UNIVERSIDAD NEBRIJA

DESARROLLO ALGORÍTMICO PARA LA SEGMENTACIÓN AUTOMÁTICA DE IMÁGENES EN LA DETECCIÓN DE ENFERMEDADES NEURODEGENERATIVAS

UNIVERSIDAD NEBRIJA GRADO EN INGENIERÍA INFORMÁTICA

MEMORIA TRABAJO FIN DE GRADO

MARÍA GONZÁLEZ HERRERO

1/09/2023

UNIVERSIDAD NEBRIJA

DESARROLLO ALGORÍTMICO PARA LA SEGMENTACIÓN AUTOMÁTICA DE IMÁGENES EN LA DETECCIÓN DE ENFERMEDADES NEURODEGENERATIVAS

UNIVERSIDAD NEBRIJA GRADO EN INGENIERÍA INFORMÁTICA

MEMORIA TRABAJO FIN DE GRADO

MARÍA GONZÁLEZ HERRERO

1/09/2023

Clara Matutano Molina (Tutora y directora)

Nieves Cubo Mateo (Cotutora)

Natalia Arias del Castillo (Codirectora)

Dña. María González Herrero autoriza a que el presente trabajo se guarde y custodie en los repositoriosde la Universidad Nebrija y además SI autoriza a su disposición en abierto.

En primer lugar, este TFG está dedicado y es un homenaje a mi abuela Nati, quien fue diagnosticada con Alzheimer hace muchos años. Desde ese diagnóstico, comencé a investigar y a concebir ideas que, en aquel momento, parecían sacadas de una película de ciencia ficción. Sin embargo, al compartir estos pensamientos con mis tutoras, Nieves y Clara, me condujeron al proyecto que estaba desarrollando Natalia en el laboratorio BRAVE. Gracias a su apoyo, a los conocimientos adquiridos y a los recursos brindados, esa idea que parecía inalcanzable tomó forma y se materializó en este trabajo.

No quiero dejar de mencionar y agradecer a mi familia, amigos y pareja, pilares esenciales que me brindaron su apoyo incondicional en los momentos más duros y complejos de este proyecto.

Índice

Glosario de términos

- → **Algoritmo de ajuste de líneas RANSAC:** Acrónimo de Random Sample Consensus. Método que estima los parámetros de un modelo matemático a partir de un conjunto de datos con valores atípicos. (Villasenor, 2023)
- → **Algoritmo de detección de bordes de Canny:** Algoritmo de visión de computador para la detección de bordes. (Kipuna, 2023)
- → **Algoritmo de esqueletización:** Proceso mediante el cual se transforma un objeto en una imagen digital con propiedades topológicas. (Ruiz, Muñoz Pérez, Bernal, & Rubio)
- → **Clasificación basada en SVM:** Método de clasificación-regresión de Máquinas de Vector de Soporte, utilizada ampliamente a problemas de clasificación múltiple y de regresión. (Rodrigo, 2017)
- → **Clustering:** Técnica de aprendizaje no supervisado, en el cual se tiene un algoritmo de agrupamiento de conjuntos de datos con características similares. (KeepCoding, 2022)
- → **Convolucional:** Tipo de red neuronal artificial donde las neuronas pertenecen a campos receptivos de manera similar a la que funciona un cerebro biológico. (IBM, s.f.)
- → **Demencia frontotemporal:** Conjunto de trastornos que dañan gradualmente los lóbulos frontales y temporales del cerebro provocando alteraciones en el pensamiento y la conducta de los pacientes. (Alzheimers, s.f.)
- → **Detector de SIFT:** Algoritmo que busca detectar y describir las características de una imagen. (Santos, Dallos, & Gaona-García, 2020)
- → **Diagrama de Gantt:** Gráfico de barras horizontales muy usado en la gestión de proyectos, usado para ilustrar el orden cronológico de las tareas y seguimiento de los logros. (Martins, 2022)
- → **ELA:** (Esclerosis Lateral Amiotrófica) Enfermedad que afecta a las neuronas del cerebro, tronco cerebral y la médula espinal que controlan el movimiento de los músculos voluntarios. Alguno de los casos se debe a un defecto genético, pero la mayoría de las veces se desconoce sus causas. (MedlinePlus)
- → **Imágenes de Brightfield:** Observación microscópica, utilizado para estudiar y visualizar en detalle muestras de tejidos biológicos mediante una luz que ilumina la muestra creando una imagen. (Yoshikura, 2023)
- → **LIME:** (Local Interpretable Model-Agnostic Explanations) Diagrama de fuerzas que muestra la intensidad mediante un gradiente, haciendo que cada una de las entradas de datos empuje a la salida hacia un lado u otro, para ayudar al usuario a visualizar la importancia de cada una de las variables. (Del Ser Lorente, 2019)

→ **Magnetoencefalografía:** Permite medir los campos magnéticos producidos por las corrientes eléctricas del cerebro, con el objetivo de mapear la función cerebral y ayudar a la identificación de ataques epilépticos. ((RSNA) & (ACR), s.f.)

→ **Max-pooling:** Proceso de discretización basado en muestras, con el objetivo de submuestrar una representación de entrada como una imagen, reduciendo su tamaño. (DataScientest, 2021)

→ **Microscopia de fluorescencia:** Microscopio al que se le adapta un accesorio de iluminación llamado “fluorescencia”, para utilizarlo tanto para la óptica convencional como para la observación de pruebas de contraste con fluorescencia. (Residuales, 2020)

→ **ReLU:** Función de activación esencial para diseñar una red neuronal que ayuda a las redes neuronales a formar modelos de aprendizaje profundo. (Data Science Team, 2021)

→ **SHAP:** (Shapley Additive exPlanations) Enfoque utilizado para explicar la salida de un modelo hecho con aprendizaje automático, que conecta la explicación local de la asignación óptica de créditos con la ayuda de los valores de Shapely. Este es un enfoque muy útil en la teoría de juegos. (Data Science Team, 2020)

→ **U-Net:** Red neuronal convolucional centrada en la clasificación de imágenes, donde la entrada en una imagen y la salida es una etiqueta, muy utilizada en casos biomédicos porque localiza el área de anormalidad que se desee. (Zhang, 2019)

→ **Up-samplea:** Proceso para aumentar las dimensiones espaciales de las características de los datos de entrada para construir una salida, del mismo tamaño que la entrada original, realizando una trayectoria expansiva de la arquitectura. (Adesanya, 2021)

Índice de ilustraciones

Índice de códigos

Índice de tablas

Resumen

El proyecto se centra en una problemática relevante en el ámbito de las enfermedades neurodegenerativas: la necesidad de una segmentación automática precisa de imágenes de espinas dendríticas. La segmentación automática es una solución prometedora para avanzar con la comprensión y tratamiento de estas enfermedades.

Se inicia con una revisión detallada de las enfermedades neurodegenerativas, resaltando como una segmentación de imágenes de neuronas eficaz puede ser una herramienta muy valiosa, y a continuación, se presenta un estudio de mercado en el cual se analizan y descartan diversos softwares y algoritmos previamente empleados en investigaciones científicas.

Con un enfoque en la estructura neuronal, el trabajo se adentra en los conceptos básico, explicando las características y funciones de la neurona, la espina dendrítica y la unión que existe entre ambas.

En la sección principal del proyecto se describen los datos de entrada, su preprocesamiento y etiquetado y se selecciona el modelo de U-Net para la tarea, exponiendo la lógica y los códigos detrás del modelo.

Finalmente, tras el entrenamiento y validación, re realiza una evaluación del modelo, cuyos resultados se muestran y se analizan.

1. Introducción

Este capítulo presenta una introducción a las bases conceptuales y justificaciones del proyecto. Se incluyen los antecedentes del trabajo, porque el proyecto surge de la colaboración entre los grupos de investigación de BRAVE y ARIES, para mejorar la segmentación de imágenes neuronales. Seguidamente se aborda la necesidad detectada, resaltando la importancia de la detección temprana de enfermedades neurodegenerativas y los desafíos actuales que existen en este campo.

A continuación, se revisa la información previa relativa al aprendizaje profundo en el análisis de imágenes médicas y la relevancia de la segmentación automática de imágenes.

Posteriormente, se justifica la solución propuesta y los objetivos generales del proyecto, haciendo resaltar la aplicación del algoritmo **U-Net** y, por último, se describen los requisitos técnicos y los asociados a la implementación de este algoritmo y el manejo de imágenes biomédicas neuronales.

1.1. Antecedentes

Este trabajo, parte de una colaboración entre los grupos de investigación BRAVE (Brain and Behaviour) y ARIES (Artificial Intelligence and Emergent Systems) de la Universidad Antonio de Nebrija, el cual busca mejorar la detección de la degeneración neuronal. Para ello se trabajará en la segmentación de imágenes de las espinas dendríticas de neuronas de ratón, a través de **imágenes de Brightfield**, mediante diferentes técnicas computacionales.

1.2. Planteamiento del problema

La necesidad de poder detectar la degeneración neuronal de forma temprana y con ello mejorar la calidad de vida de los pacientes, también se encuentra un apartado de información previa, detallando los conocimientos básicos del aprendizaje profundo y de las pruebas necesarias. Y por último se ha realizado un justificante de su resolución, en el que se detalla lo que se estima que puede ser o llegar a ser el tamaño del mercado, empresas actuales existentes que tengan proyectos publicados semejantes a este proyecto y algunos riesgos que hay que tener en cuenta.

1.2.1. Necesidad detectada

La detección temprana de la degeneración neuronal en enfermedades neurodegenerativas como la **ELA** y la **demencia frontotemporal** es crucial para permitir un diagnóstico y tratamiento temprano, así como para mejorar la calidad de vida de los pacientes. Sin embargo, la detección temprana de estas enfermedades a menudo se ve dificultada por la complejidad de las imágenes neurológicas y la falta de herramientas automatizadas adecuadas para analizarlas.

Una solución potencial para este problema es el desarrollo de un algoritmo de segmentación automática de imágenes de neuronas. Superando la incapacidad de los algoritmos actuales, este podría detectar signos precoces de degeneración neuronal en las imágenes de resonancia magnética del cerebro. Las herramientas disponibles en la actualidad enfrentan desafíos particulares al trabajar con imágenes tridimensionales de neuronas.

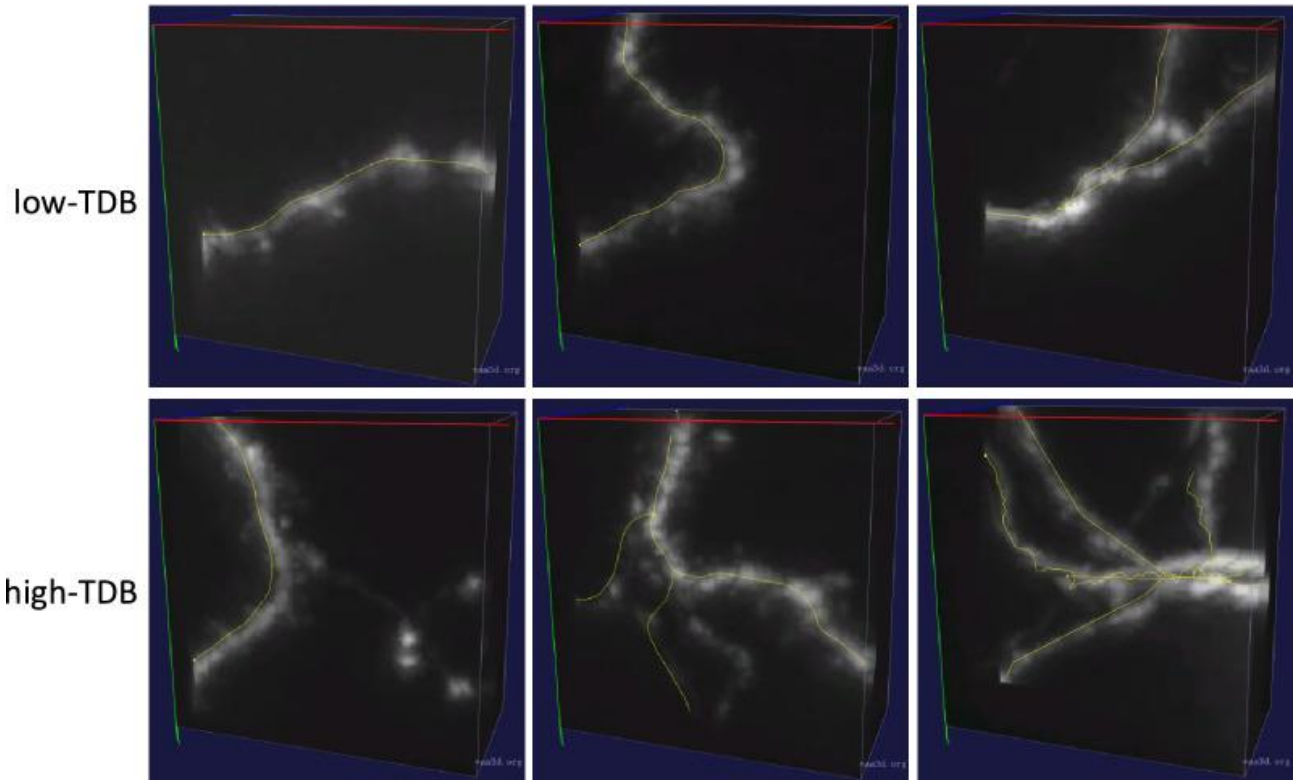


Ilustración 1 - Comparación de imágenes de baja dificultad de rastreo y de alta dificultad de rastreo. Fuente: (Yang, Huang, Wu, & Yang, 2021)

En imágenes 3D que abarcan múltiples neuronas de un cerebro de ratón completo, ciertos segmentos presentan estructuras morfológicas simples, señales robustas y mínimos ruidos. Estos segmentos (ilustración x), en los que numerosos algoritmos pueden rastrear morfologías con considerable precisión, se conocen como bloques de baja dificultad de rastreo (TDB bajos).

