UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DISEÑO, IMPLEMENTACIÓN Y EVALUACIÓN DE UNA NUEVA ESTRATEGIA DE APRENDIZAJE PARA REDES NEURONALES CONVOLUCIONALES DE TRANSFORMACIÓN ESPACIAL DE IMÁGENES (STN'S)

AUTOR: FRANCISCO NAVARRO MOYA

TUTOR: ANTONIO JOSÉ SÁNCHEZ SALMERÓN

Curso Académico: 2020-21

AGRADECIMIENTOS

Quiero agradecer a mi tutor Antonio su inestimable ayuda para la consecución de este trabajo final de máster, así como a los compañeros del Instituto de Automática e Informática Industrial (ai2) Joan, Pablo y Antonio por facilitarme la comprensión de todos los aspectos de un mundo tan complejo como las redes neuronales convolucionales.

También agradecer a mi familia, mi padre, mi hermana y mi hermano por su apoyo para la consecución de todas las metas que me he propuesto.

Pero en especial me gustaría dedicárselo a mi madre, gracias por siempre servirme de apoyo cuando lo necesitaba, gracias por inculcarme tus valores, gracias por educarme, gracias por convertirme en la persona que soy, gracias por estar siempre ahí, aunque ya no estés.

Gracias, mamá.

RESUMEN

Este trabajo consistirá en diseñar, implementar y evaluar diferentes métodos de convergencia para el desempeño de redes neuronales convolucionales de transformación espacial, en este caso trabajando sobre imágenes de nematodos (*Caenorhabditis elegans*).

Inicialmente, el trabajo se centrará en el estudio y la comprensión de este tipo de redes neuronales de forma que se puedan plantear las diferentes estrategias a seguir. Para su evaluación, se contará con un dataset de parejas de imágenes de *C. elegans*, capturadas mediante dos cámaras, y el objetivo principal de los ensayos será transformar una de las imágenes, en la cual el nematodo no aparece centrado, en la otra, en la cual sí lo estará.

Para ello, se empleará la herramienta PyCharm como medio en el cual realizar los ensayos. Dicha herramienta emplea Python como lenguaje de programación, y mediante la librería de funciones de Pytorch junto a otras librerías típicas de Python se diseñarán tanto las redes neuronales como los diferentes métodos de convergencia que se van a evaluar.

Finalmente, para la evaluación de las propuestas se emplearán diversos criterios entre los que estarán la tasa de acierto o los costes temporales de las ejecuciones. Además, se plantearán diversas aplicaciones en las cuales puedan emplearse los resultados aportados por este estudio.

Palabras Clave: procesamiento de imágenes, aprendizaje profundo, redes neuronales convolucionales, *C. elegans*, Python, Pytorch

RESUM

Aquest treball consistirà a dissenyar, implementar i avaluar diferents mètodes de convergència per a l'acompliment de xarxes neuronals convolucionals de transformació espacial, en aquest cas treballant sobre imatges de cucs (*Caenorhabditis elegans*).

Inicialment, el treball se centrarà en l'estudi i la comprensió d'aquesta mena de xarxes neuronals de manera que es puguen plantejar les diferents estratègies a seguir. Per a la seua avaluació, es comptarà amb un dataset de parelles d'imatges de *C. elegans*, capturades mitjançant dues cambres, i l'objectiu principal dels assajos serà transformar una de les imatges, en la qual el cuc no apareix centrat, en l'altra, en la qual sí que ho estarà.

Per a això, s'emprarà l'eina PyCharm com a mitjà en el qual realitzar els assajos. Aquesta eina empra Python com a llenguatge de programació, i mitjançant la llibreria de funcions de Pytorch al costat d'altres llibreries típiques de Python es dissenyaran tant les xarxes neuronals com els diferents mètodes de convergència que s'avaluaran.

Finalment, per a l'avaluació de les propostes s'empraran diversos criteris entre els quals estaran la taxa d'encert o els costos temporals de les execucions. A més, es plantejaran diverses aplicacions en les quals puguen emprar-se els resultats aportats per aquest estudi.

Paraules clau: processament d'imatges, aprenentatge profund, xarxes neuronals convolucionals, *C. elegans*, Python, Pytorch

ABSTRACT

This work will consist of designing, implementing and evaluating different convergence methods for the performance of spatial transform convolutional neural networks, in this case working on images of worms (*Caenorhabditis elegans*).

Initially, this work will focus on the study and understanding of this type of convolutional neural network so that can be proposed the different strategies to be followed. For its evaluation, there will be a dataset of *C. elegans* images pairs, captures by two cameras, and the main objective of the essays will be to transform one of these images, in which the worm does not appear in the middle of the image, into the other one, where it will be there.

For this, the PyCharm tool will be used as the means in which to carry out the essays. This tool uses Python as its programming language, through the Pytorch function library together whit others typical Python libraries, both the convolutional neural networks and the different convergence methods will be designed.

Finally, for the evaluation of the proposed methods, several index will be used, among which will be the success rate or the computational costs. In addition, several applications will be proposed in which the results provided for this study can be used.

Keywords: image processing, deep learning, convolutional neural networks, *C. elegans*, Python, Pytorch

ÍNDICE

Memoria

Presupuesto

Anexos

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Parte I

Memoria

CAPÍTULO 1. INTRODUCCIÓN

1.1. Marco del trabajo

Este trabajo final de máster busca ampliar los conocimientos adquiridos durante la realización del Máster Universitario de Ingeniería Industrial con especialidad en automática, robótica y control de procesos. Este trabajo puede situarse dentro del ámbito de la inteligencia artificial, y más concretamente en el de la visión artificial, las redes neuronales convolucionales y el "Deep learning".

Este es un campo en auge en los últimos años debido a su gran potencial para multitud de tareas, tales como la re-identificación de personas, la clasificación de objetos, la detección de objetos, el reconocimiento de texto escrito o la traducción del mismo entre otras.

Más concretamente lo que se busca en este trabajo es dar solución al problema clásico de correspondencia, el cual consiste en obtener la relación entre los pixeles de un mismo objeto de interés que aparece en dos imágenes diferentes con puntos de vista distintos. Para ello se debe obtener la homografía (o matriz de proyección) entre los pixeles de las dos imágenes, esta matriz sirve, posteriormente, para realizar la transformación de forma que se pueda obtener una de las imágenes a partir de la otra.

Tradicionalmente esta clase de problemas se ha solucionado con el cálculo de una homografía, tomando como referencia una serie de puntos de ambas imágenes, los cuales deben de corresponderse. Este tipo de solución es muy efectiva, pero puede presentar errores de emparejamiento automático de los puntos o ser muy costosa temporalmente, en el caso de que los puntos se seleccionen de forma manual, con la consiguiente introducción de error por falta de precisión en su selección.

Por todo ello, se optó por recurrir a las redes neuronales convolucionales para darle solución, ya que estas son un mecanismo muy potente y no es necesario determinar ningún punto sobre el que basar la transformación, sino que conociendo la posición inicial y final del objeto es más que suficiente.

En la resolución de este problema se ha contado como objeto de trabajo con imágenes de *C. elegans* en las que estos aparecen desde diferentes puntos de vista. Los *C. elegans* [O.1] (Figura 1) son pequeños nematodos ampliamente utilizados en el ámbito de la investigación científica debido a las propiedades tan excepcionales que estos poseen.

En primer lugar, de ellos se conoce la secuenciación completa de su genoma desde 1998, su sistema nervioso y todos sus órganos. Además, los *C. elegans* son transparentes, lo cual permite que se puedan observar fácilmente sus procesos biológicos internos con cámaras de alta resolución o a través de un microscopio.

Figura 1. Imagen de un C. elegans [I.1]

Por otro lado, estos organismos son hermafroditas lo que facilita su cría y el mantenimiento en los laboratorios de investigación. Aparte de esto su ciclo de vida está comprendido entre 2 y 3 semanas, lo cual permite realizar experimentos de ciclo de vida completo en muy pocas semanas.

Entre los experimentos más habituales que se desarrollan sobre estos nematodos destacan el ensayo de "Healthspan" [A.1] [A.2] [A.3] y el ensayo de "Lifespan" [A.4] [A.5] [A.6] [A.7]. En ambos tipos de experimentos se busca caracterizar el comportamiento de los *C. elegans* en función de su alimentación.

El primero de ellos, "Healthspan" se trabaja con *C. elegans* en los que se han introducido determinadas modificaciones genéticas que sirven de modelos de determinadas enfermedades, como, por ejemplo, enfermedades neurodegenerativas (Parkinson, Alzheimer…), y estudiar la respuesta de los *C. elegans* hacia el tratamiento con determinados fármacos.

En cuanto al estudio de "Lifespan", en él se busca caracterizar las curvas de su esperanza de vida y ver como esta se modifica frente a determinados compuestos.

1.2. Motivación

Esta investigación surgió de la necesidad de transformar imágenes de nematodos (*C. elegans*) en las cuales estos se encuentran desplazados de la posición central de la imagen para conseguir que ocupen dicha posición, con la finalidad de poder emplear estas imágenes para otras tareas como son el problema de re-identificación de los nematodos o el problema de enfoque de las imágenes.

Planteada esta necesidad se optó por recurrir a la red neuronal de transformación espacial (*"Spatial Transform Network"* STN) [A.8] para solventar dicha problemática. Este tipo de redes trabajan con una imagen de entrada en la que está el objeto que se quiere transformar y una imagen deseada u objetivo que marca lo que se quiere conseguir.

De forma habitual estas imágenes se encuentran segmentadas ya que de esta forma es más fácil centrarse en obtener la transformación necesaria para colocar el objeto de interés en la posición

deseada. El problema aquí planteado reside en que los objetos de interés que se quieren emparejar en este proyecto (*C. elegans*) son muy pequeños con respecto al tamaño de la imagen, y, por tanto, el grado de solapamiento inicial entre las dos segmentaciones de las dos imágenes es bastante reducido o incluso nulo en algunos casos.

Durante los primeros ensayos realizados este último hecho ha ocasionado problemas de convergencia al no disponer de información inicial suficiente. Por ello, se abrió esta investigación en busca de una nueva metodología de entrenamiento más robusta que permita afianzar la convergencia en estos casos. Además, aparte de afianzar la convergencia se ha intentado mantener el nivel de precisión obtenido en los ensayos realizados con las imágenes segmentadas, en los cuales si se conseguía la convergencia, y controlar que los tiempos de entrenamiento de la red no se elevasen demasiado, ya que esto puede ser un factor determinante para muchas problemáticas.

1.3. Objetivos del trabajo

El objetivo principal del proyecto es hallar una nueva metodología de entrenamiento para solucionar el problema de correspondencia, con la cual se consiga afianzar la convergencia en casos en los que el solapamiento entre el objeto de interés de la imagen de entrada a la red y el de la imagen deseada es bastante reducido, e incluso también conseguirlo para aquellos casos límites en los que no exista dicho solapamiento.

Más allá de este objetivo principal o final existen diversos objetivos secundarios o intermedios que se han ido desarrollando durante la investigación. Estos son:

- El aprendizaje sobre una nueva sección de la visión artificial como son las redes neuronales convolucionales y el "Deep learning".
- La ampliación de los conocimientos sobre el lenguaje de programación de Python.
- El manejo de la librería de funciones de PyTorch.
- La comprensión del funcionamiento de las redes neuronales de transformación espacial (STN).
- Desarrollar la capacidad para poder diseñar y optimizar una red neuronal convolucional.
- Aprender a entrenar e interpretar los resultados de una red neuronal convolucional.

1.4. Estructura del documento

La memoria de este documento se encuentra dividida en capítulos en los cuales se tratarán los diferentes aspectos sobre los que trata el trabajo, desde una breve introducción a las redes neuronales hasta la exposición de los resultados con las conclusiones que se han obtenido de ellos.

Como se ha comentado el documento comienza haciendo una introducción al mundo de las redes neuronales definiendo sus aspectos más importantes, así como los principales elementos que las componen. Además, también se introducen las redes neuronales convoluciones de transformación espacial (STN).

Más adelante, se explica el sistema de captura mediante el cual se han obtenido las imágenes que confeccionarán el dataset y también se define la configuración de un simulador que se ha empleado

para disponer de un dataset sintético con el que realizar ensayos, desde un punto de vista más controlado.

Posteriormente se expone cual ha sido la configuración para los dataset utilizados, así como la estructura y los parámetros de la red empleada para la resolución de la tarea.

A continuación, se explican las diferentes metodologías de entrenamiento que se han testeado y se presentan los resultados obtenidos para cada una de ellas seguido de la exposición de dos problemáticas en las que se pueden aplicar los resultados obtenidos por la solución escogida.

Por último, se presentan las conclusiones obtenidas de todos estos ensayos, tanto de la STN como de sus aplicaciones, y se plantean algunas mejoras para todos ellos como trabajos futuros para ampliar el trabajo realizado.

CAPÍTULO 2. REDES NEURONALES CONVOLUCIONALES

En este capítulo se definen las redes neuronales convolucionales, así como se exponen y explican cada uno de sus componentes más habituales y de los elementos adicionales necesarios para su funcionamiento [A.9]. Este apartado va orientado a facilitar la comprensión de todo el documento.

También se realiza una breve introducción a las redes neuronales de transformación espacial (STN) exponiendo diversos trabajos en los que se han empleado y definiendo su estructura básica.

2.1. ¿Qué es una red neuronal convolucional (CNN)?

Las redes neuronales convolucionales son un tipo de redes neuronales artificiales las cuales buscan imitar el funcionamiento de las neuronas conectadas a las retinas de los humanos y de los animales.

Este tipo de redes tienen como base el perceptrón diseñado por Frank Rosenblatt [A.10], el cual, frente a un conjunto de entradas mediante la definición de unos pesos de entrada, los cuales se calculan por retropropagación, es capaz de discernir cuales de las entradas cumplen con una determinada condición o característica.

Las redes neuronales artificiales están formadas por un conjunto de estos perceptrones interconectados generando una estructura multicapa mucho más compleja y profunda, en la cual cada perceptrón puede considerarse como una de las neuronas de la red, como se muestra en la Figura 2. De esta forma las primeras capas son capaces de extraer las características de más bajo nivel mientras que las más profundas se encargan de extraer características más complejas.

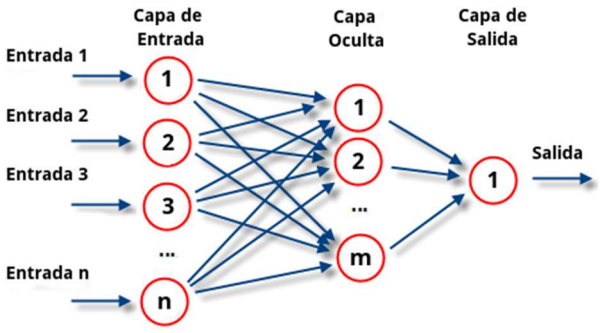


Figura 2. Esquema funcionamiento red neuronal artificial [I.2]

En las redes neuronales convolucionales se substituyen estas neuronas formadas por los perceptrones por un mecanismo de procesamiento matricial, un esquema del funcionamiento de esta nueva estructura puede observarse en la Figura 3. De esta forma se pasa de trabajar con entradas

unidimensionales a poder hacerlo con información en dos dimensiones como son las imágenes, por ello este tipo de redes son ampliamente utilizadas para resolver problemas de visión artificial.

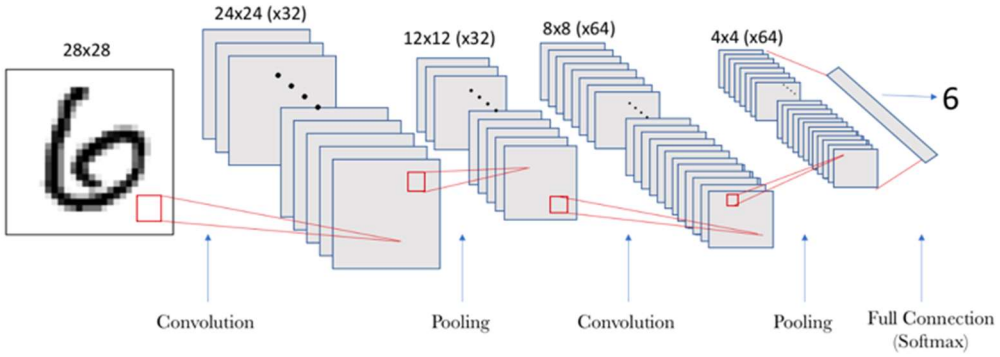


Figura 3. Esquema funcionamiento red neuronal convolucional [I.3]

2.2. Funcionamiento de una red neuronal convolucional

Una vez definida la estructura básica de una red neuronal convolucional es necesario explicar cuál es el procedimiento para trabajar con ella.

Una red neuronal convolucional trabaja con un dataset o conjunto de datos, que, habitualmente, se encuentra formado por un conjunto de entradas y el conjunto de etiquetas correspondientes, el objetivo de la red es el de, para cada una de las entradas, predecir una salida lo más parecida posible a su etiqueta asociada.

Para ello, la red dispone de millones de pesos los cuales se aplican sobre la entrada a lo largo de todas las capas que la componen hasta la obtención de la salida de la red. Pero en primera instancia estos pesos no son los idóneos para resolver el problema en cuestión, por lo que estos deben de ajustarse.

Es aquí donde entra el método de aprendizaje de una red neuronal convolucional, dicho proceso se puede dividir en tres fases, entrenamiento, validación y test. El dataset también se divide normalmente en tres para no emplear las mismas imágenes para entrenar, para validar y para test.

Durante la etapa de entrenamiento se evalúan las salidas de la red en comparación con las etiquetas, y en función de la semejanza entre estas dos se realiza la actualización de los pesos mediante retropropagación. Este proceso se repite de forma iterativa en busca del óptimo, el cual minimiza las diferencias entre las salidas de la red y las etiquetas.

Por otro lado, la etapa de validación se utiliza para ajustar los hiperparámetros de la fase de entrenamiento. Y, finalmente, la fase de test garantiza que los resultados obtenidos durante la etapa de aprendizaje (entrenamiento y validación) son aplicables sobre datos que no se han utilizado durante esta.

2.3. Componentes de una red neuronal convolucional

Una red neuronal convolucional puede dividirse en multitud de capas, las cuales determinan el tamaño y la profundidad de la red. Existen varios tipos de estas capas, los cuales se diferencian según la funcionalidad que tengan dentro de la red, es decir, de cómo modifiquen los datos a su entrada. A continuación, se explicarán los tipos más comunes, que son los siguientes:

- Capas convolucionales
- Capas de reducción
- Capas de activación
- Capas completamente conectadas

2.3.1. Capas convolucionales

Este tipo de capa es el componente base de las redes neuronales convolucionales, en ella se busca obtener las características de un grupo de pixeles vecinos de las imágenes. Para ello, se emplea un kernel o ventana (normalmente de tamaño 3x3, 5x5 o 7x7) que recorre la imagen y relaciona el valor de una neurona de una capa con un grupo de pixeles vecinos de la capa anterior en función de los pesos del kernel (Figura 4).

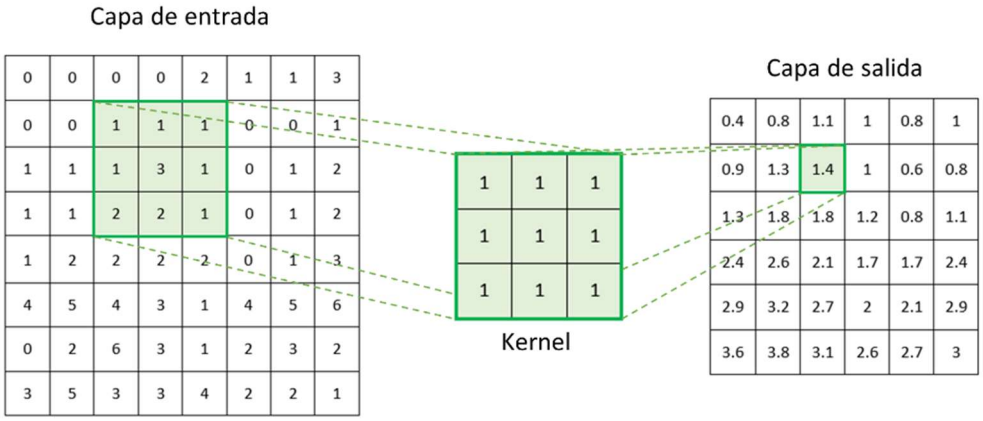


Figura 4. Ejemplo del funcionamiento de una capa convolucional

Durante la aplicación de este tipo de capas también se suele reducir el tamaño de la imagen al no poder aplicar el kernel a los pixeles de los bordes, aunque existe la posibilidad de ampliar la información de la capa de entrada con pixeles vacíos (padding) para que no se produzca esta reducción manteniendo el tamaño de entrada.

2.3.2. Capas de reducción

El cometido de esta tipología de capas es, como su propio nombre indica, el de reducir la cantidad de información presente a la salida de la capa, para ello, al igual que en las capas convolucionales, se emplea una ventana o kernel que recorre la imagen. Las capas de reducción suelen situarse entre capas convolucionales.

Este tipo de capas suele emplearse para simplificar el problema con el que se trabaja manteniendo solo la información más representativa de cada conjunto de datos.

Existen diferentes tipos de capas de reducción, entre las que destacan la capa de "average pooling" [A.11], la cual se queda con una media de los valores, y la capa de "max pooling" [A.12], que conserva el máximo valor de la ventana muestreada. En la Figura 5 se expone un ejemplo del funcionamiento de una capa "maxpooling".

