

Tecnológico de Costa Rica
Área Académica en Ingeniería en Computadores



Estrategia de ajuste de formas explícitas a imágenes con estructuras vermiformes.

Informe de Trabajo de Graduación para optar por el título de Ingeniero en
Computadores con grado académico de Licenciatura

Miguel Ángel Taylor López

Cartago, 20 de noviembre, 2017

Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.

Miguel Ángel Taylor López

Cartago, 21 de noviembre de 2017

Céd: 1-1526-0289

TEC – Área Ingeniería en Computadores (CE) Acta de Aprobación de Trabajo de Graduación

Con fundamento en lo que establece el "Reglamento de Trabajos Finales de Graduación del Instituto Tecnológico de Costa Rica", el Tribunal Examinador del Trabajo Final de Graduación, nombrado con el propósito de evaluar el proyecto final de graduación.

"Estrategia de ajuste de formas explícitas a imágenes con estructuras vermiforme"

Habiendo analizado el resultado general del trabajo presentado por los estudiantes:

Primer Apellido	Segundo Apellido	Nombre	No. De carné
Taylor	López	Miguel Ángel	201117893

Emite el siguiente dictamen:

APROBADO

CALIFICACION: 100 puntos.

REPROBADO



SE
RECOMIENDA



NO SE
RECOMIENDA

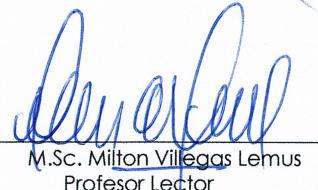
Brindarle una nueva oportunidad para la
DEFENSA PUBLICA de su Trabajo Final

NUEVA FECHA: _____

Dando fe de lo aquí expuesto firmamos


M.Sc. Aníbal Coto Cortés

Profesor Asesor


M.Sc. Milton Villegas Lemus

Profesor Lector


Dr. Roberto Pereira Arroyo

Profesor Lector

20 de Noviembre del 2017

Resumen

En este proyecto se desarrollaron herramientas computacionales para acelerar el análisis de imágenes de nematodos. Los modelos de forma explícitos son usados para localizar los nematodos en imágenes de microscopía. Una desventaja de estos modelos es que es difícil hacerlos coincidir con la imagen. Este proyecto se centró en el desarrollo de modelos de redes neuronales convolucionales que permitan ajustar una forma a la imagen.

Primero se creó una herramienta para la detección y corrección automática de datos de entrenamiento. Esto con el fin de superar la escasez de datos de entrenamiento debido a errores en el proceso de etiquetación manual. La detección de errores está basada en la búsqueda de cruces en el spline usado para interpolar la forma y conservar el sentido horario de los puntos. Por otro lado la corrección de errores utiliza algoritmos basados en la distancia Euclidiana, triangulación de Delaunay y el problema del vendedor ambulante.

Posteriormente se diseñaron y entrenaron las redes neuronales convolucionales para reconocer el borde de un nematodo. El entrenamiento de la red utiliza arquitecturas GPU para poder lidiar con la alta demanda de recurso computacional que conlleva la determinación de tanto las máscaras de filtros lineales como la convolución con ellos. En esta etapa se deben extraer conjuntos de datos para entrenamiento y pruebas a partir de la base de datos corregida. Éstos conjuntos de datos se utilizan para entrenar una red neuronal convolucional. Este proceso se repite hasta que se obtiene un modelo que satisfaga los criterios de aceptación. Se demuestra que se puede usar aprendizaje profundo para ajustar puntos al borde de un nematodo.

Palabras clave: Aprendizaje profundo, Triangulación de Delaunay, Nematodos

Abstract

In this project computational tools were developed to speed up the analysis of images of nematodes. The training of the network uses GPU architectures to be able to deal with it with the high computational cost involved in determining both the linear filter masks and the convolution with them. Explicit shape models are used to locate the nematodes in microscopy images, based on the training data. One disadvantage of these models is that it is difficult to match them with the image. This project focuses on the development of a series of convolutional neural network models that allow to fit a shape to the image.

First, a tool was created for automatic detection and correction of training data. This in order to overcome the lack of training data due to errors in the manual labeling process. Error detection is based on the search for crosses in the spline used for interpolation and conserving the clockwise direction of points. On the other hand, error correction uses algorithms based on Euclidean distance, Delaunay triangulation and traveling salesman problem.

Later, convolutional neural networks were designed and trained to recognize the edge of a nematode. The training of the network uses GPU architectures to deal with the high computational cost that involves the determination of both the linear filter masks and the convolution with them. At this stage, data sets must be extracted for training and testing from the corrected database. These data sets are used to train a convolutional neural network. This process is repeated until a model that satisfies the acceptance criteria is obtained. It is shown that deep learning can be used to adjust points to the edge of a nematode.

Keywords: Deep learning, Delaunay triangulation, Nematodes

a mis queridos padres

Agradecimientos

Al Dr. Pablo Alvarado por la ayuda y el conocimiento brindado durante el último año, sin el cuál realizar este proyecto hubiese sido imposible.

A mi novia Gretchell Ochoa, quien me brindó su apoyo y compañía en todo momento.

Miguel Ángel Taylor López

Cartago, 21 de noviembre de 2017

Índice general

Índice de figuras	17
Índice de tablas	19
Lista de símbolos y abreviaciones	21
1. Introducción	23
1.1. Objetivos y estructura del documento	25
1.1.1. Objetivo general	25
1.1.2. Objetivos específicos	25
2. Marco teórico	27
2.1. Trabajos anteriores	27
2.2. Spline	32
2.3. Derivada del Gaussiano	33
2.4. Envolvente convexa	34
2.5. Triangulación de Delaunay	35
2.6. Problema del vendedor ambulante	36
2.7. Aprendizaje profundo	37
2.7.1. Capas convolucionales	38
2.7.2. Capas de reducción	38
2.7.3. Capa de aplanamiento	38
2.7.4. Capas densas	39
2.7.5. Capas de expulsión	39
2.7.6. Funciones de activación	39
3. Sistema de ajuste de formas a la imagen	41
3.1. Representación de los datos	41
3.2. Herramienta de corrección de los datos	43
3.2.1. DoG dependiente de escala	43
3.2.2. Búsqueda de errores	44
3.2.3. Arreglar hitos invertidos	44
3.2.4. Arreglar posición de cabeza y cola	45
3.2.5. Reordenar todos los hitos	45
3.2.6. Ajuste de los puntos al borde	48

3.3.	Redes neuronales convolucionales	48
3.3.1.	Extracción de datos	48
3.3.2.	Entrenamiento y arquitecturas de red	50
4.	Resultados y análisis	55
4.1.	Corrección de datos	55
4.2.	CNN	62
4.2.1.	Función de activación	62
4.2.2.	Pruebas gráficas	62
4.2.3.	Exactitud y error	66
4.2.4.	Aceleración con GPU	68
5.	Conclusiones	71
	Bibliografía	75
A.	Keras	81
A.1.	Ambiente de desarrollo	81
A.2.	Uso de la biblioteca	81

Índice de figuras

1.1.	Identificación de esqueleto y cabeza en un nematodo	24
1.2.	Diagrama de alto nivel de la solución desarrollada.	25
2.1.	ASM aplicado a nematodos.	28
2.2.	Segmentación con <i>level sets</i>	29
2.3.	Espacios convexos de forma vs. <i>level sets</i>	29
2.4.	Segmentación múltiple con <i>multiphase graph cuts</i>	29
2.5.	Regresión 3D con DenseReg	30
2.6.	Filtros DoG	34
2.7.	Ejemplo de la envolvente convexa	35
2.8.	Algoritmo básico de la envolvente convexa	35
2.9.	Funciones de activación para CNN	40
3.1.	Pérdida de información por aproximación con spline	41
3.2.	Cálculo del ancho y largo del nematodo	42
3.3.	Diagrama de flujo de la corrección de errores	44
3.4.	Ejemplo de nematodo que cruza sobre sí mismo	46
3.5.	Vectores extraídos para entrenar las CNN	49
3.6.	Diagrama de la primera arquitectura de CNN	52
4.1.	Resultado del algoritmo de corrección de hitos invertidos	57
4.2.	Resultado del algoritmo de corrección de la posición de cola y cabeza	58
4.3.	Resultado de los algoritmos rápidos	59
4.4.	Resultado del algoritmo de reordenamiento total de hitos	60
4.5.	Resultado de ajustar los hitos al borde	61
4.6.	Aumento y regularización de los hitos por medio del spline	61
4.7.	Imagen original utilizada para probar las CNN	62
4.8.	Evaluación de datos sin <i>offset</i> en la primera arquitectura de CNN	63
4.9.	Evaluación de datos con <i>offset</i> en la primera arquitectura de CNN	63
4.10.	Resultado de evaluar una imagen en la primera arquitectura de CNN	64
4.11.	Clasificación de vectores de una imagen con la segunda arquitectura de CNN	65
4.12.	Resultado de evaluar una imagen en la tercera arquitectura de CNN	66
4.13.	Resultado combinado de la segunda y tercera arquitectura de CNN	67
4.14.	Evaluación de datos sin <i>offset</i> en la tercera arquitectura de CNN	67
4.15.	Evaluación de datos con <i>offset</i> en la tercera arquitectura de CNN	67

4.16. Error de la primera y tercera arquitectura	68
4.17. Tiempos de ejecución en segundos de una época de entrenamiento en cada arquitectura con y sin GPU	69
A.1. Archivo de configuración de Theano	82

Índice de tablas

2.1.	Aplicación de técnicas de procesamiento de imágenes en nematología	31
2.2.	Resumen de técnicas de procesamiento de imágenes	32
3.1.	Sets de datos extraídos	50
3.2.	Bloques de construcción para CNN	51
3.3.	Resumen de la primera arquitectura de CNN	52
3.4.	Resumen de la segunda arquitectura de CNN	53
3.5.	Resumen de la tercera arquitectura de CNN	53
3.6.	Resumen de arquitecturas de CNN	54
4.1.	Resultado de la detección automática de errores	56
4.2.	Tiempos de ejecución de los algoritmos de corrección de datos	56
4.3.	Resultado de la corrección automática de errores	58

Listado de símbolos y abreviaciones

Abreviaciones

ACM	Modelos activos de contorno
ASM	Modelos activos de forma
CNN	Redes neuronales convolucionales
CSS	Espacios convexos de forma
DoG	Derivada del Gaussiano
GPU	Unidad de procesamiento gráfico
NEMO	Nematode movement
PCA	Análisis de componentes principales
SRS	Segmentación y reconocimiento simultáneos
SSM	Modelos estadísticos de forma
SVM	Máquinas de soporte vectorial
TFM	Modelos con factor de textura
TSP	Traveling salesman problem
YOLO	You only look once

Notación general

A	Matriz.
$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$	
$a_{:m}$	Columna m
$a_{n:}$	Fila n
x	Vector.
$\underline{\mathbf{x}} = [x_1 \ x_2 \ \dots \ x_n]^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$	
y	Escalar.

Capítulo 1

Introducción

La economía centroamericana ha dependido casi exclusivamente de la agricultura desde principios del siglo XX. Aunque inicialmente estaba enfocada solo en el banano, la economía de Costa Rica se ha diversificado a través de los años, agregando productos como café, caña de azúcar, piña y frutos tropicales. Estadísticas del 2015 ubican al banano y frutas tropicales entre las exportaciones más importantes. El éxito de este sector depende de la aplicación de nuevas tecnologías para el manejo de plagas y contrarrestar el cambio climático, para de esta manera cumplir con las demandas del mercado mundial, tanto en volumen como en calidad del producto agrícola[30].

Los nematodos son organismos vermiciformes, comúnmente encontrados en varios tipos de suelos y en ambientes acuáticos, con una gran variedad de comportamientos, tamaños y formas. En la figura 1.1 se encuentra un ejemplo de nematodo. Se ha identificado que algunas especies son fitoparásitas, es decir que parasitan especies de plantas aferrándose a sus raíces, y se han descubierto nematodos que atacan plantaciones importantes en Costa Rica, como el banano[1]. Lidiar con los nematodos es complicado, debido a que estos organismos resisten una gran cantidad de plaguicidas, por eso se han propuesto métodos alternativos, entre ellos utilizar otros nematodos beneficiosos para combatir a los dañinos[26].

Es necesario desarrollar herramientas tecnológicas que ayuden a los agrónomos a realizar un mejor análisis de cultivos y de plagas. Por ejemplo, para analizar un cultivo en busca de nematodos, un investigador debe procesar manualmente miles de imágenes, donde cada imagen puede tardar más de 15 minutos. Este tipo de estudios es necesario porque permite al investigador inferir información sobre el comportamiento de la plaga, o sobre el efecto de algún método de control de plagas implementado. La gran cantidad de horas que requiere esta tarea, y el costo económico que implica, hacen que muchas veces sea imposible realizarla o que haya que recortar costos en otras áreas de la investigación. Se hace evidente la necesidad de contar con una herramienta, ya sea supervisada o automática, que permita realizar esta tarea de manera más eficiente y rápida.

Para solucionar este problema, se han aplicado diferentes enfoques de visión por compu-

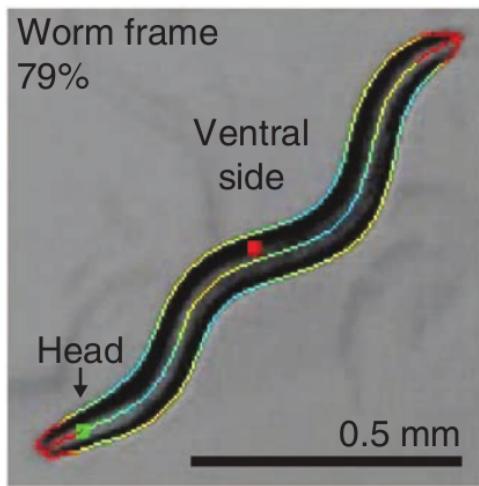


Figura 1.1: Identificación de esqueleto y cabeza en un nematodo de la especie *Caenorhabditis elegans* [57].

tadora, para automatizar el análisis de movimiento y comportamiento de los nematodos [24, 40, 58]. Este esfuerzo es relativamente reciente y se puede delimitar aproximadamente a los últimos 10 años. Es importante destacar que, a pesar de su importancia, en esta área se siguen aplicando métodos arcaicos. Lo más reciente fue aplicar las máquinas de soporte vectorial al problema [58], pero no existen aún publicaciones que apliquen técnicas más recientes como el aprendizaje profundo. Además, el área de aplicación estuvo ausente en la última conferencia mundial de visión por computadora y reconocimiento de patrones (CVPR 2017). Esto refleja la oportunidad de innovación que existe en esta área de investigación.

Las redes neuronales convolucionales (CNN) son una técnica de aprendizaje automático que permite solucionar problemas de clasificación. Su principal limitación es la gran cantidad de conocimiento a priori, en forma de datos de entrenamiento, que requieren para obtener una solución satisfactoria. Para la aplicación de segmentación de nematodos no existen bases de datos etiquetadas, puesto que es un proceso altamente costoso en cuanto a tiempo de expertos, y la variabilidad en la imágenes es fuertemente dependiente de los microscopios utilizados, lo que impone una fuerte limitación en el proyecto. Por ello, una contribución del presente trabajo es una propuesta para aumentar la cantidad de datos disponibles replanteando la tarea de clasificación en un problema de menor dimensión.

La solución desarrollada en este proyecto se puede dividir en dos etapas: adquisición de datos y diseño de las CNN. En la primera etapa se desarrolla una herramienta para detectar y corregir errores en la base de datos disponible. Esta base de datos se obtuvo mediante la etiquetación manual realizada por estudiantes de la UNA sobre datos adquiridos en el laboratorio de nematología del TEC, sede Santa Clara. Presenta errores en el orden de los puntos y su ubicación respecto al borde del organismo. Además se desarrollan varios programas para obtener conjuntos de datos de entrenamiento con diferentes características. En la segunda etapa se diseñan varias arquitecturas de red con los conjuntos de datos. Las arquitecturas de CNN propuestas aprovechan arquitecturas GPU para poder calcular

eficientemente la convolución unidimensional tanto en las fases de entrenamiento como predicción. Luego se miden los resultados de los modelos obtenidos para seleccionar una solución. La figura 1.2 muestra el diagrama de alto nivel de la solución desarrollada.

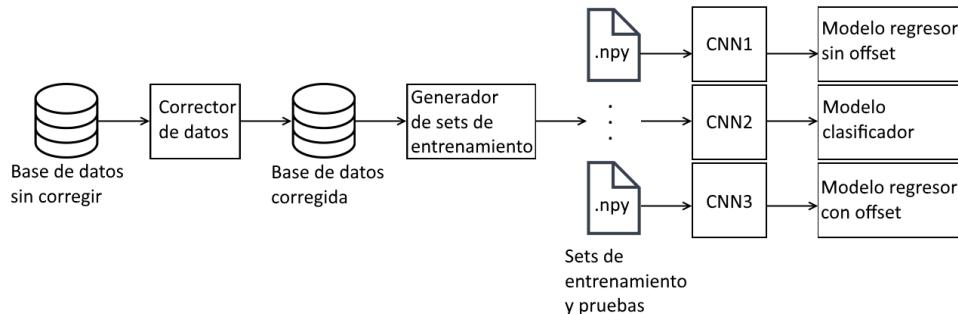


Figura 1.2: Diagrama de alto nivel de la solución desarrollada.

1.1. Objetivos y estructura del documento

El objetivo de este proyecto es desarrollar una solución algorítmica basada en redes convolucionales que genere modelos que permitan acoplar formas a imágenes con estructuras vermiformes. Adicionalmente busca generar una herramienta que permita encontrar y corregir errores en los datos de entrenamiento de forma manual, e implementar un algoritmo que permita realizar la corrección de manera automática no supervisada.

1.1.1. Objetivo general

- Proponer una solución algorítmica que permita acoplar modelos de forma explícitos a la información de imágenes con estructuras vermiformes.

1.1.2. Objetivos específicos

- Generar una herramienta que permita encontrar y corregir errores de etiquetación de los datos de entrenamiento de forma manual.
- Implementar un algoritmo que permita realizar la corrección de errores de manera automática no supervisada.
- Diseñar una función objetivo que modele la calidad de ajuste de una forma concreta a una imagen.

En el capítulo 2 se realiza un análisis del estado del arte en las áreas de segmentación de imágenes de nematología y se desarrolla el marco teórico necesario para entender la solución. En el capítulo 3 se explica la solución desarrollada. En el capítulo 4 se analizan

y discuten los resultados obtenidos con los algoritmos desarrollados. Y finalmente, en el capítulo 5 se recopilan las conclusiones obtenidas, se analizan los factores que intervinieron en el proyecto y se discute sobre los trabajos futuros que podrían desarrollarse.

Capítulo 2

Marco teórico

2.1. Trabajos anteriores

Hacer coincidir una forma previamente aprendida con una imagen, es un problema altamente estudiado en el área de visión por computadora. Existen muchas estrategias para resolver este problema y la tendencia en los últimos años ha sido resolverlo para imágenes tridimensionales [6]. A pesar de esto, está lejos de ser un problema resuelto, pues aún existen entornos en donde la forma es muy variable y el ambiente poco controlado, lo que hace imposible aplicar los métodos clásicos sin hacerles algún tipo de ajuste.

Los modelos de forma se pueden subdividir dentro de dos categorías: los rígidos en donde la forma no cambia y los no rígidos en donde la forma es variable. El método rígido más conocido es el de **la transformada de Hough**, la cual originalmente fue planteada para encontrar círculos o líneas de un tamaño determinado [18]. Más adelante este método fue generalizado para detectar formas arbitrarias [4]. A pesar de que funcionan bastante bien, los enfoques con forma rígida no son aplicables en muchas áreas porque aun en objetos rígidos, la forma puede cambiar dependiendo del ángulo que en se capture la imagen y la iluminación.

Dentro de los modelos no rígidos se encuentran los siguientes métodos:

1. Los **modelos activos de forma (ASM)** se derivan de los modelos activos de contorno (ACM). La idea de este método es obtener la forma promedio mediante un proceso de entrenamiento en donde todas las formas han sido alineadas por medio de un análisis de Procrustes[13]. Luego, la desviación de las formas respecto a esa forma promedio se modela por medio de una proyección de los datos a un espacio de menor dimensión, usualmente calculado por medio del análisis de componentes principales (PCA). De este modo, las formas se parametrizan con los componentes principales y se ajustan a la imagen iterando el movimiento de hitos a bordes de la imagen, con

otra fase de restricción de forma y utilizando restricciones en la variación permitida en cada uno de los ejes principales.

