UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado

Ingeniería Electrónica Industrial y Automática

Diseño, desarrollo y evaluación de un sistema de clasificación de objetos en imágenes que permita la monitorización de *C. elegans* mediante redes neuronales convolucionales

Antonio García Garví

Tutor: Eugenio Ivorra Martínez

Cotutor: Antonio José Sánchez Salmerón

Julio 2019

Agradecimientos

A mis padres y a mi hermana, por apoyarme y ayudarme en todo momento. Gracias a su esfuerzo ha sido mucho más fácil alcanzar todos mis objetivos.

A los compañeros del instituto ai2, y en especial a Antonio José Sánchez, por ayudarme y guiarme durante la realización de este proyecto y confiar siempre en mí.

A los compañeros que han compartido conmigo estos 4 años y me han hecho disfrutar de la universidad.

Resumen

En este proyecto se ha desarrollado un algoritmo que facilita las tareas de monitorización de nematodos *Caenorhabditis elegans* (*C. elegans)*, con el objetivo de automatizar las tareas de investigación. En concreto, se trata de un clasificador de imágenes que permite detectar si el gusano está vivo o muerto.

En primer lugar, se ha realizado un estudio de alternativas revisando proyectos similares y comparando las técnicas tradicionales de visión por computador y los algoritmos de Aprendizaje Profundo (*Deep Learning)*.

Tras analizar sus ventajas y desventajas, se ha optado por hacer uso de redes neuronales artificiales. Para poder entrenar redes neuronales, es necesario disponer de una gran cantidad de imágenes etiquetadas, por lo que se ha procedido a etiquetar la base de datos disponible.

Seguidamente, se ha planteado el uso de dos arquitecturas de redes neuronales: las redes neuronales convolucionales y las redes neuronales recurrentes. Para cada tipo de red se necesita un tipo de entrada, por lo que se han propuesto varios métodos para introducir la información.

Una vez generado el *dataset* completo, se ha divido en grupos de entrenamiento y validación. A continuación, se han implementado las redes neuronales haciendo uso del software de *Deep Learning* Pytorch y se han desarrollado diferentes pruebas para determinar la arquitectura más apropiada para resolver el problema y optimizar los resultados.

Como conclusión, se ha comprobado que la combinación de redes neuronales convolucionales y recurrentes obtiene los mejores porcentajes de precisión al clasificar las imágenes de *C. elegans*.

Palabras clave: automatización, visión artificial, monitorización*, Deep Learning*, redes neuronales artificiales

Resum

En aquest projecte s'ha desenvolupat un algorisme que facilita les tasques de monitoratge de nematodes *Caenorhabditis elegans* (*C. elegans)*, amb l'objectiu d'automatitzar les tasques d'investigació. En concret, es tracta d'un classificador d'imatges que permet detectar si el cuc està viu o mort.

En primer lloc, s'ha realitzat un estudi d'alternatives revisant projectes similars i comparant les tècniques tradicionals de visió per computador i els algorismes de *Deep Learning*.

Després d'analitzar els seus avantatges i desavantatges, s'ha optat per fer ús de xarxes neuronals artificials. Per a poder entrenar xarxes neuronals, és necessari disposar d'una gran quantitat d'imatges etiquetades, per la qual cosa s'ha procedit a etiquetar la base de dades disponible.

Seguidament, s'ha plantejat l'ús de dues arquitectures de xarxes neuronals: les xarxes neuronals convolucionals i les xarxes neuronals recurrents. Per a cada tipus de xarxa es necessita un tipus d'entrada, per la qual cosa s'han proposat diversos mètodes per a introduir la informació.

Una vegada generat el dataset complet, s'ha dividit en grups d'entrenament i validació. A continuació, s'han implementat les xarxes neuronals fent ús del programari de *Deep Learning* Pytorch i s'han desenvolupat diferents proves per a determinar l'arquitectura més apropiada per a resoldre el problema i optimitzar els resultats.

Com a conclusió, s'ha comprovat que la combinació de xarxes neuronals convolucionals i recurrents obté els millors percentatges de precisió en classificar les imatges de *C. elegans*.

Paraules clau: automatització, visió artificial, monitorització, *Deep Learning*, xarxes neuronals artificials

Abstract

In this project an algorithm has been developed that facilitates the monitoring tasks of nematodes *Caenorhabditis elegans* (*C. elegans)*, with the aim of automating research tasks. Specifically, it is an image classifier that allows to detect if the worm is alive or dead.

First of all, a study of alternatives has been carried out by reviewing similar projects and comparing traditional computer vision techniques and Deep Learning algorithms.

After analyzing their advantages and disadvantages, it has been decided to make use of artificial neural networks. In order to train neural networks, it is necessary to have a large number of tagged images, so we have proceeded to tag the database available.

Next, the use of two neural network architectures has been considered: convolutional neural networks and recurrent neural networks. For each type of network an input type is needed, so several methods have been proposed to introduce the information.

Once the complete dataset has been generated, it has been divided into training and validation groups. Next, neural networks have been implemented using the *Deep Learning* Pytorch software and different tests have been developed to determine the most appropriate architecture to solve the problem and optimize the results.

As a conclusion, it has been proved that the combination of convolutional and recurrent neural networks obtains the best percentages of precision when classifying the images of *C. elegans*.

Keywords: automation, computer vision, monitoring, Deep Learning, artificial neural networks

Contenido

Índice de figuras

Índice de tablas

Índice de ecuaciones

1. Objeto

La finalidad de este proyecto es el diseño de una aplicación informática que ayude a automatizar la monitorización de nematodos *Caenorhabditis elegans* (*C. elegans*). En concreto, el sistema debe ser capaz de detectar si están vivos o muertos a partir de las imágenes capturadas por un sistema de visión artificial.

2. Descripción y antecedentes del problema

2.1 *Lifespan y Healthspan*

En las últimas décadas, los grandes avances en la medicina y la tecnología han dado lugar a un aumento de la esperanza de vida (*Lifespan*) de la población. La Organización Mundial de la Salud estima que el número de personas que vivirán más de 85 años aumentará en un 351% en los próximos 40 años.
Sin embargo, este aumento de la esperanza de vida no se corresponde con un aumento en la calidad de vida, ya que con el envejecimiento aparecen enfermedades crónicas como el cáncer, la diabetes y la artritis[1].

Este problema a nivel global hace necesario la investigación en búsqueda de fármacos, tratamientos y alimentos que permitan paliar los efectos del envejecimiento, dando lugar a una vida más larga, pero también de mayor calidad (*Healthspan*). Muchos de los avances recientes que se han llevado a cabo en la investigación del envejecimiento, han sido gracias a los experimentos con *C. elegans.*

2.2 *C. elegans*

Son nematodos de aproximadamente de 1mm de longitud, que han sido empleados en laboratorios desde 1960 para el estudio y búsqueda de tratamientos de distintas enfermedades. Se conoce la secuencia completa de su genoma desde 1998[2], siendo el primer organismo multicelular eucariota del que se obtuvo.

Se pueden cultivar en placas de Petri de una forma sencilla y económica. Esta ventaja, unido a que son transparentes, facilita la visualización mediante microscopio de diferentes procesos biológicos. Tienen una vida corta (de 2 a 3 semanas), lo cual permite realizar experimentos y obtener resultados en poco tiempo.

El ciclo de vida de estos nematodos se ve afectado por las condiciones del medio, la alimentación y mutaciones genéticas. Por ello, se emplean en experimentos para averiguar cómo afectan estos factores a la esperanza de vida. Además, en edad adulta muestran un deterioro de sus tejidos y pérdidas de movilidad que se pueden asemejar a las de los humanos, convirtiéndolos en un organismo perfecto para experimentos de *Lifespan* y *Healthspan* [3].

Fig. 1 Imagen de C. elegans (izquierda) y de placa de Petri (derecha)

2.3 Antecedentes de la visión artificial

El aumento de la competitividad de nuestra industria es un requerimiento primordial en la situación económica actual. Un factor clave para mejorar la competitividad de la industria es aumentar la productividad incorporando sistemas automáticos en los diferentes procesos productivos. Los sistemas de inspección visual automática, basados en visión por computador, han demostrado ser una herramienta fundamental para mejorar la calidad de los procesos y productos. Estos sistemas de visión permiten la inspección continua en las líneas de producción, evitando fatigas y distracciones, y facilitando la cuantificación de las variables de calidad en prácticamente el 100% de la producción. Esto se traduce, no sólo en una mejora de la calidad final de los productos, sino también en un ahorro en términos económicos y medioambientales.

Durante las últimas décadas se han podido resolver multitud de aplicaciones mediante la implantación de sistemas de inspección automática en la industria. El principal problema a resolver en estos sistemas de visión ha sido la variabilidad de las imágenes que puede dificultar la segmentación de los objetos de interés. Para resolver este problema se han propuesto algunas técnicas robustas de segmentación que intentan paliar dicha variabilidad [4] [5].

Por otro lado, los enormes avances tecnológicos producidos en las cámaras lineales, los sensores hiperespectrales [6] [7] , los escáneres 3D [8] [9] [10] [11] y en los sistemas de procesamiento basados en FPGAs o GPUs (que son capaces de ejecutar redes neuronales convolucionales en tiempo real); están permitiendo resolver algunas aplicaciones complejas en el sector industrial que hace unos años eran impensables.

Además, los últimos avances producidos en la aplicación de técnicas de seguimiento de cuerpos articulados o flexibles [12] [13] [14] [15] [16] [17] están facilitando la detección de eventos sobre secuencias de imágenes y el reconocimiento de acciones [18] [19] .

Todas estas nuevas tecnologías y herramientas aplicadas al sector de la biotecnología se están empezando a introducir en la industria.

2.4 Necesidad de automatizar tareas de investigación

Actualmente, en la mayoría de los laboratorios se desarrollan experimentos de forma manual, empleando lupas para la inspección visual de las poblaciones de nematodos cultivados en placas de Petri. Este procedimiento requiere mucho tiempo por parte del investigador, y está sujeto a posibles errores humanos. Además, no se pueden realizar medidas precisas de parámetros como pueden ser el desplazamiento, cambio de postura, reacción ante estímulos.

Fig. 2 Inspección visual de una cepa de C. elegans en una placa de Petri. Fuente [20]

Por tanto, disponer de herramientas que consigan automatizar las tareas de investigación, permite realizar más experimentos en menor tiempo, obteniendo resultados rápidos y más precisos. Esto favorece que se produzcan avances en la búsqueda de tratamientos para mejorar la salud de las personas.

La inspección de *C. elegans* cultivados en placas de Petri estándares es un problema complejo debido sobre todo a: (1) la variabilidad de las formas que pueden adoptar estos nematodos; (2) el problema de contaminación y condensación, que requiere de técnicas de iluminación especiales y (3) el problema de agregación de varios *C. elegans*, que requiere técnicas de segmentación específicas. Para resolver estos problemas se propusieron algunas técnicas robustas de seguimiento, como por ejemplo [21] [22] y de iluminación inteligente [23].

En el Instituto Universitario de Automática e Informática Industrial (ai2) de la Universidad Politécnica de Valencia (UPV) se están desarrollando este tipo de herramientas. Actualmente, se dispone de un sistema de visión artificial que captura 30 imágenes al día de cada placa de ensayo. Las imágenes son almacenadas y posteriormente procesadas por un algoritmo de detección automática de *C. elegans*. De esta forma, se puede realizar un conteo del número de gusanos vivos presentes en la placa en cada día y elaborar curvas de supervivencia de los gusanos ante diferentes condiciones. Sin embargo, este algoritmo presenta errores y no es lo suficientemente robusto a la hora de detectar cuando el *C. elegans* está muerto.

Para dotar al sistema de mayor robustez se necesita desarrollar una aplicación que permita clasificar las imágenes con una gran precisión. De esta necesidad surge el

proyecto que se desarrolla en este trabajo de fin de grado. En concreto, se pretende diseñar un clasificador que permita al investigador saber si el *C. elegans* identificado en un día concreto está vivo o muerto.

Fig. 3 Descripción del problema

3.Estudio de necesidades

En este apartado se analizan los requerimientos técnicos del proyecto: especificaciones de rendimiento, hardware y software disponible, así como la base de datos de partida. Se realiza un pequeño estudio del estado del arte de proyectos similares y por último se plantea la solución que se va a desarrollar.

3.1 Especificaciones del encargo

El proyecto posee una serie de condicionantes que deben ser tenidos en cuenta a la hora de resolver el problema. A continuación, se describen los más importantes.

3.1.1 Especificaciones de rendimiento

El programa desarrollado debe lograr una precisión superior al 95% en la tarea de clasificación de *C. elegans* vivos o muertos.

3.1.2 Hardware

El proyecto se debe desarrollar e implementar en un ordenador del laboratorio del instituto ai2. Dicho equipo cuenta con las siguientes características:

- Procesador @Intel® Core™ i7-7700K CPU @4,20GHz x 8.
- Tiene 15,6 GiB de memoria RAM
- Unidad de Procesamiento Gráfico (GPU) GeForce GTX 1070 Ti/PCIe/SSE2

Fig. 4 Características del equipo disponible

3.1.3 Software

El equipo tiene instalado el sistema operativo libre Ubuntu 16.04 LTS. Por tanto, el proyecto debe desarrollarse en este entorno. El programa se debe desarrollar en lenguaje Python [24], que es de código abierto. Destaca por ser un lenguaje interpretado multiplataforma. Además, es multiparadigma, soportando orientación a objetos, programación imperativa y programación funcional.

3.1.4 Imágenes

Se parte de una base de datos en la que existen diferentes experimentos de *Lifespan* realizados con *C. elegans* obtenida a partir del sistema de visión artificial del grupo de investigación del instituto ai2[23]. Un experimento de *Lifespan* consiste en capturar cada día una secuencia de imágenes durante 30 segundos. Las imágenes capturadas se procesan posteriormente contando automáticamente la cantidad de *C. elegans* vivos o muertos. Cada día se dispone de una secuencia de 30 imágenes de cada placa del ensayo con una resolución de 1944x1944 píxeles. Las carpetas están organizadas tal y como se recoge en la figura:

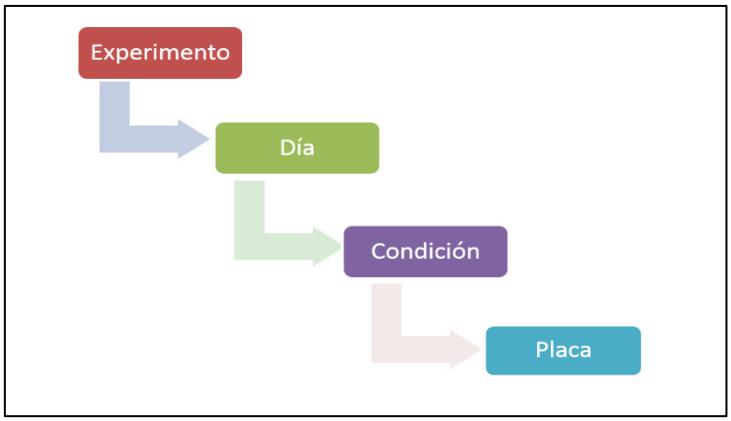

Fig. 5 Organización de la base de datos

Las imágenes se obtienen a partir de una técnica de luz trasera (*backlight)* tal y como se observa en la *Fig. 6,* de forma que los *C. elegans* aparecen de color negro y la placa de Petri de color gris. Posteriormente, se elimina todo lo que queda fuera de la circunferencia de la placa de Petri para facilitar la segmentación, obteniéndose una imagen como la de la *Fig. 7*.