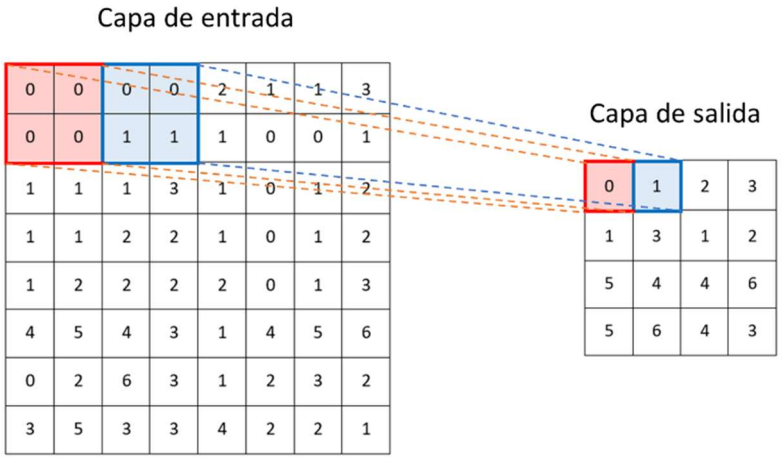
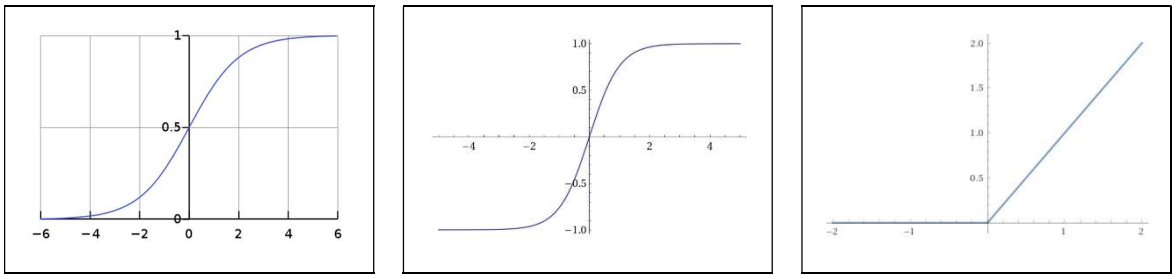


Figura 5. Ejemplo del funcionamiento de una capa de maxpooling

2.3.3. Capas de activación

Con estas capas lo que se busca es adaptar los resultados de una capa convolucional introduciendo filtros no lineales que transforman los valores de las neuronas de la capa anterior. Entre estas capas las más conocidas son las que emplean la función de activación "Sigmoid", la función de activación "Tanh" [A.13] y la función de activación "ReLU" [A.14].



Sigmoid Tanh ReLU

Figura 6. Gráficas funcionamiento de las capas de activación Sigmoid [I.4], Tanh [I.5] y ReLU [I.6]

En la Figura 6 se exponen las gráficas de las diferentes funciones de activación donde se puede observar que tanto "Sigmoid" como "Tanh" son funciones que normalizan entre dos valores, [0, 1] en el caso de "Sigmoid y [-1, 1] en el caso de "Tanh", mientras que la función de activación "ReLU" solamente elimina los valores negativos.

De esta última función existen muchas variantes que modifican ligeramente su funcionamiento ante las entradas negativas, entre ellas destacan "Leaky ReLU" [A.15] la cual permite salidas negativas, pero estas se encuentran amortiguadas, es decir la pendiente de la recta en la zona negativa es menor que la unidad, o "SELU" [A.16] la cual sigue la trayectoria marcada por una exponencial que satura negativamente a un valor menor que 0 (ecuación 1).

$$SELU(x) = \max(0, x) + \min(\lambda(e^x - 1)) \qquad (1)$$

Donde "λ" es el valor al que satura negativamente y "x" es el valor para el que se está calculando la respuesta de la función de activación "SELU".

2.3.4. Capas completamente conectadas

Este tipo de capas suelen ocupar los últimos niveles de la red neuronal convolucional. Para su funcionamiento destruyen la estructura bidimensional empleada en las capas anteriores, conectando entre si todas las neuronas de la capa convolucional anterior, generando de esta forma una nueva estructura unidimensional. Por ello, su funcionamiento es idéntico al de un perceptrón multicapa.

2.4. Hiperparámetros

En este apartado se exponen algunos hiperparámetros o elementos necesarios sin los que la red no sería capaz de desempeñar correctamente su funcionamiento. Estos elementos son:

- La función de pérdidas
- El optimizador
- Las funciones de regularización

2.4.1. Función de pérdidas (loss function)

Las funciones de pérdida son uno de los elementos clave para que una red neuronal pueda conseguir buenos resultados. Su objetivo es evaluar la respuesta de la red en comparación con el objetivo o etiqueta que se desea conseguir, y en función del resultado de esta comparación actualizar los pesos y parámetros de la red neuronal para conseguir que la respuesta obtenida con esta nueva configuración se asemeje más al objetivo deseado.

Existen numerosos tipos de funciones de pérdidas en función del tipo de comparación que se quiera realizar, por ejemplo, no es lo mismo comparar imágenes píxel a píxel para determinar su semejanza que comparar únicamente la identidad de los objetos para definir a que clase pertenecen.

Entre las funciones de pérdidas más empleadas destaca "Mean Square Error" (MSE loss), cuya fórmula se muestra en la ecuación 2. Dicha función calcula la media de las distancias euclídeas entre los valores de todos los pixeles de las dos imágenes (imagen devuelta por la red e imagen objetivo), de forma que un valor bajo de esta función significaría que las dos imágenes son semejantes mientras que un valor alto indicaría que son distintas.

$$MSE\ Loss = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m}(X_{(n,m)} - Y_{(n,m)})^2}{m * n} \qquad (2)$$

Siendo "n" y "m" el tamaño de las imágenes en cada uno de los ejes, y "$X_{(n,m)}$" e "$Y_{(n,m)}$" los valores de los pixeles de la imagen devuelta por la red y de la imagen objetivo respectivamente.

Otras funciones de pérdidas muy empleadas pueden ser "Contrastive loss" [A.17] o "Softmax loss" [A.18] para el caso de clasificación de la imagen de entrada o "Triplet loss" [A.19] para el caso de reidentificación de personas u objetos. Aparte de todas estas funciones de pérdidas ya predefinidas cabe la posibilidad de diseñar una función de pérdidas customizada si el caso de estudio lo requiere.

2.4.2. Optimizador

El optimizador es el encargado de ejecutar la actualización de los pesos de todas las capas de la red, de forma habitual esto sucede mediante retropropagación. Para ello, se realiza el cálculo del gradiente con respecto a los pesos para la función de pérdidas, y junto con un valor denominado "learning rate", el cual pondera la actualización que se va a realizar, se obtiene dicha actualización de parámetros.

Uno de los optimizadores más empleados es el "Stochastic Gradient Descend" (SGD) [A.20], en el cual se obtienen los pesos de la siguiente iteración sumando a los actuales el producto entre el gradiente de la función de pérdidas y el "learning rate" (ecuación 3).

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta(\mathcal{L}(\theta_t)) \qquad (3)$$

Donde "θ" son los parámetros de la red, "t" indica la iteración en la que nos encontramos, "η" es el "learning rate" y "$\mathcal{L}$" representa la función de pérdidas.

Otro tipo de optimizador también muy usado el optimizador "Adaptative Moment Estimation" (Adam), el cual es un poco más complejo que el anterior, ya que como su propio nombre indica no es una actualización constante como sucedía en SGD. Su funcionamiento se rige según la ecuación 4.

$$\begin{aligned} m &= \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla_\theta\big(\mathcal{L}(\theta_t)\big) \\ v &= \beta_2 \cdot v + (1 - \beta_2) \cdot \nabla_\theta\big(\mathcal{L}(\theta_t)\big)^2 \qquad (4) \\ \theta_{t+1} &= \theta_t - \frac{\eta \cdot m}{\sqrt{v + \varepsilon}} \end{aligned}$$

Donde "β_1" y "β_2" son los coeficientes de atenuación de los momentos de primer y segundo orden respectivamente (de forma habitual se emplean los valores de 0.9 y 0.999 respectivamente), "ε" es el valor de paso (que suele ser un valor muy pequeño), y, al igual que en el caso de SGD, "θ" son los parámetros de la red, "t" indica la iteración en la que nos encontramos, "η" es el "learning rate" y "$\mathcal{L}$" representa la función de pérdidas.

2.4.3. Funciones de regularización

Este tipo de funciones se emplean cuando se producen problemas de sobreajuste o "Overfiting", el cual consiste en que se consigue un buen rendimiento de la red para la fase de entrenamiento, pero no sucede lo propio para la fase de validación o test. Esto quiere decir que los pesos aprendidos durante la fase de entrenamiento se han sobreajustado a las características de las imágenes empleadas en el entrenamiento y si en las siguientes fases no se dan esas características específicas puede ocasionar un aumento del error.

Estas funciones intentan evitar la aparición de este tipo de error de diferentes maneras, por ejemplo, la función "DropOut" [A.21] oculta de forma aleatoria la información proveniente de algunas de las neuronas de la red, añadiendo variabilidad a las respuestas aportadas por la red frente a una misma entrada. Otra función de activación similar es "DropConnect" [A.22], la cual en vez de eliminar el nodo elimina únicamente alguna de las conexiones que se realizan.

2.5. Red neuronal de transformación espacial (STN)

Una vez explicados todos los conceptos básicos sobre las redes neuronales convolucionales es hora de hablar sobre la STN. La STN es una tipología de red la cual trabaja con imágenes sobre las que se necesita realizar una transformación para conseguir que la imagen resultado tengan el aspecto determinado, por lo que su objetivo es calcular esa transformación deseada.

Estas transformaciones a realizar pueden ser de diferente naturaleza, pueden ser transformaciones afines tipo traslaciones, rotaciones o escalados, transformaciones proyectivas como podría ser una homografía o transformaciones tipo "Thin Plate Spline" (TPS), que son transformaciones no lineales, en las cuales, de entrada, se determina la posición final deseada de un número determinado de puntos y la transformación se calcula en torno a ello.

2.5.1. Estado del arte

Desde que Jaderberg et al. desarrollaran la red neuronal de transformación espacial (STN) [A.8] esta ha sido empleada para la resolución de multitud de tareas, siendo en la mayoría de los casos empleada como una fase intermedia que adecua las imágenes de entrada para afrontar con mayor solvencia la problemática principal. Entre algunos de los problemas más frecuentes en los que suele emplearse este tipo de redes destacan las tareas de clasificación [A.8] [A.23] [A.24] [A.25], de detección de objetos [A.26] [A.27] [A.28] [A.29] o de correspondencia [A.30] [A.31].

EN el propio desarrollo de las STNs Max Jaderberg et al. [A.8] expusieron su uso para el caso de clasificación en el cual el objetivo de la STN era transformar la imagen de entrada para conseguir que el elemento que aparecía en la imagen fuera identificable más fácilmente, en este caso ellos la emplearon sobre los dataset "MNIST" o "Street View House Number" (SVHN) para identificar números y sobre el dataset "CUB-200-2011 birds" para identificar las diferentes partes de un pájaro. Otro ejemplo en el que han sido empleadas para identificar números ha sido en [A.23] donde se identifican los dorsales de las camisetas de jugadores de fútbol. Y, más allá de la clasificación de números se han utilizado en muchos otros ámbitos, como la clasificación de señales de tráfico en [A.24], o, en el ámbito de la medicina, para el reconocimiento de células tumorales en [A.25].

Para la tarea de detección de objetos la finalidad de la STN es transformar la imagen de forma que sea más sencillo reconocer las características del objeto a identificar, ejemplos de ello pueden ser [A.26] en el cual se transformaron imágenes térmicas de centrales eléctricas para la detección de los elementos de la misma o en [A.27] donde se realizó la detección de personas en imágenes capturadas por cámaras de ojo de pez. También se han empleado para la detección de secciones de las imágenes en las cuales las características se salen de lo normal en [A.28]. O su uso para el conteo de multitudes en [A.29], pero en este caso sobre vídeo en lugar de sobre imagen, para lo cual divide los "frames" en bloques y busca predecir la trayectoria de las personas de cada bloque, para realizar la transformación correspondiente, y si el nuevo mapa de activación resultante es distinto al "ground truth" para la siguiente imagen o "frame" significa que la multitud ha cambiado.

Por último, en cuanto al problema de correspondencia, que es la aplicación utilizada en este trabajo, se busca modificar el aspecto de una imagen y transformarlo en otro aspecto marcado como etiqueta o "ground truth". Esto es lo que se expone en [A.30], en el que se busca dotar de un

aspecto más realista a imágenes sintéticas de moscas de la fruta, de *D.rerio* y de *C. elegans*. Para ello, parte de las segmentaciones de las imágenes sintéticas y tiene como "ground truth" segmentaciones de las imágenes reales, de forma que la STN calcula la transformación a realizar para corresponder ambas imágenes. Y también encontramos esta problemática en [A.31], donde lo que se busca es obtener un modelado 3D de una cara a partir de la información procedente de imágenes del mismo individuo desde diferentes perspectivas, en este caso se emplean transformaciones tipo TPS.

2.5.2. Estructura de la red

Las redes tipo STN cuentan con una estructura base la cual puede dividirse en dos secciones. Por un lado, estaría una red neuronal convolucional de localización donde se obtiene la información del objeto en la imagen de entrada y se calcula la transformación que se va a realizar, y, por otro lado, habría una pareja de funciones encargadas de obtener la imagen transformada, la primera a partir de la transformación calculada en la fase anterior se encarga de la obtención de la malla o "grid" a aplicar y la segunda se encarga de aplicarla para obtener así la imagen transformada (Figura 7).

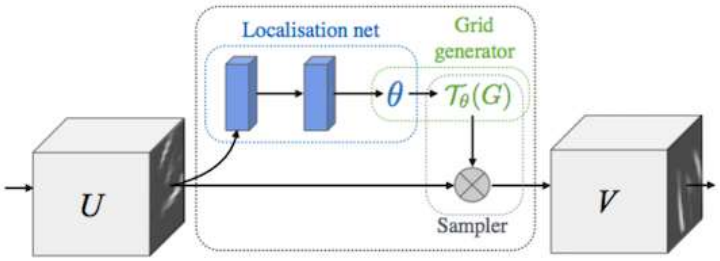


Figura 7. Esquema estructura STN [I.7]

CAPÍTULO 3. SISTEMA DE CAPTURA DE IMÁGENES

En este apartado se exponen todos los elementos de los que consta el mecanismo encargado de realizar las capturas de imágenes de *C. elegans*, las cuales constituirán el dataset sobre el que se van a llevar a cabo los ensayos, así como la metodología empleada para realizar las capturas.

Los *C. elegans* capturados se encuentran en el interior de placas Petri, las cuales son transportadas por un carro que las ilumina mediante retroiluminación. A su vez, este carro se encuentra montado en un robot cartesiano, el cual puede desplazarlo sobre el plano XY cuando sea necesario para realizar el seguimiento de los nematodos.

Aparte del carro, el robot cartesiano también transporta sobre su cabezal, elemento que se desplaza sobre el eje Z, dos cámaras de microscopía, las cuales realizan las capturas de las imágenes empleadas en este experimento, y un puntero láser, que ilumina de la zona de captura.

3.1. Robot cartesiano

El robot cartesiano empleado ha sido un robot diseñado por los integrantes del grupo de robótica del Instituto Universitario de Automática e Informática Industrial (ai2) [O.2] (Figura 8). Para su elaboración se contó con perfiles de aluminio de 20x20 mm que conforman la estructura principal y mediante una impresora 3D (BCN3D Sigma 3D printer-6) se obtuvieron todas las partes de sujeción para los distinto elementos del robot.

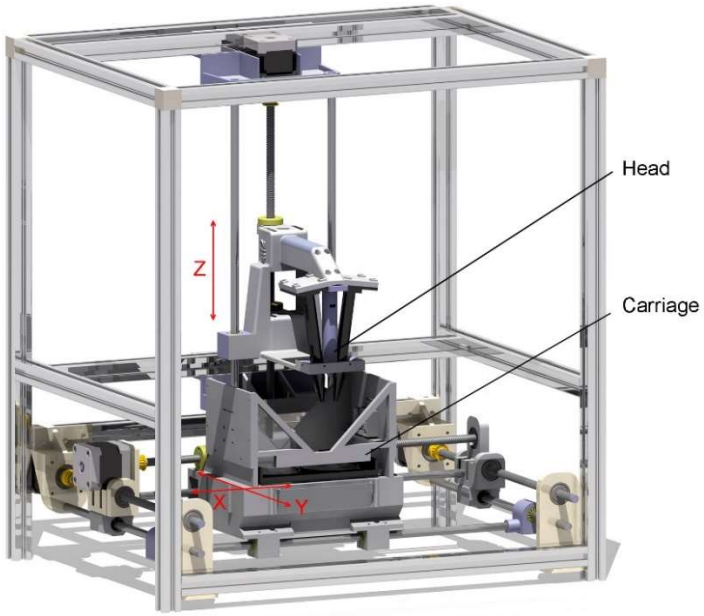


Figura 8. Robot cartesiano [I.8]

Para conseguir los movimientos del robot se dispusieron cuatro motores paso a paso Nema 17 controlados por un microcontrolador Teensy 3.6 y alimentados cada uno de ellos por un DRV8825.

3.2. Carro

El carro es el encargado del transporte las placas Petri, pudiendo llevar hasta un total de dos de forma simultánea. Para la iluminación de las placas se cuenta con un display de 7" sobre el que se colocan estas, de forma que la iluminación se produce mediante retroiluminación consiguiendo que sea lo más homogénea posible.

Aparte de las placas y el sistema de iluminación el carro también dispone de 2 cámaras de macroscopía y dos divisores de luz. Estos elementos se emplean para realizar capturas en las que aparece la totalidad la placa Petri de forma que se pueden situar todos los nematodos presentes en ella y así poder detectar su movimiento y realizar su seguimiento.

Las cámaras se han colocado en los laterales del carro mientras que los divisores se han dispuesto formando un ángulo de 45° tanto con respecto a la cámara de macroscopía como con la placa Petri, de esta forma el divisor refleja la placa Petri retroiluminada para que la cámara pueda realizar la captura sin que se deformen sus dimensiones en la imagen obtenida. Además, esta colocación de los elementos permite mantener despejada la parte superior que es la zona donde se han situado las cámaras de microscopía para realizar las capturas.

En la Figura 9 se puede observar un esquema del posicionamiento de todos estos elementos, así como una muestra de las capturas realizadas por las cámaras de macroscopía y por las cámaras de microscopía del cabezal.

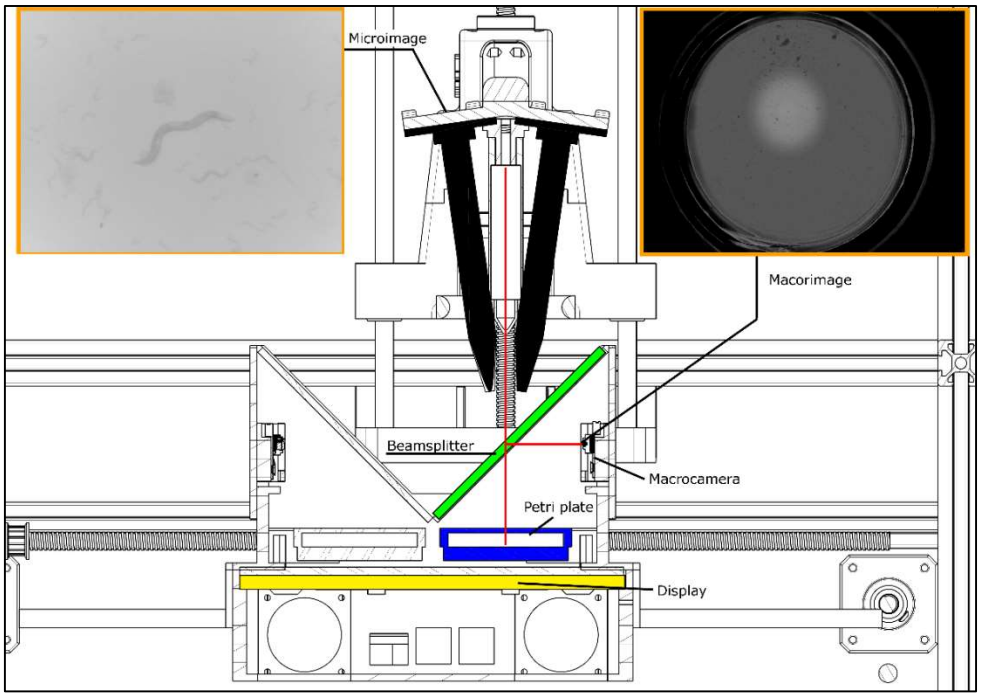


Figura 9. Configuración carro-cabezal del robot cartesiano [I.9]

3.3. Cámaras de microscopía

Las cámaras de microscopía se encuentran colocadas en el cabezal del robot cartesiano junto con un puntero láser. El láser se sitúa en posición perpendicular al plano en el que se encuentran las placas Petri. Su funcionalidad es, en primera instancia, iluminar la zona donde se va a realizar la captura, y, además, también sirve para determinar la posición del cabezal con respecto al carro, ya que en las cámaras de macroscopía se puede observar la zona iluminada y, por tanto, conocer su posición. Esto último también ayuda a la realización del seguimiento de los nematodos de forma efectiva.

En cuanto a las cámaras, son dos cámaras idénticas en lo que a especificaciones se refiere. Se trata de dos cámaras de alta resolución con una distancia focal de 102.61 mm y una resolución de 1.9 µm/pxl que realizan capturas de tamaño 2592x1944 pixeles. Cada una de ellas se encuentra controlado por una placa Raspberry Pi v.4.

Ambas cámaras se encuentran desplazadas de la perpendicular al plano de las placas Petri por un ligero ángulo (alrededor de 8º) cada una hacia un lado del puntero. Esta colocación permite obtener dos imágenes de la misma zona con diferentes perspectivas.

Pese a que la colocación de las cámaras es simétrica el nematodo no se encuentra centrado en las dos cámaras, sino que en una de ellas se encuentra desplazado hacia la parte superior derecha de la imagen, este hecho es lo que se quiere solucionar con la aplicación de las STN. A la cámara que si obtiene capturas con el nematodo centrado se referirá a partir de ahora como Micro 1, mientras que a la que captura el nematodo escorado se denominará Micro 2. Una muestra de las imágenes capturadas se observa en la Figura 10.