ASM ya se ha utilizado en el proyecto de nematodos anteriormente [41], pero tiene la desventaja de que asume una variabilidad lineal en las formas y los nematodos cambian de manera no lineal. Los resultados de dicho trabajo se pueden apreciar en la figura 2.1.

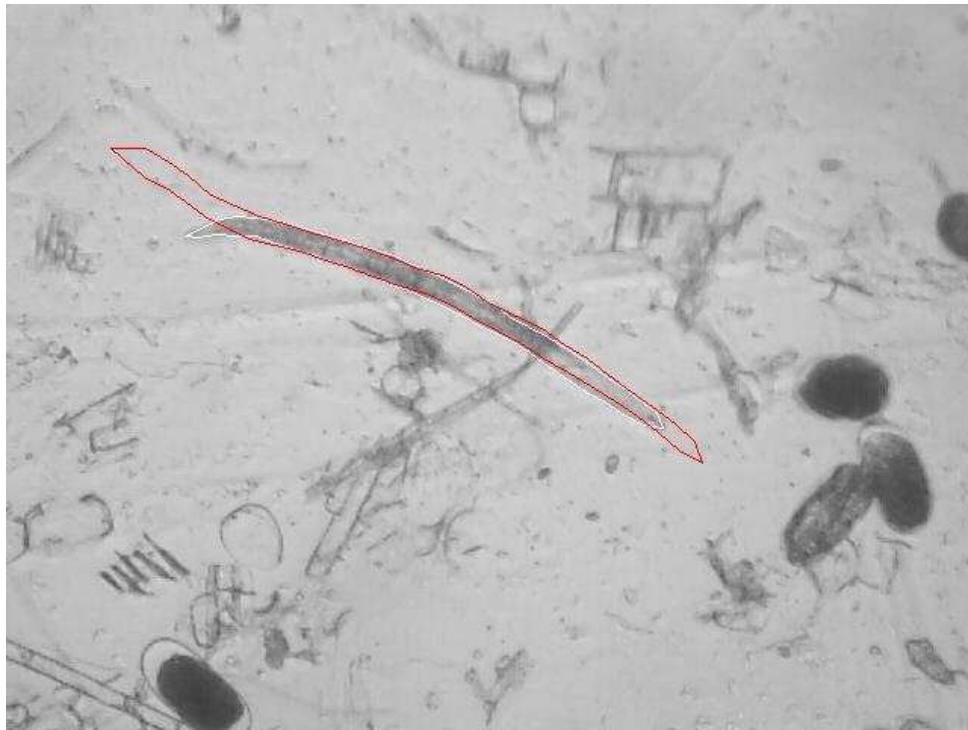


Figura 2.1: Resultado obtenido por Marín [41] al aplicar ASM a nematodos.

2. **Level sets** sigue un enfoque diferente al de ASM, ya que en lugar de buscar hitos en la imagen que delimitan una forma (modelo explícito), busca una función embebida en un espacio de más dimensiones que permite segmentar las formas en la imagen con curvas de nivel. En esta estrategia también se puede usar información previa de forma, que se incorpora mediante un modelo estadístico para determinar la función embebida [14]. El funcionamiento de *level sets* para rastrear una silueta humana se puede ver en la figura 2.2.
3. Los **espacios convexas de forma** son similares a *level sets* pero tienen la ventaja de que, al ser convexas, se puede optimizar globalmente con el descenso de gradiente de la función de error [15]. La principal diferencia es que se representa la función de forma como la probabilidad de que un píxel esté dentro de la forma (figura 2.3).
4. **Graph cuts** consiste en representar la imagen como un grafo y remover las aristas hasta obtener un numero determinado de conjuntos. Es un método que usualmente se usa para segmentar sectores similares de una imagen. Sin embargo, al incorporar

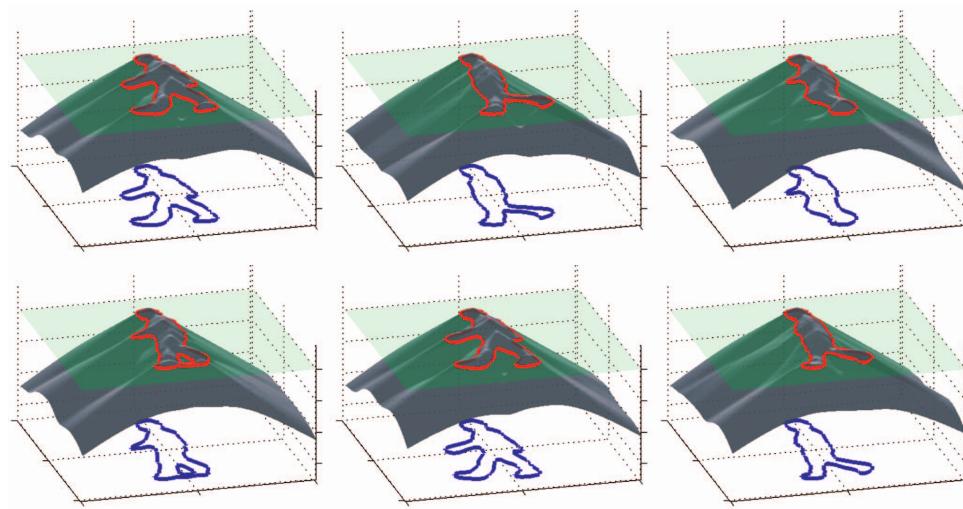


Figura 2.2: Superficies embebidas obtenidas a partir de un modelo estadístico basado en *level sets* [14].

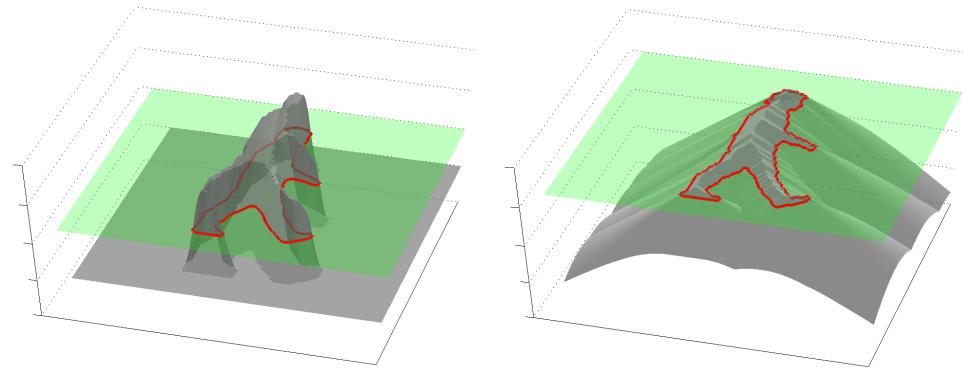


Figura 2.3: Diferencia entre la noción relajada de forma planteada en los espacios convexas de forma (izquierda) y la noción clásica de *level sets* (derecha) [15].

una penalización que depende de un entrenamiento previo con formas, se puede utilizar para segmentar formas [55]. Además, este método puede encontrar formas sobreuestas si se permiten varias etiquetas por píxel, como se observa en la figura 2.4. La desventaja es que este método requiere una inicialización cercana a la realidad y memoria adicional para la representación de la imagen como grafo.

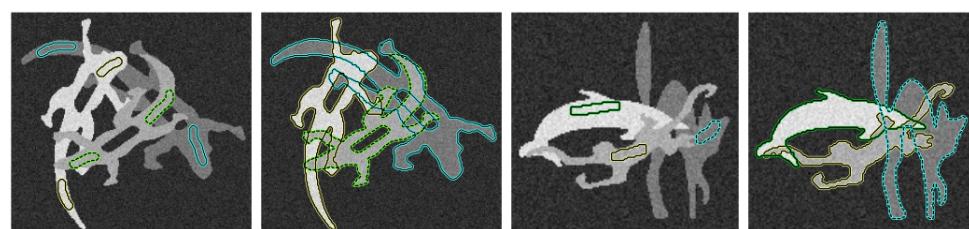


Figura 2.4: Resultado de aplicar *multiphase graph cuts* para segmentar múltiples figuras [55].

5. Los **modelos estadísticos de forma** son un método explícito que se diferencia de ASM porque parten de que existe cierto margen de error en los datos de entrena-

miento. Para solucionar esto permiten que el modelo de forma sea disperso. Asumen que cualquier forma va a ser una combinación lineal dispersa de los datos de entrenamiento. [59]. La ventaja de los modelos estadísticos es que no se sobreajustan a los datos de entrenamiento, por su naturaleza dispersa. Además, si se aplica un error aleatorio a los datos, se pueden generar transformaciones no lineales de las formas de entrenamiento [19].

En las conferencias de visión por computadora de los últimos dos años, el **aprendizaje profundo** ha sido la única solución publicada para este problema. Las redes de regresión densas como DenseReg [27], YOLO [45] [32], entre otras, han dominado los artículos sobre ajuste de forma a la imagen. Sirven hasta para encontrar formas 3D deformadas a partir de imágenes monoculares. La idea en todos estos artículos es similar: utilizar una red neuronal convolucional como regresor y a la salida aplicarle algún tipo de operación para permitir transformaciones estadísticas de la forma [27]. En la figura 2.5 se pueden ver los resultados obtenidos con este método.



Figura 2.5: Resultados obtenidos por DenseReg al ajustar un modelo 3D de rostro [27].

Es importante mencionar que aunque la mayoría de estos métodos fueron concebidos para trabajar en 2D, sus ideas son aplicables a 3D. Además existen varios enfoques para reconstruir figuras 3D a partir de imágenes 2D. “Para la reconstrucción de la forma de un objeto a partir de una sola imagen 2D, *shape-from-template* busca aparear un modelo 3D a la imagen por medio de la proyección 3D a 2D” [6].

En el área de nematodos y de microscopía en general, las técnicas que se aplican están atrasadas varios años. Mientras que en reconocimiento facial y clasificación de imágenes se siguen aplicando técnicas cada vez más modernas de *deep learning*, en el área de microscopía de nematodos se siguen aplicando técnicas básicas de segmentación. El atraso que existe en esta área se debe probablemente a la escasez de datos de entrenamiento, lo cual impide aplicar la mayoría de las técnicas modernas, y también a que las imágenes de microscopía presentan sus propios retos que no existen en imágenes a mayor escala. La tabla 2.1 resume trabajos relevantes en el área de nematología que realizan de alguna forma seguimiento automático.

Cho [10] realiza un análisis de las principales tendencias del procesamiento automático de datos en nematología. Específicamente resume en la tabla 2.2 las principales técnicas

Objetivo	Técnica		Referencias
Rastreo simple	Umbral	Umbralización que divide la imagen en fondo y nematodo con el menor error. Seguimiento por pseudoesqueleto.	Shingai(2000)[47]
	Externo	Métodos que utilizan un dispositivo para el rastreo. Umbralizan tomando como fondo la intensidad de alguna esquina, aplican operaciones morfológicas y esqueletización	Baek(2002)[3] Geng(2003)[22] Feng(2004)[20] Cronin(2005)[16] Yemini(2013)[57]
	Heurística	La detección es manual, se realiza el seguimiento por la comprobación de casos conocidos.	Hoshi(2006)[28]
	NEMO	Un filtro paso bajas reduce el ruido, umbralización, operaciones morfológicas eliminan pequeños grupos de píxeles y se utiliza el esqueleto para rastreo.	Tsibidis(2007)[52] Ramot(2008)[44]
	PCA	Análisis de componentes principales para separar el fondo del nematodo.	Buckingham(2009)[8]
	Umbral	Umbralización basada en varianza a través del tiempo.	Hurst(2014)[29]
	PCA	Rastreo externo, Eigenworms para clasificar posturas.	Venkatachalam(2016)[54]
Rastreo múltiple	Umbral	Umbralización y <i>flood-fill</i> desde los píxeles más oscuros	Swierczek(2011)[51]
	Lucas - Kanade	Método diferencial para flujo óptico	Marcellino(2012)[40]
	PCA	Segmentación manual, rastreo por análisis de componentes principales con un vector de 65 dimensiones	Stroustrup(2013)[50]
Reconocimiento	SVM	Umbralización local Niblack, descriptores de área, textura y perímetro. Clasificación con una máquina de soporte vectorial	Zhan(2015)[58]
Segmentación	TFM	Descriptor de textura e intensidad en un marco de trabajo probabilístico. <i>Markov random fields</i> con píxel promedio y 38 filtros.	Greenblum(2014)[24]
Reconocimiento y segmentación	SRS	Modelo de forma para segmentación y reconocimiento simultaneo	Qu(2011)[43]

Tabla 2.1: Resumen de técnicas, aplicadas en artículos científicos, para el procesamiento de imágenes en nematología.

de procesamiento de imágenes. El autor menciona además que el aprendizaje profundo representa una potencial fuente de innovación en esta área. Por otro lado destaca que SVM y los modelos de forma ya han sido aplicados exitosamente para identificar la sinapsis y otros objetos dentro del nematodo.

Técnica	Requiere entrenamiento?	Tiempo de ejecución	Dificultad de implementación	Usada para
Umbralización	No	Despreciable	Trivial	Segmentación básica
Filtrado	No	Bajo	Fácil	Remover ruido, remover fondo, mejorar segmentación, detección de bordes
Transformadas	No	Bajo	Moderada	Remover ruido, análisis, segmentación, detección de bordes
Aprendizaje de máquina	Si	Alto	Retadora	Análisis de datos e imágenes, segmentación, detección de objeto
Aprendizaje profundo	Si	Alto	Retadora	Análisis de datos e imágenes, detección de objetos, clasificación de imágenes.

Tabla 2.2: Resumen de técnicas de procesamiento de imágenes [10].

En el proyecto de nematodos del SIPlab, durante los años 2008 y 2009[41] se avanzó en la creación de una herramienta de software para análisis automatizado de imágenes de nematodos, gracias al trabajo realizado en tres tesis de licenciatura y una tesis de maestría. Posteriormente se propuso mejorar la herramienta desarrollada anteriormente por medio del uso de un modelo no lineal de forma, trabajo planteado en una tesis de maestría en el año 2015[25]. Actualmente se propone desarrollar una forma de aplicar el modelo de forma planteado en el 2015 a la imagen, y evaluar el método desarrollado.

El M.Sc. Jorge Arroyo se encuentra realizando su investigación de doctorado en conjunto con el SIPLab en el proyecto de nematodos. Su trabajo consiste en aplicar modelos de forma no lineales para resolver el problema de segmentación automática de nematodos. El enfoque y el marco metodológico que sigue Arroyo son diferentes a lo que plantea este proyecto, sin embargo la base de datos recopilada por él se utiliza para el entrenamiento y la verificación.

2.2. Spline

El problema del spline consiste en definir una superficie lisa a través de una serie de puntos en un espacio conocido [21]. Dicha superficie es usualmente definida por medio de

un polinomio paramétrico de orden conocido, tal que se minimice el error de interpolación. Dada una matriz P de $n \times m$ puntos distintos en el espacio, La tarea es construir una superficie lisa que pasa a través de todos los puntos de P [21]. En dos dimensiones el problema se simplifica pues se busca una curva en lugar de un plano.

La idea principal de splines es que la interpolación se realiza por bloques de n puntos, asegurando que la unión entre bloques sea suave. Es decir que tanto el valor de cada función interpolada, como el de la primera y segunda derivada (o gradientes y hessianas), sean iguales en los bordes entre bloques.

2.3. Derivada del Gaussiano

La idea del DoG surge de mezclar las propiedades de dos filtros en una sola convolución. El filtro Gaussiano actúa como un filtro paso-bajas para eliminar el ruido o construir espacios de escala [34] [56]. Los filtros derivadores permiten encontrar bordes por medio del gradiente. Los algoritmos de detección de bordes usualmente usan algún tipo de regularización Gaussiana para calcular el gradiente [9]. La selección del espacio de escalas permite discriminar adecuadamente los elementos de la imagen.

El filtro Gaussiano multidimensional puede ser separado en múltiples filtros unidimensionales aplicados en los ejes principales [53]. El filtro Gaussiano unidimensional esta definido por la función gaussiana:

$$G(x; \sigma, \mu) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2.1)$$

Para efectos de la construcción del filtro gaussiano se toma la media en cero $\mu = 0$ y se pueden ignorar las constantes de normalización para simplificar la expresión. La normalización se puede realizar una vez se calcula el filtro dividiendo la matriz obtenida entre su norma algebraica:

$$G(x; \sigma) = e^{-\frac{x^2}{2\sigma^2}} \quad (2.2)$$

Al calcular la primera derivada de (2.2) se obtiene:

$$G(x; \sigma) = -\frac{x}{\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right) \quad (2.3)$$

La escala en la que se filtra esta definida por σ^2 , normalmente se selecciona con la aproximación:

$$\sigma^2 = \frac{s+1}{6} \quad (2.4)$$

Donde s es la escala (en píxeles) que actúa como frecuencia de corte.

Para obtener el gradiente en una dirección solo se tiene que aplicar la DoG en esa dirección y un filtro Gaussiano con la misma varianza σ^2 en las direcciones perpendiculares. Esto es particularmente útil para obtener el gradiente en x y en y para una imagen. La figura 2.6 contiene los filtros DoG para calcular magnitud, componente x y componente y de la primera derivada.

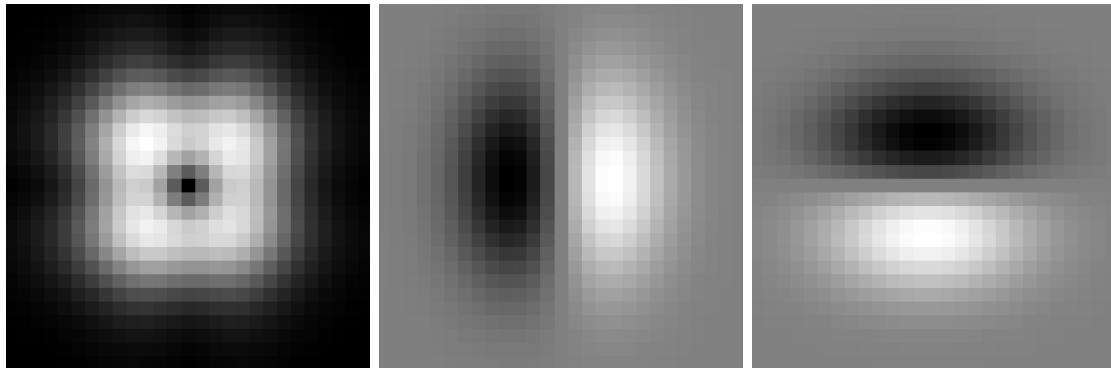


Figura 2.6: Filtros 2D de la derivada del Gaussiano. De izquierda a derecha: DoG para calcular magnitud del gradiente, DoG para la componente x del gradiente, DoG para la componente y del gradiente

2.4. Envolvente convexa

La envolvente convexa de un grupo de puntos es el grupo más pequeño que forma un polígono convexo que contiene todos los puntos [5]. Entre sus aplicaciones destacables están el cálculo rápido de diagramas de Voronoi, intersección de semiespacios en un punto y triangulación de Delaunay [5]. Esta última se utiliza en este proyecto para corregir datos de entrenamiento y el cálculo rápido de la envolvente convexa tiene un impacto en el rendimiento de dicha función. La figura 2.7 muestra un ejemplo de identificación de la envolvente convexa para un conjunto de puntos.

El algoritmo básico para calcular la envolvente convexa de un grupo de puntos $A = [a_0, a_1, \dots, a_n]$ toma $n(m + 1)$ operaciones, donde $m \leq n$ [31]. Dichas operaciones son comparaciones de ángulos y por lo tanto son computacionalmente baratas. El algoritmo primero busca el primer punto de la envolvente convexa barriendo ángulos desde un punto aleatorio, desde cada punto se barren los ángulos en la misma dirección hasta encontrar el siguiente punto. Se eliminan los puntos para las siguientes iteraciones si ya son parte de la envolvente convexa o si están dentro del área entre el punto inicial y el último punto encontrado [31]. La figura 2.8 explica geométricamente el algoritmo básico. La implementación utilizada en este proyecto forma parte de la biblioteca Qhull y usa el algoritmo *quickhull* [5] para calcular la envolvente convexa.