Sin embargo, hay segmentos más desafiantes que exhiben estructuras morfológicas complejas, como bifurcaciones y cruces, junto con señales débiles o ruido predominante. En estos, la mayoría de los algoritmos tienden a rastrear morfologías de forma inexacta o incompleta, y se les denomina bloques de alta dificultad de rastreo (TDB alto). Este algoritmo utilizaría técnicas de aprendizaje automático para analizar las imágenes y detectar cambios en la estructura y morfología de las neuronas, que podrían ser indicativos de la enfermedad. (Yang, Huang, Wu, & Yang, 2021)

Además, este algoritmo podría ser entrenado con un gran conjunto de datos de imágenes de pacientes con y sin enfermedad, lo que permitiría una detección precisa y fiable de la degeneración neuronal. También podría ser empleado en combinación con otros marcadores clínicos y biomarcadores para mejorar aún más la precisión del diagnóstico.

1.2.2. Información previa

El aprendizaje profundo se ha convertido en un tema de investigación activo en el campo del análisis de imágenes médicas. Particularmente para la segmentación automática de imágenes, el rendimiento de la segmentación ha hecho grandes avances en este campo.

Esto se debe a que las pruebas necesarias para visualizar imágenes en el cerebro en muchas ocasiones utilizan una tecnología bastante compleja, como la imagen por resonancia magnética funcional (fMRI), en la cual se generan imágenes del interior del cerebro mediante campos magnéticos fuertes, (Mskcc, 2020), o como las **Magnetoencefalografía** (MEG), prueba médica no invasiva que mide los campos magnéticos producidos por las corrientes eléctricas del cerebro, ((RSNA) & (ACR), s.f.). Ambas pruebas requieren de equipos muy sofisticados y complejos. En cambio, otras pruebas son necesarias una preparación previa, como la administración de radioisótopos o sustancias de contraste (PET y SPECT).

También en muchas pruebas influyen factores externos como el ruido, la luz, la ansiedad del paciente o la medicación bajo la que se encuentra sometido, lo que puede dificultar la interpretación de los resultados. Estas pruebas generan en su mayoría grandes cantidades de imágenes médicas y por eso la tarea clave es la segmentación de mismas, que puede realizar la localización, el análisis (cualitativo y cuantitativo) de lesiones y analizar la eficacia del tratamiento.

El método tradicional de segmentación manual requiere mucho tiempo y depende de la experiencia del médico, lo que puede llevar a un resultado u a otro. Es por lo que la aplicación de la tecnología moderna de segmentación de imágenes semiautomáticas y automáticas es tan importante.

Además, se entiende como segmentación al procedimiento típico de la visión artificial, donde se procede a la separación de la imagen en diferentes regiones de interés. Estas regiones se delimitan de su entorno y de otras regiones porque presentan características que los distinguen de ellos. Estos rasgos pueden ser su nivel de grises, su textura o posición, o una combinación de ambos, aplicándolo al problema que se intenta resolver, separando los píxeles que pertenecen al área que se va a analizar de los que no.

1.2.3. Desarrollo de la solución

En este apartado se examina el mercado, la competencia y los riesgos asociados con la segmentación automática de imágenes de neuronas y de dendritas. Se analiza el tamaño del mercado, estimado en 30 mil millones de dólares para 2024, y su demanda potencial. También cabe destacar competidores clave, como Bitplane Imaris, Fiji/Image y Neuropoly, y su impacto en el sector. Finalmente, se exponen los riesgos asociados, como la precisión del algoritmo, la calidad de la imagen y la dependencia del algoritmo, proponiendo estrategias de mitigación.

1) Tamaño del mercado

Actualmente es difícil estimar con precisión el mercado para estos algoritmos, pero se puede hacer una estimación basada en el tamaño de los ya conocidos, como el de la neurociencia, microscopia y la tecnología de imágenes.

Según un informe publicado en Zion Market Research se calcula que unos 30 mil millones de dólares se esperan en torno al año 2024 en el campo de la neurociencia, haciendo que este mercado esté en pleno crecimiento hoy en día y que tenga un papel importante en el futuro. (Zion Market Research)

Además, la segmentación automática de imágenes de neuronas y dendritas es un ara de investigación en la neurociencia y la imagenología médica, que sugiere que hay un interés creciente y una demanda potencial para este tipo de nuevas soluciones.

2) Competencia actual

Existen algunos competidores en el mercado que ofrecen soluciones similares a la segmentación automática de imágenes de neuronas y dendritas, tanto a nivel industrial como a nivel académico. Por ejemplo:

- **Bitplane Imaris:** Plataforma que analiza imágenes en 3D y 4D, entre sus herramientas incluye la segmentación de imágenes de células y neuronas. Aunque para reconocer dendritas no sirve. (Oxford Instruments, s.f.)

Limitaciones frente a U-Net: No es adecuado para reconocer espinas dendríticas, además al ser una plataforma comercial, puede que no sea tan flexible como un modelo entrenado específicamente para un conjunto de datos en particular.

- **Fiji/Image:** Software de procesamiento de imágenes gratis y de código abierto, que incluye herramientas para preprocesamiento de etiquetado de la segmentación automática y manual de imágenes de estructuras neuronales. (ImageJ, 2014)
  Limitaciones frente a U-Net: Al ser un software de procesamiento de imágenes más general, no se optimiza bien a tareas tan específicas. También mientras Fiji permite la implementación y uso de algoritmos de aprendizaje automático, U-Net es una arquitectura diseñada específicamente para la segmentación de imágenes pudiendo superar en algunas tareas a los métodos estándares de Fiji.
- **Neuropoly:** Software para la segmentación automática de imágenes que sirve para analizar estructuras neuronales complejas, incluyendo herramientas de segmentación automática y manual. (NeuroPoly, s.f.)
  Limitaciones frente a U-Net: Al ser una solución de software, su adaptabilidad y rendimiento pueden estar limitados en comparación con un modelo personalizado.

3) Riesgos

Como en todo proyecto existe una serie de riesgos que, dependiendo de su prioridad, se asumen o es necesario reevaluar el alcance de este para enfocarlo de otra manera, a continuación, se evaluan los riesgos más relevantes que he podido encontrar y que serán clave para continuar el proyecto:

- **Riesgo de precisión:** todos los algoritmos de segmentación automática, incluido el que se utiliza en este proyecto (U-Net), puede tener limitaciones en la precisión de la segmentación. Teniendo graves consecuencias en la interpretación de los datos, como un diagnóstico erróneo, lo que puede provocar tratamientos inapropiados debido a esa interpretación errónea. Otros problemas se pueden llevar a la pérdida de información valiosa, basada en la omisión de estructuras biológicas claves.
  Por eso se utilizan diferentes conjuntos de datos y se hace una evaluación final de precisión y sensibilidad con diferentes sets de datos.
- **Riesgos de calidad de imagen:** es un factor clave a la hora de tratar los datos, ya que una mala calidad o ruido en la imagen puede afectar notoriamente a los resultados que se obtengan.
  Por este motivo, todas las imágenes pasan por un preprocesamiento para ajustarlas a la calidad y condiciones óptimas necesarias.

- **Riesgos de dependencia del algoritmo:** la segmentación automática de las imágenes puede crear una dependencia del algoritmo, es decir, al tener el algoritmo como la única fuente de información para obtener los resultados, puede que con un pequeño cambio en el propio algoritmo cambie las condiciones de imagen afectando a los resultados del análisis. Por este motivo, se realiza un enfoque modular y escalable, que pueda permitir la incorporación de nuevas funcionalidades y la adaptación a diferentes situaciones y requisitos.

1.3. Objetivos generales

El principal objetivo es aplicar el algoritmo U-Net a la segmentación automática de imágenes biomédicas neuronales para detectar la degeneración neuronal, esta técnica permite la detección precisa y fiable de cambios en la estructura y morfología de las neuronas, que pueden indicar la presencia de enfermedades neurodegenerativas. El uso de U-Net en este caso permitirá obtener resultados de forma eficiente y mejorar el análisis y diagnóstico de la degeneración neuronal en comparación con las herramientas existentes en la actualidad.

El enfoque técnico y riguroso utilizado por el algoritmo garantiza una evaluación objetiva y precisa de las imágenes biomédicas, proporcionado así una valiosa herramienta para la práctica médica y la investigación de las enfermedades neurodegenerativas.

1.4. Objetivos específicos

- Mejorar la precisión del diagnóstico de enfermedades neurodegenerativas mediante la segmentación automática de imágenes de neuronas.
- Reducir la carga de trabajo de los radiólogos y neuropatólogos al automatizar el proceso de segmentación de imágenes, se reduce su carga de trabajo, haciendo que sea posible centrarse en la interpretación de los resultados.
- Mejorar la eficiencia del diagnóstico al automatizar el proceso de segmentación, se podría aumentar la velocidad y eficiencia del diagnóstico de enfermedades
- Mejorar la precisión del tratamiento al detectar los cambios precoces en la estructura de las neuronas, se podrían aplicar tratamientos tempranos.
- Generar nuevos datos para el estudio de enfermedades neurodegenerativas gracias al algoritmo U-Net se pueden generar datos precisos y confiables para el estudio de enfermedades neurodegenerativas, lo que ayudará a mejorar el conocimiento sobre estas enfermedades y a desarrollar nuevos tratamientos.

1.5. Especificaciones técnicas

- Sistema operativo: Windows 11 Pro
- Procesador: Intel i5-13500 13th Generación
- Memoria: 32GB de RAM
- Espacio de almacenamiento: al menos 843GB de espacio libre en el disco duro
- Python: 3.9.13

2. Marco teórico

2.1. Algoritmos para la segmentación de imágenes

Se enfoca en la comparación de diversos algoritmos utilizados en proyectos anteriores para la segmentación de neuronas o espinas dendríticas en imágenes.

La segmentación es una tarea clave para la detección de neurodegeneración, y una variedad de técnicas y algoritmos están disponibles para esta tarea.