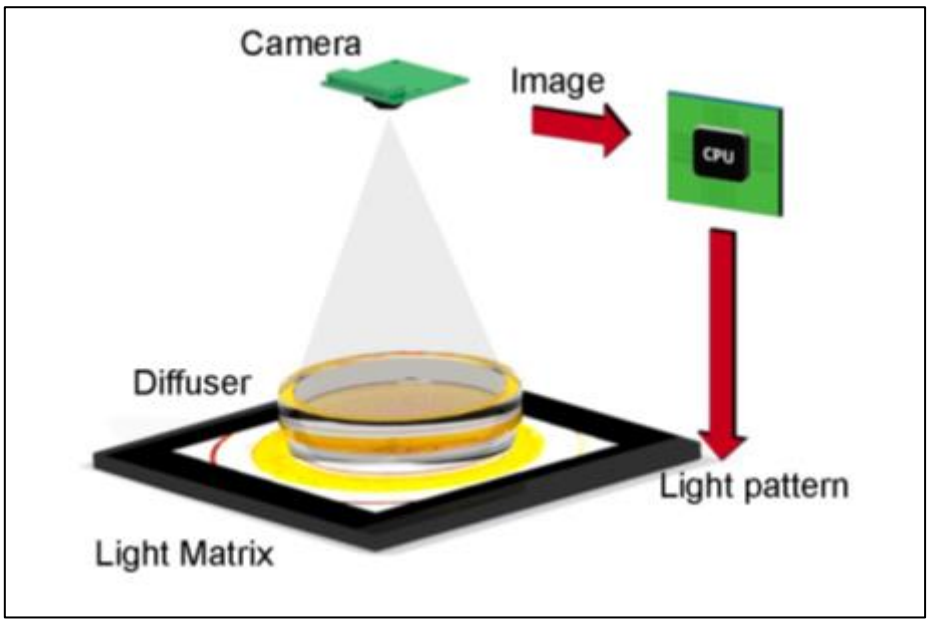

Fig. 6 Sistema de visión con control inteligente de iluminación del instituto ai2 [23]

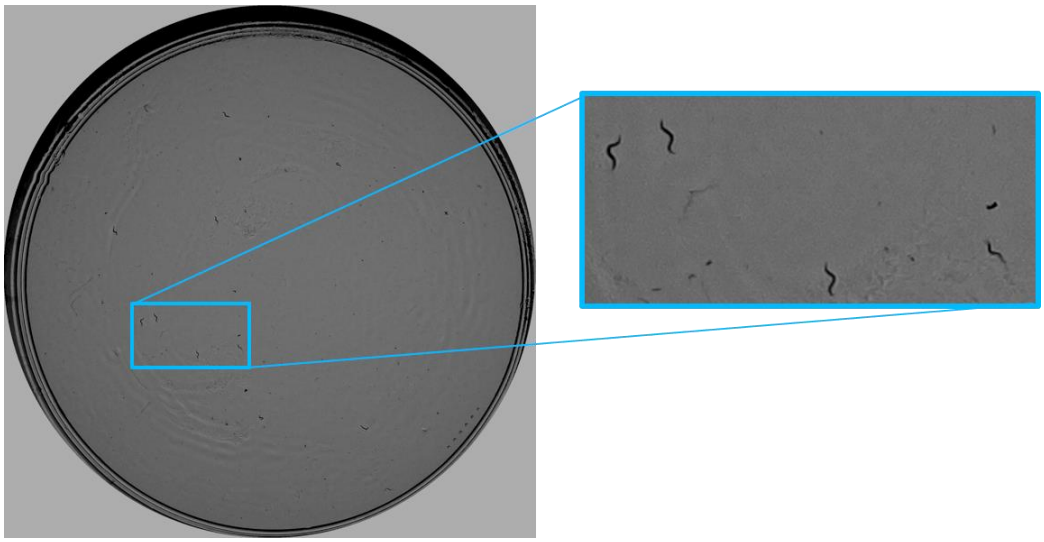

Fig. 7 Ejemplo de imagen de partida

3.2 Proyectos similares

Antes de diseñar la solución al problema, se ha investigado la existencia de proyectos que hayan resuelto problemas similares. Algunas de ellas son las siguientes:

- WorMachine. Es un software desarrollado en Matlab que permite analizar imágenes capturadas en experimentos con *C. elegans*. Para ello, realiza un

proceso secuencial dividido en tres etapas. En la primera, realiza un procesamiento de la imagen para segmentar los *C. elegans* del resto de elementos que aparecen. En la segunda, se obtienen características morfológicas (área, longitud, etc) y se identifican gusanos erróneos de la primera etapa. Por último, se emplean técnicas de *machine learning* para clasificación de características binarias y obtención de parámetros relacionados con distintos fenotipos[25].

- The *C. elegans* Lifespan Machine. Este sistema permite obtener curvas de supervivencia de *C. elegans* en diferentes experimentos. El sistema captura una secuencia de imágenes de la placa y las procesa. En primer lugar, realiza una segmentación del fondo de la imagen. A continuación, discrimina entre los gusanos y otros elementos utilizando clasificadores *Support Vector Machine* (SVM). Posteriormente, se identifican los gusanos estacionarios y se analiza la secuencia para determinar si están muertos [26].
- Automated WormScan. Este sistema emplea escáneres comerciales de fotos para capturar imágenes de experimentos con microorganismos. Se basa en la diferencia entre imágenes consecutivas de la secuencia capturada para identificar los objetos en movimiento. Destaca por su simplicidad y escaso coste computacional[27].

3.3 Alternativas y solución adoptada

Tras realizar el estudio de proyectos similares, se ha procedido al planteamiento de alternativas para resolver el problema. Se trata de un problema de clasificación de imágenes y existe la posibilidad de abordarlo empleando técnicas tradicionales de visión por computador, técnicas de *Deep Learning* o una combinación de ambos métodos. Para obtener información sobre las ventajas de cada una de estas técnicas se ha consultado el siguiente artículo[28].

3.3.1 Técnicas tradicionales de visión por computador

Existen técnicas muy empleadas en el ámbito de la visión por computador para la extracción de características como *Scale-invariant feature transform* (SIFT), *Speeded-Up Robust Features* (SURF), *Binary Robust Independent Elementary Features* (BRIEF) que permiten resolver problemas de detección de objetos y clasificación de imágenes.

Estas técnicas presentan la dificultad de tener que elegir las características que mejor describen los objetos a identificar. Todas estas características llevan asociados una gran cantidad de parámetros, que deben ser ajustados por el ingeniero de forma manual para poder definirlas.

3.3.2 *Deep Learning*

En los últimos años, los grandes avances en el campo de las redes neuronales artificiales, junto con el aumento de las prestaciones de los ordenadores (memoria, capacidad de procesamiento, consumo de energía) han permitido emplear técnicas de *Deep Learning* en el ámbito de la visión por computador.

Empleando estas técnicas no es necesario tener un gran conocimiento de la extracción de características. Simplemente es necesario disponer de un conjunto de imágenes etiquetadas con las distintas clases que queremos clasificar. Con estas imágenes se entrena a la red, y esta es capaz de aprender las características más representativas de los objetos a clasificar. Los algoritmos de *Deep Learning* obtienen en general resultados con una mayor precisión que las técnicas tradicionales. Sin embargo, requieren de una gran cantidad de imágenes, lo cual supone un alto coste de etiquetado.

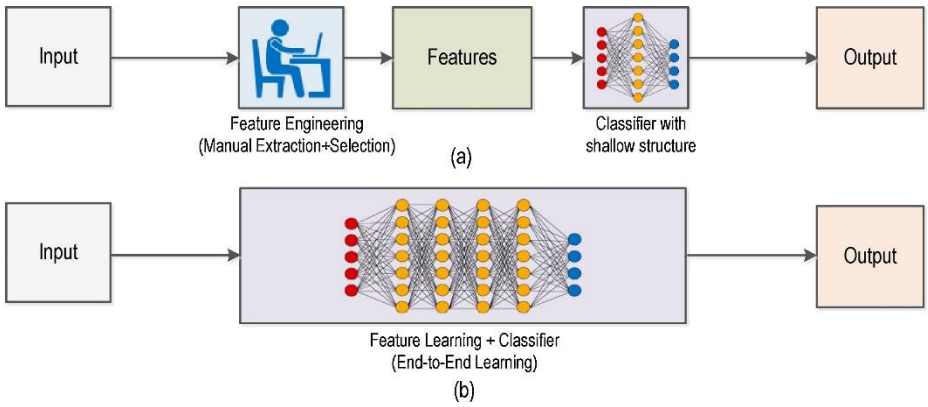

Fig. 8 Técnicas tradicionales(a) vs Deep Learning (b) [28]

3.3.3 Justificación de la solución adoptada

Teniendo en cuenta las ventajas y desventajas de las técnicas disponibles, se ha optado por emplear redes neuronales artificiales teniendo en cuenta lo siguiente:

- Se busca obtener una alta precisión, por lo que a priori las redes neuronales son una opción mejor que las técnicas tradicionales.
- Se pretende realizar el mínimo procesamiento de imagen posible. Empleando redes neuronales se evita realizar la etapa de extracción de características. De esta manera se ahorra tiempo y se necesitan menos conocimientos de las características.
- Se dispone de una gran cantidad de imágenes de resolución 1944x1944 píxeles. Esto es fundamental a la hora de entrenar redes neuronales, por tanto, se pueden aprovechar para crear un *dataset*.
- Se cuenta con un ordenador con una GPU GeForce GTX 1070 Ti/PCIe/SSE2 que permite entrenar redes en poco tiempo.

3.4 Plan de desarrollo del proyecto

Una vez decidido el uso de redes neuronales para la resolución del problema, se han establecido los siguientes pasos para llevar a cabo el proyecto:

- Investigar sobre redes neuronales, aprender el lenguaje de programación Python, uso de librerías PIL, OpenCV, Pytorch.
- Realizar el etiquetado de las imágenes, que serán divididas en dos grupos, las utilizadas para el entrenamiento de la red neuronal y las utilizadas para la evaluación del algoritmo.

- Escoger la arquitectura de la red neuronal a implementar.
- Implementar la red neuronal utilizando el lenguaje de programación Python y la plataforma software de *Deep Learning* Pytorch.
- Realizar el entrenamiento de la red utilizando las imágenes etiquetadas.
- Evaluar el sistema utilizando un conjunto de imágenes nuevas para el algoritmo y valoración del sistema en función de la tasa de acierto.
- Optimizar el sistema.

A continuación, se muestra el cronograma con la planificación de las distintas etapas del proyecto:

Cronograma Proyecto			
Tarea	**Fecha de comienzo**	**Fecha de finalización**	**Tiempo (días)**
Planteamiento de alternativas y elección de la más adecuada	02/11/2018	09/11/2018	7
Formación e investigación	10/11/2018	01/05/2019	172
Etiquetado	10/11/2018	01/05/2019	172
Diseño, programación e implementación de la solución	02/05/2019	02/06/2019	31
Testeo y optimización	02/06/2019	28/06/2019	26
Redacción de la memoria	01/06/2019	28/06/2019	27
Revisión del documento	26/06/2019	03/07/2019	7

Fig. 9 Cronograma del proyecto

4. Descripción detallada de la solución

4.1 Introducción teórica a las redes neuronales artificiales

Son modelos computacionales que reciben este nombre por su inspiración en las redes neuronales biológicas. Su unidad básica de procesamiento es la neurona. Al igual que las neuronas biológicas, presentan unas conexiones de entrada a través de las cuales reciben unos valores (inputs). En la siguiente figura se muestra el esquema básico de una neurona:

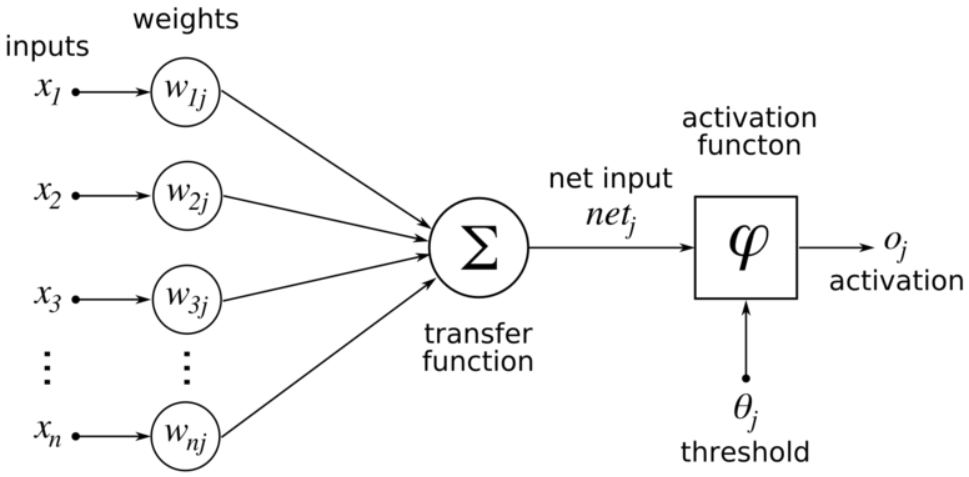

Fig. 10 Esquema de una neurona[29]

Estos valores de entrada se ponderan en función de los pesos asignados a cada uno de ellos. Es decir, cada conexión de entrada tiene asociado un valor que define en qué medida afecta cada entrada a la salida. Estos pesos se inicializan de forma aleatoria.

A partir de estos valores, la neurona realiza una suma ponderada para obtener una salida. Para poder modelar problemas no lineales se emplea lo que se conoce como funciones de activación. Algunos ejemplos son la función sigmoide, la función escalonada o la función ReLU. Estas funciones introducen no linealidades en el modelo que permiten encadenar neuronas para resolver problemas no lineales.

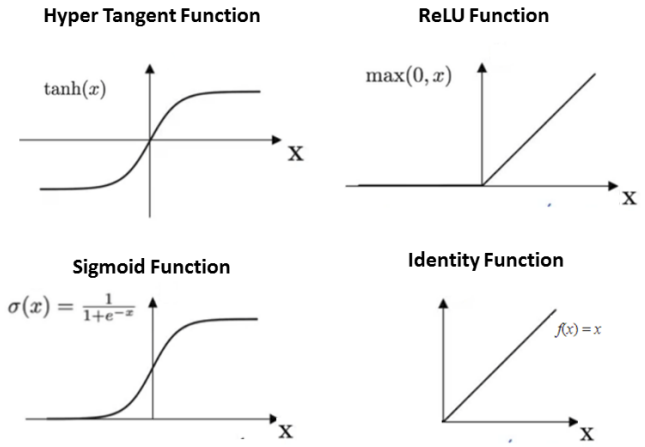

Fig. 11 Funciones de activación[30]

Las neuronas se organizan en capas, que según la posición que ocupen se denominan capa de entrada, capa oculta o capa de salida. Las capas de entrada son las encargadas de recibir los elementos que se quieren procesar o clasificar. En las capas ocultas se trata de representar las entradas de la red de forma que se facilite llegar a la predicción esperada para dicha entrada[31]. Estas capas se conectan entre sí para formar la red neuronal.

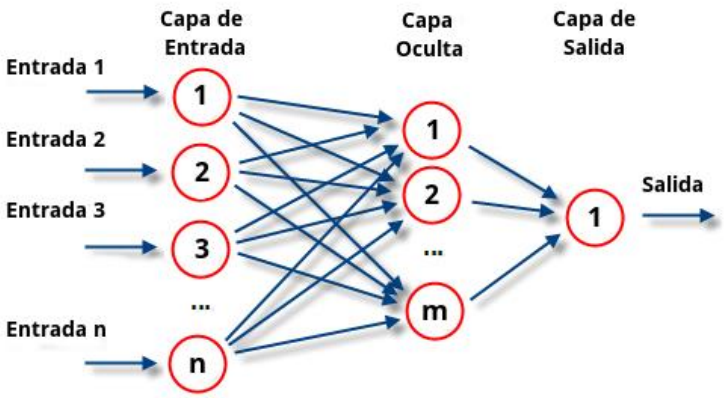

Fig. 12 Esquema de una red neuronal[29]

A medida que aumenta el número de capas se pueden resolver problemas más complejos, obteniendo un conocimiento más abstracto conforme se avanza en las capas. Esto es lo que se conoce como aprendizaje profundo (*Deep Learning*).

El proceso de entrenamiento de la red neuronal consta de los siguientes pasos:

1. Se introducen los datos de entrada y la salida esperada (etiqueta) de un problema a resolver para que la red ajuste los pesos de forma óptima para obtener dicho resultado.
2. Se calcula el error o coste. Es decir, la diferencia entre el resultado esperado y la predicción obtenida.
3. Aplicar el algoritmo de *backpropagation*. Este método permite calcular las derivadas parciales de los parámetros de la red con respecto al coste, es decir, obtener la influencia de cada parámetro en el error.
4. Ajuste de los pesos. Se actualizan los valores en función de los resultados del paso anterior haciendo uso del algoritmo del descenso del gradiente.

En la resolución de problemas de clasificación de imágenes destaca un tipo de redes por encima del resto: las redes neuronales convolucionales.

Redes neuronales convolucionales

Este tipo de redes están inspiradas en la forma en que la funciona la corteza visual humana a la hora de identificar objetos. Poseen varias capas ocultas que presentan una jerarquía. Las primeras capas extraen las características primitivas (bordes, orientaciones, pequeños patrones) y según se avanza en la red se aprenden conceptos más complejos. Finalmente, todas estas características se unen para formar el objeto.

Su arquitectura básica consiste en las siguientes capas:

- **Capa de convolución**. La operación de convolución aplica diferentes filtros a la imagen para obtener diferentes características (bordes, brillo, orientaciones). Estos filtros se inicializan de forma aleatoria y se van actualizando hasta obtener el valor óptimo. A la salida de esta capa se obtienen tantas imágenes como filtros se han aplicado (mapa de características).

- **Función de activación ReLU**. Recorre los píxeles y convierte los valores negativos en cero. Esto permite un entrenamiento más rápido, haciendo que solo pasen a la siguiente capa las características que han sido activadas.
- **Función Max-Pooling.** Reduce el tamaño de las imágenes, permitiendo disminuir el número de neuronas del modelo.
- **Capa Fully Connected.** Convierte las imágenes de la capa anterior en un vector.
- **Función de activación Softmax**. Genera las probabilidades de la salida, es decir, la probabilidad de que el objeto pertenezca a cada una de las clases posibles.

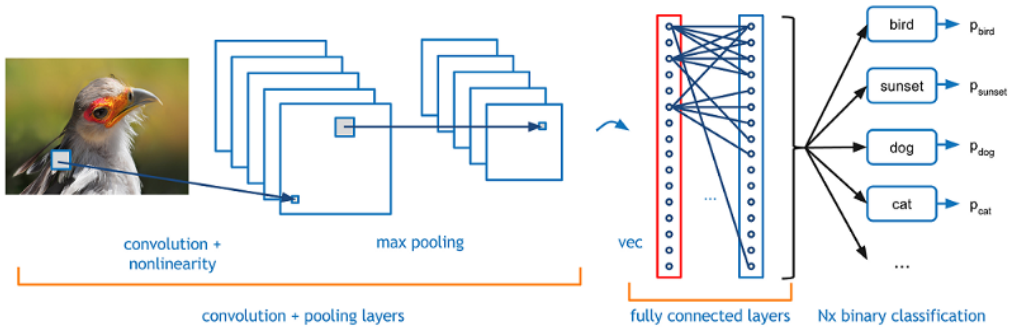

Fig. 13 Ejemplo arquitectura CNN [32]

Redes neuronales recurrentes

Este tipo de redes neuronales se emplea cuando se trabaja con datos secuenciales, como por ejemplo una secuencia de video, de voz o de texto. Funcionan de forma que la entrada de estas redes sea la entrada en el instante actual (X_t) y el estado del instante de tiempo anterior(h_{t-1}). Sin embargo, este tipo de redes presentan algunos inconvenientes:

- Mantener el estado en memoria presenta un alto coste computacional
- Son sensibles a los cambios en sus parámetros

Estos problemas hacen que estas redes no funcionen de forma correcta para las dependencias temporales largas[33].

