UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Diseño, desarrollo y validación de un sistema de visión basado en redes neuronales para automatización de ensayos con C. elegans

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Peñaranda Jara, José Julio

Tutor/a: Sánchez Salmerón, Antonio José

Director/a Experimental: Layana Castro, Pablo Emanuel

CURSO ACADÉMICO: 2023/2024

Índice

Capítulo 1. Introducción

1.1 Presentación y objetivos

En este Trabajo Final de Máster (TFM en adelante), se ha llevado a cabo el diseño, el desarrollo y la posterior validación de un sistema de visión artificial basado en redes neuronales para el seguimiento de posición de nematodos sobre una placa de Petri. Este sistema será utilizado para automatización de ensayos con Caenorhabditis elegans (C. elegans) como lifespan, healthspan, etc. Los objetivos principales del TFM son los siguientes:

a) El primer objetivo principal consiste en indicar las coordenadas x e y de cada C. elegans en la placa de Petri, de esta forma, cada vez que el sistema recibe una imagen, se hace una llamada a la función que devuelve una lista con las coordenadas de cada uno de los nematodos.
b) El segundo objetivo principal es la identificación de los C. elegans, asignando un identificador a cada uno y tratando de no perder la identificación de este a lo largo de una secuencia de imágenes. Aun cuando los nematodos pueden llegar a cruzarse, los identificadores de los nematodos no deben cruzarse.

Estos objetivos se dividen en varios subobjetivos:

- El sistema debe utilizar datos de la secuencia de imágenes anteriores a la actual para realizar la correcta identificación
- El sistema debe poder procesar cada imagen en menos de un segundo
- El sistema debe ofrecer una forma de guardar imágenes que muestren el resultado del seguimiento de los C. elegans
- La detección de los C. elegans debe hacerse mediante una red neuronal
- Para el caso de las agregaciones de nematodos se debe utilizar una red neuronal adicional
- Las redes neuronales del sistema deben entrenarse con imágenes simuladas de C. elegans sobre fondos predefinidos
- La validación tanto de las redes neuronales como de los resultados del programa debe realizarse con imágenes reales
- El sistema debe poder integrarse en un nodo de ROS2 para poder enviarle imágenes continuamente en tiempo real

1.2 Estructura de la memoria

Esta memoria está organizada de la siguiente forma:

- En el capítulo 2, Estado del arte, se presenta una investigación exhaustiva de trabajos, proyectos y productos relacionados con el realizado en este TFM, comparándolos, con el fin de identificar las novedades y características únicas del TFM.
- En el capítulo 3, Herramientas Hardware y Software, se explican todas las herramientas, dispositivos y herramientas necesarias para entender el trabajo realizado en el TFM, descrito en los capítulos siguientes.
- En el capítulo 4, Desarrollo del proyecto, se describe en detalle el funcionamiento, las características y la programación realizada en su mayoría en Python para este TFM, recogiendo todas las características implementadas en el código y sus funcionalidades. Además, también se analiza el proceso de entrenamiento, la validación de los resultados y la implementación de las redes neuronales en el proyecto. Este capítulo se divide en 3 partes:
    - 4.1) Entrenamiento, uso y validación de redes neuronales, en la que se describe el proceso de entrenamiento utilizando imágenes creadas y etiquetadas mediante el uso de un simulador, la validación de estas redes neuronales y la implementación de éstas en el programa principal.
    - 4.2) Programación en Python, en la que se describe el proceso de visión artificial y seguimiento de cada C. elegans mediante el uso de OpenCV en Python.
    - 4.3) Validación de resultados, en la que se describe todo el proceso de validación de resultados de las redes neuronales y de los resultados finales del sistema.
- En el capítulo 5, Conclusiones, se presentan las conclusiones y posibles ampliaciones, así como otras mejoras que se pueden implementar en el proyecto.
- La memoria finaliza con la sección de Bibliografía, que incluye las referencias bibliográficas utilizadas durante la realización del TFM.

Capítulo 2. Estado del arte

En este capítulo se describen diversos temas relacionados con el TFM. Estos temas son, en primer lugar, el concepto de visión artificial, tanto de manera conceptual como diferentes tecnologías y programas utilizados en este ámbito. En este apartado se indagará también diferentes técnicas de visión artificial, sus diferencias y ventajas.

En segundo lugar, se presenta una descripción del concepto de Red Neuronal, así como conceptos como neuronal o inteligencia artificial. También se hará un repaso de los procesos de entrenamiento y validación de redes neuronales, además de explicar brevemente diversos tipos de redes neuronales, entre ellas las redes YOLO Segmentation y U-Net, que son las que se utilizarán en este proyecto.

Además, se muestra un trabajo de investigación que compara este TFM con diversos proyectos relacionados con el tracking de C. elegans, resaltando los puntos fuertes de cada uno y las diferencias entre ellos.

2.1 C.elegans

Los Caenorhabditis elegans, comúnmente abreviados como C. elegans, son un organismo modelo increíblemente versátil y valioso en la investigación biológica. Estos diminutos nematodos han sido objeto de estudio durante décadas debido a su simplicidad estructural y genética, combinada con una complejidad funcional sorprendente.

Un aspecto notable de los C. elegans es su anatomía extremadamente conocida y su sistema nervioso de apenas 302 neuronas, lo que permite un análisis detallado de las conexiones neuronales y del comportamiento. Esta característica única ha hecho que los C. elegans sean fundamentales en la investigación neurocientífica, proporcionando una plataforma para estudiar desde los mecanismos básicos del sistema nervioso hasta las complejas interacciones neurales.

Además de su relevancia en neurociencia, los C. elegans han sido utilizados en una amplia gama de campos de investigación debido a su genoma completamente secuenciado y bien caracterizado. Esto ha permitido estudiar los efectos de los genes individuales y su relación con una variedad de procesos biológicos, desde el envejecimiento hasta la respuesta inmune.

Los C. elegans poseen una cubierta cuticular que recubre todo su cuerpo, proporcionándoles protección mecánica y regulación del intercambio gaseoso y de agua con el entorno. Esta cutícula está compuesta principalmente por colágeno, lo que le confiere propiedades estructurales similares a las de la piel humana.

Aunque la cutícula de los C. elegans no es análoga a la piel humana en términos de funciones específicas como la termorregulación o la sensación táctil, su composición y función básica guardan similitudes. Ambas actúan como una barrera física que protege al organismo contra agresiones externas, como patógenos o daños mecánicos, y también desempeñan un papel en la regulación del equilibrio hídrico y la homeostasis interna.

Esta similitud estructural entre la cutícula de C. elegans y la piel humana ha permitido que los estudios con este organismo modelo proporcionen información relevante sobre procesos biológicos relacionados con la integridad cutánea y la respuesta a lesiones en organismos más complejos. Además, la capacidad de manipular genéticamente a los C.

elegans ha facilitado la investigación de genes y vías metabólicas implicadas en la formación y mantenimiento de la cutícula, lo que puede tener implicaciones en la comprensión y tratamiento de enfermedades cutáneas en humanos.

Los experimentos con C. elegans abarcan diversas áreas de investigación científica. En el ámbito de la salud, se realizan estudios de HealthSpan para evaluar la calidad de vida de los nematodos a lo largo del tiempo, observando aspectos como su capacidad de movimiento, reproducción y resistencia al estrés.

Por otro lado, los experimentos de Lifespan se enfocan en determinar la duración de la vida de estos organismos bajo diferentes condiciones experimentales, lo que proporciona información valiosa sobre los factores que influyen en su longevidad y los mecanismos subyacentes al envejecimiento.

Además, los C. elegans son utilizados en investigaciones de toxicología y farmacología, donde se evalúa la respuesta de los nematodos a sustancias químicas y compuestos farmacológicos para determinar su toxicidad y posibles efectos terapéuticos. En el campo de la genética y la biología del desarrollo, estos nematodos son un valioso recurso para estudiar la función de genes específicos y entender procesos como el desarrollo embrionario y la diferenciación celular.

Hay que tener en cuenta que, al ser un organismo modelo, los resultados obtenidos con C. elegans suelen ser extrapolables a otros organismos más complejos, incluidos los humanos. Esto hace que los estudios con C. elegans sean particularmente relevantes en el desarrollo de nuevas terapias y tratamientos médicos.

En el ámbito industrial, un sistema que permita la automatización de ensayos con C. elegans ofrece varias ventajas. Por un lado, permite la realización de ensayos de alto rendimiento de manera eficiente y precisa, lo que acelera el proceso de descubrimiento de fármacos y compuestos biológicamente activos.

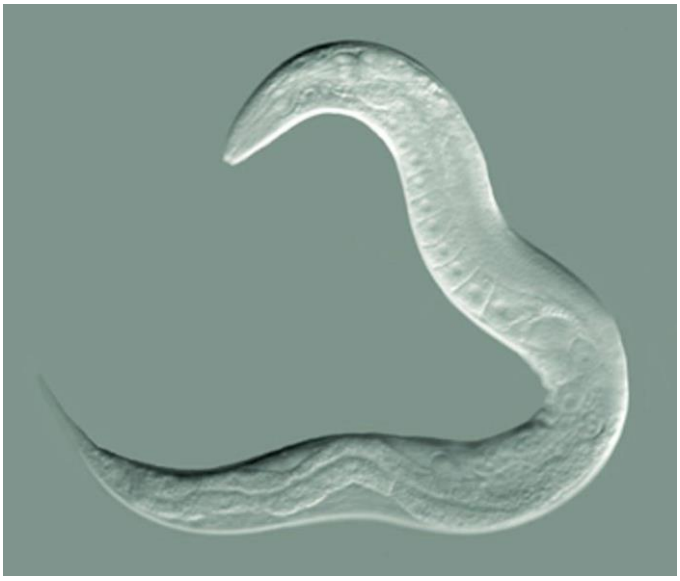

Figura 2.1.1: Imagen de un Caenorhabditis elegans [Ref 1]

2.2 Visión Artificial

La visión artificial es una disciplina que combina conceptos de inteligencia artificial, procesamiento de imágenes, y reconocimiento de patrones para emular la capacidad visual humana en sistemas computarizados. Su aplicación abarca una amplia gama de campos, desde el control de calidad industrial hasta la medicina y la robótica.

La visión artificial se basa en el procesamiento de imágenes, el reconocimiento de patrones y el aprendizaje automático para emular la capacidad visual humana en sistemas computarizados. Estos conceptos son fundamentales para entender cómo se abordan los desafíos de seguimiento de objetos en entornos complejos.

Procesamiento de imágenes

El procesamiento de imágenes es el núcleo de la visión artificial. Implica la manipulación de imágenes digitales para mejorar su calidad, extraer información relevante o realizar tareas específicas. Esto puede incluir filtrado, segmentación, detección de bordes y reconocimiento de objetos.

Diversas herramientas utilizadas en programación para el procesamiento de imagen son, por un lado, OpenCV, una biblioteca de código abierto ampliamente utilizada que proporciona una amplia gama de funciones para el análisis y procesamiento de imágenes. Por otro lado, está MATLAB, que ofrece herramientas específicas para el procesamiento de imágenes y análisis de datos. Otra herramienta importante en este ámbito es scikit-image, una biblioteca de procesamiento de imágenes de Python que proporciona una colección de algoritmos para manipular y analizar imágenes digitales. Estas herramientas ofrecen funcionalidades avanzadas para abordar una variedad de desafíos en el procesamiento de imágenes en aplicaciones de visión artificial.

El procesamiento de imágenes implica una serie de pasos y técnicas que permiten transformar datos de entrada visual en información significativa para su posterior análisis y toma de decisiones.

Preprocesamiento de Imágenes:

El preprocesamiento de imágenes es el primer paso en el procesamiento de imágenes y se utiliza para preparar las imágenes para su análisis posterior. Esto puede incluir operaciones como la corrección de la iluminación, el ajuste del contraste, la eliminación de ruido y la normalización de la escala de grises. El objetivo es mejorar la calidad de la imagen y resaltar las características de interés para un análisis más efectivo.

Filtrado:

El filtrado se utiliza para resaltar o suprimir ciertas características en una imagen mediante la aplicación de filtros o máscaras. Esto puede incluir filtros de suavizado (por ejemplo, filtro Gaussiano), filtros de realce de bordes (por ejemplo, filtro Sobel) y filtros de detección de características (por ejemplo, filtro Laplaciano). Los filtros se aplican convolucionando la imagen de entrada con un núcleo o máscara específica para resaltar ciertas características de interés.

Segmentación:

Este proceso implica dividir una imagen en regiones o segmentos significativos basados en ciertas características, como el color, la textura o la intensidad de los píxeles. Esto puede utilizarse para identificar objetos individuales en una imagen o separar el objeto de interés del fondo. Las técnicas comunes de segmentación incluyen umbrales, crecimiento de regiones, segmentación por contornos activos y segmentación por clustering.

Transformaciones Geométricas:

Las transformaciones geométricas se utilizan para modificar la geometría espacial de una imagen, como el escalado, la rotación, la traslación y la deformación. Estas transformaciones son útiles para corregir la perspectiva, alinear imágenes, o transformarlas a un espacio de coordenadas específico. Las transformaciones geométricas pueden ser lineales o no lineales, y se pueden aplicar de manera local o global a la imagen.

Aplicaciones

Diversas aplicaciones del procesamiento de imágenes van desde la corrección de imágenes para mejorar su calidad, brillo u otra característica hasta la umbralización de imágenes para obtener máscaras de imagen que pueden ser utilizadas en el reconocimiento de patrones.

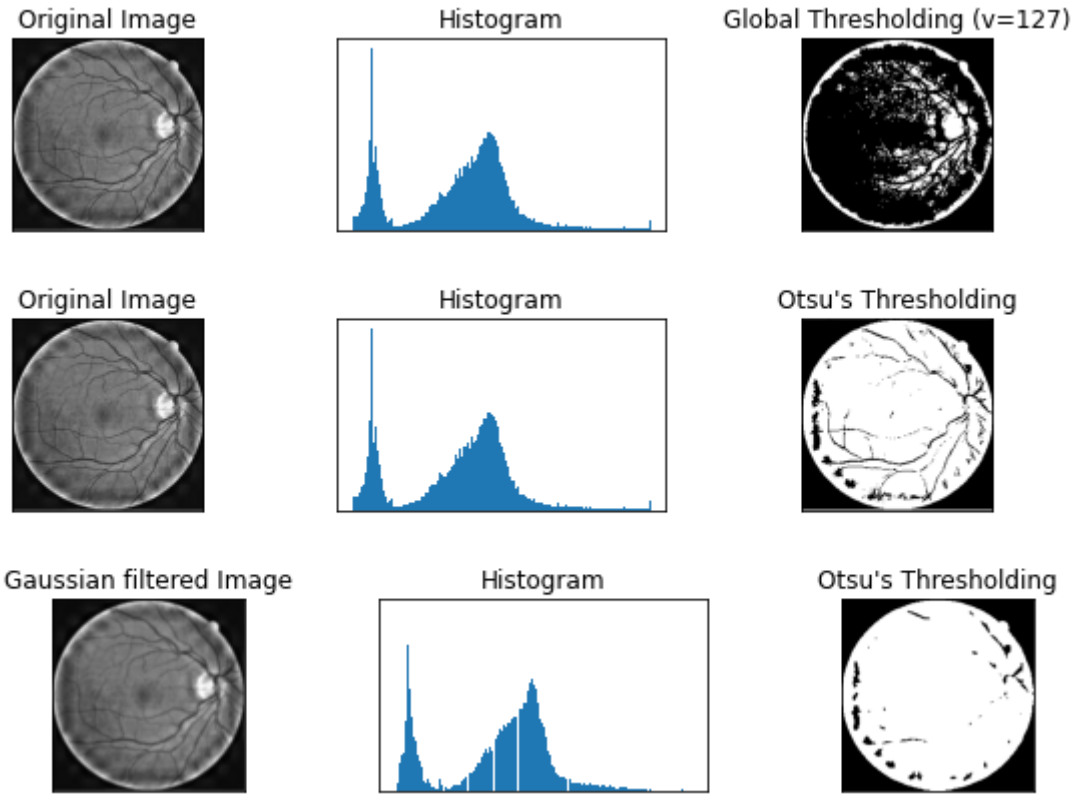

Figura 2.2.1: Ejemplos de imágenes umbralizadas [Ref 3]

Reconocimiento de patrones

El reconocimiento de patrones es crucial para interpretar la información visual. Se refiere a la identificación de características distintivas en los datos de entrada y su asociación con clases o categorías predefinidas, lo cual implica reconocer objetos, formas, texturas u otras características visuales en una imagen.

El reconocimiento de patrones implica la extracción de características relevantes de los datos, así como la identificación de patrones o estructuras en la imagen. Estos datos pueden ser en la etapa de clasificación, para poder reconocer los diferentes objetos en una imagen y asignarles una clase.

Extracción de Características

El reconocimiento de patrones se basa en una variedad de técnicas y algoritmos, un método básico en este ámbito es la extracción de características, la cual implica la identificación y extracción de características relevantes de los datos de entrada que son útiles para la clasificación o identificación. Estas características pueden ser locales, como bordes, esquinas, texturas, etc. Los cuales se pueden describir utilizando descriptores como Histogramas de Gradientes Orientados, o pueden ser características invariantes a la escala.

Clasificación

Una vez que se han extraído las características, se utiliza un algoritmo de clasificación para asignar las muestras a diferentes clases o categorías. Algunos de los algoritmos de clasificación más comunes incluyen Support Vector Machines (SVM), k-Nearest Neighbors (k-NN) y Naive Bayes, además de las Redes Neuronales, las cuales usan imágenes clasificadas o “etiquetadas” para entrenar.

Aplicaciones

El reconocimiento de patrones tiene muchas y muy variadas aplicaciones que van desde la detección de rostros en imágenes hasta la clasificación de objetos en escenas complejas. Algunos ejemplos comunes de aplicaciones incluyen reconocimiento de objetos, detección de rostros, reconocimiento de texto e incluso aplicaciones de biometría.

Aprendizaje Automático

El aprendizaje automático juega un papel fundamental en la visión artificial al proporcionar herramientas y técnicas para que los sistemas puedan aprender y mejorar su desempeño a partir de los datos. En el contexto de la visión artificial, el aprendizaje automático se utiliza para una amplia gama de tareas, como la clasificación de imágenes, la detección y segmentación de objetos, el reconocimiento facial y el seguimiento de objetos en video. Entre las técnicas de aprendizaje automático más utilizadas se encuentran las redes neuronales convolucionales, que son modelos computacionales inspirados en la estructura y funcionamiento del cerebro humano.

Las redes neuronales, especialmente las redes neuronales convolucionales (CNN), son muy efectivas en tareas de visión artificial debido a su capacidad para aprender representaciones jerárquicas de las características visuales en los datos de entrada. Estas redes están compuestas por capas de neuronas artificiales organizadas en una estructura convolucional, donde cada capa aprende a detectar características simples en regiones locales de la imagen y las capas posteriores combinan estas características para formar representaciones más complejas y abstractas. Este enfoque permite a las CNN capturar patrones complejos y variados en los datos de entrada, lo que las hace especialmente adecuadas para tareas de reconocimiento de patrones en imágenes.

Además de las CNN, existen otras técnicas de aprendizaje automático que se utilizan en visión artificial, como las máquinas de vectores de soporte, los árboles de decisión, los bosques aleatorios y los métodos de clustering. Estas técnicas pueden ser utilizadas para tareas específicas, como la clasificación de imágenes, la detección de objetos o la segmentación de imágenes, dependiendo de los requisitos del problema y las características de los datos

2.3 Redes Neuronales

Descripción y funcionamiento de una red neuronal

Una red neuronal es un modelo computacional inspirado en la estructura y funcionamiento del cerebro humano, diseñado para aprender a realizar tareas específicas a partir de datos. Consiste en un conjunto interconectado de unidades de procesamiento llamadas neuronas, organizadas en capas, que procesan y transforman la información a medida que esta fluye a través de la red.

El funcionamiento básico de una red neuronal se puede dividir en tres pasos principales: entrada de datos, procesamiento y salida de resultados.

En la entrada de datos, la red neuronal recibe datos de entrada en forma de vectores numéricos, que representan características o atributos de los datos a procesar. Estos datos pueden ser imágenes, texto, señales de audio u otros tipos de información.

Durante el procesamiento, los datos de entrada son propagados a través de la red neuronal, capa por capa, mediante operaciones matemáticas. Cada capa está compuesta por múltiples neuronas, que aplican transformaciones lineales y no lineales a los datos. Las neuronas en una capa están conectadas a las neuronas de la capa siguiente a través de conexiones ponderadas, llamadas pesos, que determinan la influencia de cada neurona en la activación de las neuronas de la capa siguiente.

Una vez que los datos han pasado por todas las capas de la red, se produce la salida de resultados que representa la respuesta de la red neuronal a los datos de entrada. Dependiendo de la tarea que esté realizando la red neuronal, esta salida puede ser una clasificación de categorías, una regresión de valores numéricos, una segmentación de imágenes, entre otros.

Durante el proceso de entrenamiento, la red neuronal ajusta sus pesos en función de la diferencia entre la salida producida por la red y la salida deseada, utilizando un algoritmo de optimización como el descenso de gradiente. Este proceso de aprendizaje permite que la red neuronal mejore gradualmente su capacidad para realizar la tarea específica para la cual ha sido entrenada.

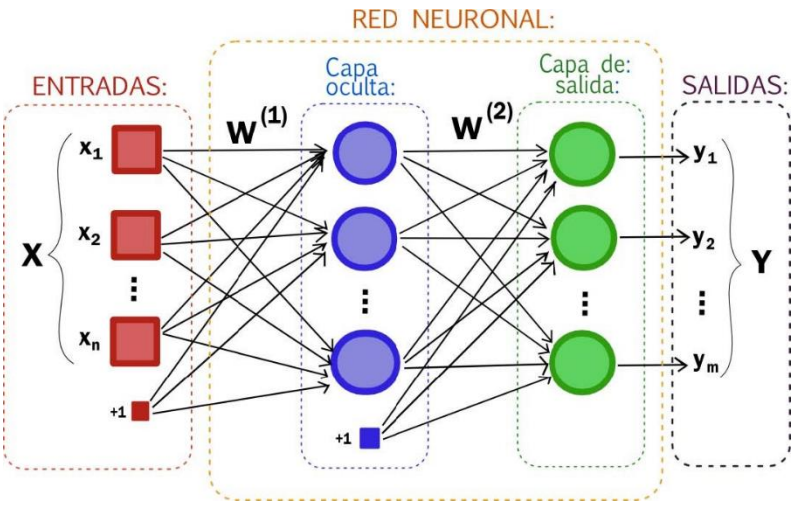

Figura 2.3.1: Componentes de una red neuronal [Ref 4]

En el contexto del procesamiento de imágenes, una red neuronal es un modelo computacional diseñado para aprender patrones y características visuales a partir de imágenes. Este tipo de red neuronal, comúnmente conocida como red neuronal convolucional (CNN), es especialmente efectiva en tareas de reconocimiento, clasificación, detección y segmentación de objetos en imágenes.

El funcionamiento de una red neuronal convolucional en el procesamiento de imágenes implica varias capas de procesamiento, cada una de las cuales realiza operaciones específicas para extraer y transformar información visual:

Capas de convolución: Estas capas aplican filtros convolucionales a la imagen de entrada para detectar características simples, como bordes, texturas y patrones básicos. Cada filtro convolucional escanea la imagen y produce un mapa de características que resalta la presencia de la característica detectada en diferentes regiones de la imagen.

Capas de activación: Después de la convolución, se aplican funciones de activación, como ReLU (Rectified Linear Unit), para introducir no linealidades en el modelo y permitir la detección de patrones más complejos en los datos.

