UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño, implementación y evaluación de técnicas genéricas de detección de C. elegans mediante redes neuronales convolucionales

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Rico Guardiola, Ernesto Jesús

Tutor/a: Sánchez Salmerón, Antonio José

Director/a Experimental: GARCIA GARVI, ANTONIO

CURSO ACADÉMICO: 2021/2022

Agradecimientos

A mi hermana, por ser mi pilar fundamental y apoyarme en todo lo que hago, su fortaleza me inspira a seguir adelante. A mis padres, por haberme hecho la persona que soy hoy y por inculcarme el valor del esfuerzo.

A mi compañera y amiga María Ronda, por haber compartido conmigo tantos momentos de felicidad y también de dificultad, mi experiencia en la Universidad no habría sido tan satisfactoria sin ella.

A mis amigos, que han estado siempre a mi lado aunque a veces no lo viera.

A los compañeros del Instituto ai2, que me han enseñado muchísimo y me han guiado para conseguir mis objetivos. Especialmente quiero a gradecer a Antonio Sánchez y a Antonio García sus consejos y el haberme abierto la puerta a un nuevo campo de conocimiento.

Resumen:
El nematodo *Caenorhabditis elegans* (*C. elegans*) ha emergido como un importante modelo animal para la investigación de enfermedades neurodegenerativas y desarrollo de nuevos fármacos y grupos alimenticios. Por este motivo, resulta interesante el desarrollo de técnicas de automatización que permitan aumentar la productividad y la precisión en el desarrollo de los ensayos.

El objetivo de este trabajo consiste en el diseño, implementación y evaluación de técnicas genéricas de detección de *C. elegans* en imágenes capturadas con varios sistemas de visión mediante distintos tipos de redes neuronales convolucionales (tipo Mask R-CNN y Yolo5) con la finalidad de poder analizar su comportamiento.

Para la realización del trabajo se parte de dos datasets de imágenes capturados con sistemas de visión diferentes, las cuales ya están etiquetadas, proporcionadas por el Instituto de Automática e Informática Industrial (ai2) de la UPV.

La implementación y entrenamiento de los modelos de aprendizaje profundo se realizará en el lenguaje de programación Python utilizando la librería Pytorch. Además, se utilizarán otras librerías de procesamiento de imágenes como OpenCV y PIL.

Finalmente, se evaluarán y optimizarán las distintas propuestas, utilizando como criterios de comparación y optimización las tasas de aciertos y los costes temporales de computación.

Palabras clave: Automatización, Visión Artificial, Aprendizaje Profundo, Redes Neuronales Convolucionales, Detección de *C. elegans*.

Resum:
El nematode *Caenorhabditis elegans* (*C. elegans*) ha emergit com un important model animal per a la investigació de malalties neurodegeneratives i desenvolupament de nous fàrmacs i grups alimentosos. Per aquest motiu, resulta interessant el desenvolupament de tècniques d'automatització que permeten augmentar la productivitat i la precisió en el desenvolupament dels assajos.

L'objectiu d'aquest treball consisteix en el disseny, implementació i avaluació de tècniques genèriques que permeten detectar C. elegans en imatges capturades amb diferents sistemes de visió artificial, amb la finalitat de poder analitzar el seu comportament (moviment, poses, etc). En concret, s'avaluaran diferents tipus de xarxes neuronals convolucionals (tipus Mask R-CNN i Yolo5).

Per a la realització del treball es parteix de dues datasets d'imatges, les quals ja estan etiquetades, proporcionades per l'Institut d'Automàtica i Informàtica Industrial (ai2) de la UPV.

La implementació i entrenament dels models d'aprenentatge profund es realitzarà en el llenguatge de programació Python utilitzant la llibreria Pytorch. A més, s'utilitzaran altres llibreries de processament d'imatges com OpenCV i PIL.

Finalment, s'avaluaran i optimitzaran les diferents propostes, utilitzant com a criteris de comparació i optimització les taxes d'encerts i els costos temporals de computació.

Paraules clau: Automatització, Visió Artificial, Aprenentatge Profund, Xarxes Neuronals Convolucionals, Detecció de *C. elegans*.

Summary:
The nematode *Caenorhabditis elegans (C. elegans)* has emerged as an important animal model for neurodegenerative illnesses investigation and develop of new medicines and alimentary groups. For this reason, it seems interesting to develop new automatization techniques that allow to increase the productivity and precision in the develop of the tests.

The objective of this work consists of the design, implementation and evaluation of generic techniques for *C. elegans* detection in images captured with different vision systems by means of convolutional neural networks (Mask R-CNN and Yolo5) with the purpose of being able to analyze their behavior.

The making of this work is based in two image datasets, with the images already labelled, provided by the Instituto de Automática e Informática Industrial (ai2) from the UPV.

The implementation and training of the deep learning methods will be programmed in Python using Pytorch library. Moreover, other processing libraries such as OpenCV and PIL will be used.

Finally, the different proposals will be evaluated and optimized, using criteria of hit rates and computational time costs as optimization and comparison criteria.

Key words: Automation, Computer Vision, Deep Learning, Convolutional Neural Network, *C. elegans* Detection.

Índice General

NOTA: El documento de Planos se ha excluido del presente trabajo puesto que, al tratarse de un proyecto de programación, no existen elementos tangibles cuyas dimensiones sean de interés ni esquemas de conexiones.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Memoria

Curso 2021/2022

Índice memoria

Índice de figuras

Índice de tablas

Índice de ecuaciones

1. Objeto

La finalidad de este trabajo fin de grado es el de desarrollar un sistema genérico de detección de *Caenorhabditis elegans* (*C. elegans*) basado en redes neuronales convolucionales. Para ello, primero se va a realizar una comparación entre dos de las principales arquitecturas de redes empleadas en la detección de objetos en imágenes: Faster R-CNN y YOLO V5s. Posteriormente, se empleará uno de los modelos para comprobar si es posible obtener un sistema capaz de detectar *C. elegans* a partir de imágenes capturadas en condiciones diferentes.

2. Descripción del problema

2.1. C. elegans

El *Caenorhabditis elegans* es una especie de nematodo que se ha convertido en un importante modelo de estudio durante las últimas décadas. Este tipo de nematodo presenta una longitud de 1 mm, es transparente y, pese a su pequeño tamaño, desarrolla ciertos comportamientos de la misma forma que otros animales: se alimentan, se reproducen de forma sexual, envejecen y tienen sistema nervioso. Su anatomía (Fig. 1) está formada por boca, faringe, intestinos, gónadas y una cutícula de colágeno. Cuando nacen pasan por un proceso de larva, que dura unos dos o tres días, hasta que llegan a su etapa adulta, en la que vivirán de dos a tres semanas, siempre que se mantenga una temperatura de 20°C [1], [2].

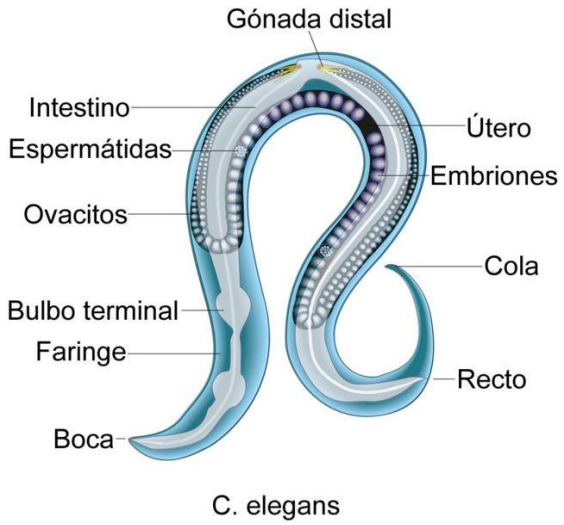

Fig. 1: Anatomía de los C. elegans [3]

Su estudio se ha popularizado debido a las aplicaciones que tiene en diferentes campos como la biología, la etología, la medicina y la agricultura. Por ejemplo, se emplean para evitar la experimentación con animales en estudios de toxicidad [4]. También se utilizan para comprender el comportamiento de nematodos parásitos que atacan a la vegetación y a los animales [5].

Algunas de las ventajas de los *C. elegans* radican en su fácil cultivo y conservación en placas de Petri (Fig. 2), donde se alimentan de la bacteria *Escherichia coli*. Además, debido a su transparencia, se puede observar fácilmente con ayuda del microscopio su evolución morfológica.

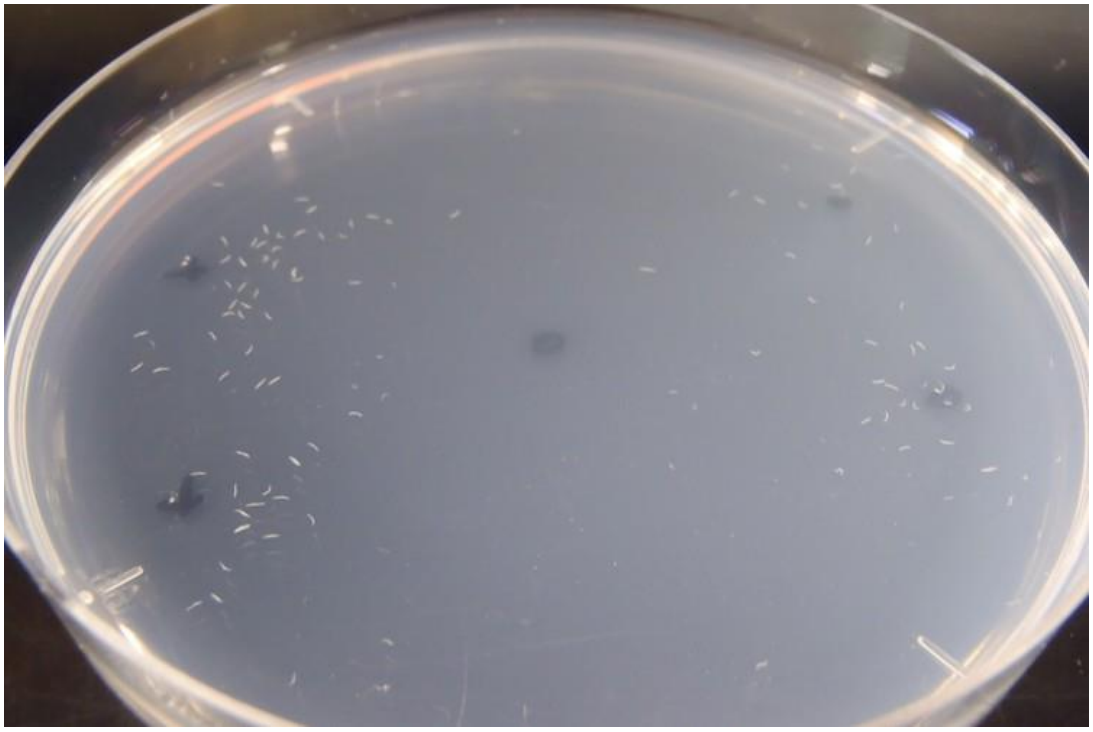

Fig. 2: C. elegans (en blanco) en una placa de Petri a tamaño real [6]

Los estudios más populares llevados a cabos con estos nematodos son *lifespan* y *healthspan*, entre otros. Estos experimentos consisten en estudiar la esperanza de vida y el deterioro físico de los nematodos, respectivamente, ya que son comparables a los de los humanos, debido a la existencia de factores que influyen tanto en humanos como en *C. elegans*: las condiciones del medio, la alimentación y las mutaciones genéticas. En muchos de estos experimentos, el primer paso para obtener la información es la detección de los nematodos, es decir, identificarlos y localizarlos.

Actualmente, estos experimentos se realizan de forma manual en la mayoría de laboratorios, pero suponen un gran gasto de tiempo y de recursos. Es por esto por lo que resulta interesante utilizar técnicas de visión artificial, automatizando tanto la captura de imágenes de los nematodos como la obtención de sus características y comportamiento.

A la hora de estudiar e inspeccionar *C. elegans* en placas de Petri existen ciertos factores que dificultan esta labor: (1) las diversas formas que pueden adoptar los nematodos; (2) el problema de la contaminación de la placa o la condensación en la tapa, que precisan de técnicas de iluminación especiales y (3) el problema de detectar los nematodos cuando existen agregaciones de varios, que requiere técnicas de segmentación específicas [7].

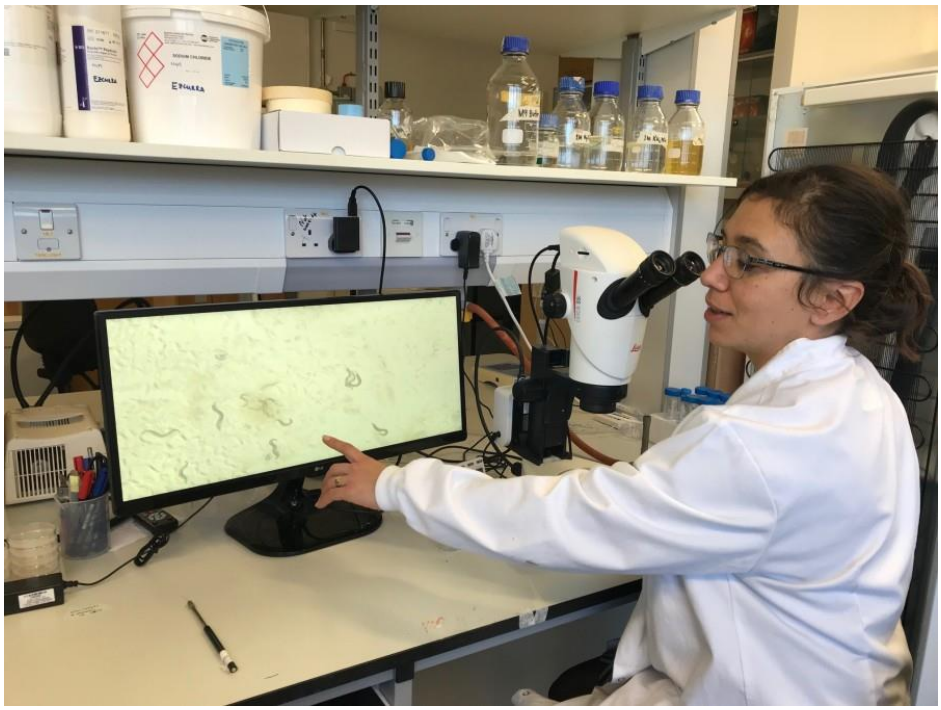

Fig. 3: Una investigadora inspecciona una placa de Petri con C. elegans en el microscopio [8]

2.2. Visión artificial

Dentro del campo de la visión artificial existe gran cantidad de pre-procesos, operaciones y algoritmos para obtener información de imágenes, como la detección y la posición de determinados objetos, la clasificación de estos, etc. En la última década, han empezado a popularizarse los algoritmos basados en *machine learning* o aprendizaje profundo, como las redes neuronales, desplazando a las técnicas tradicionalmente empleadas en visión artificial a un segundo plano en ciertas aplicaciones.

2.2.1. Métodos tradicionales de visión artificial

Las técnicas tradicionales de visión artificial se basan en extraer características de las imágenes y así obtener información de ellas. De esta forma, el aprendizaje de las características es manual y el programador o ingeniero tiene que ajustar los parámetros asociados al algoritmo o a las características que se desean detectar. Algunos de los algoritmos que se suelen emplear para detección y clasificación son *Scale-invariant feature transform* (SIFT), *Speeded-Up Robust Features* (SURF) y *Binary Robust Independent Elementary Features* (BRIEF) [7].

La gran ventaja de los algoritmos de aprendizaje profundo es que son capaces de realizar automáticamente la extracción de características de las imágenes.

2.2.2. Redes neuronales

El funcionamiento normal del cerebro humano se caracteriza por ser complejo, no lineal y en paralelo. Esto lo distingue mucho del comportamiento típico de los computadores digitales, que son secuenciales y, por tanto, capaces de realizar una única operación a la vez. Es por esto por lo que el cerebro humano es capaz de realizar análisis muy complejos en cuestión de milisegundos, como el reconocimiento de objetos, patrones y personas, mientras que los computadores tienen gastos bastante mayores para tareas más sencillas. Una de las características más importantes del cerebro es la plasticidad, entendiendo esta como la capacidad de responder a nueva información en el entorno [9].

Las redes neuronales son un potente modelo computacional que pretende simular el comportamiento de las neuronas biológicas. A los elementos básicos de procesamiento que componen las redes neuronales se les llama neuronas y se organizan en capas.

Las neuronas presentan unas conexiones de entrada por las que reciben los estímulos externos (inputs). Cada neurona realiza una suma ponderada de todas las entradas según el peso que se le asigna a cada una. Estos pesos se inicializan aleatoriamente y se van actualizando durante un entrenamiento, teniendo en cuenta la diferencia entre el resultado esperado y la predicción de la red. A partir de la suma ponderada y pasando por una función de activación, la neurona obtiene un valor a la salida. Esta función de activación se emplea para resolver las no linealidades, siendo las más comunes la función sigmoide, la tangente hiperbólica y la función ReLU (Fig. 4).

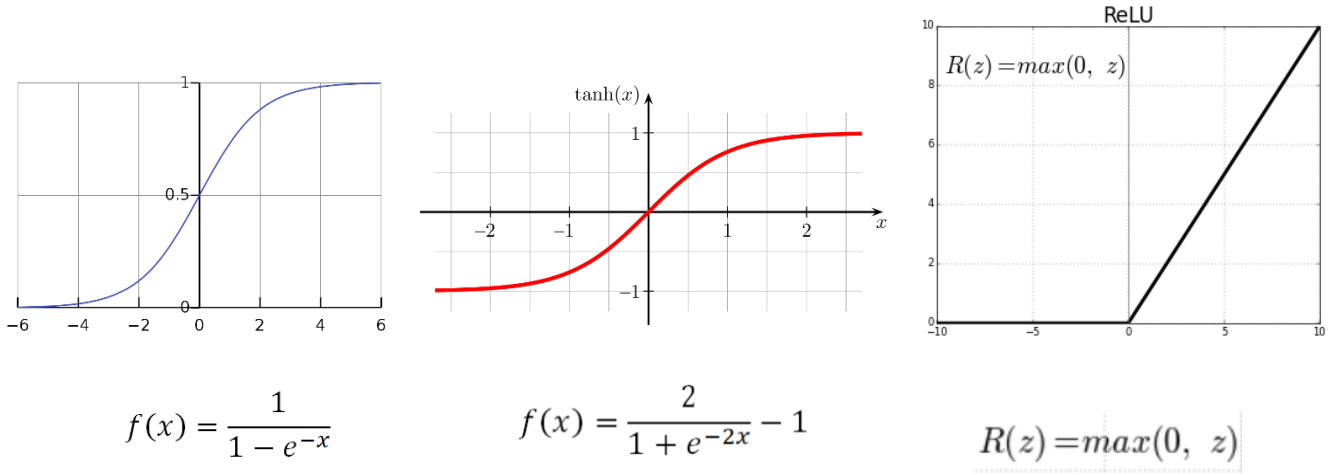

Fig. 4: Algunas funciones de activación comúnmente empleadas. De izquierda a derecha: sigmoide, tangente hiperbólica y ReLU [10]

Comparando estas neuronas con las neuronas biológicas, las entradas simulan las dendritas, los pesos simulan la sinapsis y las salidas el axón.

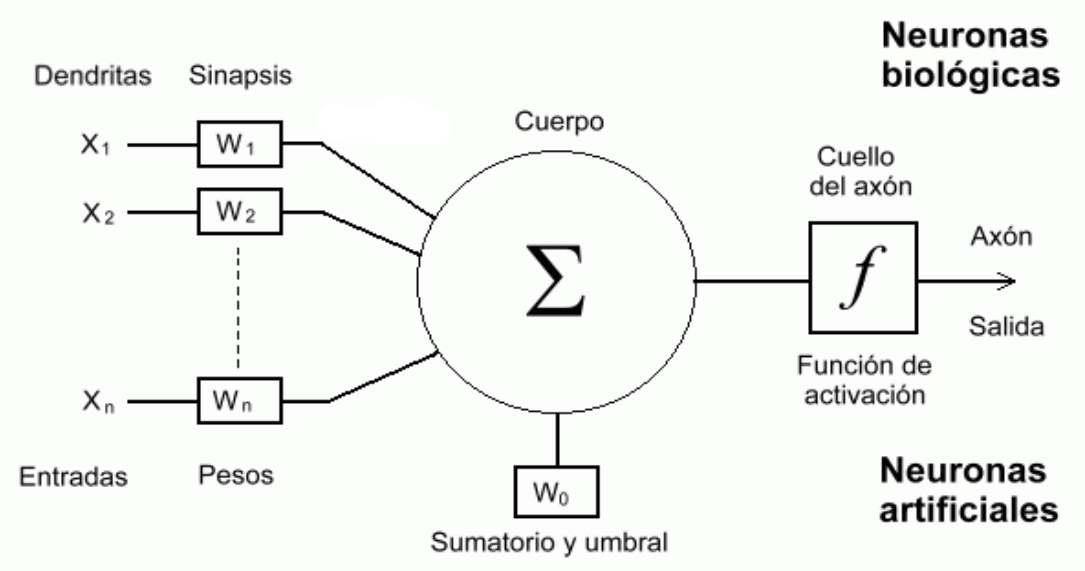

Fig. 5: Esquema de una neurona computacional y las partes equivalentes en las neuronas humanas [11]

Las redes neuronales están formadas por neuronas interconectadas que forman diferentes capas, constituyendo un sistema multicapa. La primera capa y la última corresponden con las capas de entrada y salida, respectivamente. Las demás se denominan capas ocultas (Fig. 6) [12].

