-20220109071041_300	364	368	4	4	1.10 %
L04T07-20230222120450-20230222120950_2_300_c	457	467	4	10	2.14 %
L4T2-20220223231714-20220223232215_2265	712	680	9	32	4.49 %
L4T2-20220223232215-20220223232715_2250	718	726	9	8	1.11 %
L4T2-20220223231214-20220223231714_2250	745	708	9	37	4.97 %
L4T2-20220223223210-20220223223710_2250	759	733	9	26	3.43 %
L4T2-20220223224211-20220223224711_2250	762	738	9	24	3.15 %
L4T2-20220223234717-20220223235217_2250	776	705	9	71	9.15 %
L4T2-20220222235722-20220223000001_1200	812	742	9	70	8.62 %
L4T2-20220223000501-20220223001002_2265_c	825	765	9	60	7.27 %
L4T2-20220223222709-20220223223210_2265	855	770	9	85	9.94 %
L4T2-20220223000001-20220223000501_2250	898	796	9	102	11.36 %
L4T2-20220223092555-20220223093056_2265	902	815	9	87	9.65 %

Tabla 2 – Resultados de la cuantificación de rodaballos para imágenes con zoom

La Tabla 2 está compuesta por:

- Imagen: Nombre de la imagen de test.
- N.R: Numero de rodaballos etiquetados manualmente.
- N.E: Numero de rodaballos estimados por el modelo.
- S.B: Numero de cuadrantes establecidos en el caso de haber usado segmentación de bloques, en caso negativo el valor se establece en "No".
- E.A: Error absoluto cometido medido en número de rodaballos.
- E.R: Error relativo cometido medido en %.

En cuanto a los resultados mostrados hay que destacar:

a) Para las imágenes con un número muy reducido de rodaballos (entre 50 y 100), se observa un error considerable. Esto se debe principalmente a que, al no aplicar una segmentación por bloques, no se detectan los rodaballos más pequeños. Sin embargo, al establecer una segmentación por bloques de 4 cuadrantes, el modelo segmenta muchas partes del fondo reflejadas por la luz. En este tipo de imágenes, sería recomendable utilizar una segmentación por bloques de 4 cuadrantes y ajustar el parámetro *points_per_side* a 32, esto permitirá segmentar todos los rodaballos sin incluir otras partes. Cabe mencionar que estas imágenes son las que menos importancia tienen, ya que su etiquetado manual es muy sencillo y prácticamente no requieren tiempo.

b) Para el resto de las imágenes, el modelo logra segmentar correctamente la mayoría de los rodaballos, proporcionando una estimación aceptable para la cuantificación.

5.3 Resultados de imágenes sin zoom

A continuación, se mostrarán los resultados de la cuantificación de rodaballos para las 15 imágenes sin zoom etiquetadas disponibles que se han seleccionado para verificar la aplicación.

Al igual que en el apartado anterior, en el repositorio git existe el directorio imágenes/sinZoom donde se almacenan 15 carpetas que hacen referencia a las imágenes de test y que contienen:

- Imagen original sin etiquetar.
- Archivo *.csv* generado tras el proceso de etiquetado manual descrito en el apartado 4.1.2.
- Imagen con los centros de máscaras generados por la aplicación.
- CSV de Puntos generados.
- Histograma generado.

De modo análogo al apartado anterior, se presenta la Tabla 3 que recoge los resultados del banco de imágenes etiquetadas sin zoom.

Imagen	N.R	N.E	S.B	E.A	E.R
L01T02-20221116113750-20221116114250_1_4425	407	461	9	54	13.27 %
L06T11-20230314113554-20230314114054_2_630	520	546	4	26	5.0 %
L05T02-20221121113453-20221121113953_1_4200	571	590	9	19	3.33 %
L02T04-20230123092835-20230123093335_2_390	696	726	9	30	4.31 %
L04T11-20230425110131-20230425110731_2_1260	848	768	9	80	9.43 %
L05T02-20221124114737-20221124115237_2_2160	918	859	9	59	6.43 %
L04T06-20230530112604-20230530113204_2_2850	1063	928	9	135	12.7 %
L06T05-20230501125945-20230501130545_2_450	1130	1120	9	10	0.88 %
L04T07-20230222120450-20230222120950_2_1425	1377	1388	16	11	0.8 %
L02T07-20230415122741-20230415123341_1_4440	1632	1611	16	21	1.29 %
L02T06-20230414113543-20230414114143_2_345	1819	1875	16	56	3.08 %
L02T06-20230414113543-20230414114143_2_345	1851	1875	16	24	1.3 %
L02T01-20230414105314-20230414105914_2_2880	2229	2032	16	197	8.84 %
L05T12-20221124121112-20221124121612_2_855	2347	2549	16	202	8.61 %
L05T08-20230213113139-20230213113639_2_1245	2420	2551	25	131	5.41 %
L03T02-20230310113414-20230310113914_2_2340	2480	2300	25	180	7.26 %
L05T09-20221121122050-20221121122550_2_210	2647	2546	25	101	3.82 %
L05T01-20230614101138-20230614101738_2_105	2759	3083	25	324	11.74 %
L04T01-20230112102454-20230112102954_1_4425	2984	2898	25	86	2.88 %
L06T08-20230314115713-20230314120213_2_1875	3020	2990	36	30	0.99 %
L02T04-20230523114242-20230523114842_2_4080	3273	3253	36	20	0.61 %
L05T07-20221121123225-20221121123725_2_1800	3753	4196	36	443	11.8 %
L05T11-20230614105345-20230614105945_2_2340	3966	3493	36	474	11.93 %
L04T03-20230112104308-20230112104808_2_2955	4003	4260	36	257	6.42 %
L03T10-20230310124131-20230310124631_1_4305	4408	4150	36	248	5.85 %
L04T07-20221201124721-20221201125221_2_1755	4535	4583	36	48	1.06 %
L03T03-20230310114142-20230310114642_1_4440	4583	4449	36	134	2.92 %
L04T01-20230425101519-20230425102119_2_645	4651	4359	36	292	6.28 %
L03T09-20230310124657-20230310125157_1_4470	4905	4376	36	529	10.78 %