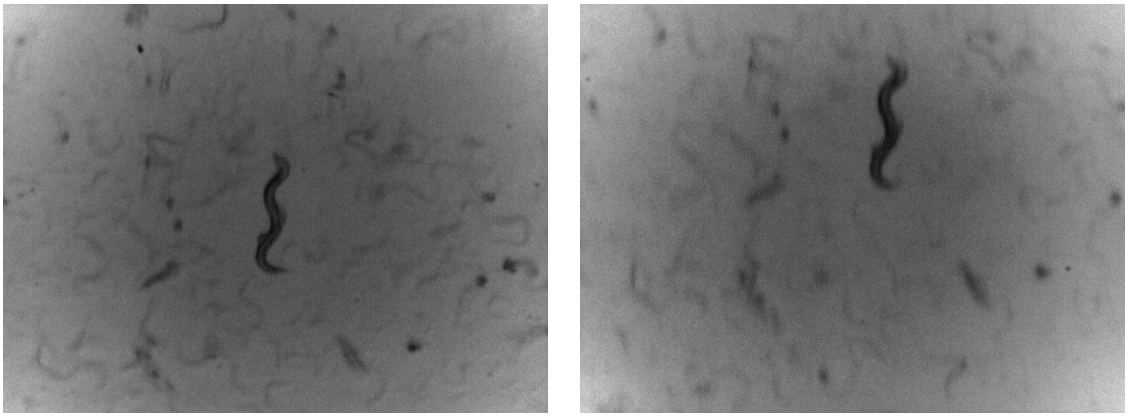


Figura 10. Imágenes obtenidas por las cámaras Micro 1 (izquierda) y Micro 2 (derecha). Estas imágenes no son exactamente las capturadas por las cámaras, sino que se les ha aplicado un filtro de aumento de contraste.

Como se ha comentado el objetivo de la STN será otorgar a las imágenes de Micro 2 el aspecto de las imágenes de Micro 1, para que estas puedan aplicarse indistintamente en otras problemáticas. Además, también sirve para solucionar el problema de enfoque, ya que como se observa en la Figura 10 también hay una pequeña diferencia de nitidez entre las dos imágenes.

3.4. Procedimiento de captura

En primer lugar, se recoge la información del posicionamiento de los nematodos y del puntero láser a través de la cámara de macroscopía, con esta información se selecciona uno de los nematodos sobre el que se van a realizar las capturas, y conociendo tanto la posición de los nematodos como la del cabezal, se envían las acciones de control para desplazar el carro hasta que el cabezal quede en la misma posición que el nematodo. En ese instante se comienzan a realizar las capturas con una frecuencia de 2 fps hasta realizar un total de 60 capturas.

Durante el proceso de obtención de esas 60 capturas si el nematodo se desplaza la cámara de microscopía de Micro 1 detecta dicho movimiento y envía una acción de control al robot cartesiano para que corrija la posición del carro de forma que el nematodo se mantenga centrado en la imagen.

Una vez completado el proceso de captura para un nematodo se selecciona otro de los nematodos de la misma placa y se vuelve a comenzar el procedimiento. Este procedimiento se repite mientras se pueda asegurar que el siguiente nematodo no ha sido capturado con anterioridad, para disponer de la misma cantidad de imágenes de todos ellos.

CAPÍTULO 4. SIMULADOR

De forma habitual los datasets empleados para trabajar con redes neuronales están formados por centenas de miles, millones o incluso billones de imágenes en algunos casos. En el caso de estudio el número de imágenes que se dispone ha sido mucho más reducido, ya que el proceso de captura es laborioso y costoso temporalmente.

En muchos estudios que trabajan con redes neuronales es normal que frente a esta problemática del tamaño del dataset se opte por el empleo de simuladores para aumentar su tamaño generando imágenes artificiales semejantes a las reales.

En el caso de estudio se ha empleado un simulador, pero este no ha sido empleado para la ampliación del dataset, pese a que el número de imágenes de las que se dispone no es muy elevado, sino que su finalidad ha sido la de generar un dataset simulado independiente con unas características similares a las que se dan en el dataset real para poder estudiar la problemática planteada en este trabajo desde un punto de vista más controlado.

El uso de un dataset simulado permite tener el control sobre las características de todas las imágenes que lo conforman, y, de esta forma, poder plantear y analizar algunos casos con características especiales.

4.1. Funcionamiento del simulador

El simulador empleado consistirá en la generación de dos cámaras virtuales cuyas capturas imiten el aspecto de las realizadas por las cámaras reales de microscopía.

Para la realización de estas capturas es necesario definir por un lado el aspecto de los objetos que se van a capturar, y, por otro las características de las cámaras (altura, inclinación, distancia focal…). Definidos estos parámetros, el simulador mediante las características de las cámaras calcula las matrices de proyección para ambas cámaras con respecto al plano en el que se encuentra el objeto, y, de esta forma, calcular la transformación a realizar sobre los puntos del objeto a representar desde el punto de vista de cada cámara.

4.2. Definición del objeto de interés

El primer paso a realizar en el simulador es la definición del objeto a representar, para ello deben definirse el posicionamiento de los puntos que conforman su silueta sobre el plano horizontal.

Como se ha comentado anteriormente lo ideal sería que un simulador generase imágenes lo más similares posibles a las empleadas en el dataset real, pero como en nuestro caso no se va a trabajar

con ambos dataset de forma simultánea este no es un aspecto crítico, además, la definición de la silueta de un nematodo no es sencilla. Es por ello, que para simplificar su definición se optó por que en las imágenes simuladas el objeto de interés representado fuese un rectángulo, ya que basta con marcar sus cuatro vértices para definir su silueta.

Además de la forma a representar otro aspecto de igual importancia es definir el tamaño del mismo, en este caso, para intentar que la problemática planteada con este tipo de imágenes sea lo más semejante posible al planteado con imágenes de *C. elegans* se optó por un tamaño de 80x300 píxeles, puesto que así la cantidad de pixeles pertenecientes al objeto de interés es semejante a la que se da en el caso de los *C. elegans*.

Una vez definidas las características básicas del objeto, con la finalidad de otorgar una mayor variabilidad a las imágenes, se establecieron ciertos parámetros para cambiar, de forma aleatoria, las características de los rectángulos. Por un lado, su tamaño se puede reducir hasta el 50% o aumentar hasta el 150% del tamaño base en cada una de las direcciones de forma independiente. Por otro, para variar su orientación se le aplica un giro de forma aleatoria sobre su centro. Y, por último, como en las imágenes originales el posicionamiento del nematodo en Micro 1 no está totalmente centrado, se ha añadido también un pequeño desplazamiento aleatorio en cada uno de los dos ejes siendo como máximo de 100 píxeles en el sentido negativo del eje y de 50 en el positivo. En la Figura 11 se muestran todos estos aspectos.

```
incX = random.random()
incY = random.random()
X = np.array(
    [[(0.75 + 0.5 * incX) * -40, (0.75 + 0.5 * incX) * -40, (0.75 + 0.5 * incX) * 40, (0.75 + 0.5 * incX) * 40],
     [(0.75 + 0.5 * incY) * -150, (0.75 + 0.5 * incY) * 150, (0.75 + 0.5 * incY) * 150, (0.75 + 0.5 * incY) * -150],
     [0, 0, 0, 0], [1, 1, 1, 1]], dtype=np.int32)

ang = 2*pi*random.random()
D = math.sqrt(X[0][0] * X[0][0] + X[1][0] * X[1][0])
X1 = np.array([[D * math.cos(ang + math.atan2(X[1][0], X[0][0])), D * math.cos(ang + math.atan2(X[1][1], X[0][1])),
                D * math.cos(ang + math.atan2(X[1][2], X[0][2])), D * math.cos(ang + math.atan2(X[1][3], X[0][3]))],
               [D * math.sin(ang + math.atan2(X[1][0], X[0][0])), D * math.sin(ang + math.atan2(X[1][1], X[0][1])),
                D * math.sin(ang + math.atan2(X[1][2], X[0][2])), D * math.sin(ang + math.atan2(X[1][3], X[0][3]))],
               [0, 0, 0, 0], [1, 1, 1, 1]], dtype=np.int32)
trasX = int(-100+150*random.random())
trasY = int(-100+150*random.random())
X = np.array([[X1[0][0] + trasX, X1[0][1] + trasX, X1[0][2] + trasX, X1[0][3] + trasX],
              [X1[1][0] + trasY, X1[1][1] + trasY, X1[1][2] + trasY, X1[1][3] + trasY], [0, 0, 0, 0],
              [1, 1, 1, 1]], dtype=np.int32)
```

Figura 11. Parámetros de configuración de los rectángulos simulados

4.3. Definición de las cámaras

Para la definición de las cámaras hace falta fijar dos tipos de parámetros, por un lado, están los parámetros intrínsecos y, por otro, los extrínsecos.

Dentro de los parámetros intrínsecos se encuentran todos aquellos propios de las cámaras. Entre ellos se encuentra la distancia focal. Ambas cámaras serán idénticas en estos aspectos, al igual que en el caso real, para la distancia focal en pixeles se ha fijado un valor de 1580 (Figura 12).

```
Parametros intrinsecos
"""

fu = 1580   # fu = f/dx
fv = 1580   # fv = f/dy
uo = 150    # uo = xo
vo = 100    # vo = yo
```

Figura 12. Parámetros intrínsecos cámara virtual

Los parámetros extrínsecos marcan aspectos externos a las cámaras, como su posicionamiento. En este tipo de parámetros sí que se producirán diferenciaciones entre las dos cámaras.

En el caso de la cámara que imita a Micro 1 (Figura 13), ella se dispone inicialmente centrada en el eje de las X y a una altura de 1000 unidades de medida en el mundo en el eje Z. A partir de esta posición se le aplica un giro de 5° en torno al eje X de forma que se encuentre un poco desviada de la perpendicular al plano de representación.

```
"""Parametros extrinsecos"""
radio = 1000
angulo = 180-5

rX = (pi/180)*(angulo)   # angulo de rotación en el eje X
rY = 0                    # angulo de rotación en el eje Y
rZ = 0                    # angulo de rotación en el eje Z
tX = 0                                  # Traslacion en el eje X
tY = math.sin((pi/180)*(angulo))*radio     # Traslacion en el eje Y
tZ = -math.cos((pi/180)*(angulo))*radio    # Traslacion en el eje Z
```

Figura 13. Parámetros extrínsecos cámara virtual Micro 1

Y para definir la posición de la cámara de Micro 2 (Figura 14) el procedimiento es semejante al seguido para Micro 1, la única diferencia fue que el giro en torno al eje X pasó a ser de 10º, de esta forma la altura del eje Z no será la misma, y, además, se incluirá una pequeña traslación en los ejes X e Y (150 unidades de medida en el mundo en el sentido negativo de cada eje) para conseguir que el objeto se encuentre escorado hacia arriba y a la derecha al igual que en las imágenes de Micro 2.

```
"""Parametros extrinsecos"""

radio = 1000
angulo = 180-10

rX = (pi/180)*(angulo) # angulo de rotación en el eje X
rY = 0      # angulo de rotación en el eje Y
rZ = 0      # angulo de rotación en el eje Z
tX = -150                 # Traslacion en el eje X
tY = math.sin((pi/180)*(angulo))*radio-150     # Traslacion en el eje Y
tZ = -math.cos((pi/180)*(angulo))*radio    # Traslacion en el eje Z
```

Figura 14. Parámetros extrínsecos cámara virtual Micro 2

4.4. Resultados del simulador

Una vez definidos todos los parámetros tanto de las cámaras como del objeto que se va a representar el simulador ya está listo para funcionar.

Las imágenes que devuelve el simulador son imágenes segmentadas de los rectángulos, con un tamaño de 2592x1944 píxeles, tamaño de captura de las cámaras reales. En la Figura 15 se puede observar un ejemplo de las imágenes devueltas por el simulador.

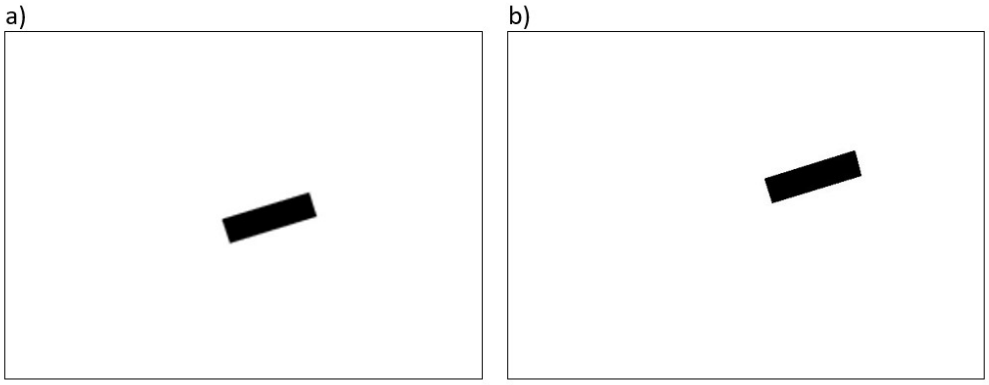


Figura 15. Imágenes devueltas por el simulador. a) Imagen Micro 1 b) Imagen Micro 2

CAPÍTULO 5. DATASETS

En esta sección de la memoria se presenta el dataset que se ha empleado para realizar los diferentes ensayos, así como las determinadas circunstancias que los han condicionado.

Como se ha explicado en el capítulo anterior el simulador empleado se utiliza para generar un dataset alternativo al formado por las imágenes de *C. elegans* capturadas en el laboratorio, es por ello que a grandes rasgos se podrán diferenciar entre dos tipos de dataset, aquellos que empleen imágenes simuladas y aquellos que usen imágenes reales de *C. elegans*.

5.1. Dataset de *C. elegans*

El dataset de *C. elegans* está formado por las imágenes capturadas en el laboratorio mediante el sistema de captura detallado en el capítulo 3. Se realizaron capturas de diferentes placas durante los dos primeros días de su disposición.

Se optó por realizar únicamente capturas durante los dos primeros días de vida de cada placa, ya que a partir de ese momento los nematodos empezaban a reproducirse, por lo que la cantidad de nematodos en la placa se multiplicaba, lo que dificultaba el seguimiento al haber agrupaciones más frecuentemente.

De todas las imágenes capturadas se optó por emplear únicamente las imágenes pertenecientes a una placa en un día en concreto, ya que las condiciones entre las diferentes placas y días cambiaban.

Por lo que, finalmente, el dataset inicial con el que se han empleado para la realización de los ensayos partía de un total de 1020 imágenes para cada una de las cámaras, Micro 1 y Micro 2.

5.1.1. Edición del dataset por problemas de captura

En los dataset adquiridos de forma propia, es decir, formados por imágenes capturadas por el propio grupo de investigación que va a realizar el estudio es habitual que se produzcan diversos problemas en la adquisición de imágenes que provoquen que se realicen capturas fuera de las condiciones habituales.

En este proyecto estas imágenes anómalas pueden ser debidas a diferentes tipologías de errores. En primer lugar, se encuentran las producidas por errores en el seguimiento de los nematodos de forma que, por ejemplo, en las imágenes tomadas por la cámara Micro 1, donde el nematodo debería aparecer centrado, este aparece fuera de la posición central.

Otra de las tipologías de error, es la producida por la desincronización en la captura de las cámaras, esto provoca que si el nematodo se encuentra en movimiento en el momento de la realización de las capturas el posicionamiento de este en las dos imágenes no sea el mismo. Este hecho, sumado a que

durante la toma de imágenes el robot hace desplazamientos para realizar el seguimiento de los nematodos, provoca que la transformación necesaria para emparejar esta tipología de imágenes diste mucho de la habitual, puesto que ya no es diferente solo la forma del nematodo, sino también su posicionamiento sobre la imagen.

De todas las imágenes que contenían errores se intentó minimizar el número de parejas de imágenes eliminadas escogiendo para ello aquellas que eran claramente diferentes, de esta forma todavía se mantiene cierta variabilidad en el dataset y, así se consigue también que el tamaño del dataset sea lo más grande posible.

Tras la eliminación de todas estas parejas la cantidad de imágenes que se van a emplear en los ensayos se ha reducido hasta 975 parejas de imágenes.

5.1.2. Tipología de imágenes

Una vez determinada la cantidad de imágenes que se han empleado otro aspecto a determinar es la naturaleza de dichas imágenes, es decir, si se va a trabajar con ellas con el aspecto de captura o se le va a realizar algún tipo de procesamiento.

En el problema de correspondencia, la forma más habitual de trabajar es con las segmentaciones del objeto de interés como sucedía en [A.30], donde se utilizaban las segmentaciones de los diferentes objetos de interés para abordar este problema, por lo que este será uno de los tipos de imágenes con los que se va a trabajar.

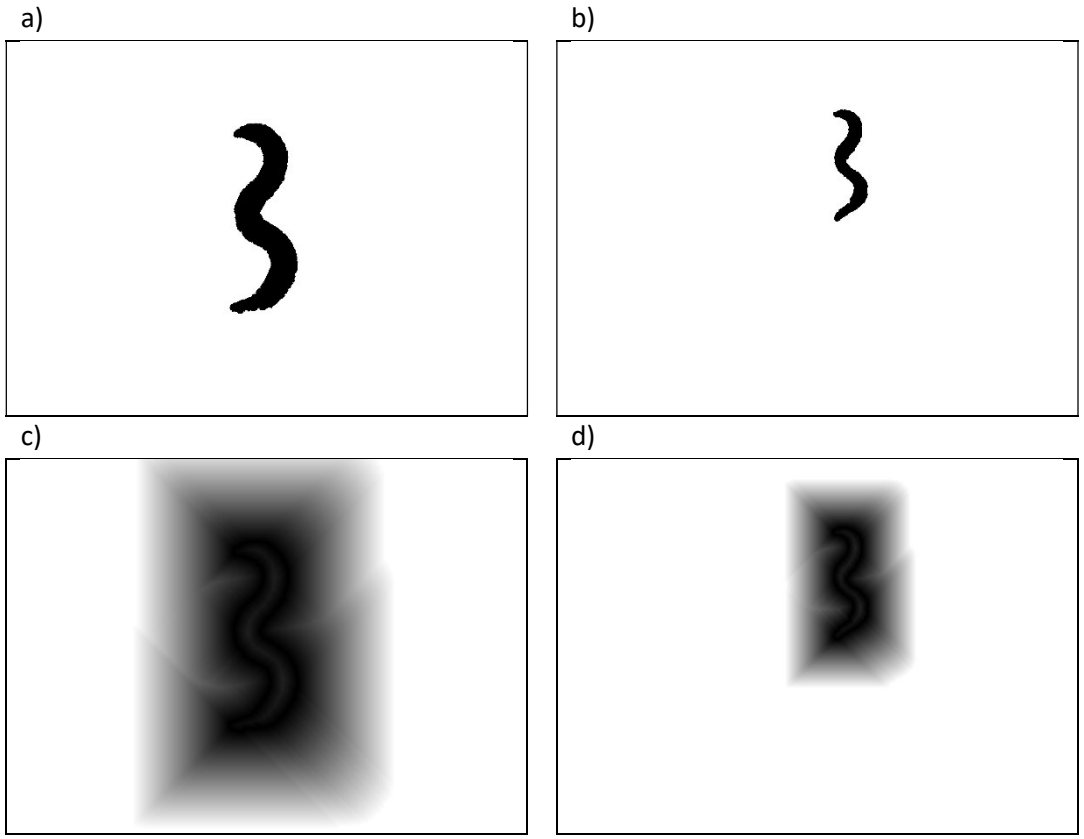


Figura 16. Imágenes dataset C. elegans. a) imagen Micro 1 recortada segmentada b) Imagen Micro 2 segmentada c) imagen Micro 1 recortada de transformación de distancias d) Imagen Micro 2 de transformación de distancias

Como se demuestra en los ensayos al emplear este tipo de imágenes se producen problemas de convergencia, por lo que se optó por emplear para los ensayos imágenes de transformación de distancias a los bordes del objeto de interés, las cuales aumentan bastante la información útil de la misma.

Para la obtención de las imágenes, tanto de las segmentadas como de las de transformación de distancias, se han empleado las funcionalidades de la librería OpenCV de Python.

Por lo que, de forma definitiva, los dataset que van a emplearse estarán formados por un total de 975 parejas de imágenes, las cuales pueden ser imágenes de *C. elegans* segmentadas o imágenes de *C. elegans* de transformación de distancias a los bordes, o, dado el caso, se puede hacer una mezcla de ambos tipos.

En la Figura 16 se expone una muestra de las imágenes que conforman cada uno de los datasets con los que se van a trabajar.

5.2. Dataset simulado

Este dataset está formado por las imágenes aportadas por el simulador de cámara virtuales empleado. Estas imágenes como se comentó en el capítulo anterior son imágenes que tienen como objeto de trabajo un rectángulo.

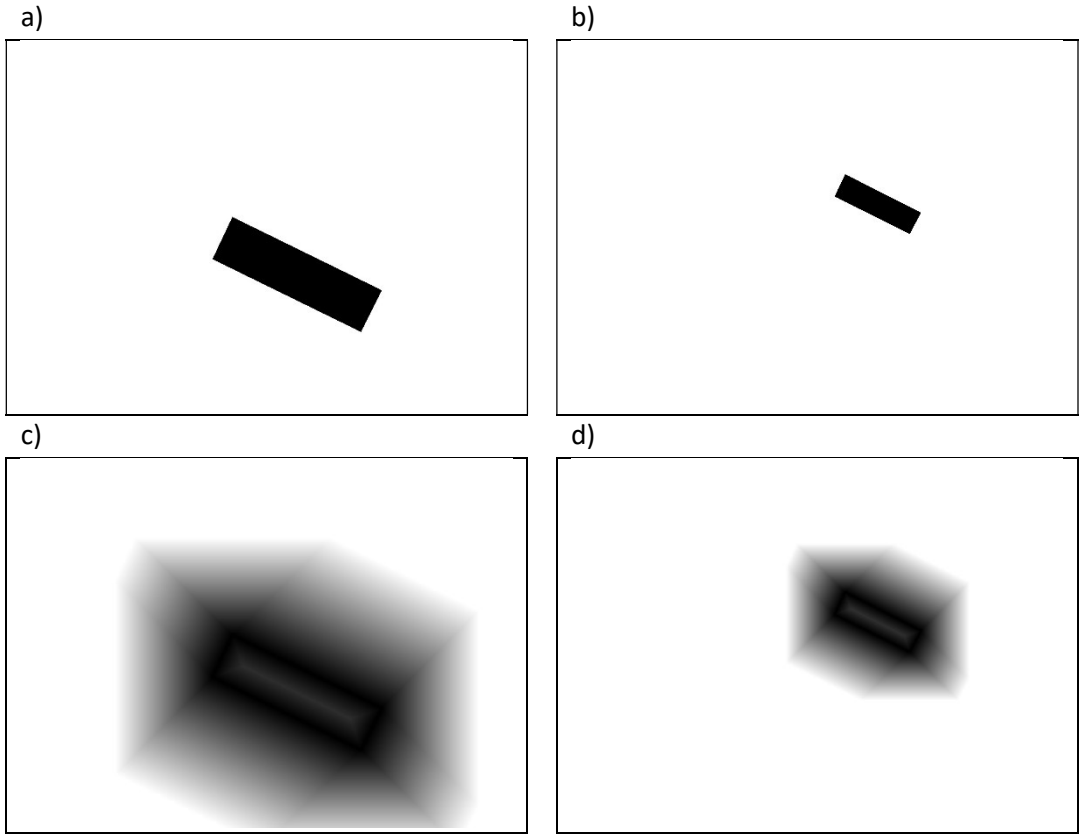


Figura 17. Imágenes dataset simulado. a) imagen Micro 1 recortada segmentada b) Imagen Micro 2 segmentada c) imagen Micro 1 recortada de transformación de distancias d) Imagen Micro 2 de transformación de distancias

Como la finalidad del dataset es generar una problemática semejante a la real, pero desde un punto de vista más controlado el número de imágenes con el que se han realizado los ensayos ha sido de 1.000 parejas de imágenes.