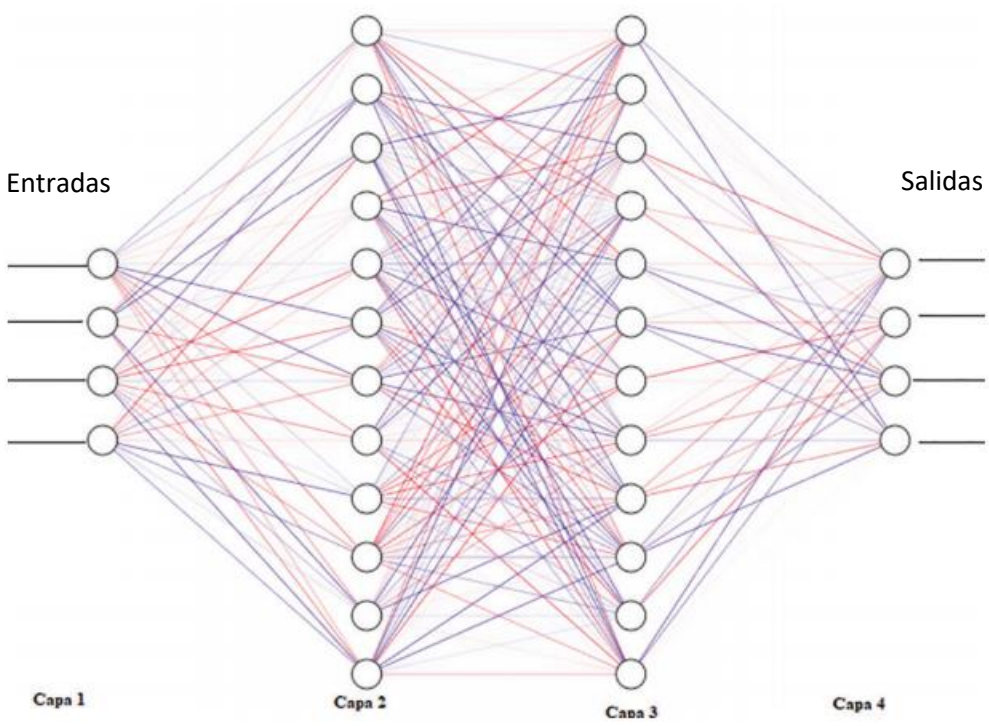

Fig. 6: Ejemplo de red neuronal artificial completamente conectada. Las capas 2 y 3 corresponden a las capas ocultas [13]

Dentro de este campo, podemos encontrar diferentes tipos de redes neuronales:

- Red neuronal monocapa: consiste en una capa de neuronas de entrada y otra de salida, siendo el tipo de red más simple.

- Red neuronal multicapa: este tipo de red dispone de capas intermedias, denominadas ocultas, además de una capa de entrada y otra de salida. Este es el esquema típico de red neuronal (Fig. 6).

- Red neuronal recurrente: esta red no tiene una estructura definida de capas, sino que pueden presentar conexiones arbitrarias entre las neuronas y pueden crear ciclos, de forma que se consigue crear la temporalidad y la red puede disponer de memoria.

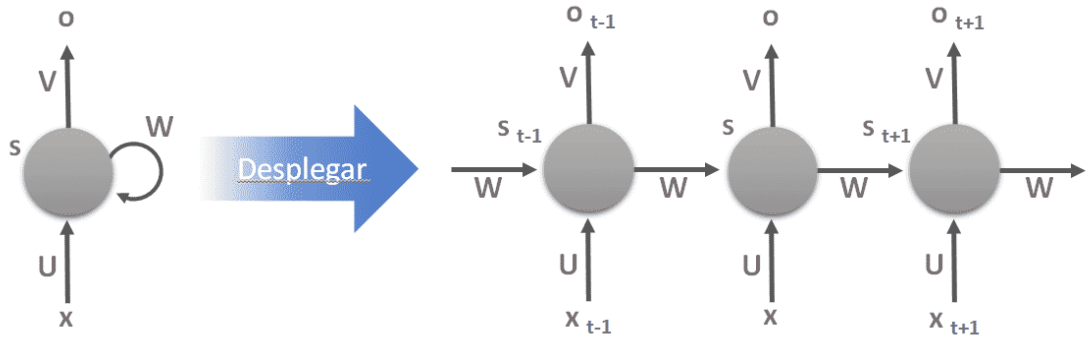

Fig. 7: Ejemplo de red neuronal recurrente [14]

- Redes de base radial: estas calculan la salida en función de la distancia a un centro determinado. La salida se obtiene como combinación lineal de las funciones de activación radiales empleadas por cada neurona.

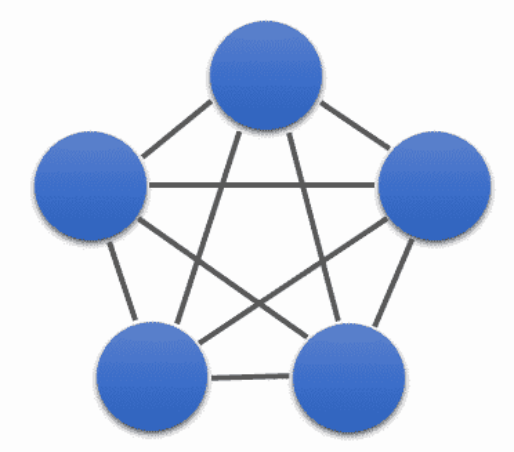

Fig. 8: Esquema de redes de base radial [14]

- Red neuronal convolucional (CNN): en este tipo de red neuronal, cada neurona se une solo con un subgrupo de las capas siguientes, de forma que se puede reducir la cantidad de neuronas necesarias y la complejidad computacional.

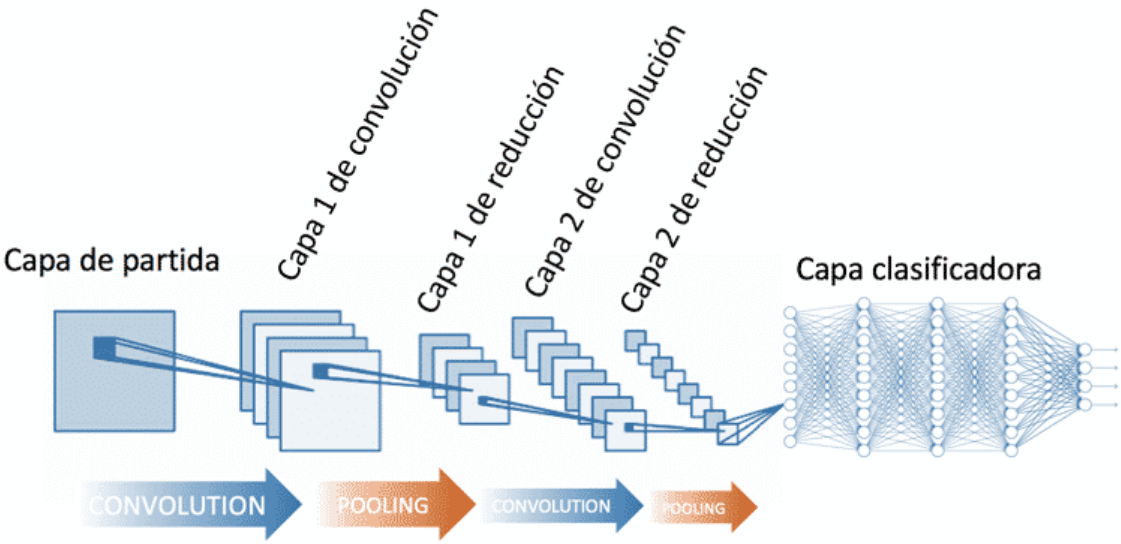

Fig. 9: Esquema de una red neuronal convolucional [14]

Las redes neuronales se utilizan en diferentes campos, gracias a que ayudan a resolver muchos problemas actuales. Algunos de estos son la inteligencia artificial (asistente de voz, robots), reconocimiento y clasificación de patrones y datos, simulación y predicción de sucesos y visión por computador [15].

Las redes neuronales están en pleno crecimiento ya que, en aplicaciones como la visión artificial, consiguen obtener mejores resultados que otros métodos tradicionales de visión artificial, como los basados en heurística.

En este trabajo se van a emplear redes neuronales convolucionales y basadas en región, ya que son ampliamente utilizadas en aplicaciones de visión artificial, especialmente para este tipo de proyectos. Las redes neuronales convolucionales basadas en región (R-CNN) se suelen utilizar para detección de objetos, ya que obtienen las cajas delimitadoras de los objetos de interés.

Para que las redes neuronales puedan realizar un aprendizaje mediante entrenamiento, es necesario contar con datasets de imágenes, cuyos objetos de interés hayan sido etiquetados previamente. En los procesos posteriores, validación y test, la red necesita disponer también de las etiquetas para poder comparar los resultados con los datos reales y así obtener métricas para evaluar la efectividad del modelo.

3. Estudio de necesidades

3.1. Especificaciones y limitaciones

Para cumplir los objetivos propuestos, se debe conseguir un sistema con las siguientes especificaciones:

-Tasa de precisión mayor del 90%.

-Tiempo por predicción menor de 40 ms para aplicaciones en tiempo real, puesto que los sistemas de monitorización de C. elegans suelen emplear frecuencias de captura de 25 fps (*frames per second*, fotogramas por segundo).

-Sistema capaz de detectar nematodos en tres canales diferentes, esto es, en 3 condiciones diferentes de captura e iluminación.

Por otro lado, existen ciertas limitaciones a la hora de conseguir los objetivos propuestos:

-Se deberán emplear los equipos informáticos proporcionados por el Instituto ai2 para el desarrollo del programa y para el entrenamiento de las redes neuronales, lo que puede afectar a los tiempos de ejecución y, por tanto, al cumplimiento total de los objetivos.

-No existe gran cantidad de datasets de imágenes de *C. elegans* publicados, por lo que solo se podrá trabajar con los datasets que se encuentren disponibles y el entrenamiento estará limitado a los canales de estos.

3.2. Proyectos similares

Antes de abordar el problema de detección y el del diseño de un sistema genérico, se han estudiado otros trabajos que han realizado aplicaciones similares, sobre todo de detección. A continuación, se enumeran algunos de ellos:

- *Diseño, desarrollo y evaluación de un sistema de clasificación de objetos en imágenes que permita la monitorización de C. elegans mediante redes neuronales convolucionales* [7]. En este trabajo se pretende desarrollar un algoritmo para automatizar las tareas de monitorización de *C. elegans*. El objetivo principal es el de obtener un clasificador de imágenes para detectar si los nematodos están vivos o muertos. Para ello, realizan ensayos con dos arquitecturas de redes neuronales: las redes neuronales convolucionales y las recurrentes, comprobando finalmente que la combinación de ambas arquitecturas es la que proporciona los mejores resultados. En este proyecto, pese a emplear técnicas similares a las que pretendemos aplicar, se enfoca en el ensayo de *lifespan.*

- *Detección de C. elegans basada en Mask R-CNN con un microscopio casero* [16]. En este trabajo, se desarrolla un sistema de adquisición con microscopio para capturar imágenes de *C. elegans* en placas de Petri. De esta forma, generan un gran dataset de imágenes de baja resolución, lo cual no es un problema puesto que emplean técnicas de aprendizaje profundo para procesar las imágenes, en vez de técnicas tradicionales que suelen requerir resoluciones mayores. El modelo de red neuronal que emplean es Mask R-CNN, para el cual necesitan obtener las máscaras de los nematodos mediante pre-procesos. Con este

modelo, consiguen localizar y clasificar a los *C. elegans*, así como predecir sus contornos y obtener máscaras con las predicciones de la red. El trabajo de este artículo se asemeja al que queremos hacer, ya que utilizan redes neuronales para detección de *C. elegans*. Sin embargo, no está tan enfocado a la obtención de un sistema genérico de detección sino, más bien, a la obtención de un sistema barato y automático de captura de imágenes y procesamiento de las mismas.

- *Aprendizaje profundo para seguimiento robusto y flexible en experimentos de comportamiento de C. elegans* [17]. En este artículo se pretende obtener información acerca del comportamiento de los *C. elegans*. Para ello, aplican un algoritmo para hacer un seguimiento de la velocidad del animal durante el desarrollo, las ratios de fecundidad y la distribución espacial en adultos reproductivos, así como el deterioro del comportamiento en poblaciones que envejecen. Para esto, utilizan métodos basados en el *machine learning*, principalmente, el modelo Faster R-CNN. Su objetivo es mostrar cómo se puede emplear Faster R-CNN para identificación y detección de *C. elegans* en diversos estados de crecimiento y entornos complejos. Además, con esto pretenden mostrar cómo las redes neuronales presentan grandes ventajas para estas aplicaciones respecto a los métodos tradicionales basados en la heurística, intentando generalizar las técnicas de detección para diferentes experimentos.

4. Análisis de alternativas

4.1. Arquitecturas de redes neuronales convolucionales

4.1.1. YOLO V5s

Yolo (You Only Look Once) es un algoritmo ampliamente utilizado en aplicaciones de detección de objetos. Su mayor ventaja radica en el pequeño tamaño de la red y, por tanto, gran velocidad de cálculo. Además, resulta bastante útil en aplicaciones de generalización como la del presente trabajo, ya que el algoritmo es capaz de aprender características generales de los objetos para aplicarlas en otras condiciones o campos [18].

Desde la publicación de Yolo en 2015, se han ido desarrollando otras variantes en los últimos años: YOLO V2, YOLO V3, YOLO V4 y YOLO V5. Para esta aplicación se ha empleado una de las últimas versiones, YOLO V5, por estar mejor actualizada.

YOLO V5 fue creada por Glenn Jocher en 2020. Esta versión ofrece ventajas respecto a las anteriores, en términos de precisión, velocidad de cálculo y tamaño de la red. Dentro de YOLO V5 existen diferentes modelos según tamaño de red y prestaciones. En la Fig. 10 se observan los diferentes modelos de YOLO V5 con sus respectivas características de tamaño de red (primera fila), velocidad de cálculo (segunda fila) y precisión en validación con el dataset COCO [19], que se trata de un dataset especialmente diseñado para entrenar la detección de objetos cotidianos (tercera fila).

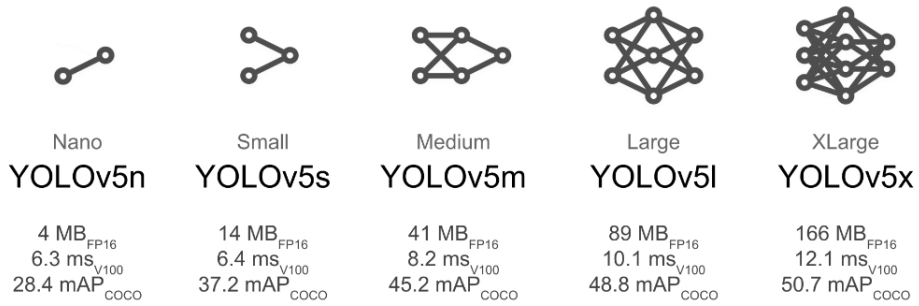

Fig. 10: Características de los diferentes modelos de YOLO V5

Para nuestra aplicación se ha optado por el modelo YOLOv5s, ya que se ha considerado que es con el que se obtiene el mejor equilibrio velocidad-precisión.

La red YOLO V5 está formada por tres partes diferenciadas: *Backbone*, *Neck* y *Head*. Tras introducir la imagen de entrada, *Backbone* trata de extraer las características más importantes de la imagen. *Neck* construye una pirámide de características, favoreciendo la detección e identificación de objetos de la misma clase en diferentes tamaños y escalas. Esto mejora el funcionamiento de la red con datos aún no aprendidos. Por último, *Head* realiza la detección final de las características de la imagen para generar cuadros delimitadores y predicción de las categorías [20], [21]. En la Fig. 11 se puede observar el esquema de funcionamiento de la red YOLO V5s.

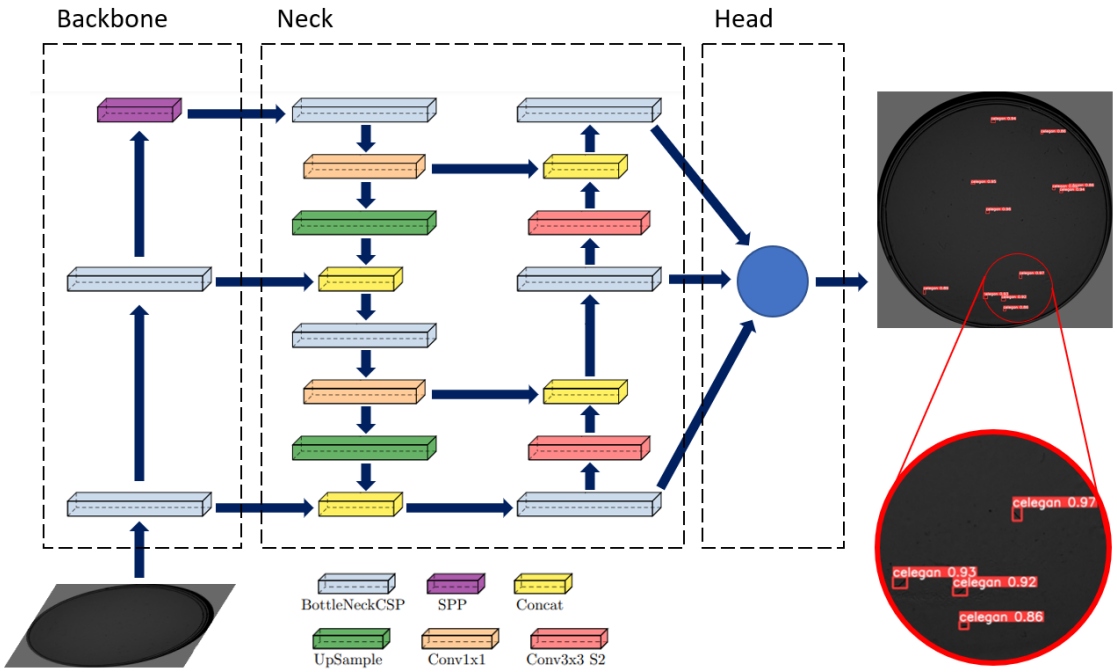

Fig. 11: Esquema de la red YoloV5s [22]

Para poder realizar un entrenamiento de la red YOLO V5s, se necesita como entrada el dataset de imágenes y las cajas delimitadoras o etiquetas de los objetos a detectar en cada imagen. Las etiquetas de los objetos en YOLO pueden proporcionarse en diferentes formatos. El que se va a emplear en este trabajo por comodidad es el formato “pytorch

txt". En este formato, se necesita un documento de texto por imagen, donde las cajas de los objetos se almacenan en las filas, con 5 términos por caja (Fig. 12).

lifespan daf2 y bebidas Ens2.0 cond=A pla=1 dia=1.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
0 0.5794753086419753 0.5835905349794239 0.008744855967078191 0.0025720164609054352
0 0.1550925925925926 0.5406378600823045 0.008744855967078191 0.010288065843621408
0 0.4238683127572016 0.24099794238683128 0.005144032921810704 0.010802469135802462
0 0.7420267489711934 0.5280349794238683 0.007716049382716084 0.008744855967078191
0 0.801954732510288 0.46604938271604934 0.008230452674897082 0.012345679012345678
0 0.24537037037037038 0.6702674897119342 0.01131687242798357 0.008230452674897082
0 0.378858024691358 0.45987654320987653 0.003600823045267487 0.013374485596707841
0 0.7952674897119342 0.5856481481481481 0.0092592592592593 0.008744855967078191
0 0.7193930041152263 0.6021090534979424 0.0036008230452675427 0.011831275720164625
0 0.6676954732510287 0.6741255144032923 0.016460905349794275 0.007716049382716084
0 0.6823559670781894 0.4045781893004115 0.00462962962962965 0.011831275720164625
0 0.7255658436213992 0.5432098765432098 0.010802469135802406 0.011316872427983515
0 0.6579218106995885 0.867798353909465 0.012345679012345623 0.008230452674897082
```

Fig. 12: Ejemplo de archivo con las etiquetas de una imagen en formato pytorch txt

La primera columna corresponde a la clase del objeto codificada con un número entero, en este caso solo existe una clase de objeto, *C. elegans* (0). La segunda y tercera columna corresponden a las coordenadas del centro de la caja, mientras que la cuarta y quinta corresponden a la altura y anchura, respectivamente (Fig. 13). Todas estas variables deben aparecer en formato normalizado, es decir, en términos relativos a la altura y anchura en píxeles de la imagen entera.

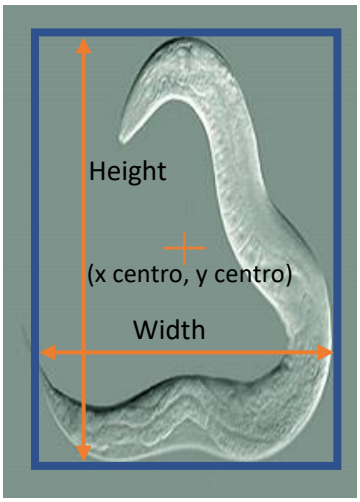

Fig. 13: Formato de caja para YOLO V5s

Además de los archivos de texto con las cajas, este formato requiere de un fichero llamado "data.yaml" donde aparece la siguiente información:

-train: directorio donde se encuentran las imágenes de entrenamiento.

-val: directorio donde se encuentran las imágenes de validación.

-test: directorio donde se encuentran las imágenes de test

-nc: número de clases de los objetos a detectar.

-names: nombres de las clases de objetos a detectar.

data.yaml