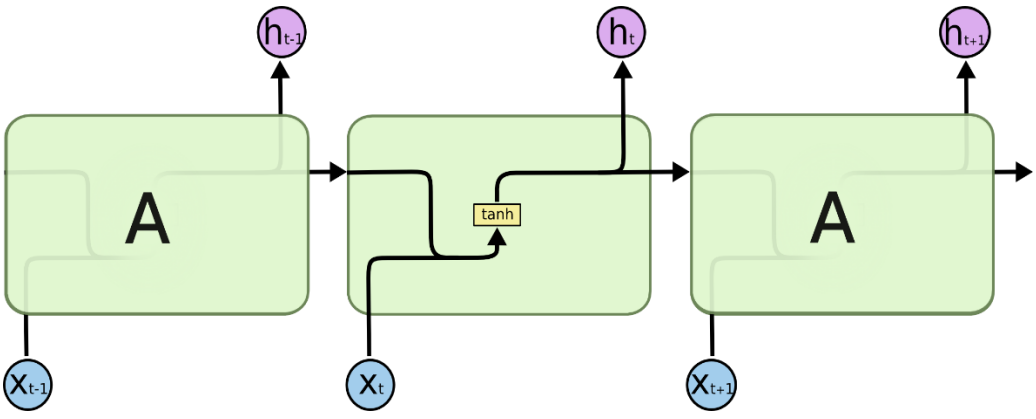

Fig. 14 Arquitectura red recurrente [34]

LSTM

Para resolver los problemas que presenta la red recurrente básica con las relaciones de tiempo a largo plazo, Hochreiter y Schmidhuber propusieron un nuevo tipo de redes recurrentes, las LSTM (Long Short-Term Memory) [35]. A diferencia de las redes recurrentes básicas, que presentan un módulo con una sola capa, estas tienen un módulo o red neuronal con cuatro capas.

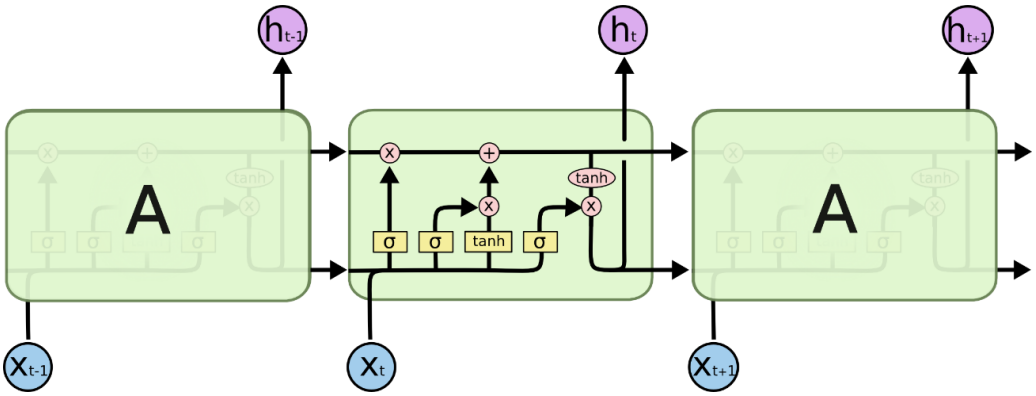

Fig. 15 Arquitectura LSTM [34]

Esta arquitectura se ha empleado en los últimos años para resolver una gran variedad de problemas secuenciales mostrando resultados mejores que la red recurrente básica.

4.2 Etiquetado

Para poder entrenar a la red neuronal es imprescindible disponer de una base de datos con imágenes etiquetadas. Cuanto mayor sea este conjunto de imágenes, mejores resultados se obtienen.

Como se ha comentado en el apartado de especificaciones del encargo, se dispone de una base de datos que contiene imágenes de ensayos realizados con *C. elegans* en placas de Petri. Sin embargo, no están etiquetadas correctamente, por lo que se ha tenido que llevar a cabo este proceso antes de poder comenzar con la programación de la red neuronal.

Para etiquetar estas imágenes se ha empleado un programa desarrollado por miembros del instituto ai2. Este programa dispone de un algoritmo de detección de *C. elegans* vivos utilizando técnicas de segmentación, lo cual facilita mucho la tarea de etiquetado. Sin embargo, este algoritmo automático puede presentar errores, por lo que se han etiquetado manualmente errores.

Al iniciar la aplicación, se selecciona la carpeta donde se encuentra el experimento que se quiere etiquetar. Una vez seleccionado, el usuario visualiza las imágenes con la detección de *C. elegans* realizada por el algoritmo, que están etiquetados con un cuadrado amarillo centrado en el cuerpo del gusano. Partiendo de esta base, se realiza el etiquetado empleando la siguiente codificación:

- **Círculo rojo y número 0.** Falsa detección. El algoritmo identifica como nematodo algo que en realidad no lo es. Puede tratarse de ruido de la imagen o de una pelusa.
- **Círculo rojo y número 8.** Nematodo muerto. Cuando observando la secuencia de imágenes no se aprecia movimiento. Se comprueba también con las imágenes del día anterior y posterior.
- **Círculo verde**. Nematodo vivo. Durante la secuencia de imágenes se observan cambios de posición, movimiento de la cabeza o de la cola.

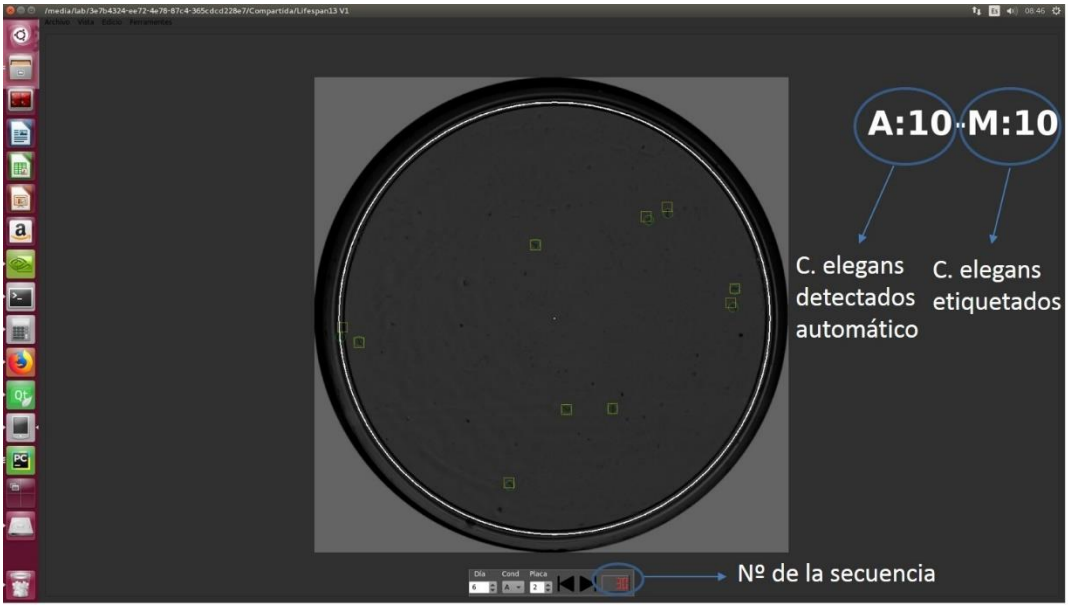

Fig. 16 Programa empleado para el etiquetado

Como se ha comentado anteriormente, en la organización de la base de datos, se dispone de 30 imágenes por cada día. El etiquetado se realiza en la última imagen del día, ya que la decisión se toma tras visualizar la secuencia completa. Tras la visualización, se identifican los posibles *C. elegans* teniendo en cuenta los siguientes indicios:

- Los nematodos tienen una longitud comprendida entre 40-60 píxeles y un ancho de 3-4 píxeles.
- El color negro del cuerpo.
- Se ha observado durante la secuencia un movimiento de tipo senoidal.

Para evitar errores en el etiquetado, se deben considerar los siguientes problemas que pueden suceder:

- Aparecen en la placa pelusas o ruido en la imagen que pueden confundirse con el gusano. Las pelusas se pueden identificar observando su movimiento, ya que se produce en forma de traslaciones en lugar de movimientos senoidales. Además, en la mayoría de los casos presentan un ancho menor y una mayor longitud que la de los *C. elegans*.

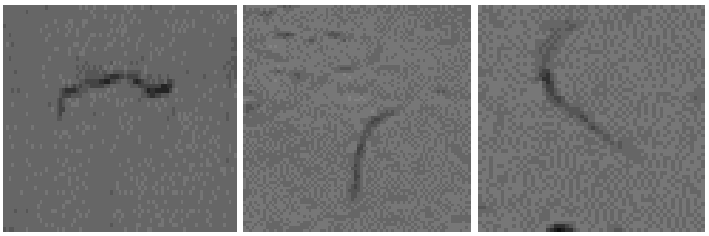

Fig. 17 Ejemplos de pelusas que aparecen la placa de ensayo

- Cuando el gusano se encuentra en los bordes de la placa o aparece ruido, contaminación o pelusas en la imagen que impide verlo.

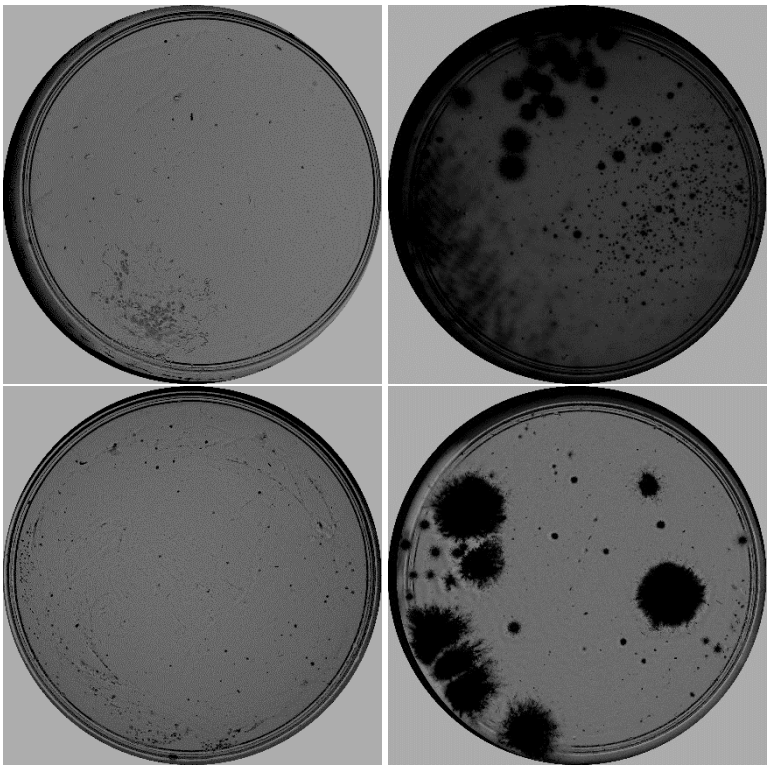

Fig. 18 Ejemplos de placas con elementos que dificultan el etiquetado

Una vez identificados los *C. elegans*, se determina si están vivos o muertos utilizando el siguiente criterio:

1. Si se ha observado movimiento en la secuencia de 30 imágenes, el *C. elegans* está vivo ese día.
2. Si durante la secuencia no se observa movimiento, se busca en la imagen del día anterior y en caso de que el gusano no estuviera en la misma posición y postura se etiqueta como vivo.
3. De lo contrario, se observa en los días posteriores y si sigue sin variar la posición se etiqueta como muerto.

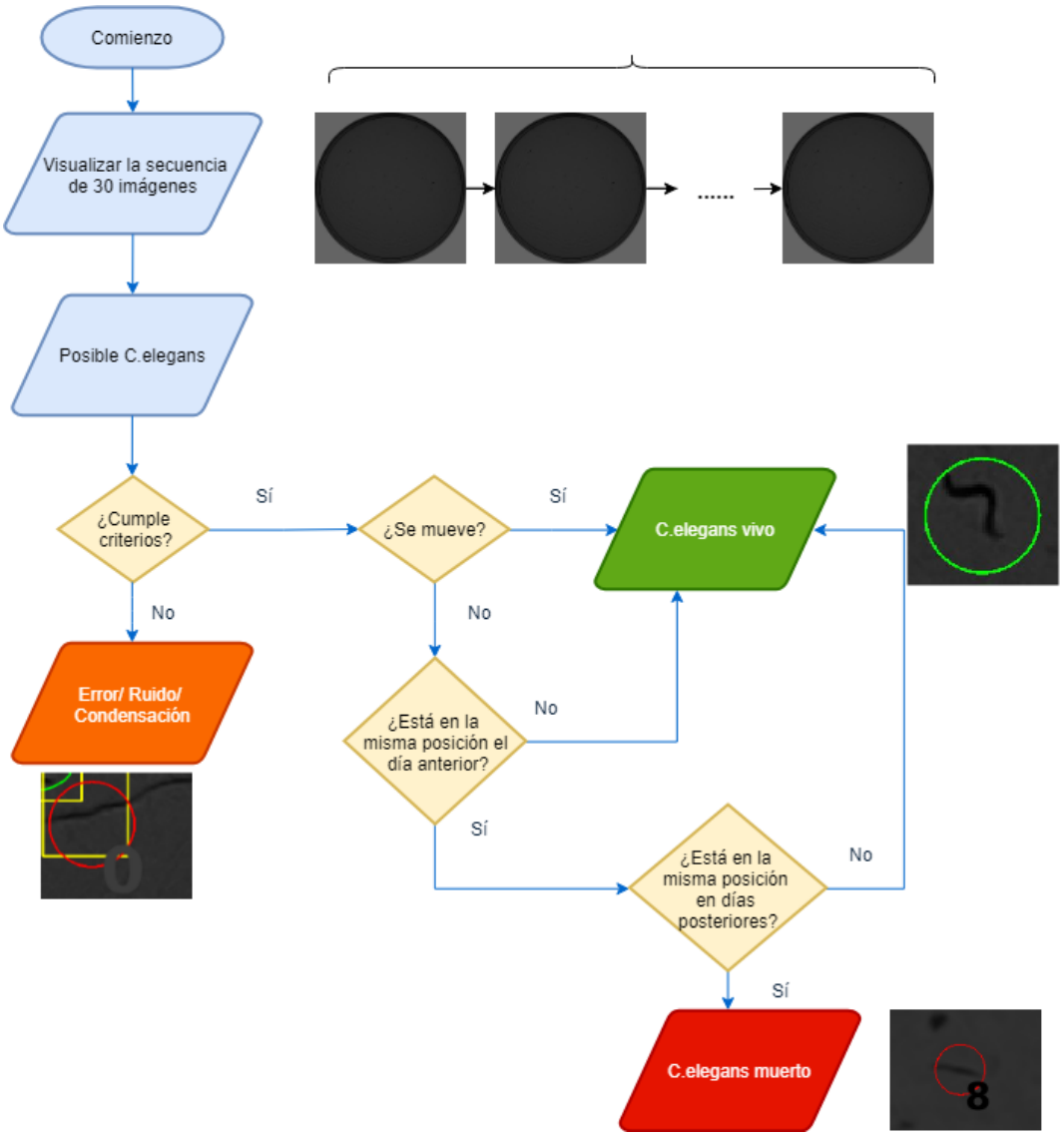

Fig. 19 Diagrama de flujo del procedimiento de etiquetado

El programa genera un archivo xml llamado conteoManual.xml y otro llamado errors.xml (*Fig. 20 y Fig. 21*) en los que se guardan las coordenadas (x,y) del centro de la etiqueta. El archivo conteoManual contiene los gusanos vivos y el de errores los que han sido etiquetados como muertos y como errores del algoritmo automático. Esto permitirá generar las imágenes de entrenamiento, que se detallan en apartados posteriores.

```
<?xml version="1.0"?>
<opencv_storage>
<CE_Manuals>
  1228 594 545 890 1221 197 917 1845 978 1766</CE_Manuals>
</opencv_storage>
```

Fig. 20 Ejemplo archivo conteoManual.xml

```
<?xml version="1.0"?>
<opencv_storage>
<CE_Errors0>
  1831 677</CE_Errors0>
<CE_Errors1></CE_Errors1>
<CE_Errors2></CE_Errors2>
<CE_Errors3></CE_Errors3>
<CE_Errors4></CE_Errors4>
<CE_Errors5></CE_Errors5>
<CE_Errors6></CE_Errors6>
<CE_Errors7></CE_Errors7>
<CE_Errors8>
  1390 1209 1241 1787</CE_Errors8>
<CE_Errors9></CE_Errors9>
</opencv_storage>
```

Fig. 21 Ejemplo archivo errors.xml

4.3 Elección de arquitecturas de redes neuronales

En esta fase, se han propuesto diferentes arquitecturas de redes neuronales para realizar la clasificación de las imágenes. El problema a resolver necesita del análisis de secuencias de imágenes, ya que la clasificación de un *C. elegans* como vivo o muerto se basa en el desplazamiento y en los cambios de forma.