Capas de pooling: Estas capas reducen la dimensionalidad de los mapas de características mediante operaciones como el max-pooling o el average-pooling, lo que ayuda a disminuir el costo computacional y a mejorar la invarianza a pequeñas variaciones en la posición de las características en la imagen.

Capas completamente conectadas: En la parte final de la red, se utilizan capas completamente conectadas para combinar las características extraídas en una representación final y realizar la clasificación o regresión de la imagen.

Durante el entrenamiento de una red neuronal convolucional para el procesamiento de imágenes, los pesos de los filtros convolucionales y las conexiones entre las neuronas se ajustan iterativamente utilizando algoritmos de optimización, como el descenso de gradiente, para minimizar una función de pérdida que mide la discrepancia entre las predicciones del modelo y las etiquetas reales de las imágenes de entrenamiento.

Una vez que la red neuronal convolucional ha sido entrenada, puede utilizarse para una variedad de tareas en el procesamiento de imágenes, como la clasificación de objetos, la detección de objetos y la segmentación semántica, contribuyendo así a una amplia gama de aplicaciones prácticas en campos como la medicina, la agricultura y la robótica, entre otros.

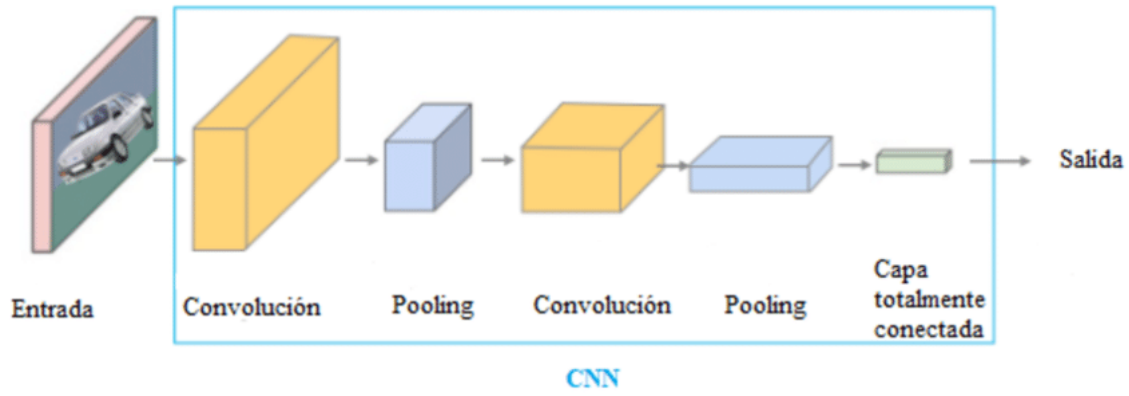

Figura 2.3.2: Capas de una red neuronal [Ref 4]

Entrenamiento y validación de una red neuronal

El proceso de entrenamiento de una red neuronal convolucional, en este caso dedicada al procesamiento de imágenes, es fundamental para lograr un modelo preciso y generalizable, es decir, que pueda ser usada con diferentes conjuntos de datos de entrada. Comienza con la preparación de los datos, donde se divide el conjunto de datos en conjuntos de entrenamiento, validación y prueba. El conjunto de entrenamiento se utiliza para ajustar los pesos de la red durante el entrenamiento, mientras que el conjunto de validación se utiliza para ajustar los hiperparámetros del modelo y evitar el sobreajuste. El conjunto de prueba se reserva para evaluar el rendimiento final del modelo.

Durante el entrenamiento, las imágenes se alimentan a la red neuronal, que consiste en capas convolucionales, de agrupación y completamente conectadas. Las capas convolucionales aprenden características como bordes, texturas y formas mediante la aplicación de filtros convolucionales a las imágenes de entrada. Estas características se pasan a través de capas de agrupación para reducir la dimensionalidad y extraer características invariantes a pequeñas transformaciones. Finalmente, las características se alimentan a capas completamente conectadas que realizan la clasificación o regresión dependiendo de la tarea.

Durante el entrenamiento, las imágenes se introducen en la red en lotes (batches). En cada iteración (epoch), la red procesa estos lotes, calcula las predicciones y compara estas predicciones con las etiquetas reales para calcular la pérdida. Luego, los pesos de la red se actualizan utilizando un algoritmo de optimización, como el descenso de gradiente estocástico, por ejemplo.

Es crucial monitorear el rendimiento del modelo durante el entrenamiento para evitar el sobreajuste (overfitting) y garantizar que el modelo generalice bien a nuevos datos. Para ello, se utiliza el conjunto de validación. Después de cada epoch, se evalúa el modelo en el conjunto de validación y se calculan métricas de rendimiento, como la precisión y la pérdida. Si el rendimiento en el conjunto de validación comienza a disminuir o se estabiliza mientras que la pérdida en el conjunto de entrenamiento sigue disminuyendo, puede ser un indicio de que el modelo se está sobreajustando.

El proceso de validación es crucial para ajustar los hiperparámetros del modelo, como la tasa de aprendizaje, la arquitectura de la red y el tamaño del lote. Se pueden probar diferentes configuraciones de hiperparámetros y seleccionar aquellas que produzcan el mejor rendimiento en el conjunto de validación. Las métricas de evaluación, como la precisión o la pérdida, se calculan en el conjunto de validación y se utilizan para tomar decisiones sobre la arquitectura del modelo y los hiperparámetros

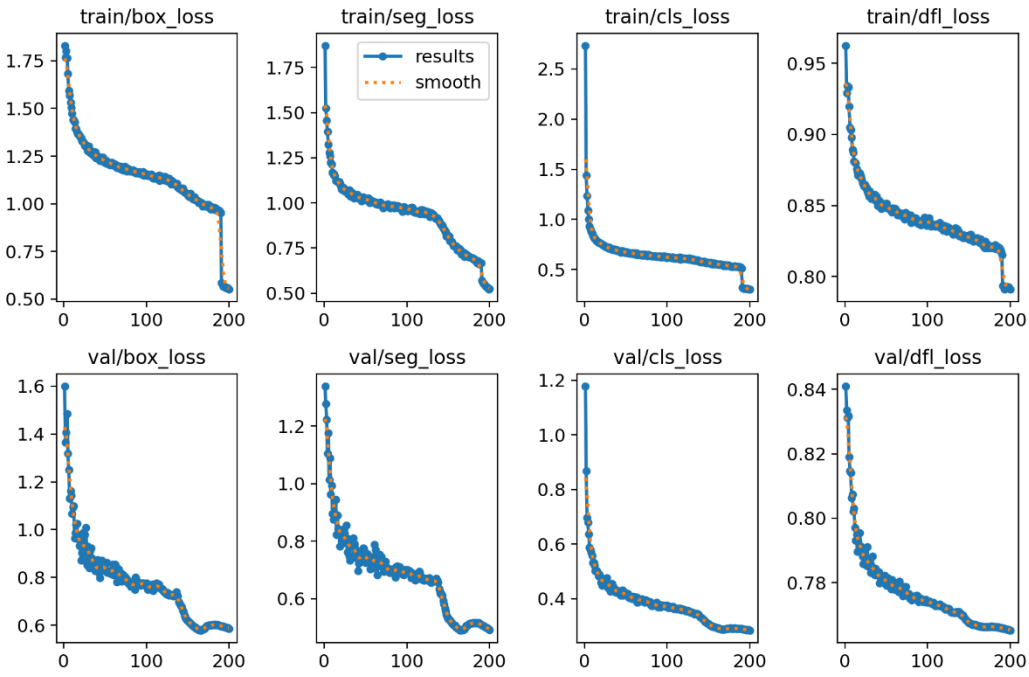

Figura 2.3.1: Hiperparámetros de una red neuronal

Tipos de redes neuronales

Existen varios tipos de redes neuronales que se han desarrollado para abordar diferentes tipos de problemas en el ámbito del aprendizaje automático y el procesamiento de datos. Algunos de los tipos más comunes se listan a continuación:

- Redes Neuronales Feedforward (FNN): Son el tipo más básico de redes neuronales, donde la información fluye en una dirección, desde la capa de entrada hasta la capa de salida sin ciclos ni realimentación. Las FNN son adecuadas para tareas de clasificación y regresión simples.
- Redes Neuronales Convolucionales (CNN): Este tipo de red neuronal, el cual es el tipo de red que ha sido usado en el desarrollo del proyecto, se compone de redes que están diseñadas específicamente para el procesamiento de datos estructurados en forma de matrices, como por ejemplo imágenes. Las capas convolucionales de estas redes permiten detectar patrones locales en las imágenes, lo que las hace especialmente efectivas en tareas de clasificación, detección y segmentación de imágenes.
- Redes Neuronales Recurrentes (RNN): Son aptas para datos secuenciales, donde la entrada tiene una estructura de secuencia temporal. Las RNN son capaces de

modelar dependencias temporales y se utilizan en tareas como el procesamiento del lenguaje natural (NLP), la traducción automática y la generación de texto.

- Redes Neuronales Generativas Adversariales (GAN): Consisten en dos redes neuronales, un generador y un discriminador, que compiten entre sí en un juego adversarial. El generador crea muestras sintéticas que son indistinguibles de las muestras reales para el discriminador. Las GAN se utilizan en aplicaciones de generación de imágenes, super-resolución, y síntesis de datos.

Los modelos de redes neuronales convolucionales (CNN) evolucionaron significativamente en el procesamiento de imágenes, y se desarrollaron varios modelos destacados para abordar diferentes tareas específicas. A continuación, se listan los modelos más populares de CNN para aplicaciones de visión artificial:

- YOLO (You Only Look Once): YOLO es un modelo de detección de objetos en tiempo real que propone una sola pasada de predicción sobre toda la imagen. Utiliza una red neuronal convolucional para predecir las coordenadas de las cajas delimitadoras y las clases de los objetos en una sola iteración. YOLO es conocido por su velocidad y precisión en la detección de objetos en imágenes y videos.
- U-Net: U-Net es un modelo de segmentación de imágenes ampliamente utilizado en aplicaciones médicas, especialmente en segmentación semántica de imágenes biomédicas, como segmentación de tejidos en imágenes de resonancia magnética o segmentación de células en imágenes microscópicas. Se caracteriza por su arquitectura en forma de U, que consta de una ruta de contracción (encoder) y una ruta de expansión (decoder), lo que permite la segmentación precisa de objetos en imágenes.
- ResNet (Residual Network): ResNet ha demostrado ser efectiva en tareas de clasificación y detección de objetos. La clave de ResNet es el uso de conexiones residuales que permiten el entrenamiento de redes más profundas sin sufrir problemas de desvanecimiento del gradiente.
- VGG (Visual Geometry Group): La red VGG, desarrollada por el Visual Geometry Group de la Universidad de Oxford, es conocida por su simplicidad y su arquitectura uniforme, que consiste en convoluciones de 3x3 y capas de max-pooling. Aunque no es tan profunda como algunas de las arquitecturas más recientes, VGG ha demostrado un rendimiento sólido en tareas de clasificación de imágenes.

2.4 Proyectos relacionados

El procesamiento de imágenes de Caenorhabditis elegans (C. elegans) mediante aprendizaje profundo es un campo en rápido crecimiento que ha demostrado ser fundamental para la investigación y análisis de este organismo modelo. En el Instituto de Automática e Informática Industrial (ai2-UPV), durante los últimos cinco años, se han desarrollado diversos sistemas de adquisición de imágenes [Ref14, Ref16, Ref7, Ref19] y técnicas automáticas de procesamiento de imágenes [Ref17, Ref18 Ref20, Ref21, Ref23, Ref24] que permiten automatizar los ensayos de lifespan [Ref15, Ref6, Ref22] o healthspan.

Además de éstos, a continuación, se recogen diversos proyectos similares o relacionados con el sistema desarrollado para este TFM con el fin de realizar una comparativa y visualizar de mejor manera el estado de la ciencia en este ámbito, así como la aportación que implica el trabajo desarrollado a la misma

1. **Deep-Worm-Tracker [Ref 5]**

Este proyecto consiste en un sistema de seguimiento avanzado diseñado para rastrear de manera precisa y en tiempo real los movimientos de los organismos modelo C. elegans. Utilizando técnicas de Deep Learning, combina el modelo de detección de objetos YOLOv5 con el sistema de seguimiento Strong SORT para superar las limitaciones de los métodos tradicionales. Este sistema logra una precisión excepcional en la detección y seguimiento de los nematodos, incluso en condiciones desafiantes como fondos ruidosos u occlusiones.

El proyecto también presenta un conjunto de datos único para la re-identificación de los nematodos, lo que minimiza los errores en el seguimiento de individuos. Con una precisión superior al 98% y tiempos de inferencia rápidos.

Strong SORT utiliza una combinación de características de apariencia y movimiento extraídas por una red neuronal para mantener la identidad de cada nematodo a través del tiempo. Esta técnica es especialmente efectiva para evitar errores de seguimiento durante los momentos en los que se pierde visibilidad en un C. elegan, un desafío común en el estudio de comportamientos animales. Gracias a esta combinación de precisión y eficiencia, Deep-Worm-Tracker se convierte en una herramienta muy válida para la investigación etológica y el seguimiento de organismos modelo.

2. **Towards generalization for Caenorhabditis elegans detection [Ref 6]**

El proyecto busca automatizar ensayos en el nematodo modelo C. elegans, aunque sus características lo hacen ideal para estudiar enfermedades neurodegenerativas y envejecimiento, las tareas de observación individual y clasificación en vivo/muerto requieren mucho tiempo de los técnicos de laboratorio. Por ende, la automatización de estos ensayos se presenta como una solución para liberar a los técnicos de tareas tediosas y acelerar la duración de los experimentos.

Para abordar este desafío, el proyecto propone un enfoque basado en el aprendizaje profundo, utilizando una red neuronal convolucional (CNN) capaz de realizar la detección

generalista de C. elegans, independientemente del sistema de captura de imágenes utilizado y de su apariencia. En lugar de depender de conjuntos de datos específicos para entrenar modelos de detección, el proyecto busca alcanzar una buena generalización de dominio, lo que permitiría al modelo detectar nematodos con mayor robustez en diferentes condiciones de captura de imágenes.

Esto se logra mediante métodos de aumento de datos que amplían la representación del dominio de C. elegans en el conjunto de datos de entrenamiento, además de hacer uso de software que permite simular imágenes de placas de Petri con C. elegans en su interior que permiten entrenar la red neuronal utilizada.

El objetivo final es desarrollar un modelo que requiera un número menor de imágenes para alcanzar resultados deseados, lo que facilitaría su implementación en una variedad de aplicaciones para la automatización de ensayos en C. elegans. Este enfoque innovador promete simplificar y acelerar significativamente la investigación en el campo de la biomedicina utilizando este modelo animal.

3. Small fexible automated system for monitoring Caenorhabditis elegans lifespan based on active vision and image processing techniques [Ref 7]

En este proyecto, se desarrolló y describió una nueva máquina de monitoreo de C. elegans, llamada SiViS, que consiste en un diseño de plataforma flexible y compacta para analizar cultivos de C. elegans utilizando placas de Petri. El sistema utiliza una técnica de iluminación activa y diferentes pipelines de procesamiento de imágenes para detectar movimiento, proporcionando un pipeline de procesamiento de imágenes automatizado. Este estudio validó tanto estos métodos como la viabilidad de la máquina SiViS para experimentos de longevidad mediante la comparación con ensayos manuales de longevidad. Los resultados demostraron que el sistema automatizado produce réplicas consistentes, y no hay diferencias significativas entre los ensayos del sistema automatizado y los ensayos manuales tradicionales.

Una parte crucial de este proyecto es la técnica de post-procesamiento empleada para corregir posibles errores en los recuentos de longevidad obtenidos automáticamente. Dado que las curvas de longevidad deben ser funciones decrecientes monótonas, cualquier error de conteo detectado podría indicar una anomalía. Esta técnica se implementa en dos etapas, considerando diferencias en la aparición de errores durante diferentes fases del ciclo de vida del C. elegans. En la primera etapa, se corrigen los errores de recuento hacia arriba, ya que es más probable que los nematodos vivos estén ocultos o agregados en esta fase inicial del ciclo de vida. En la segunda etapa, se corrigen los errores de recuento hacia abajo, ya que, en esta etapa más avanzada del ciclo de vida, la acumulación de suciedad y la disminución de la supervivencia aumentan la probabilidad de errores. Esta técnica de post-procesamiento garantiza la precisión y confiabilidad de los resultados obtenidos

4. Caenorhabditis elegans Connectomes of both Sexes as Image Classifiers [Ref 8]

En el proyecto, se construyeron dos tipos de redes neuronales artificiales (ANN) basadas en el conectoma (sistema nervioso) de los C. elegans. Primero, se crearon grafos dirigidos acíclicos (DAG) con direcciones de borde aleatorias, desorientando el flujo de información natural del conectoma. Segundo, se desarrollaron DAG que preservan las direcciones y tipos de neuronas (sensoriales, interneuronas y motoras) del conectoma, manteniendo el flujo de información biológico.

Ambas redes se entrenaron para clasificar imágenes de los conjuntos de datos MNIST y fashion-MNIST. Las redes basadas en DAG realistas mostraron una precisión superior: 99.6% para MNIST y 92.7% para fashion-MNIST, comparado con las redes de DAG aleatorios que presentaron menor precisión. Este enfoque demostró que mantener la estructura de flujo de información del conectoma es crucial para mejorar el rendimiento de las ANN.

Los resultados sugieren que las arquitecturas de ANN inspiradas en redes biológicas pueden ofrecer mejoras significativas en la precisión y eficiencia de las tareas de clasificación de imágenes. Las redes que preservan el flujo natural de información del conectoma no solo superaron a las basadas en grafos aleatorios, sino que también mostraron rendimientos comparables con redes más complejas y con mayor número de parámetros. Esto destaca el potencial de las redes neuronales biológicas para diseñar arquitecturas avanzadas de ANN.

2.5 Comparativa

En este apartado se van a desarrollar los puntos clave que diferencian al proyecto expuesto en este TFM con los proyectos relacionados comentados en el apartado anterior. Se explicará qué cosas añade y que tipo de resultados se quieren obtener para poder situar el proyecto en el panorama del estado del arte actual.

En comparación con el proyecto 1 del anterior apartado, pese a que ambos proyectos utilizan redes neuronales para hacer una identificación y seguimiento de los C. elegans, el proyecto 1 funciona mejor en el seguimiento de nematodos individuales en la placa, sin embargo, a la hora de identificar y seguir a varios C. elegans en la misma imagen simultáneamente tiene ciertos problemas en cuanto a la oclusión de estos en la imagen o cuando se pierden al salirse de los bordes de la Placa de Petri. Sin embargo, el proyecto expuesto en este TFM ofrece buenos resultados en esas situaciones y ofrece una solución al problema de la agregación de nematodos entre sí.

En cuanto al segundo proyecto expuesto en el anterior apartado, este utiliza técnicas similares para la detección de los C. elegans en una placa de Petri en diferentes condiciones, además de utilizar, al igual que en este proyecto, un simulador de imágenes de C. elegans que serán utilizadas para entrenar una red neuronal de detección de los mismos sobre una imagen. El proyecto expuesto en este TFM utiliza estos conceptos como base y realiza la detección de nematodos en la placa de Petri, además de realizar un seguimiento de cada nematodo y propone una solución a la agregación de nematodos.

El tercer proyecto desarrolla una máquina de monitoreo automático de C. elegans, sin embargo, este TFM solo ofrece una solución software al tracking de C. elegans en una imagen, no obstante, los recursos de post-procesamiento de imagen expuestos en el proyecto han servido como base para desarrollar los que finalmente han sido utilizados en este TFM.

En el cuarto proyecto se han desarrollado redes neuronales para el procesado de imágenes teniendo en cuenta la estructura del sistema nervioso de los C. elegans, se han tomado y comparado datos de precisión de los diferentes experimentos realizados y se han extraído conclusiones sobre que arquitectura de red es mejor, en este TFM también se expondrán este tipo de razonamientos para las redes neuronales que se han utilizado, sin embargo, se han utilizado arquitecturas convencionales de CNN para la detección de C. elegans, en lugar de redes neuronales basadas en el conectoma de los C. elegans.

Teniendo en cuenta lo anteriormente expuesto, se puede sacar en conclusión que existen diferentes proyectos relacionados con lo desarrollado en este TFM, sin embargo, este proyecto añade diversas funcionalidades y características que lo hacen único que a continuación se van a enumerar:

1. Seguimiento efectivo de cada uno de los C. elegans sobre la imagen.
2. Uso de una red neuronal convolucional adicional para resolver los casos en los que los C. elegans se cruzan entre sí.
3. Implementación del sistema en un nodo de ROS2 para su uso en un proyecto robotizado.
4. Obtención de resultados de menos de un segundo en cada iteración.

2.6 Relación con los Objetivos de Desarrollo Sostenible

El presente TFM se alinea de manera significativa con el Objetivo de Desarrollo Sostenible (ODS) número 3, "*Garantizar una vida sana y promover el bienestar para todos en todas las edades*", debido a diversas cuestiones en relación a la utilidad de las investigaciones sobre nematodos C. elegans y su utilización para impulsar ciertas áreas de la ciencia.

En primer lugar, la investigación biomédica sobre los C. elegans es fundamental para el avance de la salud global. Estos nematodos son un modelo biológico esencial en la investigación científica debido a su simplicidad y a la similitud genética y de rutas moleculares con los seres humanos. El sistema de visión automatizado basado en redes neuronales desarrollado en este proyecto para el análisis de estos nematodos, permite aumentar la precisión y la eficiencia de los ensayos científicos. Esto se traduce en una aceleración del proceso de investigación, posibilitando la identificación y el desarrollo de tratamientos y terapias de manera más rápida y precisa. La automatización reduce errores humanos y optimiza recursos, lo que es crucial para el avance de la biomedicina.

Además, la automatización de ensayos con C. Elegans puede contribuir significativamente a la investigación de enfermedades humanas. Este modelo se usa en gran cantidad de estudios sobre el envejecimiento, enfermedades neurodegenerativas y trastornos metabólicos, entre otros. Al mejorar la capacidad de los científicos para realizar ensayos detallados y repetitivos, el sistema desarrollado en este TFM facilita la obtención de datos más confiables. Esto puede acelerar el descubrimiento de nuevos medicamentos y tratamientos, contribuyendo directamente a la mejora de la salud y el bienestar de la población.

Por último, la implementación de tecnologías avanzadas como las redes neuronales en la investigación científica no solo impulsa la innovación tecnológica, sino que también establece un precedente para futuras aplicaciones en otras áreas de la biomedicina. La utilización de inteligencia artificial en el análisis de datos biológicos puede inspirar el desarrollo de nuevas herramientas y métodos que beneficien la investigación en salud a nivel global. De esta manera, el trabajo presentado en este TFM no solo aborda una necesidad específica en el campo de la investigación con C. Elegans, sino que también apoya los esfuerzos globales para alcanzar los objetivos de salud y bienestar establecidos por las Naciones Unidas en el ODS 3.

Capítulo 3. Análisis de requerimientos

3.1 Software utilizado

Python

Python es un lenguaje de programación de alto nivel conocido por su simplicidad y flexibilidad. Es ampliamente utilizado en diversos campos, incluida la visión artificial, gracias a sus numerosas bibliotecas especializadas, como TensorFlow, PyTorch y OpenCV. Estas bibliotecas ofrecen herramientas y funciones predefinidas que facilitan la implementación de algoritmos de redes neuronales para problemas de visión artificial, como la detección de objetos, el reconocimiento facial o la segmentación de imágenes. Python se destaca en este ámbito por su sintaxis intuitiva y legible, lo que facilita la comprensión y la depuración del código, así como por su comunidad activa que comparte recursos, tutoriales y soluciones a través de foros en línea y repositorios de código abierto.