```yaml
train: ../train/images
val: ../valid/images

nc: 3
names: ['head', 'helmet', 'person']
```

Fig. 14: Ejemplo de formato del archivo data.yaml

4.1.2. Faster R-CNN

Las redes neuronales convolucionales basadas en regiones (R-CNN) han superado a las redes neuronales convolucionales clásicas al conseguir mejores resultados en áreas de estadística y de visión artificial, entre otras. La gran ventaja de estas respecto a sus antecesoras es que son capaces de obtener la ubicación de los objetos y no solo su clase, gracias a que pueden identificar las regiones de interés (ROIs). Con los años, se han ido desarrollando nuevas versiones mejoradas de R-CNN, tales como Fast R-CNN, Faster R-CNN o Mask R-CNN, que obtienen mejores resultados en términos de precisión y con mayor velocidad [23].

Para nuestra aplicación se ha elegido Faster R-CNN debido a que es la más rápida de las mencionadas anteriormente. El modelo Mask R-CNN presenta una ventaja respecto a otros: permite obtener una segmentación de los objetos que predice en las imágenes. Sin embargo, necesita como entrada a su vez las máscaras de los objetos de interés de las imágenes, en lugar de las etiquetas. Para este trabajo se ha considerado que el trabajo y tiempo extra que supone obtener la segmentación de las imágenes de los *C. elegans* no compensa, puesto que la Faster R-CNN trabaja más rápido y con buenos resultados, así que la Mask R-CNN se ha descartado.

Faster R-CNN es una arquitectura de red neuronal basada en región propuesta en 2015 en [24]. La estructura de Faster R-CNN está formada por los siguientes módulos [22], [25]:

1) Una red neuronal convolucional cuya función es la de obtener las características de la imagen.
2) Un algoritmo de propuesta de región de interés (RPN) que genera las cajas delimitadoras candidatas, es decir, la localización de los posibles objetos en la imagen.
3) Una red que clasifica los objetos por clases y realiza la regresión de las coordenadas de las cajas delimitadoras de los objetos.

Una de las mejoras que presenta el modelo Faster R-CNN es que comparte capas convolucionales con la red encargada de la detección de objetos, disminuyendo así el coste de la propuesta. Además, se emplean unas pocas capas convolucionales para volver a calcular los límites de las regiones y las puntuaciones de objetividad en cada

ubicación. Gracias a estos cambios, se han mejorado las prestaciones de la red en términos de velocidad y rendimiento en la detección de objetos [24], [26].

En la Fig. 15 se puede observar un esquema de la red Faster R-CNN.

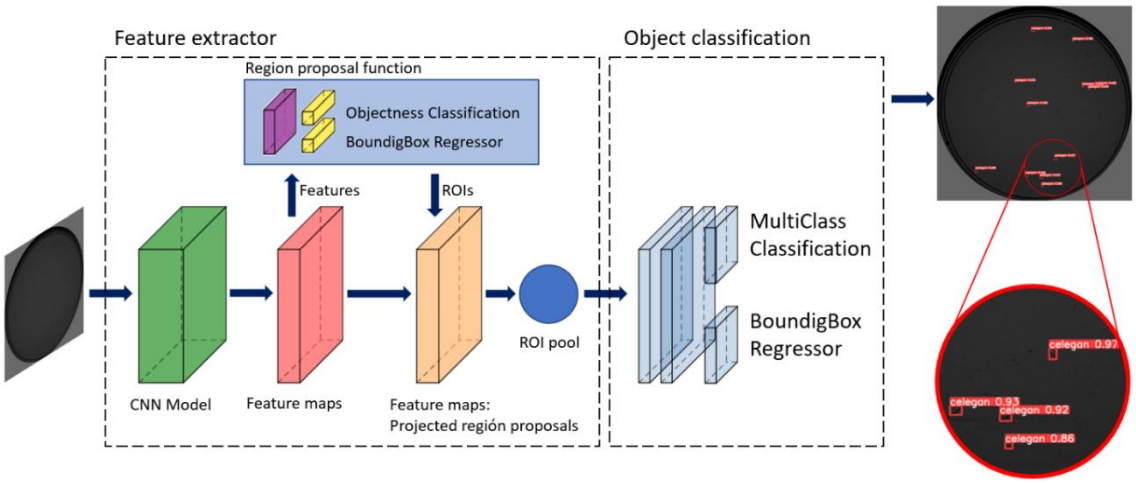

Fig. 15: Esquema de la red Faster R-CNN [22]

4.2. Métricas de evaluación de modelos

Para poder evaluar los modelos obtenidos y poder compararlos con otras alternativas, se hace uso de algunas métricas típicamente empleadas en la evaluación de modelos de visión artificial:

Intersection over Union (IoU): Mide el grado de solape entre la caja real que delimita el objeto y la predicha por la red.

$$IoU = \frac{Intersección}{Unión}$$

Ecuación 1: Intersection over Union

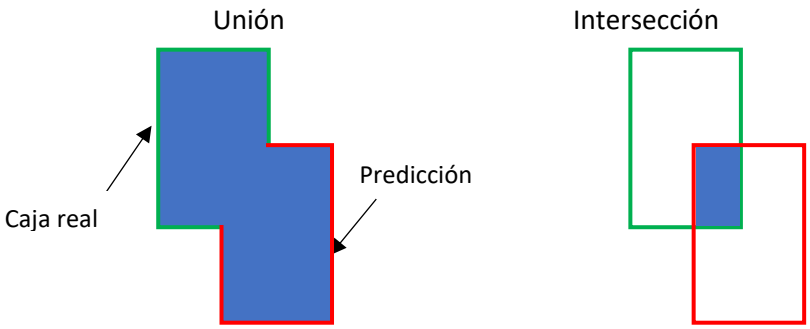

Fig. 16: Descripción gráfica de la unión y la intersección de las cajas delimitadoras

De esta forma, cuanto mayor sea el IoU, mayor será el solape entre ambas cajas y por tanto la predicción será más correcta. Este parámetro es clave para calcular las demás métricas, puesto que determina qué se considera como detecciones verdaderas y falsas según el umbral de IoU que se elija. Es decir, las predicciones cuyos IoUs sean mayores del umbral se considerarán verdaderos positivos y las que presenten un valor menor al umbral se considerarán como falso positivo.

Precision o precisión: Representa la relación entre las detecciones correctas y todas las detecciones predichas por el modelo.

$$Precision = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ Positivos}$$

Ecuación 2: Expresión para calcular la precisión

Recall o exhaustividad: Muestra la habilidad del modelo para realizar correctas detecciones de entre todas las cajas reales de los objetos.

$$Recall = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ Negativos}$$

Ecuación 3: Expresión para calcular el recall

Average Precision (AP) o precisión media: Se calcula como como el área bajo la curva *precision-recall*, condensando la información de la curva en un único parámetro.

Mean Average Precision (mAP): Esta métrica se obtiene calculando la media de los APs sobre todos los umbrales del IoU. En este caso se emplea el mAP para un IoU del 0.5 y para el rango [0.5; 0.95].

$$mAP = \frac{1}{N}\sum AP(n)$$

Ecuación 4: Expresión para calcular el mAP

Donde N es el número total de umbrales de IoU y AP(n) es la precisión media para cada umbral.

4.3. Comparación entre YOLO V5s y Faster R-CNN

Para realizar una comparación de ambos modelos, se ha realizado un entrenamiento con cada uno de ellos bajo las mismas condiciones: se ha empleado el mismo dataset de imágenes (de tamaño 1944x1944 píxeles), se han realizado los mismos aumentos de datos y se han elegido los mismos valores para los parámetros de entrenamiento.

Se ha entrenado el modelo durante 150 épocas, que a la vista de los resultados se han considerado más que suficientes. También se ha elegido un *batch size* o tamaño de lote de 5 imágenes por iteración, esto es, en cada época el modelo va tomando las imágenes de 5 en 5 hasta llegar a utilizar todas para el entrenamiento. Además, ambos modelos estaban preentrenados con el dataset COCO.

Una de las limitaciones que se encontró fue que Faster R-CNN solo trabaja con imágenes de tamaño comprendido entre 800 y 1333 píxeles, mientras que YOLO V5s puede trabajar con imágenes de cualquier tamaño mientras este tamaño sea múltiplo de 32. En caso de que esta condición no se cumpla, el algoritmo escala directamente las imágenes al múltiplo inmediatamente mayor, en este caso 1952 píxeles. Para solventar esta diferencia de funcionamiento de las arquitecturas, se ha desarrollado una función que obtiene el factor de escala que emplea YOLO y recorta las imágenes al tamaño necesario para que, tras emplear el mismo factor de escala, obtenga una imagen cuadrada y centrada de tamaño 1333 × 1333 píxeles. En este caso, la ecuación 5 muestra a qué tamaño se deben recortar las imágenes del dataset empleado.

$$Tamaño\ de\ imágenes\ recortadas = 1333 \cdot \frac{1944}{1952} = 1327\ píxeles$$

Ecuación 5: Cálculo del tamaño al que se deben recortar las imágenes

Con esto se consigue que los nematodos tengan el mismo tamaño en píxeles en ambos entrenamientos. Puesto que la mayoría de los nematodos en las imágenes se posicionan próximos al centro y se alejan de los bordes, apenas se pierde algún *C. elegans* debido al recorte, por lo que esta solución es válida.

Tras realizar el entrenamiento y validación, se han empleado imágenes de test para comprobar el funcionamiento de los modelos. Posteriormente, se ha procedido a evaluar el rendimiento de ambos en las tres fases, utilizando las métricas de evaluación, especialmente el mAP [0.5] y el coste computacional, para comparar las dos arquitecturas.

Tabla 1: Resultados de entrenamiento

Modelo	Parámetros	mAP@0.5	mAP@[0.5-0.95]
YOLO V5s	7.5M	0.961	0.762
Faster R-CNN	41.3M	0.983	0.759

Tabla 2: Resultados de validación

Modelo	mAP@0.5	mAP@[0.5-0.95]
YOLO V5s	0.957	0.776
Faster R-CNN	0.947	0.705

Tabla 3: Resultados de test

Modelo	tiempo/predicción (s)	mAP@0.5	mAP@[0.5-0.95]
YOLO V5s	0.00603	0.932	0.806
Faster R-CNN	0.04325	0.944	0.691

Los resultados muestran que ambos modelos funcionan de forma parecida, ya que los valores de rendimiento son bastante similares. Sin embargo, YOLO V5s es capaz de realizar detecciones casi 10 veces más rápido, además que el tiempo de entrenamiento es considerablemente menor debido al pequeño tamaño de la red. Además, Faster R-CNN presenta un tiempo por detección de 43 ms, por lo que estaría en el límite para aplicaciones en tiempo real (podría emplearse para alguna aplicación específica que emplee menores frecuencias de captura), mientras que YOLO V5s podría aplicarse sin problema. Es por esto que se ha seleccionado YOLO V5s para realizar el resto de entrenamientos y comprobar la hipótesis de si es posible obtener un sistema genérico de detección.

5. Justificación de la solución adoptada

5.1. Material empleado

5.1.1. Material informático

5.1.1.1. Hardware

El entrenamiento de las redes neuronales se ha realizado en el entorno de cálculo Kubeflow del Instituto ai2, que está compuesto por 16 máquinas con las siguientes características:

- Procesador Ryzen 9 3900X – 12 Cores @ 3.8 Ghz
- 128Gb de Memoria DDR4 3200MHz
- 2 GPUs Nvidia RTX 3090 24GB DDR4
- Capacidad de almacenamiento 2Tb

Y otra máquina que hace las labores de Controlador de características:

- Procesador Intel Core i9 11900K 3.5Ghz
- 64Gb de Memoria DDR4 3200MHz
- Capacidad de almacenamiento 9Tb

Estas 17 máquinas están configuradas en un cluster de Kubernetes versión 1.20.7.

Para el entrenamiento de la red neuronal es necesario que los equipos cuenten con GPU, ya que trabajan a mayor velocidad y permiten reducir considerablemente los tiempos de entrenamiento. En las fases de validación y test también se reducen los tiempos, pero la diferencia no es tan notable como en el entrenamiento, puesto que esta fase puede durar varias horas o incluso días, dependiendo del número de imágenes del dataset, el tamaño, la memoria RAM disponible y otros hiperparámetros de entrenamiento que desarrollaremos más adelante.

5.1.1.2. Entorno de programación

El entorno de Kubeflow se basa en Notebook Servers, que son similares a máquinas virtuales. Para usarlos, primero se deben elegir las especificaciones de hardware para dicho notebook, memoria RAM, GPUs y CPUs a utilizar y tamaño de disco para guardar las imágenes de *C. elegans*, modelos, etc. Después, se debe elegir la imagen de Docker. Una imagen de Docker se trata de un archivo formado por diversas capas y que permite ejecutar un código de programación. Las imágenes, según la aplicación, incluyen las bibliotecas, códigos de aplicación y demás archivos necesarios [27].

En la Fig. 17 se pueden comprobar las características de algunos de los Notebooks generados para este proyecto.

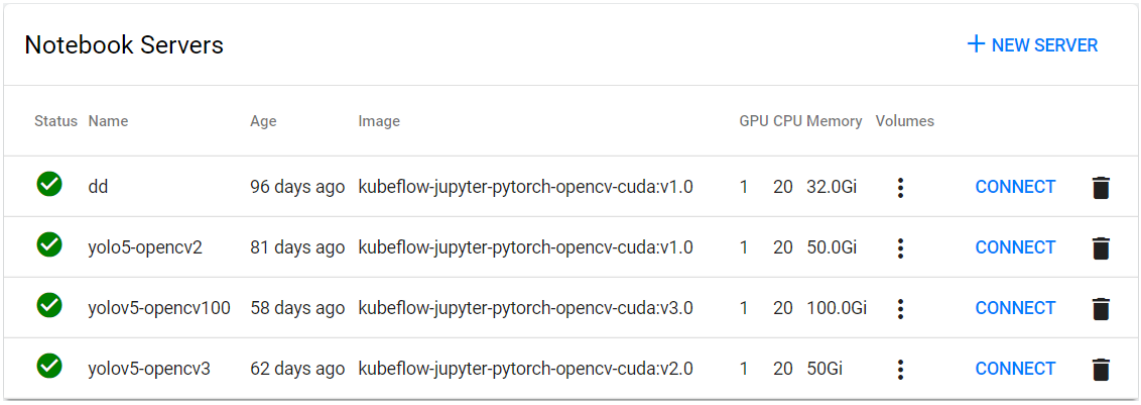

Fig. 17: Notebook Servers generados para el entrenamiento de la red neuronal

La imagen seleccionada es una imagen que permite utilizar los módulos Pytorch y OpenCV, que explicaremos a continuación.

El lenguaje de programación empleado para este proyecto es Python, ya que es un lenguaje de código abierto y de alto nivel, lo que permite aprenderlo fácilmente. Además, se trata de un lenguaje de paradigmas múltiples, que permite programación estructurada, funcional y orientada a objetos [28], [29].

La interfaz de programación que utiliza el cluster es Jupyter. En la Fig. 18 se puede observar cómo funciona esta interfaz. El código se desarrolla en celdas que se ejecutan individualmente. Una de las desventajas de este entorno es que no permite realizar *debug* del código para encontrar los errores que no se observan a simple vista, algo que sí ofrecen otros entornos como Pycharm.

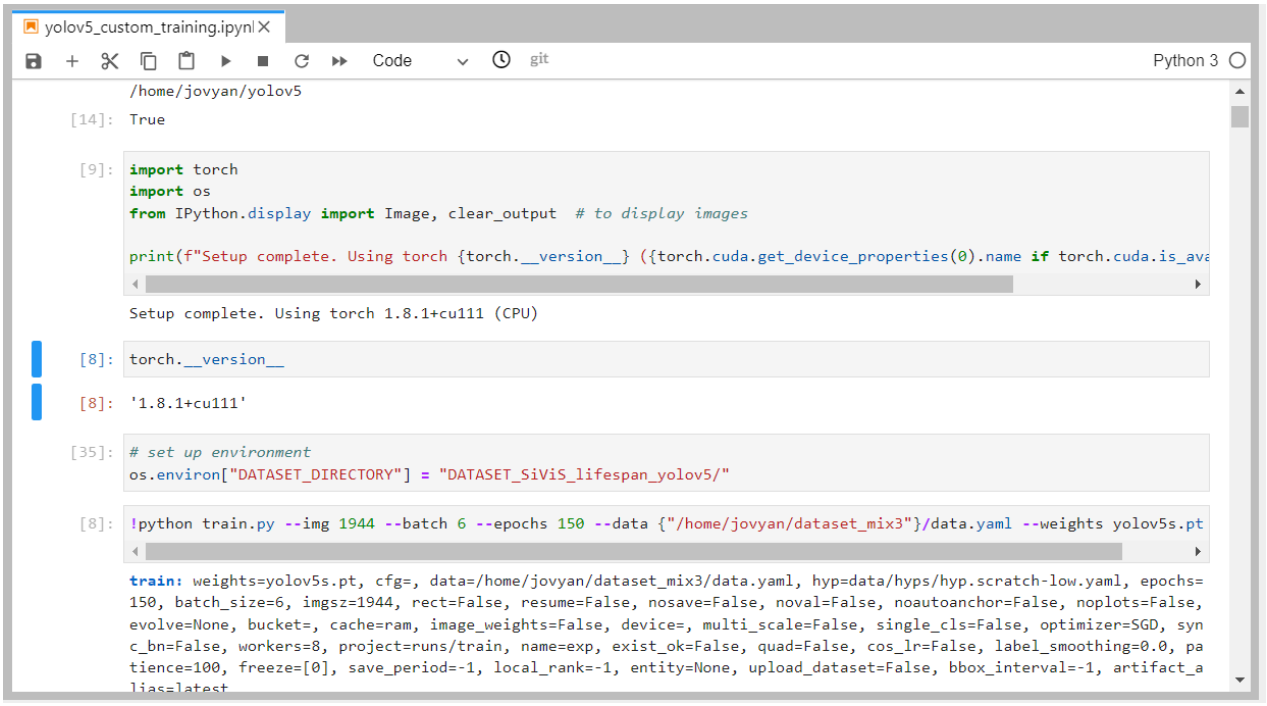

Fig. 18: Entorno de programación Jupyter

5.1.1.3. Software

Pytorch

Como ya se ha mencionado, el lenguaje de programación empleado es Python. Además, se ha utilizado la plataforma de Aprendizaje profundo Pytorch, un marco de aprendizaje automático y de código abierto basado en la biblioteca de Torch. Es comúnmente utilizado en aplicaciones de visión artificial basadas en *Machine Learning* [30].

OpenCV

Una de las librerías empleadas es OpenCV, enfocada especialmente a la visión artificial y al *machine learning*. Proporciona herramientas para procesamiento de imágenes y contiene funciones para diversas operaciones con imágenes, tales como el recorte, escalado, añadir formas, etc. Además, incluye interfaces para C++, Python, Java y Matlab y es compatibles con los principales sistemas operativos [31].