A) Proyectos relacionados con espinas dendríticas

- **Automated dendritic spine detection using convolutional neural networks on maximum intensity projected microscopic volumes:**
  Describe un método para detección y segmentación automática de espinas dendríticas a partir de imágenes de **microscopia de fluorescencia**.
  Utiliza varios métodos, el primero llamado ***Algoritmo de detección de bordes de Canny***, se centra en la identificación de bordes de las espinas dendríticas y el ***Algoritmo de ajuste de líneas RANSAC***, se utiliza para ajustar una línea a los bordes que vaya detectando. Después utiliza otros dos algoritmos más, uno de ellos para obtener la forma real de la espina llamado ***Algoritmo de esqueletización*** y el último un algoritmo basado en el umbral para separar el fondo de la imagen del resto. (Xiao, y otros, 2018)

- **Automatic dendritic spine detection using multiscale dot enhancement filters and sift features:**
  El enfoque que se propone en este artículo es el de utilizar dos algoritmos claves para la detección automática de espinas dendríticas a partir de imágenes de microscopia de fluorescencia.
  Utiliza una combinación de técnicas de mejora de puntos y de extracción de características, siendo los filtros de mejora de puntos en múltiples escalas y el **detector de SIFT**, que sirve para identificar las espinas dendríticas. (Rada, Erdil, OzgurArgunsah, Unay, & Cetin, 2014)

- **Tracking-assisted Detection of Dendritic Spines in Time-Lapse Microscopic Images:**
  Presenta dos partes principales, la detección de espinas dendríticas y el seguimiento de estas, a través de imágenes de microscopia en serie temporal.
  La primera parte, el método utiliza un algoritmo de detección de bordes en Sobel para detectar los bordes resultado los y creando una imagen binaria que representa la ubicación de las espinas dendríticas.

La segunda parte utiliza un algoritmo de seguimiento de puntos de interés basado en el algoritmo Lucas-Kanade para rastrear las espinas dendríticas a través del tiempo. Este último encuentra puntos de interés en la imagen de referencia y después los rastrea en las siguientes imágenes utilizando una técnica llamada de seguimiento de flujo óptico. (Rada, y otros, 2018)

B) Proyectos relacionados con neuronas

- **Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images:** Los autores proponen un método para la segmentación de membranas neuronales en imágenes de microscopia electrónica basado en *Redes Neuronales Convolucionales Profundas,* para extraer características relevantes de la imagen y después mediante el algoritmo de ***Clasificación basado en SVM****,* se clasifican los pixeles de la imagen en membranas neuronales o no neuronales. (Cireşan, Giusti, Gambardella, & Schmidhuber, 2012)

- **Deep Learning Segmentation of Optical Microscopy Images Improves 3D Neuron Reconstruction:**
  En este artículo se propone un método para mejorar la reconstrucción tridimensional de neuronas a partir de imágenes de microscopía óptica utilizando técnicas de *Aprendizaje Profundo*, para la segmentación automática de neuronas en imágenes bidimensionales.
  El método utilizado es el ya comentado antes de aprendizaje profundo y después el otro algoritmo basado en el ***Clustering*** para la segmentación de las células neuronales. (Rongjian Li*, 2017)

2.2. Conceptos básicos

Conceptos básicos de una neurona

La arquitectura básica del cerebro se construye a través de un proceso continuo que comienza antes del nacimiento y continúa hasta la edad adulta. La primera etapa es la formación de conexiones neuronales y habilidades simples, seguida de circuitos y de las habilidades más complejas. Esta arquitectura se basa en miles de millones de conexiones entre neuronas individuales en diferentes áreas del cerebro. (Center Director, 2015)

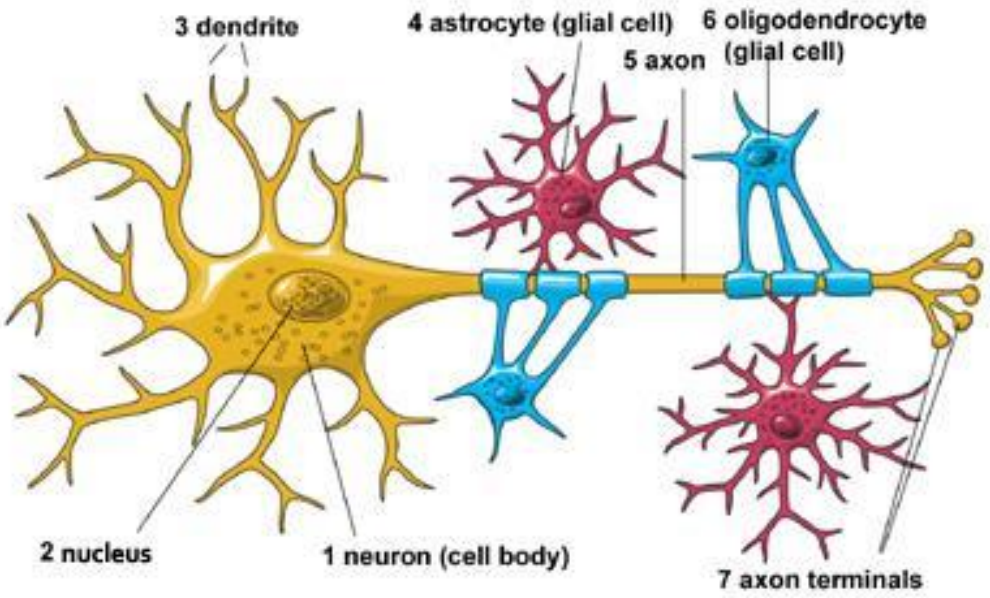


Ilustración 2 - Estructura de una neurona con sus partes. Fuente: (Brain basics: The life and death of a neuron, 2023)

Una neurona tiene tres partes básicas: un cuerpo celular y dos ramas llamadas axones y dendritas (Ilustración 2).

Dentro del cuerpo celular hay un núcleo que controla las actividades de la célula y contiene el material genético de la célula. Las dendritas tienen una forma similar a las ramas de un árbol y son responsables de recibir y transmitir mensajes.

Las neuronas también se comunican entre sí mediante el envío de sustancias químicas, llamadas neurotransmisores, a través de un pequeño espacio llamado sinapsis entre los axones y las dendritas de las neuronas vecinas cercanas. (Brain basics: The life and death of a neuron, 2023)

Conceptos básicos de una espina dendrítica

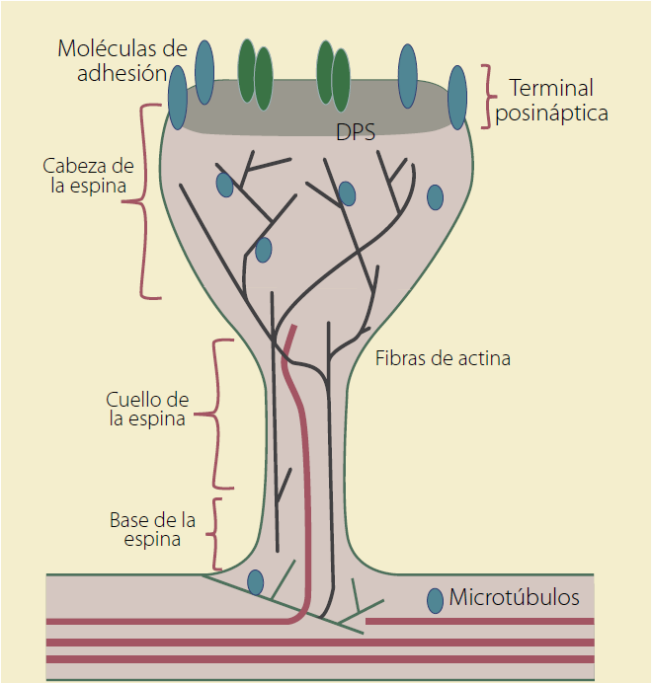


Ilustración 3 - Representación esquemática de una espina en forma de hongo en donde se muestran los elementos más representativos de su citoesqueleto. Fuente: (Valencia Segura, Colín Barenque, & Fortoul van der Goes, 2018)

Las dendritas (Ilustración 3) son un grupo de proyecciones muy ramificadas y cónicas que se extienden desde el cuerpo celular (soma) de una neurona y que a través de las cuales se conducen los impulsos nerviosos hacia ella en el sistema nervioso central y en el periférico. (Torres, 2023)

Actúan como principales puntos de conexión entre las neuronas. Es en estas espinas donde las terminaciones axonales de una neurona establecen contacto con la dendrita de otra neurona. Su función principal es la de transmitir información de una neurona a otra en el sistema nervioso, estas estructuras y todas las demás células nerviosas son importantes para que la información se transmita al cerebro, la médula espinal y en los ganglios espinales. (Junquera, s.f.)

La salud y morfología de las espinas dendríticas son críticas para la función cognitiva y la memoria, ya que si se producen cambios en la forma, tamaño o densidad de las espinas puede indicar o contribuir a trastornos neurológicos y enfermedades neurodegenerativas. En estas condiciones, las espinas dendríticas puedes sufrir alteraciones a un ritmo muy elevado, dando lugar a la pérdida de conexiones sinápticas. Estas conexiones perdidas, son una interrupción de la comunicación neuronal, que es fundamental para la función y el procesamiento adecuados del cerebro.

Por lo tanto, entender y ser capaz de identificar con precisión las morfologías de las espinas dendríticas y sus cambios pueden ser clave a la hora del diagnostico temprano y la intervención en enfermedades neurodegenerativas.

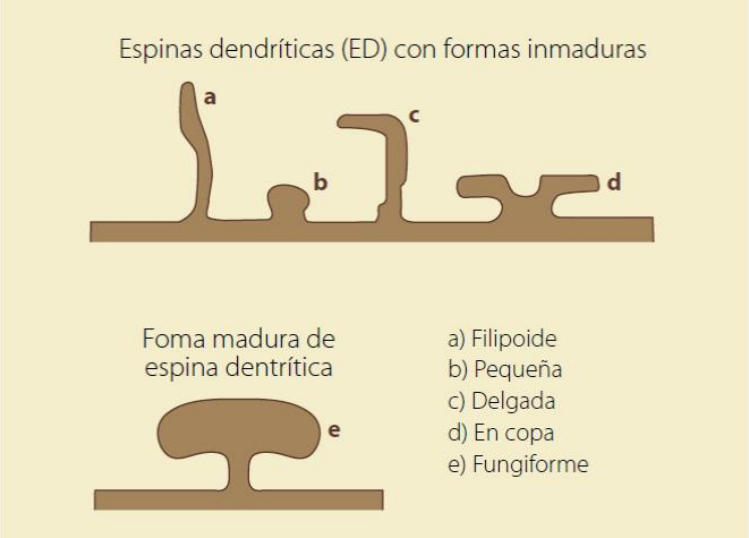


Ilustración 4 - Representación esquemática de las diversas formas de espinas dendríticas: filipodia, delgadas, cortas, en forma de copa y hongo. Fuente: (Valencia Segura, Colín Barenque, & Fortoul van der Goes, 2018)

Además, las espinas dendríticas puedes estar presentas en varios tamaños, formas y algunas alteraciones de estas, por lo que a la hora de identificarlas puede suponer un problema. Las formas más comunes: delgadas`, cortas, en forma de hongo o filopodia (Ilustración 4).