Tabla 3 – Resultados de la cuantificación de rodaballos para imágenes sin zoom

En las imágenes sin zoom, observamos buenos resultados en la estimación de conteo. De las 30 imágenes analizadas, solo 4 presentan un error entre el 10 % y el 14 %, mientras que el resto tiene un error menor al 10 %, esto indica que la aplicación logra una estimación adecuada en este tipo de imágenes que son las que más atención requieren. Parte del error puede deberse a la dificultad asociada con el etiquetado manual.

6. Presupuesto

En este apartado se especifica la lista de recursos utilizados durante el desarrollo del proyecto, así como las horas del personal requerido y el precio de estos.

El presupuesto final ha sido calculado considerando que para la realización del proyecto han hecho falta 80 horas de investigación y 270 horas en la fase de desarrollo, dando lugar a 350 horas totales empleadas.

En la Tabla 4 se muestra el presupuesto general.

Descripción	Unidad de medida	Cantidad de unidades	Precio unitario	Precio Total
Personal de desarrollo	**Hora**	**350**	**35 €**	**12.250 €**
Ordenador: **13th Intel(R) Core(TM) i7-13700F 2.10 GHz** **16,0 GB RAM** **NVIDIA GeForce RTX 306 (8 GB GPU)** **Sistema Operativo Windows 11**	**N/A**	**1**	**1300 €**	**1300 €**
Pantalla AOC	**N/A**	**1**	**340 €**	**340 €**
Editor de código Visual Studio Code (VS Code)	**N/A**	**1**	**0 €**	**0 €**
Lenguaje de programación *Python*	**N/A**	**1**	**0 €**	**0 €**
TOTAL	**13.890 €**			

Tabla 4 – Presupuesto general del proyecto

7. Impacto del proyecto

Este proyecto pone de manifiesto las siguientes implicaciones:

Industriales: Al contar con una herramienta de cuantificación del número de rodaballos en los tanques de acuicultura, las piscifactorías pueden optimizar las condiciones de cría y alimentación para maximizar la producción

Sociales: La adopción de tecnologías avanzadas como la inteligencia artificial en la acuicultura puede requerir capacitación técnica especializada para operar y mantener los sistemas automatizados. Esto podría crear oportunidades de empleo en nuevas áreas como la tecnología aplicada a la acuicultura y la gestión de datos.

Ambientales: Al optimizar el manejo de las poblaciones de peces mediante tecnologías como el conteo automático, se puede contribuir a la conservación de especies marinas al evitar la sobreexplotación y asegurar prácticas de cultivo sostenibles.

En relación con los Objetivos de Desarrollo Sostenible (ODS) planteados por las Naciones Unidas en la agenda 2030 [44].

ODS 2 - **Hambre cero:** La estimación automática del número de peces conlleva a un aumento de los procesos de producción de las piscifactorías ayudando a combatir la inseguridad alimentaria.

ODS 9 - **Trabajo decente y crecimiento económico**: Al fomentar la adopción de aplicaciones basadas en inteligencia artificial, se impulsa la contratación de personal especializado y se estimula el desarrollo tecnológico en el sector de la acuicultura.

ODS 12 **- Producción y consumo responsables**: Al introducir un método automatizado y no intrusivo para el conteo de peces en acuicultura se promueve la gestión más eficiente de los recursos acuáticos al minimizar errores humanos y optimizar la producción de peces, lo cual puede ayudar a reducir el desperdicio y promover prácticas más sostenibles en la industria acuícola.

ODS 14 - **Vida submarina**: Utilizar métodos no invasivos para el conteo de peces en tanques de acuicultura ayuda a proteger la vida marina al fomentar el cultivo de peces en entornos controlados en lugar de depender de la pesca en los océanos, lo cual reduce la presión sobre los ecosistemas marinos y contribuye a la conservación de la biodiversidad submarina.

8. Conclusiones

La aplicación aborda de manera efectiva el desafío actual de estimar el número de rodaballos de manera automática, especialmente en imágenes con más de 500 peces, donde el etiquetado manual consumía mucho tiempo y esfuerzo.