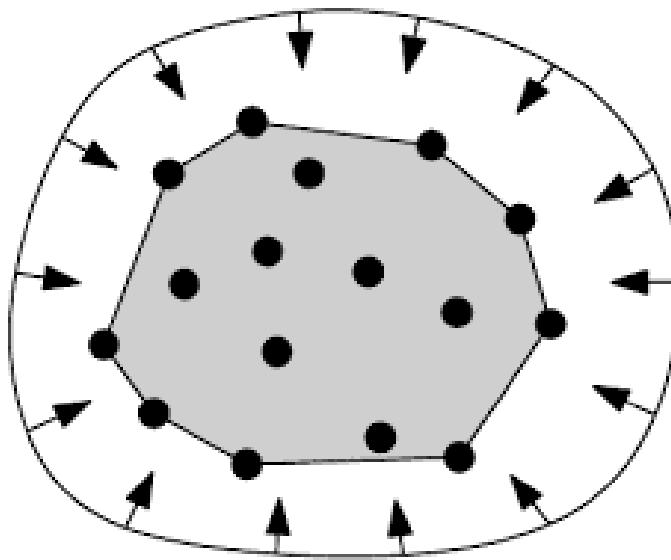


Figura 2.7: Ejemplo de la envolvente convexa [17].

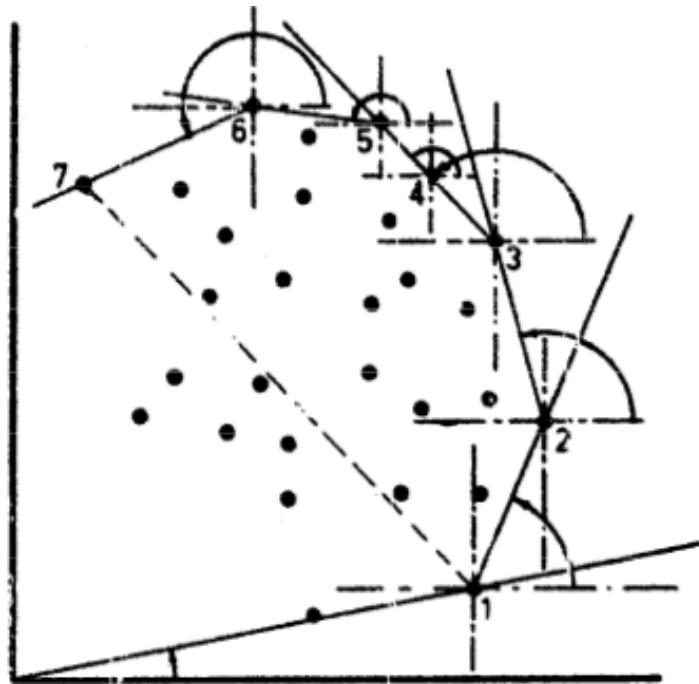


Figura 2.8: Interpretación geométrica del algoritmo propuesto por Jarvis para encontrar la envolvente convexa [31].

2.5. Triangulación de Delaunay

La triangulación de Delaunay es una triangulación con esferas circunscritas vacias: Consiste en considerar todos los subconjuntos de puntos $DT(P) \in P$ que forman un triángulo, tal que la esfera circunscrita en el triángulo esta vacía de todos los otros puntos $P - DT(P)$

[2]. Las esferas circunscritas consisten en trazar una esfera alrededor de un polígono de manera que su circunferencia toque la mayor cantidad de puntos posibles, en este caso los tres puntos que conforman cada triángulo de Delaunay.

La triangulación de Delaunay en \mathbb{R}^d puede calcularse a partir de la envolvente convexa en \mathbb{R}^{d+1} . Para determinar la triangulación de Delaunay de un grupo de puntos se deben mapear a un paraboloide y calcular su envolvente convexa. Los vértices de la envolvente convexa constituyen la triangulación de Delaunay del grupo original de puntos [7]. Utilizar la envolvente convexa para este cálculo reduce la complejidad aritmética considerablemente. La implementación de la triangulación de Delaunay disponible en la biblioteca Qhull utiliza esta optimización.

2.6. Problema del vendedor ambulante

El TSP es un problema de teoría de grafos que surge como una evolución del árbol extendido mínimo. Este consiste en proponer un método práctico para encontrar el árbol extendido de longitud mínima que cumple el siguiente teorema: Si a un grafo finito conectado se le asigna un número real positivo en cada arista (el peso o longitud de la arista), si cada peso es distinto, entre todos los árboles extendidos (camino que recorre todos los vértices del grafo), existe solo uno donde la suma de los pesos es mínima [36].

El TSP se diferencia del árbol extendido mínimo porque el árbol debe ser no ramificado [36]. Por lo tanto se transforma en un problema de encontrar la ruta mínima en un grafo cumpliendo las siguientes condiciones:

- Se deben visitar todos los vértices una vez
- El grafo debe ser conexo
- No pueden existir ciclos

Una de las primeras soluciones al TSP es el algoritmo de Prim [42]:

1. Seleccionar un vértice aleatorio V_0 para iniciar y se agrega al árbol extendido $A = \{V_0\}$
2. Considerar la arista con menor peso a entre el vértice actual V_0 y un vértice $V_1 \notin A$
3. Actualizar el peso del árbol $P(A) = P(A) + a$
4. Repetir los pasos 2. y 3. hasta que el número de vértices en A es igual al número de vértices del grafo.

Algunos caminos del algoritmo de Prim no logran llegar a una solución porque para un vértice V no existe una arista que cumpla la condición del paso 2. En este caso se debe realizar un proceso de *backtracking* hasta llegar a la solución.

2.7. Aprendizaje profundo

El aprendizaje profundo permite generar modelos computacionales que están compuestos por múltiples capas de procesamiento para aprender representaciones de datos con múltiples niveles de abstracción. El aprendizaje profundo busca la estructura subyacente en los datos mediante los parámetros de cada capa intermedia, los cuales se usan para calcular una nueva representación a partir de la representación de la capa anterior [37]. De aquí surge la principal ventaja del aprendizaje profundo, y es que mientras los métodos clásicos requieren casi siempre procesar los datos para obtener una representación intermedia o un vector de características, el aprendizaje profundo genera un modelo que recibe datos crudos y aprende con entrenamiento la mejor representación intermedia para los datos.

La teoría del aprendizaje profundo es conocida desde 1990 [38], con la primera aplicación de redes convoluciones con retropropagación para clasificar imágenes de dígitos. Sin embargo, el aprendizaje profundo tiene dos limitaciones principales que impidieron que se popularizara su uso anteriormente. La primera es que requiere una gran cantidad de datos para el entrenamiento. La capacidad que tienen las redes profundas para aprender representaciones intermedias viene al precio de tener que aumentar el número de parámetros del modelo. Un mayor número de parámetros implica que se requieran muchos más datos para entrenar. El segundo problema es que el entrenamiento de una red profunda requiere muchos recursos computacionales. Cada dato de entrenamiento debe ser pasado por una serie de convoluciones y cálculos lineales, para después utilizar alguna técnica de optimización que modifique los parámetros del modelo. Ambos problemas del aprendizaje profundo han sido solucionados en los últimos años. Gracias al internet se han generado cantidades enormes de datos en muchas áreas, aunque lamentablemente las imágenes de microscopía de nematodos no están incluidas entre ellas. Mientras que los avances en la capacidad de procesamiento, la computación de alto rendimiento con el uso de *clusters* y GPU para cálculos generales, se redujo considerablemente el tiempo requerido para entrenar este tipo de redes.

Un tipo particular de redes neuronales profundas son las llamadas redes neuronales convolucionales (CNN), en las cuales, para reducir el número de parámetros a determinar, las primeras capas se organizan como filtros digitales, en donde se aprovechan recursos computacionales de GPU para acelerar los cálculos involucrados.

A partir del 2012 el avance en el área ha crecido exponencialmente. En ese año se logró aplicar CNN para el reconocimiento de objetos, reduciendo el error considerablemente [35]. Desde entonces la comunidad del área de visión por computadora adoptó rápidamente el aprendizaje profundo y actualmente es ampliamente utilizado.

Cuando se habla de una red neuronal convolucional, normalmente se refiere a la arquitectura del modelo que produce. El cual debe contener como mínimo capas convolucionales y densas (completamente conectadas). El modelo de la CNN esta conformado por una serie de parámetros obtenidos durante el entrenamiento e información de las operaciones que se deben realizar con dichos parámetros. El modelo entrenado se puede evaluar, para

obtener una salida a partir de una entrada.

2.7.1. Capas convolucionales

El objetivo de las capas convolucionales es el de filtrar los datos de entrada, con máscaras aprendidas durante el entrenamiento, para resaltar ciertas características que le permitan a las capas superiores clasificar los datos. Son las encargadas de aprender la representación intermedia de los datos. Esta capa genera una serie de máscaras que se convolucionan con la entrada para producir un tensor a la salida. Esta capa está parametrizada por la cantidad de filtros utilizados, el tamaño de la máscara para filtrar y el tamaño del paso al filtrar [12]. Otros parámetros disponibles son el tipo de *padding*, la función de activación, inicializaciones y regularizadores [11].

Las convoluciones realizadas en estas capas implican una gran cantidad de multiplicaciones de matrices. Las multiplicaciones de matrices son operaciones que requieren una enorme cantidad de cálculos pequeños, por lo que las GPU son ideales para acelerar este cálculo. Las arquitecturas de GPU se diferencian de las de CPU porque tienen muchos más núcleos que, aunque son lentos, lo compensan con su gran número y memoria más rápida.

2.7.2. Capas de reducción

El objetivo original de las capas de reducción es disminuir la dimensionalidad de los datos. Esto se vuelve crucial cuando se tiene un gran número de parámetros en las capas convolucionales. Todas las convoluciones realizadas aumentan considerablemente el tamaño del tensor. En este tipo de capas lo único que se define es el tamaño de la vecindad que se considera para aplicar alguna operación de reducción[11].

La capa de reducción más popularmente utilizada es la de muestreo de máximos, propuesta originalmente por LeCun[39]. Existen otras opciones para la reducción, como el muestreo de medias que toma el promedio en la vecindad en lugar del máximo. El sub-muestreo es una generalización del muestreo de medias con pesos que se pueden entrenar. También se puede simplemente ignorar píxeles vecinos antes de la convolución [48].

Scherer [46] realiza una evaluación de las operaciones de reducción descritas anteriormente. En este trabajo se descubre que el muestreo de máximos permite que la red converja de manera más rápida y se reduzca el error de generalización. En conclusión el muestreo de máximos no es la única operación disponible, pero es la más simple computacionalmente y ha sido demostrado que es la que brinda los mejores resultados.

2.7.3. Capa de aplanamiento

La capa de aplanamiento (*flatten*) simplemente realiza un mapeo del tensor de salida de la última capa convolucional para que sea compatible con la entrada de las capas densas.

Consiste en aplastar las dimensiones extras del tensor, concatenando todos los vectores fila en un solo vector 1D. En los primeros artículos no era considerada una capa y simplemente se conectaban las salidas de la capa convolucional directamente a las capas densas[12].

2.7.4. Capas densas

Las capas densas combinan linealmente cada dato de la salida de la capa de aplanamiento en un vector de características 1D. Si la CNN es para clasificación, la última capa siempre es totalmente conectada con una sola unidad de salida por clase [12]. Los pesos de esta capa son adaptables y también se puede modificar su función de activación. Las neuronas en esta capa se comportan exactamente igual a las de un perceptrón multicapa.

2.7.5. Capas de expulsión

“Las redes neuronales profundas con una gran cantidad de parámetros son una técnica poderosa de aprendizaje de máquina. Sin embargo, el sobreajuste es un problema grave en dichas redes” [49]. La expulsión (*dropout*) surge como una forma elegante de evitar este problema mediante un tipo diferente de regularización. La idea es que durante cada paso hacia adelante se desactiven algunas neuronas. El porcentaje de dichas neuronas es predefinido en cada capa y su selección es aleatoria. Esto evita que las neuronas se adapten demasiado y crea una gran cantidad de redes diluidas [49]. Al final lo que se evalúa internamente es la respuesta promedio de muchas redes pequeñas.

Srivastava [49] determinó experimentalmente que una expulsión del 50% en todas las capas de la red es lo que minimiza el error de generalización. Esta adición a la red significa que se deben aumentar la cantidad de parámetros entrenables para llegar al mínimo error de entrenamiento y además se debe entrenar por más tiempo (aproximadamente 10 veces más [49]).

Cuando se utiliza *dropout* inmediatamente después de una capa de reducción, se debe ser más conservador con el porcentaje de neuronas que se desactivan. Por ejemplo, un muestreo de máximos con vecindad 2×2 ya está despreciando el 75% de los resultados de la capa anterior. Una expulsión del 50% en esta etapa significaría recortar el 87,5% de los datos, y se sale del rango recomendado de entre 50% y 80% [49].

2.7.6. Funciones de activación

La función de activación es una función que se aplica para obtener la salida de una neurona. En general las funciones de activación de las capas ocultas no tienen un gran impacto en el modelo obtenido. Solamente la función de activación de la última capa densa tiene un impacto importante. Algunas de las funciones de activación existentes son [11]: softmax, ELU, SELU (Unidad lineal exponencial escalada [33]), softplus, softsign,

ReLU, tanh, sigmoid y lineal. En la figura 2.9 se encuentran las funciones de activación más utilizadas.

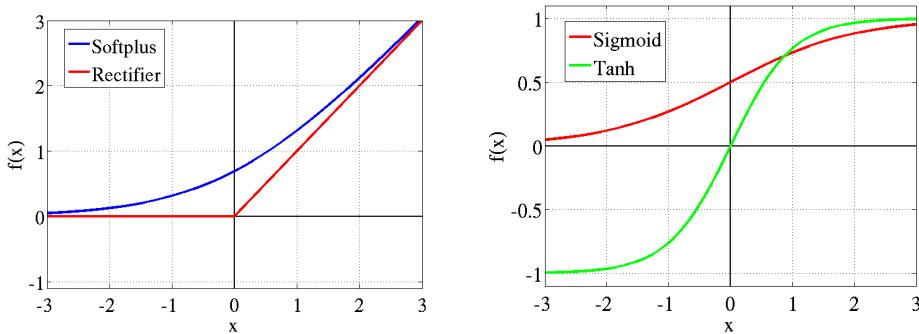


Figura 2.9: Algunas funciones de activación comúnmente utilizadas en CNN [23].

Se demostró que para redes neuronales muy grandes, usar la función de activación ReLU permite que la red se entrene mucho más rápido [23]. Es decir, que se llegue a los parámetros que minimizan el error de entrenamiento en una menor cantidad de ciclos de entrenamiento.

Capítulo 3

Sistema de ajuste de formas a la imagen

3.1. Representación de los datos

Los objetos bidimensionales son representados por una serie de puntos de interés ubicados a lo largo de su borde. Estos puntos de interés se denominan hitos (*landmarks*). Los hitos brindan una aproximación del contorno del objeto, que se reconstruye trazando un spline que los conecte. Para minimizar la pérdida de información con el spline se deben ubicar estratégicamente los hitos; ya que se requiere una mayor densidad de puntos para representar cambios abruptos en la forma. Esta pérdida de información se puede ver en la figura 3.1. De dicho spline se puede extraer la cantidad deseada de puntos equidistantes.

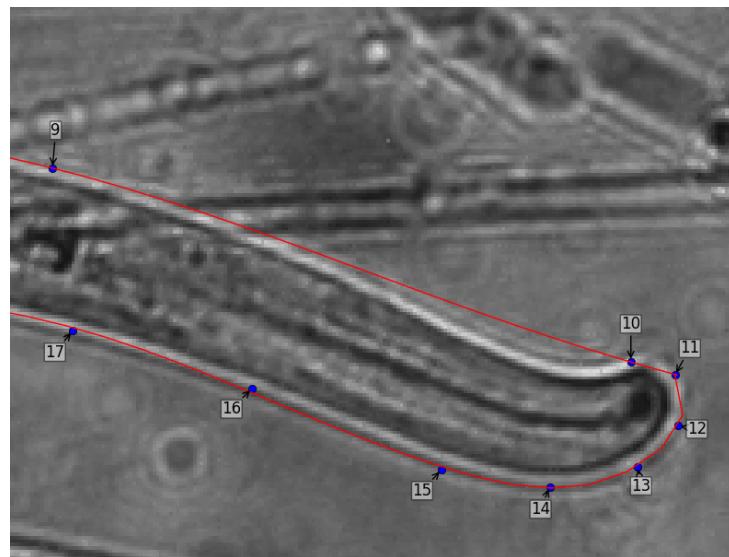


Figura 3.1: Pérdida de información por aproximación con spline.

Los n hitos son vectores en \mathbb{R}^2 que representan las coordenadas del punto en la imagen. Los hitos se combinan en una matriz $p \in \mathbb{R}^{nx2}$, tal que cada fila de la matriz p corresponde

a un hito.

Los cambios de mayor curvatura en el borde de un organismo vermiforme suceden en la cabeza y en la cola. Se debe evitar que estos cambios induzcan un error al calcular el spline. Para ello se realiza el cálculo en dos secciones, una por cada lado del nematodo. Cada subspline comparte los puntos inicial y final, que corresponden a la cabeza y la cola del organismo.

Otra consideración importante es ordenar los puntos de la matriz p de manera estándar. Se ordenan los puntos de tal manera que la cabeza siempre esté en la primera y la última fila de la matriz. Esto para facilitar la búsqueda de cola y cabeza a la hora de calcular los dos subsplines. Además, se ordenan los puntos en sentido horario para tener información sobre el interior y el exterior del nematodo dados dos puntos consecutivos.

A parte de los hitos es extraída información de ancho y largo promedio de cada nematodo. Esto con el fin de discriminarlos de acuerdo a su escala. El ancho se obtiene promediando la distancia entre puntos correspondientes de los dos subsplines. Y el largo mediante el pseudo esqueleto que pasa por el medio de cada par de puntos. En la figura 3.2 se puede observar el proceso para calcular ancho y largo.

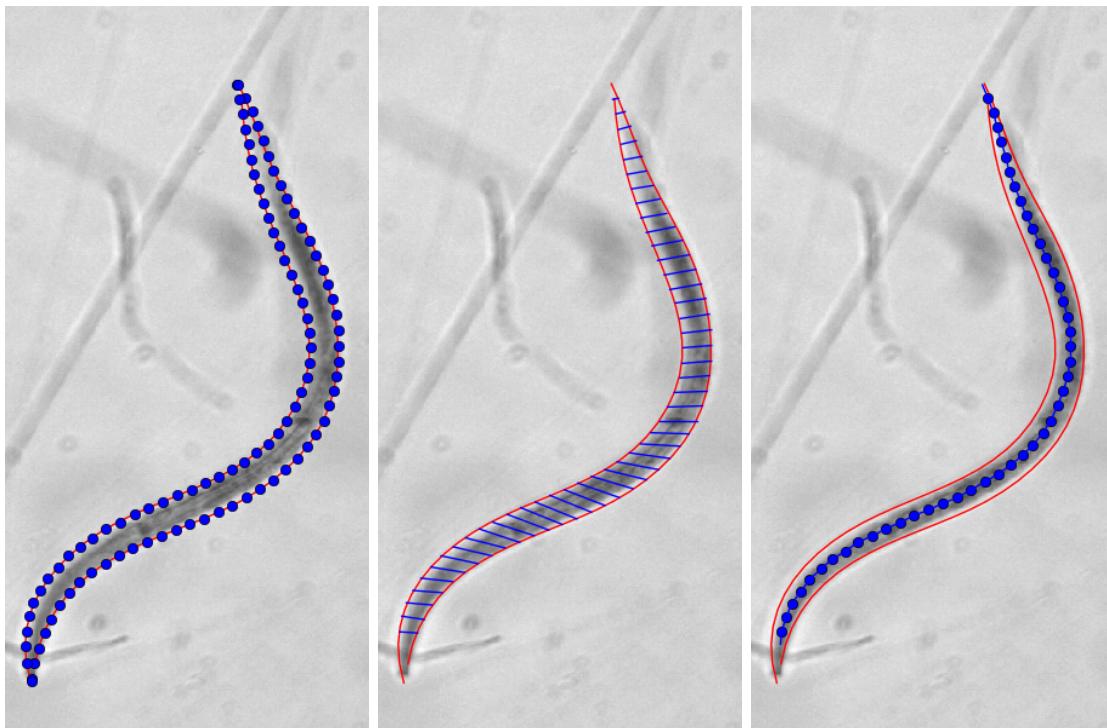


Figura 3.2: Cálculo del ancho y largo del nematodo. De izquierda a derecha: hitos, distancias entre puntos correspondientes, pseudo esqueleto.