4.3.1 Redes neuronales convolucionales

Se ha decidido emplear este tipo de redes dado su gran despeño en la tarea de clasificación de imágenes. Entrenar una red convolucional requiere una gran cantidad de imágenes, ya que están formadas por millones de parámetros que deben ser ajustados.

En los casos en los que no se dispone de un *dataset* amplio, pueden aparecer problemas a la hora de entrenar la red. Por esta razón, en lugar de diseñar una red convolucional desde cero, se ha empleado la técnica de transferencia de los pesos aprendidos (*transfer learning)*. Esta técnica consiste en utilizar modelos que ya han sido entrenados empleando un gran conjunto de imágenes. Esto permite aprovechar el conocimiento adquirido por la red para resolver nuestro problema. Se utilizan los parámetros aprendidos en las primeras capas entrenando con otro *dataset* y se ajustan las últimas capas para adaptarlas a nuestro modelo.

El modelo empleado en este proyecto ha sido la arquitectura ResNet18[36] entrenada con el *dataset* ImageNet[37], que dispone de más de 1,2 millones de imágenes etiquetadas en 1000 clases de objetos.

Esta arquitectura tiene como entrada una imagen, por lo que en apartados posteriores se proponen métodos para incluir la secuencia a analizar en una sola imagen.

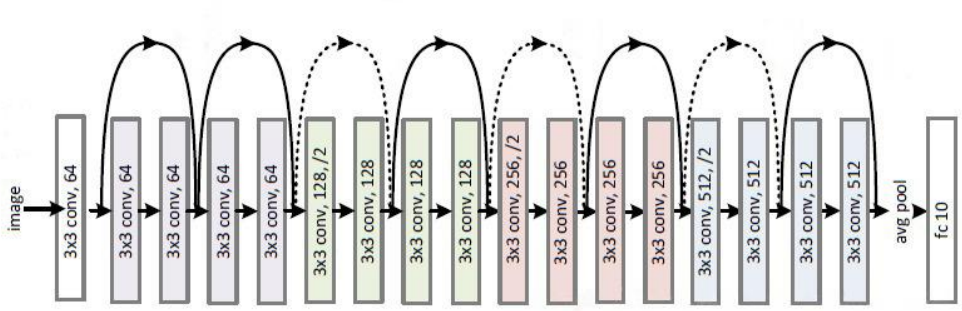

Fig. 22 Arquitectura resnet18

4.3.2 Combinación de redes neuronales convolucionales (CNN) y recurrentes (LSTM)

Como se ha comentado anteriormente, las redes neuronales convolucionales permiten extraer características de las imágenes, sin embargo, nuestro problema tiene una componente temporal, por lo que puede que esta arquitectura no sea suficiente para obtener los altos porcentajes de precisión que exige el proyecto.

Para resolver este posible inconveniente se ha planteado una alternativa que combina redes neuronales convolucionales con redes neuronales recurrentes tipo LSTM, que como se ha explicado en la introducción teórica, son las más apropiadas para las dependencias temporales largas.

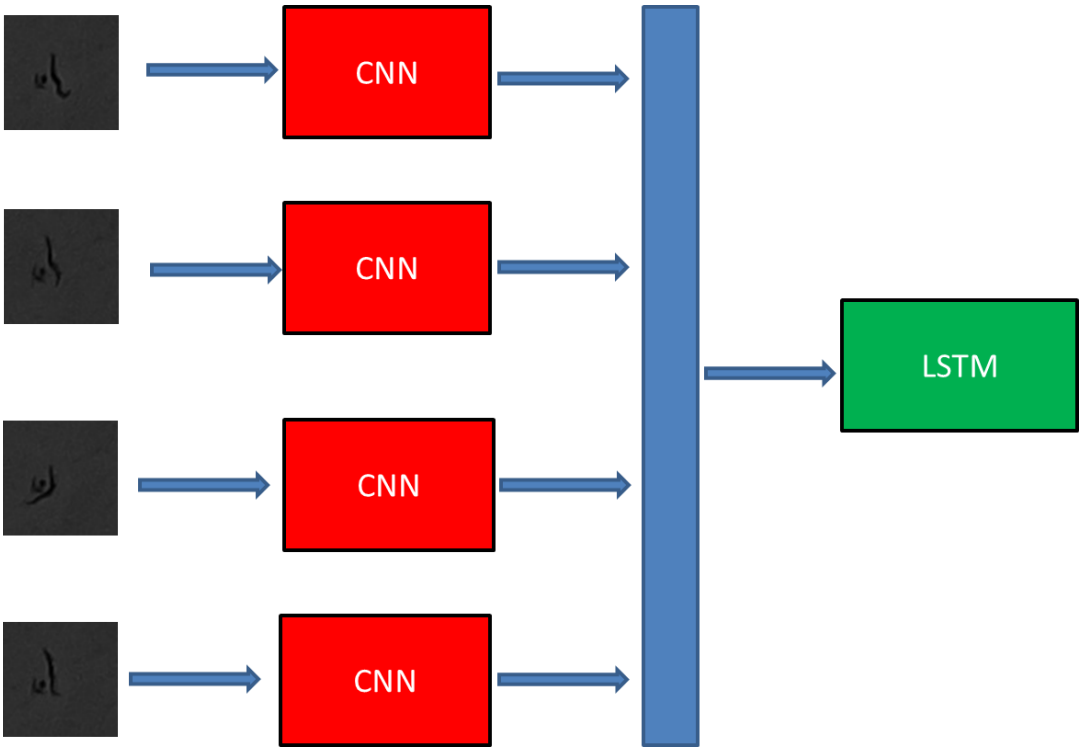

Fig. 23 Esquema propuesta CNN + LSTM

A diferencia de la arquitectura anterior, en la que se introduce toda la información en una sola imagen, con esta propuesta se introduce una secuencia de imágenes como entrada. Combinando estas arquitecturas se puede aprovechar la capacidad de la red convolucional para obtener características de las imágenes junto con la capacidad de la red LSTM de reconocer las relaciones de dinámica temporal.

4.4 Generación de imágenes

Con el objetivo de obtener un clasificador que permita discernir si un *C. elegans* está vivo o muerto, se han generado nuevas imágenes a partir de las originales y las etiquetas. La idea consiste en que el clasificador sea capaz de saber si el *C. elegans* está vivo en ese día en concreto. Para ello, debe analizar el desplazamiento y los cambios de forma del gusano.

De manera análoga al procedimiento empleado en la fase de etiquetado, la red neuronal necesita comprobar si en el día anterior se encontraba en la misma posición y forma. También debe comprobar si en el día posterior se mantiene o se modifica la posición y la forma. Por tanto, el clasificador debe tener como entrada como mínimo la imagen del día actual y las imágenes del día anterior y posterior.

Con el objetivo de introducir esta información en una sola imagen cuadrada, se han planteado dos métodos, que se describen en este apartado y se evalúan en apartados posteriores para comprobar cual obtiene mejores resultados. Estos dos métodos se emplean para el entrenamiento de la red neuronal convolucional.

El tercer método propuesto consiste en introducir la secuencia de imágenes en lugar de unirlas en una sola como se hace en los dos métodos anteriores. Este método sirve para generar las secuencias de entrada a la LSTM.

4.4.1 Método 1. Mosaico de 4 ventanas

Estas nuevas imágenes consisten en un mosaico formado por 4 sub-imágenes. Cada sub-imagen es una ventana cuadrada (recorte de la imagen original) cuyo centro es el centro de la etiqueta del *C. elegans*. Una de ellas es una ventana con el *C. elegans* etiquetado, otra corresponde a una ventana en la misma posición en la placa en el día anterior y las otras 2 corresponden a los dos días posteriores, también en la misma posición. Las imágenes de nematodos vivos se nombrarán con el código 01 y los muertos con el código 00 al principio del nombre a la hora de guardarlas para poder identificarlas posteriormente cuando se separen al organizar las carpetas de entrenamiento.

Fig. 24 Propuesta de mosaico de 4 ventanas

Elección del tamaño de las ventanas

A la hora de elegir el tamaño de la ventana se ha tenido en cuenta que el tamaño máximo de un *C. elegans* es de 60 píxeles de longitud. Sin embargo, las placas presentan una ligera variación en su posición, ya que se colocan cada día y esto puede hacer que aparezcan pequeños desplazamientos. Por este motivo, las ventanas deben ser de un tamaño mayor para asegurar que el *C. elegans* aparece completamente dentro de ella.

Para determinar estos desplazamientos, se ha empleado una funcionalidad del programa de etiquetado que muestra las coordenadas (x, y) de cada píxel de la imagen.

De esta forma, se han tomado valores de los píxeles de un gusano muerto, cuya posición no debería variar en los días posteriores, salvo que se produjese la descomposición de este o se moviera la placa. Estos valores se han tomado en zonas claramente identificables, como son la cola y la cabeza.

Tras analizar esta variación, se obtiene como conclusión que el máximo desplazamiento es de unos 15 píxeles. Para tener un margen de error, se ha considerado el máximo desplazamiento de 20 píxeles. De este modo, cada ventana cuadrada tiene 80 píxeles de lado. Por tanto, las imágenes generadas tienen un tamaño de 160 x 160 píxeles.

Generación de la ventana

Como se ha mencionado anteriormente, cada ventana cuadrada es un recorte de la imagen de la placa completa y su centro es el centro del círculo empleado para el etiquetado. A continuación, se explica el procedimiento seguido para crear la ventana.

En primer lugar, la función recibe como parámetros la ruta de la imagen a recortar, las coordenadas (x, y) de la etiqueta del *C. elegans* que se quiere analizar y el alto y ancho de la ventana.

Seguidamente, se abre la imagen, que está en formato de archivo binario comprimido y se transforma en un vector de dos dimensiones con 1944 x 1944 valores, es decir, el vector de una imagen en escala de grises. Este vector se convierte a imagen haciendo uso de la función *Image.fromarray* de la librería de procesamiento de imagen PIL.

Para recortar la imagen es necesario calcular las coordenadas de la esquina superior izquierda (w_1, h_1) y la esquina inferior derecha (w_2, h_2) de la ventana:

$$w_1 = Vx - w/2$$

$$h_1 = Vy - h/2$$

Ecuación 1 Coordenadas de la esquina superior izquierda de la ventana

$$w_2 = w_1 + w$$

$$h_2 = h_1 + h$$

Ecuación 2 Coordenadas de la esquina inferior derecha de la ventana

Donde (w_1, h_1) y (w_2, h_2) son las coordenadas (x, y) de la esquina superior izquierda y la esquina inferior derecha respectivamente. Los valores w y h son el ancho y el alto de la ventana. Por último, los valores (Vx, Vy) son las coordenadas de la etiqueta del *C. elegans*.

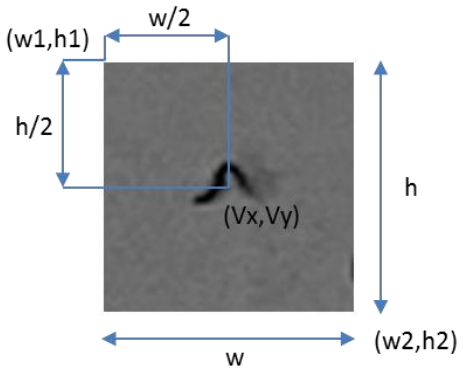

Fig. 25 Esquema del cálculo de la ventana

Es necesario tener en cuenta que existen zonas críticas en las que la ventana calculada aplicando las fórmulas anteriores puede quedar fuera de los límites de la imagen. Estas zonas son aquellas en las que (w_1, h_1) se hacen negativos y en las que (w_2, h_2) son mayores que 1944.

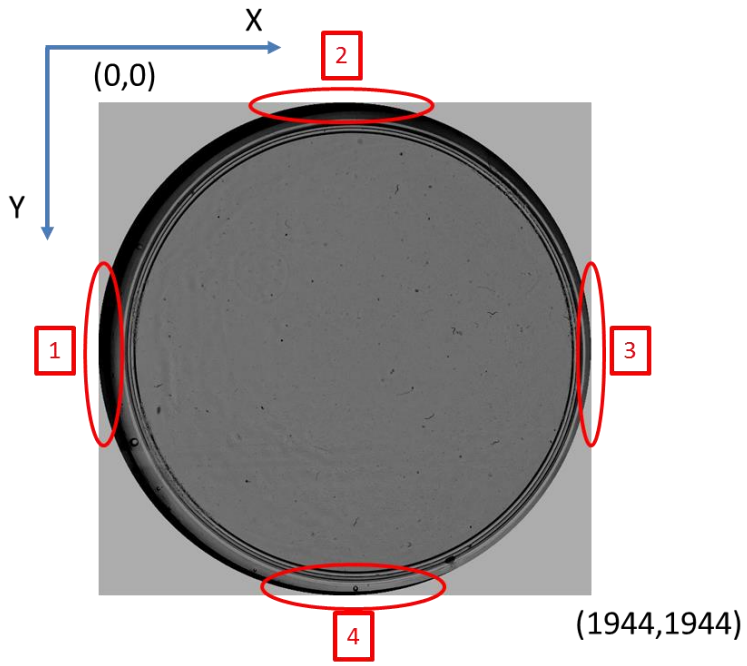

Fig. 26 Zonas críticas a la hora de generar ventanas

Para estos casos, se ha optado por desplazar la ventana de forma que no salga de los límites.

1) Caso en el que se hace negativa w_1

$$w_1 = 0$$

Ecuación 3 Cambio en las ecuaciones para el punto crítico 1

El resto de los valores se calcula con la ecuación 1 y ecuación 2

2) Caso en el que se hace negativa h_1

$$h_1 = 0$$

Ecuación 4 Cambio en las ecuaciones para el punto crítico 2

El resto de los valores se calcula con la ecuación 1 y ecuación 2

3) Caso en el que w_2 es mayor que 1944

$$w_1 = w_1 - (w_2 - 1944)$$

$$w_2 = 1944$$

Ecuación 5 Cambio en las ecuaciones para el punto crítico 3

El resto de los valores se calcula con la ecuación 1 y ecuación 2

4) Caso en el que h_2 es mayor que 1944

$$h_1 = h_1 - (h_2 - 1944)$$

$$h_2 = 1944$$

Ecuación 6 Cambio en las ecuaciones para el punto crítico 4

El resto de los valores se calcula con la ecuación 1 y ecuación 2

Una vez calculadas las coordenadas se recorta la imagen empleando la función *crop* de la librería PIL.

En el siguiente diagrama de flujo se muestra el código empleado:

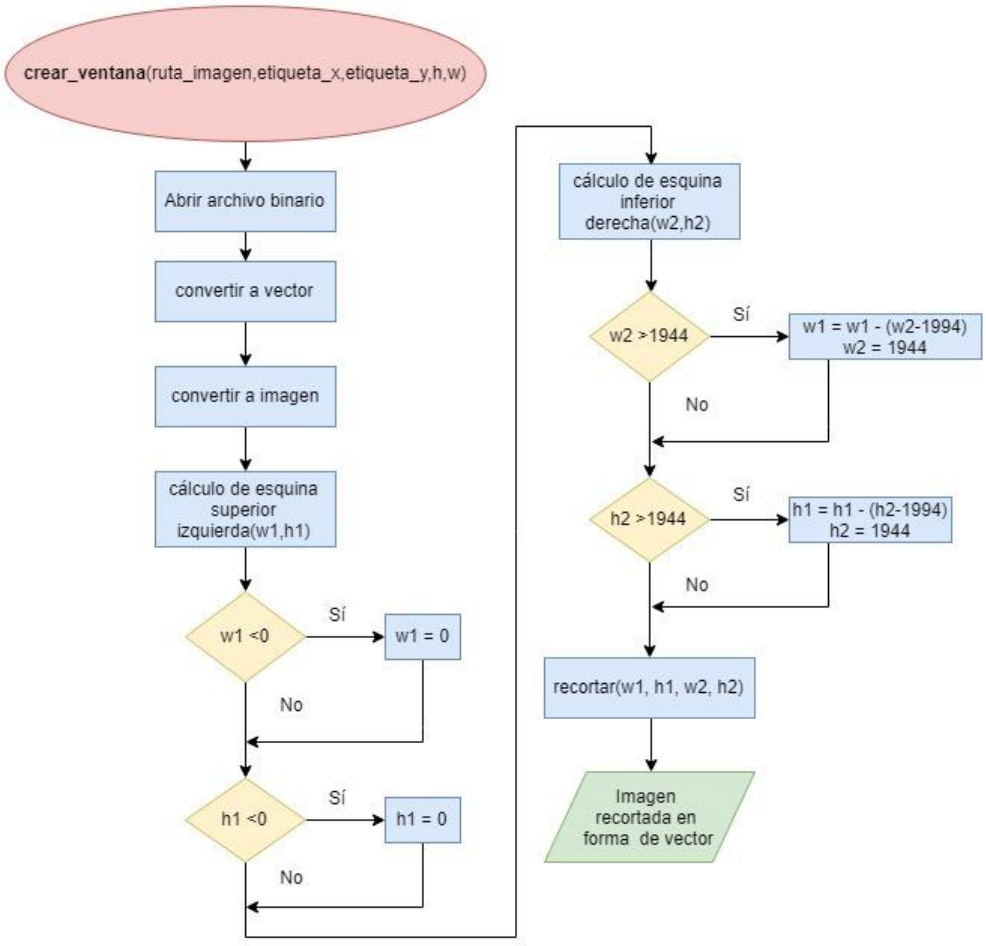

Fig. 27 Diagrama de flujo función crear ventana

Imagen final

Tras obtener las cuatro ventanas con el procedimiento explicado, se forma el mosaico uniendo las imágenes en una sola. Esto se realiza con las funciones de OpenCV *cv2.hconcat() y cv2.vconcat().*