PIL

La librería Pillow o PIL (Python *Imaging Library*) permite procesamiento de imágenes como OpenCV, pero es exclusiva de Python. En la mayoría de los casos se pueden usar ambas indistintamente, según la función que se quiera emplear. Sin embargo, los *dataloaders*, que son archivos de código cuya función es cargar las imágenes y aplicar las transformaciones necesarias para el entrenamiento, validación y test, suelen

emplear esta librería. Existen funciones para transformar las variables de tipo imagen de una librería a otra, por lo que se puede trabajar conjuntamente con ambas [32].

NumPy

NumPy es una librearía que se emplea para operaciones matemáticas de todo tipo en Python. Una de las herramientas más útiles de esta librería son los *arrays* NumPy, que funcionan de forma parecida a las matrices y a los tensores, comúnmente empleados en Python. Además, las imágenes se suelen codificar como matrices, por lo que también es cómodo en ocasiones realizar operaciones con imágenes con *arrays* NumPy. Al igual que con PIL y OpenCV, existen funciones para transformar las variables de imagen de una librería a otra [33].

5.1.2. Datasets de imágenes

Para realizar los experimentos se ha hecho uso de tres datasets de imágenes diferentes. A continuación, se explican las características de los datasets, así como las condiciones de captura.

5.1.2.1. Dataset SiViS

Este dataset fue proporcionado por el Instituto ai2, capturado mediante el prototipo SiViS para experimentos de *lifespan,* por lo que se trata del dataset del que más información se tiene. Este dataset ha sido etiquetado manualmente por el ai2 y el formato de etiquetas es en forma de archivos de texto para cada imagen donde se encuentran las coordenadas de las esquinas de las cajas delimitadoras de los nematodos. Para que YOLO V5s pueda procesar estos archivos de las cajas, estas se deben transformar al formato normalizado descrito anteriormente. Para ello se ha realizado una función sencilla en Python que realiza esta conversión y sustituye los valores anteriores por los nuevos en los archivos de texto. Las expresiones empleadas son las siguientes:

$$x_{centro} = \frac{x_{máx} + x_{mín}}{2 \cdot pixels_x}$$

$$y_{centro} = \frac{y_{máx} + y_{mín}}{2 \cdot pixels_y}$$

$$width = \frac{x_{máx} - x_{mín}}{pixels_x}$$

$$height = \frac{y_{máx} - y_{mín}}{pixels_y}$$

Ecuación 6: Expresiones para la transformación de las etiquetas a formato pytorch txt

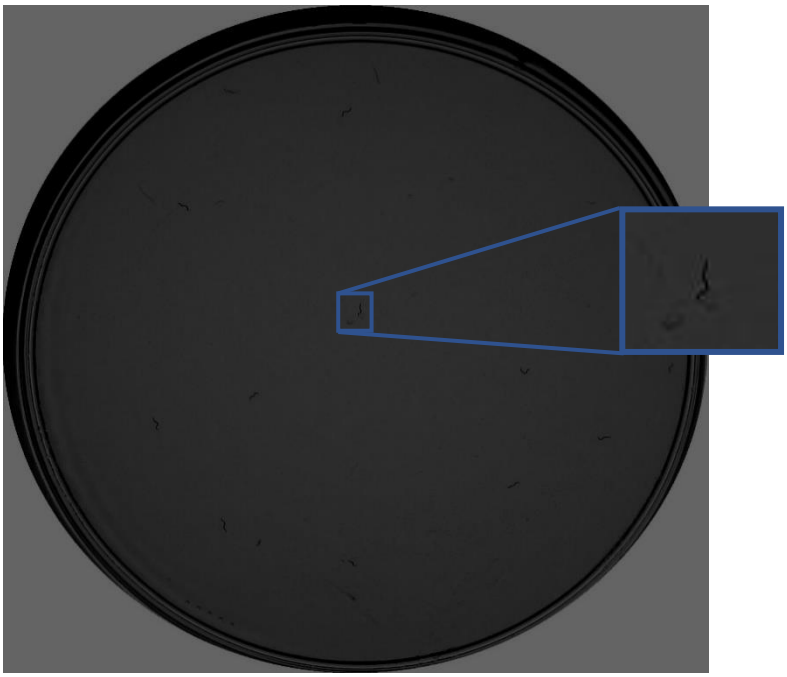

Fig. 19: Ejemplo de imagen del dataset SiViS

El dataset está formado por 1.901 imágenes divididas en tres categorías: entrenamiento, validación y test.

Los *C. elegans* empleados para los experimentos de detección fueron proporcionados por el Caenorhabditis Genetics Centre de la Universidad de Minnesota y son de la cepa N2, Bristol (wild-type y Cb1370, *daf-2*). Todos los nematodos fueron sincronizados por edad y se pipetearon en un Medio de Crecimiento de Nematodos (Nematode Growth Medium, NGM) en placas Petri de 55 mm, donde la temperatura se mantuvo a 20°C. Los nematodos se alimentaron con la cepa OP50 de *Escherichia coli*, cultivada en el centro de la placa para evitar zonas de pared ocultas, ya que los *C. elegans* tienden a permanecer en el pasto. Para reducir la contaminación por hongos, se añadió fungizone a las placas [34], y se añadió FUdR (0,2 mM) a las placas para disminuir la probabilidad de reproducción.

En el laboratorio, el operario tuvo que seguir una metodología específica para obtener las imágenes:

(1) Extraer las placas de la incubadora para ponerlas en el sistema de adquisición;

(2) examinar la tapa, antes de iniciar la captura, para comprobar que no hay condensación y limpiarla si se detecta;

(3) capturar una secuencia de 30 imágenes por placa a 1 fps y devolver las placas a la incubadora.

Este método fue diseñado para evitar la condensación en la tapa.

Método de captura de imágenes

Las imágenes se capturaron usando el sistema de monitorización desarrollado en [35], que utiliza el método de iluminación *backlight* activo propuesto en [36]. Este sistema se basa en una cámara RGB Raspberry Pi v1.3 (OmniVIsion OV5647, con una resolución de 2592 × 1944 píxeles, un tamaño de píxel de 1.4 × 1.4 µm, un campo de visión de 53.50° × 41.41°, un tamaño de lente de 1/4”, y una distancia focal de 2.9) situado en frente de un sistema de iluminación (un *display* Raspberry Pi de 7 pulgadas con una resolución de 800 × 480 a 60 fps, color RGB de 24 bits) y la placa a inspeccionar en medio de ambos dispositivos. Se utilizó una placa Raspberry Pi 3 como procesador para el control de luz. La distancia entre la cámara y la placa de Petri fue suficiente para obtener una imagen completa de la placa, y la distancia focal elegida para la cámara fue esta distancia (aproximadamente 77 mm). El sistema capturó imágenes de 1944 × 1944 píxeles a una frecuencia de 1 Hz. En la Fig. 20 se observa una representación gráfica del sistema completo de captura.

La técnica de *backlight* activo ha resultado ser efectiva para aplicaciones de *C. elegans* de baja resolución [36], [37].

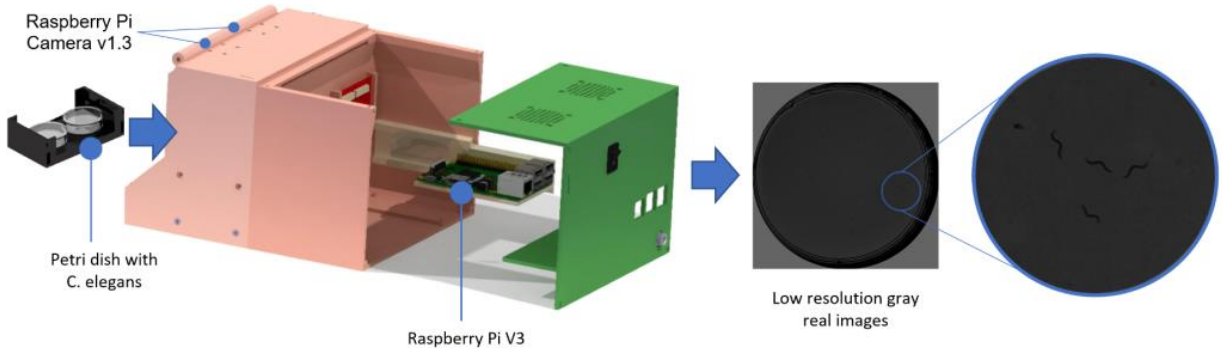

Fig. 20: Sistema de captura de imágenes, donde se observa la ubicación de los platos de Petri, así como los demás componentes [22].

5.1.2.2. Dataset DIY

Este dataset fue publicado en [16] donde las imágenes se han capturado en baja resolución mediante un microscopio casero, fabricado por los propios investigadores (*DIY*, *Do It Yourself*).

El sistema de captura de imágenes empleado (Fig. 21) se basa en una Raspberry Pi Modelo 3B+, junto con un módulo de cámara de 8 MP Raspberry Pi V2.1. Se empleó un teleobjetivo de zoom 2x, combinado con un macroobjetivo de 14x para la ampliación óptima de la placa de Petri en el sensor de la cámara. El objetivo zoom amplía la placa de Petri a la vez que permite una mayor distancia para capturar placas de Petri enteras (3,5 cm). El objetivo macro incrementa aún más el aumento y permite enfocar objetos mucho más cercanos, disminuyendo la distancia entre la placa de Petri y el sensor.

Durante la captura de imágenes, se evitó cualquier luz de fondo cerrando la unidad de captura.

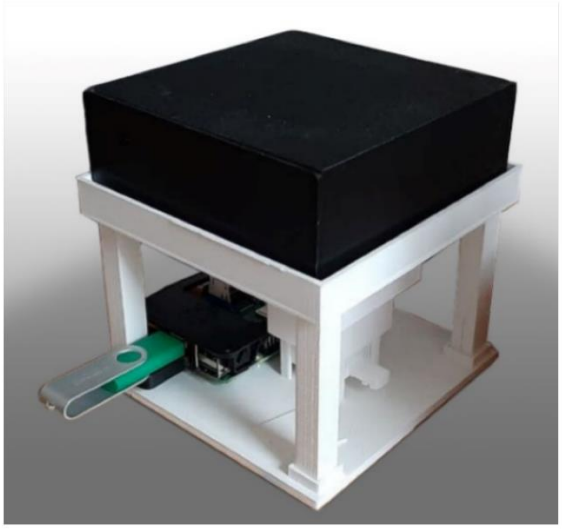

Fig. 21: Sistema completo de captura (microscopio casero) empleado [16]

En la Fig. 22 se puede observar el esquema del prototipo completo diseñado. A la izquierda se puede observar una representación gráfica del *setup* empleado para el sistema de captura, compuesto por (1) la cámara con los objetivos, (2) la placa de Petri con los *C. elegans,* (3) el sistema de iluminación, reutilizado de [38] y (4) una platina móvil para determinar la distancia correcta a los objetivos y así garantizar imágenes nítidas. Se puede comprobar en la Fig. 21 que, en el sistema final, se han intercambiado el sistema de iluminación y la cámara con los objetivos. Las imágenes se guardan mediante el procesador Raspberry Pi y, de esta forma, ya pueden utilizarse para realizar la detección de *C. elegans.* Esta parte del esquema (derecha) es parte de los experimentos que realizaron los investigadores, pero no se corresponde con este trabajo, ya que nosotros no vamos a continuar empleando una red de propuesta de región (en el artículo se emplea Mask R-CNN).

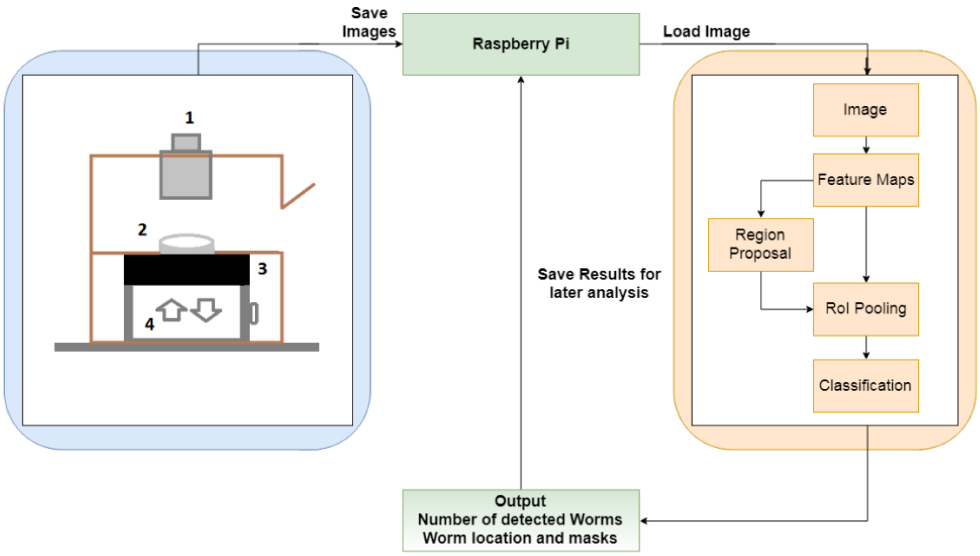

Fig. 22: Esquema del prototipo de captura y procesamiento de imágenes [16]

Las imágenes se capturaron a una frecuencia de 1 Hz, con una resolución de 3280 × 2464 píxeles y a 15 fps. Se empleó *backlight* como técnica de iluminación y una velocidad de obturación de 25 s.

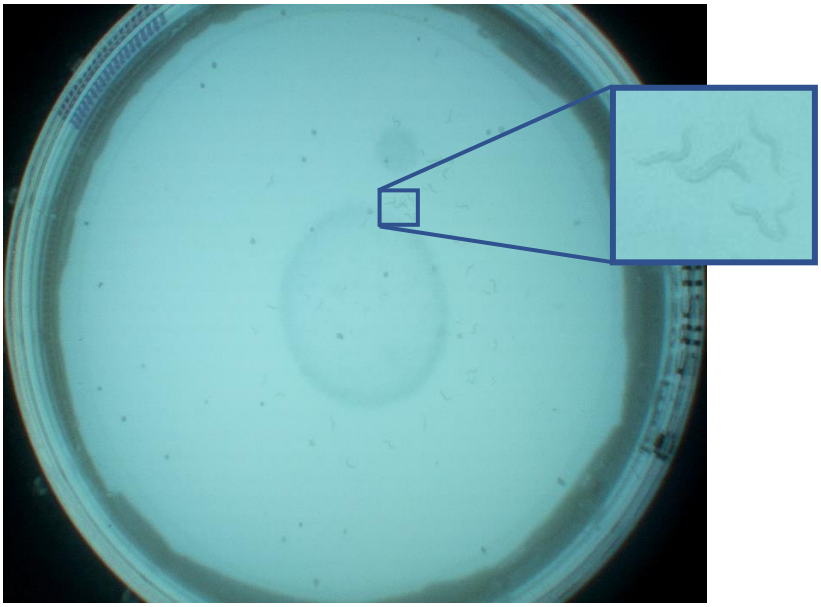

Fig. 23: Ejemplo de imagen del dataset DIY

Este dataset contiene también las máscaras (Fig. 24) de cada nematodo obtenidas mediante pre-procesos, como segmentación, binarización y detección de blobs (*binary large objects*, objetos binarios grandes que suelen ser de interés). Las máscaras son imágenes binarias donde solo aparece un *C. elegans* en blanco mientras el fondo es negro. Es por esto que a cada imagen de una placa de Petri le corresponden tantas imágenes de máscaras como nematodos aparezcan en dicha placa, por lo que la cantidad de imágenes de máscaras es elevada.

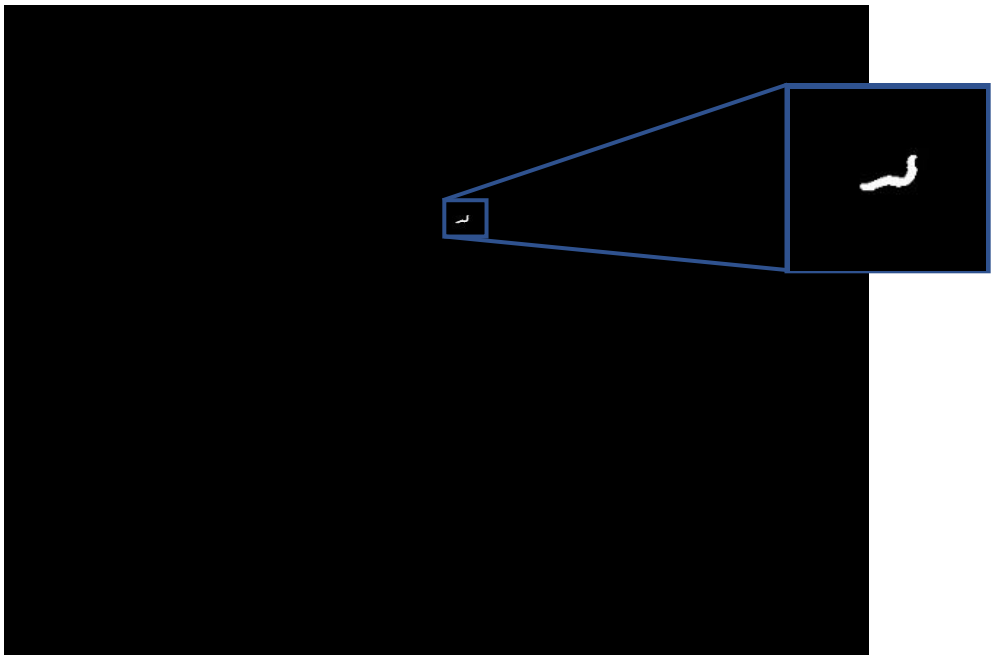

*Fig. 24: Ejemplo de máscara. Esta corresponde a uno de los nematodos de la Fig. **

El dataset está compuesto en total por 1.908 imágenes y 52.364 máscaras, divididas en entrenamiento, validación y test. Para que YOLO V5s pueda obtener la información de las etiquetas, se ha empleado una función que obtiene las coordenadas de las esquinas de las máscaras y las agrupa, convirtiéndolas al formato adecuado en un solo archivo de texto para cada imagen.

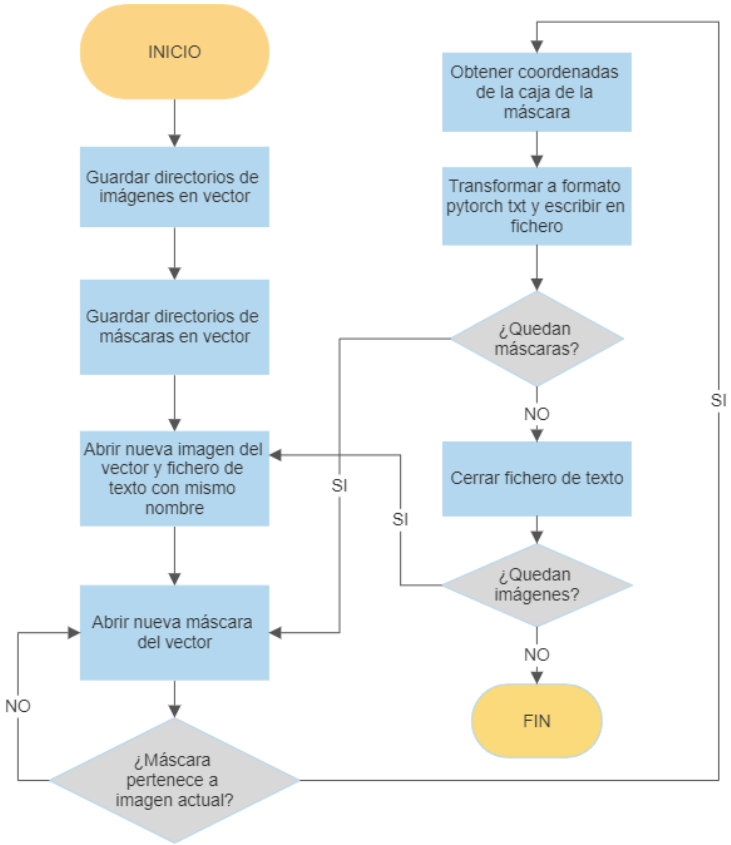

Fig. 25: Flujograma de la función de transformación de máscara a pytorch txt.

5.1.2.3. Dataset Developmental

Este dataset fue publicado en [17]. Para el ensayo se emplearon *C. elegans* de la cepa N2 y QL101 [*tph-1 (n4622) II*] y se mantuvieron a condiciones estándar de 20°C. El experimento realizado consistió en el seguimiento de un nematodo individual desde el estado de larva L2 hasta el primer día de adulto en una placa de Petri donde se cultivó alimento para el individuo. Durante el ensayo, el *C. elegans* tenía libertad para vagar por todo el campo de visión y su posición y condiciones fueron registradas continuamente en la temprana madurez.

Cuando los nematodos están en la primera etapa (L2), es cuando su tamaño es menor, por lo que un reto importante en el seguimiento fue el pequeño tamaño del individuo (a partir de unos 360 µm de longitud) y diferenciarlo del fondo debido al bajo contraste y la baja ampliación de la imagen. El ajuste de las herramientas de procesamiento de imágenes, basadas en la heurística para optimizar el pequeño tamaño y el bajo contraste de los animales jóvenes, conduce a otros desafíos a medida que el gusano crece. El contraste del fondo mejora a medida que el gusano se desarrolla; sin embargo, otros cambios sutiles en el fondo (como huevos o huellas formadas en el campo) pueden identificarse como animales cuando se utilizan las técnicas ajustadas para animales jóvenes.

Para evitar que los animales salgan del campo de visión del microscopio, se prepararon placas especiales con ácido palmítico como barrera. Estas placas se sembraron posteriormente con 10 ul de OP50 y se incubaron a temperatura ambiente durante unas 24 horas para permitir que se formara un césped fino y se almacenaron a 4 °C hasta una hora antes de su uso.

Para estos ensayos de desarrollo, se blanquearon los animales adultos para obtener los huevos. Se dejó que los huevos eclosionaran y que las larvas alcanzaran el estado larvario L1 agitando los huevos durante la noche en un *buffer* M9. A continuación, se pipetearon los L1 en una placa NGM sin sembrar y se pipetearon los animales individuales en las placas preparadas con ácido palmítico sembrado. Estas placas se parafilmaron y se incubaron a 20°C hasta que los animales alcanzaron el estadio L2 tardío (20 horas después de la siembra), momento en el que cada placa se colocó en un sistema de imágenes basado en Raspberry Pi (módulo de cámara Raspberry Pi v2). Los experimentos de desarrollo duraron 44 horas, momento en el que los gusanos alcanzaron la madurez sexual y las placas se retiraron de los sistemas de imagen.

El dataset está formado por 1.097 imágenes, divididas solamente en entrenamiento y test.

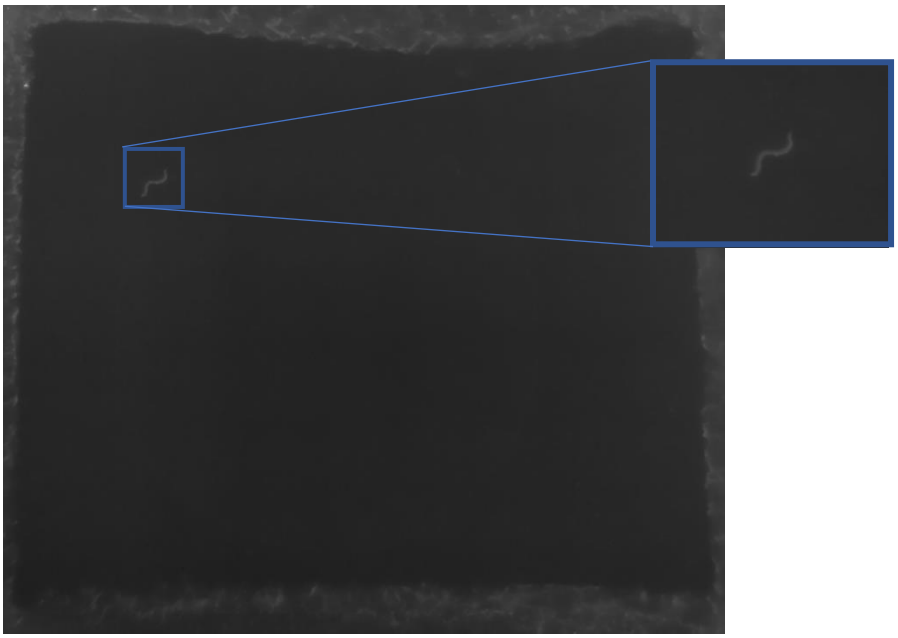

Fig. 26: Ejemplo de imagen del dataset Developmental

El dataset está etiquetado en formato xml, es decir, existe un fichero xml para cada imagen como el que aparece en la Fig.27.