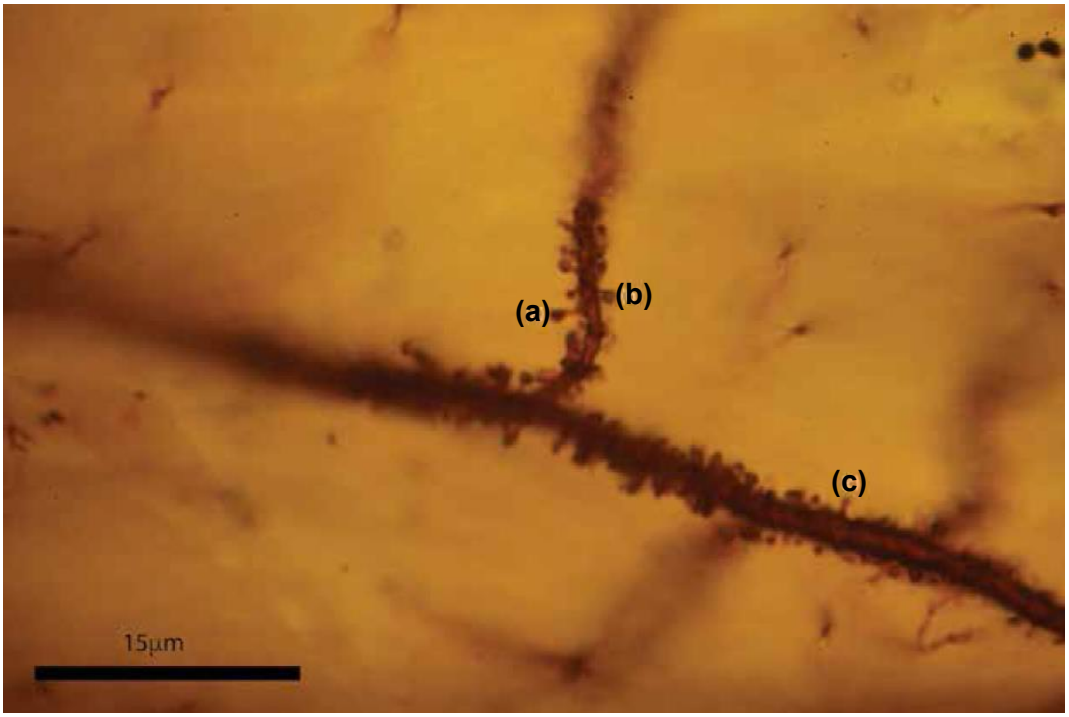


Ilustración 5 - Micrografía en campo claro de espinas dendríticas de una neurona piramidal de hipocampo CA1 procesada con técnica de Golgi, en donde se muestran diferentes tipos de espinas. Fuente: (Valencia Segura, Colín Barenque, & Fortoul van der Goes, 2018)

A través de la micrografía que se pueden observar, en la Ilustración 5 las formas comentadas anteriormente en el esquema de la ilustración 3. La imagen que se ve a continuación se trata de una neurona piramidal de hipocampo CA1, procesada con una técnica de Golgi (revela la morfología neuronal completa en tres dimensiones), en donde se muestran diferentes tipos de espinas: **a)** en forma de hongo, **b)** de cuello corto y **c)** delgada. (Raida, 2018)

2.3. Impacto de los resultados

En este apartado se exponen las oportunidades de un proyecto que pretende revolucionar el diagnóstico y tratamiento de enfermedades neurodegenerativas, explorando la generación de datos esenciales para futuras investigaciones y las oportunidades que se presentan, incluyendo el impulso a avances en inteligencia artificial y la mejora en las tecnologías de microscopía entre otros.

Oportunidades

El campo de la segmentación automática de imágenes biomédicas está en constante evolución por lo que teniendo en cuenta la posibilidad de analizar de forma rápida y precisa estructuras neuronales complejas a través de algoritmos puede tener un gran impacto en la investigación de la neurociencia, el diagnóstico y tratamiento de algunas enfermedades neurodegenerativas, como, por ejemplo:

- **Avances en inteligencia artificial:** puede ser un desafío interesante para desarrollar nuevos algoritmos de aprendizaje automático y redes neuronales profundas, para impulsar avances en la inteligencia artificial y la visión por computadora.
- **Mejora de las tecnologías de microscopía:** ayudaría a mejorar la calidad y eficiencia de las tecnologías de microscopia, para comprender de manera más clara los análisis de las estructuras neuronales y la capacidad de observación.
- **Nuevas aplicaciones en robótica y automatización:** podría inspirar nuevas aplicaciones en robótica y automatización, permitiendo la creación de sistemas autónomos que puedan limitar la capacidad del cerebro a la hora de procesar y analizar información visual.

3. Proyecto

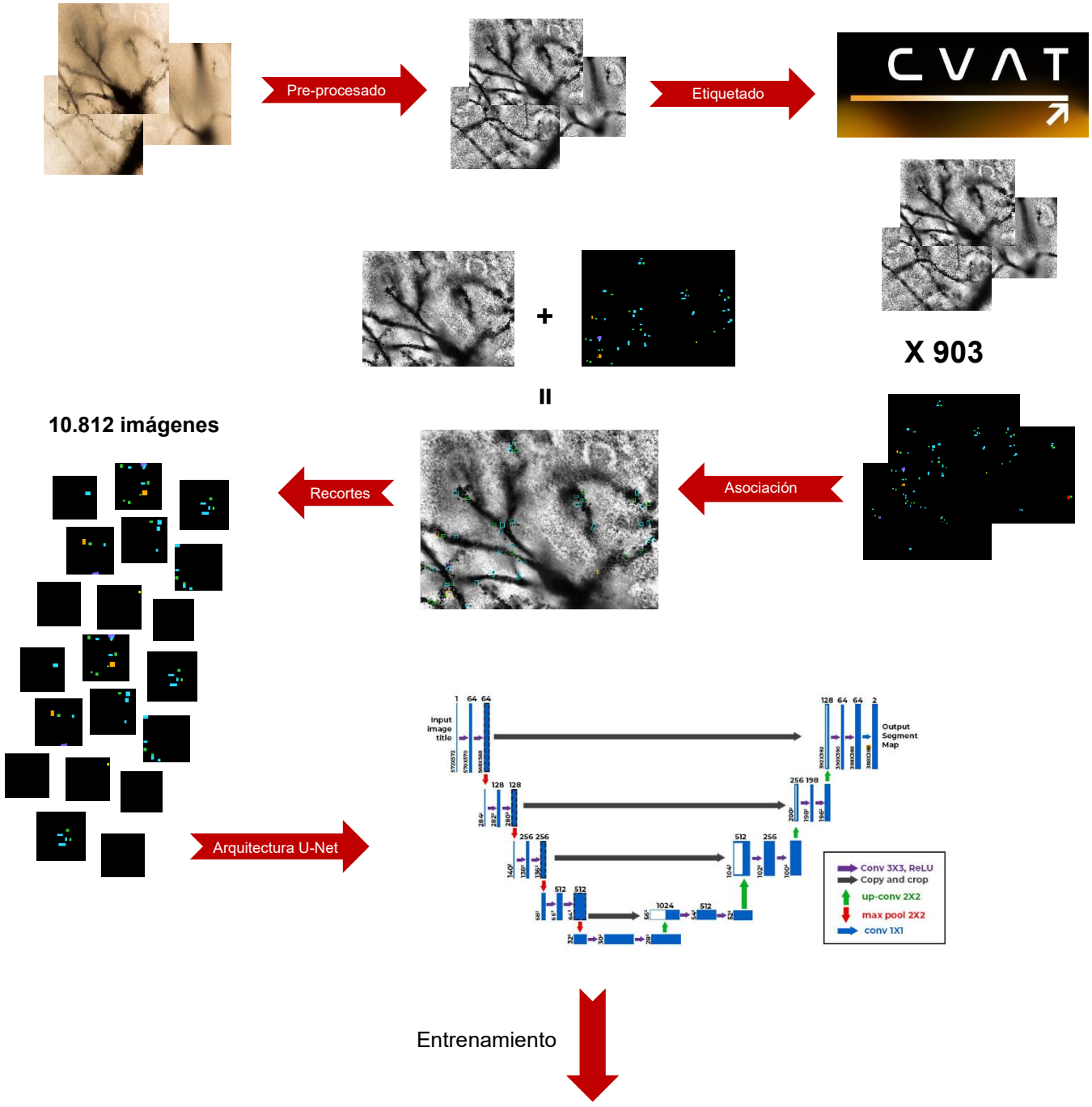
3.1. Planificación temporal

TFG_espinas_dendríticas						
índice	Nombre de tarea / Título	Asignado a	Fecha de inicio planificada	Fecha de finalización planificada	Estado	Tipo
1	**Inicio proyecto**		**28/10/2022**	**14/12/2022**		**proyecto**
1.1	Firma del anteproyecto	María González	28/10/2022	01/11/2022	Terminado	tarea
1.2	Definir un título	María González	09/12/2022	14/12/2022	Terminado	tarea
1.3	Definir índice	María González	03/11/2022	07/11/2022	Terminado	tarea
2	**Punto 1 del índice (Introducción)**		**16/01/2023**	**25/01/2023**		**proyecto**
2.1	Antecedentes	María González	16/01/2023	25/01/2023	Terminado	tarea
2.2	Planteamiento del problema	María González	16/01/2023	25/01/2023	Terminado	tarea
2.3	Objetivos generales	María González	16/01/2023	25/01/2023	Terminado	tarea
2.4	Requisitos técnicos	María González	16/01/2023	25/01/2023	Terminado	tarea
3	**Punto 2 del índice (Marco teórico)**		**15/02/2023**	**22/03/2023**		**proyecto**
3.1	Algoritmos para la segmentación de imágenes	María González	15/02/2023	22/03/2023	Terminado	tarea
3.2	Conceptos básicos	María González	15/02/2023	22/03/2023	Terminado	tarea
3.3	Implantación e impacto de los resultados	María González	15/02/2023	22/03/2023	Terminado	tarea
4	**Punto 3 del índice (Proyecto)**		**28/10/2022**	**30/08/2023**		**proyecto**
4.1	Planificación temporal	María González	28/10/2022	30/08/2023	Terminado	tarea
4.2	Análisis	María González	27/03/2023	18/04/2023	Terminado	tarea
4.3	Diseño	María González	20/04/2023	09/05/2023	Terminado	tarea
4.4	Implementación	María González	18/08/2023	30/08/2023	Terminado	tarea
4.5	Etiquetado de imágenes	María González	18/07/2023	14/08/2023	Terminado	tarea
4.6	Entrenamiento	María González	16/08/2023	29/08/2023	Terminado	tarea
4.7	Validación	María González	16/08/2023	29/08/2023	Terminado	tarea
4.8	Predicción de máscaras	María González	21/08/2023	29/08/2023	Terminado	tarea
5	**Conclusiones**		14/08/2023	31/08/2023	Terminado	**proyecto**
6	**Líneas futuras**		21/08/2023	21/08/2023	Terminado	**proyecto**
7	**Bibliografía**		16/01/2023	30/08/2023	Terminado	**proyecto**
8	**Formato de entrega**		14/08/2023	31/08/2023	Terminado	**proyecto**
9	**TFG**		28/11/2022	01/09/2023	Terminado	**proyecto**

Ilustración 6 - Diagrama de Gantt simplificado. Fuente:((miro.com, s.f.)

Se ha utilizado un **diagrama de Gantt** (Ilustración 6) como herramienta visual, facilitando así la gestión, planificación y seguimiento del proyecto. El diagrama despliega el tiempo en el eje horizontal y las distintas tareas en el eje vertical. Cada actividad se presenta con un color diferente y una barra que indica la duración y finalización de cada una.

3.2. Esquema previo de la visión general




```
Epoch 1/40
676/676 [==============================] - 4190s 6s/step - loss: 0.2573 - accuracy: 0.9741 - iou: 0.0069 - val_loss: 0.0378
- val_accuracy: 0.9942 - val_iou: 1.2358e-04
Epoch 2/40
676/676 [==============================] - 4183s 6s/step - loss: 0.0298 - accuracy: 0.9942 - iou: nan - val_loss: 0.0296 -
val_accuracy: 0.9942 - val_iou: 0.0033
Epoch 3/40
676/676 [==============================] - 4177s 6s/step - loss: 0.0275 - accuracy: 0.9942 - iou: 0.0147 - val_loss: 0.0279
- val_accuracy: 0.9942 - val_iou: 0.0023
Epoch 4/40
676/676 [==============================] - 4176s 6s/step - loss: 0.0266 - accuracy: 0.9942 - iou: 0.0177 - val_loss: 0.0489
- val_accuracy: 0.9933 - val_iou: 0.0899
Epoch 5/40
676/676 [==============================] - 4175s 6s/step - loss: 0.0259 - accuracy: 0.9943 - iou: 0.0225 - val_loss: 0.0323
- val_accuracy: 0.9941 - val_iou: 0.0363
```