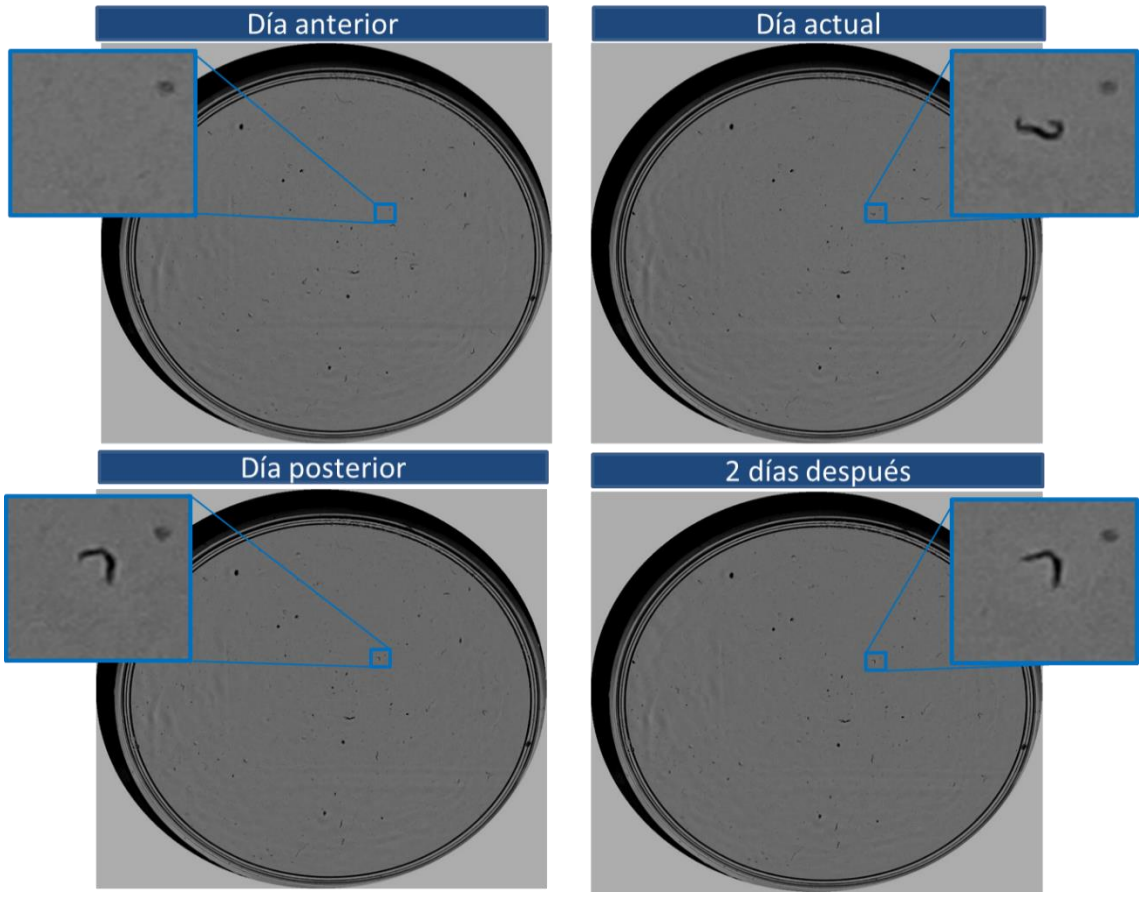

Fig. 28 Procedimiento para la extracción de ventanas

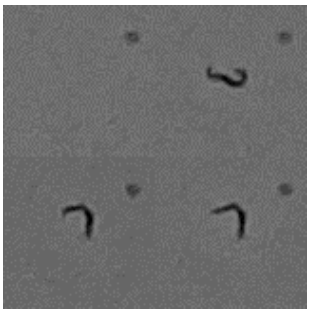

Fig. 29 Resultado final tras realizar la concatenación

A continuación, se muestran algunos ejemplos de imágenes de las dos categorías: muertos y vivos.

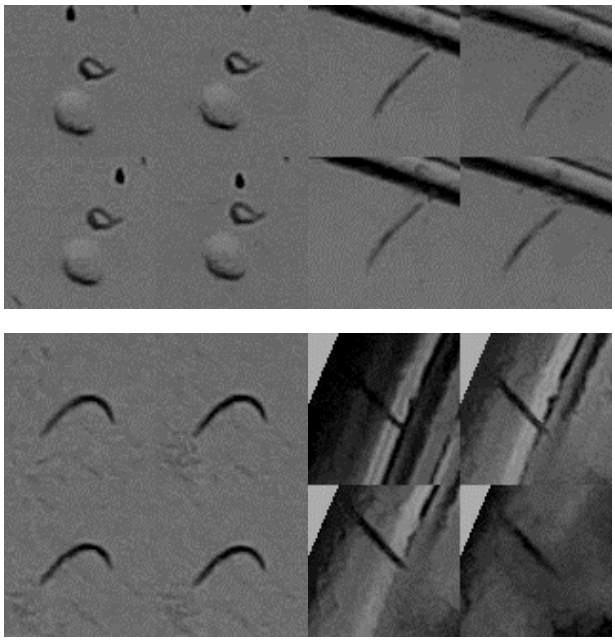

Fig. 30 Ejemplos de imágenes de C. elegans muertos con el método del mosaico de 4 ventanas

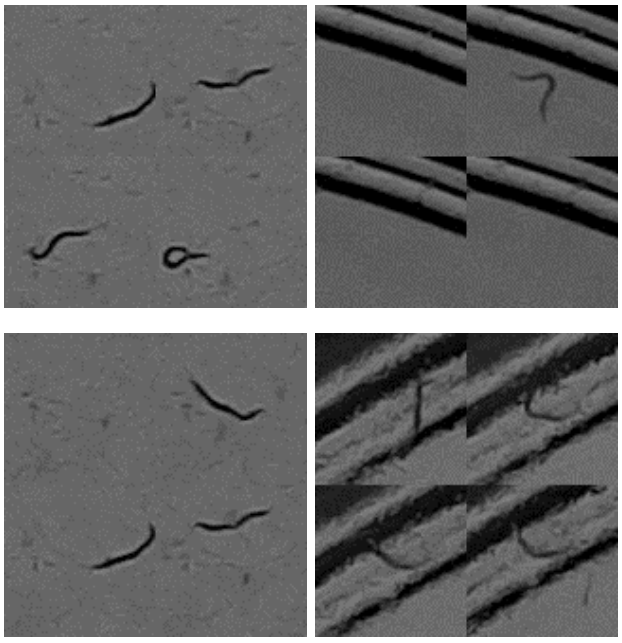

Fig. 31 Ejemplos de imágenes de C. elegans vivos con el método del mosaico de 4 ventanas

4.4.2 Método 2. Canales RGB

Las imágenes de color se pueden descomponer en el espacio RGB como la composición de tres imágenes monocromas, cada una de ellas correspondiente a un color primario: rojo, verde y azul. Por esto, se dice que una imagen de color posee tres canales. Para una imagen RGB de 24 bits, cada canal tiene 8 bits, de forma que cada píxel tiene un valor intensidad comprendido entre 0 y 255 en cada canal.

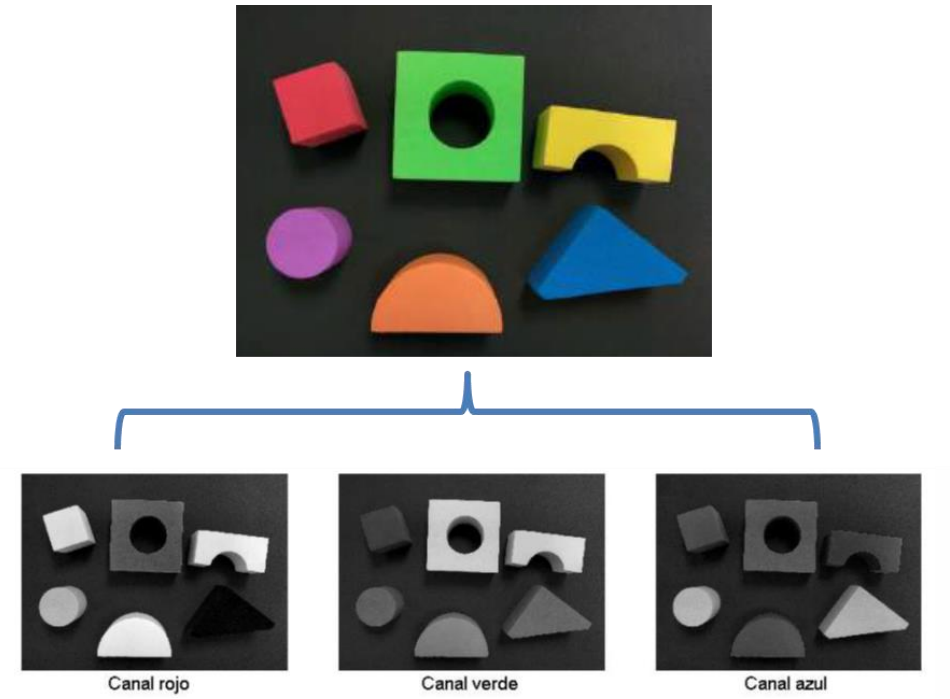

Fig. 32 Descomposición RGB de una imagen de color. Adaptada de [38]

Esta alternativa consiste en crear imágenes de color incluyendo la información de cada día en un canal. Se pretende comparar los resultados de esta alternativa con los del método anterior, para averiguar si alguna de las dos formas facilita más el aprendizaje de la red neuronal convolucional.

Para generar estas imágenes se han utilizado de nuevo las ventanas cuadradas, pero esta vez se han empleado imágenes de tres días en lugar de cuatro como en el caso anterior. De esta forma, en cada canal RGB de la imagen se ha incluido la información de un día.

Se ha cambiado el orden de RGB a BGR, porque las funciones empleadas para combinar los tres canales de las imágenes, *cv2.merge()*, son de la librería OpenCV, y esta almacena las componentes en orden inverso. De esta forma la información queda almacenada de la siguiente manera:

- Canal azul: día anterior
- Canal verde: día actual
- Canal rojo: día posterior

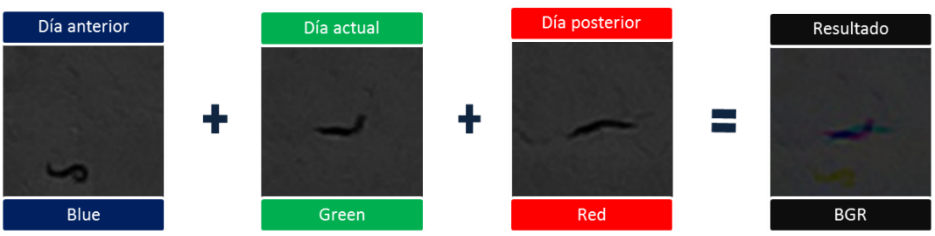

Fig. 33 Ejemplo de imagen de C. elegans vivo

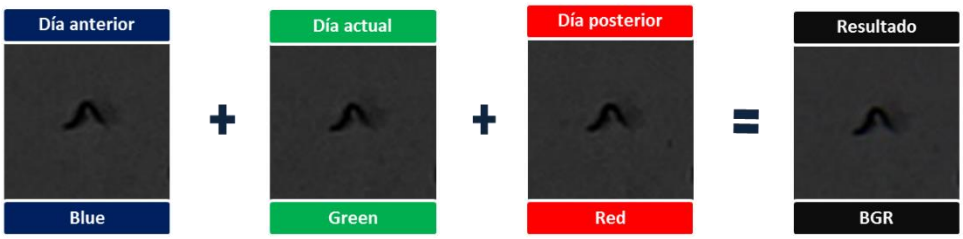

Fig. 34 Ejemplo de imagen de C. elegans muerto

Esta alternativa proporciona menos información para realizar la clasificación, ya que contiene las imágenes de 3 días en lugar de 4 como en la alternativa anterior. Además, el tamaño de la imagen es menor, al ser una ventana de 80x80 píxeles, por tanto, ocupan menos memoria.

4.4.3 Método 3. Secuencia de 4 imágenes

Este método es análogo al empleado para generar el mosaico de 4 ventanas, con la diferencia de que en este caso se genera cada ventana como imagen independiente en lugar de concatenarlas en una sola. De esta forma, se obtiene una secuencia de 4 imágenes que serán la entrada de la arquitectura CNN + LSTM.

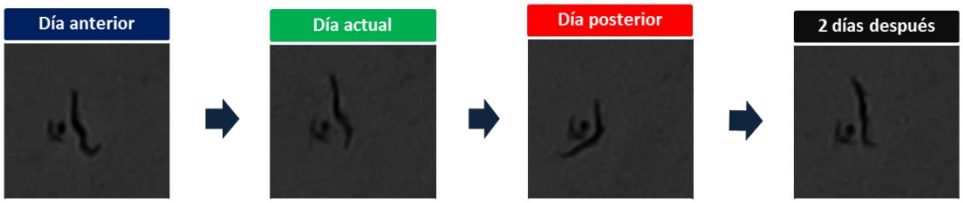

Fig. 35 Método secuencia de 4 imágenes

4.5 Diseño de experimentos para elegir la mejor alternativa

Tras generar el *dataset* mediante 3 métodos y realizar el planteamiento de distintas arquitecturas de redes neuronales surgen algunas preguntas:

1) ¿Cuál es la mejor forma de introducir los datos?
2) ¿Qué arquitectura de red neuronal obtiene precisiones más altas?
3) ¿Qué arquitectura de red neuronal aprende más rápido?
4) ¿Es capaz la red de aprender con un conjunto pequeño de imágenes?
5) ¿Qué imágenes le cuesta aprender?
6) ¿Cuál es la mejor manera de aumentar el *dataset*?

En este apartado se describen las pruebas que se han realizado para intentar resolver estas cuestiones y elegir la mejor solución para el proyecto.

4.5.1 Experimento 1

En este primer experimento se comparan los resultados de entrenar la red con las imágenes de un solo experimento frente a entrenar con un *dataset* compuesto por imágenes de todos los experimentos etiquetados. Para poder realizar esta comparativa, se deben evaluar los distintos modelos empleando el mismo *dataset* de validación.

Como *dataset* de validación se ha generado una carpeta que contiene una muestra aleatoria de imágenes de todos los experimentos. De esta forma, podemos tener una prueba significativa del modelo ante los casos que se puede encontrar en distintos experimentos.

Con este experimento se pretende comprobar con qué tipo de red se obtienen mejores resultados, qué tipo de imágenes (mosaico o RGB) permiten a la red convolucional aprender mejor y si se puede aprender entrenando con pocas imágenes o si es necesario tener un *dataset* con muchas imágenes.

4.5.2 Experimento 2

Existen diversas maneras de aumentar el *dataset,* una de ellas es aplicando transformaciones a las imágenes, como pueden ser rotaciones, centrado de imágenes, efecto espejo, etc. En este proyecto se ha planteado además la posibilidad de realizar transformaciones teniendo en cuenta los errores que presenta el algoritmo al clasificar las imágenes de validación.

Por lo anteriormente expuesto, se ha realizado un análisis de los errores de clasificación del primer experimento. Una vez identificadas estas imágenes, se han propuesto métodos para generar nuevas imágenes a partir de las disponibles que simulen estos errores. De esta forma, se busca optimizar los resultados obtenidos en las primeras pruebas realizadas.

4.6 Implementación y entrenamiento de las redes

En este apartado se explica de forma resumida el procedimiento seguido para la implementación del código necesario para programar la red neuronal y realizar el entrenamiento.

Se ha realizado empleando la plataforma de *Deep Learning* Pytorch. Esta plataforma ha sido desarrollada por el grupo de inteligencia artificial de Facebook. Se ha elegido esta opción, ya que presenta una buena documentación con gran cantidad de ejemplos y su uso en la comunidad dedicada a la investigación está creciendo mucho en los últimos años. Además, dispone de soporte para su ejecución mediante GPU utilizando la API CUDA de NVIDIA.

En primer lugar, se divide el *dataset* en imágenes de entrenamiento y validación. Un 70 % de las imágenes se emplearán para entrenar y un 30 % para validar. Esta división se ha hecho de forma que las clases estén balanceadas, es decir, que exista el mismo número de imágenes de gusanos vivos que de gusanos muertos para evitar errores en el aprendizaje.

En segundo lugar, se organizan los directorios del proyecto de forma que en la carpeta principal existan los siguientes subdirectorios:

- Carpeta **IMAGES**. Contiene el *dataset* y está a su vez dividido en las carpetas train y val, que contienen las imágenes de entrenamiento y validación respectivamente. Cada una de estas carpetas contiene dos carpetas correspondientes a las dos clases: vivos y muertos.
- Carpeta **models_pytorch**. Aquí se guardarán los modelos al final del entrenamiento para poder emplearlos como clasificador o para volver a entrenarlos si se desea.