```
▼<annotation>
   <folder>train</folder>
   <filename>img4.png</filename>
   <path>C:\Users\kebel\Dropbox (GaTech)\cheapscope\image annotations\images\train\img4.png</path>
  ▼<source>
     <database>Unknown</database>
   </source>
  ▼<size>
     <width>1640</width>
     <height>1232</height>
     <depth>1</depth>
   </size>
   <segmented>0</segmented>
  ▼<object>
     <name>worm</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
     <difficult>0</difficult>
    ▼<bndbox>
       <xmin>916</xmin>
       <ymin>699</ymin>
       <xmax>955</xmax>
       <ymax>743</ymax>
     </bndbox>
   </object>
  ▼<object>
     <name>arena</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
     <difficult>0</difficult>
    ▼<bndbox>
       <xmin>161</xmin>
       <ymin>104</ymin>
       <xmax>1523</xmax>
       <ymax>1068</ymax>
     </bndbox>
   </object>
 </annotation>
```

Fig. 27: Formato de etiquetas en XML

Para poder procesar el dataset, las etiquetas deben estar en un formato que pueda leer YOLO. Como en los anteriores datasets, se va a emplear el formato “pytorch txt”, para lo cual se ha desarrollado una función que realiza la transformación. Esta función es sencilla dado que solo hay un nematodo por imagen.

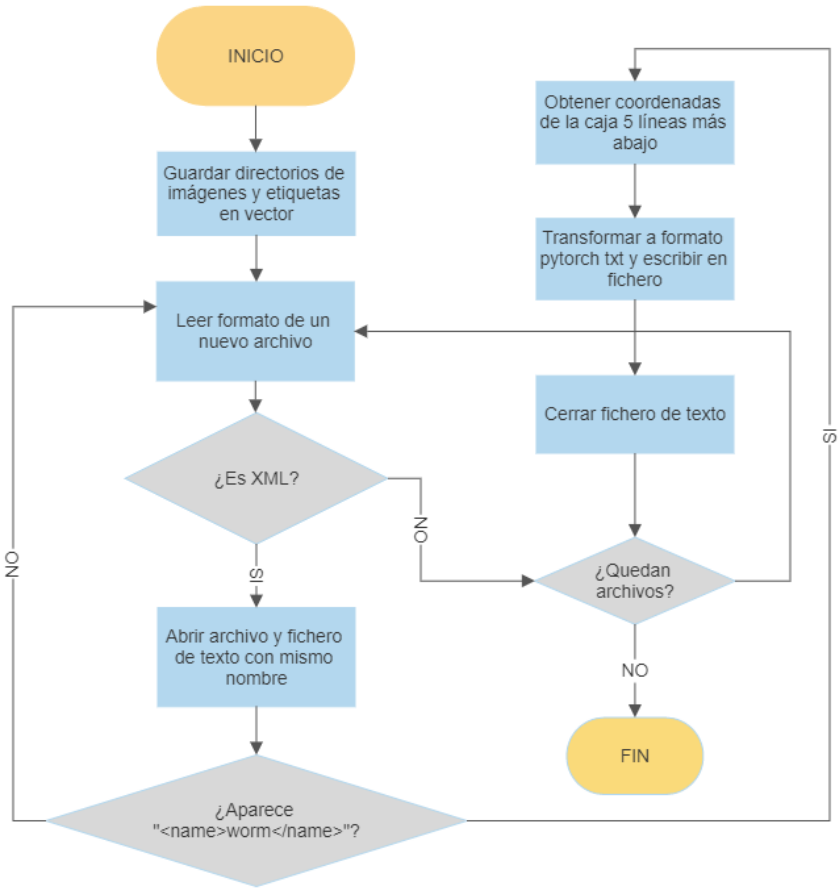

Fig. 28: Flujograma de la función de transformación de xml a pytorch txt

5.2. Entrenamiento de la red

El objetivo del entrenamiento de una red neuronal es el de ajustar los pesos de las entradas de todas las neuronas que forman la red, de forma que se vaya disminuyendo el error entre la respuesta de la capa de salida y los datos reales. Esto se consigue asignando unos pesos aleatorios al inicio y actualizándolos en cada época según el error obtenido. Es por esto que se necesita que el número de épocas de entrenamiento sea suficiente para que la red sea capaz de detectar los nematodos con bastante precisión. Tampoco conviene que el número de épocas sea muy alto ya que en este caso se obtendría un modelo sobre ajustado y, por tanto, no sería capaz de detectar *C. elegans* en condiciones diferentes, es decir, se pierde capacidad de generalización.

Para realizar el entrenamiento de la red se deben seguir los siguientes pasos:

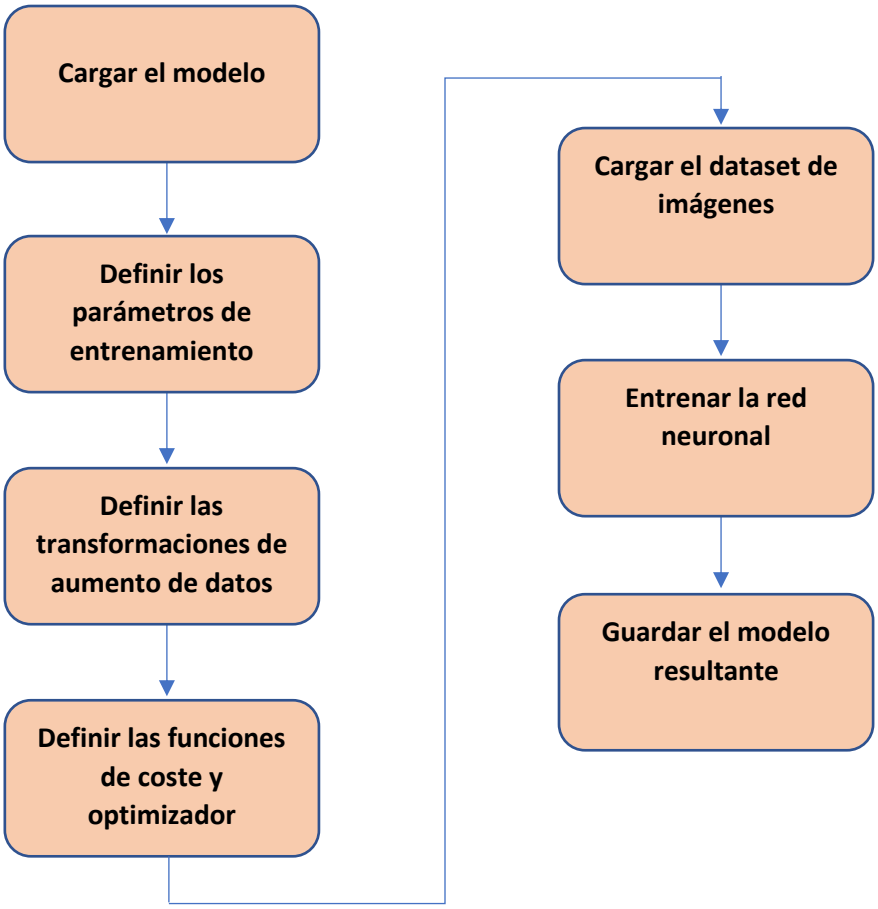

Fig. 29: Esquema de entrenamiento de la red neuronal

De la misma forma que en el entrenamiento realizado para la comparación entre Faster R-CNN y YOLO V5s, para todos los experimentos que se desarrollan a continuación se ha optado por entrenar durante 150 épocas, ya que se ha comprobado que es suficiente para conseguir buenos resultados sin sobre ajuste. El *batch size* se ha establecido en 6 (imágenes por iteración) para que el entrenamiento sea más rápido. En la comparación entre arquitecturas se eligió un valor menor porque al entrenar el modelo Faster R-CNN se necesitaba mayor memoria y no se podría entrenar con un tamaño de lote mayor.

Existen otros parámetros en el entrenamiento que se deben ajustar para conseguir mejoras en los entrenamientos y evitar posibles fallos. Estos parámetros se pueden ir variando para comprobar sus efectos, aunque en este trabajo se ha optado por mantener los valores por defecto, que son los que los desarrolladores han elegido por conseguir buenos resultados. A continuación, se explican los principales hiperparámetros y sus valores en los entrenamientos que se van a realizar:

- Optimizador SGD con momento: es un algoritmo que ayuda a acelerar los vectores gradientes en las direcciones correctas, de forma que converjan más rápido [39]. En este caso se ha elegido un valor de 0.937.

- *Weight decay*: es una técnica de regularización que sirve para evitar el problema de *overfitting* o sobre ajuste de la red con los datos de entrenamiento [40]. El valor empleado para el entrenamiento es de 0.0005.
- *LambdaLR Learning Rate*: este parámetro regula cuánto se debe cambiar el modelo según el error estimado de las actualizaciones de los pesos en cada época. Por tanto, influye en la velocidad para conseguir los pesos idóneos. Un valor muy pequeño puede provocar que el entrenamiento sea demasiado largo e incluso se atasque, mientras un valor muy alto puede provocar un mal ajuste de los pesos o un proceso de entrenamiento inestable [41], [42]. En YOLO V5s se elige un valor de *learning rate* inicial y otro final, en este caso ambos se han mantenido a 0.1.

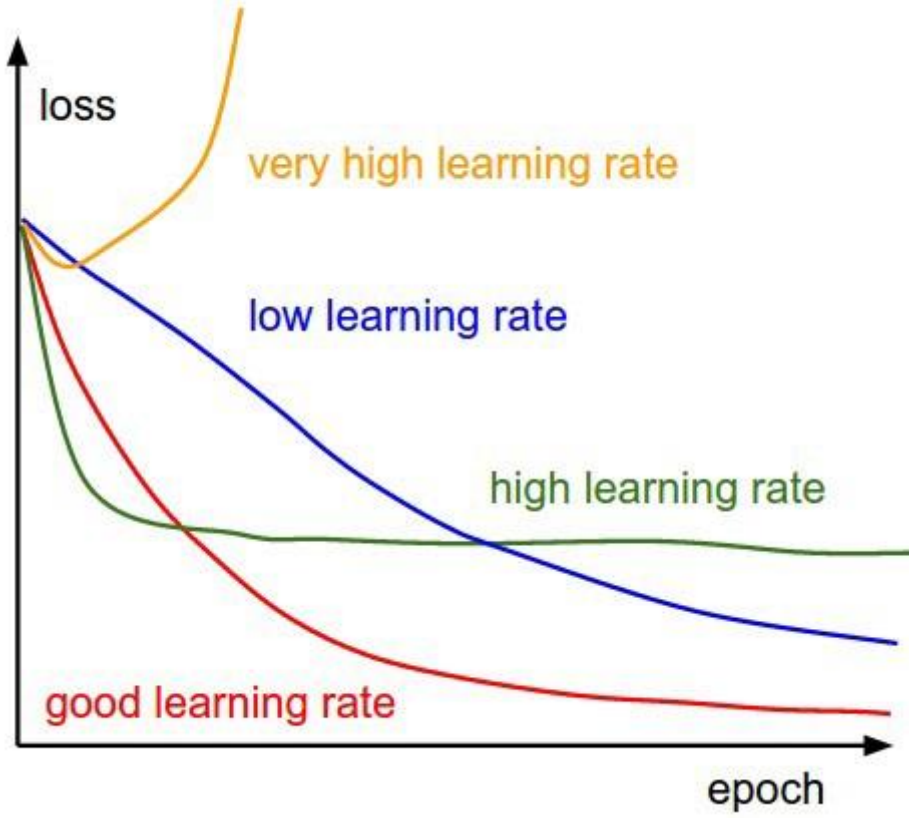

Fig. 30: Curvas que relacionan el error en la predicción y las épocas de entrenamiento para distintos valores de learning rate [42]

Por otro lado, existen ciertos parámetros que se emplean para aumento de datos, esto es, realizan transformaciones aleatorias a algunas imágenes para que, con el mismo número de imágenes a la entrada, la red entrene con una cantidad mayor ya que se utilizan las imágenes originales y las transformadas. Esto permite también que la red aprenda mejor la forma de los nematodos al entrenar con imágenes en condiciones un tanto dispares. Con estos aumentos de datos se mejora el entrenamiento en general, pero se debe ser cuidadoso de no elegir valores demasiado grandes, se debe estudiar con qué valores se mejora el entrenamiento. Algunos de estos parámetros son:

- Aumentos HSV: este parámetro realiza transformaciones de color a las imágenes según el modelo HSV: *Hue, Saturation, Value* (Matiz, Saturación, Valor o Brillo). Se ha elegido unos valores de *hue* = 0.015, *saturation* = 0.7 y *value* = 0.4.
- Flip vertical: el valor que se asigna a este parámetro indica con qué probabilidad se aplica un volteo en vertical a las imágenes. El valor empleado es de 0.5, es decir, el 50% de probabilidad.
- Image scale and translation: estos parámetros realizan transformaciones geométricas de escalado y translación: 0.5 y 0.1, respectivamente. Existen otras como giro y perspectiva, pero se han mantenido nulos.

5.3. Método experimental

Los experimentos realizados para tratar de conseguir un sistema genérico de detección son los siguientes:

5.3.1. Experimento 1

Se ha entrenado la red con el dataset SiViS y se ha comprobado su funcionamiento en el dataset DIY para comprobar el desempeño de la red en nuevas condiciones. Con este experimento se pretende observar si la red es capaz de aprender únicamente la forma del nematodo o si atiende más a otras características como el contraste con el fondo o la iluminación. De esta forma, podremos elegir los datasets de entrenamiento de forma estratégica para intentar que la red aprenda principalmente la morfología de los *C. elegans.*

La proporción de imágenes empleada para entrenamiento, validación y test es la que aparece en la tabla 4.

Tabla 4: Cantidad y proporción de imágenes de cada dataset para el experimento 1

	Entrenamiento	Validación	Test
Cantidad de imágenes	1330	190	381
Porcentaje	70%	10%	20%

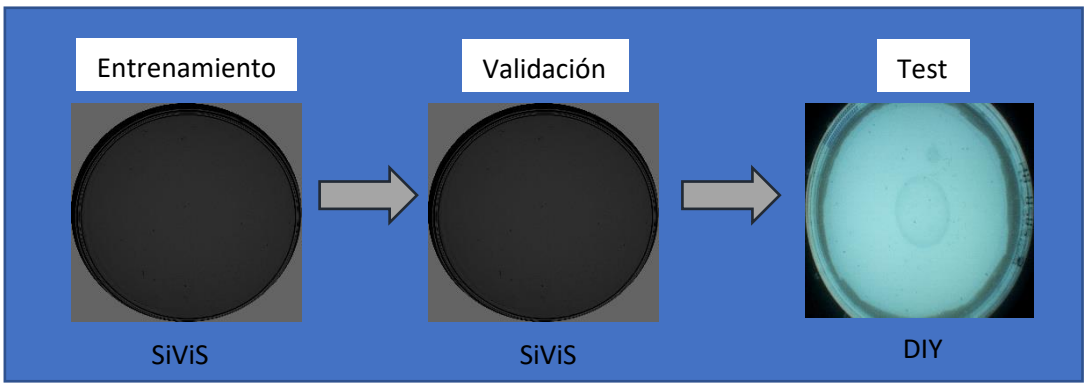

Fig. 31: Esquema del experimento 1

5.3.2. Experimento 2

Se ha obtenido un nuevo dataset juntando las imágenes de ambos datasets (SiViS y DIY) para que la red entrene con este, de forma que ya presente dos canales distintos y sea capaz de detectar *C. elegans* en dos tipos diferentes de captura de *C. elegans*. Con esto se da un paso más para conseguir un sistema generalizable.

Una vez entrenada la red, se probará con el dataset *developmental* para comprobar si el nuevo entrenamiento se puede aplicar a otro tipo de imágenes, nuevas para la red. Este dataset, pese a tener unas condiciones de iluminación similares al dataset SiViS (fondo oscuro), presenta características nuevas que pueden dificultar la tarea de detección de la red, ya que los nematodos son más pequeños y su contraste con el fondo es bastante menor, haciéndolos imperceptibles para el ojo humano en algunas imágenes.

A la hora de unir ambos datasets, para darle el mismo peso a cada uno, se ha utilizado la misma cantidad de imágenes para el entrenamiento, utilizando las restantes para test. De esta forma la proporción de imágenes para cada uno de los procesos es la que aparece en la tabla 5.

Tabla 5: Cantidad y proporción de imágenes de cada dataset para el experimento 2

	Entrenamiento	Validación	Test
Imágenes SiViS	1332	190	379
Imágenes DIY	1332	192	384
Total Imágenes	2664	382	763
Porcentaje	70%	10%	20%

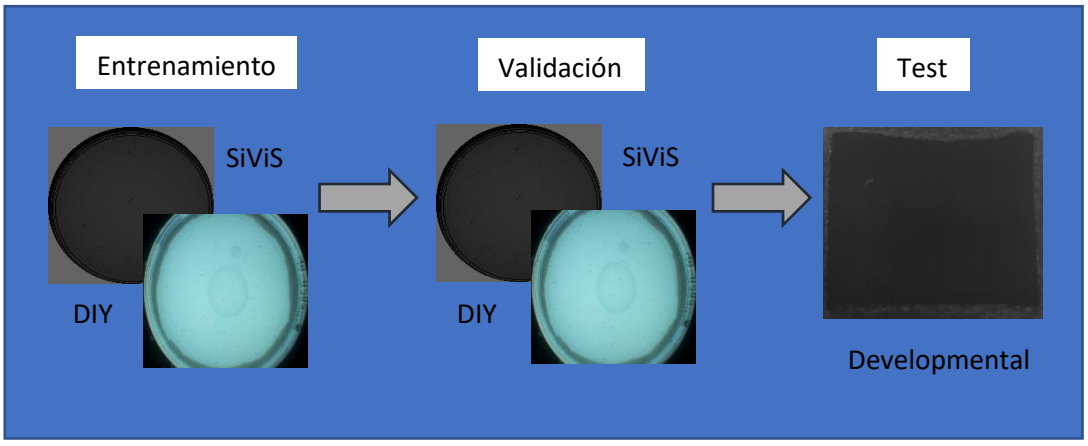

Fig. 32: Esquema del experimento 2

5.3.3. Experimento 3

En este experimento, la red se ha entrenado con los tres datasets. Este experimento es el más importante, puesto que se trabaja con tres canales diferentes de imágenes de *C. elegans* y los tamaños de dataset son bastante grandes. En este ensayo no se va a probar el funcionamiento de la red con imágenes nuevas debido a la falta de datasets publicados y etiquetados. Además, debemos ser cautos a la hora de elegir los datasets con los que intentar conseguir el objetivo final de obtener un sistema genérico de detección, no tendría sentido únicamente juntar datasets al azar y realizar entrenamientos con ellos.

En este caso, si se consigue que la red sea capaz de detectar correctamente *C. elegans* en los tres canales diferentes, sería un buen resultado, ya que lo más común es que las condiciones de captura de las placas sean similares a alguno de los tres datasets empleados para el entrenamiento.

De la misma forma que en el experimento anterior, es necesario que en el entrenamiento se empleen la misma cantidad de imágenes de cada dataset. En este caso, la diferencia de imágenes de entrenamiento del dataset developmental con respecto a los demás es bastante mayor, por lo que las imágenes restantes de los otros datasets se han dividido entre validación y test y, por tanto, la proporción de imágenes en cada proceso ahora cambia considerablemente. Este cambio es adecuado puesto que en este caso es prioritario que el entrenamiento se realice con la misma cantidad de imágenes. Para repartir las imágenes sobrantes entre validación y test se ha mantenido la relación entre la cantidad de imágenes de validación y de test, es decir, el doble de imágenes de test respecto a validación. En la tabla 6 se pueden observar las nuevas proporciones y cantidades de imágenes.

Tabla 6: Cantidad y proporción de imágenes de cada dataset para el experimento 3

	Entrenamiento	Validación	Test
Imágenes SiViS	768	377	756
Imágenes DIY	768	380	760
Imágenes Deveopmental	768	110	219
Total Imágenes	2304	867	1735
Porcentaje	47%	17.7%	35.4%

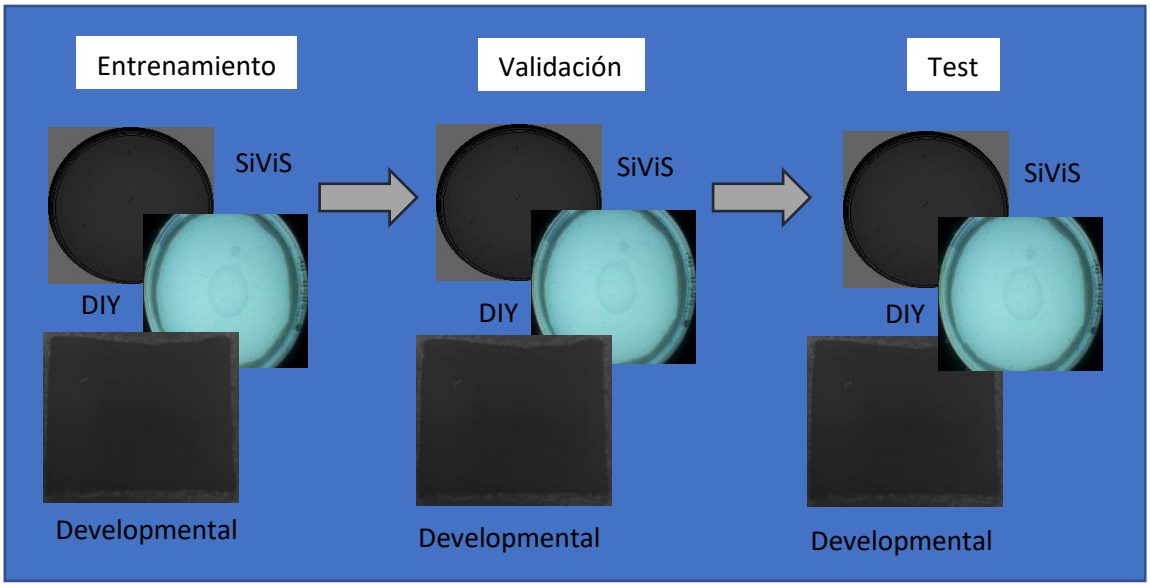

Fig. 33: Esquema del experimento 3

6. Resultados

6.1. Resultados del método experimental

Los resultados obtenidos en cada uno de los experimentos son los siguientes:

6.1.1. Experimento 1

Tabla 7: Resultados del experimento 1

	Precision	Recall	mAP [0.5]	mAP [0.5; 0.95]
Entrenamiento	0.95	0.912	0.961	0.762
Validación	0.946	0.915	0.957	0.776
Test	0	0	0	0

El repositorio de YOLO V5s está diseñado para proporcionar gráficas con las métricas del proceso realizado, así como una matriz de confusión y ejemplos de imágenes con las predicciones. Además, en el entrenamiento incluye un fichero de datos con los valores de las métricas en cada época del proceso. A continuación, se pueden observar algunas de las gráficas que proporciona YOLO en el entrenamiento y validación del primer experimento.

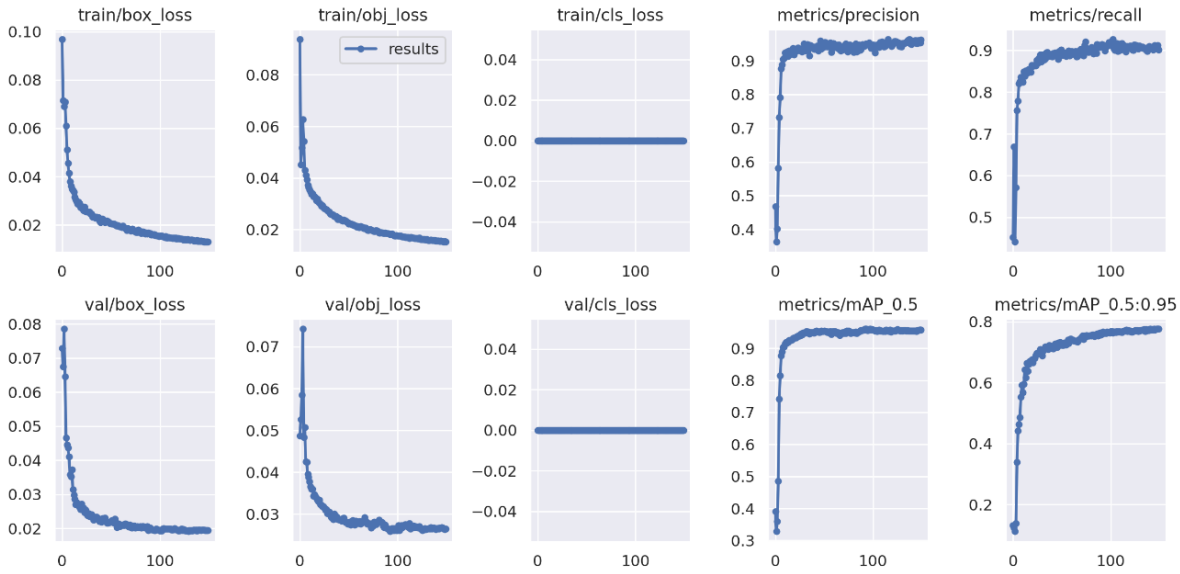

Fig. 34: Gráficas proporcionadas por YOLO en el primer experimento

Las funciones de *loss* o pérdida evalúan el error entre las predicciones de la red y los datos reales, a lo largo de las épocas de entrenamiento.