Cabe destacar que la generación de estas imágenes se ha realizado de forma pseudoaleatoria marcando la semilla para la generación de cada una de las imágenes. De esta forma se consigue que el dataset sea el mismo para todos los ensayos realizados para esta tipología de imágenes.

En este caso al tratarse de imágenes sintéticas no se producen los problemas de captura expuestos en el dataset de *C. elegans*, por lo que todas las imágenes producidas han podido ser utilizadas.

Con todo ello, y para abordar las mismas problemáticas que con el dataset de *C. elegans* las tipologías de imágenes que han conformado los dataset son las mismas, segmentadas y de transformación de distancias, o de ambas según el caso. En la Figura 17 se exponen las imágenes que conforman ambos datasets.

CAPÍTULO 6. RED NEURONAL DE TRANSFORMACIÓN ESPACIAL (STN) UTILIZADA

En el primero de los capítulos se definió la estructura de las redes neuronales de transformación espacial (STN) de una forma bastante general, pero su tamaño puede variar desde una estructura muy compleja con muchas capas hasta una muy simple con solamente un par de ellas en función de la complejidad del problema a solucionar.

Es por ello que en este apartado se realiza una explicación más en detalle de la estructura de la red empleada, así como de todos los elementos que la acompañan para realizar los entrenamientos.

6.1. Estructura de la red

El caso de estudio en el que se basa este trabajo se asemeja bastante al que se plantearon en [A.30] donde se busca solucionar este mismo problema para dar más realismo a imágenes sintéticas, además entre sus ensayos también trabajan con imágenes de *C. elegans*. Por lo que la estructura de la red en cuanto a número de capas y tipología de las mismas es idéntica a la que se empleó en [A.30].

Como se comentó en el capítulo 2 este tipo de redes se componen de dos partes bien diferenciadas, por un lado, una red neuronal de localización y, por otro, un sistema que calcula el "grid" y lo aplica sobre la imagen para obtener la imagen transformada (Figura 18).

En este caso, la primera de las partes, la red neuronal de localización, a su vez puede dividirse en dos partes, una formada por capas bidimensionales y otra constituida por capas lineales.

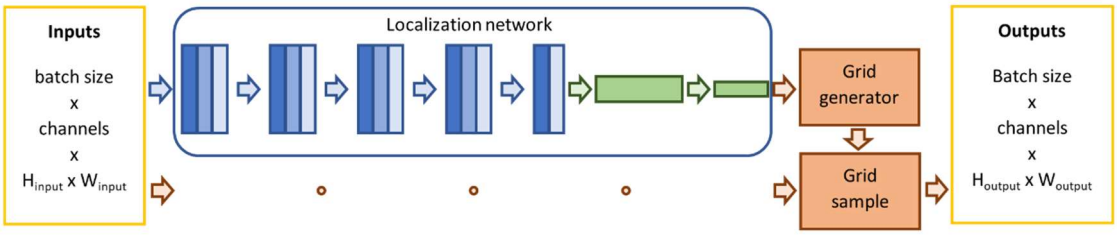


Figura 18. Esquema red neuronal de transformación espacial (STN) empleada

La primera de estas subetapas está constituida por una secuencia de capas convolucionales seguidas cada una de ellas por una capa de "maxpooling" y una capa de activación tipo "SELU", excepto para el caso de la última capa convolucional, la cual se encuentra únicamente seguida por una capa de activación tipo "SELU". Con todo ello, esta fase se dispone de un total de 5 capas convolucionales, 5 capas de activación tipo "SELU" y 4 capas de "maxpooling" en el orden expuesto.

La segunda sección de la red de localización adapta la información obtenida en la etapa anterior para definir los parámetros de la transformación que se va a realizar. Esta sección está constituida por dos capas lineales o totalmente conectadas, entre las cuales se vuelve a situar una capa de activación tipo "SELU". El resultado de la última capa es un vector de longitud 12 cuyos valores se reorganizan en una matriz de 3 filas y 4 columnas la cual representa la transformación que se va a realizar y sirve de entrada para la segunda parte de la STN.

En la tabla 1 se puede observar un resumen de las características principales de cada una de las capas.

Capas	Tamaño de salida	Detalles de las capas
Convolucional 1	[batch_size, 8, 1938, 2586]	Kernel = 7x7
MaxPooling 1	[batch_size, 8, 969, 1293]	Stride = 2
Activación SELU 1	[batch_size, 8, 969, 1293]	inplace=True
Convolucional 2	[batch_size, 10, 965, 1289]	Kernel = 5x5
MaxPooling 2	[batch_size, 10, 482, 644]	Stride = 2
Activación SELU 2	[batch_size, 10, 482, 644]	inplace=True
Convolucional 3	[batch_size, 20, 478, 640]	Kernel = 3x3
MaxPooling 3	[batch_size, 20, 239, 320]	Stride = 2
Activación SELU 3	[batch_size, 20, 239, 320]	inplace=True
Convolucional 4	[batch_size, 40, 237, 318]	Kernel = 3x3
MaxPooling 4	[batch_size, 40, 118, 159]	Stride = 2
Activación SELU 4	[batch_size, 40, 118, 159]	inplace=True
Convolucional 5	[batch_size, 10, 116, 157]	Kernel = 3x3
Activación SELU 5	[batch_size, 10, 116, 157]	inplace=True
Lineal 1	[batch_size, 32]	-
Activación SELU 6	[batch_size, 32]	inplace=True
Lineal 2	[batch_size, 12]	-

Tabla 1. Resumen características de las capas de la red

Por último, la segunda sección de la STN se encarga de obtener el "grid" que se debe aplicar sobre la imagen de entrada para conseguir que se asemeje a la imagen objetivo, para ello emplea la información obtenida en la red de localización recogida en esa matriz de 3 filas y 4 columnas (ecuación 5), la cual representa la transformación a realizar. Para ello, se han utilizado las funciones de Pytorch "affine_grid" y "grid_sample".

$$\begin{pmatrix} X_{output} \\ Y_{output} \\ Z_{output} \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} \cdot \begin{pmatrix} X_{input} \\ Y_{input} \\ Z_{input} \end{pmatrix} \qquad (5)$$

La primera de ellas se encarga de obtener dicho "grid", a esta función se le facilita como entrada la matriz 3x4 obtenida en la red de localización y el tamaño del "grid" que se quiere calcular, el cual debe ser coincidente con el tamaño de la imagen objetivo para poder realizar posteriormente la comparación entre ambas.

Y la segunda función, "grid_sample", partiendo del "grid" obtenido con la primera y de la imagen de entrada a la red, obtiene el resultado final aplicando dicho "grid" sobre esta imagen mediante una interpolación bilineal de los valores pertinentes.

Estas dos funciones, permiten el cálculo de dos tipos de transformaciones, por un lado, transformaciones tipo afín, tales como rotaciones, escalados o traslaciones, y, por otro lado, transformaciones de tipo proyectivo, como la homografía. En el caso de estudio se optó por emplear la transformación proyectiva, ya que se adapta mejor al tipo de transformación que se necesita aplicar sobre las imágenes empleadas, es debido a esto por lo que la matriz de entrada a la función "affine_grid" es de tamaño 3x4, teniendo que ser esta de tamaño 2x3 en caso de querer aplicar la transformación afín.

Además, en la función "grid_sample" se ha establecido el parámetro "padding_mode" a "border", este parámetro lo que produce es que si por algún casual la transformación obtenida por la red abarca pixeles que se encuentran fuera de la imagen en lugar de generar estos valores en negros se seleccionaría para esos pixeles el valor del borde más cercano a ellos, y de forma que sería como extender los colores de los pixeles de borde a los pixeles que quedan fuera de la imagen.

6.2. Elementos adicionales y ajuste de hiperparámetros

En este subapartado se detallan los elementos que acompañan a la red expuesta en el subapartado previo, así como el ajuste de los hiperparámetros de la red. Este ajuste de hiperparámetros es una tarea fundamental para asegurar la convergencia de la misma. En el caso de estudio los hiperparámetros se han ajustado en función de las limitaciones de la red y de la GPU empleada.

El primero de todos los hiperparámetros en fijarse fue el tamaño de lote, el cual se marcó en 8 debido al gran tamaño de las imágenes, puesto que este era el tamaño máximo que cumplía con la limitación de memoria de la GPU utilizada.

Para la definición del criterio de parada del entrenamiento se han seguido dos criterios distintos en función del dataset empleado en él. Si nos encontramos en un ensayo con dataset simulado el entrenamiento se detiene después de realizar 150 épocas, mientras que si nos encontramos en un ensayo con dataset de *C. elegans* se produce cuando se alcanza un nivel de convergencia dado, el cual se corresponde con un valor para el IoU de 0.86 (se ha observado que este valor está muy próximo al límite superior de precisión), o bien cuando el entrenamiento es incapaz de mejorar, hecho que se da cuando no consigue hacerlo en las últimas 20 épocas del entrenamiento.

Por otro lado, se contó con un optimizador tipo "Adam" para que ayude a conseguir la convergencia, cuyo parámetro de "learning rate", que es considerado también un hiperparámetro, se define mediante uno de los ensayos realizados, los cuales se encuentran expuestos en el capítulo 7.

Y, por último, otro elemento es la función de pérdidas, en este caso como lo que se busca es igualar las dos imágenes se necesita de una función de pérdidas que compare la información de las imágenes pixel a pixel, es por ello que se decidió usar la de "Mean Square Error" (MSE loss) para entrenar la STN, cuya fórmula viene representada en la ecuación 6.

$$MSE\ Loss = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m}(X_{(n,m,c)} - Y_{(n,m,c)})^2}{m * n} \qquad (6)$$

Siendo "n" y "m" el tamaño de las imágenes en cada uno de los ejes, "c" el canal de las imágenes y "X(n,m,c)" e "Y(n,m,c)" los valores de los pixeles de la imagen transformada devuelta por la red y de la

imagen objetivo respectivamente. Esta función calcula la media de las distancias entre los valores de los pixeles de las dos imágenes, evaluando de esta forma cuan parecidas son.

6.3. Métricas

Otro elemento importante por definir son las métricas que se van a emplear. Las métricas son unos índices los cuales sirven para poder interpretar más fácilmente los resultados obtenidos con la red, puesto que en muchas ocasiones los valores de la función de pérdidas son difíciles de interpretar.

Se seleccionaron diferentes tipos de métricas para estimar la bondad de los resultados de la red, cabe destacar que todas ellas se han aplicado sobre imágenes segmentadas, independientemente del ensayo que se realizase, puesto que el objetivo es transformar la imagen de Micro 2 para que coincida el posicionamiento, o, en otras palabras, las segmentaciones, de los nematodos o de los rectángulos, en función del ensayo. En caso de que el entrenamiento trabajase con imágenes de transformación de distancias estas se emplearían para obtener el "grid", pero este en vez de aplicarse sobre la propia imagen de transformación de distancias se aplicaría sobre la imagen segmentada correspondiente.

Las métricas pueden tener diferentes naturalezas en función de lo que se quiera calcular con ellas. En el caso que se está estudiando, por un lado, debido al tamaño reducido del objeto de interés en comparación con el resto de la imagen, se realizó el conteo del número de pixeles no coincidentes entre las dos imágenes, es decir, los pixeles de error y se registró su valor medio, máximo y mínimo para todas las parejas de imágenes empleadas en el entrenamiento, de esta forma se ha caracterizado mejor el error que se produce.

Por otro lado, se optó por emplear la métrica IoU (Intersection over Union), ya que refleja exactamente lo que se quiere conseguir, dicha métrica calcula el ratio entre la intersección y la unión de las segmentaciones de los objetos de interés de las dos imágenes segmentadas, como aparece reflejado en la ecuación 7.

$$IoU = \frac{X \cap Y}{X \cup Y} \qquad (7)$$

Siendo "X" el nematodo o rectángulo segmentado al que se le ha aplicado la transformación obtenida por la red e "Y" la segmentación del nematodo o rectángulo de la imagen que se dispone como etiqueta. De forma que este ratio refleja el porcentaje de superposición que se produce entre los objetos de interés de las dos imágenes. Es por ello que esta métrica ha sido utilizada como la principal fuente de comparación entre los diferentes ensayos

Y, por último, en los ensayos con imágenes de *C. elegans* también se ha monitoreado el tiempo de entrenamiento de la red para tener en cuenta el coste temporal o computacional en lo que tarda en llegar al límite de precisión establecido, ya que si este es demasiado elevado una buena alternativa en cuanto a precisión podría ser inviable en cuanto a costo.

CAPÍTULO 7. METODOS DE ENTRENAMIENTO, EXPERIMENTOS REALIZADOS Y RESULTADOS

En este apartado se exponen los métodos estudiados para solucionar el problema de correspondencia, junto con los ensayos que se han realizado y los resultados obtenidos. En primer lugar, se han comparado los ensayos en los cuales se emplean únicamente una de las tipologías de imágenes en los datasets, para determinar cuál es el tipo que genera mejores resultados, y, posteriormente, se han planteado diferentes alternativas para intentar mejorar el rendimiento del entrenamiento.

Cabe destacar que, debido a que lo que se intenta conseguir es desarrollar una nueva técnica de entrenamiento que consiga la convergencia se ha optado por prescindir de las etapas de validación y de test, ya que como se ha comentado lo que se busca es asegurar o, al menos, afianzar la convergencia del entrenamiento.

7.1. Comparación de imágenes segmentadas contra imágenes de transformación de distancias

Estos dos casos se han estudiado empleando el dataset simulado con la finalidad de analizar algunos casos específicos y obtener conclusiones más claras sobre el rendimiento de ambos datasets. En estos casos no se ha tenido en cuenta el costo computacional, puesto que su finalidad fue encontrar el método que aporte mejores rendimientos, en cuanto a precisión y robustez.

Como se ha especificado en el capítulo anterior, en ambos métodos se ha empleado como función de perdidas la función "Mean Square Error" (MSE loss), y se ha optado por el uso de un optimizador tipo "Adam" para la obtención de la convergencia.

Aparte de para la propia comparación de las dos tipologías de imágenes (segmentada o distancias), estos ensayos también han servido para ajustar el "learning rate" con el que va a trabajar el optimizador, por ello se han realizado ensayos variando dicho valor para ambas estrategias.

En concreto, para la realización ensayos se han escogido tres valores distintos de "learning rate", estos son $5e^{-5}$, $1e^{-4}$ y $5e^{-4}$. Todos estos ensayos han sido repetidos tres veces para estimar la media de los errores producidos por cada combinación. Los resultados medios obtenidos de estos ensayos se reflejan en la tabla 2.

De esta tabla 2, se puede observar que claramente al trabajar con imágenes segmentadas se obtienen mejores precisiones 0.988 para la métrica del IoU por un 0.960 registrado por los ensayos con imágenes de transformación de distancias para la misma métrica, lo que supone un 2,8% más de solapamiento entre las proyecciones del objeto de interés transformado y el usado como etiqueta.

	Error medio (px)	Min. Error (px)	Max. Error (px)	IoU
Segmentada 5e-5	869	148	2172	0.98494
Segmentada 1e-4	**559**	**72**	**1390**	**0.99040**
Segmentada 5e-4	-	-	-	-
Segmentada	**714**	**72**	**2172**	**0.98767**
Distancias 5e-5	2358	757	4441	0.96074
Distancias 1e-4	2373	602	4584	0.96055
Distancias 5e-4	2440	912	4812	0.95935
Distancias	2381	602	4812	0.96039

Tabla 2. Ensayos realizados con imágenes simuladas

Esta diferencia de precisión entre ambos tipos puede deberse, en gran medida, a que el cálculo de la imagen de transformación de distancias sobre las imágenes de Micro 2 se realiza directamente sobre la imagen con el objeto proyectado, por lo que la información que se añade no se encuentra afectada por dicha proyección. Esto puede hacer que las distancias para estos casos no sean del todo reales, aunque si son bastante aproximadas. Además, las distancias calculadas son números enteros al determinarse por los pixeles que hay al borde más cercano, por lo que también se introduce un poco de error al no ser la distancia absoluta, sino una interpolación.

En cuanto a los valores de "learning rate", en primer lugar, cabe destacar que con el mayor de ellos la red es incapaz de converger trabajando con imágenes segmentadas, mientras que sí que lo consigue cuando se utilizan imágenes con transformación de distancias.

Para el resto de valores, aunque no haya tanta diferenciación como el caso de la tipología de imágenes sí que se observa que al trabajar con un "learning rate" de $1e^{-4}$ se obtienen mejores resultados en el caso de segmentada obteniendo 0.990 de precisión para el IoU por el 0.985 que se obtiene para el "learning rate" de $5e^{-5}$. En el caso de transformación de distancias el valor de "learning rate" es prácticamente indiferente obteniendo casi la misma precisión para los tres casos, por lo que nos guiaremos por el ensayo de segmentada, el valor del "learning rate" se fijó en $1e^{-4}$.

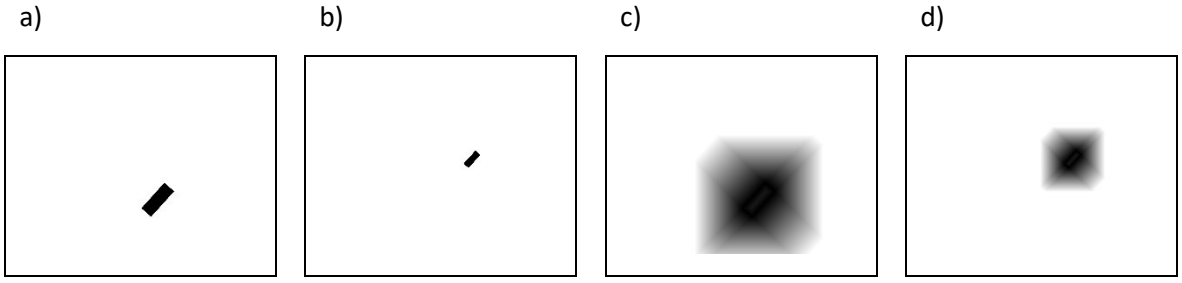


Figura 19. Imágenes dataset simulado ensayo de no superposición. a) imagen Micro 1 recortada segmentada b) Imagen Micro 2 segmentada c) imagen Micro 1 recortada de transformación de distancias d) Imagen Micro 2 de transformación de distancias

Complementariamente a estos ensayos, y para demostrar cuál de los métodos aporta una mayor robustez se ha planteado un ensayo en el cual los rectángulos generados por el simulador para todas

las imágenes de Micro 1 no se superponen en ninguno de los casos con su correspondiente rectángulo en la perspectiva de la cámara de Micro 2.

Para la realización de este ensayo, la configuración del simulador se mantiene invariante, a excepción del valor del tamaño de los rectángulos que forman las imágenes el cual se redujo hasta evitar claramente que se superpusieran, como puede observarse en la Figura 19.

Como aparece detallado en la tabla 3, al no haber superposición entre las segmentaciones de las dos imágenes el ensayo realizado con las imágenes segmentadas es incapaz de converger al no disponer de gradiente inicial, mientras que el realizado con imágenes de transformación de distancias sí que lo es debido a ese extra de información que representan las distancias a los bordes del objeto del interés. En este ensayo se ha alcanzado una precisión de 0.914 para la métrica IoU, cerca de un 5% menos a la obtenida con los rectángulos grandes, debido seguramente a que la información que sí se encuentra afectada por la proyección ha sido reducida.

	Error medio (px)	Min. Error (px)	Max. Error (px)	IoU
Segmentada	-	-	-	-
Distancias	1038	304	2276	0.91445

Tabla 3. Ensayo con imagen simulada sin solapamiento

Con estos ejemplos se demuestra empíricamente que utilizar imágenes de transformación de distancias es más robusto y puede ayudar a la convergencia. Aunque, si se consigue la convergencia, se consiguen mejores precisiones utilizando imágenes segmentadas.

7.2. Nuevos métodos de entrenamiento

Con las conclusiones obtenidas de la comparación realizada con imágenes simuladas, el primer ensayo que se realizó con imágenes de *C. elegans* fue la comprobación de si para el caso de estudio se puede obtener la convergencia empleando imágenes segmentadas. En estos ensayos, al igual que con los ensayos de imágenes simuladas reflejados en las tablas 2 y 3, para obtener una mejor representación de los resultados de las simulaciones estos han sido repetidos. Los resultados de dichos ensayos se muestran en la tabla 4.