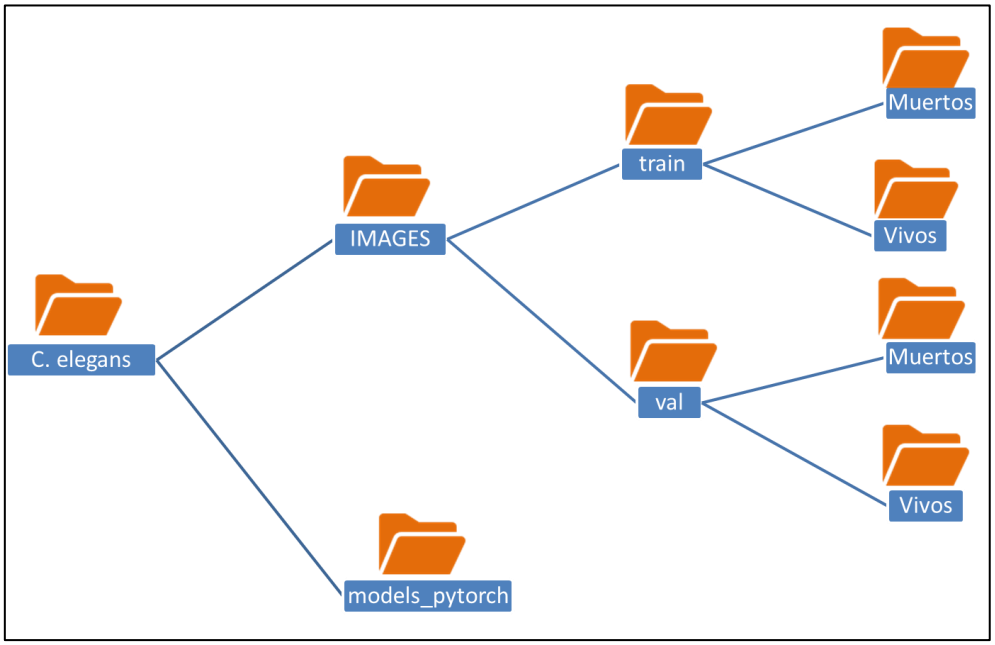

Fig. 36 Organización de las carpetas del proyecto

Una vez organizado el *dataset* y los directorios del proyecto, podemos proceder a la implementación del código. Aquí se enumeran los pasos a seguir de forma resumida, ya que para realizar este programa simplemente se ha consultado la documentación oficial de Pytorch [39], adaptando los ejemplos a nuestro caso.

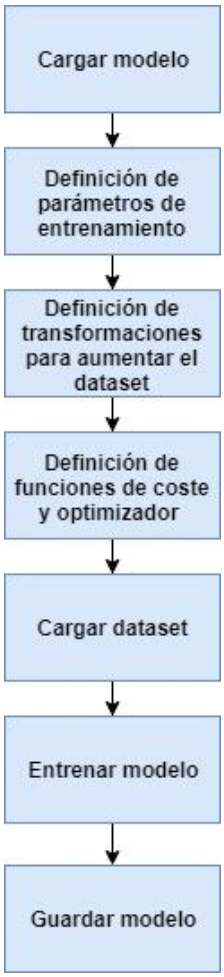

Fig. 37 Procedimiento de entrenamiento red neuronal

A continuación, se incluyen las definiciones de los principales parámetros de entrenamiento y se comentan los valores elegidos:

- ***Batch size*** (tamaño de lote). Es el número de muestras empleadas en cada iteración de la etapa de entrenamiento. Realizando unas pruebas se ha observado que este parámetro no afecta significativamente a los resultados de nuestro modelo. Sin embargo, sí que se ha comprobado que elegir un valor muy alto puede acarrear problemas de memoria. Por tanto, se ha mantenido el valor del ejemplo de la documentación de Pytorch, que es de 32 imágenes.
- ***Epoch*** (número de épocas). Es el número de veces que el algoritmo recorre todo el conjunto de muestras de entrenamiento. En este proyecto se ha ajustado a 10, porque en las primeras pruebas realizadas se observaba que la red aprende solo durante las primeras épocas y en las siguientes épocas las curvas de precisión y pérdida se estabilizan.
- ***Learning rate*** *(tasa de aprendizaje).* Este parámetro es un escalar que se utiliza durante la fase de cálculo del descenso de gradiente. Determina cuanto afecta el gradiente a la actualización de los parámetros. Elegir un valor muy pequeño hace que se necesiten muchas iteraciones para llegar al mínimo y elegir un valor

muy elevado puede hacer que nunca se alcance. Por ello, también se ha mantenido el valor del ejemplo, que es de 0,001.

5. Resultados obtenidos y comparativas

En este apartado se muestran los resultados de los experimentos realizados y se compara entre ellos. Para realizar estas comparativas se han empleado como métricas: la matriz de confusión, la precisión, el *recall, f1-score* y el tiempo de entrenamiento.

A continuación, se definen las métricas que se han empleado para realizar las comparativas:

- La **matriz de confusión** es una forma empleada para analizar los resultados de un problema de clasificación. Muestra el número de predicciones correctas y erróneas durante la validación del modelo para cada una de las clases. Esto permite identificar en que imágenes comete los fallos nuestro modelo.

	Predicción muerto	**Predicción vivo**
Etiqueta muerto	VM	FV
Etiqueta vivo	FM	VV

Tabla 1 Matriz de confusión para nuestro modelo

Donde VM representa los verdaderos gusanos muertos, FV los falsos gusanos vivos, FM los falsos gusanos muertos y VV los verdaderos gusanos vivos.

- La métrica de la **precisión** nos indica qué porcentaje de las veces que predice una clase, lo hace de forma correcta.

$$precisión\ muertos = \frac{VM}{VM + FM}$$

Ecuación 7 Fórmula precisión muertos

$$precisión\ vivos = \frac{VV}{VV + FV}$$

Ecuación 8 Fórmula precisión vivos

- La métrica ***recall*** nos indica el porcentaje de acierto al clasificar las imágenes de una clase respecto al total de muestras de dicha clase.

$$recall\ muertos = \frac{VM}{VM + FV}$$

Ecuación 9 Fórmula recall muertos

$$recall\ vivos = \frac{VV}{VV + FM}$$

Ecuación 10 Fórmula recall vivos

- La métrica **F1 score** es una media armónica entre los valores de precisión y *recall*. Se emplea esta métrica para mostrar el equilibrio del algoritmo entre las dos métricas anteriores, es decir, que clasifica correctamente las imágenes de una clase, pero no se equivoca con otras clases. Su fórmula es la siguiente:

$$F_1 = 2 \times \frac{precisión \times recall}{precisión + recall}$$

Ecuación 11 Fórmula F1 score

5.1 Experimento 1

5.1.1 Entrenamiento con las imágenes de un solo experimento

A continuación, se muestran las matrices de confusión obtenidas tras realizar 5 entrenamientos de 10 épocas con cada una de las alternativas:

	Mosaico 4 ventanas CNN	
	Predicción muerto	**Predicción vivo**
Etiqueta muerto	2195	364
Etiqueta vivo	566	1993

Tabla 2 Matriz de confusión mosaico 4 ventanas

	Imágenes RGB CNN	
	Predicción muerto	**Predicción vivo**
Etiqueta muerto	2568	369
Etiqueta vivo	794	1993

Tabla 3 Matriz de confusión imagen RGB

	Secuencia CNN + LSTM	
	Predicción muerto	**Predicción vivo**
Etiqueta muerto	2300	252
Etiqueta vivo	625	1927

Tabla 4 Matriz de confusión secuencias

En la siguiente tabla se muestran los tiempos empleados por cada alternativa en un entrenamiento:

	Entrenamiento con imágenes de un solo experimento
Mosaico CNN	37 s
RGB CNN	21 s
Secuencia CNN+LSTM	22 s

Tabla 5 Tiempos de entrenamiento

Para poder comparar los resultados obtenidos, se han agrupado en una sola tabla y se han representado gráficamente:

	Precisión muertos	**Precisión vivos**	**Recall muertos**	**Recall vivos**	**F1- score muertos**	**F1-score vivos**	**F1-score medio**
CNN mosaico	0,795	0,846	0,858	0,779	0,825	0,811	0,818
CNN RGB	0,764	0,844	0,874	0,715	0,815	0,774	0,795
CNN + LSTM	0,786	0,884	0,901	0,755	0,840	0,815	0,827

Tabla 6 Resumen métricas experimento 1 con pocas imágenes de entrenamiento

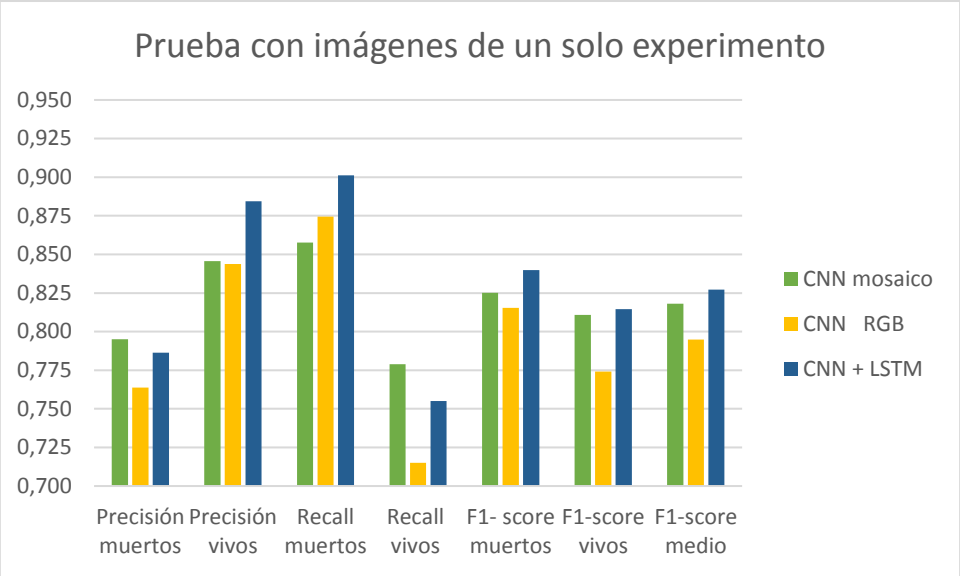

Fig. 38 Comparativa métricas experimento 1 con pocas imágenes de entrenamiento

Tras analizar los resultados del experimento 1 entrenando con imágenes de un solo experimento se obtienen las siguientes conclusiones:

- Analizando los valores *f1-score* medio, la arquitectura CNN + LSTM obtiene los mejores resultados en la clasificación de las imágenes. Se ha observado que esta arquitectura en los primeros entrenamientos obtiene peores resultados que las otras alternativas. Esto puede deberse a que las redes CNN parten de un modelo pre-entrenado (resnet18 con *dataset* ImageNet) como se ha comentado en apartados anteriores.
- La arquitectura CNN + LSTM y la CNN con imágenes RGB obtienen los menores tiempos de entrenamiento. Esto puede deberse al tamaño de las imágenes de entrada que es de 80 x 80 píxeles estas alternativas, mientras que en la del mosaico es de 160 x 160 píxeles.
- De las matrices de confusión se extrae la conclusión de que todas las arquitecturas cometen más errores al clasificar *C. elegans* vivos. Esto explica que los valores de *recall* en vivos sean bajos, mientras que los valores de precisión son altos.
- Los valores de *recall* para los gusanos muertos son elevados en comparación con los vivos, sin embargo, la precisión disminuye debido a los gusanos vivos que etiqueta como muertos.
- Los valores de acierto (82,7%) a la hora de clasificar no son lo suficientemente buenos como para pensar en utilizar un *dataset* pequeño de entrenamiento.

5.1.2 Entrenamiento con todas las imágenes

A continuación, se muestran las matrices de confusión obtenidas tras realizar 5 entrenamientos de 10 épocas con cada una de las alternativas:

	Mosaico 4 ventanas CNN	
	Predicción muerto	**Predicción vivo**
Etiqueta muerto	2288	271
Etiqueta vivo	468	2091

Tabla 7 Matriz de confusión mosaico 4 ventanas

	Imágenes RGB CNN	
	Predicción muerto	**Predicción vivo**
Etiqueta muerto	2521	416
Etiqueta vivo	466	2471

Tabla 8 Matriz de confusión imágenes RGB

	Secuencia CNN + LSTM	
	Predicción muerto	**Predicción vivo**
Etiqueta muerto	2478	74
Etiqueta vivo	251	2301

Tabla 9 Matriz de confusión secuencias

En la siguiente tabla se muestran los tiempos empleados por cada alternativa en un entrenamiento:

	Entrenamiento con todas las imágenes
Mosaico CNN	1 min 33 s
RGB CNN	49 s
Secuencia CNN+LSTM	58 s

Tabla 10 Tiempos de entrenamiento

Para poder comparar los resultados obtenidos, se han agrupado en una sola tabla y se han representado gráficamente:

	Precisión muertos	Precisión vivos	Recall muertos	Recall vivos	F1- score muertos	F1-score vivos	F1-score medio
CNN mosaico	0,830	0,885	0,894	0,817	0,861	0,850	0,855
CNN RGB	0,844	0,856	0,858	0,841	0,851	0,849	0,850
CNN + LSTM	0,908	0,969	0,971	0,902	0,938	0,934	0,936

Tabla 11 Resumen métricas experimento 1 con todas las imágenes de entrenamiento

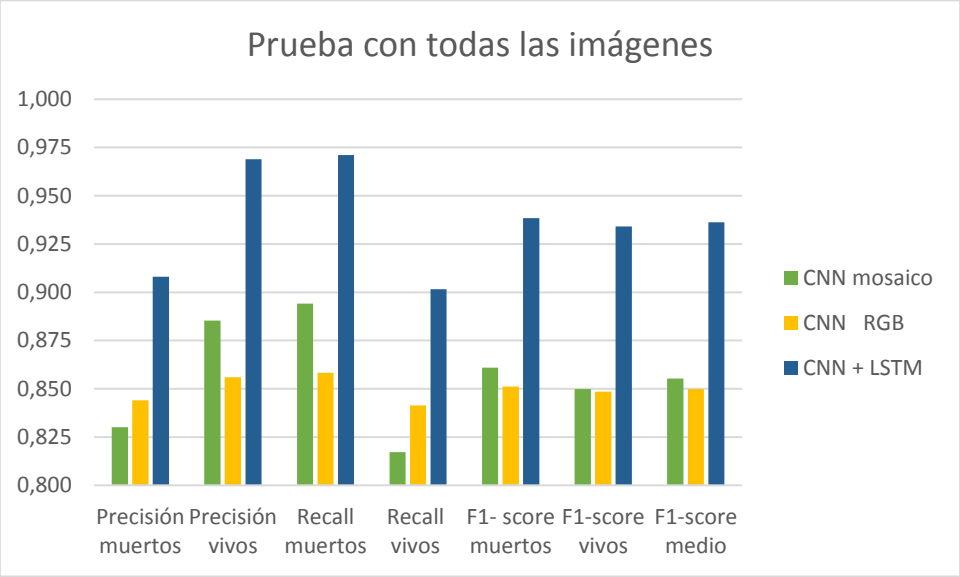

Fig. 39 Comparativa métricas experimento 1 con todas las imágenes de entrenamiento

Tras analizar los resultados del experimento 1 entrenando con todas las imágenes se han obtenido las siguientes conclusiones:

- La arquitectura CNN + LSTM obtiene los mejores resultados de acierto en todas las métricas analizadas, excepto en el tiempo de entrenamiento, pero con diferencias poco significativas. Por tanto, se puede considerar que esta es la mejor alternativa para resolver el problema de todas las propuestas realizadas en este proyecto.
- En la comparación entre la red CNN introduciendo las imágenes en forma de mosaico de 4 ventanas y las imágenes RGB se obtienen valores similares de precisión, por lo que en caso de tener que elegir una de estas propuestas sería más optima la del mosaico, que utiliza imágenes de 4 días, siendo de esta forma más fiable a la hora de detectar *C. elegans* muertos.
- Al igual que en el entrenamiento con un solo experimento, el algoritmo comete más errores a la hora de identificar gusanos vivos.
- La red CNN introduciendo las imágenes en los canales RGB obtiene los menores tiempos de entrenamiento, aunque son similares a los de la CNN + LSTM.
- Se ha comprobado, como era de esperar, que al aumentar el *dataset* de entrenamiento los resultados mejoran significativamente en el caso de la CNN + LSTM, pasando de un *f1-score* medio del 82,7% al 93,6%. En las otras alternativas ha aumentado, pero de forma más moderada.

Para averiguar qué imágenes le cuesta clasificar al algoritmo, se ha programado una prueba en la que el modelo debe clasificar las imágenes del set de validación. El programa guarda las imágenes en carpetas según la predicción que ha hecho el algoritmo. Se han nombrado de la siguiente manera: etiqueta_predicción.

Fig. 40 Carpetas test experimento 1

Tras analizar estas imágenes se han identificado posibles causas de errores:

- En imágenes donde aparece un *C. elegans* muerto y además existe ruido, puede que la red asocie este ruido con movimiento y por tanto clasifica como vivo.

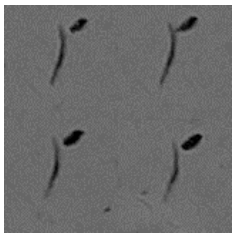

Fig. 41 Error en gusanos muertos debido al ruido

- Existen casos en los que el ruido se puede confundir con un gusano muerto y clasificar de forma incorrecta un gusano vivo.