En este caso, las gráficas *box_loss* corresponden a la función de pérdida de caja, es decir, el error en términos de tamaño y ubicación entre la caja predicha y la real. Le evolución de este parámetro en el entrenamiento es tal como se esperaba, va disminuyendo exponencialmente en las primeras épocas y, posteriormente, va disminuyendo lentamente. Según los niveles de precisión deseados, se podrían realizar más o menos épocas de entrenamiento. En la validación realizada en cada época, los resultados son muy similares.

Las funciones *obj_loss* muestran el error en la confianza de la detección de un objeto en una caja. Cuando se realiza una detección con su correspondiente caja delimitadora, la red calcula la confianza de que el objeto detectado sea de una clase determinada, esta es la confianza a la que hace referencia esta función. Está relacionada con la anterior, ya que cuanto mejor ajustada está una caja delimitadora, mayor será el nivel de confianza de la detección. Esto se observa en las gráficas de *obj_loss* del primer experimento, donde la tendencia y la forma son muy similares a las de *box_loss*.

Por último, las funciones *cls_loss* muestran el error en la clasificación de un objeto detectado. En este caso, tiene sentido que sean nulas durante todo el entrenamiento y validación puesto que solo existe una clase y no se ha generado una clase especial para el fondo. Por tanto, es evidente que todas las predicciones que se hagan serán de la clase *C. elegans* y por tanto no habrá error de clasificación.

Por otro lado, las métricas que se observan a la derecha de la Fig. 34, son las ya explicadas anteriormente. Su evolución es la deseada, crecen considerablemente en las primeras épocas y después van creciendo lentamente.

6.1.2. Experimento 2

Tabla 8: Resultados del experimento 2

	Precision	Recall	mAP [0,5]	mAP [0,5:0,95]
Entrenamiento	0.939	0.928	0.969	0.655
Validación	0.938	0.92	0.961	0.676
Test	0	0	0	0

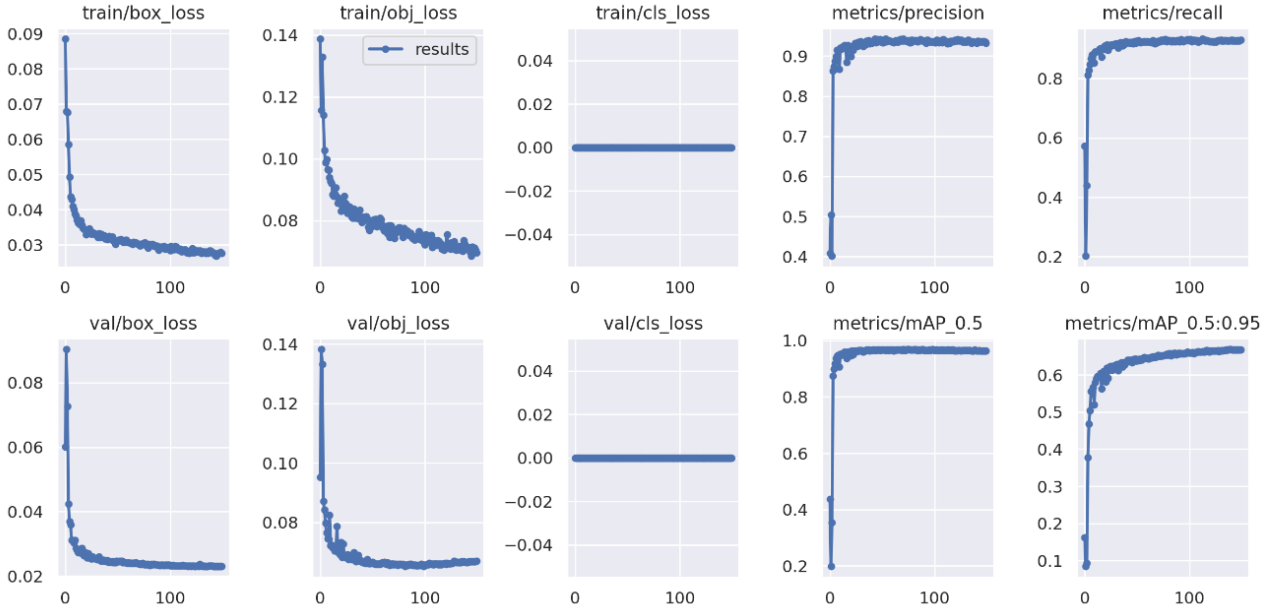

Fig. 35: Gráficas proporcionadas por YOLO en el segundo experimento

En este experimento, se comprueba que las funciones de *obj_loss* son algo diferentes a las de *box_loss*. Esto se deberá, probablemente, a que se ha introducido un nuevo tipo de imágenes y, por tanto, la red es capaz de predecir las cajas igual que en el experimento anterior pero la confianza es ligeramente inferior al existir dos tipos de *C. elegans* de la misma clase. Es por esto que las líneas de esta gráfica y de las métricas son un poco más gruesas, implica que los valores que toman las funciones varían más y no se mantienen tan estables entre épocas consecutivas.

6.1.3. Experimento 3

Tabla 9: Resultados del experimento 3

	Precision	Recall	mAP [0,5]	mAP [0,5:0,95]
Entrenamiento	0.935	0.912	0.963	0.635
Validación	0.937	0.915	0.958	0.665
Test	0.933	0.91	0.931	0.67

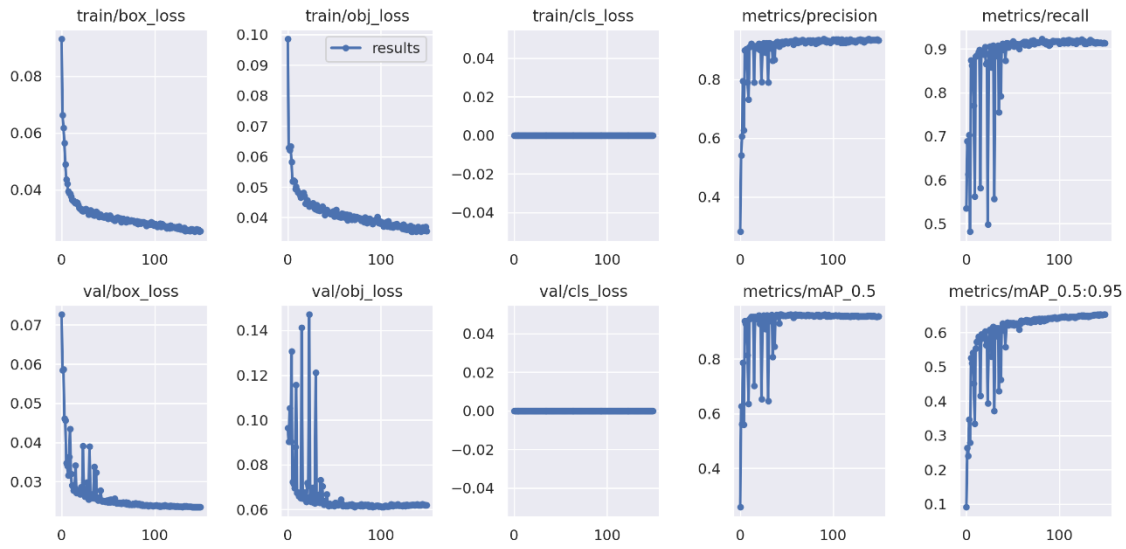

Fig. 36: Gráficas proporcionadas por YOLO en el tercer experimento

En este último experimento, se observa que la función de *obj_loss* en el entrenamiento es similar, de nuevo, a la de *box_loss*, algo que sorprende un poco al observar los cambios en las funciones de validación, donde en las primeras épocas, aproximadamente, las variaciones son más bruscas, especialmente en *obj_loss*. Este fenómeno ocurre también en las funciones de las métricas. Esto se puede deber a que se haya entrenado con más tipos de imágenes. Es muy probable que la red al principio vaya ajustando los pesos para adaptarse a alguno de los tres canales de imágenes y al realizar la siguiente comprobación del error se observe que este entrenamiento no se adapta a otro de los canales. Es por esto que el número de épocas se ha elegido lo suficientemente alto para que, finalmente, la red aprenda a detectar los nematodos en los tres tipos de imágenes.

Otra de las gráficas que proporciona YOLO y que es interesante estudiar es la curva *precision-recall.* Esta es la curva a partir de la cual se obtiene el mAP. Para que los resultados fueran ideales, esta curva debería ser cuadrada, como muestra la Fig. 37.

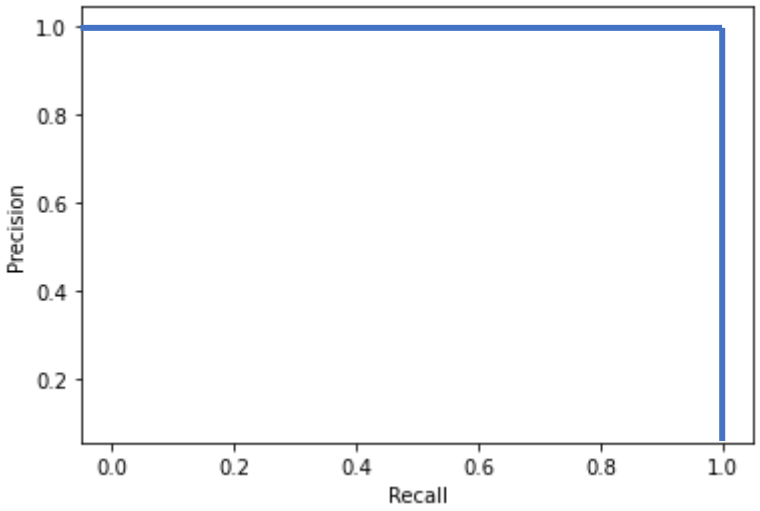

Fig. 37: Curva precision-recall de un modelo perfecto

La curva obtenida en el test del tercer experimento se aproxima considerablemente a la de un modelo ideal, por lo que podemos afirmar que el modelo funciona correctamente. En los anteriores experimentos no se ha mostrado esta curva puesto que los resultados son nulos en el proceso de test.

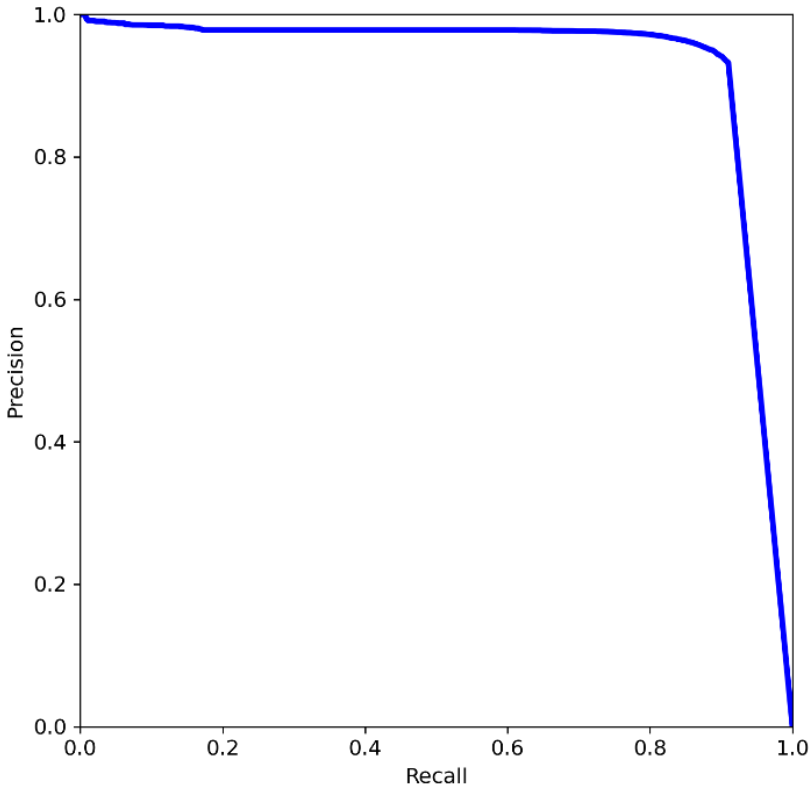

Fig. 38: Curva precision-recall del test en el experimento 3

Esta curva se genera calculando la *precision* y el *recall* para los diferentes umbrales de confianza desde 0 hasta 1. Por ejemplo, para un umbral de confianza del 0.5, se tomarán como detecciones verdaderas las que tengan un nivel de confianza mayor y como detecciones falsas las que tengan una confianza menor a este valor; a partir de esto, se calcula la *precision* y el *recall*. Cuando se van a obtener las métricas finales de test, como las que aparecen en la tabla 9, se debe elegir un único umbral de confianza. Conviene elegir el umbral de confianza que, en la curva *precision-recall*, obtenga el punto más cerca del x=1 e y=1.

6.2. Análisis de las fuentes de error

A la hora de obtener los resultados, se ha hecho uso de la programación para obtener las métricas de cada una de las imágenes, con el fin de estudiar aquellas que presenten mayores errores y así tratar de solventarlos.

Los dos posibles errores en la detección son que la red prediga que hay un nematodo donde no lo hay (error tipo 1) o que no detecte alguno de los nematodos reales (error tipo 2).

Error tipo 1

En la FIg. 39 se muestra la misma imagen, pero en la izquierda con las cajas correctas y en la derecha con las cajas predichas por la red. Se puede observar que la red es capaz de reconocer los nematodos reales, por lo que el número de falsos negativos es bajo o nulo. Sin embargo, la red predice más gusanos de los que hay presentes. Se comprueba a simple vista que el número de cajas predichas por la red (derecha, en rojo) es bastante mayor al de nematodos reales (izquierda, en blanco).

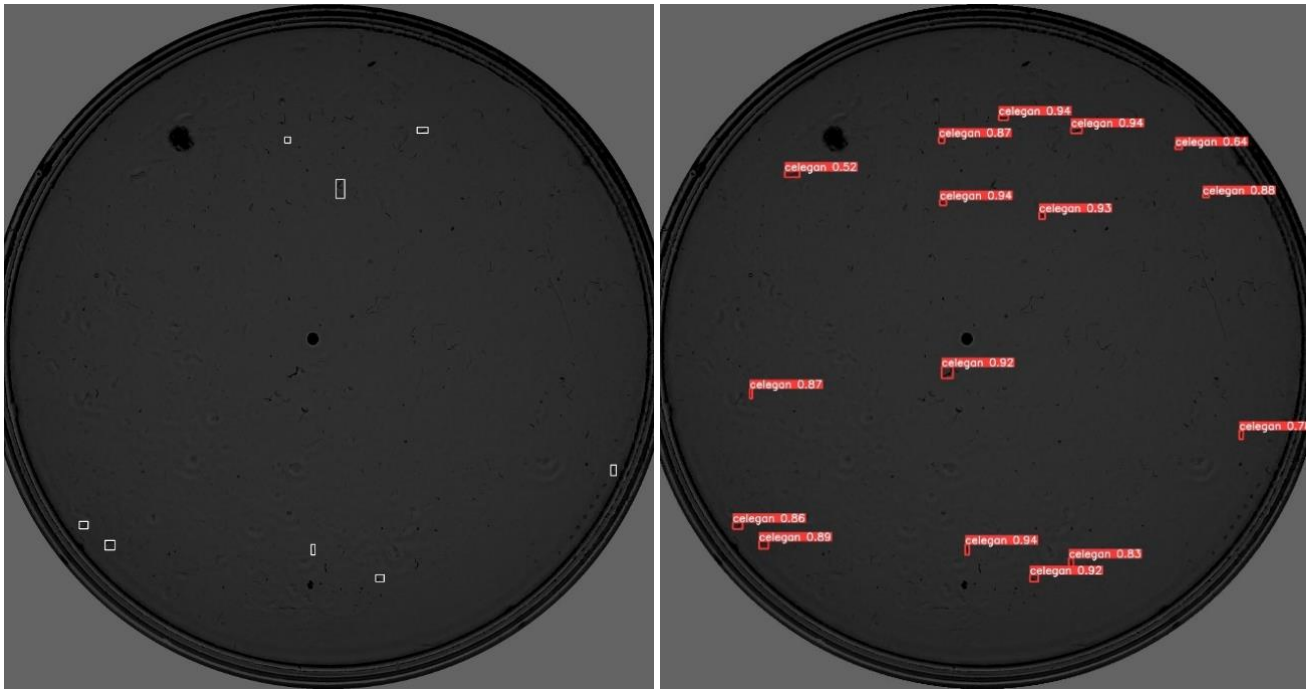

Fig. 39: Cajas correctas (izquierda) y predicciones de la red (derecha) de la misma imagen

Se ha realizado un zoom de la imagen (Fig. 40) y se ha estudiado con detenimiento para comprobar qué falla. Se ha observado que existen unas marcas en varias de las imágenes (las que más error presentan) que se parecen mucho a *C. elegans*, pero no están etiquetados como tal.

Esto se debe a que puede haber motas de polvo sobre la tapa, que tienen forma y tamaño parecidos a los *C. elegans*, pero si se observa la secuencia se puede comprobar que estas motas no se mueven en muchas de las imágenes y cuando se mueven no lo hacen de la misma forma que los nematodos. También puede deberse a que los *C. elegans* dejan un pequeño rastro en ocasiones, que es fácilmente confundible por un nematodo real. Estos suelen ser más claros, mientras que los nematodos reales tienden a presentar un contraste mayor.

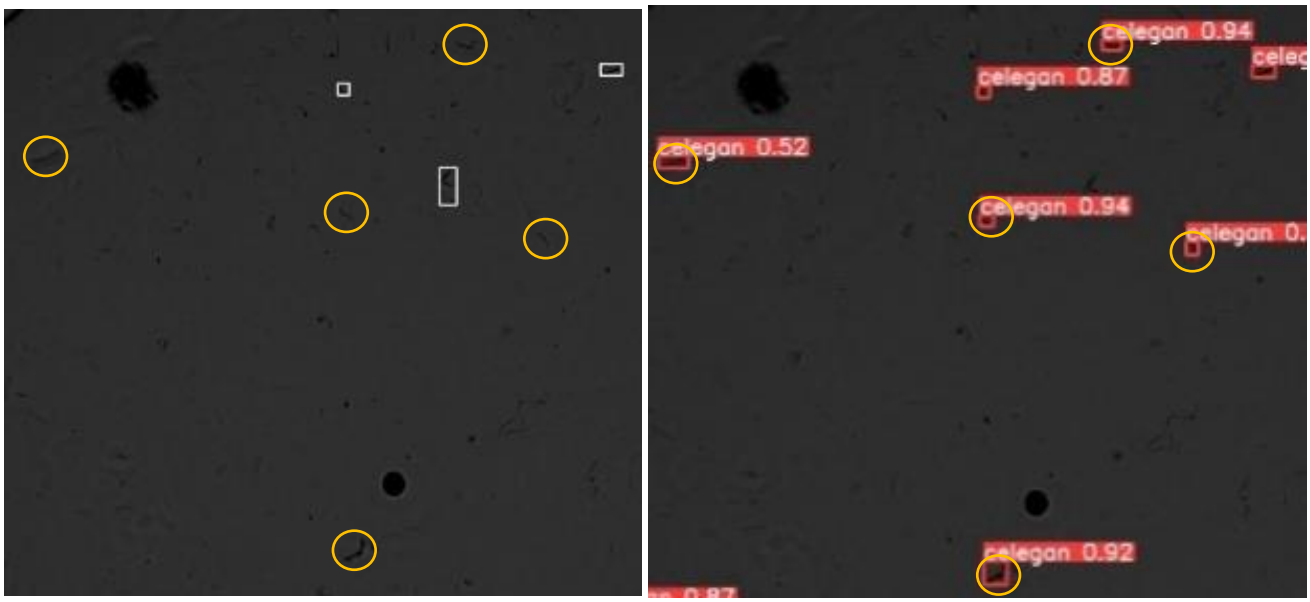

Fig. 40: Ampliación de la Fig. 39 donde se observan con mayor claridad las predicciones erróneas

El dataset DIY presenta menos imágenes con alta cantidad de fallos, pero existen algunas. En las Fig. 41 y 42 podemos comprobar algunos ejemplos de falsas detecciones de nematodos debido, seguramente, a la existencia de motas de polvo en la tapa. Debido a que las imágenes de este dataset son mayores respecto a los nematodos, se va a realizar una mayor ampliación de las imágenes para poder visualizarlas correctamente.

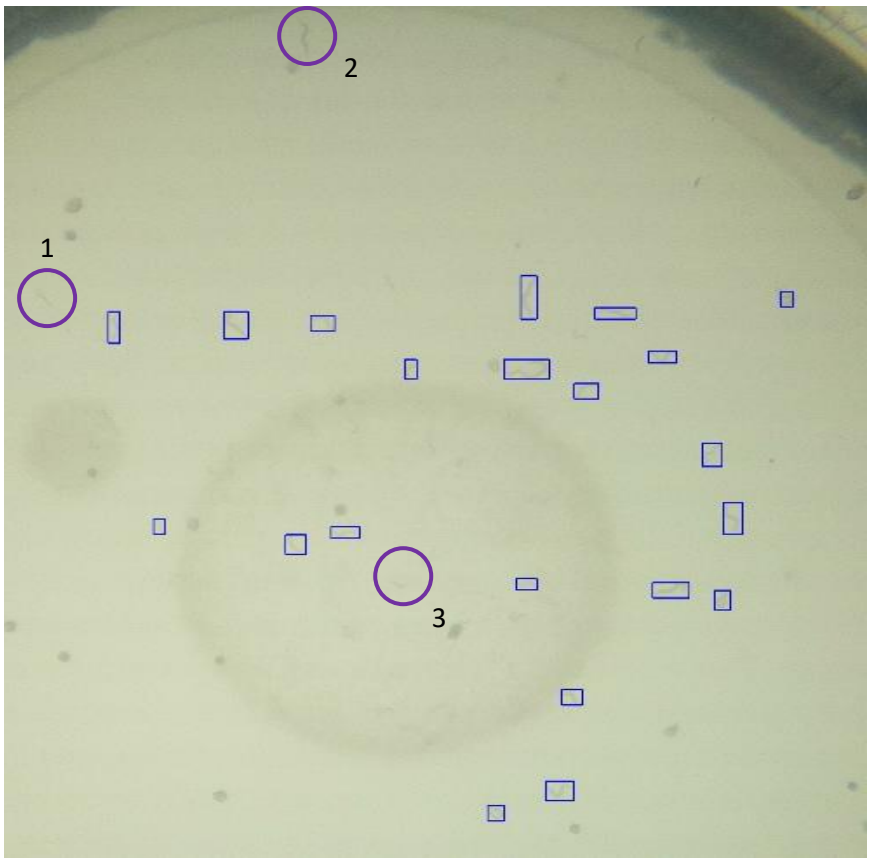

Fig. 41: Ejemplo de imagen del dataset DIY con cajas correctas

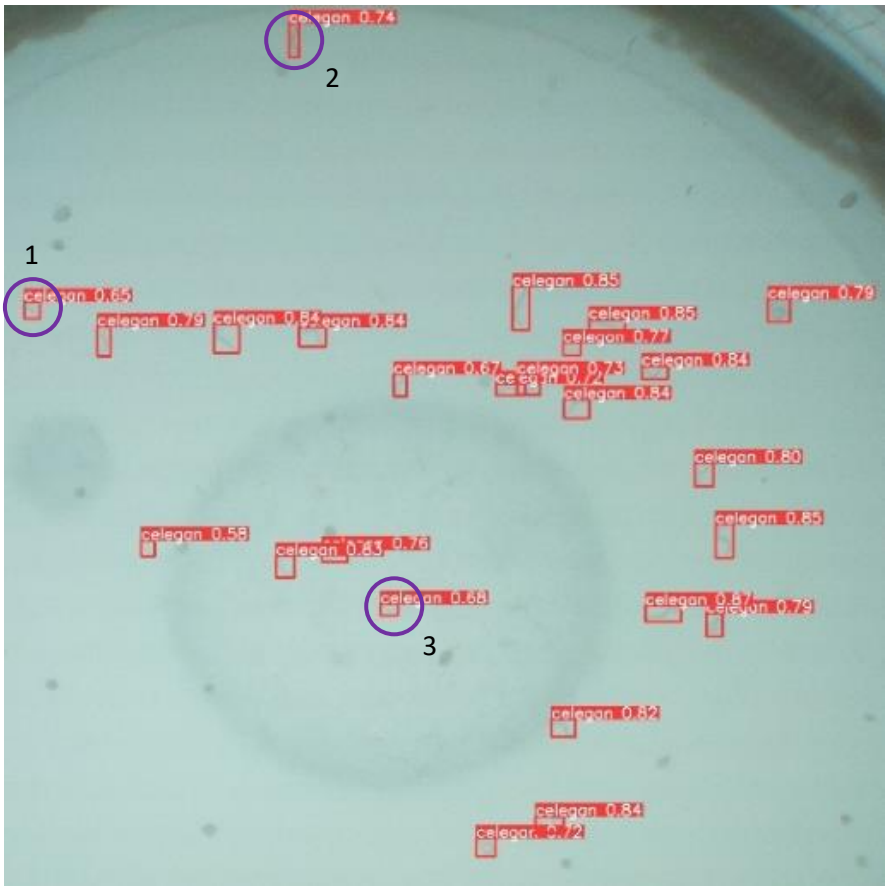

Fig. 42: Ejemplo de imagen del dataset DIY con las cajas predichas por la red

En las imágenes anteriores, se han rodeado tres ejemplos de predicciones incorrectas y a continuación se muestran esas predicciones ampliadas.

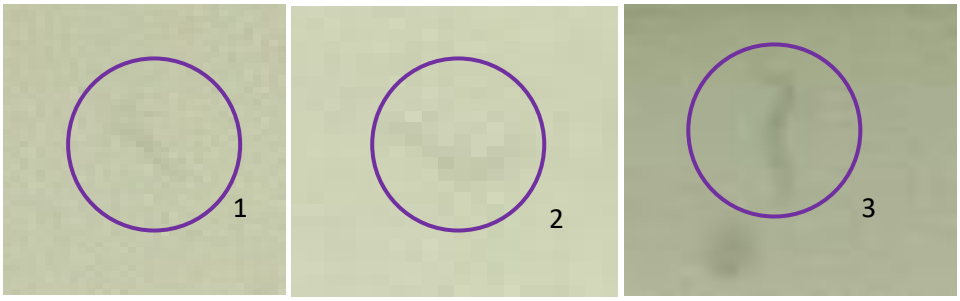

Fig. 43: Ejemplos de predicciones incorrectas de la red, probablemente por la existencia de motas de polvo

Se puede observar en estas imágenes que en las zonas donde la red predice que hay un nematodo, realmente parece ser cierto, por lo que el funcionamiento de la red es correcto y estos errores deben solventarse en el momento de la captura de las imágenes. Por suerte, este error no ocurre con tanta frecuencia, en gran cantidad de las imágenes todas las predicciones son correctas, de ahí los valores de exactitud tan altos.

Error tipo 2

En algunas ocasiones, la red no es capaz de detectar alguno de los gusanos reales. Esto ocurre principalmente por dos motivos: porque existen algunas agregaciones de dos o más gusanos o porque se encuentran en la zona donde se deposita la *Escherichia coli* y, por tanto, el contraste con el fondo es más sutil.

En la Fig. 44 se pueden observar dos ejemplos de agregaciones de *C. elegans* en una misma imagen, donde no se distingue del todo bien dónde empieza y acaba cada uno.

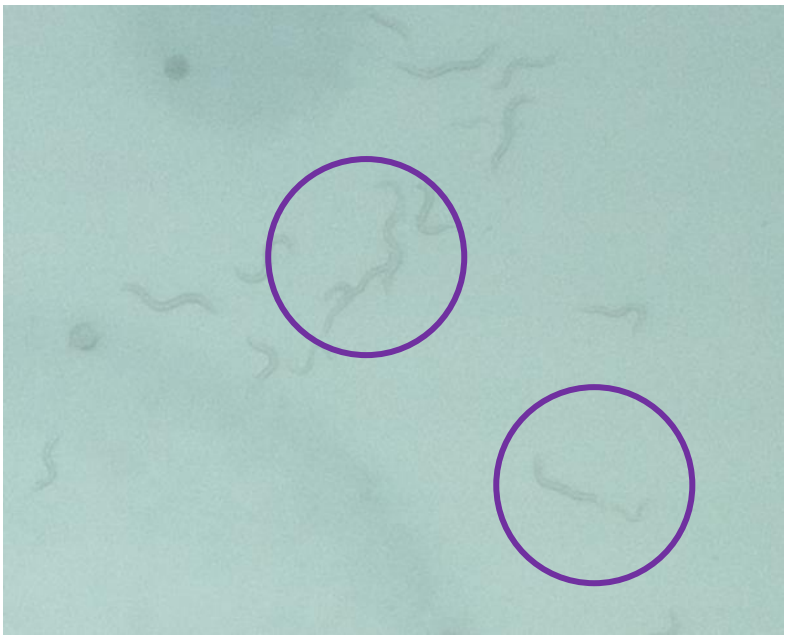

Fig. 44: Ejemplo de dos agregaciones de C. elegans en una misma imagen

En la Fig. 45 se puede comprobar que, efectivamente, en ambas agregaciones hay dos gusanos pegados, sin embargo, en el caso de arriba la red predice que solo hay uno grande y en el otro solo predice uno de los dos.

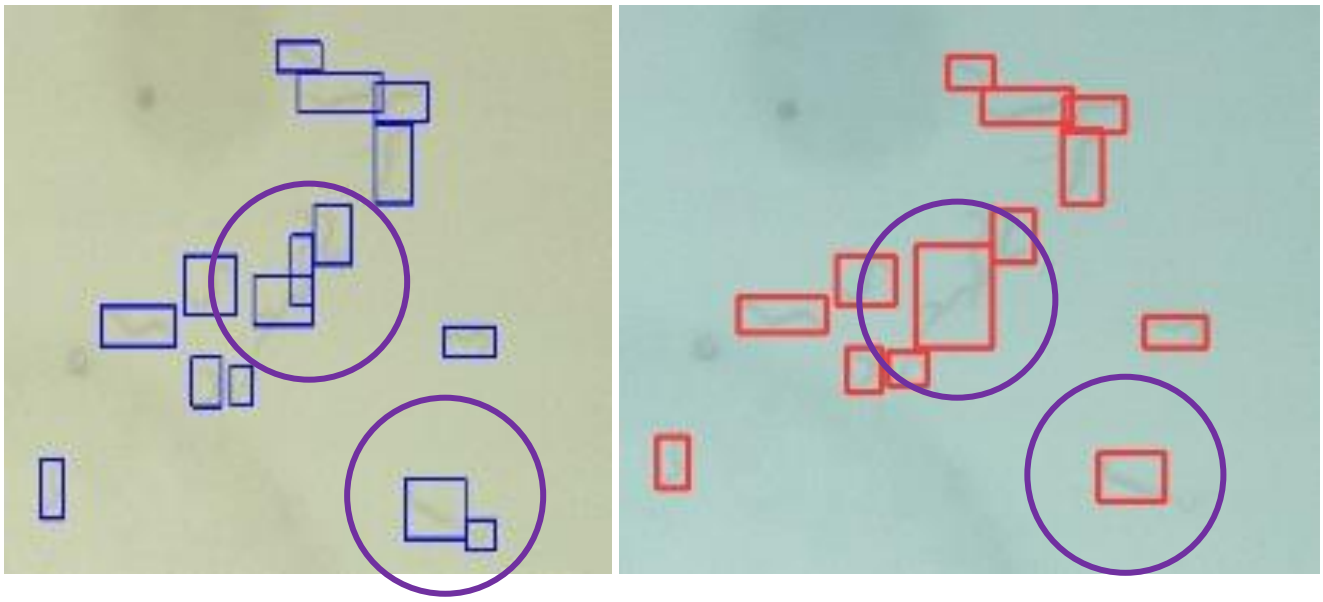

Fig. 45: Cajas reales (izquierda) y predicciones de la red (derecha)

Este problema ocurre con más frecuencia en el dataset DIY, ya que en el dataset SiViS no existen apenas agregaciones. Este problema se podría solventar utilizando datasets con un número mayor de imágenes que presenten agregaciones ya que, en nuestro caso, en los datasets empleados no aparecen muchas de ellas, por lo que la red predice un solo nematodo donde puede haber dos.

Por otro lado, los fallos provocados por la posición de los nematodos en la zona de *Escherichia coli* se producen, principalmente en el dataset DIY, aunque también ocurre en algunos casos en el dataset SiVIS. En este último, la zona de alimento no está únicamente en el centro, también existen algunos puntos en otros lugares menos centrados. En la Fig. 46 se observa que la red ha predicho varios nematodos menos de los que realmente hay.

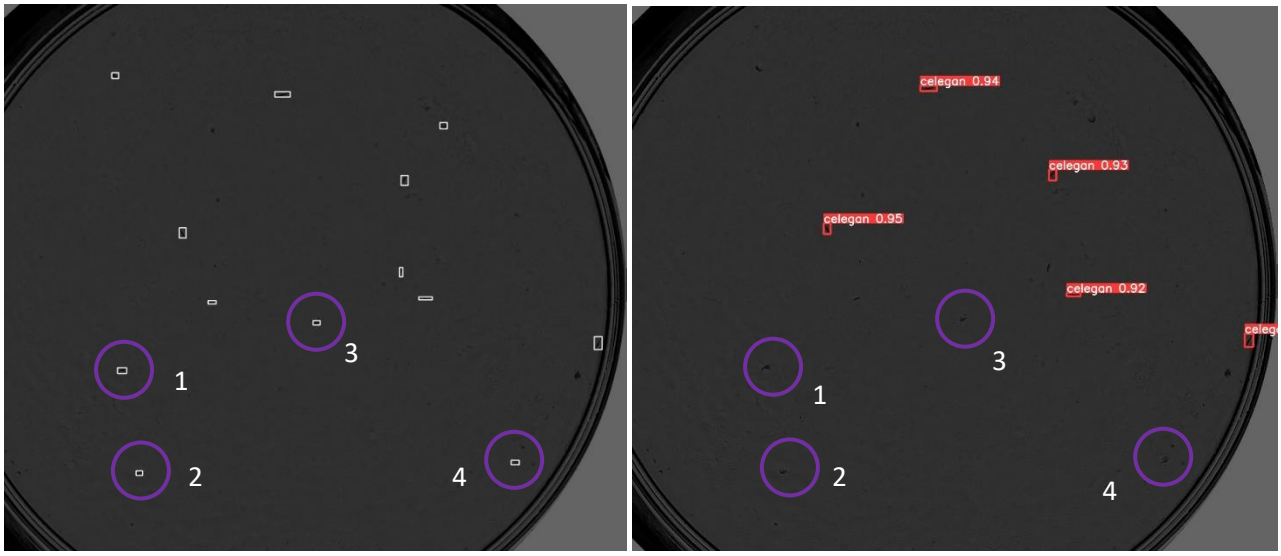

Fig. 46: Cajas correctas (izquierda) y predicciones de la red (derecha) de la misma imagen

A continuación, se muestran ampliados algunos de los errores de la Fig. 46.

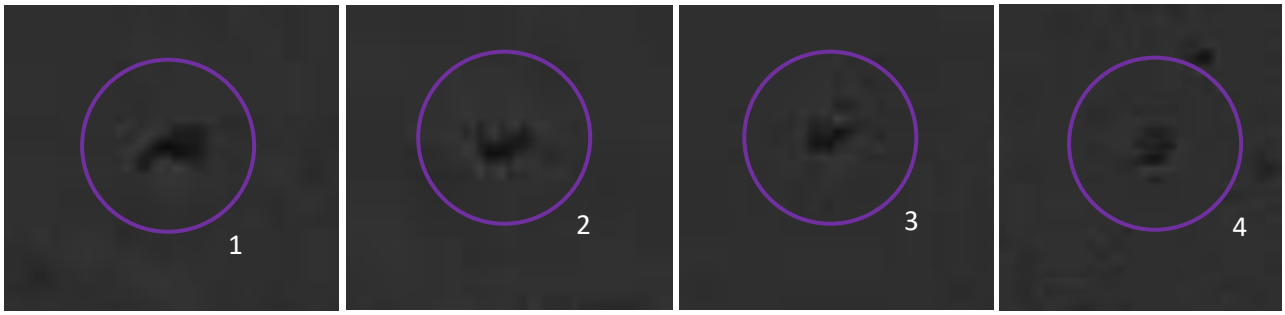

Fig. 47: Ejemplos de nematodos no predichos por la red por estar pegados al alimento

Se puede observar en estas imágenes que junto a cada nematodo existe una mancha que impide su correcta detección.

También existe alguna imagen como la de la Fig. 48 donde existe ruido que impide que la red realice detecciones.

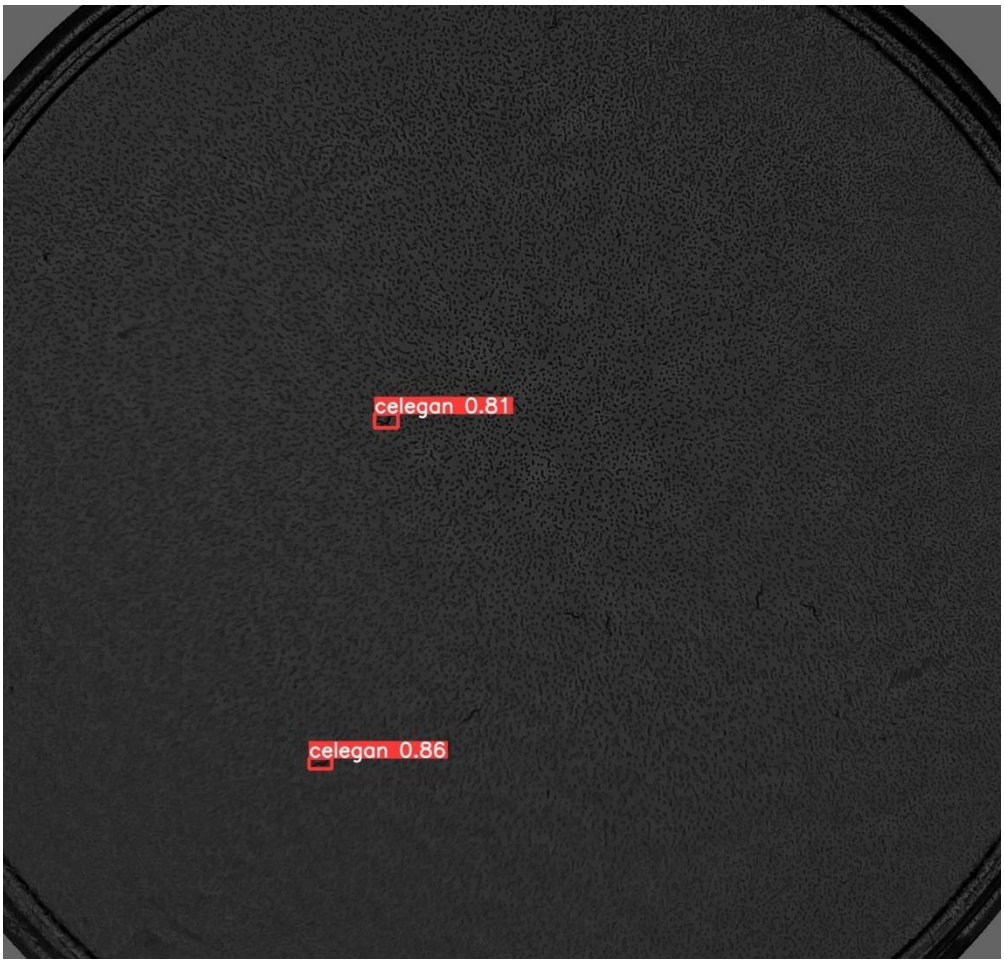

Fig. 48: Imagen del dataset SiViS con ruido

Todos estos fallos se podrían solventar mediante una mejor captura de las imágenes para que fuera más fácil distinguirlos del fondo. Esto no debería ser muy difícil de conseguir en el caso del dataset DIY, puesto que las imágenes se tomaron con un microscopio casero y a baja resolución, por lo que el margen de mejora de la captura es notable. También se podría intentar obtener más imágenes de ambos datasets donde ocurra esto, de forma que la red aprenda a distinguirlos del alimento y del posible ruido.

7. Conclusiones

7.1. Discusión de los resultados