En comparación con otros lenguajes de programación como C++, Python ofrece una ventaja significativa en el desarrollo de aplicaciones de visión artificial con redes neuronales debido a que proporciona una amplia variedad de bibliotecas optimizadas y herramientas de desarrollo que simplifican la implementación y la optimización de algoritmos de redes neuronales, mientras que en C++, pese a ofrecer mayor rapidez de ejecución, los desarrolladores suelen tener que escribir y gestionar código de bajo nivel, lo que puede ser más propenso a errores y consumir más tiempo.

Por estas razones se ha utilizado Python a la hora de desarrollar este proyecto, ya que ofrece un entorno de desarrollo más intuitivo y eficiente para problemas de visión artificial con redes neuronales, además de proporcionar gran variedad de librerías y herramientas para visión artificial y uso de redes neuronales lo que lo convierte en la elección preferida para este proyecto.

ROS 2

ROS2, o Robot Operating System 2, es una plataforma de código abierto diseñada para facilitar el desarrollo de aplicaciones robóticas. Es una evolución de ROS, con mejoras significativas en términos de rendimiento, modularidad y compatibilidad multiplataforma. ROS2 proporciona un marco flexible y escalable que permite a los desarrolladores construir sistemas robóticos complejos al proporcionar herramientas y bibliotecas para la comunicación entre componentes, la gestión de dispositivos, la integración de sensores y actuadores, y más.

En ROS2, los componentes de un sistema robótico se organizan en nodos. Un nodo es una unidad de procesamiento independiente que puede realizar tareas específicas, como controlar un sensor, ejecutar un algoritmo de procesamiento de datos o enviar comandos a un actuador. Estos nodos se comunican entre sí mediante mensajes, que son paquetes de datos estructurados que contienen información relevante para la interacción entre los nodos. Además de la comunicación basada en mensajes, ROS2 también ofrece la posibilidad de crear servicios, que son intercambios de mensajes síncronos entre un cliente y un servidor. Los servicios permiten a un nodo solicitar una acción específica a otro nodo y esperar una respuesta, lo que facilita la implementación de comportamientos más complejos en sistemas robóticos.

En el contexto de ROS2, un nodo publicador es un componente de software que envía datos o mensajes a un "topic" específico. Estos mensajes, en el contexto de este proyecto son imágenes tomadas desde una cámara a una placa de Petri que contiene a los C. elegans a los cuales se les va a realizar el seguimiento.

Por otro lado, un nodo suscriptor es un componente que recibe y procesa estos mensajes publicados por los nodos publicadores. Los nodos suscriptores están suscritos a un "topic" específico y reciben los mensajes correspondientes, en este caso, recibe las imágenes del topic y llama al programa desarrollado en Python para obtener las coordenadas de todos los nematodos.

OpenCV

OpenCV, (Open Source Computer Vision Library), es una biblioteca de código abierto diseñada para resolver problemas de visión por computadora en tiempo real. Ofrece una amplia gama de funciones y algoritmos para el procesamiento de imágenes y videos, incluyendo detección de bordes, filtrado, segmentación, seguimiento de objetos y reconocimiento de patrones. OpenCV se ha convertido en un estándar de facto en la comunidad de visión por computadora debido a su versatilidad, eficiencia y facilidad de uso.

En el contexto de las redes neuronales, OpenCV se utiliza en varias etapas del proceso de visión artificial. Por ejemplo, se utiliza para preprocesar y preparar datos de imágenes antes de alimentarlos a una red neuronal, lo que implica la manipulación de imágenes, la normalización de datos y la extracción de características. También se utiliza para el postprocesamiento de los resultados de la red neuronal, como la visualización de las predicciones, la segmentación de objetos y la evaluación del rendimiento del modelo. Además, OpenCV se puede integrar fácilmente con otras bibliotecas populares de aprendizaje profundo (Deep Learning), como TensorFlow y PyTorch, lo que permite construir y desplegar sistemas completos de visión artificial con facilidad.

En este proyecto se ha utilizado OpenCV para preprocesar las imágenes obtenidas de la cámara para alimentar a la red neuronal, además de obtener máscaras de imagen de los nematodos para utilizarlas junto a los resultados de la red neuronal para obtener resultados y realizar un postprocesado de los resultados para visualizarlos y medir el rendimiento del sistema.

Ultralytics

Ultralytics es una librería de Python especializada en visión artificial que proporciona herramientas avanzadas para el desarrollo, entrenamiento y despliegue de modelos de redes neuronales, especialmente en el contexto de detección y segmentación de objetos. Esta librería es ampliamente utilizada por su capacidad para trabajar con arquitecturas populares como YOLO (You Only Look Once) y otros modelos de redes neuronales convolucionales.

En el ámbito de la visión artificial, Ultralytics ofrece una amplia gama de funcionalidades. Para empezar, facilita el entrenamiento de modelos de detección y segmentación de objetos mediante una interfaz intuitiva que simplifica el manejo de datos, la configuración del modelo y la evaluación del rendimiento. Además, ofrece herramientas para la

visualización de resultados, lo que facilita la comprensión de cómo los modelos están interpretando las imágenes y mejorando con el tiempo.

Otra característica destacada de Ultralytics es su capacidad para integrarse con sistemas de entrenamiento distribuido, lo que permite acelerar el proceso de entrenamiento y escalar modelos para conjuntos de datos más grandes. Además, la librería ofrece soporte para diversas arquitecturas de red y conjuntos de datos, lo que la hace versátil y adaptable a una amplia variedad de aplicaciones en visión artificial, desde la detección de objetos en imágenes médicas hasta la vigilancia de seguridad en tiempo real.

3.2 Redes neuronales utilizadas

Red neuronal YOLO Segmentación

Una red YOLO (You Only Look Once) funciona mediante la división de la imagen de entrada en una cuadrícula y la predicción de las ubicaciones y las clases de los objetos dentro de cada celda de la cuadrícula simultáneamente. Utiliza una sola red neuronal convolucional para realizar esta tarea en lugar de dividir el proceso en etapas separadas. La red YOLO predice las coordenadas de los cuadros delimitadores que rodean los objetos de interés, junto con las probabilidades de clase para cada cuadro delimitador. Luego, aplica filtros para eliminar cuadros redundantes y produce una salida final que identifica los objetos detectados junto con sus ubicaciones y clasificaciones. Esto permite que la red YOLO sea rápida y eficiente, lo que la hace especialmente adecuada para aplicaciones en tiempo real, como la detección de objetos en videos.

Una red neuronal YOLO Segmentación es una variante del modelo YOLO que se centra en la tarea específica de segmentación semántica en imágenes. Mientras que el YOLO original se enfoca en la detección de objetos, la versión de segmentación se desarrolla para proporcionar una salida de segmentación precisa, donde cada píxel de la imagen se clasifica en una categoría específica. Esto significa que, en lugar de solo identificar y delimitar objetos, una red YOLO de segmentación etiqueta cada píxel, lo que permite una comprensión más detallada de la imagen en términos de sus componentes visuales.

Se ha utilizado YOLO Segmentación en lugar de YOLO Detección debido a su capacidad para proporcionar no solo detección y localización precisas de los organismos, sino también porque ofrece segmentación semántica, lo que significa que puede identificar y distinguir cada C. elegans individualmente del fondo de la placa de Petri. Esto ayuda mucho en este programa, donde se necesita un análisis detallado de cada organismo por separado para llevar a cabo estudios de comportamiento y seguimiento de movimientos. Además, la segmentación ha resultado útil para mitigar problemas de superposición entre los organismos y a mejorar la precisión general de la detección.

Figura 3.2.1: Resultados de una red neuronal YOLO en una imagen [Ref 11]

Red neuronal U-Net

U-Net es una arquitectura de red neuronal convolucional diseñada para tareas de segmentación de imágenes, como la segmentación semántica. Funciona combinando una ruta de codificación, que reduce la resolución de la imagen para capturar características generales, con una ruta de decodificación, que aumenta la resolución para producir una segmentación detallada. Además, emplea conexiones residuales entre las capas de codificación y decodificación para ayudar a transmitir información detallada.

A medida que la imagen se reduce en tamaño, durante la parte de codificación, la red conserva información crucial mediante conexiones especiales llamadas conexiones residuales, es decir, la red "aprende" características generales de la imagen, como podrían ser bordes o texturas. Estas conexiones ayudan a la red a recordar detalles importantes a medida que la imagen se procesa.

Tras esto, durante la parte de decodificación, la red utiliza las características aprendidas durante el proceso de codificación, para generar una segmentación precisa, es decir, que a medida que se aumenta la resolución, la red utiliza esas características para crear una imagen segmentada, donde cada píxel está etiquetado según el objeto al que pertenece.

Esta red neuronal se ha utilizado en el proceso de resolución de agregaciones, debido a que funciona de manera eficiente para localizar el esqueleto de un nematodo conociendo la ventana de imagen donde se encuentra el nematodo y la ventana en el instante anterior junto al esqueleto del nematodo en ese instante. Esto nos permite conocer la posición de cada nematodo en todo momento en el cual se están superponiendo dos o más nematodos.

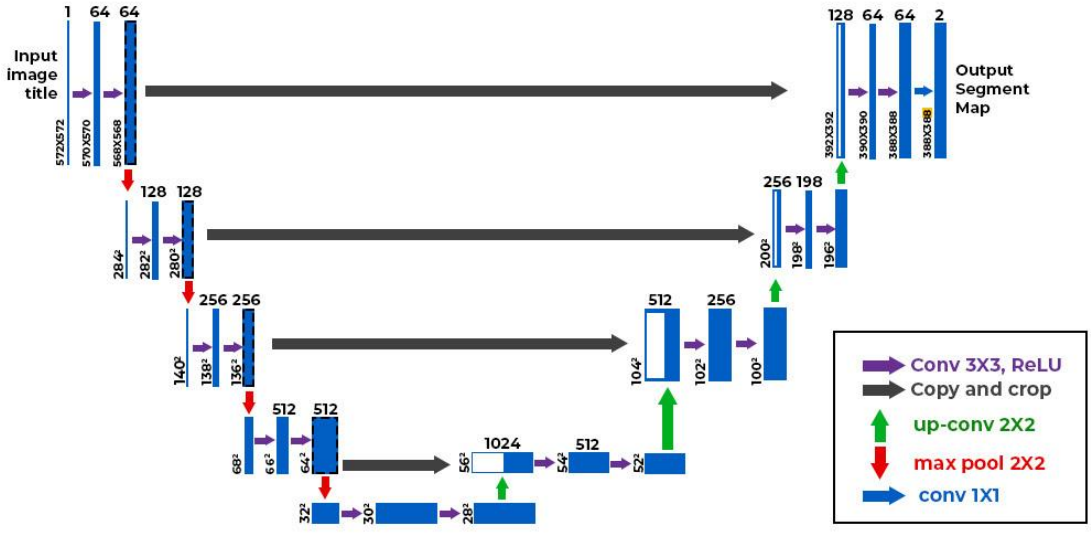

Figura 3.2.2: Esquema de una red neuronal U-Net [Ref 12]

Capítulo 4. Desarrollo del proyecto

En este capítulo se abordará todo el desarrollo realizado en este proyecto. Para poder comprender en mayor medida este apartado, primero se deben explicar los diferentes componentes del sistema y cómo interactúan entre sí para llevar a cabo el propósito principal del mismo, publicar en todo momento las coordenadas de cada nematodo en la Placa Petri.

En primer lugar, para entender de manera correcta el desarrollo del proyecto, cabe explicar que el mismo va a ser utilizado en un proyecto de valorización llamado “Multiview”. Multiview es el nombre de un prototipo diseñado en el ai2 (Instituto de Automática e Informática Industrial) de la UPV, el cual conforma un sistema de monitorización de C.elegans mediante el uso de cámaras “macro” para visualizar todos los C.elegans y una cámara “micro” con mayor resolución que se centrará en visualizar un C.elegan individualmente.

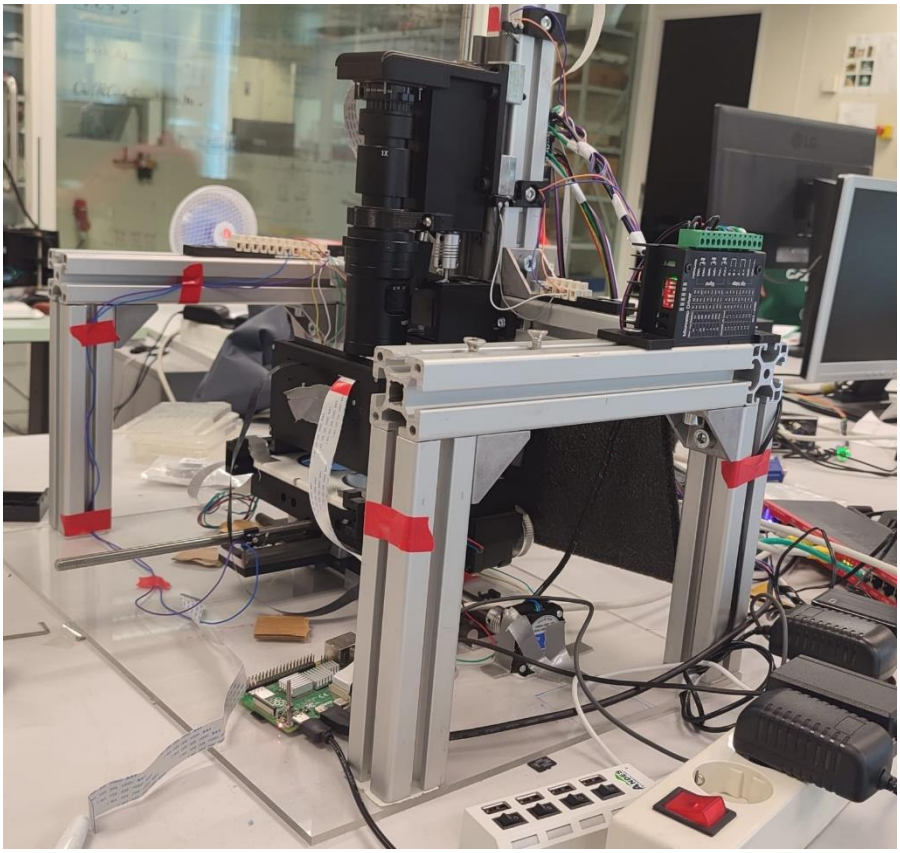

Figura 4.1.1: Fotografía del sistema Multiview

Este proyecto, a rasgos generales, funciona la siguiente forma. En la plataforma blanca se sitúan dos placas de Petri con los C.elegans a monitorizar, las cámaras “macro” ofrecen una imagen general de ambas placas de Petri, ésta imagen será utilizada por el sistema para obtener las coordenadas de cada nematodo en todo momento. Sin embargo la cámara “micro” (la cámara grande situada encima de la plataforma en la imagen), debe situarse de manera que enfoque en todo momento al C.elegan que se está monitorizando.

La posición de la cámara “micro” es fija, para poder seguir al C.elegan correctamente se utilizan varios motores que moverán la plataforma con la Placa de Petri en los ejes x e y. La

posición de la cámara “micro” se controla mediante un puntero láser que estará apuntando al punto donde está visualizando dicha cámara, hasta que logre enfocar un nematodo, en dicho momento el láser se apagará. La iluminación de las placas de Petri es irregular debido a que se gestiona con unos focos de luz situados debajo de las mismas.

El sistema desarrollado para este TFM consiste en un programa que, leyendo las imágenes que ofrece la cámara “macro” (una imagen por segundo), obtenga las coordenadas x e y de cada C.elegan y se lo comunique al resto del sistema para poder mover los motores de manera adecuada.

Teniendo esto en cuenta, se debe aclarar que la estructura final del sistema se compone de un nodo suscriptor de ROS2 el cual toma imágenes de las cámaras “macro” que se están publicando en un determinado topic. Estas cámaras toman imágenes de la Placa de Petri en la que están los C.elegans y las publica mediante un publicador de ROS2. Por otro lado, el nodo suscriptor llama a un archivo programado en Python el cual contiene todas las librerías y funciones necesarias para enviar los datos de posición de cada C.elegan por cada segundo.

De esta forma, el suscriptor obtiene los datos de posición de cada nematodo para la imagen que ha obtenido de la cámara y envía estos datos a un nodo publicador para que los publique en un topic. En el segundo siguiente, la cámara publicará otra imagen que será leía por el nodo suscriptor que realizará el mismo proceso nuevamente hasta que la cámara deje de publicar, este funcionamiento está plasmado en el diagrama a continuación

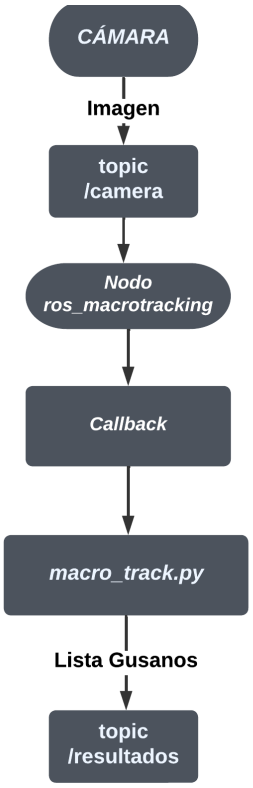

Figura 4.1.2: Diagrama de flujo del programa implementado

Sin embargo, también es necesario explicar la estructura del programa en Python desarrollado para calcular la posición de los C. elegans. En primer lugar, todos los procesos

se gestionan desde una función principal llamada *algoritmo*, a la cual llama el nodo suscriptor de ROS2. Esta función *algoritmo* realiza una llamada a otra función llamada *detectar_gusanos_en_imagen,* la cual envía la imagen actual de la cámara a la red neuronal YOLO y devuelva una serie de resultados que serán pasados de vuelta a la función *algoritmo*.

Con esta serie de datos para cada imagen, esta función gestiona una lista de objetos tipo diccionario que contiene los datos de cada objeto detectado por la red YOLO, esta lista llamada *lista_objetos_detectados* es actualizada por cada llamada a la función *algoritmo*, la cual, tras obtener los datos de la red YOLO se encarga de actualizar los datos e cada objeto detectado de la lista que postencialmente pueda ser un nematodo y añadir los nuevos objetos que hayan sido detectados por la red neuronal.

Cuando dos nematodos se encuentran demasiado cerca, la función lo toma como una posible agregación y llama a la función *anyadir_agregacion,* la cual se encarga de, mediante el uso de umbralizar la imagen mediante OpenCV, discernir si los nematodos se están tocando o no, de esta forma, si no se están tocando simplemente se modificarán sus coordenadas, sin embargo, si resulta que sí se están tocando, se creará un hilo de ejecución que llamará a la red neuronal U-Net para que resuelva la agregación de manera paralela a la ejecución del resto del programa.

Al terminar la agregación, es decir, cuando los nematodos se separan, el hilo que está gestionando dicha agregación, actualizará a los diccionarios de la lista para indicar qué nematodo es cada cual. Al final de todos procesos de la función *algoritmo*, ésta filtra los datos de la *lista_objetos_detectados* que se van a enviar al nodo suscriptor de ROS2. Todo este funcionamiento se encuentra plasmado en el diagrama que se muestra a continuación:

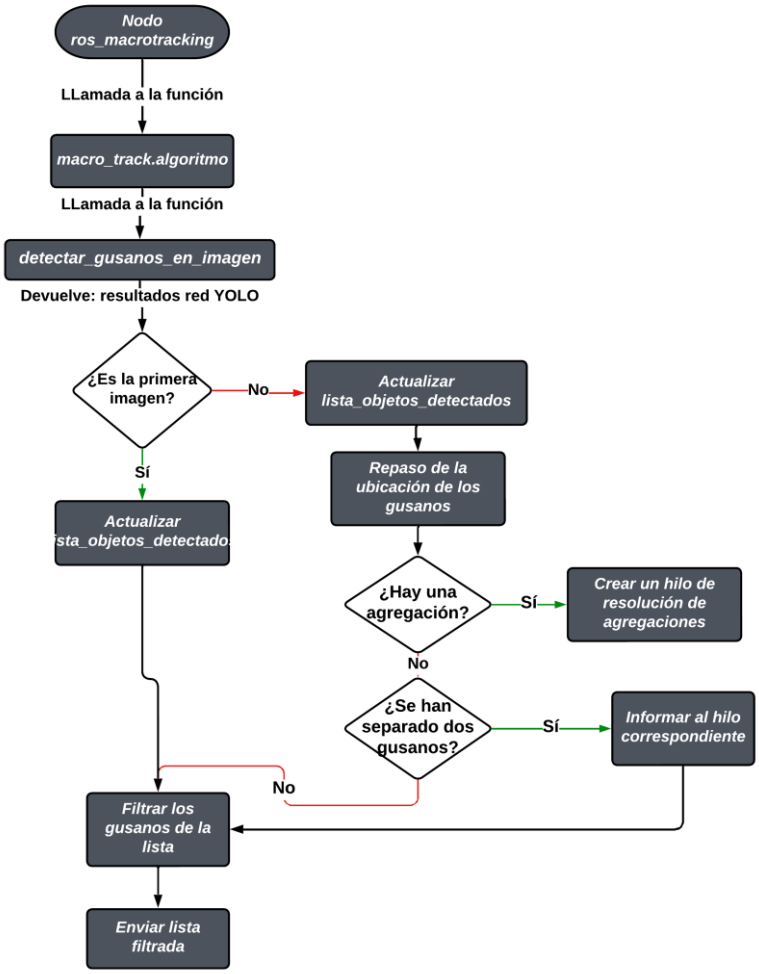

Figura 4.1.2: Diagrama de flujo del programa desarrollado en Python

Para entender mejor cómo se ha dividido este capítulo, hay que tener en cuenta que el desarrollo de este proyecto se ha llevado a cabo en 3 partes diferenciadas: la primera parte tiene que ver con el diseño, entrenamiento y validación de las redes neuronales que han sido utilizadas en el proyecto, la segunda parte compone todo lo referente a la programación en Python que se ha llevado a cabo para hacer el seguimiento de los C. elegans y la tercera parte consiste en la validación del sistema con imágenes reales.

Este proceso se ha llevado a cabo múltiples veces y se han probado gran cantidad de redes neuronales para la detección de los nematodos, así como también se ha cambiado el código en Python múltiples veces para conseguir resultados adecuados, pasando por una gran cantidad de versiones.

Los resultados de la tercera etapa, la etapa de validación de resultados, dictaminaban realmente el éxito o fracaso de las redes neuronales utilizadas y de las funcionalidades en Python añadidas al proyecto, de esta forma las dos primeras partes del desarrollo se han ido repitiendo numerosas veces a lo largo del desarrollo, por lo que, en cierta medida estas tres partes se han llevado a cabo de forma paralela hasta encontrar un buen resultado.

4.1 Entrenamiento, uso y validación de redes neuronales

Uno de los objetivos principales de este proyecto consiste en localizar a los C.elegans en la placa de Petri y obtener sus coordenadas. Para ello, tras probar con diferentes técnicas de visión artificial, se optó por la solución de utilizar redes neuronales convolucionales para detectar a estos nematodos en una imagen.

De esta forma, se valoraron diferentes tipos de redes neuronales para este propósito, entre ellas, se observó que las redes neuronales YOLO cumplían con las características necesarias para este proyecto, debido a que su enfoque de detección de objetos en una sola pasada permite una alta eficiencia computacional sin sacrificar la precisión, por lo que los resultados obtenidos de este tipo de redes son precisos y se obtienen de forma rápida.