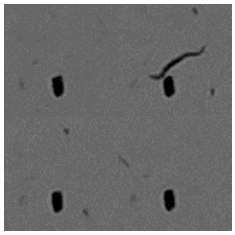

Fig. 42 Error en gusanos vivos debido al ruido

- Imágenes en las que el gusano realiza pequeños cambios de forma. En estas imágenes según el criterio de etiquetado se considera vivo, ya que, aunque el cambio es pequeño sí es significativo.

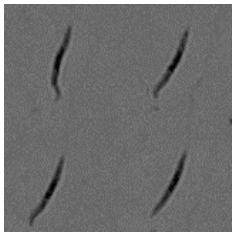

Fig. 43 Error en gusanos vivos debido a pequeños cambios de forma

- En la imagen aparece otro gusano, entonces puede que el clasificador no sepa cual debe clasificar.

Fig. 44 Error de confusión al elegir gusano

- Algunos errores proceden de imágenes con errores en la adquisición.

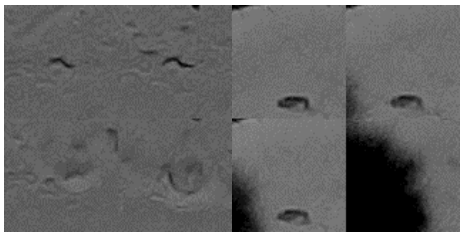

Fig. 45 Errores de adquisición

- Las imágenes del *C. elegans* cuando se encuentra en los bordes de la placa. Este tipo de imágenes genera gran cantidad de errores. Esto puede ser debido al ruido que existe en esa zona.

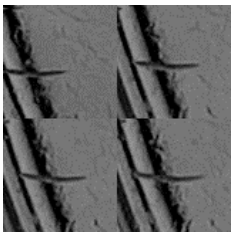

Fig. 46 Errores de gusanos en el borde de la placa

5.2 Experimento 2

En este experimento se ha aumentado el *dataset* de entrenamiento con imágenes generadas artificialmente. Estas nuevas imágenes pretenden simular errores que se han detectado en el análisis del experimento 1. Este experimento se ha realizado únicamente con la alternativa CNN + LSTM, ya que ha mostrado los mejores resultados en todas las métricas analizadas en el experimento 1.

5.2.1 Imágenes con un ruido generado de forma aleatoria.

Pretende simular los casos en los que aparecen imágenes con ruido por problemas de adquisición o contaminación de la placa. Para ello se ha realizado el siguiente procedimiento:

En primer lugar, se genera una máscara para evitar que el ruido introducido afecte al *C. elegans*. Esto se realiza en 3 pasos:

1. Se realiza una segmentación (*threshold)* de la imagen original para segmentar el gusano del fondo. Se han segmentado todos los objetos (*blobs)*, ya que tener que identificar al gusano supone un extra de procesamiento y de análisis de características.

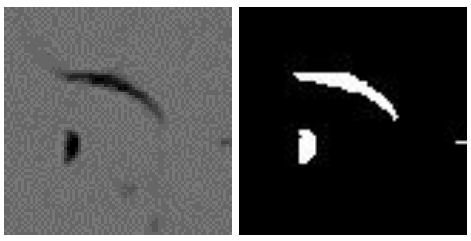

Fig. 47 Threshold imagen original

2. Se dilatan los *blobs*. Esto permite asegurar que cuando se introduzca el ruido no se modifiquen los píxeles del gusano.

Fig. 48 Dilatación del threshold

3. Se realiza la negada de la imagen anterior, es decir, lo que es blanco pasa a ser negro y viceversa. Posteriormente, se convierten a 1 los valores mayores que 0 para emplear la matriz de la imagen como máscara.

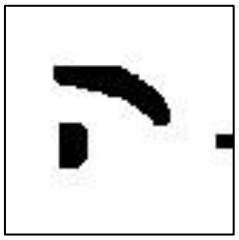

Fig. 49 Operación NOT

En segundo lugar, se genera una matriz de 80x80 y se rellenan sus valores con -20,0 o 20 de forma aleatoria. Esto permite que los valores del fondo modifiquen su color. En la imagen que se muestra a continuación, no se ve el ruido real, ya que es una matriz con números negativos y estos no se pueden representar con OpenCV. Se ha transformado a valores enteros sin sigo de 8 bits (uint8) para poder mostrar de alguna forma esta imagen.

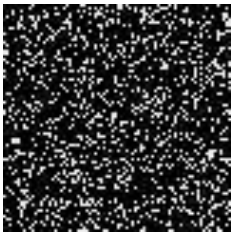

Fig. 50 Representación ruido aleatorio

En tercer lugar, se multiplica la máscara por el ruido. Con esto se consigue un ruido que no afecta al cuerpo del *C. elegans* ni al resto de blobs.

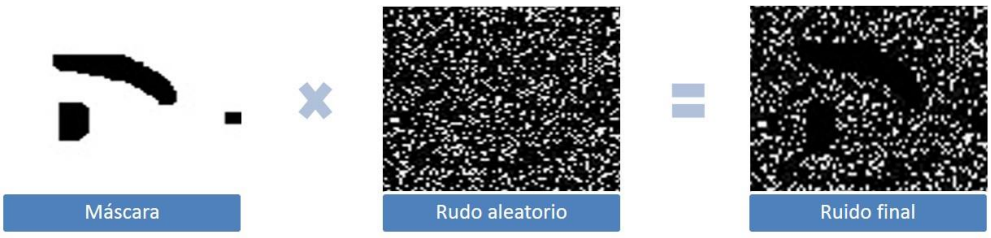

Fig. 51 Ruido aleatorio final

Por último, sumamos esta máscara generada a la imagen original obteniendo una imagen con ruido aleatorio en toda la imagen menos en los *blobs*.

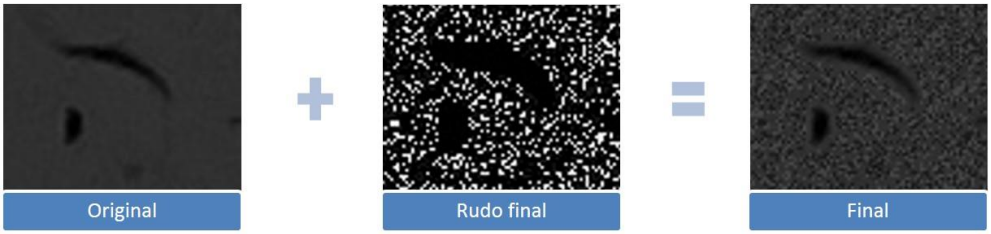

Fig. 52 Imagen final ruido aleatorio

5.2.2 Imágenes dibujando círculos en las ventanas

Este ruido introducido en las imágenes pretende simular los casos en los que aparecen otros elementos negros en la imagen que no son el *C. elegans*. Se ha hecho utilizando la función cv2.circle() de OpenCV. Para generar estas imágenes sin que el ruido solape con los *blobs* se han creado dos máscaras. Los pasos que se han seguido son los siguientes:

1. Se copia la imagen original.
2. Se dibuja un círculo negro en la imagen original.

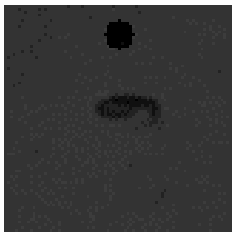

Fig. 53 Imagen original con círculo dibujado

3. Se realiza un *threshold* de la imagen original para segmentar el gusano del fondo

Fig. 54 Threshold imagen original

4. Se obtiene la negada de la imagen anterior y se convierte a 1 los valores mayores que 0 para emplear la matriz como máscara.

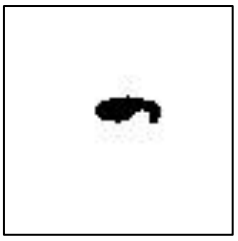

Fig. 55 Threshold inverso imagen original

5. Multiplicamos el *threshold* binarizado por la imagen original

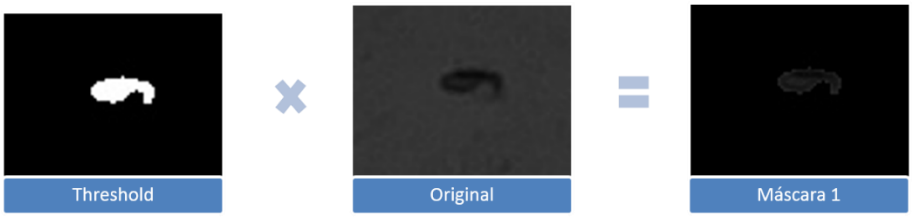

Fig. 56 Máscara 1

6. Multiplicamos la imagen con el círculo por el threshold inverso binarizado.

Fig. 57 Máscara 2

7. Se suman los resultados de los pasos 5 y 6, obteniendo imágenes con círculos situados en diferentes puntos sin afectar a los blobs.

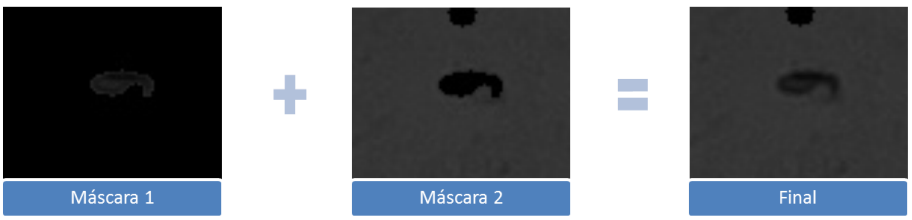

Fig. 58 Imagen final con círculos aleatorios

Este proceso se lleva a cabo en las 4 imágenes que forman la secuencia, situando estos círculos en distintos puntos del borde de la ventana de forma aleatoria. En la siguiente imagen se muestran los puntos posibles que se han programado:

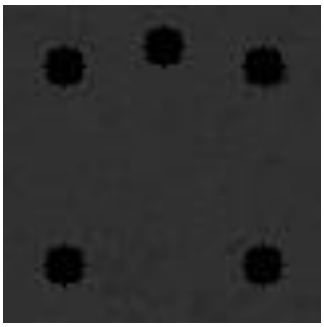

Fig. 59 Posibles círculos que se generan

Aunque este tipo de imágenes se han generado solo para el tipo secuencia, en esta memoria se muestran en el tipo mosaico para tenerlas de forma unida:

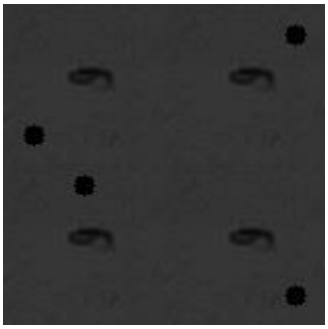

Fig. 60 Imagen dibujando círculos en las ventanas

Con estas imágenes, como se ha comentado, se pretende simular aquellos casos en los aparece un gusano muerto, pero el algoritmo lo clasifica de forma incorrecta como vivo debido a que aparece en la imagen otro elemento que no es el C. elegans moviéndose.

5.2.3 Entrenamiento con *dataset* aumentado

Tras aumentar el *dataset* de entrenamiento con las imágenes generadas artificialmente se han realizado entrenamientos de 10 épocas con este nuevo conjunto hasta que ha dejado de mejorar los resultados, validando con las mismas imágenes que se han empleado en el experimento 1.

Se ha obtenido la siguiente matriz de confusión:

	Secuencia CNN + LSTM	
	Predicción muerto	**Predicción vivo**
Etiqueta muerto	2514	38
Etiqueta vivo	82	2470

Tabla 12 Matriz de confusión de experimento con dataset aumentado

A partir de ella se calculan las métricas, que se recogen en la siguiente tabla:

	Precisión	Recall	F1 score
Muertos	0,968	0,985	0,977
Vivos	0,985	0,968	0,976

Tabla 13 Métricas dataset aumentado1

Y se representan gráficamente:

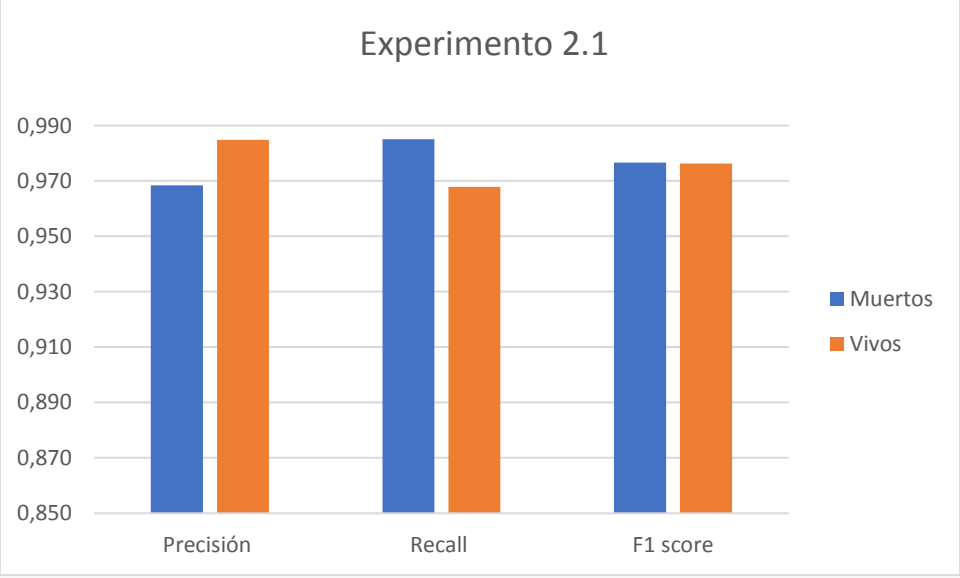

Fig. 61 Gráfica métricas dataset aumentado1

Como se observa en las métricas del experimento, los resultados han mejorado al introducir estas nuevas imágenes artificiales, llegando a un acierto del 97%.

Analizando los errores cometidos, se siguen observando errores al clasificar imágenes en las que aparecen elementos de ruido en la imagen.

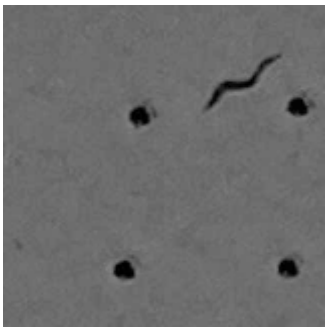

Fig. 62 Ejemplo de imagen original con ruido

Por tanto, se ha vuelto a aumentar el *dataset* empleando el método de los círculos negros, pero esta vez situándolos en una posición fija en lugar de aleatoria como en el caso anterior.

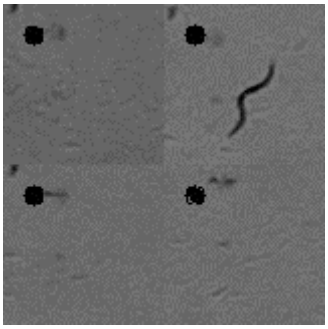

Fig. 63 Ejemplo de imagen artificial con ruido estático en las 4 imágenes

5.2.4 Entrenamiento con segundo aumento del *dataset*

De nuevo, tras aumentar el número de imágenes de entrenamiento se han realizado entrenamientos de 10 épocas hasta que se ha observado que los valores de precisión y pérdida han dejado de mejorar.

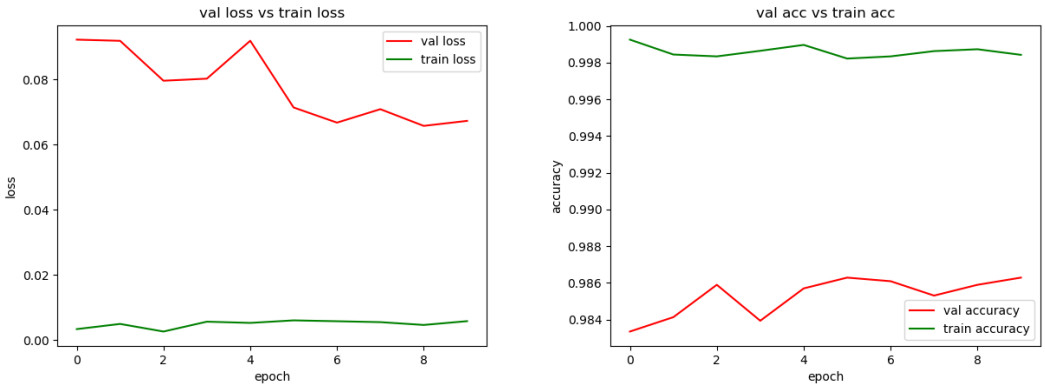

Fig. 64 Función de coste y precisión entrenamiento final

Tras esto, la matriz de confusión del modelo final es la siguiente:

	Secuencia CNN + LSTM	
	Predicción muerto	**Predicción vivo**
Etiqueta muerto	2528	24
Etiqueta vivo	46	2506

Tabla 14 Matriz de confusión final

A partir de ella se calculan las métricas, que se recogen en la siguiente tabla:

	Precisión	**Recall**	**F1 score**
Muertos	0,982	0,991	0,986
Vivos	0,991	0,982	0,986

Tabla 15 Métricas experimento final

De este experimento se extraen las siguientes conclusiones:

- Se ha observado que el hecho de aumentar el *dataset* con imágenes generadas artificialmente a partir de los fallos observados en experimentos anteriores mejora los resultados, por lo que se puede considerar en este caso una buena estrategia.
- El aumento del *dataset* ha permitido mejorar el clasificador, pasando de un *f1-score* medio del 93,6% en el primer experimento al 98,6% obtenido en el entrenamiento final.
- El algoritmo sigue teniendo mayor dificultad a la hora de identificar gusanos con cambios de postura mínimos (*Fig.43*) y en aquellas imágenes en las que el gusano se encuentra en el borde de la placa (*Fig.46*). Una de las mejoras futuras de este proyecto es la de generar artificialmente este tipo de imágenes, ya que se ha realizado alguna prueba, pero sin los resultados esperados.