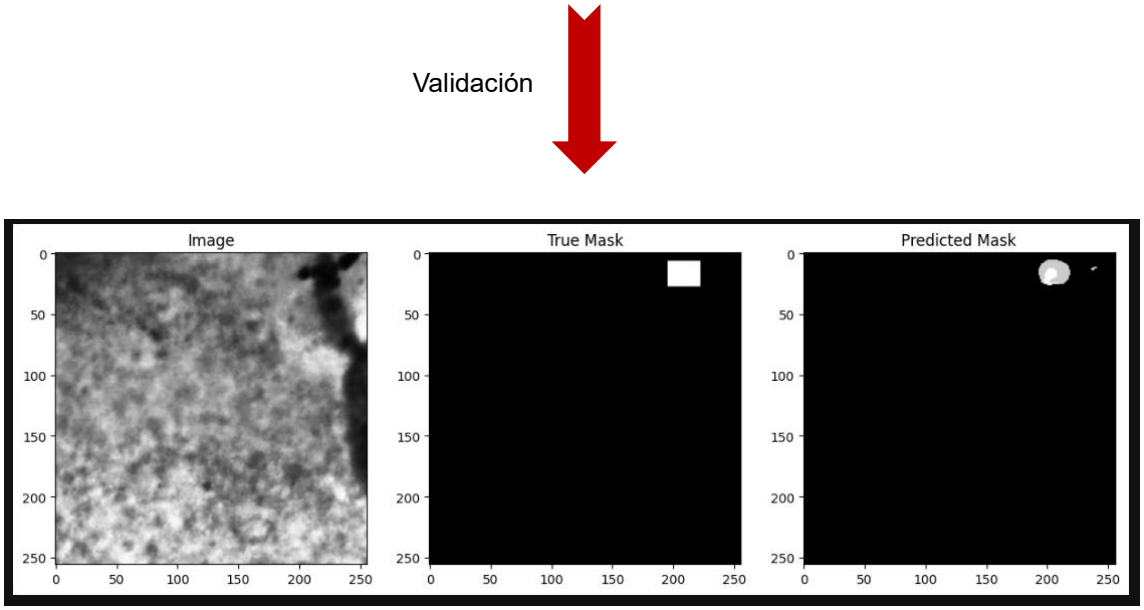


Ilustración 7 - Visión general de la segmentación de imágenes con el algoritmo. Fuente: (Word, María González, 2023)

3.3. Análisis

3.3.1. Algoritmos descartados

En el siguiente apartado se muestra una tabla comparativa que ilustra de una forma más clara los algoritmos utilizados hasta el momento para la segmentación automática de espinas dendríticas y neuronas en otros proyectos, con sus características, beneficios y limitaciones. El objetivo de este análisis es proporcionar un marco de referencia para entender la elección de las metodologías que se pretenden implementar.

Nombre	Ventajas	Desventajas
Detección de bordes de Canny	- Alta precisión de bordes. - Sensibilidad al ruido y puede reducir el ruido de una imagen. - Fácil de implementar.	- No se recomiendan para la segmentación de estructuras que no tiene los bordes claramente definidos. - No es recomendable para estructuras con formas no estándar y con gran variedad de formas y tamaños. - Puede necesitar ajustes manuales para optimizar la precisión de los bordes.
Ajuste de líneas de RANSAC	- Alta precisión para detectar líneas. - Adecuado para segmentar objetos lineales. - Fácil de implementar y de aplicar.	- Para segmentar objetos no lineales no es adecuado (necesita tener líneas muy definidas). - No es adecuado para formas y tamaños variables. - Necesita ajustes manuales para optimizar la precisión de las líneas.

Mejora de puntos	- Mejora la precisión de la segmentación corrigiendo posibles errores en la detección de puntos. - Puede ser útil para segmentar estructuras pequeñas o poco visibles.	-Necesita procesamiento adicional, haciendo que aumente el tiempo de procesamiento. -No puede corregir todos los errores de los puntos, solo en los que haya habido detección de puntos individuales.
Detección de bordes Sobel	- Alta precisión de detección de bordes. - Sensibilidad al ruido y capacidad de reducir el ruido de las imágenes. - Útil para la segmentación de estructuras lineales.	- No es adecuado para detectar bordes que no sean lineales o estructuras que no están claramente definidas - No es adecuado para segmentar estructuras con gran variedad de formas y tamaños. - Necesita ajustes manuales para optimizar la detección de los bordes.
Lucas-Kanade	- Alta precisión en objetos en movimiento. - Útil para segmentar imágenes en vídeo o secuencias de imágenes. - Útil para segmentar estructuras con formas y tamaños similares entre sí.	- No es adecuado para segmentar objetos estáticos o sin movimiento. - No es adecuado para estructuras con formas y tamaños muy diferentes. - Necesita de ajustes manuales para optimizar la precisión del seguimiento de objetos.
Redes Convolucionales Profundas	- Capacidad para aprender características y patrones complejos. - Es adecuado para segmentar estructuras complejas y grandes.	- Necesita gran cantidad de datos de entrenamiento y de tiempo para obtener la precisión requerida. - Puede ser propenso al sobreajuste si no hay suficientes datos de entrada. - Puede necesitar mayor cantidad de recursos computacionales que otros algoritmos.
Clasificación basada en SVM	- Capacidad para clasificar objetos en diferentes categorías. - Adecuado para la segmentación de objetos con características distintivas. - Puede ser útil para la segmentación de estructuras con formas y tamaños similares.	- Necesita un preprocesamiento muy extenso para obtener resultados relevantes. - Pude necesitar ajustes manuales para optimizar la clasificación. - No es adecuado para segmentar objetos con tamaños muy diferentes y complejos.
Aprendizaje Profundo	- Capacidad para aprender características y patrones complejos. - Puede mejorar la precisión de segmentación frente a otros algoritmos. - Puede ser adecuado para la segmentación de estructuras complejas y grandes.	- Requiere de una cantidad significativa de datos de entrenamiento y tiempo para obtener una precisión aceptable. - Puede ser propenso al sobreajuste si los datos son insuficientes.

		- Necesita mayor cantidad de recursos computacionales que otros algoritmos.
Clustering	-Capacidad para agrupar píxeles con características similares. -Puede ser útil para la segmentación de estructuras con formas y estructuras similares. - Útil para la segmentación de objetos con características distintivas.	- Necesita de un preprocesamiento adecuado de la imagen para obtener resultados relevantes. - Puede ser propenso a errores de segmentación si los datos no son suficientes. - No es adecuado para la segmentación de objetos con formas y tamaños muy diferentes o complejos.

Tabla 1 - Tabla comparativa de algoritmos descartados. Fuente: (Word, María González, 2023)

3.3.2. Librerías seleccionadas

A continuación, se exponen las librerías seleccionadas para la programación del algoritmo en Python.

Esta elección se debe a las ventajas que Python ofrece en términos de simplicidad sintáctica, lo que facilita su uso para programas de aprendizaje profundo y redes neuronales. Y se explicará para que se ha utilizado cada librería en el algoritmo.

Librerías de Python	Versión	Utilidad
Cv2	4.7.0	Trabaja con imágenes en Python, incluyendo la carga, visualización, manipulación y filtrado de imágenes. (Zyprian, 2023)
Imageio.v2	2.27.0	Trabaja con los datos de las imágenes como grandes cantidades de datos y formatos como el .tif. (PyPI, s.f.)
Matplotlib.pyplot	3.6.3	Módulo de Matplotlib que propone funciones sencillas para generar gráficos de líneas barras, puntos, histogramas… (DataScientest, 2023)
Numpy	1.24.1	Se especializa en el cálculo numérico y en el análisis de datos de grandes volúmenes. (Alberca, 2022)
os	3.9.13	Módulo que permite acceder a las funcionalidades del Sistema Operativo. (Uniwebsidad, s.f.)
scikit-image	0.21.0	Especializado en el procesamiento de imágenes, incluyendo algoritmos de segmentación, transformación, análisis, morfología… (Tecnología | Uniandes, 2022)

scikit-learn	1.3.0	Proporciona acceso a versiones eficaces de muchos algoritmos comunes. (datascientest, 2022)
TensorFlow	2.11.0	Facilita la creación de modelos de aprendizaje automático. (TensorFlow, s.f.)

Tabla 2 - Librerías utilizadas. Fuente: (Word, María González, 2023)

3.4. Diseño

3.4.1. Conjunto de datos (imágenes)

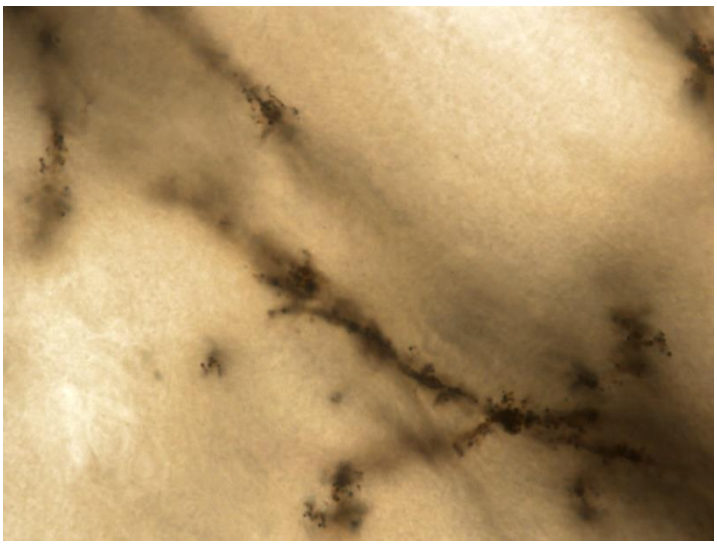


Ilustración 8 - Muestra de imagen con la técnica de Brightfield. Fuente: (Laboratorio BRAVE, Universidad Antonio de Nebrija)

Las imágenes empleadas son de tipo **brightfield** (Ilustración 8), obtenidas mediante técnicas de microscopía que emplean una iluminación directa sobre la muestra, lo que permite observar hasta los detalles más finos y estructuras de las espinas dendríticas.

Pero este tipo de imágenes presenta varios desafíos (Ruszczycki, y otros, 2012), entre ellos encontramos:

- **Sombras:** las técnicas de iluminación directa pueden generar sombras que dificultan la distinción entre la espina y el fondo.
- **Formaciones atípicas:** en ocasiones, las espinas dendríticas pueden presentar malformaciones o fusionarse con estructuras adyacentes, provocando que se complique su identificación.

- **Ruido de fondo:** el ruido de las imágenes puede formar parte de las imágenes debido por varias razones, desde imperfecciones propias del equipo hasta interferencias durante el proceso de captura.

Preprocesado de las imágenes

Antes de empezar con el etiquetado, se ha llevado a cabo un proceso de preprocesamiento para mejorar la calidad de las imágenes. Al principio, las imágenes presentaban algo de ruido y variaciones en la iluminación. Para solucionar estos problemas, se han seguido los siguientes pasos:

1. **Conversión a escala de grises:** las imágenes primero se convierten a escala de grises para eliminar las variaciones de color y reducir la dimensionalidad de la información, para facilitar el etiquetado.
   Para ello se ha creado este script para realizar el proceso a todas las imágenes:

```
# Recorre todas las carpetas, subcarpetas y archivos
for subdir, dirs, files in os.walk(root_folder):
    for file in files:
        # Comprueba si el archivo es una imagen .tiff
        if file.endswith('.tif'):
            # Crea la ruta completa al archivo
            input_path = os.path.join(subdir, file)

            # Crea la ruta de salida para el archivo, preservando la estructura de subcarpetas
            relative_path = os.path.relpath(input_path, root_folder)
            output_path = os.path.join(output_folder, relative_path)

            # Crea las subcarpetas necesarias en la ruta de salida
            os.makedirs(os.path.dirname(output_path), exist_ok=True)

            # Carga la imagen
            image = io.imread(input_path)

            # Comprobar si la imagen ya está en escala de grises
            if len(image.shape) > 2 and image.shape[2] == 3:
                # Convierte la imagen a escala de grises
                image_gray = color.rgb2gray(image)

                # Convierte la imagen de escala de grises a uint8 para guardarla correctamente
                image_gray = img_as_ubyte(image_gray)

                # Guarda la imagen resultante en la ruta de salida
                io.imsave(output_path, image_gray)
            else:
                # Guarda la imagen original en caso de que ya esté en escala de grises
                io.imsave(output_path, image)

print("¡Listo!")
```

Código 1- Script para convertir las imágenes originales en imágenes en la escala de grises. Fuente: (Visual Studio Code, María González, 2023)

Al tener las imágenes en varias carpetas y subcarpetas, el primer paso (Código 1) es recorrerlas todas y busca en ellas si hay archivos .tif (formato de archivo para almacenar imágenes de mapa de bits), crea la ruta de salida y comprueba si la imagen ya está en escala de grises si no es así la convierte en escala de grises y la guarda.