Como vemos en los resultados, en cuanto a las imágenes con zoom, para aquellas en las que hay un número superior a 200 rodaballos, que son las que comienzan a ser más laboriosas de etiquetar manualmente, el error relativo apenas supera el 10 % y en las imágenes sin zoom que requieren un tiempo de etiquetado manual significativo (alrededor de 30 minutos por imagen), el error no excede el 13.5 %. Es crucial enfatizar la importancia de que los usuarios realicen una selección precisa de los cuadrantes durante el proceso de segmentación de bloques, ya que esto garantiza la obtención de los resultados deseados.

El mayor error producido en las imágenes con zoom coincide con una menor aparición de objetos en ellas. En estas circunstancias, es más probable que el modelo segmente partes del fondo que están afectadas por la luz, lo cual no puede ser compensado con los errores relacionados con los grupos de rodaballos, dado que este tipo de imágenes no los contiene. Esto no es un gran problema ya que el tiempo de etiquetado manual requerido para estas imágenes es de en torno a 5 minutos. En cuanto al error producido en las imágenes sin zoom, el error puede deberse a: una mayor aparición de grupos de rodaballos lo que dificulta la segmentación precisa, error humano cometido en el etiquetado manual debido a la dificultad o a la segmentación de partes del tanque.

Los resultados presentados corresponden a los parámetros de configuración de los procesos de segmentación y postprocesamiento analizados en los apartados 4.4.2 y 4.5.1, respectivamente. Los investigadores, gracias a la interfaz presentada, pueden estudiar las diferentes salidas del modelo para variaciones de estos parámetros y adaptar la aplicación para otro tipo de imágenes de entrada.

Aunque la aplicación logra estimar adecuadamente el número de rodaballos, no alcanza la precisión de la segmentación manual que establece el centroide exacto de cada pez. Por lo tanto, cuando se requiera una segmentación completamente precisa de las imágenes, se podrá integrar la salida del modelo usando el archivo generado a través de la opción de exportación “CSV Puntos” (analizado en el punto 4.6) con la aplicación de etiquetado manual mencionada en el punto 4.1.2. Así, la aplicación de etiquetado manual en Python abrirá la imagen utilizando el CSV generado por el modelo, permitiendo añadir o eliminar manualmente los puntos establecidos. Esto reducirá significativamente el tiempo inicial de etiquetado.

8.1 Trabajos futuros

Gracias a la facilidad de integración del modelo en Python y a los datos exportados, se pueden prever futuras integraciones en la aplicación. Esto permitirá optimizar su funcionalidad y mejorar los procesos de análisis de datos de manera más eficiente.

1. Entre los atributos de todas las máscaras exportadas por la aplicación se incluye el área, medida en píxeles contenidos dentro de cada máscara. Para la exportación del histograma, estas máscaras se ordenan según su tamaño. Esto permite estimar el tamaño promedio de los objetos, lo que facilita calcular el tamaño medio de los rodaballos. Así, se puede detectar cuando el modelo segmenta grupos en lugar de individuos, mejorando el proceso de segmentación y conteo.

2. Como se detalló en el apartado 4.4.1, actualmente se utiliza la función *SamAutomaticMaskGenerator* para segmentar todos los objetos según los parámetros de entrada. Sería beneficioso integrar la funcionalidad de *SAMPredictor* y utilizar los puntos de segmentación o *bounding boxes* vistos en el apartado 4.2.1. Esto permitiría corregir las segmentaciones incorrectas directamente en la interfaz de Napari de forma manual después del proceso automático.

3. Se puede integrar el modelo SAM original junto con sus diferentes tamaños de *image encoders* analizados en el punto 4.2.2, además del modelo Mobile SAM actual, y ofrecer al usuario la opción de elegir qué modelo utilizar. Esto permitirá comparar y estudiar las salidas de segmentación de los diferentes modelos en equipos con recursos computacionales elevados.

9. Referencias

[1] Amy McKeever & National Geographic Staff, «How overfishing threatens the world's oceans,» 07 02 2022. [En línea]. Disponible: https://www.nationalgeographic.es/animales/la-sobrepesca. [Ultimo acceso: 04 06 2024]

[2] Andrea Márquez, «Sobrepesca: qué es, causas, consecuencias y soluciones,» 17 12 2020. [En línea]. Disponible: https://www.ecologiaverde.com/sobrepesca-que-es-causas-consecuencias-y-soluciones. [Ultimo acceso: 05 06 2024]

[3] Hannah Farrow, «What is aquaculture?,» 14 04 2023. [En línea]. Disponible: https://www.nationalgeographic.com/environment/article/aquaculture-explainer-seaweed-fish-benefits-challenges. [Ultimo acceso: 05 06 2024]

[4] FAO, «Versión resumida de El estado mundial de la pesca y la acuicultura 2024. La transformación azul en acción.». Roma. 2024

[5] Asociación de empresas de acuicultura de Andalucía, «Las fases de la acuicultura,» 2021. [En línea]. Disponible: https://acuiculturaparatodos.com/que-es-la-acuicultura/acuicultura-marina/las-fases-de-la-acuicultura/. [Ultimo acceso: 07 06 2024]

[6] Alvaro Saavedra, «INTELIGENCIA ARTIFICIAL: MACHINE & DEEP LEARNING,» 10 09 2018. [En línea]. Disponible: https://www.privatewallmag.com/inteligencia-artificial-machine-deep-learning/. [Ultimo acceso: 14 06 2024]