3.2. Herramienta de corrección de los datos

La base de datos disponible para el proyecto consiste en 5785 imágenes en formato PNG de tamaño 1024×768 píxeles. Estas imágenes fueron extraídas de 13 videos distintos de microscopía y etiquetadas manualmente. De estas imágenes solo 4852 tienen sus respectivos datos de hitos, cola y cabeza. De estos datos 65 % presentaban algún tipo de error, lo que disminuye la cantidad efectiva de datos disponibles solo a 1673. Dado a la escasez de datos de entrenamiento y a la intención de utilizar aprendizaje profundo, fue necesario desarrollar una herramienta que permitiera corregirlos. Debido a que la corrección manual podría tardar mucho tiempo, es necesario encontrar una forma de detectar errores automáticamente y corregirlos.

El objetivo de la herramienta de corrección de datos es de detectar datos con alguno de los siguientes errores:

- Los hitos están en desorden (cruces entre sí o intercambiados de lugar)
- Los hitos están ordenados en sentido anti-horario
- La cabeza y la cola no están en las posiciones que deberían
- Los hitos están ubicados lejos del borde

Además también debe:

- Corregir errores automáticamente
- Llevar un registro de los datos que no pudieron ser corregidos para revisarlos manualmente
- Guardar los hitos siguiendo el estándar definido por conveniencia
- Mover los hitos y subsplines para que se ajusten mejor al borde

3.2.1. DoG dependiente de escala

Como parte del procesamiento de datos se filtra la imagen con un kernel de derivada del gaussiano dependiente de la escala. El ancho y la varianza de la función gaussiana utilizada para crear el kernel depende linealmente de un término de escala. De esta manera se aprovecha la propiedad del DoG que actúa como paso bajas, que se ajusta a la escala de cada imagen, a la vez que calcula la derivada direccional. El término de escala, determinado empíricamente, que brinda los mejores resultados es el ancho mínimo del nematodo, este término determina la frecuencia de corte del filtro. Tanto el ruido como el interior del nematodo son atenuados conservando la información del borde, a diferentes escalas.

3.2.2. Búsqueda de errores

La estrategia de búsqueda pretende reducir la carga computacional al mínimo. Para esto primero se prueban los métodos de solución a los errores más comunes que requieren menos cálculos. El diagrama de flujo para la búsqueda de errores se muestra en la figura 3.3.

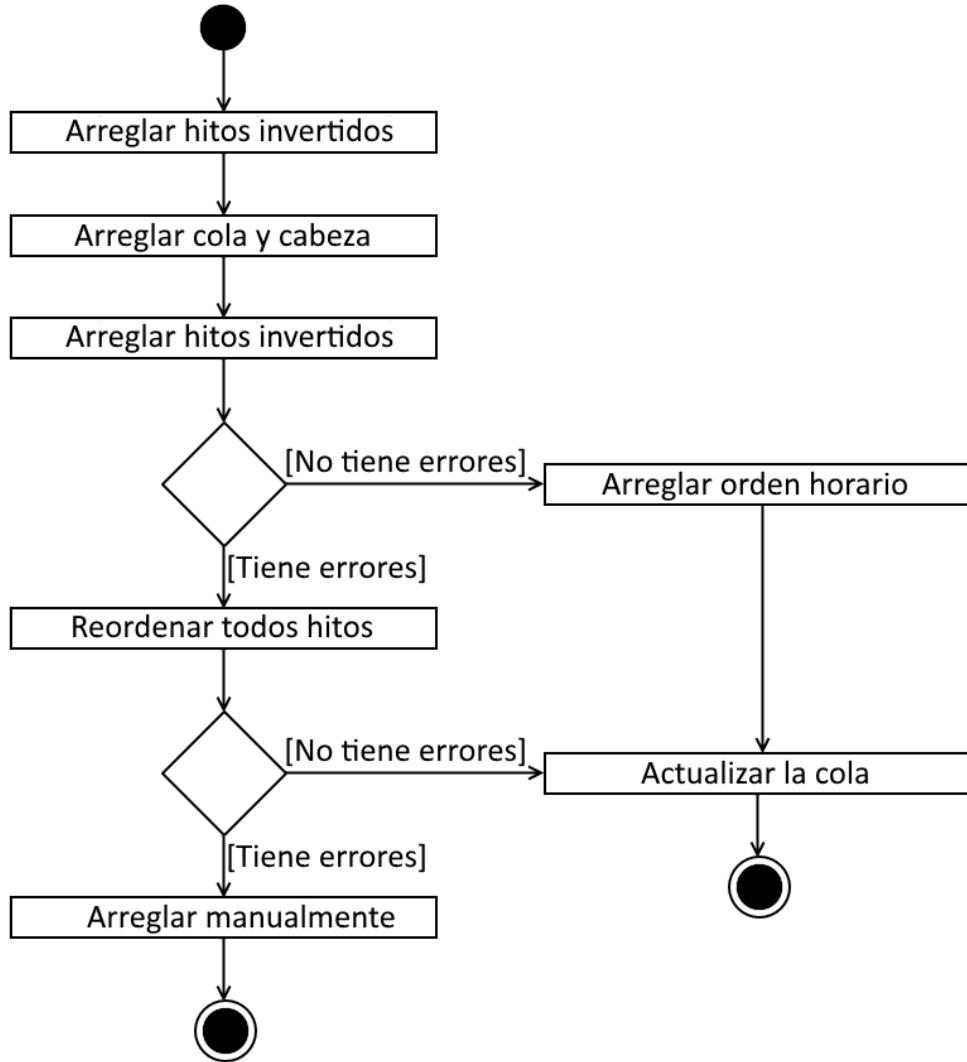


Figura 3.3: Diagrama de flujo de la corrección de errores

Un error que no se tomó en cuenta es que algunas veces la cola esta etiquetada como cabeza y viceversa. Esto se ignora pues no hace mucha diferencia para este proyecto porque lo que se busca es definir el contorno. En el futuro podría ser necesario arreglar estas etiquetas para usar los datos para entrenar un modelo de forma.

3.2.3. Arreglar hitos invertidos

El error más común en la base de datos es que solamente un par de hitos consecutivos están invertidos. El algoritmo que se sigue para detectar este tipo de errores esta basado

en la distancia Euclíadiana. Consiste en que para un grupo hitos $p_n, p_{n+1}, p_{n+2}, p_{n+3} \in P$, si estuvieran en orden, la distancia de p_{n+1} y p_{n+2} a sus respectivos vecinos debe ser menor que las mismas distancias al invertir los hitos. Este proceso iterativo se ejecuta por separado en cada subspline para evitar intercambiar puntos entre los dos lados del nematodo. Además no se toma en cuenta la cola ni la cabeza del nematodo.

3.2.4. Arreglar posición de cabeza y cola

Este error consiste en que los hitos están ordenados correctamente, pero la cabeza y la cola están en posiciones arbitrarias de la matriz de puntos. Usualmente se encuentran ambas al inicio o al final. La causa de este error es que la herramienta para etiquetar datos requería que se especificara cual punto de interés correspondía a la cabeza y cola manualmente. Entonces una práctica común fue primero marcar la cabeza y la cola y luego el resto de hitos o por el contrario hacerlo al final.

El algoritmo seguido para resolver este error es similar al anterior. Se eliminan los hitos correspondientes a la cola y la cabeza de la matriz de puntos y mediante la distancia Euclíadiana se busca la mejor posición para reinsertarlas. Finalmente se reordena la matriz con desplazamientos para que la cabeza quede en la primera posición y se inserta una copia al final. Esto último para seguir el estándar definido anteriormente para la representación de los datos.

3.2.5. Reordenar todos los hitos

Todos los algoritmos anteriores asumen que en su mayoría los hitos están ordenados y solo algunos de ellos presentan errores. Pero hay muchos casos en la base de datos donde los puntos están ordenados arbitrariamente. Para estos casos se requiere un algoritmo más robusto, que logre de un grupo de puntos sin orden alguno encontrar el camino que forme un polígono. Además este polígono debe seguir las reglas de forma del nematodo y pasar por todos los puntos.

La primera aproximación a este problema es utilizar la envolvente convexa para obtener un polígono inicial e insertar los puntos restantes de alguna manera. El problema es que muchas veces esto resulta en que los primeros puntos se inserten en una posición equivocada y sea imposible insertar el resto de los puntos siguiendo las reglas definidas. Los mecanismos de inserción que se siguen con esta estrategia son los siguientes:

- Insertarlos en la posición con menor distancia Euclíadiana a sus vecinos
- No permitir que se creen cruces entre las aristas
- Que se modifique lo menos posible el ángulo de las aristas de la envolvente convexa
- Penalizar más de una pasada por máximos del gradiente para evitar que la arista cruce al otro borde del nematodo

Para que el problema se pudiera solucionar en el tiempo disponible, se tuvo que asumir que el nematodo no tiene cruces en su borde. Es decir, el nematodo no puede pasar sobre sí mismo. En el ambiente donde se tomaron las imágenes es posible que esto suceda, sin embargo en la base de datos se encontraron muy pocas ocurrencias de este fenómeno (menos del 0.1 %). Por lo tanto esta suposición no representa una pérdida considerable de datos de entrenamiento. Un ejemplo de este caso se ilustra en la figura 3.4.

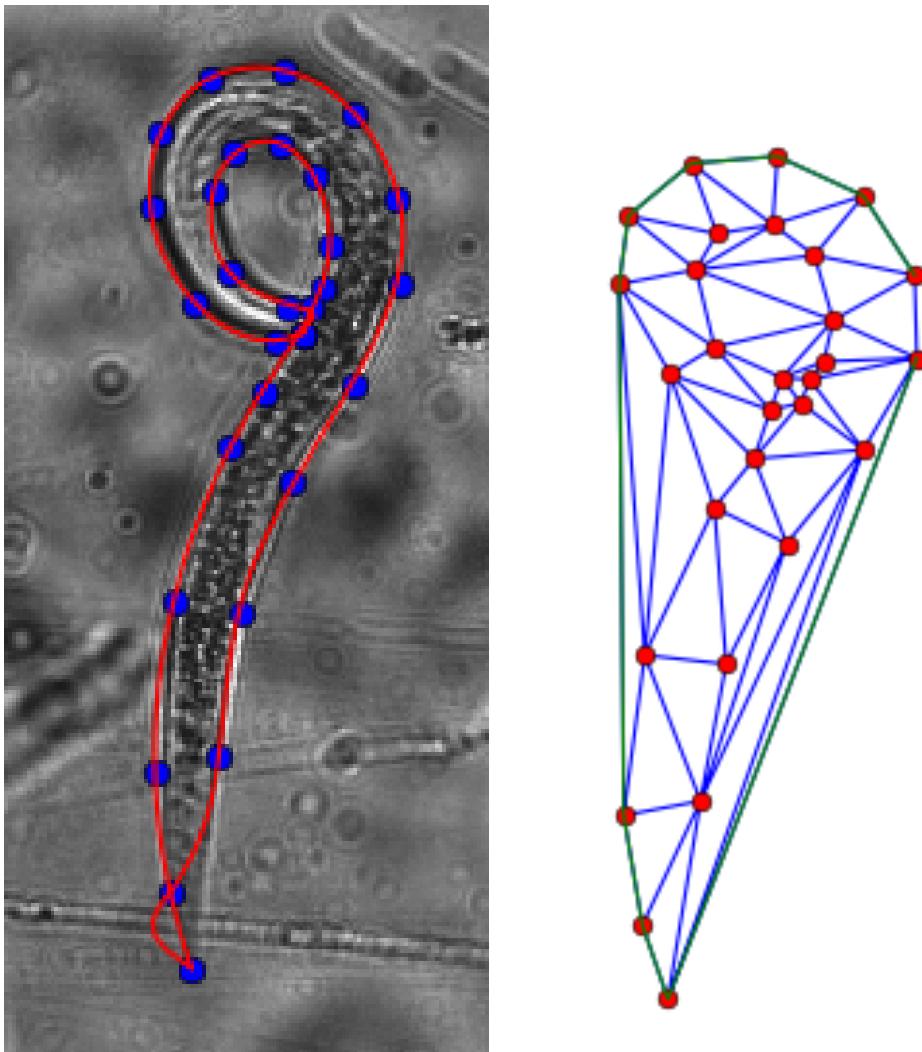


Figura 3.4: Ejemplo de nematodo que cruza sobre sí mismo

El otro enfoque requiere transformar el conjunto de puntos P de una matriz a un grafo. Si se extrae el grafo de puntos con todas las posibles aristas se puede usar alguna variación del TSP para crear el polígono. Esto presenta dos problemas: El primero es que TSP consume muchos recursos, es un problema NP-completo. El segundo es que en el grafo de todas las aristas posibles hay varias soluciones posibles con TSP, y se busca solo una específica (el borde del nematodo).

Para solucionar las dificultades con TSP se utiliza la transformada de Delaunay para reducir el número de aristas posibles y una aproximación del TSP con heurísticas basadas en el conocimiento de la forma de un nematodo. La parte más crítica es definir el peso

de las aristas del grafo de manera adecuada para balancear entre el valor del gradiente y la distancia. Se debe tomar en cuenta de alguna forma la magnitud del gradiente para asegurarse que siga por el borde del nematodo y no se pase al otro borde. También penalizar la distancia para que no se salte un punto por ir a otro que está más lejos pero sigue mejor el borde.

Se probaron varias opciones de peso para los grafos que toman en cuenta las dos consideraciones anteriores:

- Maximizar el gradiente en la dirección perpendicular al borde
- Maximizar la magnitud del gradiente
- Promediar el gradiente ya sea en la dirección del borde o perpendicular

También se probaron otras opciones que solo toman en cuenta el gradiente:

- Usar sumas cuadradas sin raíz
- Sumar 1 - el gradiente normalizado respecto a su máximo

Las sumas cuadradas crecen muy rápido entonces se priorizan distancias pequeñas, y con la suma del gradiente normalizado no se diferencian mucho puntos a la misma distancia.

El algoritmo desarrollado esta basado en el algoritmo de Prim para resolver el TSP y se le agregan algunas consideraciones, basadas en el conocimiento de forma del nematodo, para facilitar que llegue a una solución:

Algoritmo de ordenamiento Delaunay + Gradientes:

1. Generar un grafo con la triangulación de Delaunay de los puntos mediante la biblioteca Qhull.
2. Pesar las aristas de acuerdo a $\sum \sqrt{G \cdot d}$ (gradiente en la dirección del borde)
3. Empezar el recorrido en la cabeza
4. Obtener los 2 nodos alcanzables desde la cabeza con las aristas de menor peso.
5. Seleccionar el nodo en sentido horario
6. En el nuevo nodo:
 - 6.1. Si tiene 2 aristas o más:
 - 6.1.1. Los candidatos son los 2 nodos más cercanos
 - 6.1.2. Se elige el candidato que tenga mayor producto punto normalizado con la dirección que se venía siguiendo
 - 6.2. Si tiene 1 arista:

- 6.2.1. Se elige este nodo
- 6.3. Si no tiene aristas y aún hay nodos sin visitar:
 - 6.3.1. El algoritmo falló (el nematodo pasa sobre sí mismo)
- 6.4. Si no tiene aristas y ya no hay nodos:
 - 6.4.1. El algoritmo termina exitosamente
- 7. Se mueve al nodo seleccionado y se elimina el nodo anterior del grafo
- 8. Se repite 6 y 7 hasta llegar a una solución

3.2.6. Ajuste de los puntos al borde

La última corrección que se realiza es ajustar los hitos y los puntos del spline a la imagen. Para este ajuste se mueven los puntos al máximo del DoG en una dirección y ventana. La dirección esta determinada por la perpendicular a la línea entre los dos vecinos de cada punto. La ventana, por otra parte, se define manualmente basándose en la escala del nematodo y los resultados observados.

3.3. Redes neuronales convolucionales

Las capacidades de las redes neuronales convolucionales para clasificación y regresión son utilizadas aquí para crear dos modelos. El primero tiene como objetivo clasificar si un vector unidimensional extraído en las cercanías de un máximo de gradiente cualquiera contiene o no información de un nematodo. Y la segunda realiza una clasificación como aproximación de una tarea de regresión para determinar de un vector que se sabe contiene un nematodo, donde está el borde exactamente.

3.3.1. Extracción de datos

Se requiere desarrollar una estrategia de aumento de datos para poder utilizar CNN en la solución, ya que con 3865 datos se podrían entrenar muy pocos parámetros. Además se tendría que buscar una estrategia para reducir la dimensionalidad de los datos, pues se tienen muy pocos datos de muchas dimensiones. La estrategia que se seleccionó permite aumentar los datos y reducir su dimensionalidad al mismo tiempo. Esta extrae vectores de píxeles de longitud fija que corresponden al contenido de la imagen a lo largo de un segmento de recta perpendicular al gradiente. La idea es que con estos vectores se pueda entrenar una o varias CNN para identificar el borde del nematodo. Se utiliza una función gaussiana para atenuar los gradientes lejanos al nematodo. De esta manera se evita tomar como datos de no-nematodo a algún otro nematodo que aparezca en la imagen. Un ejemplo de los vectores tomados para entrenamiento, sin desplazamiento, se encuentra en la figura 3.5

Gracias a la interpolación con splines se pueden recalcular los hitos de manera que haya la misma cantidad en cada lado y la distancia entre ellos sea uniforme. Este paso es fundamental para entrenar la CNN porque evita que se extraigan datos redundantes. Además permite aumentar la cantidad de datos mediante la inclusión de nuevos hitos.

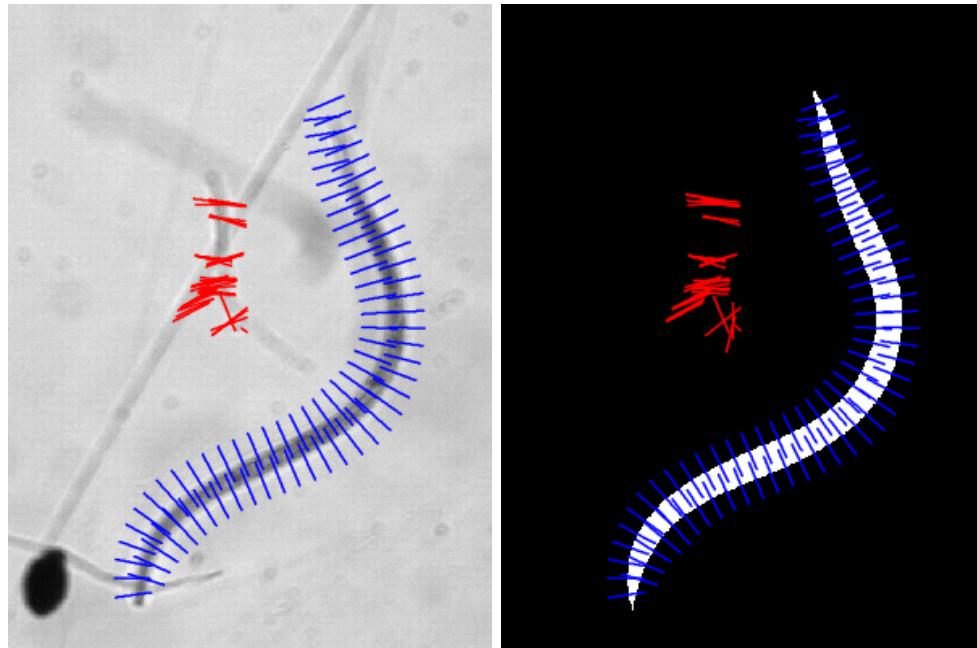


Figura 3.5: Vectores extraídos para entrenar las CNN. En azul se encuentran los de nematodo y en rojo los de fondo. A la izquierda se muestra la imagen de entrada y a la derecha la máscara obtenida en la primera parte del proyecto.