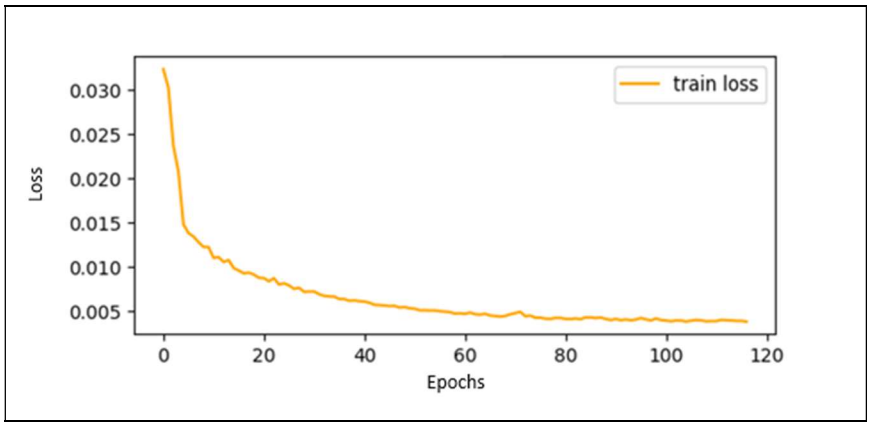
	Error medio (px)	Min. Error (px)	Max. Error (px)	IoU	Tiempo (h)
Ensayo 1	4907	2020	29923	0.86299	3:44:30
Ensayo 2	4968	1978	19600	0.86130	4:01:30
Ensayo 3	4985	2270	29667	0.86184	3:58:00

Tabla 4. Ensayos realizados con imágenes segmentadas de C.elegans

Como refleja la tabla 4, para este caso sí que es capaz de conseguir la convergencia llegando a obtener una precisión del 0.863 para el índice IoU. Pese a ello, durante la realización de los ensayos también se ha observado que en alguno no se consiguió la convergencia, esto puede deberse al bajo rango de superposición inicial entre las segmentaciones de los nematodos. En la Figura 20 a, b se muestran dos gráficas de la evolución de los resultados de la función de pérdidas, la primera de ellas se corresponde

con un caso en el que la red ha sido capaz de converger, mientras que en la segunda se refleja uno en el que no lo ha conseguido.

a)



b)

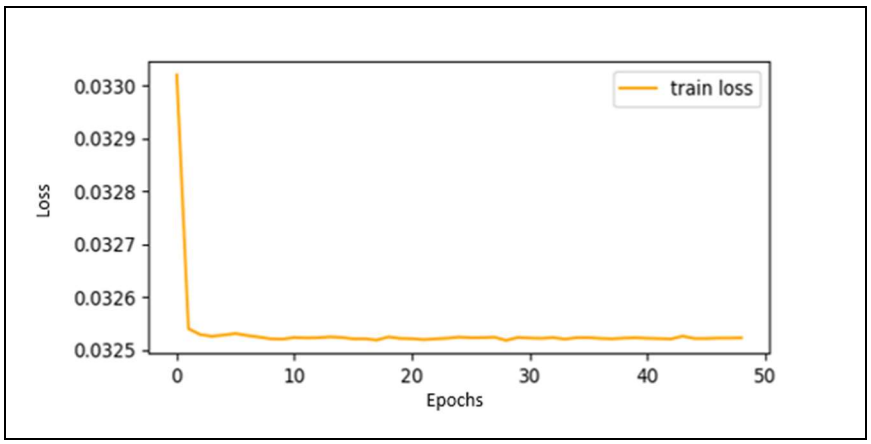


Figura 20. Gráficas de evolución del loss ensayo con imágenes segmentadas. a) Ensayo de convergencia normal b) Ensayo de convergencia donde no se ha conseguido la convergencia

Por ello, y con el fin de aprovechar la robustez del método utilizando las imágenes de distancias sin perder la precisión de las imágenes segmentadas se optó por diseñar una nueva estrategia mixta de entrenamiento en el cual se empleen las dos tipologías de imagen.

La nueva estrategia consiste en que durante las primeras épocas del entrenamiento se empleen imágenes de transformación de distancias y, cuando se alcance un nivel de precisión determinado, se permute a imagen segmentada para alcanzar una mayor precisión (Figura 21), intentando evitar así, los problemas de convergencia. En este caso se ha tomado que el cambio entre imagen de distancias y segmentada suceda cuando la función de pérdidas devuelva un valor inferior a 0.017, lo que se corresponde con valor de IoU en torno a 0.6.

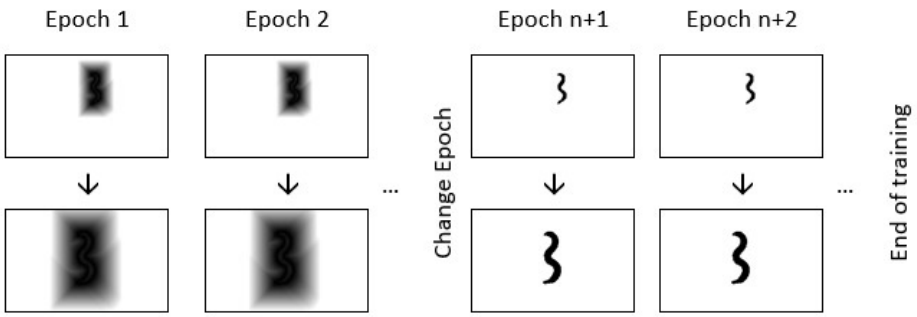


Figura 21. Esquema metodología de entrenamiento mixta

Cabe destacar que, en este método mixto, a parte del cambio en el dataset, se ha realizado un cambio de optimizador en la permutación, ya que, aunque en ambos casos se emplee el mismo tipo de función de pérdidas (MSE loss), el emplear un tipo de imagen distinto supone también una variación del problema de optimización. Por lo que se han empleado dos optimizadores distintos para cada uno de los tipos de imágenes, ambos de tipo Adam y con un learning rate de $1e^{-4}$.

	Error medio (px)	Min. Error (px)	Max. Error (px)	IoU	Tiempo (h)
Ensayo 1	5039	2313	29621	0.85871	4:40:39
Ensayo 2	5474	1999	29501	0.85436	4:16:11
Ensayo 3	5154	2457	17990	0.86132	5:11:40

Tabla 5. Ensayos realizados con la metodología de entrenamiento mixta e imágenes de C. elegans

Como se observa en la tabla 5 esta nueva metodología mixta alcanza valores de precisión semejantes a los obtenidos con imagen segmentada, aunque el costo computacional del entrenamiento es algo más alto. Con esta nueva metodología se consigue aumentar la robustez, aunque cuando ocurre el cambio se produce un aumento del error, llegando en ocasiones a valores de error incluso superiores a los del inicio (Figura 22). Esto puede provocar que, en determinadas ocasiones, el ensayo no sea capaz de converger, al igual que ocurría con el método de imágenes segmentadas.

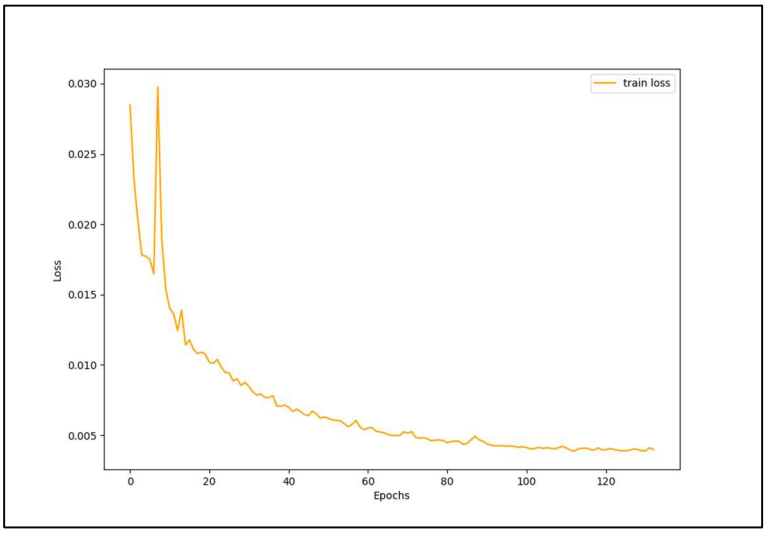


Figura 22. Gráfica de evolución del loss para la metodología de ensayo mixta

Debido a esto se planteó una estrategia mixta modificada. En ella, al igual que en la estrategia mixta existe una diferenciación entre las primeras épocas y el resto. Pero, en este caso, en estas primeras épocas no se entrenan únicamente con imágenes de distancias, sino que se han intercalado épocas con imágenes de distancias y segmentadas (Figura 23).

Cabe destacar que en este nuevo método de entrenamiento también se emplearán dos optimizadores, permutando entre ellos de la misma manera que se permuta en el tipo de imagen utilizada.

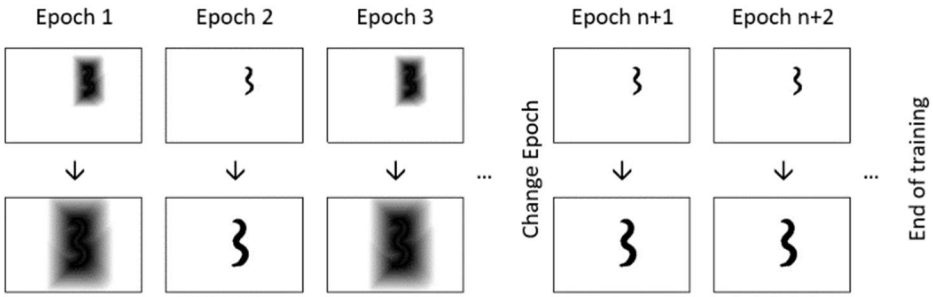


Figura 23. Esquema metodología de entrenamiento mixta modificada

De esta forma, con esta nueva metodología, los pesos de la red obtenidos al final de esta primera fase se adecuan mejor a las características de las imágenes segmentadas y así se evita el aumento excesivo del error al realizar el cambio, como puede observarse en la gráfica de la Figura 24. En este caso al igual que con la metodología mixta el cambio se produce cuando se alcanza el valor para la función de pérdidas de 0.017.

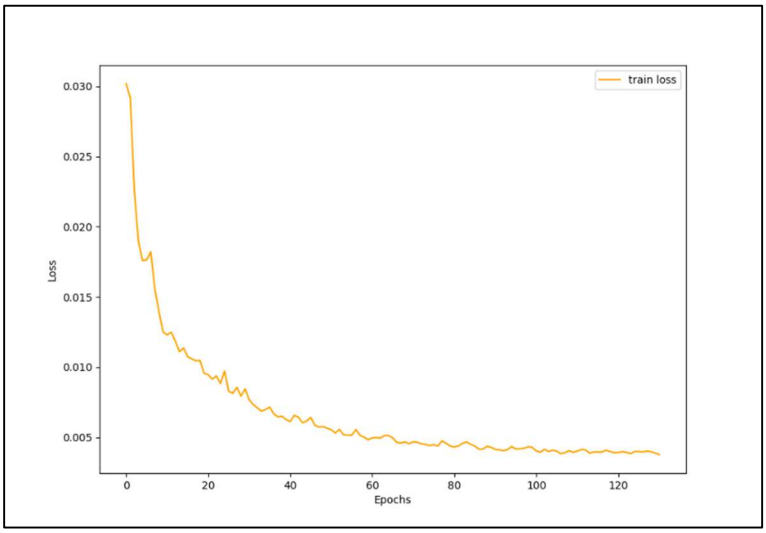


Figura 24. Gráfica de evolución del loss para la metodología de ensayo de entrenamiento mixta modificada

Los datos de estas simulaciones (tabla 6) al igual que con el caso anterior alcanzan unos niveles de precisión semejantes a los obtenidos con imágenes segmentadas, y en este caso, el coste computacional es algo menor que los obtenidos para la metodología mixta, llegando a obtener valores del orden de los ensayos de imágenes segmentadas. Esto último, se debe principalmente al hecho de evitar ese pico de error en la realización del cambio, evitando así retroceder en el aprendizaje.

	Error medio (px)	Min. Error (px)	Max. Error (px)	IoU	Tiempo (h)
Ensayo 1	4936	2097	30088	0.86205	4:02:14
Ensayo 2	5124	2026	18606	0.85914	4:11:54
Ensayo 3	5279	2346	29410	0.85668	5:13:22

Tabla 6. Ensayos realizados con la metodología de entrenamiento mixta modificada e imágenes de C.elegans

Por tanto, esta estrategia consigue una mayor robustez sin afectar en exceso al nivel de precisión ni al costo computacional. Como última prueba, se ha comprobado si esta metodología es capaz de converger también en aquellos casos en los que no se producen solapamientos, para confirmar el aumento de robustez que se ha producido.

En primera instancia los ensayos con esta metodología eran incapaces de converger al emplear el dataset sin solapamiento, ya que al realizar el primer cambio de distancias a segmentada el nivel de solapamiento obtenido no era suficiente y divergía en ese momento. Por ello, se optó por realizar un "warm-up" para el valor del "learning rate" del optimizador que trabaja con imágenes segmentadas, y de esta forma evitar que en las primeras épocas se separase en exceso de la solución obtenida por las imágenes de distancias.

En este caso el valor de inicio del "learning rate" para el optimizador empleado para imágenes segmentadas será de $2e^{-5}$ y posteriormente a cada época realizada con imágenes segmentadas se ha ido incrementado su valor en $2e^{-5}$ hasta llegar al valor de $1e^{-4}$, de esta forma después de las 5 primeras épocas que se empleen estas imágenes, las cuales corresponden a la fase de entrenamiento intercalado con imágenes de distancias, se llega a ese valor.

	Error medio (px)	Min. Error (px)	Max. Error (px)	IoU
Ensayo 1	197	36	434	0.98244
Ensayo 2	170	29	417	0.98486
Ensayo 3	222	27	688	0.98039

Tabla 7. Ensayos realizados con la metodologia de entramiento mixta modificada e imagenes simuladas sin solapamiento

En la tabla 7 se observa que con esta modificación este último método es capaz de converger alcanzando unos niveles de precisión superiores a 0.98 para el IoU, lo que supone alrededor de un 7% de mejora con respecto a los obtenidos para el caso de imágenes de distancias (tabla3). Por tanto, con estos ensayos queda demostrado el aumento de robustez del método junto con la conservación de un nivel de precisión y coste computacional propio de ensayos con imágenes segmentadas.

CAPÍTULO 8. APLICACIONES

En este capítulo se exponen unas aplicaciones propuestas como posibles trabajos futuros sobre las que pueden emplearse los resultados obtenidos en este estudio. En ellas, al igual que en los trabajos más vanguardistas, este tipo de estrategia va a ser utilizada para adecuar las imágenes a las condiciones necesarias para poder utilizarlas.

Se han desarrollado dos aplicaciones sobre las que poder emplear los resultados de la STN. Por un lado, se encuentra el problema de enfoque, este problema surge debido a la diferencia existente en esta característica entre las imágenes de Micro 1 y Micro 2. Y, por otro lado, se encuentra el problema de re-identificación de los nematodos, problemática muy interesante que solucionaría los problemas en el seguimiento de los *C. elegans* al producirse agrupaciones.

Estas dos problemáticas ya se han empezado a desarrollar, por lo que también se muestran los primeros resultados obtenidos para ambos casos.

8.1. Problema de enfoque

Como se observa en las imágenes capturadas por las dos cámaras (Figura 10) existe, además de un posicionamiento distinto por parte de los nematodos, una clara diferenciación en cuanto al enfoque de ambas imágenes. La imagen capturada por Micro 1 es claramente más nítida que la obtenida por la cámara de Micro 2. Esto seguramente puede deberse al posicionamiento de ambas cámaras y a su interacción con el divisor de luz que se monta en el carro.

Para plantear la resolución del problema de enfoque se necesita disponer de la misma imagen enfocada y desenfocada, la primera de ellas sirve en este caso como imagen objetivo a la que se quiere llegar, mientras que la segunda, la imagen desenfocada, se utiliza como entrada para la red empleada.

Es aquí donde entra la solución obtenida por las STN. El objetivo es emplear como etiquetas las imágenes de Micro 1 y usar de entrada a la red las imágenes transformadas por la STN, que corresponde a la cámara Micro 2, creando así el dataset para los ensayos.

En cuanto a la red que se va a emplear para esta nueva tarea, se optó por una red tipo "encoder-decoder". Este tipo de redes se caracterizan porque la salida de la red es otra imagen que normalmente posee el mismo tamaño que la imagen de entrada, lo cual es condición necesaria para este problema.

A las redes "encoder-decoder" también se les conoce como "U-net" debido a la estructura de estas redes en forma de "U" (Figura 25). Esta estructura se divide en dos partes, en la primera de ellas que corresponde con el "encoder" se reduce el tamaño de las capas, aunque aumenta los canales de salida, y, por tanto, su profundidad, y en la segunda que es el "decoder" se realiza el proceso contrario, se va

aumentando el tamaño de las capas, pero se reduce su profundidad, hasta que quede únicamente una capa que se corresponde con la imagen de salida.

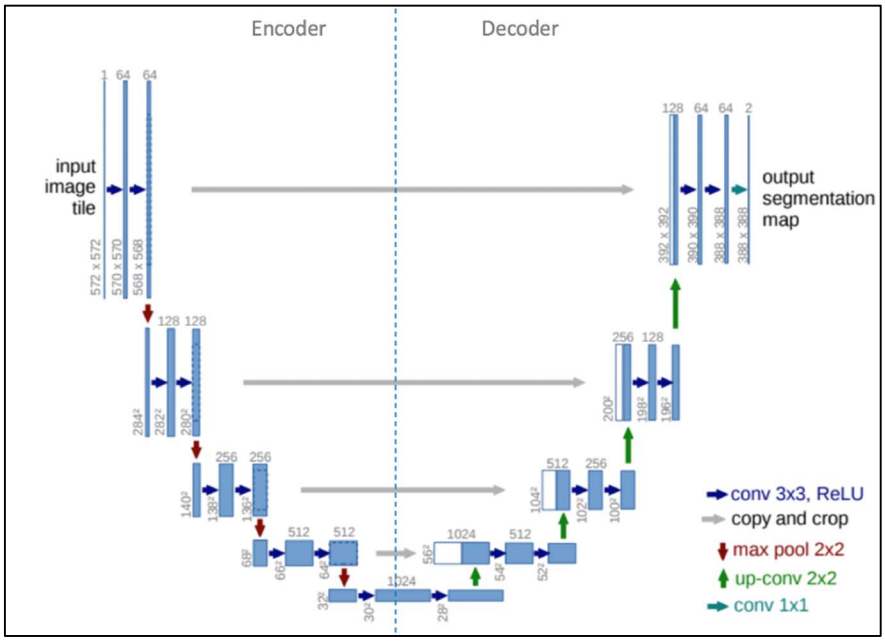


Figura 25. Ejemplo red neuronal "encoder-decoder" [I.10]

En cuanto a la función de pérdidas, se han probado dos tipos distintos de ellas. Primero se optó por emplear la función "MSE loss", y, después, como lo que se necesita que se enfoque es el nematodo y no el fondo, se planteó realizar una modificación sobre la función "MSE loss" de forma que solo se fijase en los pixeles que lo forman.

Una vez seleccionados todos estos aspectos con los que se van a trabajar se han realizado los primeros ensayos obteniendo los resultados mostrados en la Figura 26.

Los resultados mostrados corresponden a un ensayo realizado con la función de pérdidas customizada para que la red únicamente se centre en el enfoque del nematodo. En ellos se puede observar que el este ha ganado algo de definición en determinadas zonas, por lo que esta solución deja entrever que es posible conseguir abordar esta tarea.

Además, en las imágenes de pseudocolor, aunque se observan diferencias entre la imagen devuelta por la red y la imagen objetivo sí que se aprecia la aparición de algunos colores nuevos que en la imagen de entrada a la red no se encuentran.

En cuanto al fondo si se puede observar que dista mucho con respecto al de la imagen objetivo, pero esto es algo que ya se sabía que iba a pasar puesto que es una zona sobre la cual no se tiene el control.

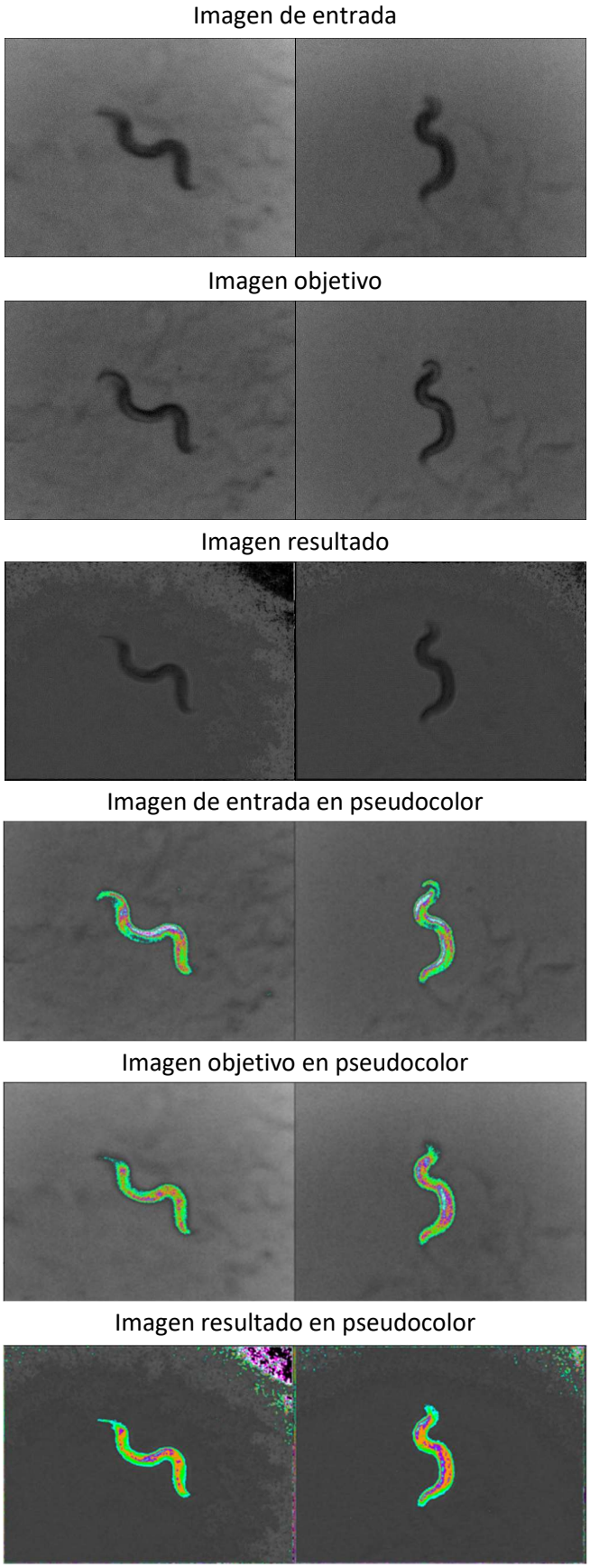


Figura 26. Resultados problema de enfoque

8.2. Problema de re-identificación

La tarea de re-identificación planteada surge debido a los problemas observados para realizar el seguimiento de los nematodos cuando se producen colisiones entre nematodos o se colocan en paralelo, situaciones que pueden llegar a darse en determinadas ocasiones durante la vida de estos nematodos. Es por ello que, si se consigue que la red obtenga características diferenciadoras entre estos, pueda reconocerse cual es cada uno de ellos una vez se haya disuelto ese agrupamiento y continuar con el seguimiento del nematodo correcto.