2. **Aplicación de Ecualización Adaptativo de Histograma (AHE):** se ha empleado esta técnica de AHE para mejorar el contraste de las imágenes. Esta técnica es muy útil en imágenes biomédicas ya que ajusta el contraste de forma adaptativa, haciendo que el resultado sea un resalte en detalle de las áreas específicas y mejorando la identificación de las estructuras de interés.
   Se ha creado este script para realizar el proceso a todas las imágenes:

```
# Recorre todas las carpetas, subcarpetas y archivos
for subdir, dirs, files in os.walk(root_folder):
    for file in files:
        # Comprueba si el archivo es una imagen .tiff
        if file.endswith('.tif'):
            # Crea la ruta completa al archivo
            input_path = os.path.join(subdir, file)

            # Crea la ruta de salida para el archivo, preservando la estructura de subcarpetas
            relative_path = os.path.relpath(input_path, root_folder)
            output_path = os.path.join(output_folder, relative_path)

            # Crea las subcarpetas necesarias en la ruta de salida
            os.makedirs(os.path.dirname(output_path), exist_ok=True)

            # Carga la imagen
            image = io.imread(input_path)

            # Aplica la ecualización adaptativa de histograma
            image_eq = exposure.equalize_adapthist(image, clip_limit=0.03)

            # Guarda la imagen resultante en la ruta de salida
            io.imsave(output_path, img_as_uint(image_eq))


print("¡Listo!")
```

Código 2 - Script para pasar a las imágenes por un proceso de Ecualización adaptativa de histograma. Fuente: (Visual Studio Code, María González, 2023)

Al igual que para el anterior script primero se sigue un proceso para navegar entre las diferentes carpetas y subcarpetas donde se tienen alojados los diferentes archivos, se buscan las imágenes que acaben en el formato .tif y (Código 2) se le aplica el proceso de AHE, donde “clip_limit = 0.03” se refiere a que este parámetro es evitar que el contraste se amplifique demasiado en los lugares de la imagen que son muy estrechas, es decir, en los lugares en los cuales hay pocos píxeles con un rango limitado de valores de intensidad y estar al 0.03 (3%) significa que ese es el límite que se ha tenido en cuenta para que no se sobrepase y aparezcan problemas de demasiado ruido o que la imagen se vea demasiado exagerada.

3. **Cambio de formato de .tif a .jpg:** debido a que cada imagen pesa demasiado, incluso GigaBytes, se ha optado por cambiarlas a formato .jpg para que facilitar el trabajo, debido al gran volumen de imágenes con las que se cuenta, además de otros factores que se han tenido en cuenta para tomar esta decisión, como la velocidad de procesamiento o la visualización.

   Se ha creado este script para realizar el proceso a todas las imágenes:

```
# Recorre recursivamente todas las subcarpetas
for dirpath, dirnames, filenames in os.walk(input_folder):
    for filename in filenames:
        # Si el archivo es una imagen .tif
        if filename.lower().endswith('.tif'):
            # Construye las rutas de entrada y salida
            input_path = os.path.join(dirpath, filename)
            output_path = os.path.join(output_folder,
                                       os.path.relpath(dirpath, input_folder), filename[:-4] + '.jpg')

            # Crea la carpeta de salida si no existe
            os.makedirs(os.path.dirname(output_path), exist_ok=True)

            # Abre la imagen con imageio
            image = imageio.imread(input_path)

            # Si la imagen tiene más de 2 dimensiones, toma solo el primer canal
            if image.ndim > 2:
                image = image[..., 0]

            # Si la imagen es de 16 bits, la convierte a 8 bits
            if image.dtype == np.uint16:
                image = (image / 256).astype(np.uint8)

            # Guarda la imagen en formato .jpg
            imageio.imsave(output_path, image)

print("¡Listo! Todas las imágenes .tif se han convertido a .jpg.")
```

Código 3 - Script para convertir las imágenes de formato .tif a formato .jpg. Fuente: (Visual Studio Code, María González, 2023)

Como esta serie de scripts (Código 3), en este último, como en los anteriores se navega por todas las carpetas y subcarpetas recursivamente y se buscan los archivos .tif,, se crean las variables de las rutas de entrada y salida, después se lee la imagen con “imageio”, que sirve para cargar el archivo .tif, se comprueba que el número de canales de la imagen sea mayor de 2 dimensiones (esto se hace ya que la imágenes de las que se están partiendo están en escala de grises, por lo que si encontrase alguna imagen que no sea en escala de grises no pasaría por el proceso). Y si la imagen es de 16 bits se convierte a 8 bits, dividiendo todos los valores entre 256, esto se hace así, debido a que el formato .jpg no soporta 16 bits por canal.

Obteniendo como resultado final (Ilustración 9) esta visualización de imagen, de la que se parte para etiquetarla en el siguiente paso del proceso.

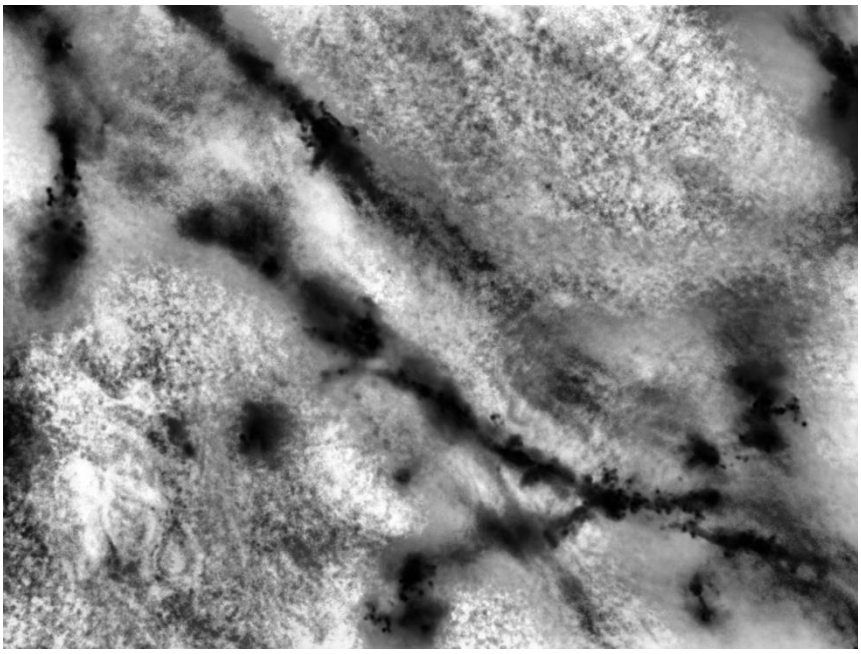


Ilustración 9 - Muestra de imagen preparada para la etiquetación. Fuente: (Laboratorio BRAVE, Universidad Antonio de Nebrija)

3.4.2. Funcionamiento del entorno de trabajo

Se ha utilizado dos entornos de trabajo para funciones diferentes el primero el de etiquetado de las imágenes y el segundo el proceso de entrenamiento del algoritmo con el conjunto de datos.

- **Etiquetado de fotos:** para el etiquetado de imágenes se ha utilizado la herramienta de CVAT.AI (Computer Vision Annotation Tool), desarrollada únicamente con el propósito de facilitar la tarea de anotar y etiquetar las imágenes o vídeos para la formación de modelos de aprendizaje automático.

  Se trata de una web con una interfaz muy fácil de usar y un conjunto de herramientas avanzadas que han sido de gran ayuda a la hora de marcar los objetos de interés con precisión. (Cvat.ai, s.f.)

  Se han etiquetado 6 tipos de espinas dendríticas y se han asociados todas ellas a 6 colores diferentes para distinguir los diferentes tipos (Ilustración 11):

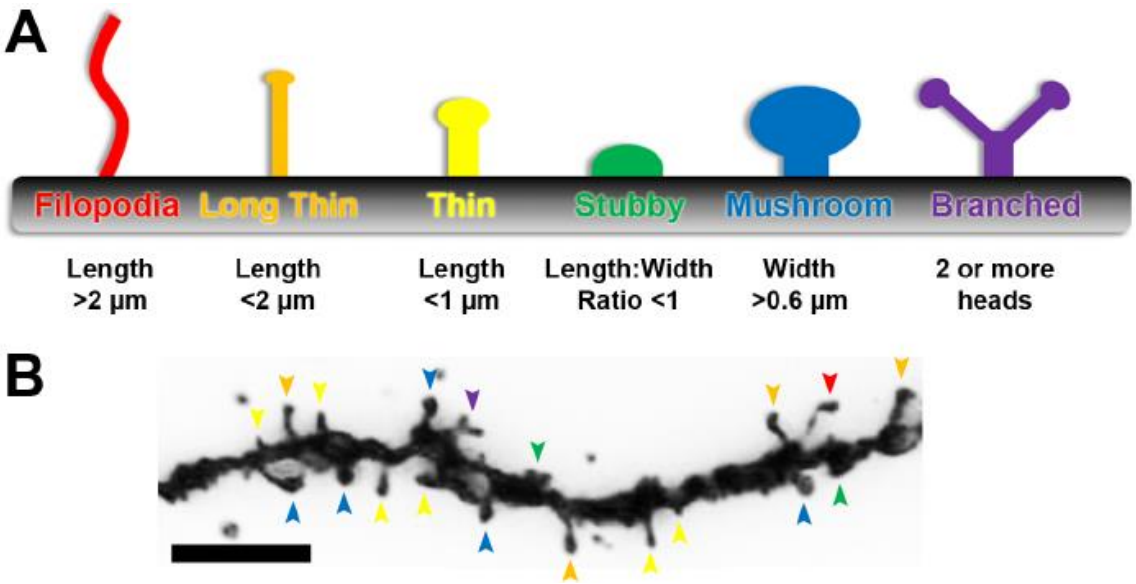


Ilustración 11 - Tipos de espinas dendríticas, con sus colores correspondientes. Fuente: (researchgate.net, Cagla Eroglu)

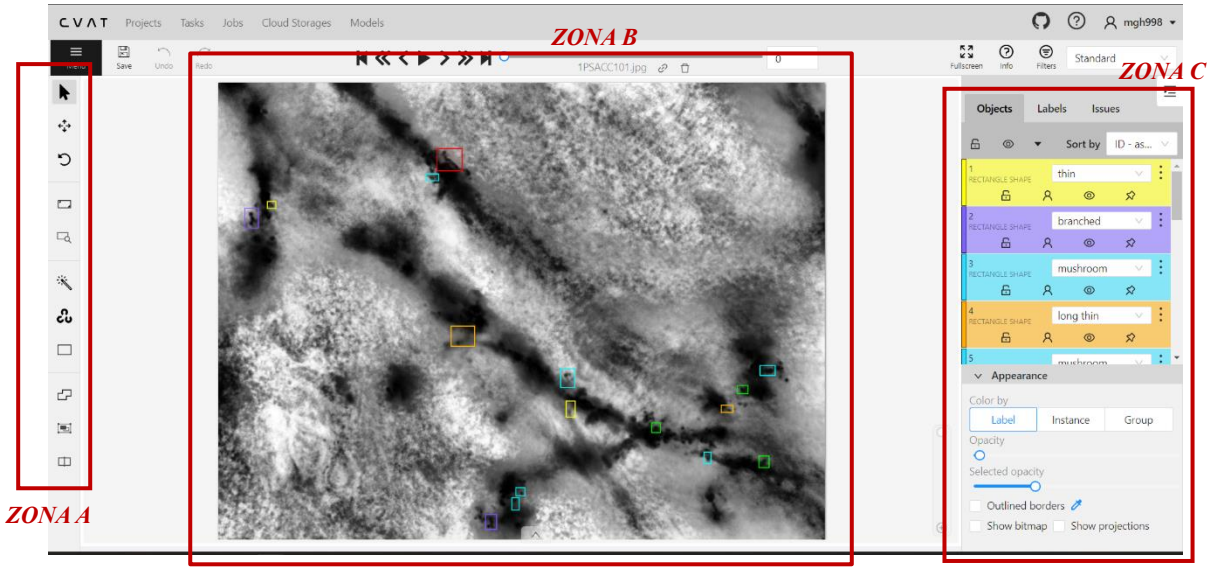


Ilustración 10 - Entorno de trabajo para etiquetar fotos. Fuente: (cvat.ai, María González Herrero, 2023)

A la hora de etiquetar se ha contado con este entorno de trabajo (Ilustración 10):

La zona A: Se encuentran las herramientas de etiquetado, en este caso se utiliza el rectángulo para elegir en un desplegable el tipo de espina dendrítica y asociarla con un color.