[7] SAS, «Aprendizaje automático: Qué es y por qué es importante,» 2023. [En línea]. Disponible: https://www.sas.com/es_es/insights/analytics/machine-learning.html. [Ultimo acceso: 12 06 2024]

[8] DataScientest, «Machine Learning: definición, funcionamiento, usos,» 13 12 2021. [En línea]. Disponible: https://datascientest.com/es/machine-learning-definicion-funcionamiento-usos. [Ultimo acceso: 12 06 2024]

[9] José Antonio Sánchez, «¿Cómo aprenden las máquinas? Machine Learning y sus diferentes tipos,» 31 08 2020. [En línea]. Disponible: https://datos.gob.es/es/blog/como-aprenden-las-maquinas-machine-learning-y-sus-diferentes-tipos. [Ultimo acceso: 13 06 2024]

[10] Rafael Martínez Quiles, «Explorando el Zero Shot Learning: Una innovadora metodología en el aprendizaje automático,» 29 05 2023. [En línea]. Disponible: https://blogs.upm.es/pasd/2023/05/29/explorando-el-zero-shot-learning-una-innovadora-metodologia-en-el-aprendizaje-automatico/. [Ultimo acceso: 20 06 2024]

[11] Laura Rodríguez, «Algoritmos de Machine Learning,» 23 12 2022. [En línea]. Disponible: https://blog.damavis.com/algoritmos-de-machine-learning/. [Ultimo acceso: 16 06 2024]

[12] Datademia, «¿Qué es Deep Learning y qué es una red neuronal?,» 2021. [En línea]. Disponible: https://datademia.es/blog/que-es-deep-learning-y-que-es-una-red-neuronal. [Ultimo acceso: 15 06 2024]

[13] Pablo Huet, «¿Qué son las redes neuronales y sus aplicaciones?,» 13 04 2023. [En línea]. Disponible: https://openwebinars.net/blog/que-son-las-redes-neuronales-y-sus-aplicaciones/. [Ultimo acceso: 16 06 2024]

[14] Zoumana Kelta, «Introducción a las redes neuronales convolucionales (CNNs),» 04 2024. [En línea]. Disponible: https://www.datacamp.com/es/tutorial/introduction-to-convolutional-neural-networks-cnns. [Ultimo acceso: 18 06 2024]

[15] Diego Calvo, «Red Neuronal Convolucional (CNN),» 07 2017. [En línea]. Disponible: https://www.diegocalvo.es/red-neuronal-convolucional/. [Ultimo acceso: 21 06 2024]

[16] Jordi de la Torre, «REDES GENERATIVAS ADVERSARIAS (*GANs*). FUNDAMENTOS TEÓRICOS Y APLICACIONES,» Universitat Oberta de Catalunya. 21 02 2023

[17] Jordi Torres, «Generative Adversarial Networks,» 15 09 2019. [En línea]. Disponible: https://torres.ai/generative-adversarial-networks/. [Ultimo acceso: 22 06 2024]

[18] Fineproxy, «ViT (Transformador de visión),» 2022. [En línea]. Disponible: https://fineproxy.org/es/wiki/vit-vision-transformer/. [Ultimo acceso: 21 06 2024]

[19] Javier del Pino, «Vision Transformers: Transformers aplicados a imágenes,» 23 12 2023. [En línea]. Disponible: https://www.elladodelmal.com/2023/12/vision-transformers-transformers.html. [Ultimo acceso: 22 06 2024]

[20] Belén Ryba Maciejewska, «Exploración de *Vision Transformer* para clasificación de células normalesde sangre periférica,» pp 20-23, 2021.

[21] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby N, «An image is worth 16x16 words: Transformers for image recognition at scale,» *Google Research, Brain Team*. 03 06 2021

[22] Adrián Hernández Rodríguez, «Introducción a los mecanismos de atención,» 26 05 2019. [En línea]. Disponible: https://mlearninglab.com/2019/05/26/introduccion-a-los-mecanismos-de-atencion/. [Ultimo acceso: 22 06 2024]

[23] IBM, «¿Qué es la segmentación semántica?,». [En línea]. Disponible: https://www.ibm.com/es-es/topics/semantic-segmentation. [Ultimo acceso: 23 06 2024]

[24] Aula21, «Python: qué es, para qué sirve y cómo se programa,». [En línea]. Disponible: https://www.cursosaula21.com/que-es-python/. [Ultimo acceso: 23 06 2024]

[25] Javier Jiménez, «Científico de datos: así es y así se forma uno en esta profesión cada vez más demandada,». 10 2020 [En línea]. Disponible: https://www.xataka.com/otros/cientifico-datos-asi-profesion-demandada. [Ultimo acceso: 24 06 2024]

[26] Developer Resource Center, «Qué es PyTorch: una guía completa,». 04 05 2022 [En línea]. Disponible: https://developer.oracle.com/es/learn/technical-articles/what-is-pytorch. [Ultimo acceso: 23 06 2024]

[27] Observatorio Español de Acuicultura, «Cultivo del rodaballo,» [En línea]. Available: https://www.observatorioacuicultura.es/sites/default/files/images/adjuntos/libros/04_cuaderno_rodaballo.pdf. [Último acceso: 24 06 2024].