De esta forma, en un inicio, se realizó el entrenamiento y posterior validación de diversas redes neuronales YOLO Detección, dichas redes neuronales no ofrecían resultados demasiado buenos para el proyecto, ya que muchos nematodos de la placa no eran detectados por la red neuronal, además de que había ruido en la imagen que sí que lo detectaba como si fuese un C.elegan.

Tras esto, se optó por utilizar redes neuronales Yolo Segmentation, en este caso se obtuvieron resultados mucho más favorables, en la mayoría de los casos, todos los nematodos de las imágenes de validación eran detectados por la red, a pesar de que la velocidad de la red neuronal no era mucho más lenta que las redes neuronales Yolo Detección, por lo que seguía siendo una solución adecuada al problema.

Sin embargo aún había otro problema el cual se debía afrontar en este proyecto, el cual es el hecho de que los C.elegans se mueven con total libertad por la placa de Petri, lo cual puede provocar que se solapen y se crucen en algunos momentos. Esto hace muy difícil poder discernir qué nematodo es cada uno.

En todas las redes neuronales, se observó que cuando se produce una agregación (cuando un nematodo pasa por encima o debajo de otro), la red detecta a la agregación como un solo nematodo, lo que complica saber qué nematodo es cada uno cuando los nematodos se vuelven a separar.

De esta forma se optó por utilizar una red U-Net, entrenada con diversas imágenes de agregaciones de nematodos y sus esqueletos, de forma que, para una imagen, los esqueletos de ambos nematodos en esta imagen y una imagen posterior, pueda predecir los esqueletos de los nematodos en esta imagen posterior.

Esta red neuronal sería usada para obtener los esqueletos de ambos nematodos en todo momento cuando los nematodos se encuentran agregados, de forma que, al separarse, se pueda saber qué nematodo es cada uno y así no perder la referencia de posición de los C. elegans.

4.1.1 Red YOLO Segmentation

Una red neuronal YOLO Segmentation, es un tipo de arquitectura de red neuronal convolucional utilizada para la detección y segmentación de objetos en imágenes. YOLO Segmentation segmenta cada objeto en la imagen a nivel de píxel, lo que significa que proporciona una comprensión más detallada y granular de la ubicación espacial de los objetos en la imagen.

Para trabajar con redes YOLO en Python, es común utilizar la librería Ultralytics. Esta librería proporciona una interfaz sencilla y potente para entrenar, evaluar y utilizar modelos YOLO de manera eficiente. Además de esto, ofrece funcionalidades adicionales como visualización de resultados, integración con sistemas de entrenamiento distribuido y compatibilidad con diversos conjuntos de datos y arquitecturas de red.

La primera red neuronal que fue entrenada fue la red responsable de detectar a los C. elegans en la imagen. Finalmente se ha optado por utilizar una red YOLO Segmentation, sin embargo, el proceso de elección del modelo de red neuronal fue extenso debido a la multitud de opciones que se tenía, se optaron por modelos de red neuronal U-Net, Yolo Detection y también Yolo Pose, sin embargo, debido a los resultados obtenidos se decidió utilizar Yolo Segmentation para este propósito.

El proceso de diseño y entrenamiento de la red neuronal se ha repetido por cada red neuronal entrenada para este propósito, sin embargo, a continuación, se entrará en detalle en este proceso sólo para el caso de la red neuronal que ha sido finalmente implementada.

Obtención de datos de entrada

Para entrenar una red YOLO se necesita un set de imágenes, el cual se dividirá el 70% de las mismas para entrenamiento, el 15% para validación y el 15% para test. De esta forma la red será entrenada con el conjunto de entrenamiento, cambiando los pesos de la red neuronal hasta conseguir un resultado satisfactorio.

Cada imagen se debe corresponder con un archivo de texto que actuará como máscara de imagen, conteniendo en cada línea la clase a la que pertenece cada objeto de la imagen seguido de los píxeles en los cuales se encuentra dicho objeto. En este caso solo hay objetos de la clase “Celegan”, cuyo id es 0, todos los demás objetos de las imágenes se toman como fondo o ruido.

Durante el entrenamiento, la red YOLO Segmentation se optimiza para aprender a predecir las máscaras de segmentación de los objetos en las imágenes de entrada. Esto se logra a través de la minimización de una función de pérdida que compara las máscaras de segmentación predichas por la red con las máscaras reales de los objetos en el conjunto de datos (los archivos de texto correspondientes a cada imagen).

Este proceso de optimización se lleva a cabo mediante algoritmos de descenso de gradiente, ajustando los pesos y sesgos de la red para mejorar su capacidad para segmentar con precisión los objetos, de forma que la red pueda generalizar y segmentar correctamente objetos en nuevas imágenes.

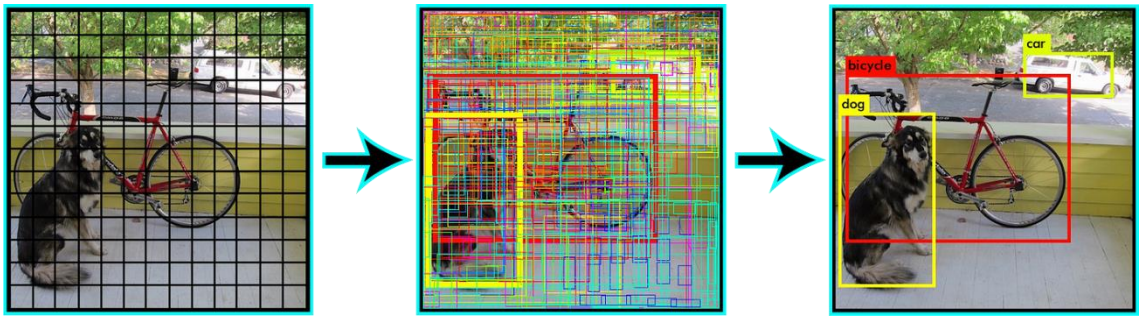

Figura 4.2.1: Proceso de optimización de una red YOLO [Ref 11]

Al proceso de generar las máscaras de imagen se le conoce como “etiquetado”, este proceso puede ser muy largo y laborioso si se hace a mano, ya que, para YOLO Segmentation, hay que indicar en un fichero de texto todos los píxeles los cuales está ocupando cada C.elegan en cada imagen, para Yolo Detection solo se necesita indicar las coordenadas x e y de la esquina inferior izquierda del objeto, su ancho y su alto, lo cual facilita un poco el proceso, sin embargo sigue siendo un proceso muy largo.

Existe una gran cantidad de herramientas online para realizar el etiquetado de las imágenes, sin embargo, el proceso puede llevar una gran cantidad de tiempo, ya que cada imagen puede costar fácilmente media hora en ser etiquetada a mano, además de que el etiquetado para redes con arquitectura YOLO Segmentation es más difícil de realizar.

Sin embargo, debido a limitaciones como las comentadas anteriormente y para contar con un gran número de imágenes de entrenamiento, se hizo uso de un simulador de imágenes programado en C, que permite generar secuencias artificiales de 30 imágenes de C.elegans sobre placas de petri.

Este simulador funciona de la siguiente forma: haciendo uso de imágenes que se usarán como fondo de la imagen final, detecta la zona de la imagen que se corresponde a una la placa de petri, tras esto genera una secuencia de coordenadas suficientemente separadas entre sí y alrededor de esas coordenadas genera una figura similar a un nematodo, teniendo en cuenta una serie de patrones ya definidos.

Las imágenes usadas como fondos para el simulador son imágenes de varias placas de Petri distintas que han sido tomadas con diferentes condiciones de iluminación para que la red neuronal generalice sus resultados lo más posible. Esto es necesario debido a que el sistema va a trabajar con placas de Petri que van a ser iluminadas con foco de luz el cual puede aparecer en cualquier parte de la placa de Petri, generando una zona circular con una iluminación mucho más intensa que el resto de la placa.

Además de lo comentado anteriormente, en las imágenes de la placa de Petri en las que el sistema debe detectar a los C. elegans aparecerá un pequeño punto blanco, el cual es producido por el láser que indica la posición de la cámara “micro”, éste laser, al cabo de unas imágenes, acabará apuntando al nematodo al que se esté realizando el seguimiento en ese momento. Por lo tanto, algunas de las imágenes usadas como fondo contarán también con el puntero láser apareciendo en zonas aleatorias.

A continuación, se muestran diversas imágenes que se han usado como fondo para el simulador:

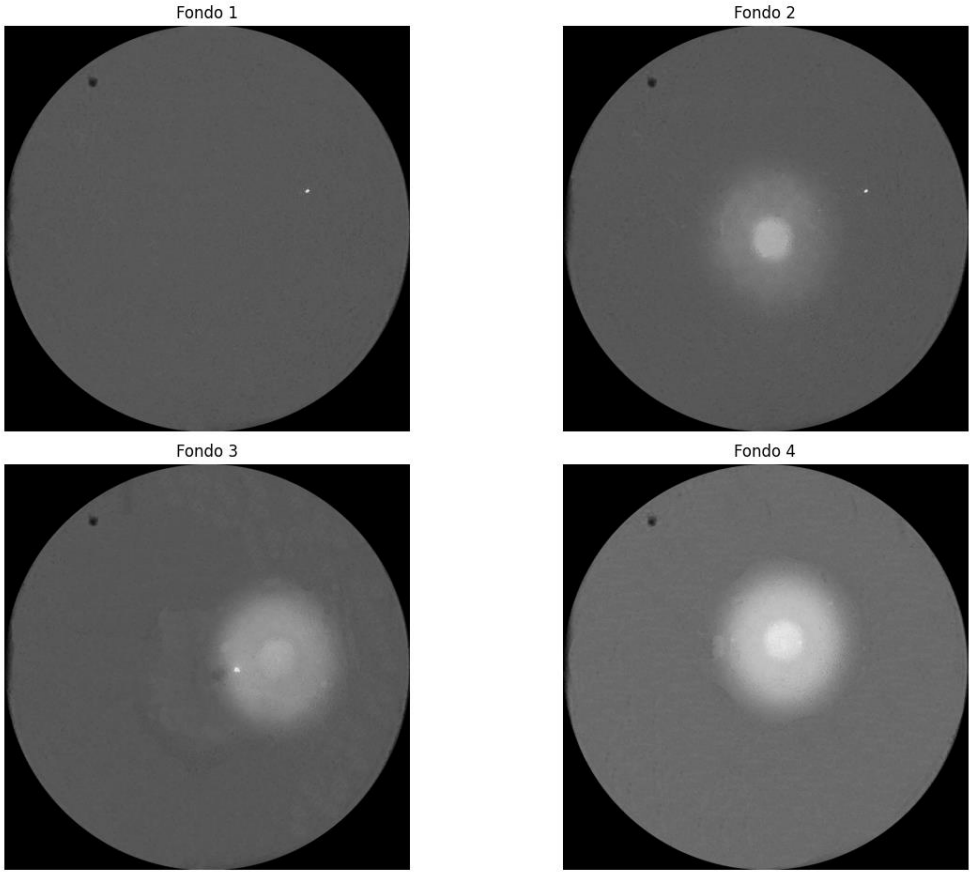

Figura 4.2.2: Imágenes utilizadas como fondos para el simulador

Este simulador puede generar secuencias de 30 imágenes con estos nematodos, las cuales han servido para llevar a cabo diversas pruebas de tracking. Para generar estas secuencias de imagen, el simulador guarda las coordenadas iniciales y finales de cada nematodo, tras esto genera una figura algo reducida en un extremo y algo alargada en el otro, con dirección aleatoria, controlando que los objetos generados no se salgan de la zona de la Placa de Petri ni se crucen entre sí.

También se pueden generar secuencias de imágenes en las que una cantidad de C. elegans definida por el usuario estén agregados en un inicio o se encuentren en paralelo, para que la red neuronal aprenda también a localizar los nematodos que se encuentren en situaciones similares. Además de esto el usuario también puede definir el número de nematodos que se van a generar en cada secuencia de imágenes.

Los nematodos generados con el simulador varían su color teniendo en cuenta la iluminación de la imagen de fondo en las coordenadas en las que se encuentra el nematodo, de esta forma la red neuronal a entrenar generalizará de manera más eficiente a los nematodos que se encuentren en el foco de luz o en la parte oscura de la imagen. A continuación, se muestran diversas imágenes generadas mediante este simulador:

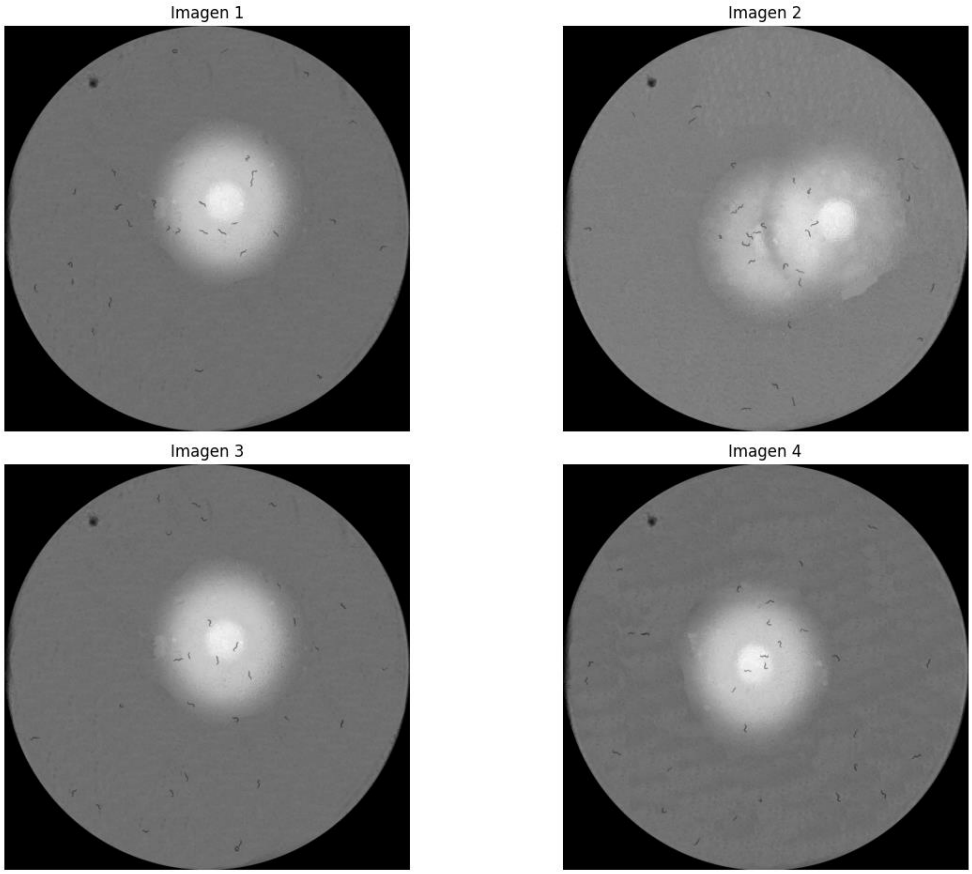

Figura 4.2.3: Imágenes obtenidas con el simulador

De esta forma, generando imágenes con este método, no solo se obtuvieron gran cantidad de imágenes en poco tiempo, sino que también, debido a que el simulador maneja la información de cada píxel que ocupa cada nematodo, la generación de máscaras también se realizó de manera automática. Esto permite entrenar redes neuronales de manera mucho más rápida con diversas disposiciones de objetos en las imágenes.

Este simulador fue modificado para poder realizar el etiquetado de manera automática guardando los datos correspondientes de cada nematodo en cada imagen en un archivo de texto. Además, en cierto momento de la validación se detectó que las redes neuronales YOLO entrenadas con este método muchas veces no detectaban C. elegans que se encontraban cerca de los bordes, de manera que se forzó a que el simulador generase más nematodos en los bordes de la placa, lo cual mejoró los resultados del entrenamiento.

Entrenamiento

Una vez obtenidas las imágenes simuladas con las que la red sería entrenada, el siguiente paso consistía en comenzar con el entrenamiento de la red como tal. Entrenar una red YOLO es muy sencillo para un programador en Python, debido a que todas las configuraciones necesarias como el número de epochs (lotes), el tamaño de imagen con el que trabajará la red y demás variables referentes al Data Augmentation que se explicarán a detalle más adelante; se pueden hacer en una sola línea de código.

Debido a que entrenar una red neuronal es un proceso que consume muchos recursos de la CPU y tarda mucho tiempo en un ordenador de sobremesa común o un portátil, se ha

hecho uso de un cluster en Kubeflow, al cual se accede mediante JupyterLab. Este cluster perteneciente al ai2 cuenta con las siguientes características:

- Memoria RAM: 100Gb
- CPU: 10
- Número de GPUs: 1
- Marca de las GPU: Nvidia

Con estas características el cluster ha permitido entrenar las redes neuronales de manera mucho más rápida y eficiente.

La red utiliza imágenes de 928x928 píxeles, estas dimensiones han sido asignadas realizando diversas pruebas, debido a que imágenes de mayores dimensiones requerían de un mayor coste computacional, lo que se traducía en un incremento en el tiempo que tarda cada imagen en ser procesada por la red Yolo. A continuación, se muestra una lista de los intervalos de tiempo que tardan los diferentes tamaños de imagen en la misma red neuronal:

Dimensión de las imágenes (px)	Tiempo medio de ejecución (s)
1800x1800	0.2
1600x1600	0.14
928x928	0.046
864x864	0.044

Se han utilizado 50 epochs para el entrenamiento porque las redes entrenadas con mayor número de epochs no aportaban mejores resultados y significaban un gasto de tiempo muy grande.

El data augmentation es una técnica que aumenta la cantidad y la diversidad de datos de entrenamiento, lo que implica generar nuevas muestras de entrenamiento mediante técnicas como rotación, traslación, cambio de escala o modificación del brillo de las imágenes originales. Esto aumenta la diversidad del conjunto de datos, lo que ayuda a mejorar la capacidad de generalización del modelo y a evitar el sobreajuste.

El data augmentation utilizado en el entrenamiento de esta red neuronal es el siguiente:

- Degrees=0.25
- Scale=0.3
- Translate=0.1
- Perspective=0.01

Durante el entrenamiento de la red neuronal, estos parámetros controlan las transformaciones aplicadas a las imágenes de entrada y sus máscaras correspondientes. En este caso, "Degrees=0.25" indica que se aplicarán rotaciones aleatorias de hasta 0.25 veces el ángulo original, "Scale=0.3" implica cambios de escala aleatorios de hasta el 30% de la imagen original, "Translate=0.1" significa que se realizarán traslaciones aleatorias de hasta el 10% del tamaño de la imagen, y "Perspective=0.01" indica que se aplicarán distorsiones de perspectiva aleatorias con una probabilidad del 1%. Estas transformaciones ayudan a la red neuronal a aprender patrones más robustos y a mejorar su capacidad para detectar y segmentar objetos en diversas condiciones.

Resultados obtenidos

Cada red neuronal que ha sido entrenada siguiendo este procedimiento ha sido probada con imágenes reales para probar sus resultados y su comportamiento en imágenes similares a las que realizará la detección de nematodos para el sistema, pese a que también se realizaron pruebas de rendimiento con el conjunto de test, los resultados obtenidos en imágenes reales han sido mucho más influyentes a la hora de definir si una red neuronal era adecuada para el sistema o no.

A continuación, se muestra la matriz de confusión obtenida del entrenamiento de la red con el conjunto de test. La matriz muestra el número de píxeles acertados para cada clase, en este caso, como solo se tiene la clase "Celegan", muestra el número de píxeles de la imagen identificados como C. elegans y el número de pixeles identificados como fondo en vertical y el número de píxeles reales que conforman los celegans y el número de pixeles reales del fondo en vertical.

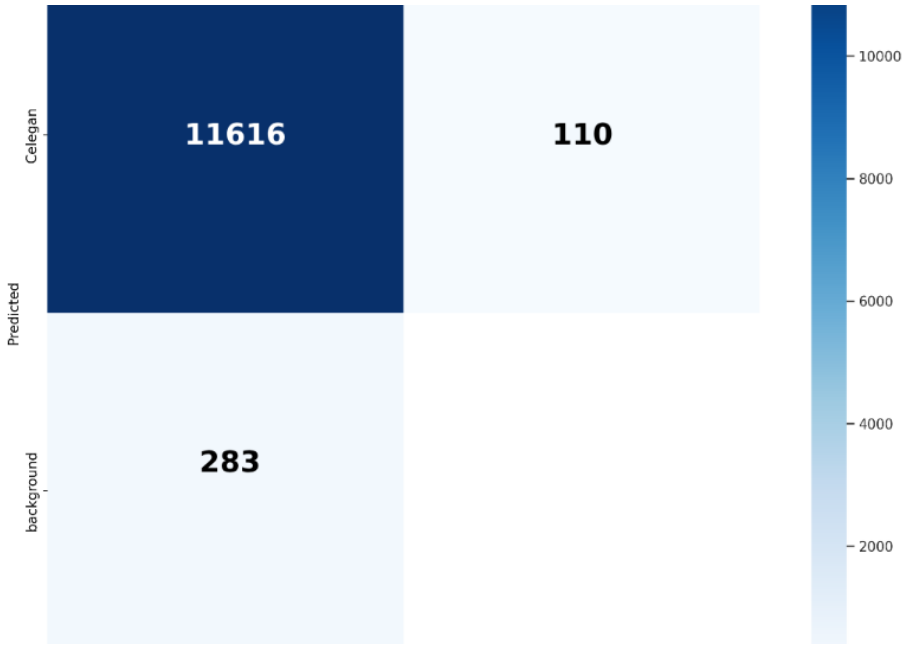

Figura 4.2.4: Matriz de confusión de la red neuronal YOLO

Como se puede apreciar, la gran mayoría de píxeles están bien identificados, sin embargo, hay que tener en cuenta que los números de aciertos serán mucho mayores debido a que está contando los píxeles, no el número de C. elegans bien identificados.

Como ya se ha explicado, los resultados obtenidos validando la red neuronal con imágenes reales han tenido más peso a la hora de evaluar el rendimiento de la red. Por ello se ha probado la red neuronal con gran cantidad de imágenes tomadas de placas de Petri con C. elegans en su interior. Los resultados obtenidos de este proceso de muestran a continuación.