La zona B: Es el entorno de trabajo en el cual se van etiquetando todas las fotos con sus diferentes colores señalando que son tipos diferentes.

La zona C: Aparece una lista de todas las etiquetas encontradas en dicha imagen.

Por último, se elige como formato de exportación, Segmentation Mask 1.1, debido a que es el mejor para la segmentación de imágenes biomédicas según la forma de etiquetado que se ha seguido, porque genera máscaras binarias para cada objeto de interés. Esto ayuda a una integración directa con los algoritmos de segmentación, como el algoritmo con el que se trabajará U-Net, donde cada píxel es clasificado de forma que pertenezca o no a un objeto de interés.

- **Jupyter Notebook:** esta herramienta es una parte esencial para la investigación y el desarrollo en el aprendizaje automático y los pasos previos a este. Se elige este entorno de trabajo por una serie de ventajas que lo hacen el adecuado:

    1. Interactividad: al crearse los códigos en celdas individuales, facilita mucho el poder testear y depurar solo por fragmentos necesarios sin tener que estar ejecutando todo el programa entero cada vez.
    2. Visualización en Tiempo Real: se pueden generar y visualizar gráficos directamente en cada cuaderno, para que sea más fácil el análisis y la interpretación de los resultados.
    3. Integración con Bibliotecas: se integra perfectamente con las bibliotecas más populares en el aprendizaje automático, como TensorFlow, Pandas, Keras…

    Principalmente, se crean tres cuadernos:

    1. Asociación de cada etiqueta con su imagen: una vez etiquetadas las imágenes con el programa de CVAT, se asocia cada una con su respectiva máscara, asegurando así una correcta correspondencia entre los datos y las etiquetas.
    2. Preprocesamiento de las imágenes: como el algoritmo U-Net necesita que las imágenes tengan un tamaño específico, en concreto cuadrados de 256px, se utilizan técnicas para recortarlas y máscaras de forma uniforme sin perder información.
    3. Entrenamiento y validación: en la fase de entrenamiento y validación del modelo, se hace un seguimiento detallado del rendimiento, ajuste de los parámetros y visualización del resultado en tiempo real.

Al final la combinación de la herramienta CVAT para el etiquetado y los cuadernos de Jupyter para el procesamiento, entrenamiento y validación, permiten desarrollar un flujo de trabajo eficiente y sistemático, para garantizar en todo momento que los resultados sean los esperados y óptimos en la segmentación de las espinas dendríticas.

3.4.3. Procesamiento y análisis de features

Para el procesamiento de las imágenes de espinas dendríticas, se trabaja con el algoritmo U-Net debido a que se ajusta a los parámetros requeridos, además de ser una arquitectura de red neuronal convolucional (CNN) diseñada específicamente para la segmentación de imágenes biomédicas. En este campo se ha demostrado que es extremadamente eficiente, adoptándose a gran variedad de campos dentro de la imagen médica. (Ronneberger, Fischer, & Brox, 2015)

U-Net tiene este nombre debido a que su arquitectura se asemeja a una “U” y destaca por su capacidad de segmentar imágenes con precisión, es decir, asigna una etiqueta a cada píxel de la imagen para indicar a la categoría que pertenece (por ejemplo, una espina dendrítica tipo filopodia o tipo stubby). En vez de solo clasificar una imagen completa, U-Net proporciona una clasificación detallada a nivel de píxel.

Esta arquitectura (Ilustración 12) consta de 3 secciones, la contracción (codificación), la representación más profunda (cuello de botella) y la sección de expansión (decodificador).

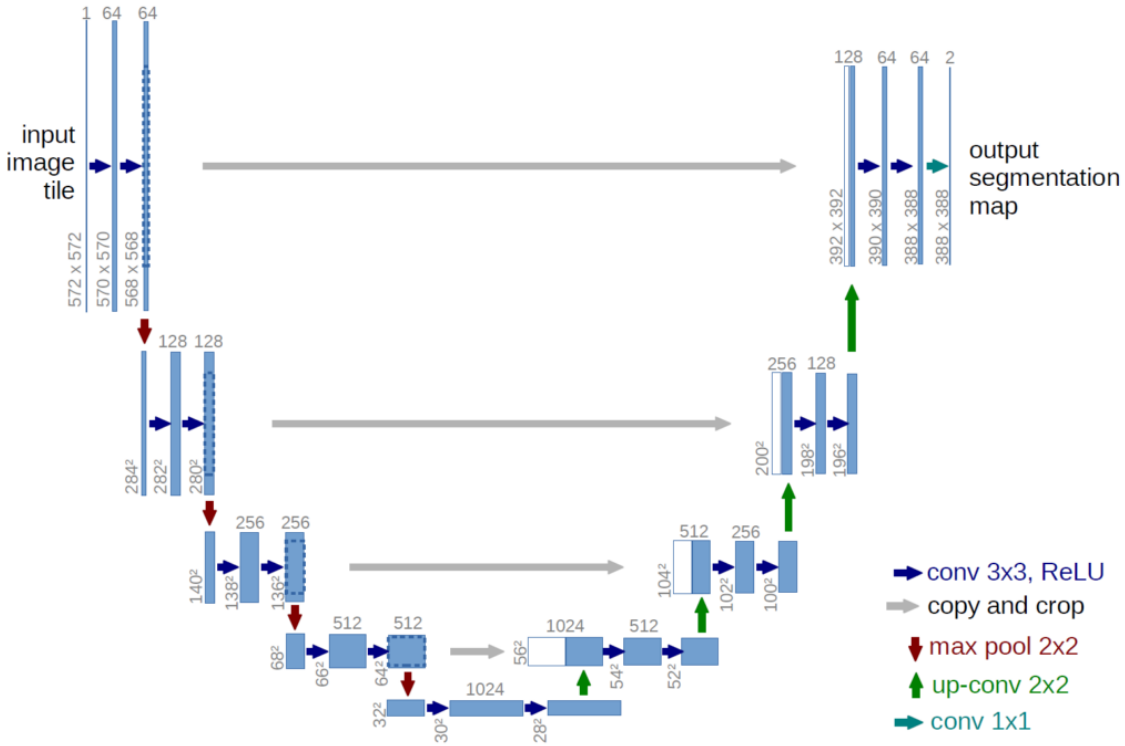


Ilustración 12 - Arquitectura del algoritmo de U-Net. Fuente: (DataScientest, 2022)

1. **Contracción:** empieza en el lado izquierdo de la arquitectura, en ella se captura la información de la imagen, es decir, se empieza con la imagen que se desee procesar.
   A continuación, cada bloque del lado del codificador tiene dos capas convolucionales seguidas de una operación de **max-pooling** con cada paso hacia abajo en la "U", la imagen se reduce a la mitad en la dimensión, pero duplicándose en profundidad, es decir, la resolución espacial disminuye, pero la red extrae y retiene cada vez más características en cada nivel.
2. **Representación más profunda:** después de haber pasado por el último bloque de max-pooling del lado izquierdo, hay otro bloque más **convolucional**, siendo esta parte de la red la que tiene una representación más abstracta de la imagen de entrada.
3. **Expansión:** a medida que se asciende por el lado derecho de la "U", la imagen va aumentando se **up-samplea** (proceso por el cual se aumenta el tamaño de una imagen, es decir, convierte una imagen de menor resolución a una de mayor resolución) para aumentar su dimensión. Después de cada proceso de up-sampling, se realiza una operación de contracción, combinando características de la capa actual con las características correspondientes al lado izquierdo. Este proceso también es conocido con el nombre de conexiones de salto.
   En este lado cada etapa también cuenta con dos capas convolucionales y a medida que se asciende por la "U", la resolución espacial de las características se va incrementando y la profundidad se reduce a la mitad.
   Al final de la red, después del último bloque convolucional del lado derecho, hay una última capa convolucional que reduce el número de canales al número de clases deseadas. (Sankesara, 2019)

3.5. Implementación

3.5.1. Segmentation Mask

Antes de iniciar el proceso de entrenamiento, es una parte fundamental garantizar que cada imagen tenga su etiqueta correspondiente correctamente asociada.

En este libro de Jupyter se ha encargado de emparejar automáticamente cada imagen con su respectiva etiqueta, asegurando que las entradas (imágenes) y las salidas (etiquetas) estén alineadas.

Funcionamiento:

```python
# --------- CARGA DE DATOS -----------
# Función para obtener todos los archivos con una extensión específica de una carpeta y subcarpetas
def get_all_files_recursive_image(root, extension=".jpg"):
    files = []
    for subdir, _, filelist in os.walk(root):
        for file in filelist:
            if file.endswith(extension):
                files.append(os.path.join(subdir, file))
    return files

# Función para obtener todos los archivos con un sufijo específico de una carpeta y subcarpetas
def get_all_files_recursive_mask(root, suffix="_mask.png"):
    files = []
    for subdir, _, filelist in os.walk(root):
        for file in filelist:
            if file.endswith(suffix):
                files.append(os.path.join(subdir, file))
    return files

# Rutas a las carpetas raíz
root_images_path = "F:\\espinas_descomprimidas\\image_tag"
root_masks_path = "F:\\espinas_descomprimidas\\image_tag"

# Obtener listas de rutas completas
image_files = get_all_files_recursive_image(root_images_path, ".jpg")
mask_files = get_all_files_recursive_mask(root_images_path, ".png")

# Cargar imágenes y máscaras en listas
images = [Image.open(f) for f in image_files]
masks = [Image.open(f) for f in mask_files]
```

Código 4 - Carga de datos de entrada de las imágenes y las máscaras de capa del programa de asociación de imágenes con su máscara de capa. Fuente: (Visual Studio Code, María González, 2023)

Los pasos previos del programa son cargar las librerías necesarias, cargar los datos, que en este caso son imágenes .jpg y para las etiquetas acaban en *"_mask.png"*. Se añaden las rutas (Código 4) de donde se van a obtener ambos datos de entrada y se cargan en listas.

```python
# --------- TRANSFORMACIÓN DE MÁSCARAS -----------
# Definir un diccionario para mapear colores RGB a etiquetas numéricas
color_map = {
    (0, 0, 0): 0,       # negro = fondo
    (125, 98, 255): 1, # morado = branched
    (255, 0, 0): 2,     # rojo = filopodia
    (255, 171, 1): 3,  # naranja = long thin
    (43, 220, 255): 4, # azul = mushroom
    (61, 207, 61): 5,  # verde = stubby
    (255, 255, 0): 6,  # amarillo = thin
}
```
✓ 0.0s

Código 5 - Diccionario para mapear los colores de la máscara de capa del programa de asociación de imágenes con su máscara correspondiente. Fuente: (Visual Studio Code, María González, 2023)

Se define un diccionario asignado (Código 5) a cada color que pertenece a un tipo de espina dendrítica un código numérico, incluido el fondo de la máscara que es el 0 de las etiquetas numéricas.

```python
# Función para convertir máscaras de color a etiquetas numéricas
def mask_to_label(mask_image):
    label_mask = Image.new("L", mask_image.size)
    pixels = label_mask.load()
    for i in range(mask_image.width):
        for j in range(mask_image.height):
            pixel_color = mask_image.getpixel((i, j))
            if pixel_color in color_map:
                pixels[i, j] = color_map[pixel_color]
            else:
                print(f"Color no definido encontrado: {pixel_color}. Asignando al fondo (0).")
                pixels[i, j] = 0  # asigna al fondo o cualquier valor que desees
    return np.array(label_mask)

# Aplicar la función a todas las máscaras y convertir imágenes a arrays
labeled_masks = [mask_to_label(mask) for mask in masks]
images = [np.array(img) for img in images]

# Luego de procesar las imágenes y máscaras, guardar en archivos .npy
np.save('processed_images.npy', images)
np.save('processed_masks.npy', labeled_masks)
```

Código 6 - Función para convertir máscaras de clor a etiquetas numéricas del programa de asociación de imágenes con su máscara de capa. Fuente: (Visual Studio Code, María González, 2023)

La función (Código 6) convierte una imagen de máscara RGB en una matriz de etiquetas numéricas usando el diccionario que se ha definido antes con el nombre de *color_map*.