[28] S.González.Barquín, *Turbot Classify Aplication in Matlab*, Madrid: ETSIST, 2023.

[29] S.González.Barquín, *Turbot Classify Aplication Python*, Madrid: ETSIST, 2023.

[30] DataScientest, «U-NET : todo lo que tienes que saber sobre la red neuronal de Computer Vision,» 26 04 2022. [En línea]. Disponible: https://datascientest.com/es/u-net-lo-que-tienes-que-saber [Ultimo acceso: 24 06 2024]

[31] Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollar, P., & Girshick, R. «Segment Anything», *Meta AI Research, FAIR*. 05 04 2023

[32] *OpenAI*, «CLIP: Connecting text and images,» 05 01 2021. [En línea]. Disponible: https://openai.com/index/clip/. [Ultimo acceso: 25 06 2024]

[33] Meta AI, «Segment Anything,» 2023. [En línea]. Disponible: https://segment-anything.com/. [Ultimo acceso: 18 06 2024]

[34] ArcGis, «Segment Anything Model (SAM),» 05 2023. [En línea]. Disponible: https://doc.arcgis.com/es/pretrained-models/latest/imagery/introduction-to-segment-anything-model-sam-.htm. [Ultimo acceso: 25 06 2024]

[35] Zhang, C., Han, D., Qiao, Y., Kim, J. U., Bae, S.-H., Lee, S., & Hong, C. S. «Faster Segment Anything: Towards Lightweight SAM for Mobile Applications», *Kyung Hee University*. 01 07 2023

[36] Parez, S., Dilshad, N., Alghamdi, N. S., Alanazi, T. M., & Lee, J. W. «Visual Intelligence in Precision Agriculture: Exploring Plant Disease Detection via Efficient Vision Transformers», *Department of Software, Sejong University, Seoul, Republic of Korea*. 2023

[37] Margaret Rouse, «Destilación del conocimiento en el aprendizaje profundo,» 30 01 2024. [En línea]. Disponible: https://www.techopedia.com/es/definicion/destilacion-conocimiento. [Ultimo acceso: 25 06 2024]

[38] Rodrigo Ceron, «Inteligencia artificial hoy: datos, entrenamiento e infrerencias,» 26 12 2019. [En línea]. Disponible: https://www.ibm.com/blogs/systems/mx-es/2019/12/inteligencia-artificial-hoy-datos-entrenamiento-e-infrerencias/. [Ultimo acceso: 25 06 2024]

[39] Santillan, I., & Terol-Villalobos, I. R. «Transformación de imágenes a color a escala de grises aplicando un espacio de color coherente con la percepción visual humana, » 2008

[40] Hanzi Mao, «segment-anything,» *GitHub,* 02 05 2023. [En línea]. Disponible: https://github.com/facebookresearch/segment-anything/tree/main. [Ultimo acceso: 26 06 2024]

[41] Chaoining Zhang, «MobileSAM,» *GitHub,* 20 12 2023. [En línea]. Disponible: https://github.com/ChaoningZhang/MobileSAM/tree/master. [Ultimo acceso: 26 06 2024]

[42] Brian Northan, «tnia-python,» *GitHub,* 22 03 2024. [En línea]. Disponible: https://github.com/True-North-Intelligent-Algorithms/tnia-python/tree/cfccfb6583a2826d7d7b07ca4e5c2d6365fea06b. [Ultimo acceso: 24 06 2024]

[43] Napari Contributors, «Napari: a multi-dimensional image viewer for Python,». [En línea]. Disponible: https://napari.org/stable/. [Ultimo acceso: 27 06 2024]

[44] N. Unidas, «La Asamblea General adopta la Agenda 2030 para el Desarrollo Sostenible,» UN, 25 08 2015. [En línea]. Disponible: https://www.un.org/sustainabledevelopment/es/2015/09/la-asamblea-general-adopta-la-agenda-2030-para-el-desarrollo-sostenible/. [Último acceso: 29 06 202

Anexo 1: Funciones

En este Anexo se mostrarán las funciones pertenecientes al código Python más relevantes de la aplicación a las cuales se hace referencia en el documento indicando la clase a la que pertenecen.

A.1.1. convertRGB (Utils.py)

```
1  def convertRGB(imagen: Union[np.ndarray, None])
2      try:
3          imagenGris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
4
5          altura, anchura = imagenGris.shape
6
7          imagenRGB = np.zeros((altura, anchura, 3), dtype=np.uint8)
8
9          imagenRGB[:, :, 0] = imagenGris
10         imagenRGB[:, :, 1] = imagenGris
11         imagenRGB[:, :, 2] = imagenGris
12
13         return imagenRGB
14     except Exception:
15         raise
```

A.1.2. recortarCuadrantes (Utils.py)

```
1  def recortarCuadrantes(imagen,numCuadrantes)
2      try:
3          altura, anchura, _ = imagen.shape
4          raiz = int(np.sqrt(numCuadrantes))
5          alturaCuadrante = altura // raiz
6          anchuraCuadrante = anchura // raiz
7          cuadrantes = []
8          for i in range(raiz):
9              for j in range(raiz):
10                 inicioY = i * alturaCuadrante
11                 inicioX = j * anchuraCuadrante
12                 cuadrante = imagen[inicioY:inicioY + alturaCuadrante,
13                                    inicioX:inicioX + anchuraCuadrante]
14                 cuadrantes.append(cuadrante)
15
16         return cuadrantes
17     except Exception:
18         raise
```