El trabajo desempeñado aquí por la STN serviría para aumentar el dataset capturado, ya que se disponen de muy pocas imágenes por cada nematodo, además aquí nuevamente no se pueden aprovechar las imágenes capturadas de una misma placa en los diferentes días, ya que no puede asegurarse que los nematodos capturados sean los mismos para los dos días, y tampoco que todos ellos sean distintos.

En este problema también podrían llegar a emplearse los resultados de la solución del problema de enfoque, aunque disponer de imágenes de los nematodos con diferentes niveles de nitidez también puede aportar información relevante y lo haría más robusto ante cambios en esta característica.

El dataset en este caso pasaría a estar formado por las imágenes de Micro 1 y las imágenes de Micro 2 transformadas por las STN, las cuales todas ellas servirían de entrada a la red y como etiqueta se usaría un número de identificación que se le aportó a las imágenes de cada nematodo para poder diferenciarlas.

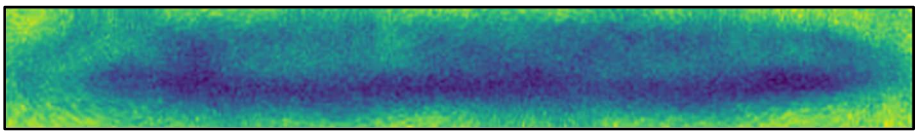
En una primera instancia se realizaron ensayos que dieron buenos resultados, pero se observó que esto podría ser debido a que, como las imágenes capturadas formaban parte de una sola secuencia en la que el nematodo capturado podría moverse poco, la red aprendía información de la forma que tenían los nematodos en vez de características internas de los mismos, algo que se quiere evitar.

Fue por esto último por lo que a las imágenes de entrada se les decidió aplicar una transformación tipo TPS para colocar todos los nematodos en la misma posición (estirados horizontalmente), evitando así que la red de re-identificación pueda aprender información proveniente de la forma adoptada por los nematodos. También se realizó un recorte ajustado al tamaño de los nematodos para disminuir el tamaño de la imagen y la cantidad de fondo que aparecía en ellas.

En adición a todo esto, se han utilizado diferentes técnicas de aumento de datos para obtener una mayor variabilidad en las imágenes de entrada y así disponer de un dataset más amplio sin necesidad de realizar más capturas. Entre las técnicas que se han empleado se encuentran la realización de "flips" (volteos) horizontales y verticales de forma aleatoria y la oclusión de parte de la imagen con rectángulos de color negro, para que pueda aprender a identificar los nematodos sin necesidad de disponer de toda la información. Una muestra de estas imágenes se expone en la Figura 27.

Para realizar una primera estimación de los resultados que son capaces de obtenerse con este nuevo proyecto se han realizado pruebas con una red neuronal muy sencilla formada por 2 capas convolucionales con su correspondiente capa de activación, en este caso tipo "ReLU", y capa de "maxpooling", 2 funciones de regularización tipo "DropOut" y 2 capas lineales con una nueva capa de activación "ReLU" como paso intermedio entre ambas.

a)



b)

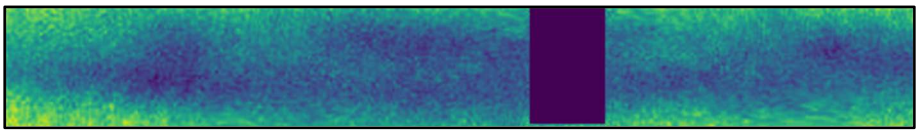


Figura 27. Imágenes del dataset para el problema de re-identificación representadas en pseudocolor. A) Imagen sin aumento de datos b) Imagen con aumento de datos

Cabe destacar que en esta problemática la función de pérdidas que se ha empleado ha sido la "Triplet loss" (ecuación 8), este tipo de función de perdidas es muy utilizada para temas de re-identificación. Para trabajar con ella se debe de agrupar el dataset en ternas de imágenes, estas tienen que contener dos imágenes de la misma identidad, una de estas servirá de ancla y la otra como muestra de la misma identidad que el ancla, y otra perteneciente a una identidad distinta. En esta función de pérdidas lo que se busca es minimizar la distancia de los resultados obtenidos por las dos imágenes de la misma identidad, maximizando la misma entre los resultados de la imagen ancla y de la que es de una identidad distinta.

$$\mathcal{L}_{triplet} = \frac{1}{N} \cdot \sum_{i=1}^{N} \max\left\{ d_{(a,p)}^{(i)} - d_{(a,n)}^{(i)} + m, 0 \right\} \qquad (8)$$

Donde "N" es el número de lotes del entrenamiento, $d_{(a,p)}^{(i)}$ y $d_{(a,n)}^{(i)}$ son las distancias entre las soluciones del ancla y la muestra de la misma identidad (positiva) y entre las soluciones del ancla y la muestra de otra identidad (negativa), respectivamente, y "m" es un margen de seguridad que marca la distancia mínima que puede haber entre la negativa más cercana al ancla y la positiva más alejada.

Con todos estos ajustes se han realizado los primeros ensayos que han dado unos resultados bastante buenos. En la Figura 28, se puede observar los resultados obtenidos en la fase de entrenamiento.

En esta imagen se observa una gráfica del posicionamiento final de todas las muestras empleadas durante el entrenamiento representadas por nubes de puntos de colores distintos. En ellas vemos una clara tendencia a formar grupos diferenciados, aunque sí que hay algunos casos en los que se entremezclan.

Para verificar el aprendizaje de la red se ha recurrido a la fase de validación empleando un grupo de imágenes diferentes a las que se han usado durante el entrenamiento, como ya se había explicado en el capítulo 2.

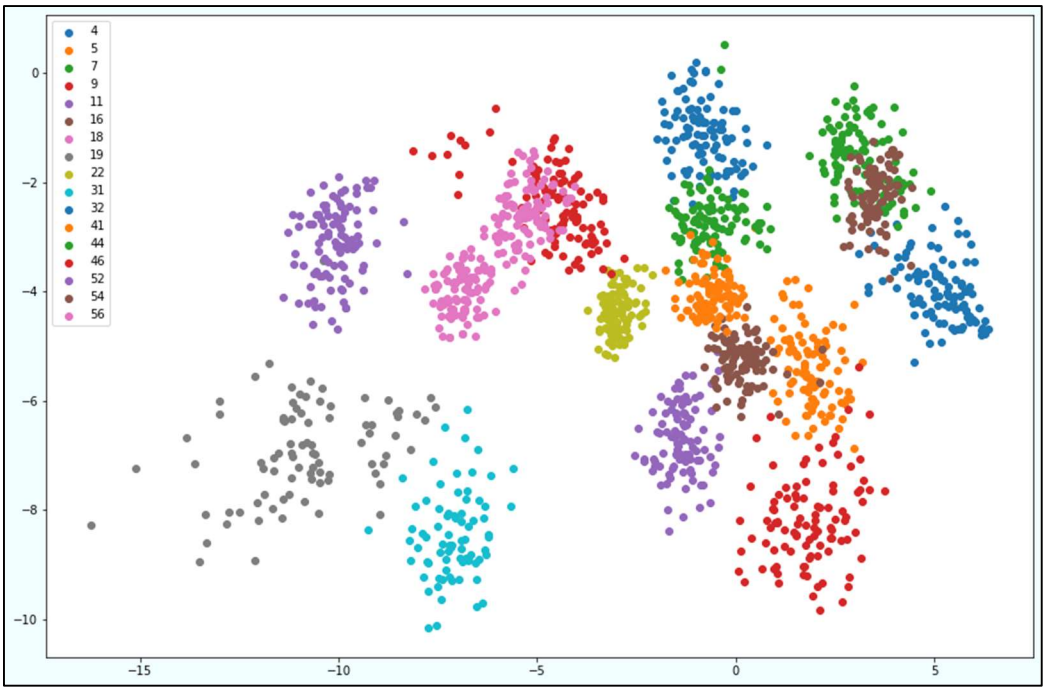


Figura 28. Resultados del entrenamiento de re-identificación

Con la finalidad de estudiar los resultados se han seleccionado una serie de índices que los reflejen, el primero de ellos ha sido una matriz de confusión adaptada al caso de estudio, ya que la matriz de confusión normal abarca solamente la disyunción entre dos alternativas.

Identidades etiquetadas \ Identidades predichas	4	5	7	9	11	16	18	19	22	31	32	41	44	46	52	54	56
4	18	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	18	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
7	1	0	17	0	0	0	0	0	0	0	0	2	0	0	0	0	0
9	0	1	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	15	3	0	0	0	0	0	2	0	0	0	0	0
16	0	3	0	0	2	15	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	16	0	1	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	17	0	0	1	0	2	0	0	0
31	0	0	0	0	0	0	0	1	0	15	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	4	0
41	0	0	2	0	0	1	0	0	0	0	0	17	0	0	0	0	0
44	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	5	0
46	0	0	0	0	0	0	0	0	1	0	0	0	0	14	0	0	5
52	0	0	0	0	0	0	0	1	0	0	0	0	0	0	19	0	0
54	0	0	0	0	0	0	0	0	0	0	1	0	4	0	0	14	0
56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20

Tabla 8. Matriz de confusión de identidades para el problema de re-identificación

Como se observa en la tabla 8, la mayor parte de los nematodos los consigue identificar correctamente, obteniendo un 70% de acierto en el peor de los casos (etiqueta 46). En esta matriz también se pueden ver reflejados los resultados aportados en la gráfica de la Figura 28, por ejemplo, entre las etiquetas 54 y 44 se observa que hay una gran confusión, lo cual ya se observaba en la gráfica ya que están representados por los dos grupos de puntos de la esquina superior derecha.

Por otro lado, también se han obtenido resultados en términos globales calculando el porcentaje de acierto total, el cual es de un 85.84%.

Para ratificar estos buenos resultados se ha empleado una herramienta de Python denominada "Captum". Esta herramienta analiza en que zonas de la imagen hacen efecto los pesos más significativos, es decir, en donde se fija mayormente la red para decidir la identidad del *C. elegans*.

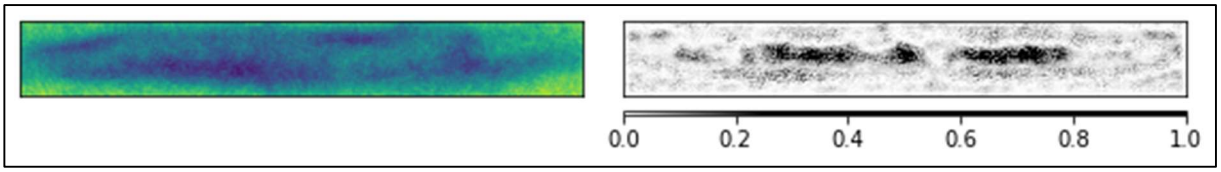


Figura 29. Resultado análisis de "Captum"

En la Figura 29 se muestra el resultado devuelto por esta herramienta. En esta figura se observa a la izquierda la imagen de entrada en pseudocolor y a la derecha un gráfico en el cual se marca con mayor intensidad los píxeles más significativos mientras que se quedan en blanco los que aportan menos información. De esta última se puede apreciar como la mayor parte de los puntos significativos se encuentran en la parte central del gusano, dejando entrever que la información que aprende la red es información interna del gusano y no de irregularidades en su contorno, lo cual es una gran señal ya que puede indicar que los nematodos estudiados poseen rasgos característicos que los diferencian unos de otros.

CAPÍTULO 9. CONCLUSIONES

En este trabajo se buscaba encontrar una metodología que fuese capaz de solucionar el problema de correspondencia empleando una red neuronal de transformación espacial sobre imágenes en las cuales el nivel de solapamiento entre los diferentes puntos de vista de los objetos era bastante reducido, o llegando a ser incluso nulo para determinados casos.

En los ensayos mostrados se han planteado una serie de técnicas de entrenamiento para dar solución a este problema, basadas en la incorporación de imágenes de transformación de distancias al entrenamiento, diferenciándose en el uso que le dan.

Viendo los resultados aportados por los ensayos realizados puede afirmarse que se ha conseguido establecer una nueva metodología de entrenamiento capaz de afianzar la convergencia para aquellos casos en los que el nivel de superposición de las dos perspectivas de los objetos sea reducido e incluso nulo.

Por tanto, se ha demostrado empíricamente que la incorporación de imágenes de transformación de distancias al entrenamiento, intercalando épocas que usan este tipo de imágenes con épocas que empleen imágenes segmentadas durante la primera fase del mismo, consigue solventar el problema de correspondencia con una mayor robustez impidiendo que se produzcan problemas durante la convergencia inicial.

Además, esta solución consigue alcanzar el nivel de precisión propio de los ensayos con imágenes segmentadas y sin aumentar en exceso el costo computacional del entrenamiento.

En cuanto a las aplicaciones de los resultados obtenidos por esta nueva metodología los resultados que se han obtenido han sido positivos, y, aún más sabiendo que estos se tratan de entrenamientos preliminares para comprobar la viabilidad de las problemáticas.

Esto es especialmente notable en el caso del problema de re-identificación en el cual se alcanza una precisión superior al 85% para una configuración bastante sencilla.

Aparte del cumplimiento con el objetivo principal del proyecto y de la obtención de buenos resultados en las aplicaciones para las que se han testeado, también se ha logrado cumplir con todos los objetivos secundarios del mismo. Aumentando el conocimiento sobre redes neuronales convolucionales y todos los elementos que las rodean durante su realización, así como con el lenguaje de programación de Python y sus librerías de funciones.

CAPÍTULO 10. TRABAJOS FUTUROS

En este apartado se busca plantear diversas opciones para la ampliación de este trabajo en busca de mejorar los resultados obtenidos. Entre ellas se citan posibilidades tanto para el trabajo con las STN como para las aplicaciones que se han expuesto. Estas son:

1. Plantear la utilización de transformaciones tipo TPS para los ensayos de convergencia de la STN, aunque pueda ser un método más costoso inicialmente al tener que definir determinados puntos que corresponder puede que sea más efectivo y necesite menos tiempo de entrenamiento.
2. Plantear la resolución del problema de correspondencia con un mayor número de imágenes aumentado aún más la variabilidad del dataset.
3. Realizar un mejor ajuste de hiperparámetros para el problema de enfoque, así como hasta un cambio en la tipología o la profundidad de la red o la tipología de algunos de sus elementos como el optimizador o la función de pérdidas empleada.
4. Testear una red más profunda con más complejidad para el caso de re-identificación que pueda aprender un mayor número de características de los *C. elegans* con la finalidad de que la diferenciación de estos sea mucho más efectiva.
5. Realizar experimentos de re-identificación con más tipologías de aumento de datos para hacer los resultados más sólidos frente a diferentes cambios en las imágenes.
6. Afinar el ajuste de los hiperparámetros empleados para el problema de re-identificación, como el tipo de función de pérdidas o el optimizador y el "learning rate".

CAPÍTULO 11. BIBLIOGRAFÍA

[A.1] Di Rosa G, Brunetti G, Scuto M, et al. "Healthspan Enhancement by Olive Polyphenols in *C. elegans* Wild Type and Parkinson's Models." *Int J Mol Sci*. (2020). https://doi.org/10.3390/ijms21113893.

[A.2] Hahm JH, Kim S, DiLoreto R, et al. "C. elegans maximum velocity correlates with healthspan and is maintained in worms with an insulin receptor mutation." *Nat Commun*. (2015). https://doi.org/10.1038/ncomms9919.

[A.3] Onken B, Driscoll M. "Metformin induces a dietary restriction-like state and the oxidative stress response to extend C. elegans Healthspan via AMPK, LKB1, and SKN-1." *PLoS One*. (2010). https://doi.org/10.1371/journal.pone.0008758.

[A.4] Puchalt JC, Sánchez-Salmerón AJ, Ivorra E, Genovés Martínez S, Martínez R, Martorell Guerola P. "Improving lifespan automation for Caenorhabditis elegans by using image processing and a post-processing adaptive data filter." *Scientific Reports* (2020). https://doi.org/10.1038/s41598-020-65619-4.

[A.5] Puchalt JC, Sánchez-Salmerón AJ, Martorell Guerola P, Genovés Martínez S. "Active backlight for automating visual monitoring: An analysis of a lighting control technique for Caenorhabditis elegans cultured on standard Petri plates." *PLoS One*. (2019). https://doi.org/10.1371/journal.pone.0215548.

[A.6] Puchalt JC, Sánchez-Salmerón AJ, Ivorra E, Llopis S, Martínez R, Martorell P. "Small flexible automated system for monitoring Caenorhabditis elegans lifespan based on active vision and image processing techniques." *Scientific Reports*. (2021). https://doi.org/10.1038/s41598-021-91898-6.

[A.7] Puchalt JC, Layana Castro PE, Sánchez-Salmerón AJ. "Reducing Results Variance in Lifespan Machines: An Analysis of the Influence of Vibrotaxis on Wild-Type *Caenorhabditis elegans* for the Death Criterion." *Sensors (Basel)*. (2020). https://doi.org/10.3390/s20215981.

[A.8] Jaderberg, Max, K. Simonyan, Andrew Zisserman and K. Kavukcuoglu. "Spatial Transformer Networks." *NIPS* (2015). arXiv:1506.02025 [cs.CV].

[A.9] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, Tsuhan Chen. "Recent advances in convolutional neural networks." *Pattern Recognition, Volume 77*, (2018): 354-377. https://doi.org/10.1016/j.patcog.2017.10.013.

[A.10] Rosenblatt, F. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review* vol. 65,6 (1958): 386-408. https://doi.org/10.1037/h0042519.

[A.11] T. Wang, D.J. Wu, A. Coates, A.Y. Ng. "End-to-end text recognition with convolutional neural networks" *Proceedings of the International Conference on Pattern Recognition (ICPR*), (2012): 3304–3308. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6460871.

[A.12] Y. Boureau, J. Ponce, Y. LeCun. "A theoretical analysis of feature pooling in visual recognition" *Proceedings of the International Conference on Machine Learning (ICML),* (2010): 111–118. https://icml.cc/Conferences/2010/papers/638.pdf.

[A.13] Y.A. LeCun, L. Bottou, G.B. Orr, K.-R. Müller. "Efficient backprop." *Neural Networks: Tricks of the Trade - Second Edition*, (2012): 9–48. https://doi.org/10.1007/978-3-642-35289-8_3.

[A.14] V. Nair, G.E. Hinton. "Rectified linear units improve restricted Boltzmann machines." *Proceedings of the International Conference on Machine Learning (ICML),* (2010): 807–814. https://icml.cc/Conferences/2010/papers/432.pdf.

[A.15] A. L. Maas, A. Y. Hannun, A. Y. Ng. "Rectifier nonlinearities improve neural network acoustic models." *Proceedings of the International Conference on Ma- chine Learning (ICML)*, 30, (2013). http://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[A.16] D. A. Clevert, T. Unterthiner, S. Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)." *Proceedings of the International Conference on Learning Representations (ICLR)*, (2016). arXiv:1511.07289 [cs.LG].

[A.17] U. Shaham, R.R. Lederman. "Learning by coincidence: siamese networks and common variable learning." *Pattern Recognit* (2017). https://doi.org/10.1016/j.patcog.2017.09.015.

[A.18] W. Liu, Y. Wen, Z. Yu, M. Yang. "Large-margin softmax loss for convolutional neural networks" *Proceedings of the International Conference on Machine Learning (ICML),* (2016): 507–516. arXiv:1612.02295 [stat.ML].

[A.19] F. Schroff, D. Kalenichenko, J. Philbin. "Facenet: a unified embedding for face recognition and clustering" *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (2015): 815–823. https://doi.org/10.1109/CVPR.2015.7298682.

[A.20] R.G.J. Wijnhoven, P.H.N. de With. "Fast training of object detection using stochastic gradient descent." *International Conference on Pattern Recognition (ICPR)*, (2010): 424–427. https://doi.org/10.1109/ICPR.2010.112.

[A.21] G.E. Hinton, N. Srivastava, A.Krizhevsky, I.Sutskever, R.R.Salakhutdinov "Im proving neural networks by preventing co-adaptation of feature detectors." *CoRR* abs/1207.0580 (2012). arXiv:1207.0580 [cs.NE].

[A.22] L. Wan, M. Zeiler, S. Zhang, Y.L. Cun, R. Fergus. "Regularization of neural networks using dropconnect." *Proceedings of the International Conference on Machine Learning (ICML)* (2013): 1058–1066. http://proceedings.mlr.press/v28/wan13.pdf.

[A.23] Li, Gen, Shikun Xu, X. Liu, L. Li and C. Wang. "Jersey Number Recognition with Semi-Supervised Spatial Transformer Network." *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2018): 1864-18647. https://doi.org/10.1109/CVPRW.2018.00231.

[A.24] García, Á. A., Juan Antonio Álvarez-García and L. M. Soria-Morillo. "Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods." *Neural networks: the official journal of the International Neural Network Society* 99 (2018): 158-165. https://doi.org/10.1016/j.neunet.2018.01.005.

[A.25] Aubreville, M., Maximilian Krappmann, C. Bertram, R. Klopfleisch and A. Maier. "A Guided Spatial Transformer Network for Histology Cell Differentiation." *VCB*M (2017). arXiv:1707.08525 [cs.CV].

[A.26] Lin, Y., Menglin Wang, Chao Gu, Jiafeng Qin, Demeng Bai and J. Li. "A Cascaded Spatial Transformer Network for Oriented Equipment Detection in Thermal Images." *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)* (2018): 1-5. https://doi.org/10.1109/EI2.2018.8582248.

[A.27] Qian, Yeqiang, Ming Yang, X. Zhao, Chunxiang Wang and Bing Wang. "Oriented Spatial Transformer Network for Pedestrian Detection Using Fish-Eye Camera." *IEEE Transactions on Multimedia* 22 (2020): 421-431. https://doi.org/10.1109/TMM.2019.2929949.