Con el primer experimento se ha intentado comprobar si el entrenamiento de la red con un solo dataset podría ser suficiente para que el modelo fuera capaz de detectar *C. elegans* en imágenes con características diferentes. Se confiaba en que la red fuera capaz de hacerlo mediante la forma característica del nematodo y el contraste con el fondo. Como cabía esperar, las predicciones eran nulas, comprobando así la gran influencia de las condiciones de captura en la detección. El hecho de que la red hubiera entrenado con imágenes con fondo negro y el dataset de test tuviera el fondo claro

puede haber sido un factor determinante. Además, por la iluminación, los nematodos en el dataset DIY aparecen casi transparentes, gris claro, mientras que en el dataset SiViS aparecen en negro. Es por esto que el contraste de los nematodos es diferente en ambos datasets y la red no es capaz de distinguirlos correctamente.

En el segundo experimento la red ya se ha entrenado con dos datasets y por tanto debería ser capaz de detectar *C. elegans* en dos canales distintos. Aún con este modelo mejorado para la generalidad, no es capaz de detectar *C. elegans* en el dataset *developmental*. Este dataset tiene fondo oscuro como el dataset SiViS, por lo que en un primer momento se pensó que podría funcionar correctamente, pero los resultados a la hora de detectar en este dataset han sido nulos. Esto puede deberse a que el fondo es más oscuro que en SiViS y que los *C. elegans* son blancos o gris en lugar de negros, por lo que el contraste con el fondo es diferente. Como ya se ha mencionado, en algunas imágenes el contraste es tan pequeño que es casi imperceptible al ojo humano, mientras en los dos primeros datasets empleados, se puede localizar el nematodo a simple vista. En cuanto tamaño, son bastante similares en cuanto al grosor del nematodo en píxeles.

Finalmente, en el último experimento se han obtenido los resultados de test con los mismos 3 datasets con los que se ha entrenado la red. Esto es debido a la dificultad para encontrar datasets de *C. elegans* publicados, puesto que este campo de estudio está en desarrollo, pero aún no tiene la popularidad suficiente para encontrar grandes cantidades de imágenes o muchos datasets. Es por esto que con este experimento solo se pretende que el modelo sea capaz de detectar *C. elegans* en los 3 canales de imágenes ya mencionados y no con imágenes nuevas. Los resultados obtenidos son bastante satisfactorios y por tanto este sistema ya es capaz de detectar *C. elegans* con bastante precisión en imágenes con tres condiciones diferentes de captura e iluminación.

7.2. Futuros trabajos

En futuros trabajos se podrían realizar diferentes entrenamientos con otros datasets, elegidos de forma estratégica, y los empleados en este trabajo, comprobando cuáles de ellos aportan mejoras en la generalidad.

Otra posible mejora para este propósito sería realizar pruebas con otras arquitecturas de redes neuronales, como las nuevas versiones de YOLO (YOLO V6) o incluso volver a utilizar Faster R-CNN para ver si es capaz de generalizar mejor, aunque se empeoraría el tiempo de detección y el coste computacional.

Con estas medidas, quizá se podría conseguir el objetivo final que es el de obtener un sistema que sea capaz de detectar cualquier *C. elegans*, bajo ciertas limitaciones, lo cual podría ayudar a muchos institutos de investigación de *C. elegans* con menos recursos o con imágenes de peor calidad.

La prueba final que demostraría el cumplimiento de este objetivo consistiría en obtener suficientes imágenes dispares de *C. elegans* de Internet, por ejemplo, crear un dataset con estas imágenes, etiquetarlo, y probar el modelo final con estas imágenes. Si los resultados fueran satisfactorios, se concluiría que el modelo es generalizable para cualquier dataset de imágenes de *C. elegans.*

8. Referencias

[1] D. Sadava y W. H. Purves, *Vida / Life: La ciencia de la biologia / The Science of Biology*. Ed. Médica Panamericana, 2009.

[2] «Caenorhabditis_elegans». https://www.quimica.es/enciclopedia/Caenorhabditis_elegans.html (accedido 25 de agosto de 2022).

[3] «C. elegans, la criatura transparente que ha revelado cosas sorprendentes sobre cómo nuestros cuerpos funcionan... y fallan», *BBC News Mundo*. Accedido: 13 de septiembre de 2022. [En línea]. Disponible en: https://www.bbc.com/mundo/noticias-56889706

[4] P. R. Hunt, «The C. elegans model in toxicity testing», *Journal of Applied Toxicology*, vol. 37, n.º 1, pp. 50-59, 2017, doi: 10.1002/jat.3357.

[5] Z. Mayoral Peña *et al.*, «El nematodo Caenorhabditis elegans como modelo para evaluar el potencial antihelmíntico de extractos de plantas», *Revista mexicana de ciencias pecuarias*, vol. 8, n.º 3, pp. 279-289, sep. 2017, doi: 10.22319/rmcp.v8i3.4504.

[6] J. Hongo, «Japan Researchers Screen for Cancer Using Nematodes and Urine», *Wall Street Journal*, 12 de marzo de 2015. Accedido: 13 de septiembre de 2022. [En línea]. Disponible en: https://www.wsj.com/articles/BL-JRTB-19456

[7] A. García Garví, «Diseño, desarrollo y evaluación de un sistema de clasificación de objetos en imágenes que permita la monitorización de C. elegans mediante redes neuronales convolucionales», p. 67.

[8] «Research», *Ezcurra Lab*, 23 de julio de 2018. https://marinaezcurralab.com/research/ (accedido 8 de septiembre de 2022).

[9] F. Izaurieta y C. Saavedra, «Redes Neuronales Artificiales», ago. 2022.

[10] B. AI, «Redes neuronales», *Medium*, 14 de noviembre de 2019. https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb (accedido 8 de septiembre de 2022).

[11] «Redes Neuronales: una visión superficial - Fernando Sancho Caparrini». http://www.cs.us.es/~fsancho/?e=72 (accedido 8 de septiembre de 2022).

[12] C. A. Ruiz, M. S. Basualdo, y D. J. Matich, «Redes Neuronales: Conceptos Básicos y Aplicaciones.», p. 55.

[13] E. F. Franco y R. J. Ramos, «Aprendizaje de máquina y aprendizaje profundo en biotecnología: aplicaciones, impactos y desafíos», *cac*, vol. 2, n.º 2, pp. 7-26, dic. 2019, doi: 10.22206/cac.2019.v2i2.pp7-26.

[14] D. Calvo, «Clasificación de redes neuronales artificiales», *Diego Calvo*, 13 de julio de 2017. https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/ (accedido 8 de septiembre de 2022).

[15] J. González, «Qué es y qué aplicaciones tiene una red neuronal artificial», *Deyde DataCentric*, 19 de mayo de 2021. https://www.datacentric.es/blog/insight/red-neuronal-artificial-aplicaciones/ (accedido 8 de septiembre de 2022).

[16] S. Fudickar, E. J. Nustede, E. Dreyer, y J. Bornhorst, «Mask R-CNN Based C. Elegans Detection with a DIY Microscope», *Biosensors*, vol. 11, n.º 8, Art. n.º 8, ago. 2021, doi: 10.3390/bios11080257.

[17] K. Bates, K. Le, y H. Lu, «Deep learning for robust and flexible tracking in behavioral studies for C. elegans». bioRxiv, p. 2021.02.08.430359, 10 de febrero de 2021. doi: 10.1101/2021.02.08.430359.

[18] P. Jiang, D. Ergu, F. Liu, Y. Cai, y B. Ma, «A Review of Yolo Algorithm Developments», *Procedia Computer Science*, vol. 199, pp. 1066-1073, ene. 2022, doi: 10.1016/j.procs.2022.01.135.

[19] «COCO - Common Objects in Context». https://cocodataset.org/#home (accedido 13 de septiembre de 2022).

[20] Z. Chen *et al.*, «Plant Disease Recognition Model Based on Improved YOLOv5», *Agronomy*, vol. 12, n.º 2, Art. n.º 2, feb. 2022, doi: 10.3390/agronomy12020365.
[21] W. Souidene Mseddi, M. A. Sedrine, y R. Attia, «YOLOv5 Based Visual Localization For Autonomous Vehicles», en *2021 29th European Signal Processing Conference (EUSIPCO)*, ago. 2021, pp. 746-750. doi: 10.23919/EUSIPCO54536.2021.9616354.
[22] E. J. Rico Guardiola, P. E. Layana Castro, A. García Garví, y A.-J. Sánchez Salmerón, «Caenorhabditis elegans detection using YOLOv5 and Faster R-CNN networks», presentado en OL2A: International Conference on Optimization, Learning Algorithms and Applications 2022, Portugal, Aceptado, pendiente de publicación.
[23] P. Bharati y A. Pramanik, «Deep Learning Techniques—R-CNN to Mask R-CNN: A Survey», en *Computational Intelligence in Pattern Recognition*, Singapore, 2020, pp. 657-668. doi: 10.1007/978-981-13-9042-5_56.
[24] S. Ren, K. He, R. Girshick, y J. Sun, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks», en *Advances in Neural Information Processing Systems*, 2015, vol. 28. Accedido: 25 de agosto de 2022. [En línea]. Disponible en: https://proceedings.neurips.cc/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html
[25] E. Del Hoyo Dorado, «Diseño, implementación y evaluación de una red neuronal convolucional para la detección de puntos principales en naranjas», p. 100.
[26] Q. Fan, L. Brown, y J. Smith, «A closer look at Faster R-CNN for vehicle detection», en *2016 IEEE Intelligent Vehicles Symposium (IV)*, jun. 2016, pp. 124-129. doi: 10.1109/IVS.2016.7535375.
[27] R. KeepCoding, «¿Qué es una imagen en Docker? | KeepCoding Tech School», 28 de abril de 2022. https://keepcoding.io/blog/que-es-una-imagen-en-docker/ (accedido 29 de agosto de 2022).
[28] R. KeepCoding, «Ventajas y Desventajas de Python | KeepCoding Tech School», 22 de septiembre de 2021. https://keepcoding.io/blog/ventajas-y-desventajas-de-python/ (accedido 29 de agosto de 2022).
[29] «Welcome to Python.org», *Python.org*. https://www.python.org/about/ (accedido 29 de agosto de 2022).
[30] «PyTorch». https://www.pytorch.org (accedido 29 de agosto de 2022).
[31] «About», *OpenCV*. https://opencv.org/about/ (accedido 30 de agosto de 2022).
[32] «Pillow». https://pillow.readthedocs.io/en/stable/index.html (accedido 30 de agosto de 2022).
[33] «NumPy - About Us». https://numpy.org/about/ (accedido 30 de agosto de 2022).
[34] T. Stiernagle, *Maintenance of C. elegans*. WormBook, 2006. Accedido: 12 de septiembre de 2022. [En línea]. Disponible en: https://www.ncbi.nlm.nih.gov/books/NBK19649/
[35] J. C. Puchalt, A.-J. Sánchez-Salmerón, E. Ivorra, S. Llopis, R. Martínez, y P. Martorell, «Small flexible automated system for monitoring Caenorhabditis elegans lifespan based on active vision and image processing techniques», *Sci Rep*, vol. 11, n.º 1, Art. n.º 1, jun. 2021, doi: 10.1038/s41598-021-91898-6.
[36] J. C. Puchalt, A.-J. Sánchez-Salmerón, P. M. Guerola, y S. G. Martínez, «Active backlight for automating visual monitoring: An analysis of a lighting control technique for Caenorhabditis elegans cultured on standard Petri plates», *PLOS ONE*, vol. 14, n.º 4, p. e0215548, abr. 2019, doi: 10.1371/journal.pone.0215548.
[37] «Multiview motion tracking based on a cartesian robot to monitor Caenorhabditis elegans in standard Petri dishes | Scientific Reports». https://www.nature.com/articles/s41598-022-05823-6 (accedido 12 de septiembre de 2022).
[38] J. Bornhorst, E. J. Nustede, y S. Fudickar, «Mass Surveilance of C. elegans—Smartphone-Based DIY Microscope and Machine-Learning-Based Approach for Worm Detection», *Sensors*, vol. 19, n.º 6, Art. n.º 6, ene. 2019, doi: 10.3390/s19061468.

[39] V. Bushaev, «Stochastic Gradient Descent with momentum», *Medium*, 5 de diciembre de 2017. https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d (accedido 7 de septiembre de 2022).
[40] «Weight Decay in Neural Networks - Programmathically», 16 de octubre de 2021. https://programmathically.com/weight-decay-in-neural-networks/ (accedido 7 de septiembre de 2022).
[41] J. Brownlee, «Understand the Impact of Learning Rate on Neural Network Performance», *Machine Learning Mastery*, 24 de enero de 2019. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/ (accedido 7 de septiembre de 2022).