A.1.3. __init__ (TurbotSAM.py)

```
def __init__(self,points_per_side,points_per_batch,pred_iou_thresh,
             stability_score_thresh,stability_score_offset,
             box_nms_thresh,crop_n_layers,crop_nms_thresh,
             crop_overlap_ratio,crop_n_points_downscale_factor,
             min_mask_region_area):

    try:
        self.checkpoint = "models/mobile_sam.pt"
        self.modelType = "vit_t"
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
        self.sam =sam_model_registry[self.modelType](self.checkpoint)
        self.sam.to(device=self.device)
        self.sam.eval()

        self.generadorMascaras = SamAutomaticMaskGenerator(
            model = self.sam,
            points_per_side = points_per_side,
            points_per_batch = points_per_batch,
            pred_iou_thresh = pred_iou_thresh,
            stability_score_thresh = stability_score_thresh,
            stability_score_offset = stability_score_offset,
            box_nms_thresh = box_nms_thresh,
            crop_n_layers = crop_n_layers,
            crop_nms_thresh = crop_nms_thresh,
            crop_overlap_ratio = crop_overlap_ratio,
            crop_n_points_downscale_factor =crop_n_points_downscale,
            min_mask_region_area = min_mask_region_area,
        )
    except Exception:
        raise
```

A.1.4. generarMascarasPorCuadrante (TurbotSAM.py)

```
def generarMascarasPorCuadrante(self, cuadrantes: List[np.ndarray],
                                postprocesamiento: bool)
    try:
        mascarasPorCuadrante = []
        numCuadrantes = len(cuadrantes)
        aux = 90 if not postprocesamiento else 50
        cont = 0

        for cuadrante in cuadrantes:
            cont += 1
            masks = self.generarMascaras(cuadrante)
            mascarasPorCuadrante.append(masks)
            porcentaje = (cont / numCuadrantes) * aux
            yield porcentaje, mascarasPorCuadrante, cont
    except Exception:
        raise
```

A.1.5. superponerMascaras (ProcesarMascaras.py)

```python
def superponerMascaras(mascarasPorCuadrante: List[List[Dict]],
                       dimensiones: Tuple[int, int])

    try:
        alturaTotal, anchuraTotal = dimensiones
        raiz = int(np.sqrt(len(mascarasPorCuadrante)))
        alturaCuadrante = alturaTotal // raiz
        anchuraCuadrante = anchuraTotal // raiz

        mascarasSuperpuestas = []

        idx = 0
        for i in range(raiz):
            for j in range(raiz):
                mascaras = mascarasPorCuadrante[idx]

                for mascara in mascaras:
                    segmentacion = mascara['segmentation']
                    nuevaSegmentacion = np.zeros((alturaTotal,
                                                  anchuraTotal),
                                                  dtype=bool)
                    y, x = np.where(segmentacion)

                    y += i * alturaCuadrante
                    x += j * anchuraCuadrante

                    nuevaSegmentacion[y, x] = True

                    mascaraSuperpuesta = {
                        'segmentation': nuevaSegmentacion,
                        'area': mascara['area'],
                        'bbox': mascara['bbox'],
                        'predicted_iou': mascara['predicted_iou'],
                        'point_coords': mascara['point_coords'],
                        'stability_score': mascara['stability_score'],
                        'crop_box': mascara['crop_box']
                    }
                    mascarasSuperpuestas.append(mascaraSuperpuesta)

                idx += 1

        return mascarasSuperpuestas
    except Exception:
        raise
```

A.1.6. pintarCentroidesMascaras (ProcesarMascaras.py)

```
def pintarCentroidesMascaras(mascaras: List[Dict[str, any]])
    try:

        altura, anchura = mascaras[0]['segmentation'].shape
        img = np.zeros((altura, anchura), dtype=np.uint8)

        centroides = []

        for mascara in mascaras:
            centroide = center_of_mass(mascara['segmentation'])


            centroideY, centroideX = map(int, centroide)
            cv2.circle(img, (centroideX, centroideY), 3,
                       (255, 255, 255), -1)

            centroides.append((centroideX, centroideY))

        return img, centroides
    except Exception:
        raise
```

A.1.7. mostrarLabels (ProcesarMascaras.py)

```
def mostrarLabels(mascaras: List[Dict[str, any]])
    try:
        if len(mascaras) == 0:
            return None

        listaMascaras = []

        for ann in mascaras:
            segmentation = ann['segmentation']

            mascara = np.zeros((segmentation.shape[0],
                                segmentation.shape[1]),
                               dtype=np.uint8)

            mascara[segmentation] = 1

            listaMascaras.append(mascara)

        labels = np.zeros_like(listaMascaras[0],
                                    dtype=np.uint32)
        for i, mascara in enumerate(listaMascaras, start=1):
            labels[mascara == 1] = i

        return labels
    except Exception:
        raise
```