[A.28] Zhang, Xinsheng, Teng Gao and Dongdong Gao. "A new deep spatial transformer convolutional neural network for image saliency detection." *Design Automation for Embedded Systems* 22 (2018): 243-256. https://doi.org/10.1007/s10617-018-9209-0.

[A.29] Fang, Yanyan, Biyun Zhan, Wandi Cai, Shenghua Gao and B. Hu. "Locality-Constrained Spatial Transformer Network for Video Crowd Counting." *2019 IEEE International Conference on Multimedia and Expo (ICME)* (2019): 814-819. https://doi.org/10.1109/ICME.2019.00145.

[A.30] Li, Siyuan, Semih Günel, Mirela Oštrek, Pavan Ramdya, P. Fua and Helge Rhodin. "Deformation-Aware Unpaired Image Translation for Pose Estimation on Laboratory Animals." *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020): 13155-13165. https://doi.org/10.1109/CVPR42600.2020.01317.

[A.31] Bhagavatula, Chandrasekhar, Chenchen Zhu, K. Luu and M. Savvides. "Faster than Real-Time Facial Alignment: A 3D Spatial Transformer Network Approach in Unconstrained Poses." *2017 IEEE International Conference on Computer Vision (ICCV)* (2017): 4000-4009. https://doi.org/10.1109/ICCV.2017.429.

[I.1] López Michelone, M. Imagen de *C. elegans.* (2017). Fecha de acceso: 11/07/2021. Disponible en: https://www.unocero.com/ciencia/avances-con-c-elegans-un-gusano-simulado-con-redes-neuronales/.

[I.2] Wikipedia. Perceptrón multicapa (2004). Fecha de acceso: 28/06/2021. Disponible en: https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa.

[I.3] Torres, J. Imagen esquema red neuronal convolucional (2018). Fecha de acceso: 05/07/2021. Disponible en: https://torres.ai/deep-learning-inteligencia-artificial-keras/.

[I.4] Wikipedia, la enciclopedia libre. Sigmoid Function, Gráfica "Sigmoid" (2008). Fecha de acceso: 29/06/2021. Disponible en: https://en.wikipedia.org/wiki/Sigmoid_function.

[I.5] Neural networks activation functions. Gráfica "Tanh" (2021). Fecha de acceso: 29/06/2021. Disponible en: https://security.kiwi/docs/neural-network-activation-functions/.

[I.6] Neural networks activation functions. Gráfica "ReLU" (2021). Fecha de acceso: 29/06/2021. Disponible en: https://security.kiwi/docs/neural-network-activation-functions/.

[I.7] Jaderberg, Max. Esquema red neuronal convolucional de transformación espacial (2015). Fecha de acceso: 25/06/2021. Disponible en: "Spatial Transformer Networks." *NIPS*.

[I.8] Puchalt, J. Robot cartesiano (2021). Imagen perteneciente a la solicitud de patente en OEPM con referencia P202130492.

[I.9] Puchalt, J. Configuración del carro y el cabezal del robot cartesiano (2021). Imagen perteneciente a la solicitud de patente en OEPM con referencia P202130492.

[I.10] Ronneberger, O. Esquema red neuronal "Encoder-decoder" (2015). Fecha de acceso: 11/06/2021. Disponible en: "U-Net: Convolutional Networks for Biomedical Image Segmentation." *LNCS*. https://doi.org/10.1007/978-3-319-24574-4_28.

[O.1] Wikipedia, la enciclopedia libre. *Caenorhabditis elegans* (2001). Fecha de acceso: 04/07/2021. Disponible en: https://es.wikipedia.org/wiki/Caenorhabditis_elegans.

[O.2] Patente del robot cartesiano. Disponible en la solicitud de patente en OEPM con referencia P202130492.

[O.3] Documento "Recomendaciones en la elaboración de presupuestos en actividades de I+D+I" (2018). Fecha de acceso: 12/07/2021. Disponible en: https://wiki.upv.es/confluence/download/attachments/265093261/Guia_Presupuesto_2018.pdf?version=1&modificationDate=1557920979734&api=v2.

Parte II

Presupuesto

CAPÍTULO 12. PRESUPUESTO

Aquí se encuentra detallado la evaluación del costo económico del proyecto para su total desarrollo, teniendo en cuenta que no se dispone inicialmente de ninguno de los recursos necesarios para llevarlo a cabo.

La unidad monetaria empleada para la realización del presupuesto ha sido el EURO (€), por tanto, como la unidad mínima de medida es el CÉNTIMO DE EURO (0.01€) todas las cantidades presupuestadas han sido redondeadas a la segunda cifra decimal.

El tiempo de desarrollo de este trabajo se ha estimado en un total de 300 horas de trabajo. Y para la estimación de todos los precios y amortizaciones se han seguido las indicaciones reflejadas en el documento "Recomendaciones en la elaboración de presupuestos en actividades de I+D+I" [O.3].

12.1. Mano de obra

Para la realización del proyecto se ha contado con la participación a tiempo total de un titulado superior en ingeniería industrial, el cual cuenta con un precio por hora estimado de 25 €/h y de un catedrático de universidad a tiempo parcial cuyo coste temporal es de 51.4 €/h.

12.2. Materiales y software

En la realización de este proyecto se han utilizado una serie de materiales, equipos y softwares cuya vida útil o tiempo de licencia respectivamente, es superior al tiempo de desarrollo del proyecto, por lo que solamente se ha computado la parte proporcional del coste en función del tiempo de empleo. También se tendrá en cuenta el cálculo del presupuesto el periodo de amortización de estos elementos siguiendo los criterios que se reflejan en la tabla 9.

Clasificación económica del gasto	Amortización (años)
Adquisición de equipo para procesos de información	6
Adquisición de aplicaciones informáticas	6
Adquisición de equipos didácticos y de investigación	10

Tabla 9. Amortizaciones

Cabe destacar que la mayoría de los softwares que se han empleado en el desarrollo de este proyecto son de libre acceso, por lo que el coste de su uso será nulo, de igual forma aparecen detallados en todas las tablas del presupuesto. En la tabla 10 se reflejan todos los precios unitarios de los elementos empleados.

Equipo/Software	Descripción	Precio total (€)	Precio unitario (€/h)
Ordenador de sobremesa	Ordenador con CPU Intel core i7 9700 K 3.6 GHz, GPU NVIDIA GEFORCE RTX 2080 Ti y sistema operativo Ubuntu	2.500,00 €	0,21 €/h (Amortización en 6 años y utilización durante una jornada laboral de 8 horas durante los 253 días laborables del año)
Ordenador portátil	Ordenador portátil con CPU Intel core i7 10750 H 2.6 GHz, GPU NVIDIA GEFORCE RTX 2060 y sistema operativo Windows 10	1.100,00 €	0,09 €/h (Amortización en 6 años y utilización durante una jornada laboral de 8 horas durante los 253 días laborables del año)
Robot cartesiano	Robot cartesiano diseñado y construido por miembros del equipo del Instituto de Automática e informática Industrial	22.000,00 €	1,09 €/h (Amortización en 10 años y utilización durante una jornada laboral de 8 horas durante los 253 días laborables del año)
Microsoft Office	Suite ofimática con diversas aplicaciones de escritorio como Word o Excel	99,00 € (licencia anual)	0,05 €/h (Amortización en 6 años y utilización durante una jornada laboral de 8 horas durante los 253 días laborables del año)
PyCharm	Entorno de desarrollo integrado de software libre especializado en el lenguaje de programación de Python	0,00 €	0,00 €
Google Colab	Software de trabajo en la nube basado en notebooks de Jupyter con acceso a GPU y TPU de forma gratuita	0,00 €	0,00 €
AnyDesk	Software de control remoto	0,00 €	0,00 €

Tabla 10. Precios unitarios de equipos y softwares

12.3. Cuadro de precios descompuestos

STN001 h Búsqueda de información

Búsqueda de información sobre redes neuronales convolucionales, todos los elementos que la componen y ejemplos para plantear las problemáticas sobre las que se trabajan

Código	Unidad	Descripción	Rendimiento	Precio unitario (€/h)	Importe (€/h)
1		**Materiales**			
	h	Ordenador portátil	1,000	0,09	0,09
2		**Mano de obra**			
	h	Titulado superior en ingeniería industrial	1,000	25,00	25,00
	h	Catedrático de universidad	0,200	51,40	10,28
			Subtotal materiales y mano de obra:		**35,37**
3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	35,37	0,71
				Costes directos (1+2+3)**:**	**36,08**

STN002 h Adquisición de imágenes

Adquisición de imágenes de *C. elegans* en el laboratorio por medio del robot cartesiano

Código	Unidad	Descripción	Rendimiento	Precio unitario (€/h)	Importe (€/h)
1		**Materiales**			
	h	Ordenador de sobremesa	1,000	0,21	0,21
	h	Robot cartesiano	0,900	1,09	0,98
2		**Mano de obra**			
	h	Titulado superior en ingeniería industrial	1,000	25,00	25,00
	h	Catedrático de universidad	0,400	51,40	20,56
			Subtotal materiales y mano de obra:		**46,75**
3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	46,75	0,94
				Costes directos (1+2+3)**:**	**47,69**

STN003 h Implementación de los códigos

Desarrollo de los códigos de entrenamiento de la red neuronal convolucional de la STN y de las aplicaciones en las que se han empleado los resultados de la misma

Código	Unidad	Descripción	Rendimiento	Precio unitario (€/h)	Importe (€/h)
1		**Materiales**			
	h	Ordenador de sobremesa	0,500	0,21	0,11
	h	Ordenador portátil	1,000	0,09	0,09
	h	Google Colab	0,500	0,00	0,00
	h	AnyDesk	0,500	0,00	0,00
	h	PyCharm	0,500	0,00	0,00
2		**Mano de obra**			
	h	Titulado superior en ingeniería industrial	1,000	25,00	25,00
	h	Catedrático de universidad	0,200	51,40	10,28
			Subtotal materiales y mano de obra:		**35,48**
3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	35,48	0,71
				Costes directos (1+2+3)**:**	**36,19**

STN004 h Entrenamiento de las redes neuronales

Entrenamiento de las redes neuronales convolucionales desarrolladas para la solución de las problemáticas planteadas

Código	Unidad	Descripción	Rendimiento	Precio unitario (€/h)	Importe (€/h)
1		**Materiales**			
	h	Ordenador de sobremesa	7,000	0,21	1,47
	h	Ordenador portátil	2,000	0,09	0,18
	h	Google Colab	2,000	0,00	0,00
	h	AnyDesk	7,000	0,00	0,00
	h	PyCharm	7,000	0,00	0,00
2		**Mano de obra**			
	h	Titulado superior en ingeniería industrial	1,000	25,00	25,00
	h	Catedrático de universidad	0,100	51,40	5,14
			Subtotal materiales y mano de obra:		**31,79**
3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	31,79	0,64
				Costes directos (1+2+3)**:**	**32,43**

STN005 h Estudio de los resultados

Estudios de los resultados obtenidos en los entrenamientos de las diferentes redes neuronales desarrolladas

Código	Unidad	Descripción	Rendimiento	Precio unitario (€/h)	Importe (€/h)
1		**Materiales**			
	h	Ordenador de sobremesa	0,700	0,21	0,15
	h	Ordenador portátil	1,000	0,09	0,09
	h	Google Colab	0,300	0,00	0,00
	h	AnyDesk	0,700	0,00	0,00
	h	PyCharm	0,700	0,00	0,00
	h	Microsoft Office	0,600	0,05	0,03
2		**Mano de obra**			
	h	Titulado superior en ingeniería industrial	1,000	25,00	25,00
	h	Catedrático de universidad	0,300	51,40	15,42
			Subtotal materiales y mano de obra:		**40,69**
3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	40,69	0,81
				Costes directos (1+2+3)**:**	**41,50**

STN006 h Redacción de la memoria y el presupuesto

Redacción de los documentos explicativos del trabajo y del presupuesto, así como de la revisión y corrección de los mismos

Código	Unidad	Descripción	Rendimiento	Precio unitario (€/h)	Importe (€/h)
1		**Materiales**			
	h	Ordenador portátil	1,000	0,09	0,09
	h	Microsoft Office	1,000	0,05	0,05
2		**Mano de obra**			
	h	Titulado superior en ingeniería industrial	1,000	25,00	25,00
	h	Catedrático de universidad	0,200	51,40	10,28
			Subtotal materiales y mano de obra:		**35,42**
3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	35,42	0,71
				Costes directos (1+2+3)**:**	**36,13**

12.4. Mediciones

Código	Unidad	Descripción	Medición
STN001	h		
		Búsqueda de información	50,00
STN002	h		
		Adquisición de imágenes	30,00
STN003	h		
		Implementación de los códigos	40,00
STN004	h		
		Entrenamiento de las redes neuronales	70,00
STN005	h		
		Estudio de los resultados	70,00
STN006	h		
		Redacción de la memoria y el presupuesto	40,00

12.5. Presupuesto parcial

En este apartado se representa el costo total de las unidades de obra en función de las mediciones realizadas para cada una de ellas.

Código	Unidad	Descripción	Medición	Precio (€/h)	Importe (€)
STN001	h				
		Búsqueda de información	50,00	36,08	1.804,00
			Importe total:		**1.804,00**

Código	Unidad	Descripción	Medición	Precio (€/h)	Importe (€)
STN002	h				
		Adquisición de imágenes	30,00	47,69	1.430,70
			Importe total:		**1.430,70**

Código	Unidad	Descripción	Medición	Precio (€/h)	Importe (€)
STN003	h				
		Implementación de los códigos	40,00	36,19	1.447,60
			Importe total:		**1.447,60**

Código	Unidad	Descripción	Medición	Precio (€/h)	Importe (€)
STN004	h				
		Entrenamiento de las redes neuronales	70,00	32,43	2.270,10
			Importe total:		**2.270,10**

Código	Unidad	Descripción	Medición	Precio (€/h)	Importe (€)
STN005	h				
		Estudio de los resultados	70,00	41,50	2.905,00
			Importe total:		**2.905,00**

Código	Unidad	Descripción	Medición	Precio (€/h)	Importe (€)
STN006	h				
		Redacción de la memoria y el presupuesto	40,00	36,13	1.445,20
			Importe total:		**1.445,20**

12.6. Presupuesto de ejecución material

Código	Descripción	Importe (€)
STN001	Búsqueda de información	1.804,00
STN002	Adquisición de imágenes	1.430,70
STN003	Implementación de los códigos	1.447,60
STN004	Entrenamiento de las redes neuronales	2.270,10
STN005	Estudio de los resultados	2.905,00
STN006	Redacción de la memoria y el presupuesto	1.445,20
	Presupuesto de ejecución material:	**11.302,60**

La suma total del presupuesto de ejecución material asciende a: ONCE MIL TRESCIENTOS DOS EUROS CON SESENTA CÉNTIMOS.

12.7. Presupuesto de ejecución por contrata

Descripción	Importe (€)
Presupuesto de ejecución material	11.302,60
Gastos generales 13%	1.469,34
Beneficio industrial 6%	678,16
Presupuesto total	13.450,10
I.V.A. 21%	2.824,52
Presupuesto de ejecución por contrata:	**16.274,62**

Por tanto, la suma total del presupuesto del presente trabajo asciende a un total de: DIECISEIS MIL DOSCIENTOS SETENTA Y CUATRO EUROS CON SESENTA Y DOS CÉNTIMOS.

Parte III

Anexos

ANEXO I. CÓDIGO SIMULADOR

```
import numpy as np
import cv2
import os
from PIL import Image
from torchvision.transforms import transforms
from math import pi
import math
import random


def simulador(semilla):
    random.seed(semilla)
    incX = random.random()
    incY = random.random()
    X = np.array(
        [[(0.75 + 0.5 * incX) * -40, (0.75 + 0.5 * incX) * -40, (0.75 + 0.5 * incX) * 40, (0.75 + 0.5 * incX) * 40],
         [(0.75 + 0.5 * incY) * -150, (0.75 + 0.5 * incY) * 150, (0.75 + 0.5 * incY) * 150, (0.75 + 0.5 * incY) * -150],
         [0, 0, 0, 0], [1, 1, 1, 1]], dtype=np.int32)


    ang = 2*pi*random.random()
    D = math.sqrt(X[0][0] * X[0][0] + X[1][0] * X[1][0])
    X1 = np.array([[D * math.cos(ang + math.atan2(X[1][0], X[0][0])), D * math.cos(ang + math.atan2(X[1][1], X[0][1])),
                   D * math.cos(ang + math.atan2(X[1][2], X[0][2])), D * math.cos(ang + math.atan2(X[1][3], X[0][3]))],
                  [D * math.sin(ang + math.atan2(X[1][0], X[0][0])), D * math.sin(ang + math.atan2(X[1][1], X[0][1])),
                   D * math.sin(ang + math.atan2(X[1][2], X[0][2])), D * math.sin(ang + math.atan2(X[1][3], X[0][3]))],
                  [0, 0, 0, 0], [1, 1, 1, 1]], dtype=np.int32)
    trasX = int(-100+150*random.random())
    trasY = int(-100+150*random.random())
    X = np.array([[X1[0][0] + trasX, X1[0][1] + trasX, X1[0][2] + trasX, X1[0][3] + trasX],
                  [X1[1][0] + trasY, X1[1][1] + trasY, X1[1][2] + trasY, X1[1][3] + trasY], [0, 0, 0, 0],
                  [1, 1, 1, 1]], dtype=np.int32)
    """## **CONFIGURACION DE LA CAMARA**
    Parametros intrinsecos"""
```

```
fu = 1580   # fu = f/dx
fv = 1580   # fv = f/dy
uo = 150    # uo = xo
vo = 100    # vo = yo

"""Parametros extrinsecos"""
radio = 1000
angulo = 180-5
rX = (pi/180)*(angulo)   # angulo de rotación en el eje X
rY = 0                   # angulo de rotación en el eje Y
rZ = 0                   # angulo de rotación en el eje Z
tX = 0                   # Traslacion en el eje X
tY = math.sin((pi/180)*(angulo))*radio     # Traslacion en el eje Y
tZ = -math.cos((pi/180)*(angulo))*radio    # Traslacion en el eje Z

"""Construccion de la matriz de proyeccion"""
K = np.array([[fu, 0, uo],[0, fv, vo],[0, 0, 1]])
K1 = np.array([[fu, 0, uo, 0],[0, fv, vo, 0],[0, 0, 1, 0]])
Rx = np.array([[1, 0, 0],[0, math.cos(rX), -math.sin(rX)],[0, math.sin(rX), math.cos(rX)]])   # matriz de traslación según rotación en el eje X
Ry = np.array([[math.cos(rY), 0, math.sin(rY)],[0, 1, 0],[-math.sin(rY), 0, math.cos(rY)]])   # matriz de traslación según rotación en el eje Y
Rz = np.array([[math.cos(rZ), -math.sin(rZ), 0],[math.sin(rZ), math.cos(rZ), 0],[0, 0, 1]])   # matriz de traslación según rotación en el eje Z
R = np.dot(np.dot(Rx,Ry),Rz)        # Matriz de rotación que da la orientación de la camara en el espacio 3D
T = np.array([[tX], [tY], [tZ]])    # Matriz de traslacion que situa la camara en el espacio 3D

"""Matriz de proyeccion"""
P = np.dot(K1,np.linalg.inv(np.concatenate((np.concatenate((R, T),axis=1),np.array([[0, 0, 0, 1]])))))

"""Conjunto de puntos en la imagen 2D"""
U=np.dot(P,X)

U = np.concatenate([[U[0]/U[2]],[U[1]/U[2]],[np.ones(U[0].shape)]])

"""# Segunda cámara"""
X = np.array(
   [[(0.75 + 0.5 * incX) * -20, (0.75 + 0.5 * incX) * -20, (0.75 + 0.5 * incX) * 20, (0.75 + 0.5 * incX) * 20],
    [(0.75 + 0.5 * incY) * -60, (0.75 + 0.5 * incY) * 60, (0.75 + 0.5 * incY) * 60, (0.75 + 0.5 * incY) * -60],
    [0, 0, 0, 0], [1, 1, 1, 1]], dtype=np.int32)

D = math.sqrt(X[0][0] * X[0][0] + X[1][0] * X[1][0])
```

```
X1 = np.array([[D * math.cos(ang + math.atan2(X[1][0], X[0][0])), D * math.cos(ang + math.atan2(X[1][1], X[0][1])),
          D * math.cos(ang + math.atan2(X[1][2], X[0][2])), D * math.cos(ang + math.atan2(X[1][3], X[0][3]))],
          [D * math.sin(ang + math.atan2(X[1][0], X[0][0])), D * math.sin(ang + math.atan2(X[1][1], X[0][1])),
          D * math.sin(ang + math.atan2(X[1][2], X[0][2])), D * math.sin(ang + math.atan2(X[1][3], X[0][3]))],
          [0, 0, 0, 0], [1, 1, 1, 1]], dtype=np.int32)

X = np.array([[X1[0][0] + trasX, X1[0][1] + trasX, X1[0][2] + trasX, X1[0][3] + trasX],
          [X1[1][0] + trasY, X1[1][1] + trasY, X1[1][2] + trasY, X1[1][3] + trasY], [0, 0, 0, 0],
          [1, 1, 1, 1]], dtype=np.int32)

fu = 1580 # fu = f/dx
fv = 1580 # fv = f/dy
uo = 150 # uo = xo
vo = 100 # vo = yo

"""Parametros extrinsecos"""
radio = 1000
angulo = 180-10

rX = (pi/180)*(angulo) # angulo de rotación en el eje X
rY = 0                 # angulo de rotación en el eje Y
rZ = 0                 # angulo de rotación en el eje Z
tX = -150                      # Traslacion en el eje X
tY = math.sin((pi/180)*(angulo))*radio-150    # Traslacion en el eje Y
tZ = -math.cos((pi/180)*(angulo))*radio    # Traslacion en el eje Z

K = np.array([[fu, 0, uo],[0, fv, vo],[0, 0, 1]])
K1 = np.array([[fu, 0, uo, 0],[0, fv, vo, 0],[0, 0, 1, 0]])
Rx = np.array([[1, 0, 0],[0, math.cos(rX), -math.sin(rX)],[0, math.sin(rX), math.cos(rX)]]) #matriz de traslación según rotación en el eje X
Ry = np.array([[math.cos(rY), 0, math.sin(rY)],[0, 1, 0],[-math.sin(rY), 0, math.cos(rY)]]) #matriz de traslación según rotación en el eje Y
Rz = np.array([[math.cos(rZ), -math.sin(rZ), 0],[math.sin(rZ), math.cos(rZ), 0],[0, 0, 1]]) #matriz de traslación según rotación en el eje Z
R = np.dot(np.dot(Rx,Ry),Rz)    # Matriz de rotación que da la orientación de la camara en el espacio 3D
T = np.array([[tX], [tY], [tZ]]) # Matriz de traslacion que situa la camara en el espacio 3D
P2 = np.dot(K1,np.linalg.inv(np.concatenate((np.concatenate((R, T),axis=1),np.array([[0, 0, 0, 1]])))))
U2 = np.dot(P2,X)
U2 = np.concatenate([[U2[0]/U2[2]],[U2[1]/U2[2]],[np.ones(U2[0].shape)]])

esquinas = []
for i in range(4):
```

```
  esquinas.append([int(U[0][i])+1296, int(U[1][i])+976])
esquinas = np.asarray(esquinas)
esquinas= esquinas.reshape(1,4,2)
Fondo = 255*np.ones((1944, 2592), np.uint8)
micro1 = cv2.fillPoly(Fondo,esquinas,0)

esquinas = []
for i in range(4):
  esquinas.append([int(U2[0][i])+1296, int(U2[1][i])+976])
esquinas = np.asarray(esquinas)
esquinas= esquinas.reshape(1,4,2)
Fondo = 255*np.ones((1944, 2592), np.uint8)
micro2 = cv2.fillPoly(Fondo,esquinas,0)

return micro2, micro1
```