[42] A. Rakhecha, «Understanding Learning Rate», *Medium*, 7 de julio de 2019. https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de (accedido 7 de septiembre de 2022).

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Pliego de condiciones

Curso 2021/2022

Índice

1. Objeto

La finalidad de este proyecto es el de desarrollar un sistema genérico de detección de *Caenorhabditis elegans* (*C. elegans*) basado en redes neuronales convolucionales. Para ello, se realizarán tres entrenamientos diferente de la arquitectura de red YOLO V5s combinando tres datasets de imágenes con condiciones de captura e iluminación diferentes.

2. Condiciones de los materiales

2.1. Descripción

2.1.1. Equipos informáticos

Solo se empleará material informático, hardware y software, ningún otro material será necesario para el proyecto.

Toda la programación del proyecto se realizará en el lenguaje de programación Python. Para la programación de funciones que no estén basadas en aprendizaje profundo se empleará cualquier ordenador con una memoria RAM mínima de 2 GB. Para las funciones que realicen operaciones con imágenes será necesario que estén instaladas las librerías PIL, OpenCV y NumPy.

Para la programación del entrenamiento de las redes neuronales, el equipo deberá además tener instalada la herramienta pytorch. Para instalar pytorch, si el sistema operativo del equipo es Linux, deberá ser una versión de glibc no inferior a la v2.17. En el caso de Windows, la versión será no inferior a Windows 7 o Windows Server 2008 r2. El proveedor de pytorch recomienda Windows 10. En el caso de macOS, la versión será no inferior a macOS 10.15 (Catalina). Para el uso de Pytorch, la versión de Python será la 3.7, como mínimo.

En cuanto al hardware, en cualquier sistema operativo será necesario que el sistema cuente con una NVDIA GPU para aprovechar al máximo Pytorch. En caso de emplear un ordenador sin GPU, los tiempos de entrenamiento podrán aumentar excesivamente. El equipo deberá contar con una memoria RAM mínima de 32 GB. La memoria de disco será no menor a 32 GB para contener todos los datasets de imágenes y ficheros de programación

2.1.2. Datasets de imágenes

Las imágenes deberán estar divididas en tres carpetas: entrenamiento, validación y test. Se comprobará que los datasets de imágenes estén etiquetados con alguno de los formatos que acepta YOLO. En caso negativo, se deberá transformar el formato de etiquetas. Se recomienda transformar al formato “pytorch txt”. Para este formato, se generará un fichero de texto, llamado “data.yaml” con las ubicaciones de las carpetas

de imágenes de entrenamiento, validación y test, con el número de clases y con las etiquetas de cada clase.

En los experimentos donde se utilicen varios datasets en el entrenamiento se comprobará que la red entrena con la misma cantidad de imágenes de cada dataset. Esta condición no es necesaria para validación y test, pero sí será necesario que la relación entre cantidad de imágenes de validación y test sea de 1/2.

2.2. Control de calidad

Para la comprobación de los requisitos de hardware, en Windows se pulsará “Windows + R” y en “Ejecutar” se escribirá “dxdiag” y se pulsará sobre “Aceptar”. En Linux, sobre la línea de comandos se escribirá “lshw”. En mac, se pulsará el icono de la manzana en la esquina superior izquierda y, en el menú desplegable, se seleccionará “Acerca de este Mac”. Para comprobar que el equipo posee una NVIDIA GPU y está disponible, en una consola de Python se escribirán los comandos “import torch” y “torch.cuda.is_available()”, a lo que Python deberá responder con “True”.

Para comprobar los requisitos de software, en un terminal se escribirá “pip list”, aparecerá una lista con las librerías instaladas y sus versiones. Deberá aparecer “Pillow” (PIL), “numpy”, “opencv-python” y “torch” (pytorch). Para comprobar la versión de Python, se escribirá en el terminal “python –version”.

3. Condiciones de la ejecución

3.1. Descripción

Para el entrenamiento de la red, se elegirán los valores de los parámetros de entrenamiento: tamaño de lote y número de épocas. El tamaño de lote será no superior a 8 y el número de épocas será no inferior a 50, siendo recomendable unos valores de 6 y 150, respectivamente. También se elegirá el modelo con el que se van a entrenar los pesos de la red, en este caso, “yolov5s”. Se escribirá en cada experimento la ubicación del fichero “data.yaml” para que la red tenga acceso a las imágenes. Se elegirá el tamaño de imagen con el que se va a trabajar. Se recomienda utilizar el tamaño de las imágenes del dataset y, en los casos donde se emplean varios datasets, se elegirá el tamaño medio que permita que los nematodos sean de tamaño en píxeles similar. Por último, se elegirán los valores de los hiperparámetros de entrenamiento y de aumento de datos. Se recomienda emplear los predefinidos por el proveedor del modelo.

Una vez realizadas las comprobaciones y elegidos los parámetros, se escribirá y ejecutará la siguiente línea de código en una consola o cuaderno de Python:

!python train.py --img “tamaño de imagen” --batch 6 --epochs 150 --data {“ubicación del archivo”}/data.yaml --weights yolov5s.pt –cache

Los valores entre comillas se deberán sustituir por los adecuados.

3.2. Control de la ejecución

Cuando la red empiece a entrenar, se mostrarán todos los parámetros e hiperparámetros de entrenamiento, se comprobará que son los elegidos previamente. Además, deberá aparecer "CUDA:0", que indica que la GPU está disponible para funcionar y aparecerá la versión de la misma.

Si tras ejecutar el código que inicia el entrenamiento aparecen mensajes de error, se deberá leer dicho mensaje cuidadosamente para poder solventarlo. En la mayoría de los casos se deberá a un error de código, por lo que se comprobará detenidamente el programa.

Posteriormente, se comprobará que todas las imágenes se cargan en la RAM del equipo. Si durante este proceso, la barra de progreso se detiene se deberá comprobar la cantidad de memoria RAM ejecutándose y cerrar las tareas innecesarias. Si tras esto sigue surgiendo el mismo problema, se reiniciará el equipo informático.

Si todo funciona correctamente, deberá aparecer la ubicación donde se guardarán los resultados y modelo resultante del entrenamiento y deberá aparecer una tabla con los siguientes elementos:

Epoch gpu_mem box obj cls labels img_size

Class Images Labels P R mAP@.5 mAP@

Esta tabla se irá agrandando y actualizando tras cada época. Si los valores de los últimos elementos son nulos durante más de tres épocas, deberá reiniciarse el entrenamiento con previa comprobación de los parámetros y las imágenes seleccionadas, puesto que implicaría que la red no aprende. Cada época debe durar un tiempo inferior a 20 minutos. Si se supera ese tiempo, se deberá comprobar que la GPU está disponible ya que puede haber ocurrido un fallo y estar entrenando en CPU.

Al acabar el entrenamiento, deberá empezar automáticamente el proceso de validación, mostrado mediante un mensaje que expresa "Validating". Tras esto, deberán aparecer los resultados con las métricas de validación y, de nuevo, la ubicación donde se han guardado los resultados y el modelo resultante del entrenamiento. Por último, se accederá a dicha ubicación para comprobar que se ha guardado el modelo y que aparecen gráficas y hojas de cálculos con los resultados. Las métricas de validación y entrenamiento (*precision*, *recall* y mAP 0.5) deberán tener un valor no inferior a 0.9 para considerar el modelo fiable.

4. Prueba de servicio

Para realizar una prueba del modelo, se utilizarán las imágenes de test. En una consola o cuaderno de Python se ejecutará el siguiente código:

!python val2_edited.py --weights “ubicación del modelo best.pt” --img “tamaño de imagen” --data {"ubicación del archivo"}/data.yaml --task test --save-txt --save-conf --save-hybrid --batch-size 1

Esto devolverá, pasados unos minutos, las métricas de evaluación de la prueba, que deberán ser no inferiores a 0.9 para considerar la prueba satisfactoria. También devolverá la ubicación donde se guardan las gráficas de resultados y las etiquetas de las cajas predichas por la red.

Por último, se ejecutará el siguiente código en una consola o cuaderno de Python:

!python detect.py --weights “ubicación del modelo best.pt” --img “tamaño de imagen” --conf “umbral de confianza” --source “ubicación de la carpeta de test”

Esto devolverá la ubicación donde se guardan los resultados de la prueba, esto es, las imágenes con las predicciones de la red en forma de cajas delimitadoras de los *C. elegans.* Se tomarán algunas imágenes al azar para visualizar si el modelo es correcto.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Presupuesto

Curso 2021/2022

Índice

Índice de tablas

Índice de ecuaciones

1. Cuadro de precios elementales

En este trabajo al estar basado principalmente en programación, no se cuenta con materiales o componentes debido a que no se trabaja con objetos tangibles, sino virtuales. Los principales gastos vienen de la mano de obra de ingeniería para programar y desarrollar el proyecto y los equipos necesarios para ello. Tampoco existen gastos derivados de las licencias de los programas empleados puesto que, en su mayoría, los relacionados con Python, son de software libre y para los demás se cuenta con licencias proporcionadas por la UPV. De los equipos empleados, se va a tener en cuenta la amortización de los mismos y el mantenimiento del hardware del *cluster*.

$$Coste(e1) = \frac{Precio \cdot tiempo\ uso}{Vida\ útil} + Mantenimiento = \frac{5000€ \cdot 0.5\ años}{5\ años} + 50 = 600€$$

$$Coste\ (e2) = \frac{Precio \cdot tiempo\ uso}{Vida\ útil} = \frac{900€ \cdot 0.5\ años}{5\ años} = 150€$$

$$Coste\ (e3) = \frac{Precio \cdot tiempo\ uso}{Vida\ útil} = \frac{1200€ \cdot 0.5\ años}{3\ años} = 200€$$

Ecuación 7: Cálculo de la amortización de los equipos informáticos

También se va a considerar el valor del mantenimiento y resolución de problemas del *cluster* del ai2 por parte de un ingeniero informático.

Tabla 10: Cuadro de precios elementales

Ref.	Unidad	Descripción	Precio (€)
		Maquinaria	
e1	ud.	*Cluster* ai2	600,00
e2	día	Ordenador ai2	150,00
e3	ud.	Ordenador personal	200,00
		M.O.D	
h1	h	Ingeniero técnico industrial	20,00
h2	h	Ingeniero informático	10,00

2. Presupuesto

Dado que el proyecto no cuenta con varios módulos, se ha realizado el presupuesto por el método de costes según naturaleza.

Para contar la cantidad de horas de trabajo del ingeniero técnico industrial se ha considerado los créditos ECTS que corresponden al TFG, 12 créditos. Teniendo en cuenta que, según el Espacio Europeo de Educación Superior, 1 crédito ECTS es equivalente a 25 horas de trabajo, el número de horas asciende a 300. El número de horas del ingeniero informático para resolución de errores se ha estimado según experiencias anteriores.

Tabla 11: Presupuesto total

Ref.	Ud.	Descripción	Precio	Cantidad	Total
		Maquinaria			
e1	ud.	*Cluster* ai2	600,00	1	600,00
e2	ud.	Ordenador ai2	150,00	1	150,00
e3	ud.	Ordenador personal	200,00	1	200,00
		M.O.D			
h1	h	Ingeniero técnico industrial	20,00	300	6000,00
h2	h	Ingeniero informático	20,00	20	400
		Medios auxiliares			
%		M.A. sobre costes directos	0,04	7350	294
				TOTAL (Presupuesto de ejecución material)	7644
				Gastos generales 6%	458,64
				Beneficio industrial 13%	993,72
				TOTAL (Presupuesto de ejecución por contrata)	9096,36
				IVA 21%	1910,24
				PRESUPUESTO TOTAL	11006,60

El presupuesto total del proyecto asciende a **ONCE MIL SEIS EUROS Y SESENTA CÉNTIMOS.**