Se extraen varios sets de datos para probar distintas estrategias de solución. La primer diferencia entre los datos es el tamaño del vector que se toma. Se recurrió a la estadística del ancho promedio entre todos los nematodos (37 píxeles) para llegar a 3 medidas posibles: 18, 37 y 74. Hay que considerar además cuantos vectores se toman por nematodo, demasiados podrían resultar en información redundante y muy pocos en un faltante de datos. La otra determinación es si los datos se toman con o sin desplazamiento. Una red entrenada con datos desplazados es más robusta pero requiere más recursos para entrenarse. Y finalmente si los datos generados son para clasificación (hay o no un nematodo) o regresión (dónde está el nematodo). Para todos los grupos de datos se extrae aleatoriamente un 80 % para entrenamiento y un 20 % para pruebas, tomando en cuenta que se tengan datos de todos los individuos en cada porción. La tabla 3.1 realiza un síntesis de los conjuntos de datos extraídos para entrenar las CNN.

Los datos se toman sin realizar interpolación pues se espera que la CNN aprenda a hacerla de alguna manera más efectiva. En los sets de datos con desplazamiento, para los primeros se toman solo algunos desplazamientos aleatorios y para los demás se toman todos los desplazamientos posibles, esto para generar una mayor cantidad de datos. Para los datos que no son nematodo se toman puntos máximos del gradiente extraídos con una simplificación del detector de bordes Canny [9]. Algunos vectores se salen de la imagen, en especial los de tamaño 74. Estos vectores no se pueden tomar en cuenta y son descartados.

Descripción	<i>n</i>	<i>v</i>	Offset	Tipo	Total de datos
Sets de datos sin desplazamiento, el objetivo es centrar los datos en una etapa previa y por medio de una regresión determinar si es o no un borde de nematodo. También sirven para determinar cual tamaño del vector utilizar.	18 37 74	40 40 80	No No No	Reg Reg Reg	290238 288264 555747
Similar al set anterior pero tiene datos tanto de nematodos como de fondo con un desplazamiento aleatorio.	18 37 74	40 40 40	Si Si Si	Reg Reg Reg	290135 288074 553913
Set de datos con todos los desplazamientos posibles. Tiene como objetivo por medio de una regresión, en una red más grande, determinar tanto la existencia del nematodo como su desplazamiento.	74	80	Si	Reg	49828879
Tienen el mismo objetivo que el anterior, pero dividen la tarea en dos pasos. Primero determinar si hay o no un nematodo y luego su desplazamiento.	37 37	40 80	Si Si	Class Reg	9545106 10290201

Tabla 3.1: Propiedades de los sets de datos extraídos para entrenar las CNN. Donde *n* es el número de píxeles por vector y *v* el número de vectores por nematodo. En caso de tener datos de fondo o no nematodo también se tienen *v* vectores.

3.3.2. Entrenamiento y arquitecturas de red

Las arquitecturas se desarrollan combinando las capas disponibles en la biblioteca Keras¹ [11]: Conv1D, MaxPooling1D, Dropout, Flatten y Dense. Se combinan en dos tipos de bloques. Los convolucionales contienen dos capas Conv1D, una capa MaxPooling1D para reducir la cantidad de datos y una capa Dropout para evitar el sobreajuste. Después de esto se utiliza Flatten para convertir los datos multidimensionales en un vector y bloques totalmente conectados que consiste en una capa Dense seguida por una de Dropout. La tabla 3.2 muestra los bloques de construcción que se utilizan para diseñar los modelos. La cantidad de filtros por cada capa Conv1D o la cantidad de neuronas en Dense depende de la cantidad de parámetros que se quieran entrenar. La activación de la última capa densa usualmente es 'softmax', pero en este caso debe ser 'sigmoid' para la regresión.

Se sigue la estrategia sugerida por Srivastava [49] en el diseño de las redes. Primero se entrena una red con un número de parámetros hasta llegar al error óptimo de entrenamiento. Luego se duplica el número de parámetros agregando una expulsión del 50 % para evitar el sobreajuste y aumentar el error de generalización. El error en la red sin expulsión converge con entre 2 y 7 épocas. Se sigue la recomendación de Srivastava [49] y se entrena

¹El apéndice A contiene más información sobre esta biblioteca.

Bloque convolucional	Bloque denso
Conv1D(kernel=3, padding='same', activation='relu')	Dense(activation='relu')
Conv1D(kernel=3, padding='same', activation='relu')	Dropout(0,5)
MaxPooling1D(pool=2)	
Dropout(0,25)	

Tabla 3.2: Bloques de construcción para CNN. Entre los bloques convolucionales y los densos siempre debe haber una capa de Flatten para acoplar dimensionalmente las entradas y salidas.

con 100 épocas la red con expulsión, para garantizar que se llegue al menor error posible de generalización. En los casos donde se hace un muestreo de máximos para reducir el tamaño de los datos se hace una expulsión más conservadora del 25% para evitar que se pierda demasiada información y el modelo se pierda y no converja.

Se realizan 3 arquitecturas diferentes de CNN: La primera tiene como objetivo principal determinar el tamaño de entrada n que genera el menor error, las otras dos utilizan el tamaño determinado anteriormente e intentan dividir la tarea en clasificación y regresión para obtener mejores resultados. La cantidad de filtros en las capas convolucionales y la cantidad de neuronas en la capa densa, dependen del tamaño de la entrada y la salida. Esto con el fin de que a mayor número de dimensiones también haya mayor número de parámetros. Además se toma como medida que no hayan más parámetros que el 20% de los datos disponibles para entrenamiento. Los parámetros están distribuidos uniformemente entre la sección convolucional y la sección densa, y de manera incremental entre las capas de la sección convolucional y decremental entre las capas de la sección densa.

La primera arquitectura realiza una regresión con los primeros sets de entrenamiento (sin desplazamiento y con desplazamiento aleatorio), a diferentes tamaños de vector. Se utiliza un tamaño fijo en las capas convolucionales para evitar generar demasiados parámetros. El *padding* para las convoluciones en esta arquitectura es *valid*, lo que significa que en cada se ignoran los extremos y por lo tanto se reduce la dimensión en 2. El resumen de la arquitectura se encuentra en la tabla 3.3. La figura 3.6 contiene una representación gráfica de las operaciones aplicadas en la primera arquitectura de CNN.

La segunda arquitectura es similar a la anterior, pero se le hacen algunos ajustes para realizar una clasificación en lugar de una regresión. Primero se le agrega al final una capa de una sola neurona (para decidir nematodo o no nematodo). Luego se alteran los pesos de las últimas capas para distribuir mejor los parámetros. El resumen de la arquitectura se encuentra en la tabla 3.4

La tercera arquitectura es más grande que las anteriores. Se utilizan dos bloques convolucionales y dos bloques densos. Tanto la cantidad de filtros como la de neuronas están definida en múltiplos de 37 (dimensión de entrada), a diferencia de las arquitecturas ante-

Capa	Salida	#Param	Salida	#Param	Salida	#Param
Entrada	(18, 1)	0	(37, 1)	0	(74, 1)	0
Conv1D	(16, 32)	128	(35, 32)	128	(72, 32)	128
Conv1D	(14, 64)	6208	(33, 64)	6208	(70, 64)	6208
MaxPooling	(7, 64)	0	(16, 64)	0	(35, 64)	0
Dropout	(7, 64)	0	(16, 64)	0	(35, 64)	0
Flatten	(448)	0	(1024)	0	(2240)	0
Dense	(36)	16164	(74)	75850	(148)	331668
Dropout	(36)	0	(74)	0	(148)	0
Dense	(18)	666	(37)	2775	(74)	11026
Total	-	23166	-	84961	-	349030

Tabla 3.3: Resumen de las dimensiones de salida y los parámetros por capa de la primera arquitectura de CNN implementada

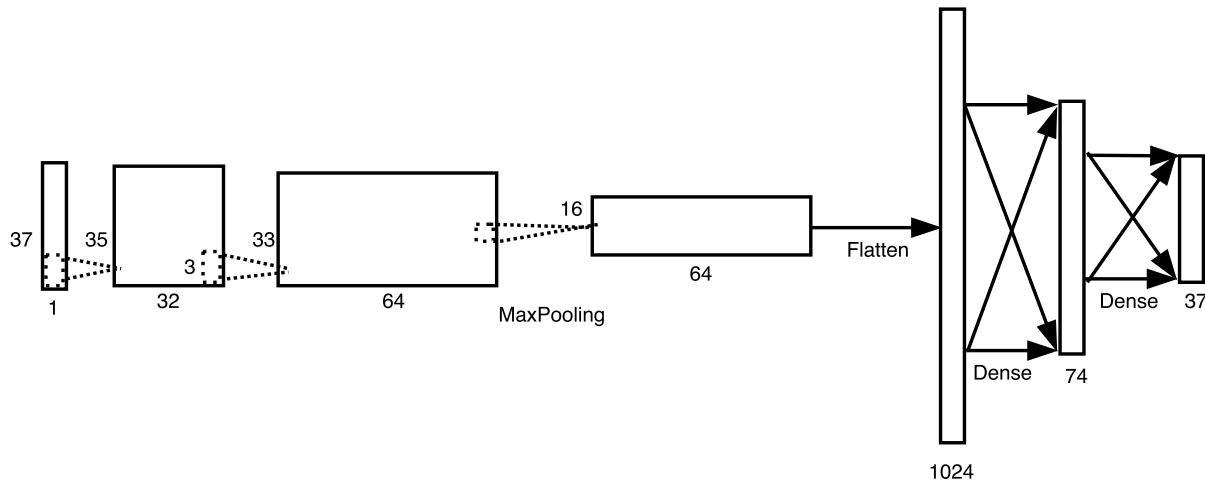


Figura 3.6: Diagrama de la primera arquitectura de CNN con vectores de 37 píxeles

riores donde algunos hiperparámetros son determinados arbitrariamente. El *padding* para las convoluciones se cambia a *same* para evitar perder la información de los bordes. Al igual que la primera se utiliza para regresión, sin embargo hay que destacar que esta arquitectura tiene muchos más parámetros y datos de entrenamiento, para intentar reconocer datos desplazados. Esta arquitectura también presenta una mejor distribución de parámetros entre sus capas y una mejor relación entre parámetros y datos de entrenamiento. El resumen de la arquitectura se encuentra en la tabla 3.5

Para el entrenamiento de las CNN se aprovechan las arquitecturas GPU para realizar convoluciones por medio de paralelismo a nivel de datos. El paralelismo de datos consiste en replicar la operación en cada núcleo y usar cada réplica en una fracción diferente de los datos de entrada. Keras tiene esta utilidad incorporada a través del *backend* de Theano, lo que permite crear una versión paralela del modelo y lograr un *speedup* casi lineal en las capas convolucionales [11].

Capa	Salida	#Param
Entrada	(18, 1)	0
Conv1D	(35, 32)	128
Conv1D	(33, 64)	6208
MaxPooling	(16, 64)	0
Dropout	(16, 64)	0
Flatten	(1024)	0
Dense	(64)	65600
Dropout	(64)	0
Dense	(32)	2080
Dense	(1)	33
Total	-	74049

Tabla 3.4: Resumen de las dimensiones de salida y los parámetros por capa de la segunda arquitectura de CNN implementada

Capa	Salida	#Param
Entrada	(37, 1)	0
Conv1D	(37, 74)	296
Conv1D	(37, 74)	16502
MaxPooling	(18, 74)	0
Dropout	(18, 74)	0
Conv1D	(18, 148)	33004
Conv1D	(18, 148)	65860
MaxPooling	(9, 148)	0
Dropout	(9, 148)	0
Flatten	(1332)	0
Dense	(111)	14796
Dropout	(111)	0
Dense	(74)	8288
Dropout	(74)	0
Dense	(37)	2775
Total	-	274688

Tabla 3.5: Resumen de las dimensiones de salida y los parámetros por capa de la tercera arquitectura de CNN implementada

El paralelismo a nivel de instrucciones consiste en ejecutar diferentes partes de un mismo modelo en diferentes núcleos del GPU. Este tipo de paralelismo funciona mejor en modelos diseñados con arquitecturas paralelas, por ejemplo, un modelo con dos ramas que se calculan simultáneamente. Este tipo de paralelismo no pudo ser explotado ya que se utiliza un modelo secuencial. El problema no es lo suficientemente complejo ni cuenta con

la cantidad de datos para justificar el uso de un modelo paralelo.

Arquitectura	Descripción	Entrenamiento	Parámetros
1	Regresión para encontrar la longitud del nematodo con el borde centrado y clasificación en nematodo y fondo	Datos de nematodo y fondo centrados	84961
2	Clasificación de vectores en nematodo o fondo	Datos de nematodo y fondo con desplazamiento aleatorio	74049
3	Regresión para encontrar la posición y longitud del nematodo	Solo datos de nematodo con desplazamiento aleatorio	274688

Tabla 3.6: Resumen de arquitecturas de CNN

Capítulo 4

Resultados y análisis

En este capítulo se presentan los resultados de las pruebas realizadas con las herramientas desarrolladas. Primero se presentan los resultados obtenidos con la herramienta de corrección de datos. Después, los resultados obtenidos con las diferentes CNN. Finalmente, un algoritmo que utiliza los modelos resultantes de las CNN para ajustar una forma a las imágenes de nematodos.

4.1. Corrección de datos

La detección de errores se realiza con dos algoritmos. El primero basado en la búsqueda de cruces en el spline y el segundo en que se mantenga el mismo sentido para cada trío de puntos (horario o anti-horario). Los errores detectados para cada archivo de la base de datos se encuentra en la tabla 4.1. Bajo la columna “Errores por cruces” se encuentran los errores detectados por el primer algoritmo y en “Errores por orden” los detectados por el segundo. Además, como dichos errores no son mutuamente excluyentes, se incluye en la tabla los datos que tienen alguno de los dos errores.

Como se puede ver en la tabla 4.1 el 65.5 % de los datos en el conjunto de entrenamiento presentan algún tipo de error. Si se llegan a corregir estos datos se podría duplicar la cantidad de datos disponibles para entrenamiento.

Los primeros algoritmos de corrección son 100 veces más rápidos que el algoritmo con triangulación de Delaunay. Los tiempos de ejecución de los tres algoritmos se encuentran en la tabla 4.2. Como se mencionó en el capítulo anterior los primeros algoritmos pueden corregir una gran parte de los datos. Como son mucho más rápidos, no representa una carga correrlos para intentar corregir cada dato. El algoritmo más pesado no toma tanto tiempo gracias al uso de Qhull para calcular rápidamente la triangulación de Delaunay y las aproximaciones incorporadas al algoritmo de TSP.

El error de hitos invertidos es bastante común en la base de datos. Consiste en que dos hitos vecinos se encuentran en posiciones opuestas. Se corrige mediante el algoritmo

Set	Datos totales	Errores por cruces	Errores por orden	# de datos erroneos	% de datos erroneos
file1	982	565	582	853	86.9
file2	287	155	129	218	76.0
file3	672	330	412	567	84.4
file4	365	85	112	170	46.6
file5	69	12	45	50	72.5
file6	502	71	237	273	54.4
file7	140	29	71	89	63.6
file8	727	105	356	402	55.3
file9	181	36	75	100	55.2
file10	18	4	8	9	50.0
file11	297	32	97	114	38.4
file12	474	69	212	249	52.5
file13	137	16	78	84	61.3
Total	4851	1509	2414	3178	65.5

Tabla 4.1: Resultado de la detección automática de errores en cada uno de los sets de datos de entrenamiento.

Algoritmo	Tiempo (s)
Corrección de hitos invertidos	0.002208944
Corrección de la posición de la cola y la cabeza	0.002564385
Reordenar todos los hitos	0.247633273

Tabla 4.2: Tiempos de ejecución promedio de los algoritmos de corrección de datos en un procesador Intel Core i7-2637M @ 1.70GHz

descrito en la sección 3.2.3. Un ejemplo de este error junto con su corrección mediante el algoritmo implementado se encuentra en la figura 4.1.

La posición de la cola y cabeza del nematodo también suele estar mal. En la mayoría de los casos ambas se ubican al inicio o al final de la lista. Se corrige mediante el algoritmo descrito en la sección 3.2.4. Un ejemplo de este error junto con su corrección mediante el algoritmo implementado se encuentra en la figura 4.2.

Por último, en los casos donde aún después de los algoritmos baratos, se siguen detectando errores, se ejecuta un reordenamiento de todos los puntos. El algoritmo basado en la triangulación de Delaunay y TSP se describe en la sección 3.2.5. En la figura 4.3 se muestra un nematodo que tiene los dos errores más comunes y fueron corregidos sin necesidad de ejecutar el algoritmo de reordenamiento. Por el contrario, la figura 4.4 muestra un nematodo que no pudo ser corregido con los primeros algoritmos y fue necesario ejecutar el reordenamiento.

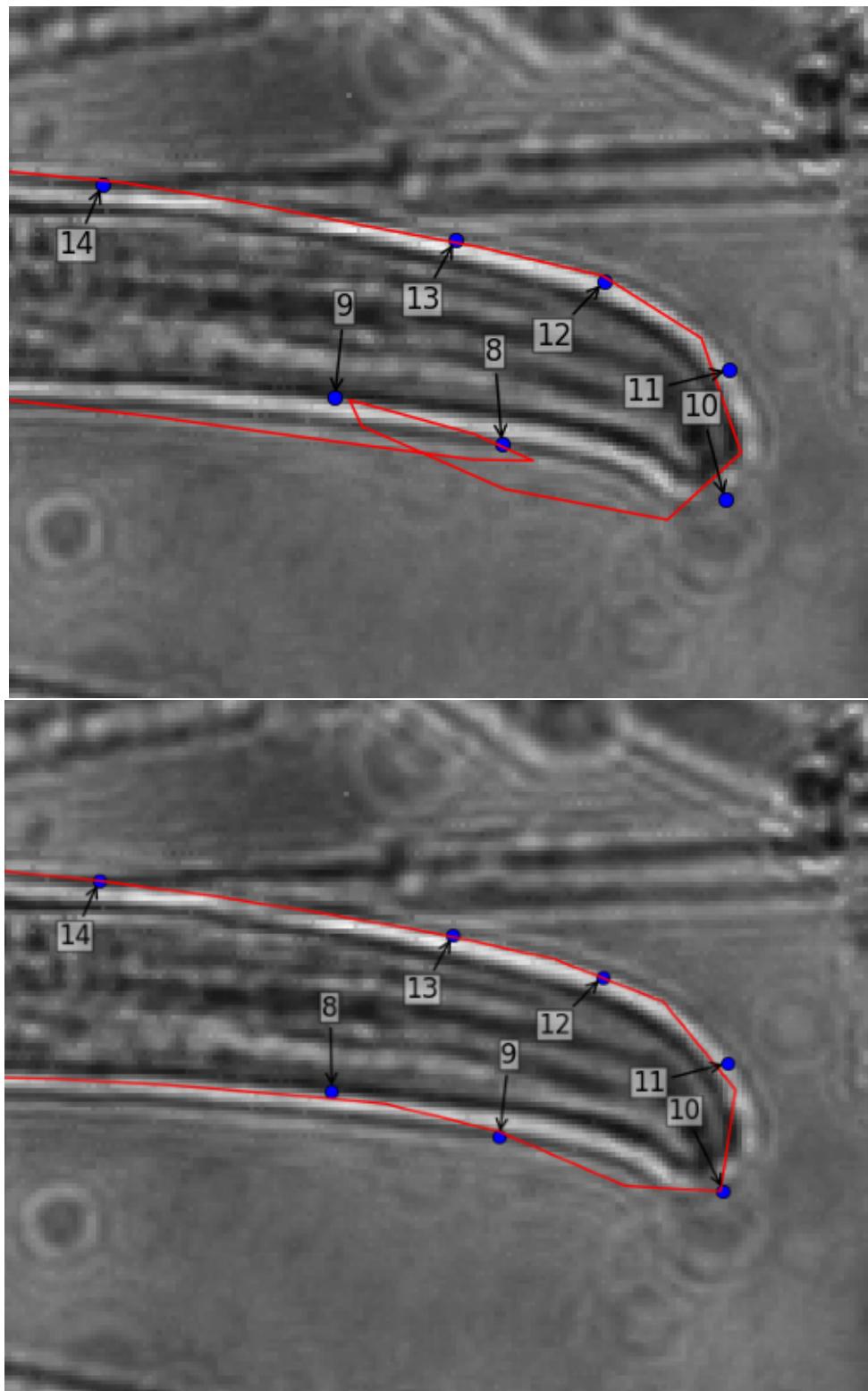


Figura 4.1: Resultado del algoritmo de corrección de hitos invertidos

En la tabla 4.3 se muestran los resultados de aplicar la herramienta de corrección automática de datos a la base de datos. En dicha tabla se puede observar que se logra corregir de manera completamente no supervisada la mayoría de los datos que presentan