ANEXO II. CÓDIGO STN DATASET SIMULADO

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import cv2
import os
import torch.optim as optim
import matplotlib.pyplot as plt
from PIL import Image
from tqdm import tqdm
from torchvision.transforms import transforms
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from Simulador import simulador
from math import pi
import math
import random
from torch.autograd import Variable

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
if device.type == "cuda":
    torch.cuda.get_device_name()
print(device)

random.seed(2020)
a = np.arange(1,1001)
random.shuffle(a)
x_train = a
batch_size = 8
epochs = 150
path = '/home/ingivision/PycharmProjects/STN/Simulador'
```

```
class transf(Dataset):
    def __init__(self, images, transform=None):
        self.transform = transform
        self.images = images
        self.recorte2 = transforms.Compose([
                                    transforms.ToPILImage(),
                                    transforms.CenterCrop((972, 1296)),
            ])

    def __len__(self):
        return len(self.images)

    def __getitem__(self, item):

        dist, target_dist = simulador(self.images[item])
        seg = 255 - np.asarray(dist)
        target_seg = 255 - np.asarray(target_dist )
        t, thresh = cv2.threshold(dist, 127, 255, cv2.THRESH_BINARY)
        canny = cv2.Canny(thresh,0,255)
        canny = 255 - canny
        dist = cv2.distanceTransform(canny,cv2.DIST_C,3)
        [x, y] = np.where(dist > 255)
        dist[x,y] = 255
        dist = 255 - dist
        dist = dist.astype(np.uint8)
        t, thresh = cv2.threshold(target_dist , 127, 255, cv2.THRESH_BINARY)
        canny = cv2.Canny(thresh,0,255)
        canny = 255 - canny
        target_dist  = cv2.distanceTransform(canny,cv2.DIST_C,3)
        [x, y] = np.where(target_dist  > 255)
        target_dist [x,y] = 255
        target_dist  = 255 - target_dist
        target_dist  = target_dist .astype(np.uint8)

        if self.transform:
            dist = self.transform(dist)
            seg = self.transform(seg)
            target_seg = self.recorte2(target_seg)
            target_seg = self.transform(target_seg)
```

```
        target_dist  = self.recorte2(target_dist )
        target_dist  = self.transform(target_dist )

    return dist, target_dist, seg, target_seg

transform=transforms.Compose([
                    transforms.ToTensor(),
            ])

train_ds = transf(x_train,
            transform=transform)

train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True)

class AffineNet(nn.Module):
  def __init__(self, input_nc=1, transform=torch.tensor([1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0], dtype=torch.float)):
    super(AffineNet, self).__init__()
    # Spatial transformer localization-network

    self.affine_stl = nn.Sequential(
      nn.Conv2d(input_nc, 8, kernel_size=7),
      nn.MaxPool2d(2, stride=2),
      nn.SELU(True),
      nn.Conv2d(8, 10, kernel_size=5),
      nn.MaxPool2d(2, stride=2),
      nn.SELU(True),
      nn.Conv2d(10, 20, kernel_size=5),
      nn.MaxPool2d(2, stride=2),
      nn.SELU(True),
      nn.Conv2d(20, 40, kernel_size=3),
      nn.MaxPool2d(2, stride=2),
      nn.SELU(True),
      nn.Conv2d(40, 10, kernel_size=3),
      nn.SELU(True),
    )

    self.stl_fc = self.fc_loc = nn.Sequential(
      nn.Linear(182120, 32),
      nn.SELU(True),
```

```
            nn.Linear(32, 4 * 3)
        )

        self.stl_fc[2].weight.data.zero_()
        self.stl_fc[2].bias.data.copy_(transform)

    def forward(self, x):
        xs = self.affine_stl(x)
        xs = xs.view(-1, 182120)
        theta = self.stl_fc(xs)
        self.theta = theta.view(-1, 3, 4)
        grid = F.affine_grid(self.theta, (x.size(0),1,1,972, 1296))

        return grid, theta.cpu()

model = AffineNet().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.0001)
loss_fun = nn.MSELoss(reduction='mean')
criterion = loss_fun.to(device)

def train_dist(model, dataloader, epoch):
    model.train()
    running_loss = []
    for step, (dist, target_dist, seg, target_seg) in enumerate(tqdm(dataloader, desc="Training")):
        dist = dist.to(device)
        seg = seg.to(device)
        target_dist = target_dist.to(device)
        target_seg = target_seg.to(device)

        optimizer.zero_grad()
        grid, theta = model(dist)
        out = F.grid_sample(dist.view(-1, 1, 1, 1944, 2592), grid, padding_mode="border")
        out = out.view(-1, 1, 972, 1296)

        loss = criterion(out, target_dist)
        loss.backward()
        optimizer.step()
        out2 = F.grid_sample(seg.view(-1, 1, 1, 1944, 2592), grid, padding_mode="border")
        out2 = out2.view(-1, 1, 972, 1296)
```

```
    loss = criterion(out2, target_seg)
    if step==0:
      trans_dist = torch.mean(theta,axis=0)
    else:
      trans_dist = trans_dist*step/(step+1) + torch.mean(theta,axis=0)/(step+1)
    running_loss.append(loss.cpu().detach().numpy())
    if step == 0:
      img3 = np.dstack((255 - out2[0][0].cpu().detach().numpy()*255, 255 - target_seg[0][0].cpu().detach().numpy()*255, 255 - out2[0][0].cpu().detach().numpy()*255))
      cv2.imwrite(f"/home/ingivision/PycharmProjects/STN/Ensayos para documentación/ imágenes/epoch{epoch}.jpg", img3)
  return (np.mean(running_loss),trans_dist)


def train_seg(model, dataloader, epoch):
  model.train()
  running_loss = []
  iou = 0
  for step, (dist, target_dist, seg, target_seg) in enumerate(tqdm(dataloader, desc="Training")):
    seg = seg.to(device)
    target_seg = target_seg.to(device)


    optimizer2.zero_grad()


    grid2, theta = model(seg)
    out2 = F.grid_sample(seg.view(-1, 1, 1, 1944, 2592), grid2, padding_mode="border")
    out2 = out2.view(-1, 1, 972, 1296)


    loss = criterion(out2, target_seg)
    loss.backward()
    optimizer2.step()
    running_loss.append(loss.cpu().detach().numpy())
    if step == 0:
      img3 = np.dstack((255 - out2[0][0].cpu().detach().numpy()*255, 255 - target_seg[0][0].cpu().detach().numpy()*255, 255 - out2[0][0].cpu().detach().numpy()*255))
      cv2.imwrite(f"/home/ingivision/PycharmProjects/STN/Ensayos para documentacion/imagenes/epoch{step}.jpg",img3)


    for i in range(len(out2)):
      t, thresh = cv2.threshold(out2[i][0].cpu().detach().numpy()*255, 127, 255, cv2.THRESH_BINARY)
      img1 = thresh.astype(np.uint8)
```

```
        ref = target_seg[i][0].cpu().detach().numpy()*255
        intersection = np.logical_and(ref, img1)
        union = np.logical_or(ref, img1)
        iou += np.sum(intersection) / np.sum(union)

  return(np.mean(running_loss), iou/len(x_train))

l = 10000000
m = 10000000
n = 10000000
o = 10000000

train_loss = []
val_loss = []
seg_not_improve = 0
dist_not_improve = 20
epoch_change = 0

for epoch in tqdm(range(epochs), desc="Epochs"):
  print(f"Epoch {epoch + 1} of {epochs}")

  if epoch < 10:
    if epoch%2 == 0:
      train_epoch_loss, trans_dist = train_dist(model, train_loader, epoch)
    else:
      if epoch == 1:
        optimizer2 = optim.Adam(model.parameters(), lr=0.00002)
      train_epoch_loss, iou = train_seg(model, train_loader, epoch)
      for g in optimizer2.param_groups:
        if g['lr'] < 0.0001:
          g['lr'] += 0.00002
    train_loss.append(train_epoch_loss)
    if train_epoch_loss < n:
      n = train_epoch_loss
  else:
    if epoch_change == 0:
      print("Cambio")
      epoch_change = epoch + 1
    train_epoch_loss, iou = train_seg(model, train_loader, epoch)
```

```
        train_loss.append(train_epoch_loss)
        if train_epoch_loss < m:
            m = train_epoch_loss
            seg_not_improve = 0
            torch.save(model.state_dict(), '/home/ingivision/PycharmProjects/STN/Ensayos para documentacion/Modelos/Simulador/best_modelSTN_train_dist.pth')
        else:
            seg_not_improve += 1

        dist_not_improve += 1
    print("Train_Loss: {:.6f}".format(train_epoch_loss))

plt.figure(figsize=(10, 7))
plt.plot(train_loss, color='orange', label='train loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

total = 0
mine = 10000000000000
maxe = 0
iou = 0
model.load_state_dict(torch.load('/home/ingivision/PycharmProjects/STN/Ensayos para documentacion/Modelos/Simulador/best_modelSTN_train_dist.pth'))
test_results = []
loss = []
model.eval()
j=0
k=0
with torch.no_grad():
    for (dist, target_dist, seg, target_seg) in tqdm(train_loader):
        grid, iou= model(seg.to(device))
        grid = grid.cpu()
        results = F.grid_sample(seg.view(-1, 1, 1, 1944, 2592), grid, padding_mode="border")
        results = results.view(-1,1,972,1296)
        loss_i = loss_fun(results, target_seg)
        loss.append(loss_i.cpu().detach().numpy())
        for i in range(len(dist)):
```

```
        a = 0
        ref = target_seg[i][0].detach().numpy()*255
        img1 = results[i][0].detach().numpy()*255
        ref = ref.astype(np.uint8)
        img1 = img1.astype(np.uint8)
        t, thresh1 = cv2.threshold(img1, 127, 255, cv2.THRESH_BINARY)

        intersection = np.logical_and(ref, thresh1)
        union = np.logical_or(ref, thresh1)
        iou_a = np.sum(intersection) / np.sum(union)

        img = np.asarray(ref, dtype=np.int32)
        thresh1 = np.asarray(thresh1, dtype=np.int32)

        diff1 = np.abs(ref - thresh1)

        a = int(sum(sum(diff1)) / 255)

        if a<mine:
           mine = a
        if a>maxe:
           maxe = a
        iou += iou_a
        total += a
      j+=8

media = total/len(x_train)
iou = iou/len(x_train)

print("Train loss min dist: {:.6f}".format(o))
print("Train loss min: {:.6f}".format(m))
print("Train loss: {:.6f}".format(np.mean(loss)))
print(total, media, mine, maxe, iou)
print("Epoca de cambio de modelo: {:.1f}".format(epoch_change))
```

ANEXO III CÓDIGO STN DATASET DE *C. ELEGANS*

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import cv2
import os
import torch.optim as optim
import matplotlib.pyplot as plt
from PIL import Image
from tqdm import tqdm
from torchvision.transforms import transforms
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
import random

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

if device.type == "cuda":
    torch.cuda.get_device_name()
print(device)

path = '/home/ingivision/PycharmProjects/STN/La_Fe2/Micro2_trans_2'
images = os.listdir(path)
ori_images = []
target_images = []

for i in range(len(images)):
  ori_images.append(f"DistMicro2C/{images[i]}")
  target_images.append(f"DistMicro1C_Pequeño/{images[i]}")

x_train = ori_images
y_train = target_images
```

```
random.seed(2020)
batch_size = 8
epochs = 500
path = '/home/ingivision/PycharmProjects/STN/La_Fe2'
Segmentada = False


class transf(Dataset):
    def __init__(self, images, labels, transform=None):

        self.transform = transform
        self.images = images
        self.labels = labels

    def __len__(self):
        return len(self.images)

    def __getitem__(self, item):
        dist = Image.open(f"{path}/{self.images[item]}")
        target_dist = Image.open(f"{path}/{self.labels[item]}")
        dist = np.asarray(dist)
        target_dist = np.asarray(target_dist)
        x,y = self.images[item].split("/")
        seg = Image.open(f"{path}/SegMicro2C/{y}")
        target_seg = Image.open(f"{path}/SegMicro1C_Pequeño/{y}")
        seg = np.asarray(seg)
        target_seg = np.asarray(target_seg)
        if self.transform:
            dist = self.transform(dist)
            seg = self.transform(seg)
            target_seg = self.transform(target_seg)
            target_dist = self.transform(target_dist)

        return dist, target_dist, seg, target_seg

transform=transforms.Compose([
                        transforms.ToTensor(),
            ])

train_ds = transf(x_train,
```

```
            y_train,
            transform=transform)

train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True)

class AffineNet(nn.Module):
    def __init__(self, input_nc=1, transform=torch.tensor([1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0], dtype=torch.float)):
        super(AffineNet, self).__init__()
        # Spatial transformer localization-network

        self.affine_stl = nn.Sequential(
            nn.Conv2d(input_nc, 8, kernel_size=7),
            nn.MaxPool2d(2, stride=2),
            nn.SELU(True),
            nn.Conv2d(8, 10, kernel_size=5),
            nn.MaxPool2d(2, stride=2),
            nn.SELU(True),
            nn.Conv2d(10, 20, kernel_size=5),
            nn.MaxPool2d(2, stride=2),
            nn.SELU(True),
            nn.Conv2d(20, 40, kernel_size=3),
            nn.MaxPool2d(2, stride=2),
            nn.SELU(True),
            nn.Conv2d(40, 10, kernel_size=3),
            nn.SELU(True),
        )

        self.stl_fc = self.fc_loc = nn.Sequential(
            nn.Linear(182120, 32),
            nn.SELU(True),
            nn.Linear(32, 4 * 3)
        )

        self.stl_fc[2].weight.data.zero_()
        self.stl_fc[2].bias.data.copy_(transform)

    def forward(self, x):
        xs = self.affine_stl(x)
        xs = xs.view(-1, 182120)
```

```
        theta = self.stl_fc(xs)
        self.theta = theta.view(-1, 3, 4)
        grid = F.affine_grid(self.theta, (x.size(0),1,1,972, 1296))

        return grid, theta.cpu()

model = AffineNet().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.0001)
loss_fun = nn.MSELoss(reduction='mean')
criterion = loss_fun.to(device)

def train_dist(model, dataloader, epoch):
    model.train()
    running_loss = []
    iou = 0
    for step, (dist, target_dist, seg, target_seg) in enumerate(tqdm(dataloader, desc="Training")):
        dist = dist.to(device)
        seg = seg.to(device)
        target_dist = target_dist.to(device)
        target_seg = target_seg.to(device)

        optimizer.zero_grad()
        grid, theta = model(dist)
        out = F.grid_sample(dist.view(-1, 1, 1, 1944, 2592), grid, padding_mode="border")
        out = out.view(-1, 1, 972, 1296)

        loss = criterion(out, target_dist)
        loss.backward()
        optimizer.step()

        out2 = F.grid_sample(seg.view(-1, 1, 1, 1944, 2592), grid, padding_mode="border")
        out2 = out2.view(-1, 1, 972, 1296)

        loss = criterion(out2, target_seg)
        running_loss.append(loss.cpu().detach().numpy())

        for i in range(len(out2)):
            t, thresh = cv2.threshold(out2[i][0].cpu().detach().numpy()*255, 127, 255, cv2.THRESH_BINARY)
            img1 = 255-thresh.astype(np.uint8)
```

```
            ref = 255-target_seg[i][0].cpu().detach().numpy()*255
            intersection = np.logical_and(ref, img1)
            union = np.logical_or(ref, img1)
            iou += np.sum(intersection) / np.sum(union)
    return (np.mean(running_loss),iou)


def train_seg(model, dataloader, epoch):
    model.train()
    running_loss = []
    iou = 0
    for step, (dist, target_dist, seg, target_seg) in enumerate(tqdm(dataloader, desc="Training")):
        seg = seg.to(device)
        target_seg = target_seg.to(device)


        optimizer2.zero_grad()


        grid2, theta = model(seg)
        out2 = F.grid_sample(seg.view(-1, 1, 1, 1944, 2592), grid2, padding_mode="border")
        out2 = out2.view(-1, 1, 972, 1296)


        loss = criterion(out2, target_seg)
        if loss > 0.05:
            continue
        loss.backward()
        optimizer2.step()
        running_loss.append(loss.cpu().detach().numpy())


        for i in range(len(out2)):
            t, thresh = cv2.threshold(out2[i][0].cpu().detach().numpy()*255, 127, 255, cv2.THRESH_BINARY)
            img1 = 255-thresh.astype(np.uint8)
            ref = 255-target_seg[i][0].cpu().detach().numpy()*255
            intersection = np.logical_and(ref, img1)
            union = np.logical_or(ref, img1)
            iou += np.sum(intersection) / np.sum(union)


    return(np.mean(running_loss), iou/len(x_train))


l = 10000000
m = 10000000
```

```
n = 10000000
o = 10000000

train_loss = []
val_loss = []
seg_not_improve = 0
dist_not_improve = 20
epoch_change = 0

for epoch in tqdm(range(epochs), desc="Epochs"):
  print(f"Epoch {epoch + 1} of {epochs}")

  if n > 0.017:
    train_epoch_loss, trans_dist = train_dist(model, train_loader, epoch)
    if epoch%2 == 0:
      train_epoch_loss, trans_dist = train_dist(model, train_loader, epoch)
    else:
      if epoch == 1:
        optimizer2 = optim.Adam(model.parameters(), lr=0.0001)
      train_epoch_loss, iou = train_seg(model, train_loader, epoch)
    train_loss.append(train_epoch_loss)
    if train_epoch_loss < n:
      n = train_epoch_loss
  else:
    if epoch_change == 0:
      epoch_change = epoch + 1

    train_epoch_loss, iou = train_seg(model, train_loader, epoch)
    train_loss.append(train_epoch_loss)
    if train_epoch_loss < m:
      m = train_epoch_loss
      seg_not_improve = 0
      torch.save(model.state_dict(), '/home/ingivision/PycharmProjects/STN/Ensayos para documentacion/Modelos/Gusanos/best_modelSTN_train_distseg.pth')
    else:
      seg_not_improve += 1
      if seg_not_improve == 20:
        epoch_fin = epoch - 20
    if (seg_not_improve >= 20) or iou > 0.86:
```

```
            break


        dist_not_improve += 1
    print("Train_Loss: {:.6f}".format(train_epoch_loss))


total = 0
mine = 10000000000000
maxe = 0
iou = 0
miniou = 1
maxiou = 0
model.load_state_dict(torch.load('/home/ingivision/PycharmProjects/STN/Ensayos para documentacion/Modelos/Gusanos/best_modelSTN_train_distseg.pth'))
test_results = []
loss = []
model.eval()
num = len(x_train)
with torch.no_grad():
    for (dist, target_dist, seg, target_seg) in tqdm(train_loader):
        grid, theta= model(seg.to(device))
        grid = grid.cpu()
        results = F.grid_sample(seg.view(-1,1,1,1944,2592), grid, padding_mode="border")
        loss_i = loss_fun(results.view(-1, 1, 972, 1296), target_seg)
        loss.append(loss_i.cpu().detach().numpy())


        for i in range(len(dist)):
            a = 0
            ref = 255-target_seg[i][0].detach().numpy()*255
            img1 = results[i][0][0].detach().numpy()*255
            ref = ref.astype(np.uint8)
            img1 = img1.astype(np.uint8)
            t, thresh1 = cv2.threshold(img1, 127, 255, cv2.THRESH_BINARY)
            thresh1 = 255-thresh1
            intersection = np.logical_and(ref, thresh1)
            union = np.logical_or(ref, thresh1)
            iou_a = np.sum(intersection) / np.sum(union)
            img = np.asarray(ref, dtype=np.int32)
            thresh1 = np.asarray(thresh1, dtype=np.int32)
```

```
        diff1 = np.abs(ref - thresh1)


        a = int(sum(sum(diff1)) / 255)


        if a<mine:
            mine = a
        if a>maxe:
            maxe = a


        iou += iou_a
        if iou_a<miniou:
            miniou = iou_a
        if iou_a>maxiou:
            maxiou = iou_a
        total += a
media = total/num
iou = iou/num
print(num)
loss = np.mean(loss)
print(loss)
print("Train loss: {:.6f}".format(np.mean(loss)))
print(total, media, mine, maxe, iou, miniou, maxiou)


print("Epoca de cambio de modelo: {:.1f}".format(epoch_change))


plt.figure(figsize=(10, 7))
plt.plot(train_loss, color='orange', label='train loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('/home/ingivision/PycharmProjects/STN/Ensayos para documentacion/gráficas de loss/entrenamiento intercalado/new_loss.png')
plt.show()
```