A.1.8. procesarMascaras (ProcesarMascaras.py)

```python
def procesarMascaras(imagenEtiquetada: np.ndarray,
                     mascarasInfo: List[Dict[str, any]],
                     imagenOriginal: np.ndarray, min_size: int,
                     max_size: int, min_intensity: int)

    try:
        mascarasFiltradas = []
        ordenarMascaras = sorted(mascarasInfo,
                                 key=lambda x: x['area'],
                                 reverse=True)

        for enum, mascaraInfo in enumerate(ordenarMascaras):
            segmentation = mascaraInfo['segmentation']
            area = mascaraInfo['area']
            mnarray = np.array(segmentation, dtype=bool)
            mnarray[imagenOriginal < min_intensity] = False
            pixels = imagenOriginal[mnarray]

            if pixels.size == 0:
                continue

            mnarray = morphology.opening(mnarray,
                                         morphology.disk(3))

            if area > min_size and area < max_size:
                imagenEtiquetada[mnarray] = enum + 1
                mascarasFiltradas.append(mascaraInfo)

        return imagenEtiquetada, mascarasFiltradas
    except Exception:
        raise
```

A.1.9. __iniciarSegmentacionThread (NapariSAM.py)

```python
def __iniciarSegmentacionThread(self) -> None:

    try:
        thread = threading.Thread(target=
                                  self.__iniciarSegmentacion)
        thread.start()
    except Exception as e:
        self.log.append(f"<span style='color: red;'>
                        [ERROR]</span> Ha ocurrido un error
                        al generar un hilo: {str(e)}")
```

A.1.10.__cargarMascaras (NapariSAM.py)

```
1  def __cargarMascaras(self) -> None:
2      try:
3          if self.mascarasGeneradas is not None:
4              self.__agregarLabel(self.mascarasGeneradas,
5                                  "Mascaras SAM")
6              self.mascarasGeneradas  = None
7
8          elif self.puntosGenerados is not None:
9              self.__agregarLabel(self.puntosGenerados,
10                     "Centros de Mascaras SAM")
11             self.puntosGenerados  = None
12
13         elif self.mascarasProcesadas is not None:
14              self.__agregarLabel(self.mascarasProcesadas,
15                     "Mascaras Post Procesamiento")
16              self.mascarasProcesadas  = None
17
18         elif self.puntosProcesados is not None:
19              self.__agregarLabel(self.puntosProcesados,
20                     "Centros de Mascaras Post Procesamiento")
21              self.puntosProcesados  = None
22
23         else:
24             pass
25
26         self.__actualizarBarraProgreso(self.porcentajeProgreso)
27     except Exception as e:
28         self.log.append(f"<span style='color: red;'>
29                         [ERROR]</span>Ha ocurrido un
30                         error al agregar las mascaras
31                         o puntos generados/procesados
32                         al visor: {str(e)}")
```

Anexo 2: Manuales de usuario

A.2.1 Manual de instalación

En este anexo se detallarán los pasos para poder hacer uso de la aplicación, para ello hay que tener en cuenta:

a) Se requiere tener instalado en el sistema *Python v3.8*.
b) Es necesario que el sistema operativo sea *Windows*.
c) Es necesario tener instalado *GIT* para poder clonar el repositorio pertinente.
d) Para poder hacer uso de la *GPU* es necesario que nuestro dispositivo cuente con una tarjeta gráfica *NVIDIA* de lo contrario se usará la *CPU*.
e) Se recomienda la instalación del entorno de desarrollo *Visual Studio Code.*

A continuación, se detallan los pasos a seguir:

1) Clonación del repositorio

Primeramente, debemos solicitar el acceso al equipo de GAMMA al repositorio de git *TURBOTSAM.*

Clonamos el repositorio usando el comando: *git clone https://github.com/IvanViedma/TURBOTSAM.git*

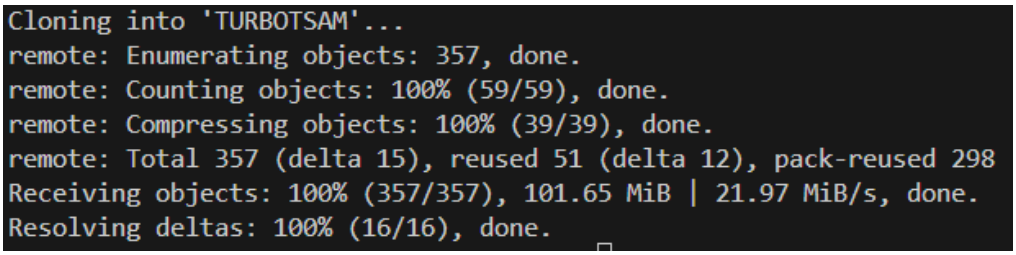

Figura 60 – Clonación de repositorio

Una vez clonado deberíamos ver el siguiente contenido:

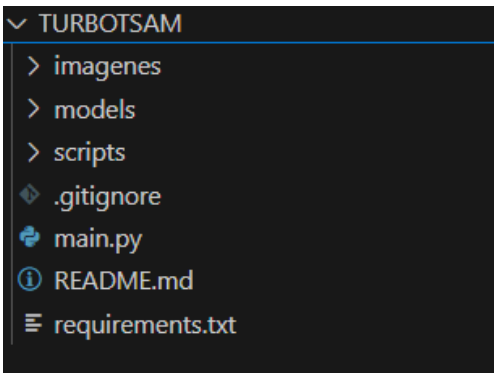

Figura 61 – Raíz del repositorio

2) Creación de un entorno virtual

En nuestro caso utilizaremos *Virtual Studio Code* para la creación del entorno, para ello en el buscador ejecutamos *Create Environment.*

Figura 62 – Creación del entorno (1)

Seleccionamos *Python 3.8, Venv* y elegimos la opción *requirements,* damos en *Aceptar*:

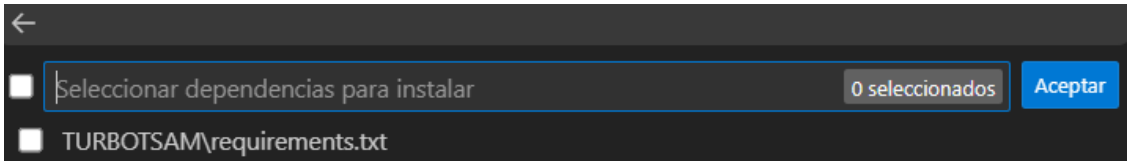

Figura 63 – Creación del entorno (2)

Una vez hecho esto habremos creado el entorno principal e instalado la mayoría de las dependencias. Debemos de asegurarnos de que el entorno esté activado (ver Figura 64).

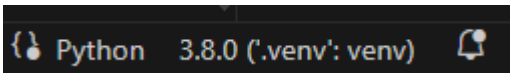

Figura 64 – Creación del entorno (3)

3) Instalación de librerías *PyTorch* sin compatibilidad *CUDA*

Si no disponemos de tarjeta gráfica *NVIDEA* debemos seguir estas instrucciones, de lo contrario avanzaremos hasta el punto 4.

Instalamos las librerías relacionadas con *pytorch*. Para ello ejecutamos el comando *pip install timm.* Ahora si ejecutamos el archivo *main.py* la aplicación se lanzará y podremos usarla, pero veremos el siguiente mensaje en el cuadro de salida:

Figura 65 – *CUDA* no disponible

Esto quiere decir que no se usará la memoria *GPU*, deberemos quedarnos en este punto si nuestro sistema no tiene una tarjeta gráfica *NVIDEA* y por lo tanto se usará la *CPU* para el procesamiento lo que llevará a tiempos muchos mas elevados de segmentación.

4) Instalación librerías *PyTorch* con compatibilidad *CUDA*

Si nuestro sistema si tiene una tarjeta gráfica *NVIDEA* podremos instalar *pytorch* con compatibilidad *CUDA* para hacer uso de la memoria *GPU*. Para ello, deberemos irnos a la web: https://pytorch.org/

Figura 66 – Selección de comando PyTorch

En esta sección deberemos de elegir nuestra configuración, en nuestro caso el sistema operativo es *Windows*, vamos a instalarlo haciendo uso de *pip* con *Python* y nuestra versión de *CUDA* es 12.1 (esto se comprueba en las propiedades de nuestra tarjeta gráfica). Por lo tanto, el comando que ejecutaremos en nuestro entorno virtual será: *pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121*

Por último debemos ejecutar el comando *pip install timm*

Al lanzar la aplicación ejecutando *main.py* ya podremos hacer uso de la *GPU* (ver Figura 67) y la aplicación estará lista para usarse:

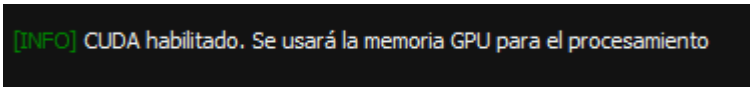

Figura 67 – *CUDA* disponible

A.2.2 Manual de uso

En este anexo se explicará el uso general de la aplicación.

1) Cargamos la imagen haciendo uso del *widget CARGAR IMAGEN.*
2) Seleccionamos el tipo de imagen: con zoom o sin zoom.
3) Seleccionamos el numero de cuadrantes en caso de querer hacer segmentación por bloques, en caso contrario, lo dejamos por defecto en *Sin cuadrantes.*
4) Si queremos visualizar mejor los cuadrantes seleccionamos la opción *Mostrar sectores.*
5) Si recomienda dejar los parámetros de segmentación por defecto, pero si se quieren modificar se debe comprender en que afecta su modificación en el proceso de segmentación.
6) Si se desea filtrar mejor la salida de la segmentación podremos activar la opción *Activar postprocesamiento* y editar sus parámetros o dejarlos por defecto.
7) Antes de iniciar la segmentación si se sabe con antelación el número de rodaballos presentes en la imagen podremos indicarlo en el *widget Número de rodaballos a comparar* para poder obtener los errores producidos.
8) A medida que se vayan generando la imagen de máscaras y los centroides irán apareciendo en el visor de *Napari* (ver Figura 48).
9) Los resultados se mostrarán en el cuadro de salida (ver Figura 58).
10) Una vez finalizado el proceso que podemos seguir en la barra de progreso podremos exportar los resultados explicados en el punto 7 del apartado 4.6.