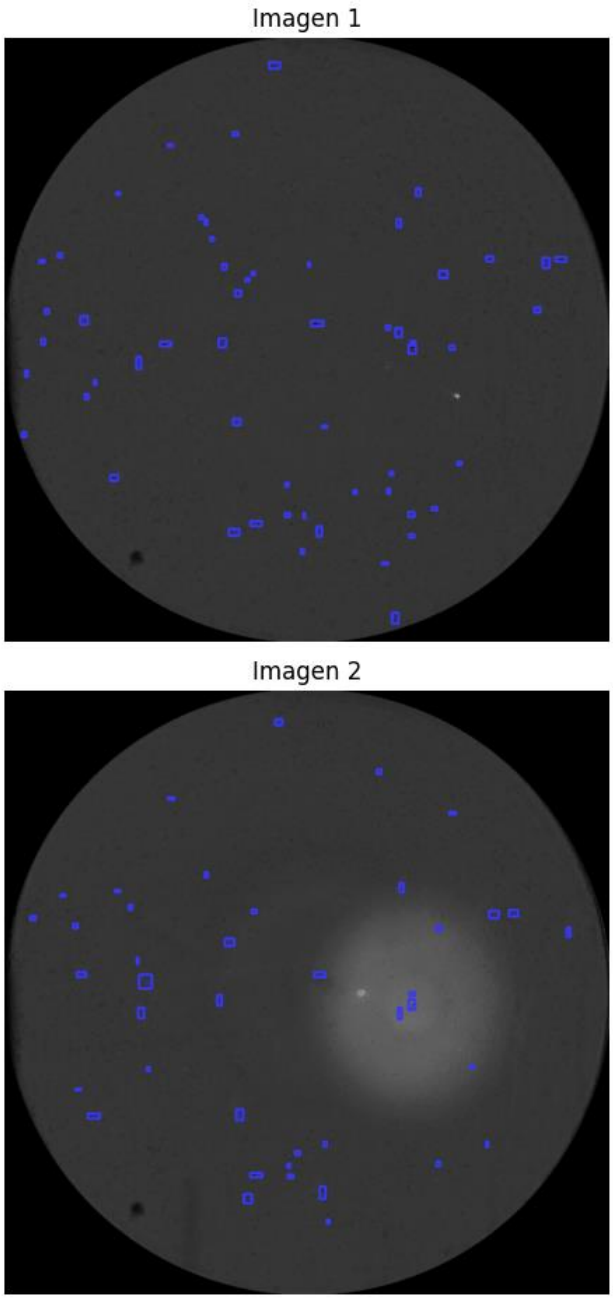

Figura 4.2.4: Resultados de la red neuronal YOLO sobre imagen real

Mediante estas imágenes se han obtenido diversos parámetros de la red neuronal que han ayudado a la hora de decidir cuál de las redes neuronales es la ideal para ser incorporada en el sistema. Este proceso se explica en detalle en el apartado 4.3.1 de este documento.

4.1.2 Red U-Net

La red neuronal utilizada para predecir los esqueletos de los C.elegans que se encuentran agregados para poder resolver las agregaciones de manera adecuada es una U-Net que ha sido entrenada con este propósito con un conjunto de imágenes de agregaciones de nematodos. Se ha elegido este modelo de red neuronal convolucional por su capacidad para extraer características de las imágenes.

La característica distintiva de la U-Net es su estructura de codificador-decodificador, que permite capturar tanto información de contexto como detalles de alta resolución. En la etapa del codificador, la red reduce progresivamente la resolución espacial de la imagen mientras extrae características relevantes. Luego, en la etapa del decodificador, la red reconstruye la imagen segmentada a partir de las características de bajo y alto nivel, restaurando así la resolución espacial.

Se debe tener en cuenta que esto facilita la segmentación precisa de objetos incluso en imágenes de alta resolución, sin embargo el coste computacional de trabajar con este tipo de redes es mayor al de trabajar, por ejemplo, con redes YOLO, por lo que la red tardará más tiempo en procesar cada imagen.

Para entrenar y utilizar una U-Net, no se necesita una librería específica como el caso de ultralytics para las redes YOLO, sin embargo generalmente se requieren librerías como TensorFlow o PyTorch, así como bibliotecas adicionales para manipulación de imágenes como OpenCV y herramientas para la visualización de datos como Matplotlib.

Para implementar las redes neuronales U-Net en Python se hace uso de la librería unet_model_AL, la cual proporciona una interfaz sencilla y eficiente para construir, entrenar y desplegar modelos U-Net en tareas de segmentación de imágenes médicas y biológicas. Esta librería está optimizada para trabajar con conjuntos de datos voluminosos y ofrece funcionalidades avanzadas para la manipulación y preprocesamiento de imágenes, así como para la evaluación del rendimiento del modelo.

Obtención de datos de entrada

Para obtener los datos necesarios para entrenar la U-Net también se ha hecho uso del simulador, en este caso especificando que en las imágenes debe de haber únicamente 2 nematodos y 2 de ellos deben estar agregados, esto produce imágenes de placas de Petri en las cuales solo hay dos nematodos que se están todando.

De esta forma el simulador también genera los archivos .pts necesarios para etiquetar las imágenes para el correcto entrenamiento de la U-Net. Estos archivos se componen de 3 líneas, la primera de ellas, que empieza con el carácter “P” indica las coordenadas en la imagen de los pixeles que componen los esqueletos de uno de los nematodos en la imagen. La segunda línea, que comienza con el carácter “C” indica el valor del color de ese pixel en escala de grises (0 para negro – 255 para blanco). La tercera línea indica el ancho del nematodo en ese píxel, es decir, si tomamos desde ese píxel, cuantos píxeles de distancia hay hasta salirse del nematodo hacia un lado y hacia el otro de éste.

Las tres últimas líneas indican lo mismo pero para el otro nematodo de la agregación. De esta forma, la red neuronal solo necesita un archivo .pts por imagen para entrenar, en lugar de dos archivos por imagen.

La red neuronal, para entrenar, toma ventanas de 100x100 píxeles centradas en un nematodo y en otro respectivamente y las compara con el mismo archivo .pts para aprender y de esta forma actualiza los pesos en cada momento, de manera que por cada imagen obtiene 2 ventanas, cada una centrada en un nematodo.

```
1 P530 771;529 771;528 771;527 771;526 772;525 771;524 772;523 773;522 774;522 775;522 776;522 777;522
  778;522 779;522 780;522 781;522 782;522 783;521 784;521 785;521 786;521 787;520 787;519 787;518 788;517
  789;516 788;515 788;514 788;513 788;512 787;511 787;510 786;509 785;509 784;508 784;507 783;506 782
2 C44.000000;41.500000;38.000000;37.000000;37.333332;36.833332;36.833332;34.166668;32.000000;32.500000;31.66
  6666;31.000000;30.000000;30.333334;29.833334;29.833334;30.000000;30.000000;30.833334;31.333334;30.000000;3
  0.166666;31.000000;31.833334;31.666666;32.000000;31.833334;31.500000;32.000000;33.000000;33.500000;33.8333
  32;34.333332;35.500000;37.333332;39.500000;39.500000;41.666668
3 W1.500000;1.530000;1.868889;2.034444;2.500000;2.500000;2.277778;2.107778;2.166667;2.181111;2.118889;2.1555
  55;2.500000;2.500000;2.500000;2.500000;2.500000;2.500000;2.452222;2.405555;2.471111;2.422222;2.570000;2.54
  4444;2.500000;2.500000;2.107778;2.166667;2.166667;2.166667;2.068889;2.023333;2.251111;1.938889;1.585556;1.
  500000;1.500000;1.500000
4
5 P498 772;498 771;499 771;500 771;500 770;501 769;502 769;503 769;504 769;505 770;506 771;506 772;507
  773;508 774;508 775;508 776;509 776;510 777;511 777;512 778;513 778;514 779;515 779;516 779;516 780;516
  781;517 781;518 781;519 782;519 783;519 784;519 785;519 786;519 787;519 788
6 C41.833332;40.500000;37.500000;34.166668;32.166668;32.333332;31.166666;28.500000;28.333334;29.833334;27.16
  6666;28.666666;29.333334;29.166666;31.000000;30.500000;30.000000;28.666666;27.000000;27.666666;28.333334;2
  8.166666;30.333334;31.500000;31.833334;33.166668;33.333332;32.833332;31.500000;32.000000;31.500000;31.1666
  66;35.666668;39.500000;43.333332
7 W1.500000;1.500000;1.500000;1.500000;1.500000;1.500000;1.653333;2.010000;2.107778;1.773333;1.773333;1.7733
  33;1.773333;2.166667;2.166667;2.181111;2.181111;2.166667;1.953333;1.566667;1.843333;1.938889;2.313333;2.12
  5556;2.010000;2.068889;2.023333;2.023333;1.925555;2.082222;1.703333;1.843333;1.566667;1.500000;1.500000
8
9
```

Figura 4.2.5: Ejemplo de una imagen etiquetada para la red neuronal U-Net

Entrenamiento

Esta red neuronal trabaja con diversos datos para hacer cada predicción, por un lado trabaja con la imagen de los nematodos que quiere predecir sus esqueletos, por otro lado trabaja con la imagen anterior en el tiempo a dicha imagen y por último también tiene en cuenta el esqueleto, en la imagen anterior, del nematodo que está prediciendo, de forma que la salida de la red neuronal es el esqueleto de ese nematodo pero en la imagen actual

Debido a que el simulador produce las imágenes en secuencias ordenas de 30 imágenes donde los nematodos simulados se van moviendo, la red neuronal puede saber en todo momento del entrenamiento qué imagen es la imagen anterior a la que está prediciendo, de esta forma la red puede entrenar sabiendo cual es la imagen anterior a la actual y sus respectivos esqueletos almacenados en el archivo .pts correspondiente.

Para entrenar esta red neuronal, se ha indicado que solo hay una clase (esqueleto de C. elegan) y que el tamaño de la imagen es de 100x100 píxeles. No se ha utilizado data augmentation para preservar la estructura espacial de las características de la imagen debido al uso que hace la U-Net de éstas así como para garantizar la consistencia entre imágenes y etiquetas.

Se utilizado un número máximo de 100 iteraciones para el entrenamiento de esta red neuronal para tratar de garantizar los resultados más exactos posibles ya que la red debe predecir cada pixel del esqueleto del C.elegan en cuestión. Se ha utilizado un 70% de imágenes para el proceso de entrenamiento, un 15% para la validación y un 15% de test, siendo un total de 6000 imágenes utilizadas en este proceso.

Resultados obtenidos

A continuación se muestran diversos resultados de la U-Net con datos tomados del conjunto de test, de esta forma se muestran a la izquierda las ventanas de imagen (arriba la de la imagen anterior y abajo la de la imagen actual) y a la derecha los esqueletos del nematodo en el que se centra la imagen, siendo el de arriba el esqueleto obtenido mediante los archivos .pts y el de abajo el esqueleto que ha predicho la red neuronal.

Figura 4.2.6: Resultados de la red neuronal U-Net en imágenes simuladas

A lo largo de todas las iteraciones del entrenamiento, la pérdida de los resultados ha ido bajando hasta obtener una pérdida de 0.001 para los datos de entrenamiento y una pérdida del 0.0022 para los datos de validación. De esta manera a continuación se muestra una gráfica con el decremento de la pérdida con respecto al tiempo (conforme pasaban las iteraciones), se puede observar que al principio la red aprendía muy rápidamente y conforme aumentaban las iteraciones el aprendizaje se hizo más lento.

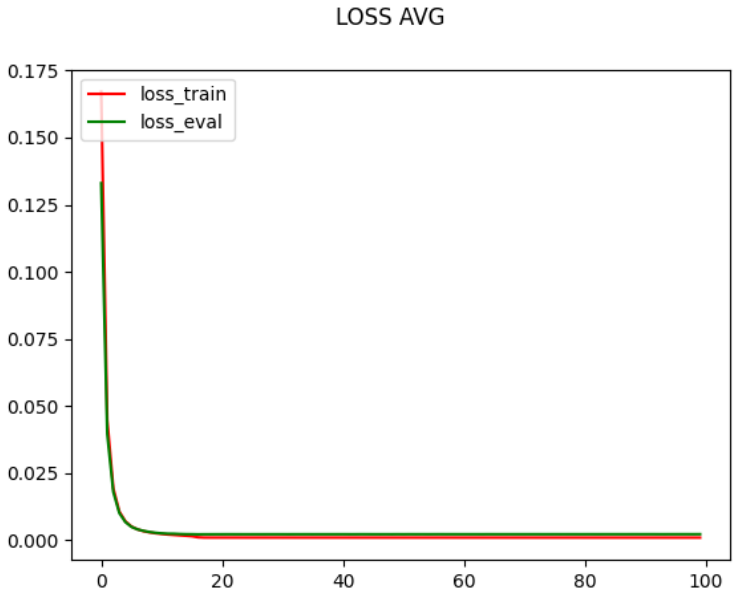

Figura 4.2.7: Gráfica de la pérdida de datos en cada iteración en el entrenamiento de la U-Net

Los resultados obtenidos con los datos de test nos indican que el entrenamiento debería proporcionar una red que otorgue resultados satisfactorios, sin embargo esto no es una garantía segura de que vaya a ser así. Por ello se han utilizado imágenes reales para probar el comportamiento de la red neuronal. Algunos resultados que proporciona la red neuronal con agregaciones en imágenes reales se muestran a continuación:

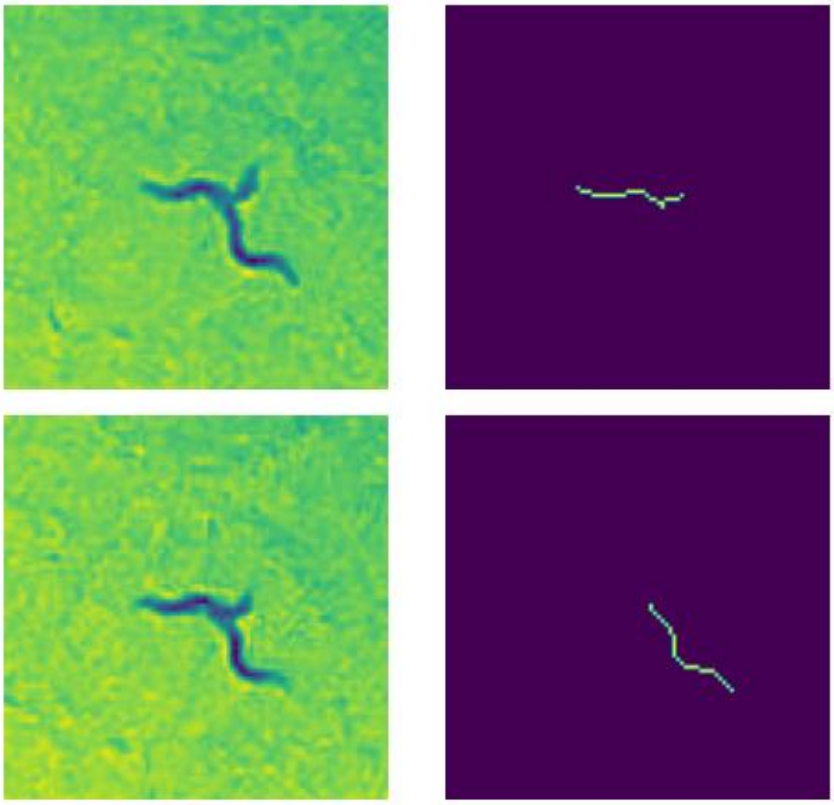

Figura 4.2.6: Resultados de la red neuronal U-Net en imagen real

De la validación con estas imágenes se han obtenido una serie de parámetros con los que se ha medido la calidad de esta red neuronal, esta serie de parámetros y cómo se han obtenido se muestran en el capítulo 4.3.1 de este documento.

4.2 Programación en Python

Además del uso de redes neuronales, era necesario para el proyecto diseñar, programar y verificar un programa que leyese las imágenes obtenidas del topic de la cámara “micro” y, haciendo uso de librerías como OpenCV y las redes neuronales entrenadas previamente, realice un seguimiento de los C. elegans en la imagen, publicando en otro topic, una lista con objetos que contengan en id y las coordenadas de cada nematodo.

El objetivo principal de este programa es poder enviar las coordenadas de cada nematodo en cada instante con una velocidad no inferior a un segundo y no perder la referencia de ningún nematodo, ni siquiera confundir a los nematodos cuando se crucen, es decir, resolver de manera adecuada las agregaciones.

Se ha decidido utilizar Python como principal lenguaje de programación para este proyecto debido a su gran cantidad de librerías para trabajar con imágenes, así como su gran facilidad para trabajar con éstas y para implementar redes neuronales gracias también al uso de librerías como ultralytics y unet_model_AL, descritas previamente en este documento.

Sin embargo, hay que tener en cuenta de que Python es un lenguaje de programación más lento que otros como C o C++, lo cual puede ser perjudicial para el sistema debido a que el mismo debe publicar en un periodo inferior a un segundo por imagen. Debido a esto, se ha optimizado el código lo más posible para que su respuesta sea la más rápida posible en cada instante.

4.2.1 Estructura de clases y librerías

La estructura de clases que se ha seguido a la hora de diseñar este sistema se muestra a continuación:

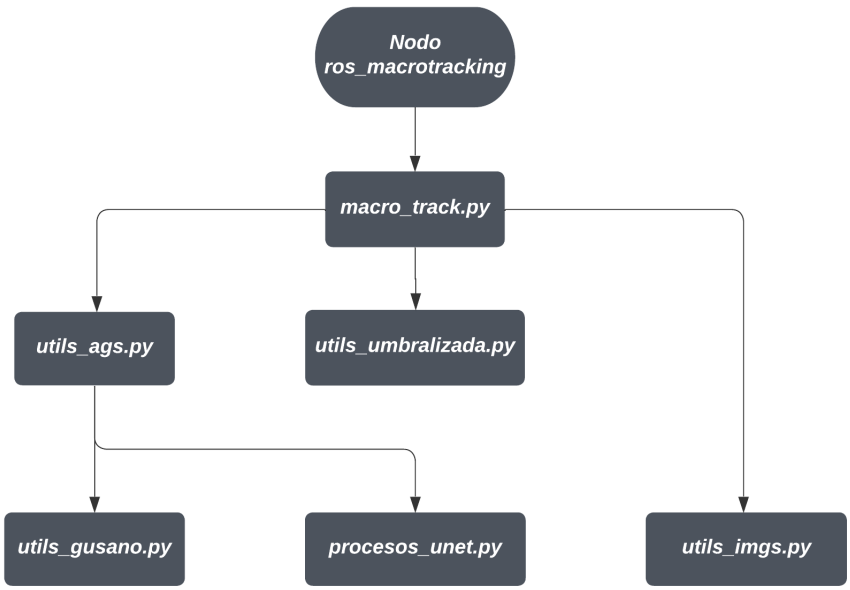

Figura 4.3.1: Diagrama con la estructura de clases y librerías del programa

De esta forma el archivo ros_macrotracking.py contiene el nodo de ROS2 encargado de leer las imágenes del topic correspondiente, dichas imágenes, que se recibirán con una frecuencia de una imagen por segundo, son las correspondientes a las cámaras “macro”. De esta forma el nodo de ROS2 llama a una función Python llamada “algoritmo”, que se encuentra en el archivo “macro_track.py” que será la que devuelva la lista con los datos de los C. elegans que sebe ser publicada.

Esta función “algoritmo” actúa como una especie de nexo con el resto de funciones de otros archivos. Al recibir como parámetro una imagen llama al archivo “utils_imgs.py” en concreto a la función “detectar_gusanos_en_imagen”, la cual utiliza a la red neuronal YOLO para obtener los datos de cada nematodo detectado por ésta en la imagen. También obtiene una másara de imagen que corresponde a diversos objetos de la imagen que podrían ser nematodos, esta imagen la obtiene llamado a “umbralizar_imagen” en el archivo “utils_umbralizada.py”, la cual obtiene esta imagen mediante el uso de un AdaptiveThreshold de OpenCV.

Tras esto, la función “algoritmo” crea un diccionario con los datos de cada nematodo y lo actualiza en la lista de objetos detectados por la red YOLO mediante la función “actualizar_gusano” del archivo “utils_gusano.py”, si el diccionario corresponde a un objeto nuevo, lo crea y añade a la lista de objetos.

Tras esto la función llama al archivo “utils_ags.py”, en concreto a la función “repaso_de_objetos” la cual tratará de encontrar a los C. elegans de la imagen que no hayan sido detectados en la imagen por la red YOLO, así como tratar de identificar las agregaciones de nematodos y resolverlas.

Tras ello, la función “algoritmo” actualiza las variables globales como “contador_imagenes”, filtra los objetos de la lista de objetos detectados que pueden ser nematodos y crea una segunda lista de diccionarios con los datos de interés para ser enviados de cada uno de los nematodos y los devuelve al archivo “ros_macrotracking.py”

La función “repaso_de_objetos”, si detecta una agregación en la imagen llamará a la función “anyadir_agregacion”, la cual, si se trata de una agregación nueva, creará un hilo de ejecución que tratará de resolver la agregación. Este hilo de ejecución llamará al archivo “procesos_unet.py” el cual hará uso de la U-Net para predecir los esqueletos de los nematodos y resolver las agregaciones de manera exitosa.

El funcionamiento del sistema completo, así como de los procesos que se realizan en cada función, serán explicados en detalle más adelante en este mismo capítulo.

4.2.2 Uso de ROS2

Debido a que la estructura general del proyecto parte de leer una serie de imágenes que publican las cámaras "micro" de un topic, hubo que programar un nodo de ROS2 que leyese las imágenes de dicho topic, obtuviese una lista con las coordenadas de cada C. elegan para cada imagen y publicaste esa lista en otro topic para que el sistema de Multiview haga uso de esta.

La estructura de este nodo de ROS2 es la siguiente: Se crea una clase ImageSuscriber la cual contiene un suscriptor que lee del topic en el cual las cámaras están publicando las imágenes, un objeto imagen_cv el cual será la imagen leída del topic en cada momento, un objeto array_gusanos el cual contendrá la lista con las coordenadas de cada nematodo en todo momento y un publicador que será usado para transmitir la lista de los nematodos al resto del sistema.

De esta forma, cuando se reciba una imagen del buffer, se llamará a un callback, el cual guardará la imagen recibida en imagen_cv, llamará a la función "algoritmo", la cual devolverá la lista con las coordenadas de cada nematodo, tras ello guardará la lista como un String y lo enviará al topic correspondiente mediante el publicador. Este proceso se repetirá de manera exactamente igual para cada imagen de la secuencia, a continuación se muestra un diagrama de flujo con las operaciones que lleva a cabo el nodo de ROS2

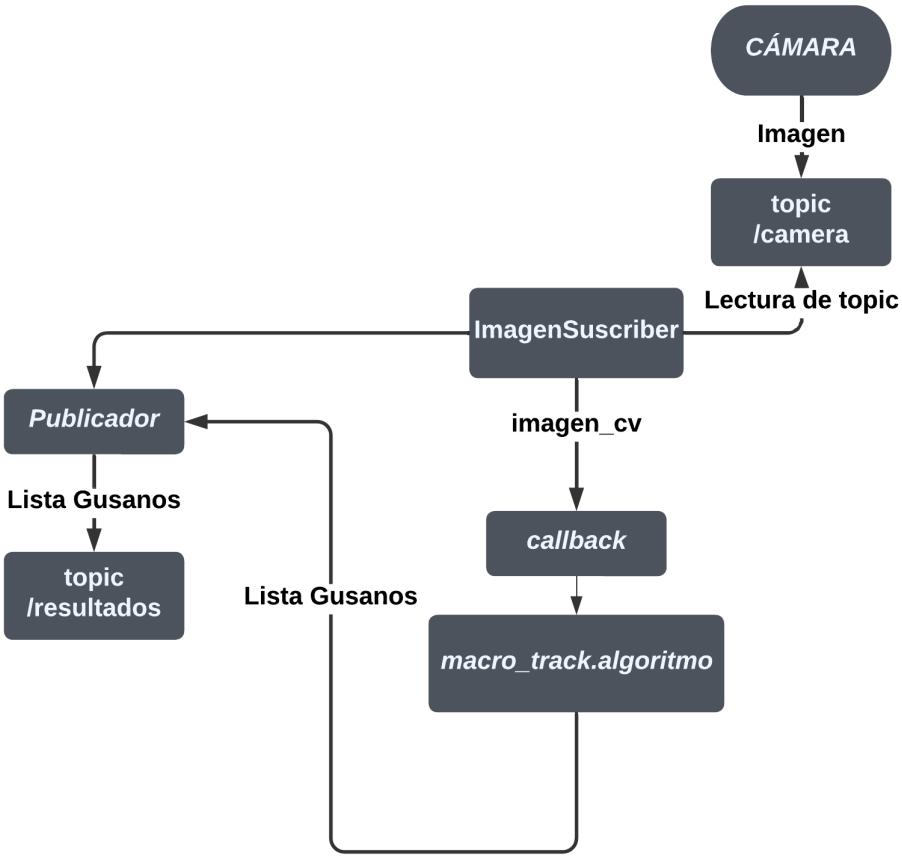

Figura 4.3.2: Diagrama de flujo del programa desarrollado en ROS2

4.2.3 Uso de OpenCV

En este apartado se mencionan todos los archivos programados en Python programados para procesar cada imagen. Tanto los procesos referentes a la implementación de la red neuronal YOLO en el sistema, como el resto de los procesos que hacen uso de diversas funcionalidades de OpenCV para seguir de manera efectiva a cada C. elegan en la placa a lo largo de la secuencia de imágenes.