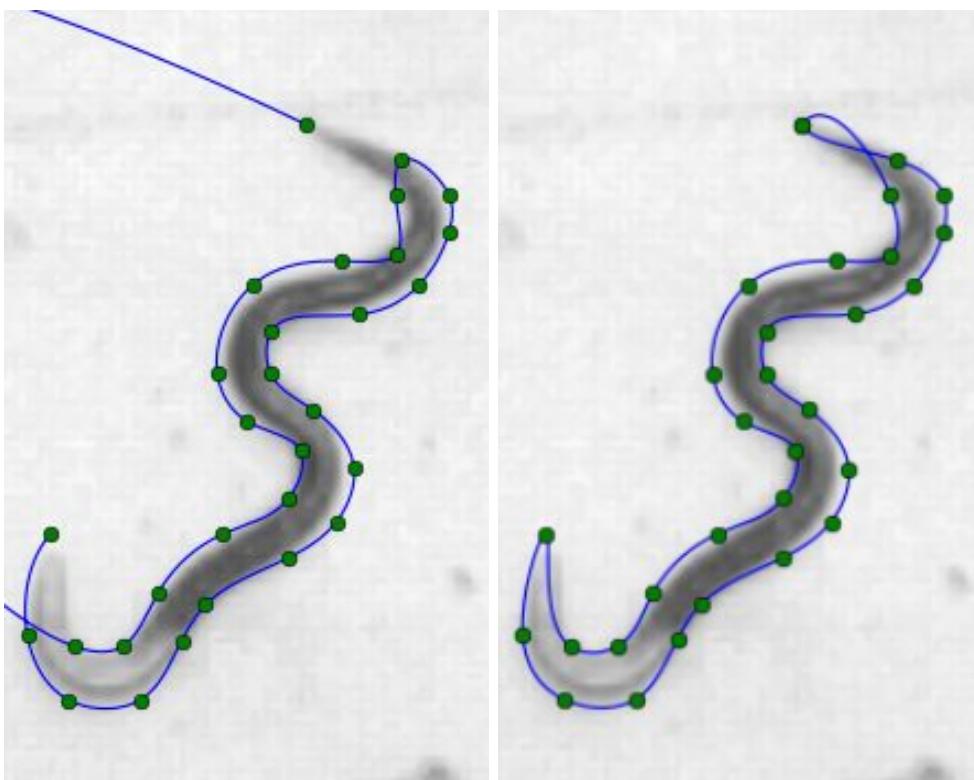


Figura 4.2: Resultado del algoritmo de corrección de la posición de cola y cabeza

errores. Se logró corregir automáticamente el 70 % de los datos que presentaban algún tipo de error.

Datos	# de datos	% de datos
Total	4851	100.00
Errores por cruces	1509	31.11
Errores por orden	2412	49.76
Algún tipo de error	3178	65.51
Sin errores	1673	34.49
Corregidos	2192	45.19
Sin corrección	986	20.33
Sin errores tras la corrección	3865	79.67

Tabla 4.3: Resultado de la corrección automática de errores en la base de datos.

Solo 986 datos (20 % de los datos totales) siguen presentando errores después de la corrección. La herramienta permite corregir manualmente estos datos, sin embargo, se tarda aproximadamente 3 minutos por cada dato, lo cual significaría casi 2 semanas de trabajo invertidas solo en corregir estos 986 datos. Se decide descartar estos datos por falta de tiempo.

Se pudo duplicar la cantidad de datos disponibles para entrenar las CNN. Arreglar manualmente estos datos, aparte del tiempo necesario para desarrollar una herramienta,

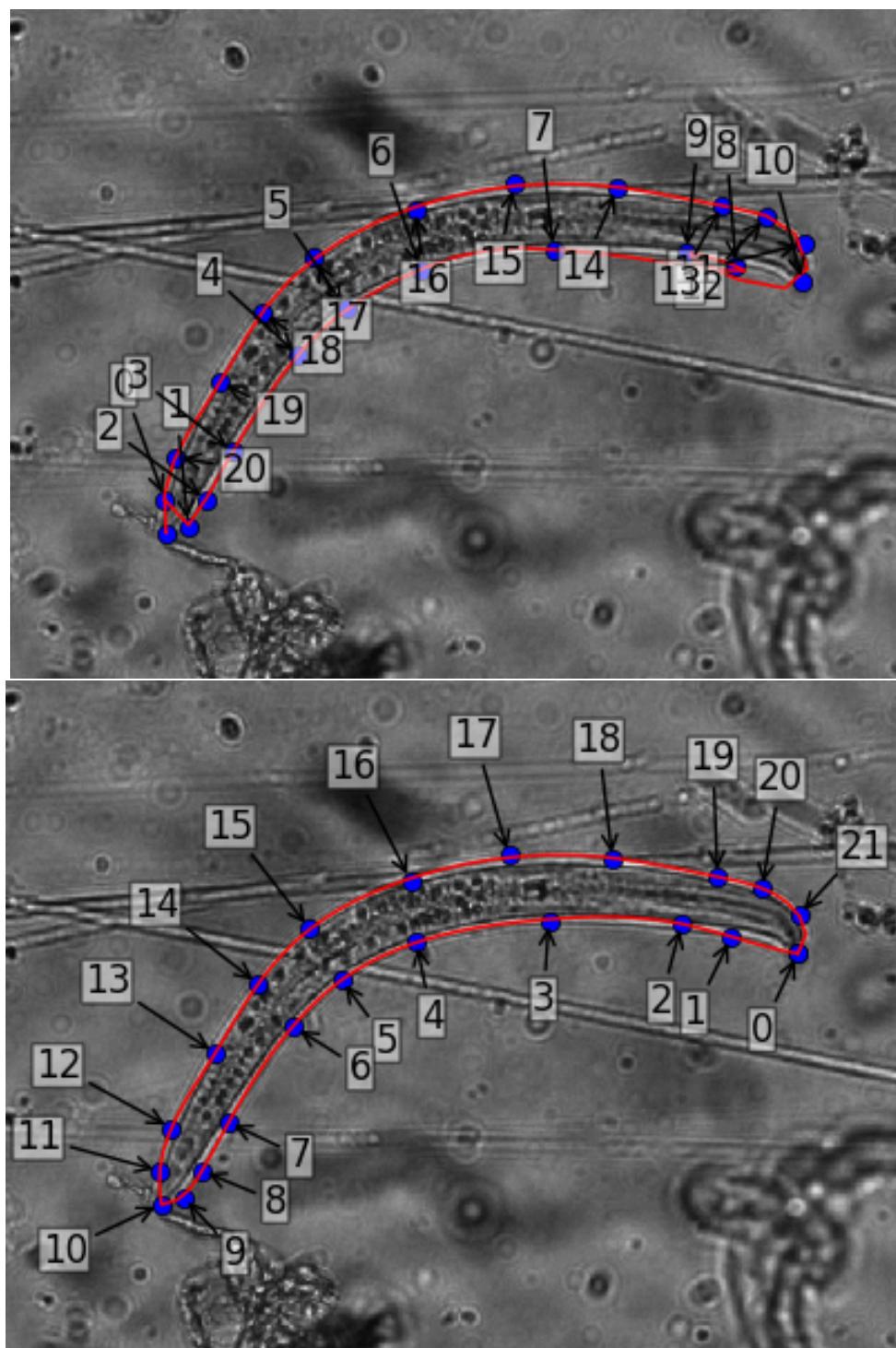


Figura 4.3: Resultado los algoritmos rápidos para corrección de errores

hubiera tomado aproximadamente 3 semanas de trabajo. Todo el proceso de los algoritmos de corrección se lleva a cabo en 2 semanas, por lo cual hay un ahorro considerable de tiempo. Se debe tomar en cuenta también, que esta herramienta se puede utilizar para etiquetar nuevos datos sin errores o corregir nuevos datos en trabajos futuros.

Después de corregir errores, se procede a ajustar mejor los hitos y el spline al borde del

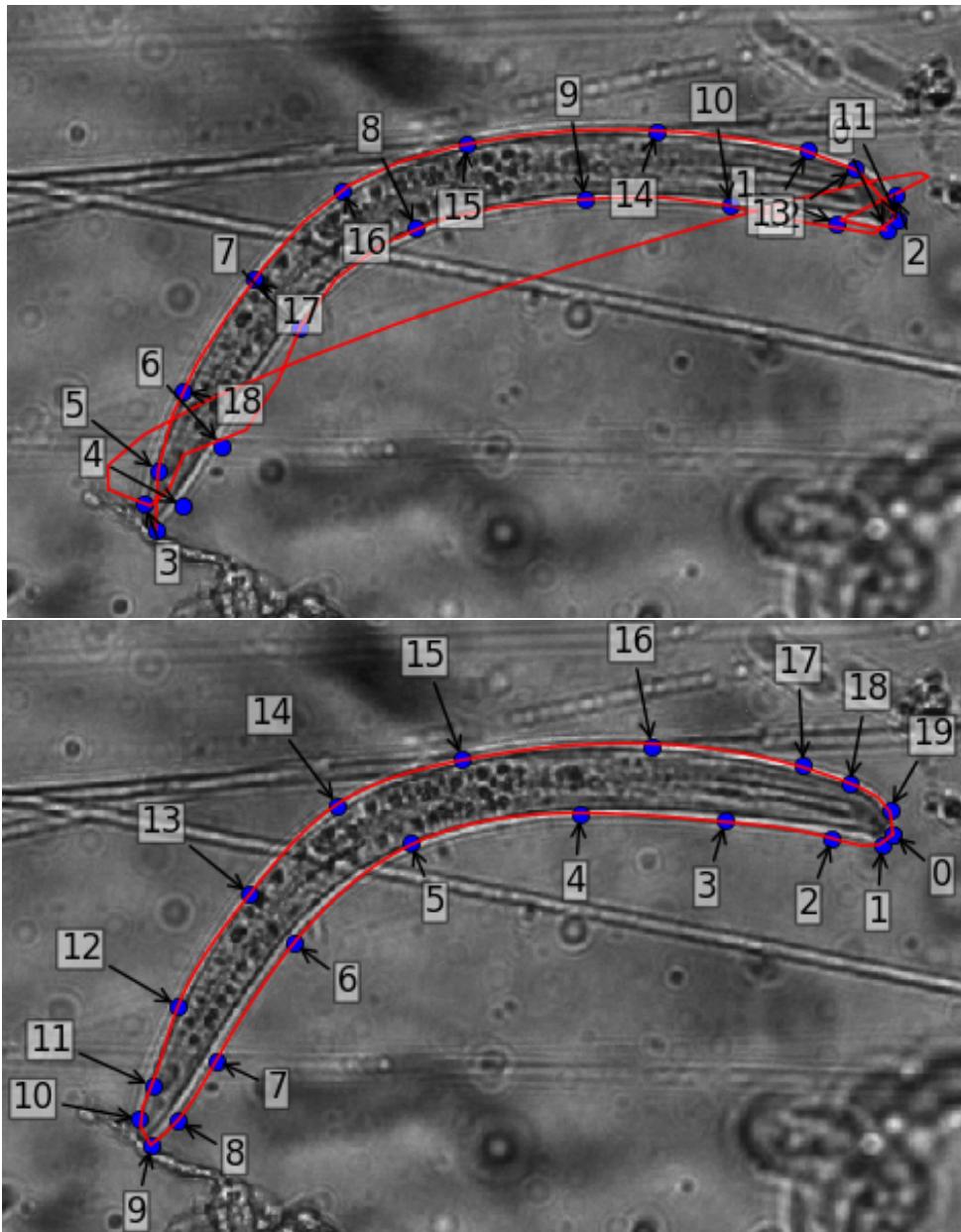


Figura 4.4: Resultado del algoritmo de reordenamiento total de hitos

nematodo. Esto con el fin de corregir puntos que fueron colocados de 1 a 4 píxeles del borde durante la etiquetación manual. En la figura 4.5 se puede ver como afecta este ajuste a los datos de entrenamiento.

El mejor ajuste se obtuvo con 2 píxeles. Darle la libertad al algoritmo de moverse una mayor cantidad de píxeles genera que tome en cuenta ruido, en algunos casos donde el ancho del nematodo es menor a 10 píxeles.

La maldición de la dimensión indica que al aumentar la dimensionalidad de los datos, la distancia en el espacio crece exponencialmente y los datos se vuelven dispersos. Por lo tanto, se requiere que la cantidad de datos aumente también exponencialmente para obtener resultados confiables. Debido a este efecto y a la cantidad de parámetros de la

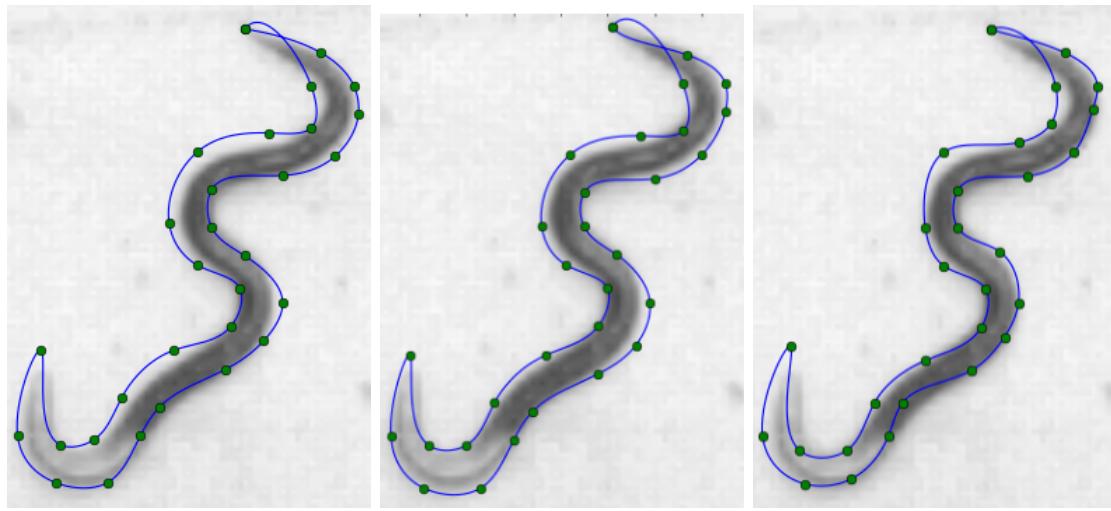


Figura 4.5: Resultado de ajustar los hitos al borde obtenido con DoG. De izquierda a derecha: 0 píxeles de ajuste, 1 píxel, 2 píxeles

CNN, se requieren más datos para entrenamiento que los que se tienen disponibles por nematodo. Se utiliza la interpolación con splines para regularizar la distancia entre hitos y aumentar la cantidad de datos. Una restricción presente es el nematodo con menor longitud del conjunto de entrenamiento (22 píxeles). En este caso utilizar más de 22 hitos por lado esta creando datos redundantes. Por este motivo se intenta no sobrepasar los 40 hitos por lado y solo en algunos casos utilizar 80. La figura 4.6 muestra el aumento y regularización de los hitos por medio del spline.

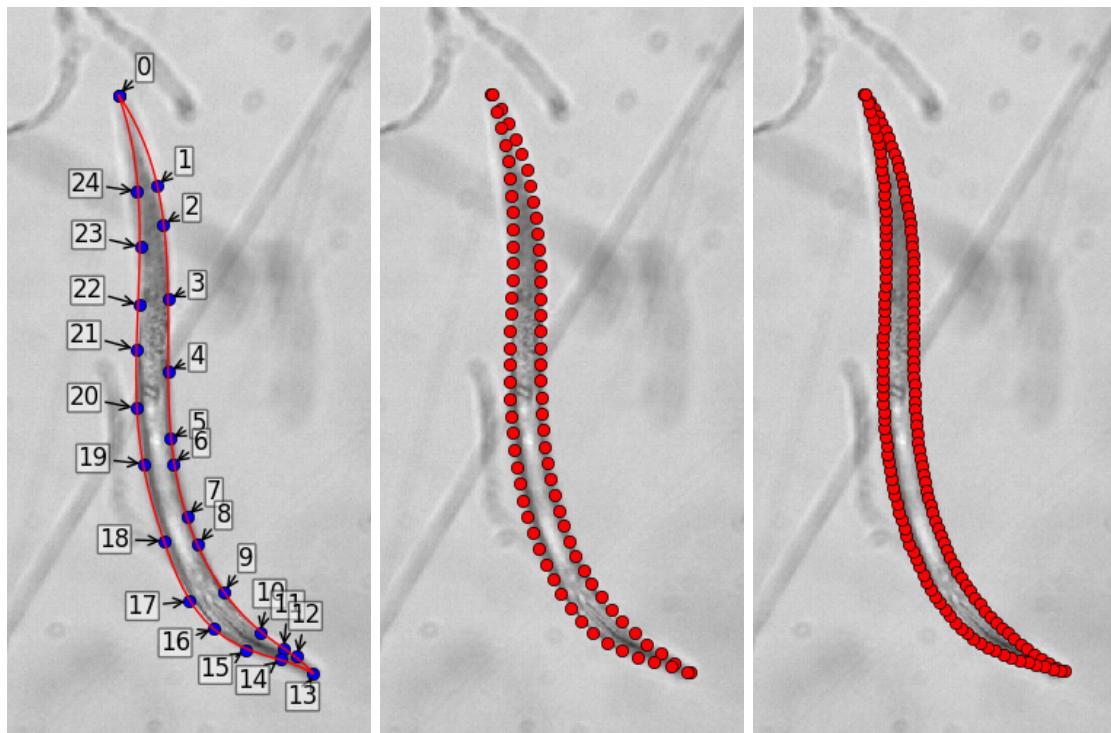


Figura 4.6: Aumento y regularización de los hitos por medio del spline. La imagen central muestra la interpolación con 40 hitos por cada lado y la de la derecha con 80.

4.2. CNN

4.2.1. Función de activación

Como predice la teoría, utilizar ReLU como función de activación logra mejorar la cantidad de épocas necesarias para que la red converja. Se realizaron las pruebas con la tercera arquitectura ya que es la que más tiempo tarda en cada época, corriendo en GPU, cada una tarda 297 segundos en promedio. Utilizando sigmoide en todas las capas la red tarda 51 épocas (15045s) en converger y utilizando ReLU tarda 16 épocas (4720s). En este experimento se utilizó una versión de la red sin expulsión para que convergiera más rápido.

4.2.2. Pruebas gráficas

Durante esta sección se utiliza la imagen de la figura 4.7 para demostrar gráficamente el funcionamiento de los modelos de CNN diseñados anteriormente. Esta imagen forma parte de la base de datos.