6. Presupuesto

Para la realización del presupuesto se ha tenido en cuenta la carga académica del TFG, que corresponde a 12 ECTS. Cada ECTS equivale a 25 horas, por tanto, el número de horas de dedicación al trabajo asciende a 300 horas.

En el cálculo del coste de la mano de obra se ha tomado una estimación del salario de un graduado en ingeniería técnica industrial de 20 €/h. También se ha tenido en cuenta la labor de dirección y revisión del proyecto por parte del tutor, cuyo salario se ha estimado en 30 €/h.

6.1 Costes de mano de obra

En el cálculo de los costes de mano de obra se ha hecho un desglose aproximado del tiempo dedicado a las distintas tareas que se planificaron al comienzo del proyecto. En base a dichas horas y a las estimaciones de salario anteriormente comentadas se obtiene el siguiente desglose:

MANO DE OBRA				
Ud	**Descripción**	**Precio(€/h)**	**Cantidad(h)**	**Total(€)**
h	Planteamiento de alternativas y elección de la más adecuada	20	20	400
h	Formación e investigación	20	110	2200
h	Etiquetado	20	40	800
h	Diseño, programación e implementación de la solución	20	60	1200
h	Testeo y optimización	20	30	600
h	Redacción de la memoria	20	30	600
h	Revisión del documento	20	10	200
h	Dirección y revisión	30	20	600
SUBTOTAL MANO DE OBRA			**320**	**6600**

Tabla 16 Desglose costes mano de obra

6.2 Costes de equipamiento

En este apartado se incluyen los elementos de hardware y software empleados para la realización del proyecto.

En cuanto al software, no se ha tenido que realizar ningún gasto, ya que todas las herramientas empleadas son *open source*. A continuación, se enumeran dichos programas:

- El software Python.
- Librería Pytorch para las aplicaciones de *Deep Learning*.
- IDE PyCharm para el desarrollo del código.
- Librerías OpenCV y PIL para el procesamiento de imágenes.
- Licencias de Microsoft office, que son gratuitas al tener un convenio con la UPV.

En el apartado de hardware, se ha tenido en cuenta que el equipo empleado no se ha adquirido para este proyecto en concreto, sino que era el que se disponía en el laboratorio. Además, también se ha empleado un ordenador personal para la búsqueda de información y elaboración de la memoria y la presentación. Por tanto, se ha calculado su amortización durante el tiempo que se ha desarrollado el proyecto multiplicando el precio por el periodo de uso y dividiendo entre la vida útil.

$$C = \frac{P \times t}{T}$$

Ecuación 12 Cálculo de amortización de equipos

COSTES DE EQUIPAMIENTO						
Ud	**Descripción**	**Cantidad**	**Vida útil (años)**	**Periodo de uso (años)**	**Precio (€)**	**Coste (€)**
ud	PC lab	1	3	0,6	1500	300
ud	GPU lab	1	3	0,6	1000	200
ud	PC personal	1	3	0,6	500	100
SUBTOTAL COSTES DE EQUIPAMIENTO						**600**

Tabla 17 Desglose costes de equipamiento

6.3 Presupuesto total

En este apartado se recogen los gastos totales del proyecto. En primer lugar, se suman los subtotales de los costes de mano de obra y de equipamiento, obtenido así los gastos de ejecución material. A este valor, se le aplican los porcentajes correspondientes a gastos adicionales (13%) y beneficio industrial (6%), y se suman obteniendo el presupuesto de ejecución por contrata. Por último, se aplica el IVA (21%) y se suma obteniendo el presupuesto total del proyecto

COSTE DEL PROYECTO	
Concepto	**Importe (€)**
Mano de obra	6600
Costes de equipamiento	600
Presupuesto de ejecución material	7200
Gastos generales (13%)	936
Beneficio industrial (6%)	4320
Presupuesto de ejecución por contrata	12456
IVA (21%)	2615,76
TOTAL	**15071,76**

Tabla 18 Desglose presupuesto total

El coste total del proyecto asciende a **QUINCE MIL SETENTA Y UN EUROS CON SETENTA Y SEIS CÉNTIMOS.**

7. Conclusiones y mejoras futuras

7.1 Conclusiones

Este proyecto planteaba como objetivo diseñar un algoritmo que permitiera discernir si un nematodo *C. elegans* se encuentra vivo o muerto con una alta precisión.

En primer lugar, se ha realizado una revisión del estado del arte, encontrando proyectos en los que se emplean tanto técnicas tradicionales de visión por computador como una combinación de estas con *Machine Learning*. Todos estos proyectos presentan en común el uso de procesamiento de las imágenes para clasificar los nematodos. Analizando las ventajas y desventajas de resolver el problema mediante técnicas tradicionales o emplear *Deep Learning*, se ha optado por estas últimas considerando el equipo disponible, que permite entrenar redes neuronales en poco tiempo y el ahorro de tiempo que supone no tener que realizar una etapa de extracción de características.

En segundo lugar, se ha investigado el estado del arte en el uso de redes neuronales artificiales para la clasificación de imágenes, donde las redes neuronales convolucionales han demostrado ser una excelente opción. Sin embargo, nuestro caso no solo requería de un reconocimiento de las características de los *C. elegans*, sino también una comprensión de la dinámica temporal, ya que la clasificación se realiza en base a una secuencia de imágenes. Por ello, surgía la duda de si la red convolucional sería capaz de obtener los resultados esperados. Para solucionar este problema, se ha planteado como alternativa adicional el uso de las redes neuronales recurrentes, en concreto las LSTM, en combinación con las convolucionales.

Una vez definidas las arquitecturas que se querían probar, se ha realizado el etiquetado de las imágenes y se ha construido el *dataset*, generando distintos tipos de imagen en función de las características de cada tipo de red: imágenes en forma de mosaico de 4 ventanas y combinación de 3 canales RGB para la red convolucional, y una secuencia de 4 imágenes para la red LSTM.

Tras conformar el *dataset* completo, se ha dividido en carpetas de entrenamiento y validación, con un 70 % de las imágenes para entrenar y un 30% para evaluar el modelo. Las carpetas se han balanceado para que presenten el mismo número de imágenes de cada clase (vivos y muertos) para garantizar que el algoritmo aprende de forma correcta.

A continuación, se han realizado diferentes pruebas con las distintas arquitecturas y tipos de imágenes obteniendo como principales conclusiones:

- Como era de esperar, la combinación de redes convolucionales con LSTM obtiene los mejores resultados.
- El uso de modelos pre-entrenados permite obtener buenos resultados en poco tiempo.
- Con un *dataset* reducido es posible obtener buenos resultados, pero insuficientes. Cuanto mayor es el número de muestras, mejores resultados se obtienen.

Como fase final del proyecto, se ha buscado optimizar los resultados obtenidos centrándonos en la arquitectura que combina redes convolucionales con recurrentes, ya que es la que mejores resultados obtiene.

Para llevar a cabo esta optimización, se ha realizado un análisis de las imágenes que más le costaba clasificar al algoritmo diseñado y se ha aumentado el *dataset* con imágenes generadas artificialmente. Este aumento de imágenes ha permitido al algoritmo pasar de una precisión del 93,6 % a un 98,6%.

En síntesis, se ha conseguido diseñar y desarrollar un clasificador capaz de discernir entre *C. elegans* vivos y muertos con una alta precisión, objetivo principal de este proyecto final de grado.

7.2 Mejoras futuras

En el desarrollo de este proyecto se han planteado diversas alternativas y pruebas, sin embargo, dada la extensión del trabajo, han quedado algunas mejoras pendientes que se pueden implementar en un futuro. Algunas de ellas son:

- Modificar la forma de introducir las imágenes, empleando secuencias de imágenes más largas.
- Emplear un método automático de generación de nuevas imágenes para aumentar el *dataset*, evitando de esta forma tener que etiquetar. Esto se puede hacer de diversas maneras: una opción es aplicar transformaciones a las imágenes que ya se disponen, tal y como se ha realizado en este proyecto. Otra alternativa, es crear un simulador de *C. elegans* y generar imágenes nuevas a partir de él.
- En la arquitectura de las redes neuronales convolucionales se ha empleado una red pre-entrenada (resnet18) con 18 capas de profundidad, se podría emplear una arquitectura superior para tratar de mejorar los resultados, como por ejemplo resnet50.
- En el método que combina redes neuronales convolucionales con la LSTM se podría emplear una red convolucional pre-entrenada, de esta forma no se partiría de cero al entrenar.
- Desarrollar una interfaz de usuario que integre el modelo para crear un entorno sencillo e intuitivo que permita el uso del algoritmo desarrollado.

Bibliografía

[1] A. Bansal, L. J. Zhu, K. Yen, y H. A. Tissenbaum, «Uncoupling lifespan and healthspan in *Caenorhabditis elegans* longevity mutants», *Proceedings of the National Academy of Sciences*, vol. 112, n.º 3, pp. E277-E286, ene. 2015.

[2] T. C. elegans S. Consortium*, «Genome Sequence of the Nematode C. elegans: A Platform for Investigating Biology», *Science*, vol. 282, n.º 5396, pp. 2012-2018, dic. 1998.

[3] A. Olsen, *Ageing: lessons from C. Elegans*. New York, NY: Springer Berlin Heidelberg, 2016.

[4] J. V. Benlloch, M. Agusti, A. Sanchez, y A. Rodas, «Colour segmentation techniques for detecting weed patches in cereal crops», en *Proc. of Fourth Workshop on Robotics in Agriculture and the Food-Industry*, 1995, pp. 30–31.

[5] A. J. Sánchez, W. Albarracin, R. Grau, C. Ricolfe, y J. M. Barat, «Control of ham salting by using image segmentation», *Food Control*, vol. 19, n.º 2, pp. 135-142, feb. 2008.

[6] R. Grau, A.-J. Sánchez-Salmerón, J. Girón, E. Ivorra, A. Fuentes, y J. Barat, «Nondestructive assessment of freshness in packaged sliced chicken breasts using SW-NIR spectroscopy», *Food Research International*, vol. 44, pp. 331-337, ene. 2011.

[7] E. Ivorra, A.-J. Sánchez-Salmerón, S. Amat, J. Barat, y R. Grau, «Shelf life prediction of expired vacuum-packed chilled smoked salmon based on a KNN tissue segmentation method using hyperspectral images», *Journal of Food Engineering*, vol. 178, ene. 2016.

[8] C. Ricolfe-Viala y A. Sanchez-Salmeron, «Optimal conditions for camera calibration using a planar template», en *2011 18th IEEE International Conference on Image Processing*, 2011, pp. 853-856.

[9] «Continuous monitoring of bread dough fermentation using a 3D vision Structured Light technique | Elsevier Enhanced Reader». [En línea]. Disponible en: https://reader.elsevier.com/reader/sd/pii/S0260877414000119?token=E455E11711AF4AF2D6663C592A9559CA8542B40FDE5789AEA5835051614ED8D7974E6E7D333AE13D143D96513FEC9C6C. [Accedido: 29-jun-2019].

[10] E. Ivorra, A. J. Sánchez, J. G. Camarasa, M. P. Diago, y J. Tardaguila, «Assessment of grape cluster yield components based on 3D descriptors using stereo vision», *Food Control*, vol. 50, pp. 273-282, abr. 2015.

[11] S. Amat, E. Ivorra, A.-J. Sánchez-Salmerón, J. Barat, y R. Grau, «Relationship between fermentation behavior, measured with a 3D vision Structured Light technique, and the internal structure of bread», *Journal of Food Engineering*, vol. 146, oct. 2014.

[12] E. Berti, A.-J. Sánchez-Salmerón, y F. Benimeli, «Kalman Filter for Tracking Robotic Arms Using low cost 3D Vision Systems», presentado en ACHI 2012 - 5th International Conference on Advances in Computer-Human Interactions, 2012.

[13] E. Martínez Bertí, A. J. Sánchez Salmerón, y F. Benimeli, «Human robot interaction and tracking using low cost 3D vision systems», en *Romanian Journal of Technical Sciences - Applied Mechanics*, 2012, vol. 7, pp. 151-168.

[14] E. Martinez-Berti, I. Ai, A. J. Snchez-Salmern, y C. Ricolfe-Viala, «Human Pose Estimation for RGBD Imagery with Multi-Channel Mixture of Parts and Kinematic Constraints», vol. 15, p. 8, 2016.

[15] E. Berti, A.-J. Sánchez-Salmerón, y C. Viala, «4-Dimensional deformation part model for pose estimation using Kalman filter constraints», *International Journal of Advanced Robotic Systems*, vol. 14, p. 172988141771423, may 2017.

[16] E. Berti, O. Nina, A.-J. Sánchez-Salmerón, y C. Viala, «Optimized 4D DPM for Pose Estimation on RGBD Channels using Polisphere Models», 2017, pp. 281-288.

[17] E. Berti, A.-J. Sánchez-Salmerón, y C. Ricolfe-Viala, «Dual Quaternions as Constraints in 4D-DPM Models for Pose Estimation», *Sensors*, vol. 17, p. 1913, ago. 2017.

[18] M. Bosch-Jorge, A.-J. Sánchez-Salmerón, y C. Ricolfe-Viala, «Visual-based human action recognition on smart phones based on 2d and 3d descriptors», *Int. J. Patt. Recogn. Artif. Intell.*, vol. 26, n.º 08, p. 1260009, nov. 2012.

[19] M. Bosch-Jorge, A.-J. Sánchez-Salmerón, Á. Valera, y C. Ricolfe-Viala, «Fall detection based on the gravity vector using a wide-angle camera», *Expert Systems with Applications*, vol. 41, n.º 17, pp. 7980-7986, dic. 2014.
[20] «Biology Professor Javier Apfeld awarded NSF CAREER grant for early career faculty», *Tim Briggs Photography*. [En línea]. Disponible en: https://timbriggsphoto.com/northeastern-cos/2018/9/7/biology-professor-javier-apfeld-awarded-nsf-career-grant-for-early-career-faculty. [Accedido: 23-jun-2019].
[21] A. Sánchez y C. Ramos, «SEGUIMIENTO VISUAL DE OBJETOS UTILIZANDO TÉCNICAS DE PREDICCIÓN», p. 6.
[22] A.-J. Sánchez-Salmerón y J. Marchant, «Fast and robust method for tracking crop rows using a two point Hough transform», 1997.
[23] 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.
[24] «Welcome to Python.org», *Python.org*. [En línea]. Disponible en: https://www.python.org/about/. [Accedido: 11-jun-2019].
[25] A. Hakim *et al.*, «WorMachine: machine learning-based phenotypic analysis tool for worms», *BMC Biol*, vol. 16, n.º 1, p. 8, ene. 2018.
[26] N. Stroustrup, B. E. Ulmschneider, Z. M. Nash, I. F. López Moyado, J. Apfeld, y W. Fontana, «The C. elegans Lifespan Machine», *Nat Methods*, vol. 10, n.º 7, pp. 665-670, jul. 2013.
[27] T. Puckering, J. Thompson, S. Sathyamurthy, S. Sukumar, T. Shapira, y P. Ebert, «Automated Wormscan», *F1000Res*, vol. 6, p. 192, ene. 2019.
[28] J. Walsh *et al.*, *Deep Learning vs. Traditional Computer Vision*. 2019.
[29] Crisvill, «Avances en redes neuronales - Medium en español», *Medium*, 23-feb-2017. [En línea]. Disponible en: https://medium.com/espanol/avances-en-redes-neuronales-705c2efe53d2. [Accedido: 06-jul-2019].
[30] «Neural Network Models in R», *DataCamp Community*, 18-ene-2019. [En línea]. Disponible en: https://www.datacamp.com/community/tutorials/neural-network-models-r. [Accedido: 06-jul-2019].
[31] Y. LeCun, Y. Bengio, y G. Hinton, «Deep learning», *Nature*, vol. 521, n.º 7553, pp. 436-444, may 2015.
[32] «conv-net2.png (2762×944)». [En línea]. Disponible en: https://flickrcode.files.wordpress.com/2014/10/conv-net2.png. [Accedido: 12-jun-2019].
[33] Y. Bengio, P. Simard, y P. Frasconi, «Learning long-term dependencies with gradient descent is difficult», *IEEE Transactions on Neural Networks*, vol. 5, n.º 2, pp. 157-166, mar. 1994.
[34] «Understanding LSTM Networks -- colah's blog». [En línea]. Disponible en: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accedido: 13-jun-2019].
[35] S. Hochreiter y J. Schmidhuber, «Long short-term memory», *Neural Comput*, vol. 9, n.º 8, pp. 1735-1780, nov. 1997.
[36] K. He, X. Zhang, S. Ren, y J. Sun, «Deep Residual Learning for Image Recognition», *arXiv:1512.03385 [cs]*, dic. 2015.
[37] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, y L. Fei-Fei, «ImageNet: A Large-Scale Hierarchical Image Database», p. 8.
[38] E. Gutierrez Alegre, M. Pajares, y A. de la Escalera Hueso, *Conceptos y métodos en visión por computador*. España: s.l., 2016.
[39] «Transfer Learning Tutorial — PyTorch Tutorials 1.1.0 documentation». [En línea]. Disponible en: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html. [Accedido: 23-jun-2019].