Conjuntamente con los datos extraídos de la red neuronal YOLO, los cuales indican las cajas que delimitan a los diferentes nematodos de la Placa de Petri, también se hace uso de una máscara de imagen obtenida mediante cv2.AdaptiveThreshold la cual contiene los píxeles de los objetos que resaltan con el fondo de la imagen, en este caso, quitando algo de ruido, los píxeles blancos de la máscara corresponden con los C. elegans de la imagen.

Mediante esta máscara de imagen se realizan operaciones como discernir que nematodo es cada cual cuando 2 o más nematodos se encuentran muy cercanos, detectar la posición de un C. elegan para los instantes en los que la red YOLO no lo detecte o también para detectar agregaciones, es decir, para detectar cuando 2 o más nematodos se están solapando o tocando unos a otros.

En el momento que 2 o más nematodos se solapan o tocan, poder realizar el seguimiento de éstos sin perder la referencia de cual es cada uno es muy difícil realizarlo utilizando solamente OpenCV o la red neuronal YOLO. Para resolver este problema se van a utilizar hilos en paralelo al programa principal y se hará uso de otra red neuronal como se explica más adelante en el siguiente apartado.

Sin embargo, todo lo referente al seguimiento de cada C.elegans individual a través de la secuencia de imágenes y la obtención de características del mismo va a ser explicado en detalle en este apartado.

Implementación de la red neuronal YOLO

Como ya se ha comentado anteriormente, la función “algoritmo” es a la que llama el nodo de ROS2, lo primero que hace esta función es reescalar la imagen obtenida desde la cámara, de una imagen de 1800x1800 píxeles a una imagen de 928x928 píxeles. Esto se hace para poder pasarla a la red YOLO para que detecte los C. elegans de la imagen.

Antes de llamar a la red neuronal para la detección de los C. elegans en la imagen, se ha de calcular la máscara umbralizada de la imagen actual la cual será de mucha ayuda más adelante, esta máscara se calcula con cv2.AdaptiveThreshold y se trata de borrar la mayor cantidad de ruido de la misma, así como de evitar detectar el borde de la propia placa de Petri en la misma.

Posteriormente se llama a la función “detectar gusanos en imagen” que devolverá los resultados obtenidos enviando la imagen reescalada a la red neuronal YOLO, dichos resultados contienen una lista con máscaras de cada objeto detectado y otra lista con las cajas que encierran dichos objetos, además de otros datos de interés como el tiempo de detección que van a ser utilizados para evaluar el rendimiento del sistema.

Tras obtener los resultados de la red YOLO, la función algoritmo comprueba si la imagen procesada es la primera de la secuencia de imágenes, si esto sucede, para cada objeto detectado por la red YOLO se calculará su centroide y el radio de la circunferencia mínima

que lo encierra, además de otros datos de interés como su ancho, alto y los radios a y b de la elipse que lo encierra. Todos estos datos se guardan en un objeto “gusano” que es guardado dentro de una lista “lista_objetos_detectados”.

Si la imagen procesada no es la primera de la secuencia de imágenes, para cada objeto detectado por la YOLO se le compara con todos los objetos de la “lista_objetos_detectados” para saber si se trata del mismo objeto que se ha movido de una imagen a otra. Si la suma del radio del objeto detectado y uno de los objetos de la lista es menor a la distancia que separa sus centros, se toma como que son el mismo objeto, por lo que se actualizan los datos del objeto de la lista, de hecho, se toma nota de si se ha movido demasiado poco de una imagen a otra para detectar a los C. elegans que permanezcan quietos toda la secuencia.

Si según este proceso, el objeto detectado por la YOLO es uno nuevo, se detectará si dicho objeto está muy cerca de una agregación que se está produciendo, ya que cuando las agregaciones suceden, a veces la YOLO detecta objetos que no existen cerca de éstas, lo cual es perjudicial ya que estos objetos no son C.elegans, sino parte de un C.elegan que ha detectado la YOLO mientras éstos se están solapando, por ello a estos objetos se los detecta y elimina.

Siguiendo esta lógica, si el nuevo nematodo cumple con la característica de que el radio de su circunferencia es mayor a un umbral establecido, se añade a la lista. Cada objeto de la lista cuenta con un campo llamado “probabilidad gusano” el cual aumentará por cada vez que sea detectado por la red neuronal YOLO, de forma que la función filtrará a los objetos de esta lista con un umbral que se comparará con el valor de este campo, de esta forma solo se enviarán los nematodos cuya “probabilidad gusano” sea mayor a dicho umbral.

Todo este funcionamiento se ve reflejado en el siguiente diagrama de flujo:

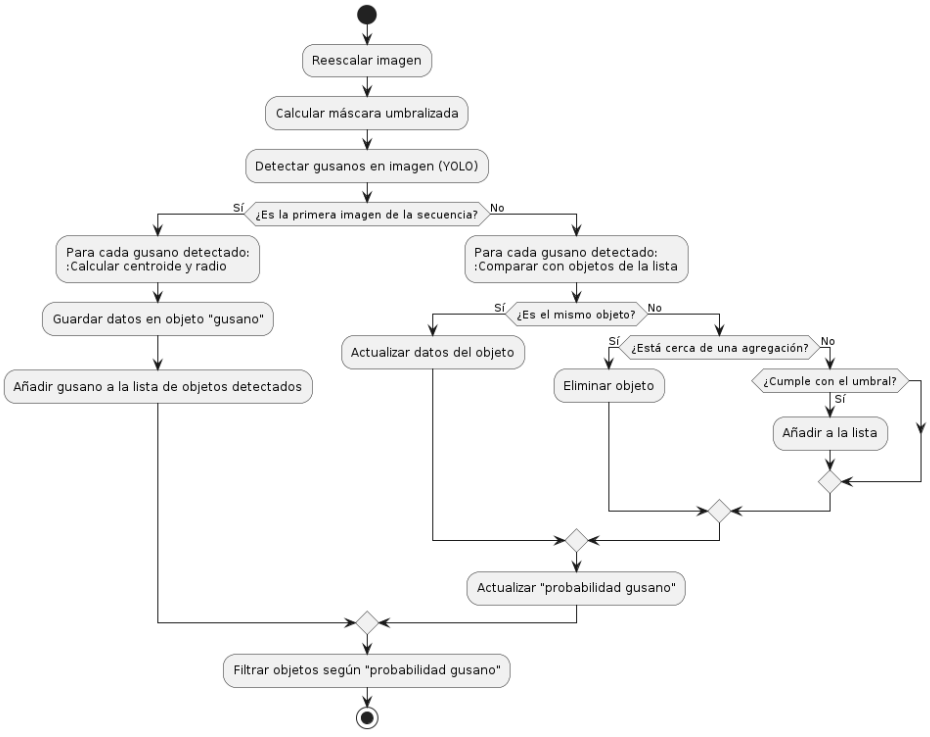

Figura 4.3.3: Diagrama de flujo de la implementación de la red YOLO en Python

Seguimiento y extracción de características de los C. elegans

Tras agregar todos los objetos detectados por la red neuronal, se llama a una función "repaso de objetos" la cual comprobará diversas cuestiones recorriendo todos los objetos de la lista. Lo primero es detectar si un nematodo de la lista cuyo radio es mejor al mínimo o no se ha movido en toda la secuencia, se encuentra en la imagen umbralizada, si esto es así, el objeto es eliminado de la lista.

También esta función comprueba si hay nematodos que estén demasiado cerca, es decir, que hay dos nematodos cuyas circunferencias mínimas estés solapadas. Si este es el caso, el sistema comprueba mediante la máscara de imagen si se están tocando o no. Si los nematodos no se están tocando, simplemente se calcula qué nematodo es cada uno por proximidad.

Si fuese el caso de que ambos nematodos efectivamente se están tocando, se llamaría a la función "añadir agregación", la cual se encargará de gestionar los objetos de tipo "agregacion", los cuales serán usados para resolver estas situaciones, además de gestionar la lista que guarda este tipo de objetos.

También esta función se encarga de actualizar las coordenadas de los nematodos que no hayan sido detectados por la red neuronal YOLO en la imagen actual, comprobando su ubicación en la máscara de imagen y actualizándola.

Además, esta función también se encarga de gestionar las agregaciones las cuales no han podido ser resueltas mediante el uso de hilos de ejecución y de la red neuronal U-Net, de esta forma, obteniendo los últimos centroides de los esqueletos calculados por la U-Net, se comparan con los centroides de los esqueletos de los nematodos en el instante que se volvieron a separar, de forma que se asignan los id de nematodo correspondiente a cada uno por proximidad. Este método solo se usa con los hilos de ejecución que hayan devuelto un código de error.

También esta función gestiona los datos de la imagen anterior a la actual, de manera que, por ejemplo, si se produce una agregación, se realizará una esqueletización de los nematodos que estén agregados, pero esta esqueletización se llevará a cabo en la imagen anterior, cuando estos nematodos no estaban solapados aún.

Esqueletizar un objeto consiste en obtener una máscara de imagen con un conjunto de píxeles los cuales forman una línea de un píxel de grosor que atraviese dicho objeto de un extremo a otro, para el caso de los C. elegans consiste en obtener la máscara de imagen del C. elegan correspondiente y filtrarla hasta que sea de un píxel de ancho. Estos esqueletos serán cruciales a la hora de resolver las agregaciones

También se extraen los esqueletos de los nematodos en el instante en el que los nematodos de la agregación se separan, de manera que se puedan comparar con los resultados obtenidos por la red U-Net. Ambos esqueletos se guardan como campos del objeto gusano correspondiente en la lista.

Además de esto, en todo momento se calculan y actualizan datos del nematodo como el tiempo que lleva quieto, los id de los nematodos con los que se encuentra agregados, si se encuentra en una agregación y los datos de la elipse que lo encierra. Estos datos serán muy importantes a la hora de resolver las agregaciones.

El funcionamiento anteriormente descrito que compone la siguiente parte del código desarrollado en Python se muestra explicado en el siguiente diagrama de flujo, el cual parte del diagrama de flujo anteriormente descrito:

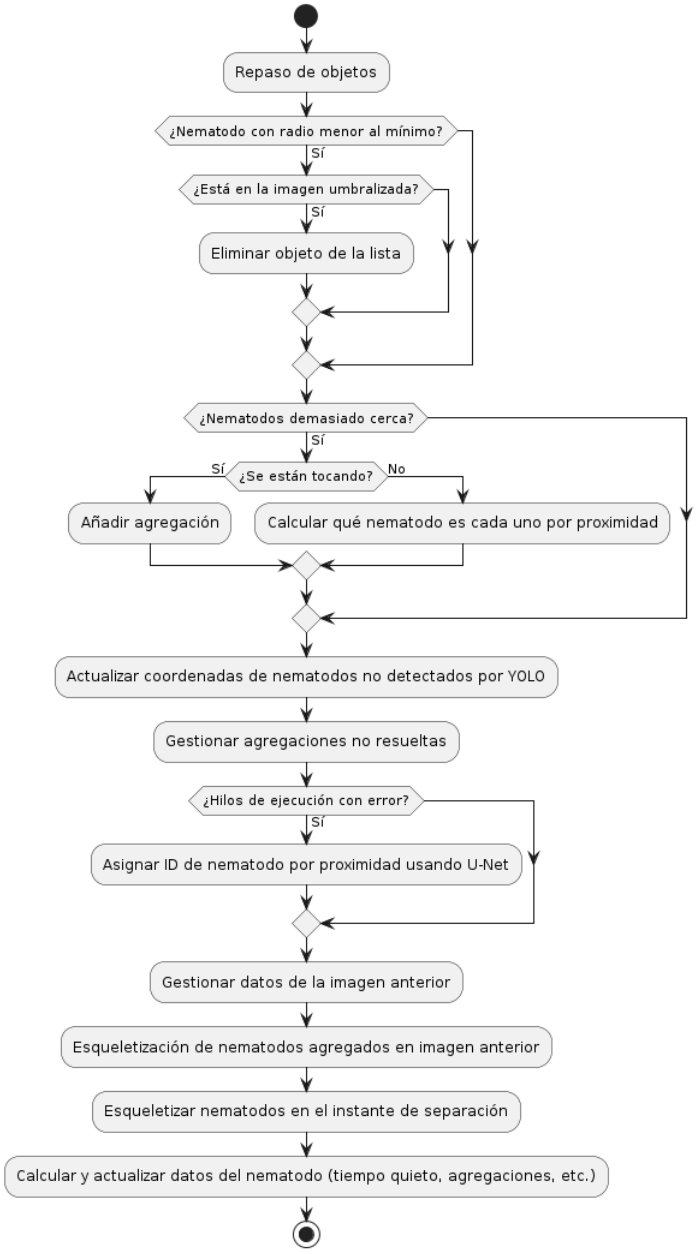

Figura 4.3.4: Diagrama de flujo del repaso de objetos detectados por la YOLO en Python

4.2.4 Uso de hilos

Como ya se ha comentado anteriormente, uno de los grandes problemas que debe de afrontar este sistema es la preservación de los id en los C. elegans cuando sucede una agregación, es decir, poder identificar qué nematodo es cada uno si estos se cruzan.

Para ello se ha optado por una solución que involucra una nueva red neuronal, en este caso una U-Net, la cual permite obtener el esqueleto de un C.elegan cuando se encuentra agregado con otro. Para ello, la red neuronal, utiliza como referencia una ventana de 100x100 píxeles en el instante actual con el C. elegan a esqueletizar en el centro de la ventana, otra ventana igual, pero en la imagen anterior y el esqueleto del C. elegan en la imagen anterior

Sin embargo, los resultados obtenidos por la U-Net no representan exactamente al esqueleto del nematodo en cuestión. Por ello el sistema calcula todos los posibles esqueletos que puede tener el nematodo en dicha imagen y filtra el correcto comparando con los resultados obtenidos por la U-Net. Como estos cálculos resultan muy costosos dependiendo de la agregación, se ha optado por utilizar hilos de ejecución para lograr que el programa tarde menos de 1 segundo en procesar cada imagen.

Implementación de hilos en el sistema

Siguiendo la lógica descrita anteriormente, el sistema crea un hilo de ejecución cada vez que se detecte una agregación entre C. elegans. Todos los hilos de ejecución llaman a la misma función “función hilo agregaciones” para realizar el procedimiento de resolución de esqueletos. Para cada instante en el que los nematodos se encuentren agregados, esta función comprobará si efectivamente siguen agregados, en ese caso, hará uso de la U-Net para calcular los esqueletos de los nematodos agregados en esa imagen. Si la función detecta que los nematodos ya no están agregados, el sistema comparará los esqueletos reales de los C. elegan con los predichos por la U-Net para discernir que nematodo es cada uno.

Cada hilo tiene un identificador propio, el cual es un número entero único e irrepetible en el programa, además de una variable “k” que indica el número de la imagen en la secuencia de imágenes que está procesando, estos datos van a ser de utilidad para la correcta gestión de las imágenes a la hora de resolver agregaciones.

Cada agregación detectada en el hilo principal de ejecución se guarda como un objeto de tipo “agregación” en una “lista agregaciones”, estos objetos guardan datos como el id de los nematodos que se han agregado, el número de imagen en el que se ha producido la agregación, el centroide de esta y una variable booleana que indica si se ha resuelto o no.

Cuando se detecta una agregación en el hilo principal, se comprueba si existe otro objeto de tipo agregación en la lista con los mismos id y que no está resueto, si éste es el caso, no se realiza ninguna operación, pero en caso contrario, el objeto agregación se añade a la lista, se crea el evento que corresponde al hilo junto con su id, se crea el hilo y por último se ejecuta indicando los valores de la agregación correspondiente, además de la imagen donde se ha producido la agregación y la imagen anterior a ésta en la secuencia de imágenes.

Para el correcto uso de las imágenes a la hora de resolver las agregaciones, se ha implementado un buffer de imágenes en el cual se van almacenando las imágenes en las

que cualquier agregación se esté produciendo, junto al número de imagen que le corresponde en la secuencia. De esta forma, cuando la función deja de hacer uso de dicha imagen, comprueba si algún otro hilo podría llegar a necesitarla y en caso contrario la elimina del buffer.

Además de esto, en el hilo principal de ejecución, mientras no se detecte que los nematodos de la agregación en cuestión se han separado, la imagen recibida se guarda en el buffer junto a su número de imagen. Cuando el hilo principal detecta que los C. elegans de la agregación se han separado, calcula los esqueletos de cada uno y los guarda como un campo en el objeto que les representa en la lista, además de esto, indica al hilo correspondiente a esa agregación que los esqueletos reales de los C. elegans están calculados. Esto lo realiza mediante el uso de eventos.

El proceso descrito anteriormente para la creación de los hilos y gestión de los datos que deben devolver se encuentran descritos en el siguiente diagrama de flujo:

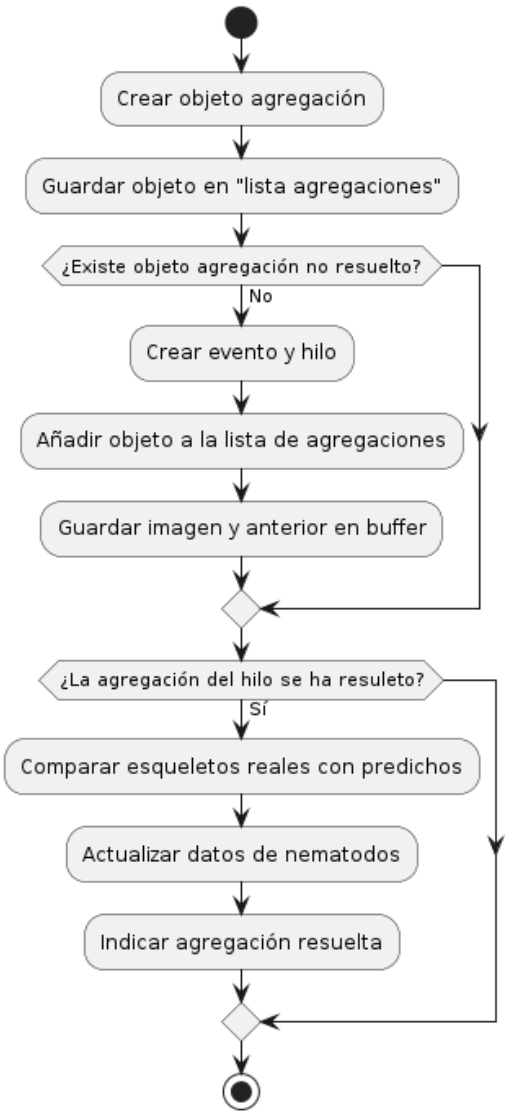

Figura 4.3.5: Diagrama de flujo de la gestión de hilos en Python

Implementación de eventos y semáforos

Cada hilo de ejecución del sistema debe leer del buffer compartido la imagen correspondiente y eliminar las imágenes que no sean necesarias para el sistema, por ello, al haber más de un hilo de ejecución del buffer leyendo, añadiendo y eliminando datos de manera asíncrona, se ha hecho uso de semáforos para la correcta gestión de los datos del buffer.

No solo el buffer es el único elemento compartido entre hilos, sino que la lista de objetos detectados, con los objetos "gusano" que corresponde a cada C. elegan, también se modifica por todos los hilos de manera asíncrona, ya que al resolverse la agregación una vez el hilo ha finalizado, los datos de los C. elegans cambian y eso debe verse reflejado en la lista.

De esta forma se ha implementado como variable global un semáforo con un valor de 1, de forma que actúa como un mutex, de manera que cada vez que se vaya a insertar una imagen en el buffer, eliminar una imagen del mismo o modificar algún valor de cualquier objeto "gusano" de la lista, se realizará una operación "P" sobre el semáforo, decrementando su valor de manera que ningún otro hilo de ejecución podrá acceder a estas secciones críticas, es decir, no podrá modificar o leer de estas dos variables compartidas. Una vez se hayan hecho las operaciones pertinentes, es decir, se haya salido de la sección crítica, se realizará una operación "V" sobre el semáforo, incrementando su valor para que otro hilo de ejecución pueda entrar en esta sección crítica.

También se ha tenido en cuenta el caso de que se quiera escribir en el buffer en el momento en el que éste se encuentre lleno o que se quiera borrar del mismo un dato cuando éste se encuentra vacío. Para gestionar estos casos se hace uso de otro semáforo implementado como un atributo del propio buffer, de forma que decremente su valor si el buffer está lleno al escribir o vacío al eliminar un dato y se aumente el valor del semáforo si la acción de escribir o borrar se ha podido llevar a cabo.

Además de lo comentado hasta ahora, todos los hilos necesitan comparar los resultados que han obtenido gracias a la U-Net con los esqueletos reales de los C. elegans una vez se hayan separado, para poder avisar al hilo correspondiente de que el hilo principal ha esqueletizado los C. elegans, se hace uso de un evento, asociado al hilo de ejecución correspondiente por medio de su id.

Para realizar esta operación de manera correcta, se ha hecho uso de una lista de tuplas que almacena en cada una el evento correspondiente a un hilo de ejecución y su id, de esta forma, cuando una agregación se haya separado y el hilo principal lo detecte, buscará en esa lista y activará el evento correspondiente. Los hilos de ejecución no podrán comparar sus resultados con los esqueletos reales hasta que su evento correspondiente haya sido activado, en ese momento actualizarán los datos de los nematodos pertinentes en el array y el hilo se cerrará.

Implementación de la U-Net

A continuación, se va a explicar en detalle el uso que se le ha dado a la U-Net en este sistema, así como todo el procedimiento que se lleva a cabo para calcular los esqueletos de los nematodos en cada instante.

Cada hilo de ejecución lee la imagen correspondiente del buffer, calcula su máscara y comprueba con OpenCV si los nematodos de la imagen se están tocando, si este es el caso se llamará a la función “obtener esqueletos gusanos”, la cual devuelve los esqueletos de los nematodos para esa imagen, en el instante inicial se asume que los nematodos efectivamente se están tocando, tras obtener los esqueletos predichos, éstos se guardan como los esqueletos de la imagen anterior a la vez que la imagen del instante actual se guarda como la imagen anterior, tras esto el hilo lee la siguiente imagen del buffer y repite el proceso.

La función “obtener esqueletos gusanos” recorta una ventana de imagen para cada uno de los nematodos de la agregación en la imagen anterior, centrando cada ventana en uno de los nematodos, posteriormente recorta la misma ventana con las mismas coordenadas en la imagen actual. Estos datos se guardan en un objeto de tipo “Worm” con todo lo necesario para calcular el esqueleto del C. elegan al que corresponda, también se guardan los esqueletos de ambos nematodos y las coordenadas de las esquinas de las ventanas.