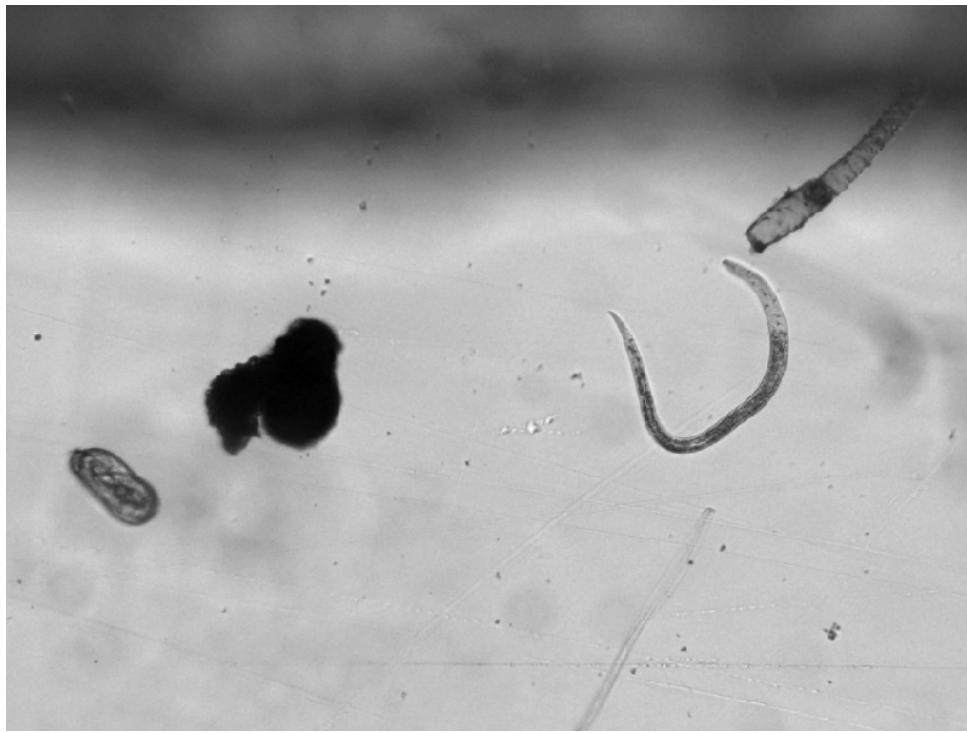


Figura 4.7: Imagen original utilizada para probar las CNN

Una forma de visualizar los resultados es evaluar vectores de la base de datos de pruebas y mostrar la respuesta esperada contra la respuesta producida por la red. Esto es lo que se realiza en las figuras 4.8, 4.9, 4.14 y 4.15. En dichas figuras primero se muestra el contenido del vector en escala de grises, debajo de él la respuesta esperada en escala de verde y negro, y por último la respuesta de la red en escala de azul y negro.

La figura 4.8 es el resultado de evaluar la base de datos sin *offset* que contiene tanto nematodos como fondo en la primera arquitectura. Los resultados de la ubicación del nematodo son muy cercanos a los reales, con solo 2 o 3 píxeles de error en algunos casos. Logra identificar correctamente los vectores que corresponden al fondo y regresar una respuesta nula.

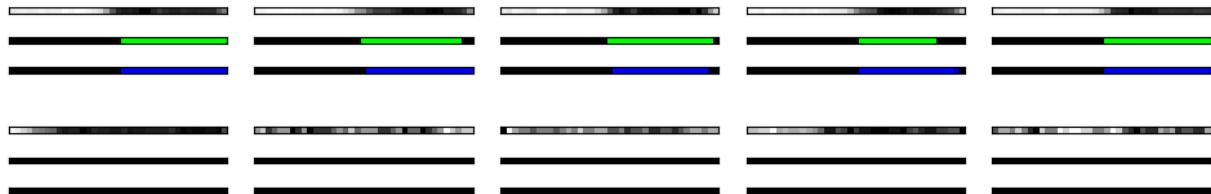


Figura 4.8: Evaluación de datos sin offset en la primera arquitectura de CNN.

La figura 4.9 es el resultado de evaluar la base de datos con *offset* que solo tiene datos de nematodo en la primera arquitectura. Los resultados tienen un error mayor al de evaluar los datos sin *offset*, esto debido a que la red nunca se entrenó para ver nematodos que no tuvieran su borde centrado en el vector, y por lo tanto los considera como fondo. En general, esta red logra ubicar bien los vectores con el borde a una distancia de máximo 3 píxeles del centro. Si se aplicara un preprocesamiento a los datos para eliminarles el desplazamiento se podrían mejorar estos resultados.



Figura 4.9: Evaluación de datos con *offset* en la primera arquitectura de CNN

La figura 4.10 fue obtenida mediante un proceso de remodelado de la imagen de entrada y evaluación en la primera arquitectura de CNN. Primero se modifica la imagen para formar vectores de 37 píxeles en cada fila, se evalúan en la red y se obtiene una respuesta. Luego se repite el mismo proceso pero extrayendo los vectores de cada columna. Finalmente se combinan las dos respuestas conservando el máximo y se sobreponen a la imagen original con una opacidad de 50 %. De esta manera se resalta con blanco los puntos que la red considera son nematodo y se atenua con negro el fondo.

La figura 4.11 muestra el resultado de extraer vectores de 37 píxeles en cada fila de la imagen y clasificarlos con la segunda arquitectura de CNN. La respuesta de la arquitectura es continua, donde un 1 representa 100 % de certeza de que la entrada es nematodo y un 0 lo contrario. Esta respuesta se umbraliza en distintos valores para que se pueda visualizar más fácilmente. Para cada umbral se resalta con un vector blanco de 37 píxeles los puntos en los que se encuentra un nematodo y se sobreponen a la imagen original con una opacidad del 50 %. La red solo reconoce nematodos en dirección perpendicular al borde, por este motivo, en esta prueba solo se detectan las secciones verticales del nematodo.

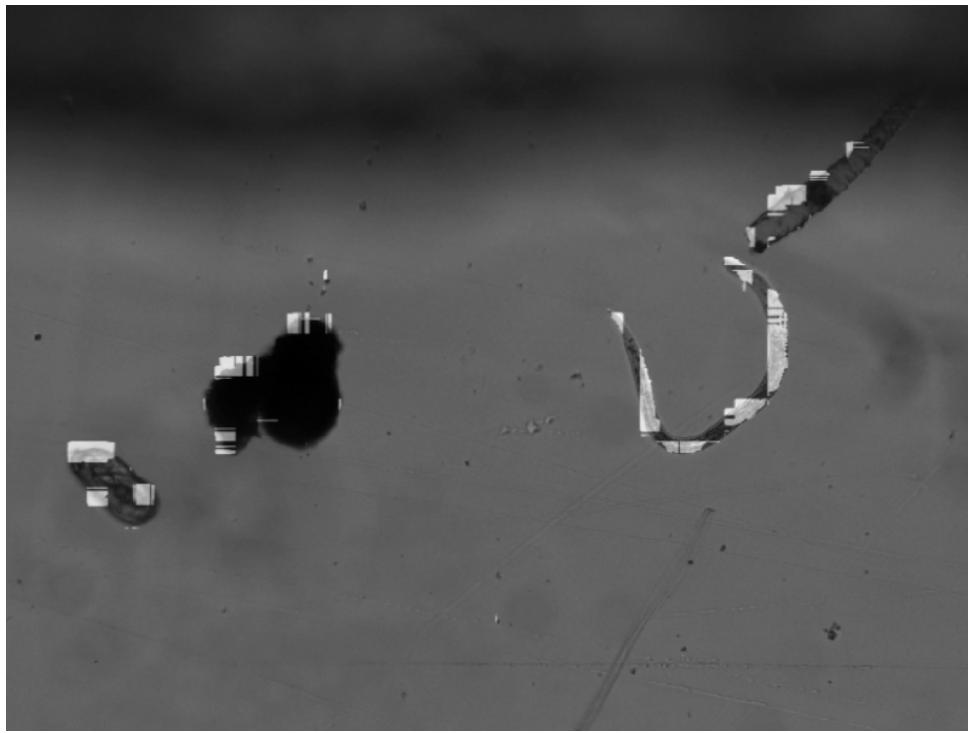


Figura 4.10: Resultado de evaluar una imagen en la primera arquitectura de CNN

En la figura 4.12 se utiliza el mismo proceso de remodelado y evaluación para visualizar el resultado de la tercera arquitectura de CNN. Solo se evalúan vectores horizontales de 37 píxeles, no se toma en cuenta la dirección vertical. Como la red fue entrenada solo con datos de nematodo, siempre busca que en el vector haya un nematodo. Si no logra encontrar un nematodo la salida es gris, esto quiere decir que no se sabe dónde hay nematodo y dónde fondo porque no se encontró ningún píxel de nematodo. En los casos en donde existe un nematodo en el vector se distingue bien entre nematodo (blanco) y fondo (negro).

La figura 4.13 combina los resultados de la segunda y tercera arquitectura de CNN. Primero se clasifica cada vector horizontal y vertical para ver si contiene nematodo. La red debe tener al menos 95 % de certeza que el vector es nematodo para que éste sea considerado. Si el vector se clasifica como fondo simplemente se pinta de negro en la imagen resultante. Si se clasifica como nematodo se evalúa en la tercera arquitectura para saber exactamente donde se encuentra el nematodo. Finalmente se combinan las respuestas horizontales y verticales. Esta combinación de arquitecturas es la que produce los mejores resultados. Aunque algunas secciones de los objetos que no corresponden a nematodo fueron detectadas como nematodo, este error se puede disminuir por medio de operaciones morfológicas.

La figura 4.14 es el resultado de evaluar la base de datos sin *offset* que contiene tanto nematodos como fondo en la tercera arquitectura. Para los datos de nematodo se logra una segmentación aun mejor que la de la primera arquitectura. La red tiene problemas con los datos de fondo ya que siempre trata de buscar un nematodo.

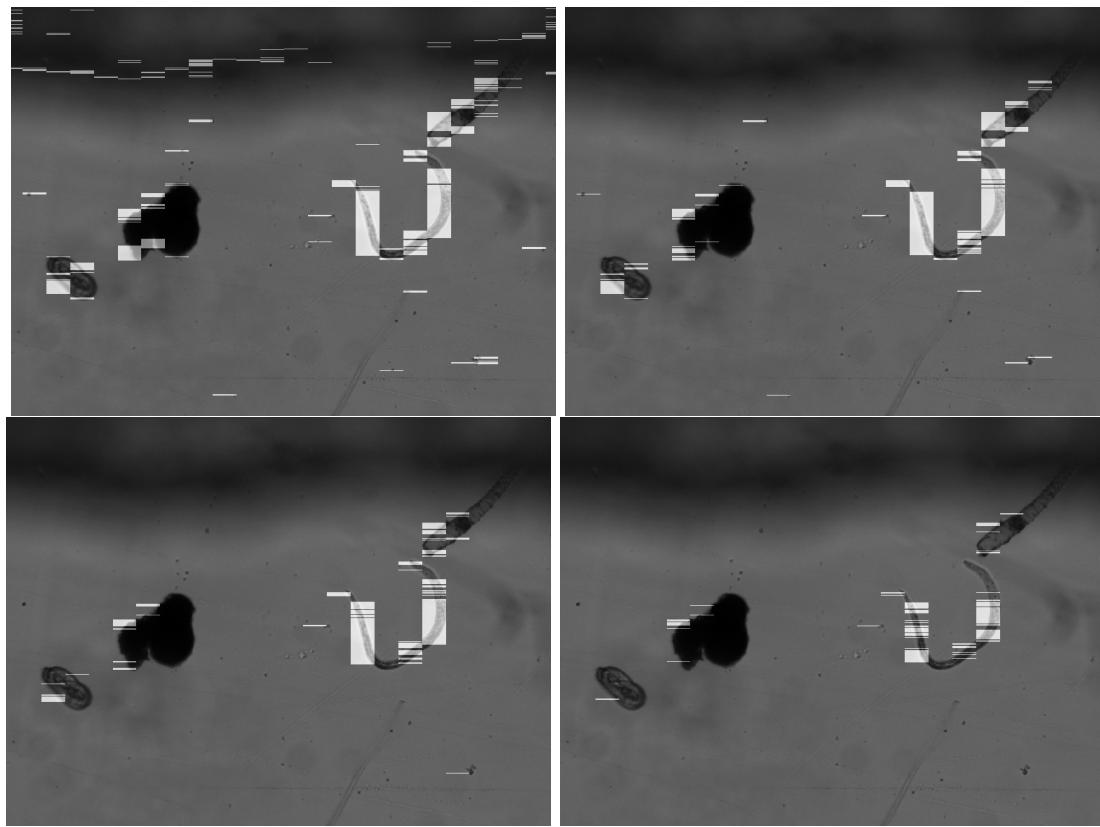


Figura 4.11: Clasificación de vectores de una imagen con la segunda arquitectura de CNN.
En la imagen de la esquina superior izquierda se define el umbral de decisión en 90 %, en la superior derecha 95 %, en la inferior izquierda 97,5 % y en la inferior derecha 98,75 %

La figura 4.15 es el resultado de evaluar la base de datos con *offset* que solo tiene datos de nematodo en la primera arquitectura. Logra identificar la posición del nematodo aún con desplazamiento.

La determinación para dividir el problema en dos etapas (segunda y tercera arquitectura) se tomó debido a los resultados obtenidos. Se diseñaron varias arquitecturas que no tuvieron resultados relevantes, debido principalmente a dos problemas:

1. Cuando se le agrega *offset* a los datos de nematodo se aumenta la variabilidad de la entrada. Cada neurona debe aprender por separado. Cuando se le pasa a la red los datos centrados las neuronas centrales están aprendiendo la posición, las de la izquierda nunca ven un nematodo y las de la derecha aprenden longitud. En general, con datos centrados el problema es mucho más sencillo.
2. Existe un desbalanceo de los datos de entrenamiento. Los datos de nematodo tienen al menos la mitad de fondo y los de fondo son completamente fondo. Por este motivo, a nivel de píxel, se está entrenando en realidad con 75 % de datos de fondo y 25 % de nematodo.

Debido a estos problemas, la salida que disminuye el error de la red puede ser un valor

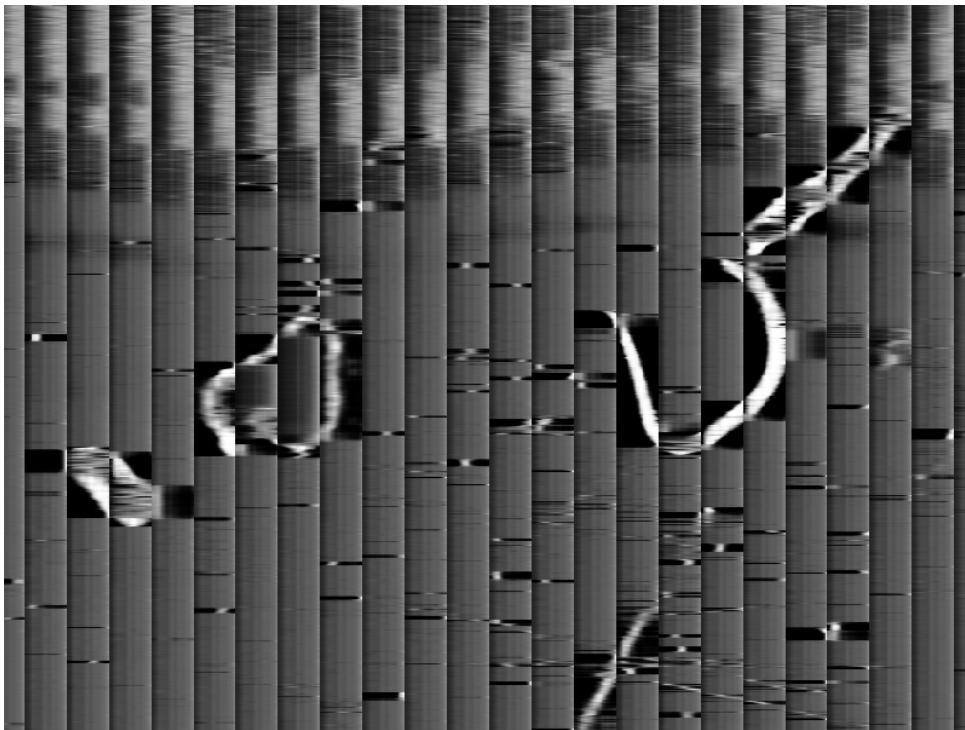


Figura 4.12: Resultado de evaluar una imagen en la tercera arquitectura de CNN

arbitrario de gris para todos los píxeles o algún tipo de distribución guassiana. Al dividir la solución en dos etapas se solucionan ambos problemas. No existe desbalanceo pues la segunda arquitectura de red se entrena con 50 % de cada dato a nivel de vector y la tercera con 50 % a nivel de píxel. Ademas, se simplifica el problema de ubicar el nematodo con offset porque la tercera arquitectura de red no debe preocuparse por clasificar.

4.2.3. Exactitud y error

Para asignarle métricas de exactitud a cada red se utiliza la distancia euclíadiana entre vectores. Para que esta medida sea estándar entre los distintos tamaños de vector e indique un porcentaje se regulariza respecto a la mayor distancia en el espacio. Como los valores del vector solo pueden tomar valores entre 0 y 1 el espacio esta acotado, y la distancia máxima esta determinada por la diagonal (distancia entre el vector de ceros y el vector de unos). La figura 4.16 muestra esta medida de error para la primera y tercera arquitectura de CNN. El error de entrenamiento se refiere al error con el subconjunto de datos que se apartó para entrenamiento, (80 %) aunque la red no se haya entrenado con estos datos.

Medir la exactitud del clasificador es más sencillo pues ya existen muchas métricas para realizar esta tarea. La segunda arquitectura logra un 94 % de exactitud utilizando el error cuadrático medio como métrica.

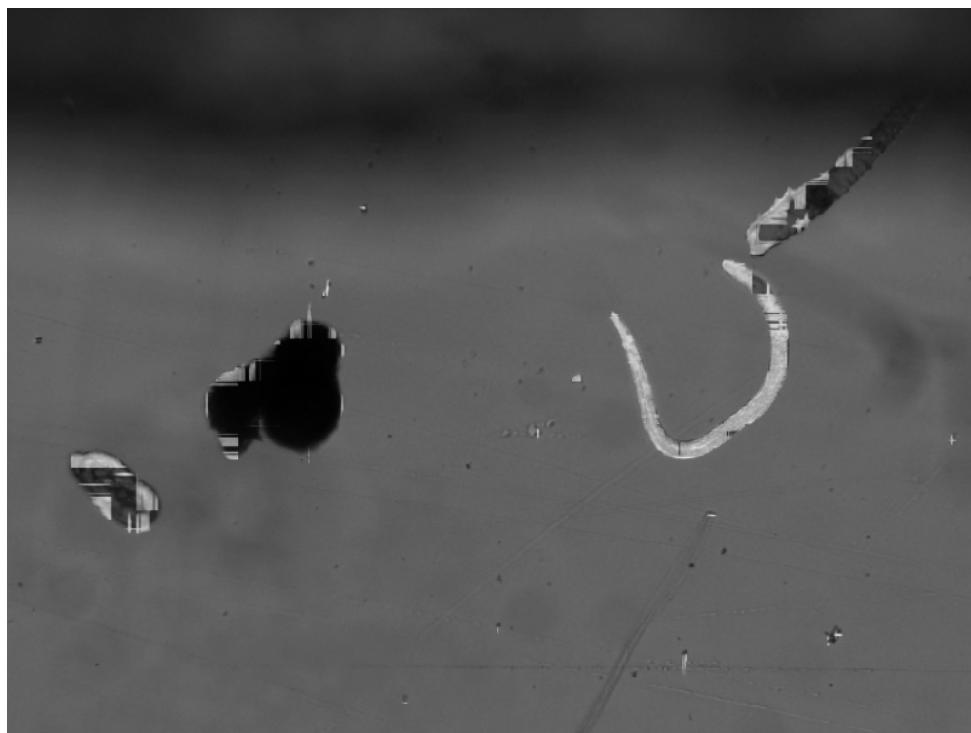


Figura 4.13: Resultado combinado de la segunda y tercera arquitectura de CNN, con un umbral de decisión para la clasificación del 95 %

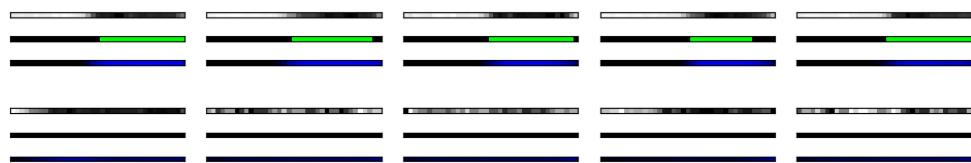


Figura 4.14: Evaluación de datos sin *offset* en la tercera arquitectura de CNN

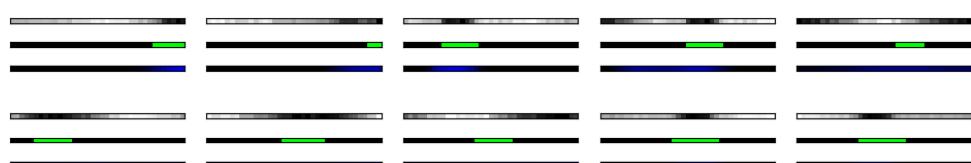


Figura 4.15: Evaluación de datos con *offset* en la tercera arquitectura de CNN

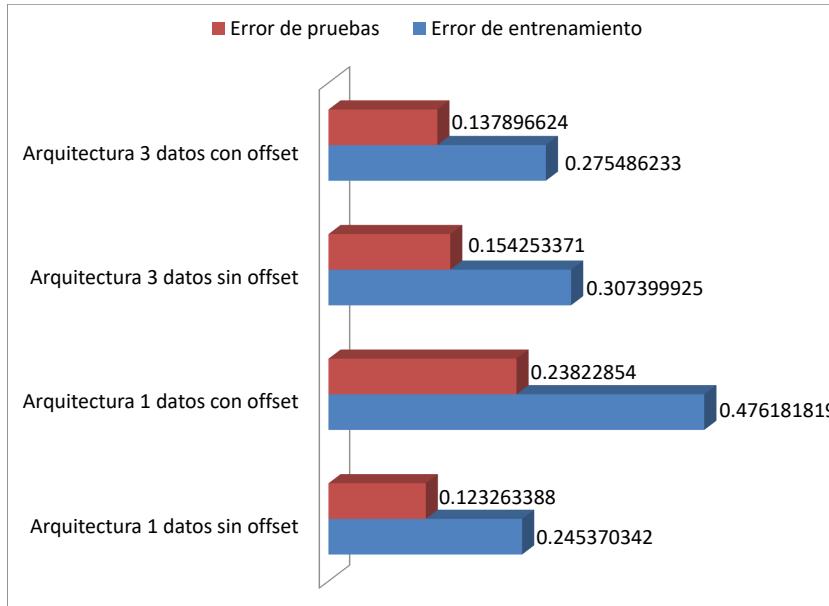


Figura 4.16: Error de la primera y tercera arquitectura, utilizando conjuntos de datos con y sin *offset*. Ver tabla 3.6 para la descripción de cada arquitectura.