Haciendo uso de estas ventanas, calcula 3 vectores que serán usados por la U-Net, un vector “C” que contiene para cada píxel del esqueleto del nematodo su color en escala de grises, otro vector “W” que contiene para cada píxel del esqueleto del nematodo su distancia mínima hasta el final de la máscara, obteniendo así la anchura del nematodo en cada píxel de su esqueleto; y un último vector “L” de solo una posición que contiene el largo en píxeles del esqueleto del nematodo. Estos vectores se guardarán como campos del objeto “Worm”

Con todos estos datos, la función realiza una llamada a la U-Net, la cual devolverá el esqueleto del nematodo predicho por la misma, el cual se guardará como último campo del objeto “Worm”. Con todos estos datos la función predice los posibles esqueletos que puede tener el nematodo a raíz de su esqueleto en la imagen anterior y filtra el esqueleto que más píxeles tenga en común con los resultados obtenidos mediante la U-Net. Este esqueleto es el que devuelve la función

Si se produce un error al esqueletizar un nematodo en la agregación o se predicen demasiados posibles esqueletos para uno de los nematodos de la agregación, se da por hecho que la ventana de imagen está recortada o la esqueletización se ha hecho de manera errónea, por lo que se devuelve un mensaje de error.

Cuando la función devuelve un mensaje de error, el hilo modifica el valor del campo “Error Unet” de los nematodos de la agregación a “True” y guarda el centroide del esqueleto predicho para cada nematodo, tras esto el hilo se hierra. Cuando el programa principal al hacer un repaso de objetos detecta que se ha producido un error, compara los centroides de los esqueletos cuando los nematodos se separan con los centroides obtenidos por el hilo y decide qué nematodo es cada uno mediante proximidad, tras esto se actualiza la agregación de la lista para indicar que ha sido resuelta.

Si no se produce ningún error, el hilo de la agregación debe de seguir calculando los esqueletos de cada nematodo hasta que estos se separen, en ese momento se espera a que ocurra el evento que indica que el hilo principal también ha detectado que los nematodos se han separado (esto debería ocurrir en la misma imagen, por lo tanto, en la mayoría de los casos, el hilo principal habrá detectado que se han separado los C. elegans y habrá activado el evento antes de que el hilo se ponga a esperar).

Cuando ocurre el evento se comparan los esqueletos reales de los nematodos con los obtenidos mediante el proceso que ha llevado a cabo el hilo, se calcula su similitud y proximidad y de esta forma se actualizan los nematodos, terminando el hilo actualizando el valor correspondiente en la variable agregación de la lista y cerrando el hilo.

Siguiendo la lógica descrita anteriormente, el sistema crea un hilo de ejecución cada vez que se detecta una agregación entre C. elegans. Todos los hilos de ejecución llaman a la misma función "*funcion _hilo_agregaciones"* para realizar el procedimiento de resolución de esqueletos. Para cada instante en el que los nematodos se encuentren agregados, esta función comprobará si efectivamente siguen agregados. El proceso dentro de esta función se detalla en el siguiente flujograma:

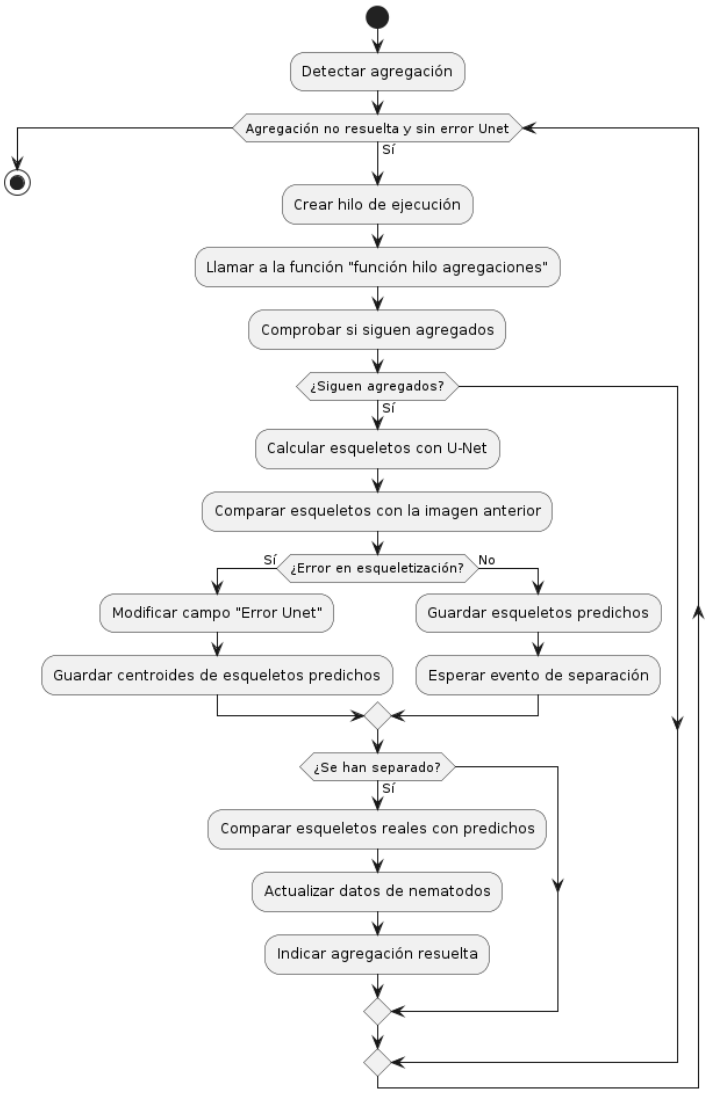

Figura 4.3.6: Diagrama de flujo de la función "*funcion_hilo_agregaciones*"

4.3 Validación de resultados con imágenes reales

Este apartado de la memoria está dedicado a todas las pruebas, test y demás procesos que se han llevado a cabo para validar el correcto funcionamiento del programa, para ello lo primero que se deben tener en cuenta son los diferentes aspectos que debe cumplir para poder decir que funciona de manera adecuada, los cuales se listan a continuación:

- El sistema debe detectar todos los nematodos de la placa o en su defecto la mayor parte de éstos
- El sistema debe ofrecer resultados de coordenadas de cada C. elegan con un periodo inferior a un segundo
- Al producirse una agregación el sistema debe ser capaz de detectarla y resolver qué nematodo es cada uno cuando los C. elegans agregados se separen
- Se debe ofrecer información sobre si cada nematodo se encuentra en una agregación o no

De esta forma podemos inferir que tanto la red neuronal YOLO como la red neuronal U-Net deben ofrecer resultados con un mínimo de calidad por lo que ambas redes neuronales deben ser validadas con diversos criterios que se mostrarán más adelante en la memoria.

Además de esto, el sistema en sí programado en Python debe cumplir con una serie de requisitos y pasar diferentes pruebas para determinar su correcto funcionamiento, todo este proceso de validación será documentado y mostrado en los posteriores apartados de este documento.

4.3.1 Datos de validación de las redes neuronales utilizadas

La base de este sistema de detección y seguimiento de C. elegans mediante visión artificial son las redes neuronales utilizadas para este propósito. En primera instancia la red neuronal YOLO la cual está dedicada a detectar los C. elegans que se encuentran en cada imagen y en segunda instancia la red neuronal U-Net la cual se encarga de predecir el siguiente esqueleto de cada C. elegan en una imagen cuando se produce una agregación para poder diferenciar cual es cada uno al separarse los nematodos.

Al entrenar ambos tipos de redes neuronales se obtienen diversos parámetros los cuales se pueden comparar para saber qué red neuronal de cada tipo es la más adecuada para el sistema que se está diseñando. Además de esto, validando cada red neuronal con imágenes reales, se pueden realizar diversas pruebas y mediciones para tener otra perspectiva de cómo funciona cada red neuronal con este tipo de imágenes.

De esta forma, antes de integrar una red neuronal en el sistema, se han llevado a cabo diferentes comparaciones y pruebas para averiguar si estas redes van a ser adecuadas para el sistema y qué tipo de resultados van a aportar a la estructura del proyecto.

Validación de la red neuronal YOLO

La red YOLO Segmentation que se ha utilizado en el sistema, la cual ha sido utilizada para detectar a los C. elegans sobre la imagen, ha sido la red elegida de entre varias redes

neuronales debido a su rendimiento medido mediante diferentes pruebas con imágenes reales. Al igual que varias otras redes YOLO que se han entrenado con el propósito de ser integradas en el sistema, se han medido los resultados de la red mediante pruebas con imágenes reales similares a las que debe recibir el sistema cuando esté finalmente implementado.

A continuación, se explica detalladamente este proceso de validación y comparación de las diferentes redes neuronales entrenadas para ser utilizadas en el sistema.

Conjunto de Datos Utilizados:

Los datos utilizados para validar las redes neuronales son conjuntos de imágenes tomadas con un periodo de un segundo de diferentes experimentos realizados con C. Elegans. Estos conjuntos de datos simulan al publicador de ROS2 que debe enviar una imagen por segundo al sistema cuando esté finalmente implementado.

Algunas imágenes utilizadas para la validación se muestran a continuación:

Se han utilizado 4 conjuntos grandes de datos, un conjunto con 266 imágenes con 60 nematodos, otro conjunto de 350 imágenes con 20 nematodos, y unos conjuntos tercero y cuarto con 45 nematodos y 60 y 49 imágenes respectivamente.

Como se puede observar, todas las imágenes del conjunto cuentan con diversas características propias de las imágenes que debe recibir el sistema como son, arrugas en la placa, iluminación irregular y un puntero laser que se mueve por la placa.

Métricas de Evaluación:

A continuación, se listan las diferentes métricas utilizadas para medir el rendimiento de la red neuronal en orden de importancia para facilitar la comprensión del proceso de comparación de las redes neuronales:

- Velocidad: Tiempo en segundos que tarda la red en ofrecer resultados.
- Precisión: Porcentaje de detecciones correctas sobre el total de detecciones de la red
- Recall: Porcentaje de detecciones correctas sobre el total de nematodos reales

Proceso de Validación:

Este proceso de validación se ha llevado a cabo alimentando a todas las redes neuronales que se han evaluado con cada conjunto de datos y obteniendo la media de cada métrica obtenida en los 4 conjuntos. Esto se ha hecho mediante el uso de un script de Python que calcula las diferentes métricas. Con estas métricas tomadas para los 4 conjuntos de datos, se calculó la media de cada métrica y los datos fueron almacenados en una lista, de manera que se pudieran comparar de manera sencilla

Comparación entre Modelos:

	Train_36 (Red utilizada)	Train_29	Train_34	Train_42	Train_16
Velocidad	0.083s	0.13s	0.15s	0.13s	0.114s
Precisión	0.77	0.78	0.73	0.77	0.73
Recall	0.92	0.84	0.88	0.83	0.85

Resultados y Análisis de la red utilizada:

De la red neuronal utilizada finalmente se han obtenido las siguientes figuras mediante la validación de esta con imágenes reales.

La siguiente gráfica es la curva Precisión-Exhaustividad (o recall) la cual muestra cómo la red neuronal equilibra la proporción de predicciones correctas (precisión) con la proporción de ejemplos positivos identificados correctamente (exhaustividad) aumentando poco a poco un umbral de confianza. Como se puede observar en la figura a continuación, al aumentar dicho umbral, el valor de precision se aumenta, sin embargo, el valor de exhaustividad es bajo, al aumentar este valor disminuyendo el umbral, la precisión disminuye hasta que llega un punto que baja de manera agresiva.

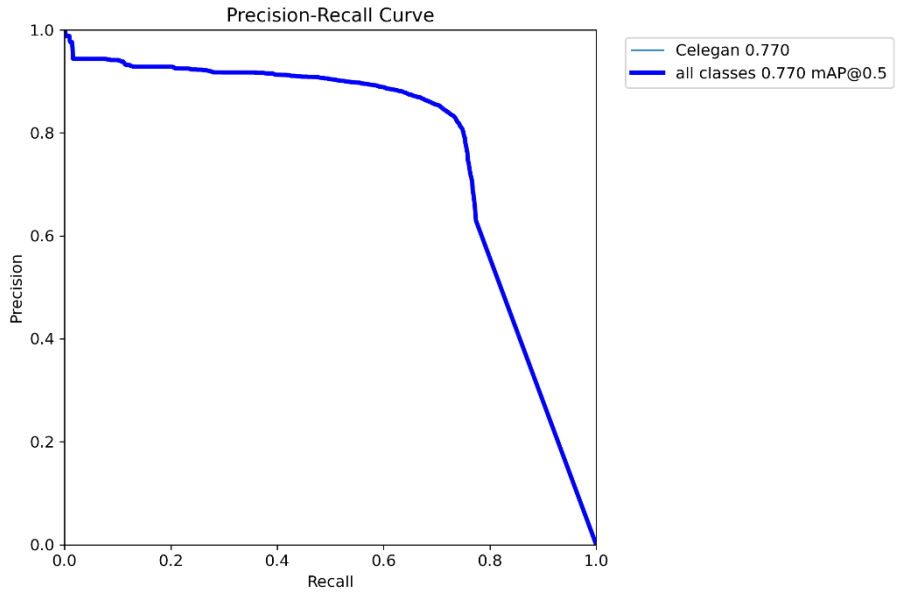

Figura 4.4.1: Curva de Precision-Exhaustividad de la red YOLO

La siguiente figura muestra la matriz de confusión de la YOLO utilizando un conjunto de imágenes reales.

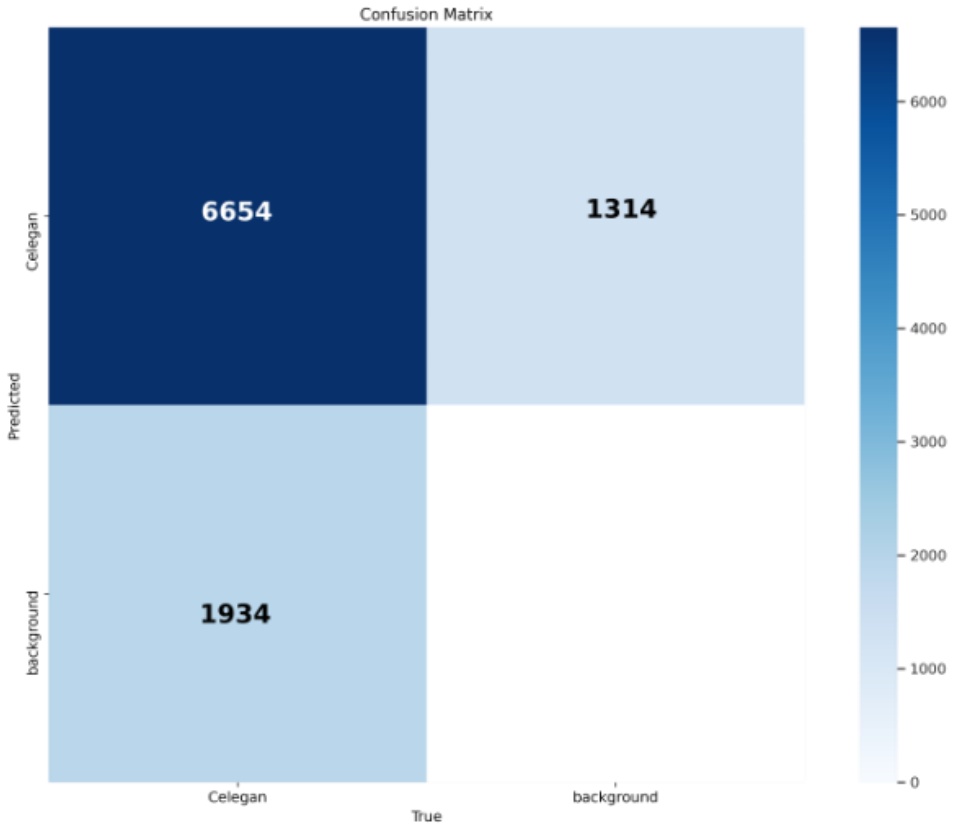

Figura 4.4.2: Matriz de confusión de la red YOLO con imagen real

Teniendo en cuenta estos resultados y las conclusiones extraídas al examinar los resultados de la red neuronal sobre diversas fotografías reales, podemos extraer diferentes características de esta red neuronal:

- La red ofrece resultados con un tiempo menor a 0.1 segundos
- Tanto en precisión como en recall la red ofrece un buen resultado
- La red neuronal utilizada ofrece resultados con una tasa de verdaderos positivos (o recall) bastante alta, por lo que la mayoría de C. elegans que se detectan en las imágenes son C. elegans realmente.
- La red neuronal consigue identificar a la mayoría de los nematodos en las imágenes reales.
- Los nematodos no detectados en una imagen en un instante, la mayoría de las veces son detectados correctamente en el siguiente instante.
- Tras un recorrido de varias imágenes se puede observar que se han detectado la gran mayoría de los nematodos al menos una vez
- El ruido que detecta como C. elegan tiene un área bastante inferior en comparación al área media de los C. elegan detectados, por lo que es fácil realizar un filtrado.

Validación de la red neuronal U-Net

La red neuronal U-Net ha sido utilizada en el sistema para extraer los esqueletos de los nematodos en el momento en que se produzca una agregación, de esta manera, cuando los nematodos se separen, se puede saber con certeza qué nematodo es cada cual y de esta forma no perder el seguimiento de cada nematodo.

A continuación, se explica en detalle qué proceso se ha seguido para validar estas redes neuronales.

Conjunto de Datos Utilizados:

Los datos utilizados para validar las redes neuronales son ventanas de imagen de 100x100 pixeles de las diferentes agregaciones que se han producido en los conjuntos de imágenes reales.

Algunas imágenes utilizadas para la validación se muestran a continuación:

Métricas de Evaluación:

A continuación, se listan las diferentes métricas utilizadas para medir el rendimiento de la U-Net:

- Precisión: Porcentaje píxeles bien detectados de un esqueleto sobre los pixeles del esqueleto detectado
- Recall: Porcentaje de pixeles bien detectados sobre los pixeles del esqueleto real
- Velocidad: Tiempo en segundos que tarda la red en ofrecer resultados.
- IoU (Intersection over Union): El IoU calcula la superposición entre la caja delimitadora predicha por el modelo y la caja delimitadora real del objeto.
- Dice Coefficient: Mide la superposición entre la segmentación predicha y la real.
- Hausdorff Distance: Mide la máxima distancia del conjunto de predicciones al conjunto real.

Proceso de Validación:

Este proceso de validación se ha llevado a cabo alimentando a todas las redes neuronales que se han evaluado con las distintas agregaciones y obteniendo la media de cada métrica. Los datos se han almacenado en una tabla para su posterior comparación.

Comparación entre Modelos:

	Unet_0	**Unet_1 (Red utilizada)**	**Unet_2**	**Unet_3**
Precisión	0. 5912	0.8968	0.7971	0.42
Recall	0.6656	0.8678	0.7595	0.52
Velocidad	0.9078s	0.4296s	0.5275s	0.5488s

Resultados y Análisis de la red utilizada:

Teniendo en cuenta estos resultados, en la siguiente tabla se muestran los coeficientes obtenidos con imagen real para evaluar la U-Net que se ha implementado finalmente en el sistema:

Dice Coefficient	0.8806
IoU	0.7971
Precisión	0.8968
Recall	0.8678
Hausdorff Distance	1.8711

También se han obtenido las gráficas de ROC y Precision-Exhaustividad (Recall) utilizando el mismo conjunto de imágenes reales.

La curva precisión-exhaustividad (recall) muestra cómo un modelo de clasificación binaria equilibra la proporción de predicciones correctas (precisión) con la proporción de ejemplos positivos identificados correctamente (exhaustividad) aumentando poco a poco un umbral de discretización. Como se puede observar en la figura a continuación, al aumentar dicho umbral, el valor de precision se aumenta, sin embargo, el valor de exhaustividad es bajo, al aumentar este valor disminuyendo el umbral, la exhaustividad aumenta, pero disminuye la precisión.

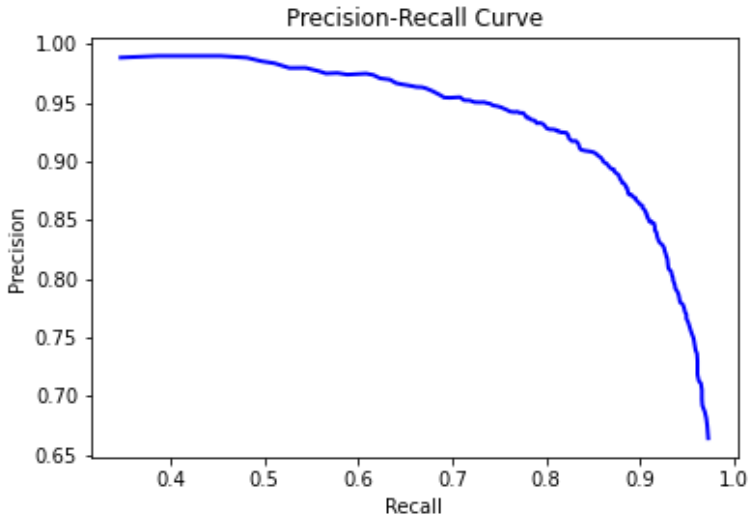

Figura 4.4.3: Curva de Precision-Exhaustividad de la U-Net

Por otro lado, la curva ROC (Receiver Operating Characteristic) muestra el rendimiento de un modelo de clasificación binaria a diferentes niveles de umbral de decisión. En esta curva, el eje X representa la tasa de falsos positivos (FPR), mientras que el eje Y representa la tasa de verdaderos positivos (TPR), también conocida como sensibilidad. A medida que ajustamos el umbral de decisión del modelo, podemos observar cómo varían estas tasas. Idealmente, buscamos un área bajo la curva (AUC) cercana a 1, lo que indica un mejor

rendimiento del modelo en la clasificación de las clases positiva y negativa, en el caso de este proyecto, la figura se muestra a continuación, podemos observar que el AUC es de 0.81, un valor válido para lo requerido en este proyecto.

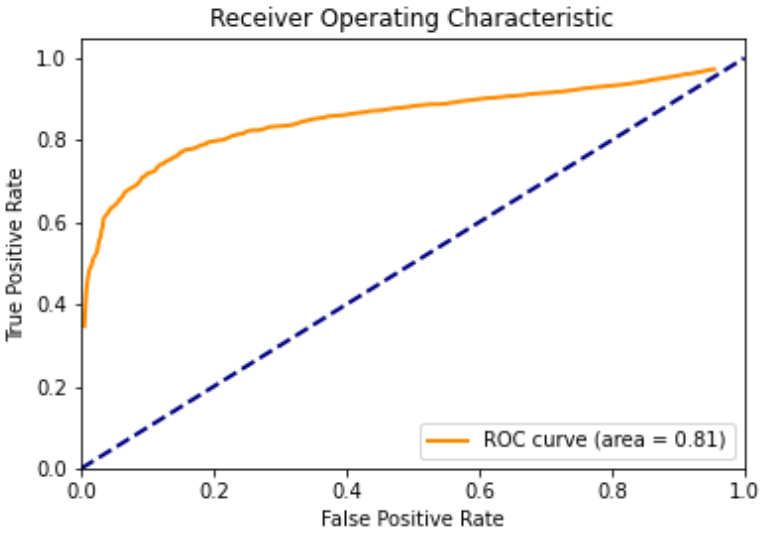

Figura 4.4.4 Curva de ROC de la U-Net

También se ha logrado obtener la siguiente matriz de confusión de la U-Net con el mismo conjunto de datos con imágenes reales:

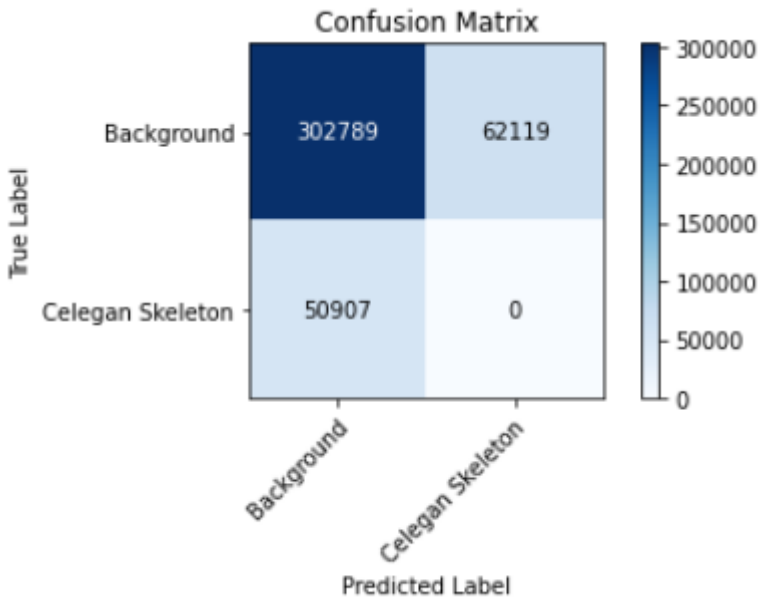

Figura 4.4.4 Matriz de confusió de la U-Net con imagen real

De los resultados expuestos en este subapartado, así como analizando los resultados que la red proporciona para las diferentes imágenes que son procesadas por el sistema completo, se pueden extraer las siguientes conclusiones:

- La velocidad de respuesta de la red neuronal es menor a medio segundo, lo que se traduce en menos carga computacional para el sistema
- Los resultados de los esqueletos que predice la red neuronal coinciden en gran parte con la realidad
- Los esqueletos mal predichos por la red neuronal son en gran parte debidos a una mala esqueletización del C. elegan en el instante anterior
- Pese a que una parte de los esqueletos predichos no coinciden con la realidad, siguen siendo de utilidad a la hora de filtrar entre los diferentes esqueletos posibles que puede adoptar el C.elegan en el siguiente instante, lo cual es el resultado que buscaba el sistema.

4.3.2 Validación del sistema completo

Con el sistema ya en pleno funcionamiento, se han realizado una serie de pruebas para medir parámetros de interés de este y así poder evaluar su comportamiento, los cuales han sido medidos realizando pruebas con el sistema utilizando sets de imágenes reales de experimentos con C. elegans, estos sets de imágenes componen una secuencia ordenada en el tiempo la cual muestra el comportamiento de los C. elegans en la placa. Habiendo tomado una imagen de la placa Petri por segundo, estos sets de imágenes son similares a la entrada que recibiría el sistema de un nodo publicador en ROS2 que enviase una imagen por segundo.

Los resultados que se pueden extraer de esta validación son imágenes de la placa de Petri a las cuales se les ha añadido la circunferencia que envuelve a cada nematodo junto a su identificador, de esta forma, pasando de imagen a imagen se puede observar cómo de correcto es el seguimiento de cada uno de los nematodos y su comportamiento al producirse una agregación, a continuación, se muestra un ejemplo de una imagen real de una placa de Petri con los nematodos detectados:

Figura 4.4.5: Resultados del programa sobre una placa de petri real

Para poder visulizar mejor los resultados, a continuación, se muestran ampliaciones de la placa en diversas zonas con sus C. Elegans detectados y etiquetados por el sistema:

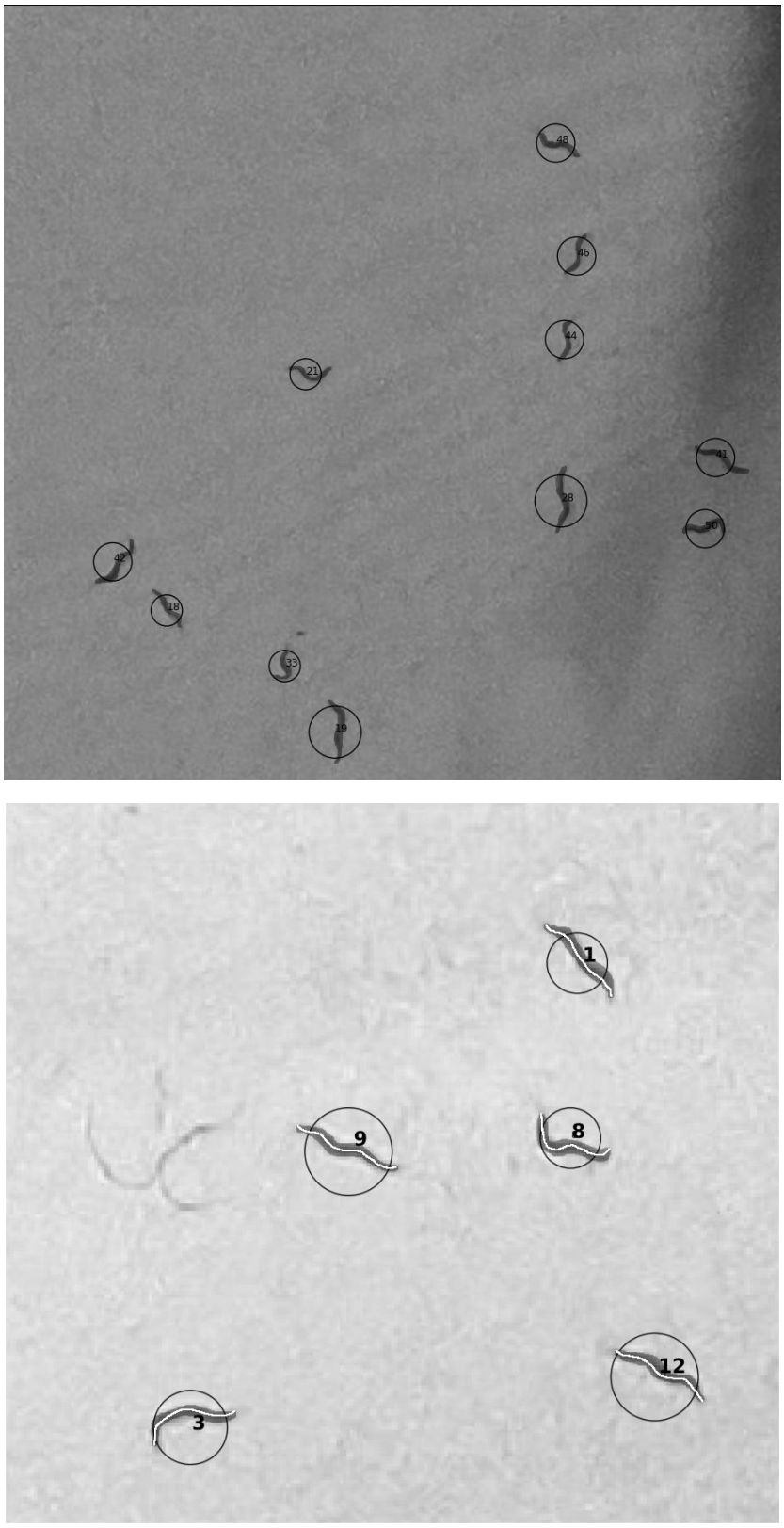

Figura 4.4.6: Resultados del programa sobre diversas zonas de una placa de petri real

Al producirse una agregación, podemos comprobar de manera satisfactoria que el sistema, en un inicio indica que los nematodos se mantienen en el lugar de la agregación, tras separarse indican cuales son los posibles id de cada nematodo y al completarse el hilo que gestiona la agregación, se indica qué nematodo es cada uno de manera correcta, este proceso se muestra en las imágenes a continuación:

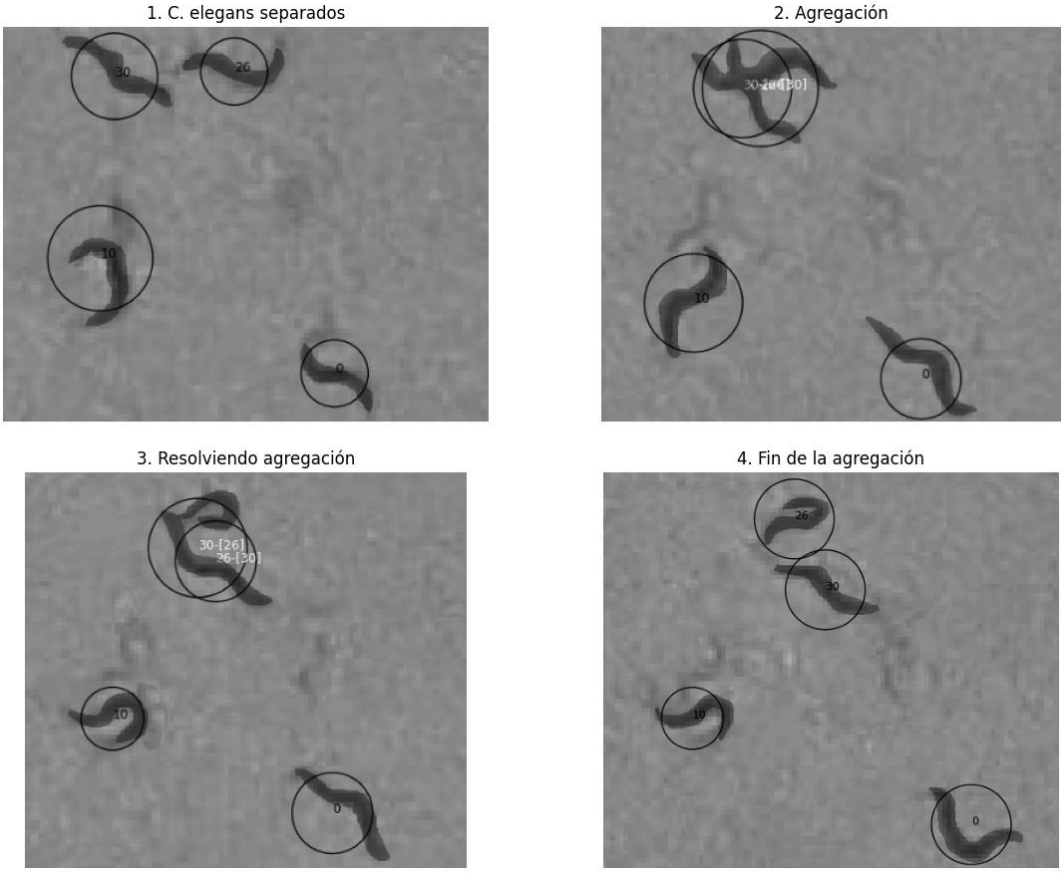

Figura 4.4.7: Resultados del programa sobre una agregación real de nematodos

Los diferentes parámetros que se han medido en el sistema se listan a continuación:

- Porcentaje de nematodos encontrados: Porcentaje de los nematodos que ha detectado el sistema sobre los nematodos que se encuentran en la placa de Petri
- Nematodos perdidos: Nematodos que han sido detectados en la red neuronal y se ha perdido su seguimiento en el programa
- Agregaciones no resueltas: Numero de agregaciones que no han sido resueltas satisfactoriamente por la U-Net
- Porcentaje de agregaciones correctas: Porcentaje de agregaciones que la U-Net ha resuelto correctamente frente a todas las que han ocurrido
- Tiempo de procesamiento máximo
- Tiempo de procesamiento medio
- Tiempo de procesamiento medio si no hay hilos en segundo plano
- Tiempo de procesamiento medio si hay hilos en segundo plano

En la siguiente tabla se pueden comprobar el valor de cada uno de estos parámetros en el sistema programado:

Parámetro	Medida
Porcentaje de nematodos encontrados	93.33%
Porcentaje de nematodos perdidos	6.67%
Porcentaje de agregaciones correctas:	87.50%
Porcentaje de agregaciones no resueltas:	12.50%
Tiempo de procesamiento máximo	0.5664s
Tiempo de procesamiento medio	0.3243s
Tiempo de procesamiento sin hilos	0.3051s
Tiempo de procesamiento con hilos	0.3369s

Como se puede observar, utilizando una secuencia de imágenes reales tomadas desde cámaras "macro", el sistema ofrece las coordenadas de todos los C. elegans de la placa de Petri de manera eficiente, con un tiempo de respuesta menor a un segundo y resolviendo en su mayoría las agregaciones de C. elegans que se producen a lo largo de la secuencia.

Capítulo 5. Conclusiones

En este Trabajo Final de Máster (TFM), se ha desarrollado, implementado y validado un sistema de visión artificial basado en redes neuronales para el seguimiento de la posición de C. elegans sobre una placa de Petri. Este sistema se ha diseñado para automatizar ensayos biológicos como lifespan y healthspan. A continuación, se presentan las conclusiones, destacando cómo se han completado los objetivos principales y secundarios del proyecto:

Objetivos Principales

Detección y Coordenadas de C. elegans

El primer objetivo principal, que consiste en indicar las coordenadas x e y de cada C. elegan en la placa de Petri, se ha completado con éxito. Para ello, se ha utilizado YOLO Segmentación, un modelo de red neuronal que permite detectar y segmentar los nematodos en cada imagen capturada. El sistema utiliza diversas funciones programadas en Python, utilizando librerías como OpenCV, para procesar cada imagen y devolver una lista con las coordenadas de cada nematodo. Este proceso asegura que, por cada imagen recibida, las coordenadas de los nematodos se identifican de manera precisa y eficiente.

Identificación Persistente de C. elegans

El segundo objetivo principal, que implica la asignación de un identificador único a cada C. elegan y la persistencia de dicha identificación a lo largo de una secuencia de imágenes, también se ha logrado satisfactoriamente. El sistema logra identificar de forma correcta sin confundir los identificadores de los C. elegans utilizando una U-Net, la cual actúa como una red neuronal de soporte, además de diversas funciones programadas en python que se ejecutan en paralelo al programa principal haciendo uso de hilos de ejecución. Esta funcionalidad es crucial para asegurar que los identificadores no se mezclen, permitiendo un seguimiento continuo y preciso de cada C. elegan a lo largo de todo el experimento.

Objetivos Secundarios

Uso de Datos de Imágenes Anteriores

Para cumplir con el subobjetivo de utilizar datos de secuencias de imágenes anteriores para mejorar la identificación, el sistema guarda diversas características correspondientes al C. elegan en cuestión en la imagen, tras ello las compara con la imagen siguiente para actualizar el objeto que representa al C. elegan anterior, de esta manera el comportamiento de este es monitorizado en toda la secuencia de imágenes

Procesamiento en Tiempo Real

Se ha logrado que el sistema procese cada imagen en menos de un segundo, gracias a la optimización del código en Python y el uso eficiente de librerías como OpenCV. Esta capacidad de procesamiento rápido es esencial para su aplicación en ensayos automatizados, permitiendo una respuesta con un tiempo menor a 1 segundo.

Detección Mediante Redes Neuronales

La detección de los C. elegans se ha llevado a cabo utilizando YOLO Segmentación, lo cual ha permitido una identificación precisa de los nematodos. Además, para resolver los casos de agregaciones de nematodos, se ha utilizado una red U-net adicional, mejorando la capacidad del sistema para manejar situaciones complejas y garantizando una segmentación adecuada en escenarios de alta densidad de nematodos.

Entrenamiento con Imágenes Simuladas

Las redes neuronales del sistema se han entrenado con imágenes simuladas de C. elegans sobre fondos predefinidos. Este enfoque ha sido clave para desarrollar modelos robustos y generalizables, que han demostrado un rendimiento eficaz cuando se han validado con imágenes reales. La utilización de datos simulados ha asegurado la versatilidad y precisión del sistema en condiciones reales.

Integración con ROS2

Finalmente, el sistema se ha integrado con éxito en un nodo de ROS2, permitiendo el envío continuo de los centroides de los C. elegans y la recepción de imágenes en tiempo real. Esta integración es fundamental para la aplicación del sistema en entornos de investigación automatizada, facilitando la monitorización y el análisis en tiempo real de los ensayos con C. elegans.

5.1 Trabajo futuro

Como posibles implementaciones futuras a este proyecto se barajan varias opciones como el uso de un sistema de tracking para detectar el láser que utiliza el sistema Multiview para indicar la posición de su cámara “macro” y de esta forma poder indicar la distancia entre el láser y el nematodo sobre el que se quiere realizar un seguimiento con esta cámara.

Además, se valora el uso de entrenar una red neuronal adicional para extraer diversas características de los C. elegans sobre la imagen, como pueden ser sus esqueletos, su trayectoria o si se encuentran vivos o muertos de manera más precisa.

Bibliografía

[1] TopDoctors, *C. elegans: ¿por qué este gusano es tan útil para la ciencia y la medicina?,* 2021. https://www.topdoctors.cl/articulos-medicos/c-elegans-por-que-este-gusano-es-tan-util-para-la-ciencia-y-la-medicina/ (último acceso: abril 2024)

[2] Simon Fraser University, *What is C. elegans?,* 2008 https://www.sfu.ca/biology/faculty/hutter/hutterlab/research/Celegans.html (último acceso: abril 2024)

[3] Hugo Andrade, Soraya Lucia Sinche, Pablo Hidalgo. *Descripción del funcionamiento de una red neuronal convolucional (CNN),* 2024 https://www.researchgate.net/figure/Figura-1-Descripcion-del-funcionamiento-de-una-red-neuronal-convolucional-CNN-10_fig1_348825166 (último acceso: abril 2024)

[4] Amit Chauhan, OpenCV: *Adaptive and Otsu Threshold in Image Processing with Python,* 2023. https://amitprius.medium.com/opencv-adaptive-and-otsu-threshold-in-image-processing-with-python-648b64129876 (último acceso: mayo 2024)

[5] Shoubhik Chandan Banerjee, Khursheed Ahmad Khan y Rati Sharma. *Deep-worm-tracker: Deep learning methods for accurate detection and tracking for behavioral studies in C. elegans,* 2023. https://www.sciencedirect.com/science/article/pii/S016815912300196X?via%3Dihub (último acceso: junio 2024)

[6] Santiago Escobar-Benavides, Antonio García-Garví y Pablo E. Layana-Castro. Towards generalization for Caenorhabditis elegans detection, 2023. https://www.csbj.org/action/showPdf?pii=S2001-0370%2823%2900352-5 (último acceso: junio 2024)

[7] Joan Carles Puchalt, Antonio-José Sánchez-Salmerón, Eugenio Ivorra, Silvia Llopis, Roberto Martínez y Patricia Martorell. *Small fexible automated system for monitoring Caenorhabditis elegans lifespan based on active vision and image processing techniques*, 2021 https://riunet.upv.es/bitstream/handle/10251/183627/Puchalt-RodriguezSanchezIvorra%20-%20Small%20flexible%20automated%20system%20for%20monitoring%20Caenorhabditis%20el....pdf?sequence=1&isAllowed=y (último acceso: junio 2024)

[8] Changjoo Park y Jinseop S. Kim, *Caenorhabditis elegans Connectomes of both Sexes as Image Classifiers*, 2023 https://www.en-journal.org/journal/view.html?uid=624 (último acceso: mayo 2024)

[9] OpenCV, *OpenCV Documentation,* 2024 *https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html* (último acceso: junio 2024)

[10] Python, Python Documentation. Threading, 2024 https://docs.python.org/3/library/threading.html (último acceso: junio 2024)

[11] Ultralytics, YOLO Documentation, 2024 https://docs.ultralytics.com/ (último acceso: mayo 2024)

[12] TensorFlow, TensorFlow U-Net Documentation, 2024 https://u-net.readthedocs.io/en/latest/ (último acceso: mayo 2024)

[13] Naciones Unidas. Objetivos de Desarrollo Sostenible. https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ (último acceso: julio 2024)

[14] Puchalt, J. C., Sánchez-Salmerón, A. J., Martorell Guerola, P., & Genovés Martínez, S. (2019). Active backlight for automating visual monitoring: An analysis of a lighting control technique for Caenorhabditis elegans cultured on standard Petri plates. PloSone, 14(4), e0215548. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0215548 (último acceso: junio 2024)

[15] Puchalt, J. C., Sánchez-Salmerón, A. J., Ivorra, E., Genovés Martínez, S., Martínez, R., & Martorell Guerola, P. (2020). Improving lifespan automation for Caenorhabditis elegans by using image processing and a post-processing adaptive data filter. Scientific reports,10(1), 8729. https://www.nature.com/articles/s41598-020-65619-4 (último acceso: junio 2024)

[16] Puchalt, J. C., Layana Castro, P. E. & Sánchez-Salmerón, A. J. (2020). Reducing results variance in lifespan machines: an analysis of the influence of vibrotaxis on wild- type Caenorhabditis elegans for the death criterion. Sensors, 20(21), 5981. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7660079/ (último acceso: junio 2024)

[17] Layana Castro, P. E., Puchalt, J. C., & Sánchez-Salmerón, A. J. (2020). Improving skeleton algorithm for helping Caenorhabditis elegans trackers. Scientific Reports, 10(1), 22247. https://www.nature.com/articles/s41598-020-79430-8 (último acceso: junio 2024)

[18] Layana Castro, P. E., Puchalt, J. C., García Garví, A., & Sánchez-Salmerón, A. J. (2021). Caenorhabditis elegans multi-tracker based on a modified skeleton algorithm. Sensors, 21(16), 5622. https://pubmed.ncbi.nlm.nih.gov/34451062/ (último acceso: junio 2024)

[19] Puchalt, J. C., Gonzalez-Rojo, J. F., Gómez-Escribano, A. P., Vázquez-Manrique, R. P., & Sánchez-Salmerón, A. J. (2022). Multiview motion tracking based on a cartesian robot to monitor Caenorhabditis elegans in standard Petri dishes. Scientific reports, 12(1), 1767. https://www.nature.com/articles/s41598-022-05823-6 (último acceso: junio 2024)

[20] Navarro Moya, F., Puchalt, J. C., Layana Castro, P. E., García Garví, A., & Sánchez-Salmerón, A. J. (2022). A new training strategy for spatial transform networks (STN's). Neural Computing and Applications, 34(12), 10081-10092. https://link.springer.com/article/10.1007/s00521-022-06993-0 (último acceso: junio 2024)

[21] Rico-Guardiola, E. J., Layana-Castro, P. E., García-Garví, A., & Sánchez- Salmerón, A. J. (2022, October). Caenorhabditis elegans detection using yolov5 and faster r-cnn networks. In International Conference on Optimization, Learning Algorithms and Applications (pp. 776-787). Cham: Springer International Publishing. https://link.springer.com/chapter/10.1007/978-3-031-23236-7_53 (último acceso: junio 2024)

[22] García-Garví, A., Layana-Castro, P. E., & Sánchez-Salmerón, A. J. (2023). Analysis of a C. elegans lifespan prediction method based on a bimodal neural network and uncertainty estimation. Computational and Structural Biotechnology Journal, 21, 655-664. https://www.sciencedirect.com/science/article/pii/S2001037022005906 (último acceso: junio 2024)

[23] Castro, P. E. L., Garví, A. G., & Sánchez-Salmerón, A. J. (2023). Automatic segmentation of Caenorhabditis elegans skeletons in worm aggregations using improved U-Net in low-resolution image sequences. Heliyon, 9(4). https://www.sciencedirect.com/science/article/pii/S2405844023019229 (último acceso: junio 2024)

[24] Layana Castro, P. E., García Garví, A., Navarro Moya, F., & Sánchez-Salmerón, A. J. (2023). Skeletonizing Caenorhabditis elegans Based on U-Net Architectures Trained with a Multi-worm Low-Resolution Synthetic Dataset. International Journal of Computer Vision, 1-17. https://link.springer.com/article/10.1007/s11263-023-01818-6 (último acceso: junio 2024)