4.2.4. Aceleración con GPU

La figura 4.17 muestra los tiempos promedios de realizar una época de entrenamiento para cada una de las arquitecturas. La cantidad de épocas requeridas para que la red converja varía para cada arquitectura.

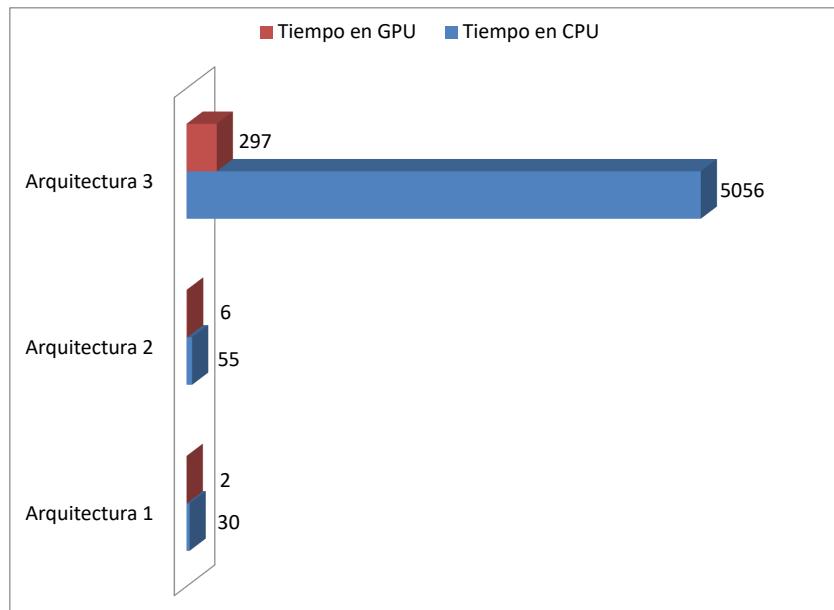


Figura 4.17: Tiempos de ejecución en segundos de una época de entrenamiento en cada arquitectura con y sin GPU. Ejecutado en un CPU Intel Core i7-770 @ 3.60GHz y un GPU NVIDIA GeForce GTX 1060. Ver tabla 3.6 para la descripción de cada arquitectura.

Capítulo 5

Conclusiones

Este trabajo propone un algoritmo nuevo para corregir datos de forma de nematodos, que combina la triangulación de Delaunay junto con una aproximación del TSP. Aunque resuelve un problema muy específico, no se encontró en la literatura un algoritmo para encontrar un polígono válido vermiforme a partir de un grupo de puntos.

De los datos disponibles para este trabajo, un 65 % tienen errores que los hacen inutilizables. La herramienta implementada logra corregir automáticamente el 70 % de los datos con algún tipo de error. Esto duplica la cantidad de datos disponibles para el entrenamiento de las CNN.

Se implementó un algoritmo de ajuste de puntos a la imagen basado en el gradiente obtenido con DoG. El gradiente con DoG dependiente a la escala es mucho menos ruidoso que el obtenido con otros métodos. Aún así el ajuste máximo realizado se tuvo que limitar a pocos píxeles para evitar ajustes erróneos. El interior del nematodo es altamente variable, por lo que es imposible utilizar solo gradientes para encontrar el borde.

Durante el entrenamiento de las CNN, se utilizaron capas de expulsión para evitar el sobreajuste de las CNN. Además, el uso de funciones de activación ReLU en todas las capas, excepto la última, mejoró la velocidad de convergencia de la red sin sacrificar eficacia.

Se utilizó exitosamente aprendizaje profundo para ajustar una forma al borde del nematodo. Se tuvo que reducir la dimensionalidad de los datos de entrenamiento como una estrategia de aumento de datos. Los modelos obtenidos pueden clasificar vectores 1D en nematodo o no-nematodo, de manera estricta (con el borde centrado) o relajada; y determinar en qué dirección hay un borde de nematodo.

Se explotaron las arquitecturas GPU para acelerar el cálculo de las convoluciones en las CNN. Gracias a esto, se logró disminuir el tiempo de entrenamiento a un 6 % del que se tarda solo utilizando CPU. Esto permitió el desarrollo de arquitecturas más complejas dentro del tiempo disponible para el proyecto.

Parte del trabajo futuro es implementar un detector de cola y cabeza que permita corregir

el error donde se etiquetan de manera opuesta. Además, modificar el algoritmo para que permita que el nematodo cruce sobre sí mismo.

En cuanto al proyecto de detección de nematodos el paso siguiente es incorporar los modelos obtenidos con aprendizaje profundo a los algoritmos existentes. Se debe desarrollar un programa que utilice CNN para ajustar las formas obtenidas con ADM [25], para reconocer estructuras vermiformes.

P

Bibliografía

- [1] Mario Araya, Alfonso Vargas, Carlos Castro, and Alexander Cheves. Determinación del tipo de planta adecuado para el monitoreo de poblaciones de fito nematodos en plátano (musa aab). *Uniciencia*, 14(1):5–10, 2016.
- [2] Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.
- [3] Joong-Hwan Baek, Pamela Cosman, Zhaoyang Feng, Jay Silver, and William R. Schafer. Using machine vision to analyze and classify *caenorhabditis elegans* behavioral phenotypes quantitatively. *Journal of neuroscience methods*, 118(1):9–21, 2002.
- [4] Dana H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.
- [5] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.
- [6] Florian Bernard, Frank R. Schmidt, Johan Thunberg, and Daniel Cremers. A combinatorial solution to non-rigid 3d shape-to-image matching. *arXiv preprint arXiv:1611.05241*, 2016.
- [7] Kevin Q Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, 9(5):223–228, 1979.
- [8] Steven D. Buckingham and David B. Sattelle. Fast, automated measurement of nematode swimming (thrashing) without morphometry. *Bmc Neuroscience*, 10(1):84, 2009.
- [9] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.
- [10] Yongmin Cho, Charles L. Zhao, and Hang Lu. Trends in high-throughput and functional neuroimaging in *caenorhabditis elegans*. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 9(3), 2017.
- [11] F Chollet. Keras documentation, 2016.

- [12] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237. Barcelona, Spain, 2011.
- [13] Timothy F. Cootes and Christopher J. Taylor. Active shape models-'smart snakes'. In *BMVC*, volume 92, pages 266–275, 1992.
- [14] Daniel Cremers. Dynamical statistical shape priors for level set-based tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1262–1273, 2006.
- [15] Daniel Cremers, Frank R. Schmidt, and Frank Barthel. Shape priors in variational image segmentation: Convexity, lipschitz continuity and globally optimal solutions. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–6. IEEE, 2008.
- [16] Christopher J. Cronin, Jane E. Mendel, Saleem Mukhtar, Young-Mee Kim, Robert C. Stirbl, Jehoshua Bruck, and Paul W. Sternberg. An automated system for measuring parameters of nematode sinusoidal movement. *BMC genetics*, 6(1):5, 2005.
- [17] Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Introduction*. Springer, 2008.
- [18] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.
- [19] Shireen Y. Elhabian and Ross T. Whitaker. From label maps to generative shape models: A variational bayesian learning approach. In *International Conference on Information Processing in Medical Imaging*, pages 93–105. Springer, 2017.
- [20] Zhaoyang Feng, Christopher J. Cronin, John H. Wittig, Paul W. Sternberg, and William R. Schafer. An imaging system for standardized quantitative analysis of c. elegans behavior. *BMC bioinformatics*, 5(1):115, 2004.
- [21] James Ferguson. Multivariable curve interpolation. *Journal of the ACM (JACM)*, 11(2):221–228, 1964.
- [22] W. Geng, P. Cosman, J. H. Baek, C. C. Berry, and W. R. Schafer. Quantitative classification and natural clustering of caenorhabditis elegans behavioral phenotypes. *Genetics*, 165(3):1117–1126, Nov 2003.
- [23] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.
- [24] Ayala Greenblum, Raphael Sznitman, Pascal Fua, Paulo E. Arratia, and Josue Sznitman. Caenorhabditis elegans segmentation using texture-based models for motility phenotyping. *IEEE transactions on biomedical engineering*, 61(8):2278–2289, 2014.

- [25] Michael Grüner. *Active Dictionary Models: A framework for non-linear shape modeling*. Tesis de maestría, Instituto Tecnológico de Costa Rica, 2015.
- [26] Tomás Guzmán. Prospección, caracterización y evaluación de las relaciones de organismos benéficos para el control de nematodos patógenos en condiciones del trópico. *Proyecto de Investigación.Instituto Tecnológico de Costa Rica*, 2005.
- [27] Rıza Alp Güler, George Trigeorgis, Epameinondas Antonakos, Patrick Snape, Stefanos Zafeiriou, and Iasonas Kokkinos. Densereg: Fully convolutional dense shape regression in-the-wild. *arXiv preprint arXiv:1612.01202*, 2016.
- [28] Katsunori Hoshi and Ryuzo Shingai. Computer-driven automatic identification of locomotion states in *caenorhabditis elegans*. *Journal of neuroscience methods*, 157(2):355–363, 2006.
- [29] Rebecca JM Hurst, Thomas Hopwood, Amanda L. Gallagher, Frederick A. Partridge, Timothy Burgis, David B. Sattelle, and Kathryn J. Else. An antagonist of the retinoid x receptor reduces the viability of *trichuris muris* in vitro. *BMC infectious diseases*, 14(1):520, 2014.
- [30] Pablo Imbach, Megan Beardsley, Claudia Bouroncle, Claudia Medellin, Peter Läderach, Hugo Hidalgo, Eric Alfaro, Jacob Van Etten, Robert Allan, and Debbie Hemming. *Climate change, ecosystems and smallholder agriculture in Central America: an introduction to the special issue*, 2017.
- [31] Ray A Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information processing letters*, 2(1):18–21, 1973.
- [32] Saumya Jetley, Michael Sapienza, Stuart Golodetz, and Philip HS Torr. Straight to shapes: Real-time detection of encoded shapes. *arXiv preprint arXiv:1611.07932*, 2016.
- [33] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.
- [34] Jan J Koenderink. The structure of images. *Biological cybernetics*, 50(5):363–370, 1984.
- [35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [36] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.
- [37] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

- [38] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.
- [39] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [40] Chris Marcellino, Jiri Gut, KC Lim, Rahul Singh, James McKerrow, and Judy Sakanari. Wormassay: a novel computer application for whole-plate motion-based screening of macroscopic parasites. *PLoS neglected tropical diseases*, 6(1):e1494, 2012.
- [41] Juan Esteban Marín. *Ubicación de un nematodo en imágenes digitales utilizando modelos activos de forma*. Tesis de licenciatura, Instituto Tecnológico de Costa Rica, 2009.
- [42] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell Labs Technical Journal*, 36(6):1389–1401, 1957.
- [43] Lei Qu, Fuhui Long, Xiao Liu, Stuart Kim, Eugene Myers, and Hanchuan Peng. Simultaneous recognition and segmentation of cells: application in *C. elegans*. *Bioinformatics*, 27(20):2895–2902, 2011.
- [44] Daniel Ramot, Brandon E. Johnson, Tommie L. Berry Jr, Lucinda Carnell, and Miriam B. Goodman. The parallel worm tracker: a platform for measuring average speed and drug-induced paralysis in nematodes. *PloS one*, 3(5):e2208, 2008.
- [45] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.
- [46] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. *Artificial Neural Networks–ICANN 2010*, pages 92–101, 2010.
- [47] Ryuzo Shingai. Durations and frequencies of free locomotion in wild type and gabaergic mutants of *caenorhabditis elegans*. *Neuroscience research*, 38(1):71–84, 2000.
- [48] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.
- [49] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

- [50] Nicholas Stroustrup, Bryne E. Ulmschneider, Zachary M. Nash, Isaac F. López-Moyado, Javier Apfeld, and Walter Fontana. The *caenorhabditis elegans* lifespan machine. *Nature methods*, 10(7):665–670, 2013.
- [51] Nicholas A. Swierczek, Andrew C. Giles, Catharine H. Rankin, and Rex A. Kerr. High-throughput behavioral analysis in *c. elegans*. *Nature methods*, 8(7):592–598, 2011.
- [52] George D. Tsibidis and Nektarios Tavernarakis. Nemo: a computational tool for analyzing nematode locomotion. *BMC neuroscience*, 8(1):86, 2007.
- [53] Lucas J Van Vliet, Ian T Young, and Piet W Verbeek. Recursive gaussian derivative filters. In *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 1, pages 509–514. IEEE, 1998.
- [54] V. Venkatachalam, N. Ji, X. Wang, C. Clark, J. K. Mitchell, M. Klein, C. J. Tabone, J. Florman, H. Ji, J. Greenwood, A. D. Chisholm, J. Srinivasan, M. Alkema, M. Zhen, and A. D. Samuel. Pan-neuronal imaging in roaming *caenorhabditis elegans*. *Proceedings of the National Academy of Sciences of the United States of America*, 113(8):E1082–8, Feb 23 2016.
- [55] Nhat Vu and BS Manjunath. Shape prior segmentation of multiple objects with graph cuts. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.
- [56] Andrew Witkin. Scale-space filtering: A new approach to multi-scale description. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP’84.*, volume 9, pages 150–153. IEEE, 1984.
- [57] Eviatar Yemini, Tadas Jucikas, Laura J. Grundy, André EX Brown, and William R. Schafer. A database of *caenorhabditis elegans* behavioral phenotypes. *Nature methods*, 10(9):877–879, 2013.
- [58] Mei Zhan, Matthew M. Crane, Eugeni V. Entchev, Antonio Caballero, Diana Andrea Fernandes de Abreu, QueeLim Ch’ng, and Hang Lu. Automated processing of imaging data through multi-tiered classification of biological structures illustrated using *caenorhabditis elegans*. *PLoS computational biology*, 11(4):e1004194, 2015.
- [59] Shaoting Zhang, Yiqiang Zhan, Maneesh Dewan, Junzhou Huang, Dimitris N. Metaxas, and Xiang Sean Zhou. Sparse shape composition: A new framework for shape prior modeling. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1025–1032. IEEE, 2011.

Apéndice A

Keras

Keras es un *framework* en Python que permite implementar CNN de manera facil e intuitiva. Como *backend* utiliza código C++ de las bibliotecas Theano o Tensorflow. Para este proyecto se decidió utilizar el *backend* de Theano porque se encuentra más optimizado para trabajar con CUDA por medio de cuDNN (primitivas aceleradas con GPU para CNN).

A.1. Ambiente de desarrollo

Todas las bibliotecas necesarias se instalan por medio del administrador de paquetes de Python, Anaconda (4.3.27)

Lista de paquetes: keras, theano, libgpuarray, numpy, scipy, mkl, nose, sphinx, pydot-ng, matplotlib, opencv y qt.

Además se requiere CUDA 9.0 driver y toolkit; y cuDNN 7.0, disponibles a través del sitio web de nvidia. Es necesario agregar la carpeta “lib” de CUDA (o “lib64”) a la variable de ambiente \$LD_LIBRARY_PATH.

Keras utiliza por defecto Tensor Flow, por este motivo, el archivo de configuración de Keras (keras.json) se debe modificar para utilizar el *backend* y ordenamiento de Theano.

El archivo de configuración de Theano (.theanorc) se muestra en la figura A.1.

A.2. Uso de la biblioteca

El modelo secuencial de Keras consiste en un objeto a que se le pueden apilar las capas de la red neuronal convolucional. Las capas utilizadas en este proyecto son Dense, Dropout, Activation, Flatten, Conv1D y MaxPooling1D.

Es necesario modelar los datos para que sean compatibles con la etapa de entrada de

```

[[global]
floatX=float32
device=cuda0
optimizer=fast_run

[blas]
ldflags = -L/usr/local/lib -lopenblas

[nvcc]
fastmath = True

[dnn]
include_path=/usr/local/cuda-9.0/include/
library_path=/usr/local/cuda-9.0/lib64/

[cuda]
root=/usr/local/cuda-9.0/

```

Figura A.1: Archivo de configuración de Theano

Keras. En el caso unidimensional la entrada es una matriz de 3 dimensiones, la primera es la cantidad de datos, la segunda el tamaño y la tercera siempre es 1 (orden de dimensiones de Theano). Cada dato debe estar entre 0 y 1. El tipo de dato se define en el archivo de configuración y para este proyecto son números de punto flotante de 32b.

Los datos de salida también se deben transformar. La salida se debe representar con matrices o arreglos de clases donde la posición en el arreglo indica la clase y un 1 o 0 si pertenece o no a esa clase.

La única capa que requiere que se especifique la forma de la entrada es la primera. Las otras capas se conectan automáticamente entre sí a menos que los tipos de dato sean incompatibles.

Las funciones de activación disponibles en Keras son: softmax, ELU, SELU (Unidad lineal exponencial escalada [33]), softplus, softsign, ReLU, tanh, sigmoid y lineal. Se recomienda usar ReLU en las capas intermedias para acelerar el entrenamiento de la red [23], y seleccionar la activación de la última capa de acuerdo a la aplicación.

El uso de la capa Dropout previene el sobreajuste de la red [49].

La capa Flatten aplasta los datos (los hace 1D) antes de pasarlos a las capas densas.

La función “model.compile” permite compilar y entrenar el modelo con un conjunto de datos, optimizando una función de pérdida. Tanto la función de pérdida como el optimizador a utilizar son parámetros de la función. Las funciones de pérdida disponibles son: mean_squared_error, mean_absolute_error, mean_absolute_percentage_error, mean_squared_logarithmic_error, squared_hinge, hinge, categorical_hinge, logcosh, categorical_crossentropy, sparse_categorical_crossentropy, binary_crossentropy, kullback_leibler_divergence, poisson y cosine_proximity. Los optimizadores disponibles son: SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax y Nadam.

Por último “model.evaluate” permite evaluar la función de pérdida del modelo para otro conjunto de datos y “model.predict” regresa las salidas evaluadas con el modelo para un conjunto de entradas.