Lo primero que se hace es crear una nueva imagen en escala de grises del mismo tamaño que la máscara de entrada e itera dentro de cada píxel de la máscara consultando el color con el diccionario de *color_map* para obtener la etiqueta numérica correspondiente.

Además, si encuentra algún otro color que no son los establecidos como las 6 dendríticas, se le asigna el valor de 0 correspondiente al fondo para gestionar cualquier excepción. Para finalizar se devuelve las etiquetas como una matriz numpy y se guardan.

Tarda tres horas en ejecutarse y grabarse los datos.

También dado que U-Ney requiere de imágenes de 256 x 256 píxeles y con las que se cuentan tienen una dimensión de 1360 x 1024 píxeles, se ha optado por dividir en múltiples fragmentos de cuadrados de 256 píxeles, porque si se redimensionaban directamente las imágenes podría resultar pérdida de información valiosa y una disminución de la resolución.

Funcionamiento:

```python
# Función para extraer mosaicos de 256x256
def extract_patches(img_arr, mask_arr, patch_size=256):
    img_patches = []
    mask_patches = []

    for img, mask in zip(img_arr, mask_arr):
        for i in range(0, img.shape[0], patch_size):
            for j in range(0, img.shape[1], patch_size):
                if (i + patch_size < img.shape[0]) and (j + patch_size < img.shape[1]):
                    img_patch = img[i:i+patch_size, j:j+patch_size]
                    mask_patch = mask[i:i+patch_size, j:j+patch_size]

                    img_patches.append(img_patch)
                    mask_patches.append(mask_patch)

    return np.array(img_patches), np.array(mask_patches)
```

Python

Código 7 - Función de extracción de mosaicos de 256 x 256 px para el programa de recortes de mosaicos. Fuente: (Visual Studio Code, María González, 2023)

Primero (Código 7) se cargan las librerías necesarias, después se crea la función para extraer mosaicos de 256 x 256 píxeles y lo mismo de su máscara correspondiente. En la función *extract_patches* se pasan dos listas de matrices que representan las imágenes (*img_arr*) y las máscaras (*mask_arr*), después para cada una, recorre la función en pasos del tamaño del mosaico y extrae todos los mosaicos de la imagen y de la máscara siempre que no se salgan de los límites.

Una vez extraídos se añaden a las listas correspondientes para cada imagen y máscara (*img_patches* y *mask_patches*).

El programa no tarda en ejecutarse más de un minuto.

3.5.2. Entrenamiento y Desarrollo del Modelo

El entrenamiento del modelo es uno de los pasos más importantes en el proceso de desarrollo, debido a que en esta fase es donde el modelo se ajusta a los parámetros necesarios para minimizar el error entre las predicciones y los datos reales.

A continuación, se desglosará el código.

```python
# --------- IMPLEMENTACIÓN DE LA MÉTRICA IoU -----------
def iou(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32) # Convertimos y_true a float32

    y_pred_labels = tf.argmax(y_pred, axis=-1)
    y_pred_labels = tf.cast(y_pred_labels, tf.float32)  # Convertimos y_pred_labels a float32
    y_pred_labels = tf.expand_dims(y_pred_labels, -1)

    intersection = tf.reduce_sum(y_true * tf.cast(tf.equal(y_true, y_pred_labels), tf.float32))
    union = tf.reduce_sum(y_true) + tf.reduce_sum(y_pred_labels) - intersection

    return intersection / union
```

Código 8 - Implementación de la métrica IoU. Fuente: (Visual Studio Code, María González, 2023)

La métrica IoU (Código 8) es utilizada en tareas de detección y segmentación de objetos, midiendo el solapamiento entre dos áreas, que generalmente son las áreas de la imagen verdadera y de la predicción.

La función *IoU* toma dos tensores (*y_true* y *y_pred)*, siendo uno la imagen verdadera y otro la predicción respectivamente y devuelve el IoU, para evaluar la calidad de las predicciones.

```python
# --------- CARGA DE DATOS -----------
images = np.load('C:\\Users\\Maria\\Documents\\TFG\\espinas_descomprimidas\\processed_images.npy')
labeled_masks = np.load('C:\\Users\\Maria\\Documents\\TFG\\espinas_descomprimidas\\processed_masks.npy')

# --------- DIVISIÓN DE DATOS -----------
X_train, X_val, y_train, y_val = train_test_split(images, labeled_masks, test_size=0.2, random_state=42)
```

Código 9 - Carga de las imágenes y máscaras y división de los datos. Fuente: (Visual Studio Code, María González, 2023)

En cuanto a la carga de datos (Código 9), se cargan tanto las imágenes como las máscaras previamente procesadas. Después esos datos pasan por una división de los conjuntos y subconjuntos de entrenamiento y prueba/validación.

También la variable *test_size = 0.2*, especifica que el 20% del conjunto total de datos será separado para la validación y el 80% se usará en el entrenamiento. Y *random_state = 42*, es un parámetro que establece un estado aleatorio, para asegurar que cada vez que se ejecute el código, se obtiene la misma división de datos.

```
# --------- DEFINICION DE MODELO U-NET -----------
def unet_model(input_size=(256, 256, 1)):
    inputs = Input(input_size)

    # Contraccion
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = BatchNormalization()(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)
    conv2 = BatchNormalization()(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)
    conv3 = BatchNormalization()(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(512, 3, activation='relu', padding='same')(pool3)
    conv4 = BatchNormalization()(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    conv5 = Conv2D(1024, 3, activation='relu', padding='same')(pool4)
    conv5 = BatchNormalization()(conv5)
    conv5 = Dropout(0.5)(conv5)

    # Expansion
    up1 = UpSampling2D(size=(2, 2))(conv5)
    merge1 = concatenate([conv4, up1], axis=3)
    conv6 = Conv2D(512, 3, activation='relu', padding='same')(merge1)
    conv6 = BatchNormalization()(conv6)

    up2 = UpSampling2D(size=(2, 2))(conv6)
    merge2 = concatenate([conv3, up2], axis=3)
    conv7 = Conv2D(256, 3, activation='relu', padding='same')(merge2)
    conv7 = BatchNormalization()(conv7)

    up3 = UpSampling2D(size=(2, 2))(conv7)
    merge3 = concatenate([conv2, up3], axis=3)
    conv8 = Conv2D(128, 3, activation='relu', padding='same')(merge3)
    conv8 = BatchNormalization()(conv8)

    up4 = UpSampling2D(size=(2, 2))(conv8)
    merge4 = concatenate([conv1, up4], axis=3)
    conv9 = Conv2D(64, 3, activation='relu', padding='same')(merge4)
    conv9 = BatchNormalization()(conv9)

    # Capa de salida con 7 clases (incluyendo el fondo)
    outputs = Conv2D(7, 1, activation='softmax')(conv9)

    # Compilar modelo
    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy', iou])
    return model
```

Código 10 - Arquitectura del modelo U-Net. Fuente: (Visual Studio Code, María González, 2023)

En el modelo (Código 10), se han seleccionado funciones y métricas basadas en las características del problema y los datos disponibles. La selección de una arquitectura menos profunda se ha elegido a limitaciones computaciones, por lo que se ha compensado esta elección de menos profundidad con funciones de pérdida efectivas.

La función *unet_model* cuenta con una entrada de *(256, 256, 1)* que se refiere a imágenes de 256 x 256 píxeles y un único canal, esto se be a que se trabaja con imágenes en escala de grises.

Se construye la arquitectura que consta de dos partes:

- **Fase de contracción:** En esta fase se condensa la información y la resolución espacial de la imagen disminuye mientras que la resolución de las características aumenta como ya se ha mencionado con anterioridad.
  En cada nivel de esta fase se usan capas de convolución. La elección de usar capas en aumento comienza con la de 64, 128, 256 y 512 nodos, indicando la estructura de doble profundidad que se va duplicando en cada nivel. Sigue el tamaño del filtro, que es 3, seleccionado este tamaño de filtro porque es lo suficiente grande para capturar bordes o texturas, pero no demasiado grande para que sea muy costosa su ejecución.
  La siguiente variable es *activation = 'relu'*, se refiere a la función **ReLU** (Rectified Linear Unit), que transforma los valores negativos en 0 y a los positivos no los hace ningún cambio. Sirve para aprender representaciones más complejas.
  Y la variable *Padding = 'same'* se utiliza para que la imagen conserve las dimensiones originales después de la convolución.
  La función de *MaxPooling2D* especifica el tamaño de cada ventana del bloque y son la entrada de los convx.
- **Fase de expansión:** Se encarga de recuperar la resolución espacial que se ha perdido durante la fase de contracción y refina la segmentación.
  Los UpSampling se encargan de aumentar la resolución y toman de entrada las fases de convolución (*conx*) anterior, después la función *concatenate*, une el resultado que corresponda con su mapa de características de la fase de contracción.
  Esto combina la información de localización de la fase de expansión con datos de la etapa de contracción. La convolución en esta fase es parecida a la contracción, pero con la finalidad de refinar el resultado.

Se termina con la capa de salida y la compilación del modelo.

En la capa de salida se activan 7 clases ya que cuenta el fondo como una clase más, donde cada uno de los 7 canales contiene un mapa de activaciones para cada clase diferente, el siguiente es el tamaño del kernel que es 1 debido q que la convolución se realza sobre un único píxel a la vez. Y activation = 'softmax' es la función de activación que convierte las activaciones en probabilidades, es decir, para cada píxel en la imagen de salida, se tendrán 7 valores (uno por canal) que sumarán uno, y el canal con el valor más alto es la clase que se ha predicho para ese píxel.

Para la compilación del modelo, la primera línea crea el modelo de Keras a partir de las entradas y salidas definidas anteriormente y para la segunda línea configura el modelo para el entrenamiento con unas características en concreto.

Optimizer = 'adam' es una variante de un método de descenso de gradiente que se utiliza para ajustar los pesos del modelo durante el entrenamiento. Y *metrics = ['accuracy']*, sirve para que durante el entrenamiento se pueda monitorear también la precisión, indicando el porcentaje de las predicciones del modelo que es correcta.

```
# --------- ENTRENAMIENTO DEL MODELO ----------- (puede tardar varios días)

# Definir el early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=5, verbose=1, restore_best_weights=True)
# 'patience' es el número de épocas sin mejora después de las cuales se detendrá el entrenamiento.
# 'restore_best_weights' se asegura de que al final del entrenamiento el modelo regrese a los pesos
# de la época con la mejor pérdida de validación.

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=40,
    batch_size=16,
    callbacks=[early_stop]  # aqui se añade el early stopping
)

# Visualizar la gráfica de pérdida
train_loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(12, 6))
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper right')
plt.show()
```

Código 11 - Fase de entrenamiento del modelo. Fuente: (Visual Studio Code, María González, 2023)

El entrenamiento (Código 11), se refiere al proceso por el cual el modelo “aprende” de los datos proporcionados para que aprenda patrones y relaciones subyacentes en los datos de entrenamiento, de modo que pueda hacer predicciones precisas de datos no vistos previamente.

Se utiliza *early_stop* para detener el entrenamiento si no ve mejoras en la pérdida de validación después de 5 épocas (*patience = 5)*, evitando el sobreajuste y ahorrando tiempo y recursos computacionales.

La variable *history*, guarda información sobre el entrenamiento. La función *fit()* es un método de Keras utilizado para entrenar el modelo, alimentando el modelo con datos y ajustando los pesos del modelo para minimizar la función de pérdida. La variable *batch_size = 16*, es el tamaño del lote que determina cuantas muestras se usaran en cada actualización de los pesos, poder poner un tamaño mayor o menor dependerá de la potencia y carga computacional que cada equipo soporte. La variable *epoch = 50*, se refiere a cuando una época pase a través de todo el conjunto de datos de entrenamiento. Y las funciones de *callbacks = [early_stop]*, se aplica en ciertas etapas del proceso en este caso se utiliza *early_stop*, que permite detener el entrenamiento si el modelo deja de mejorar.

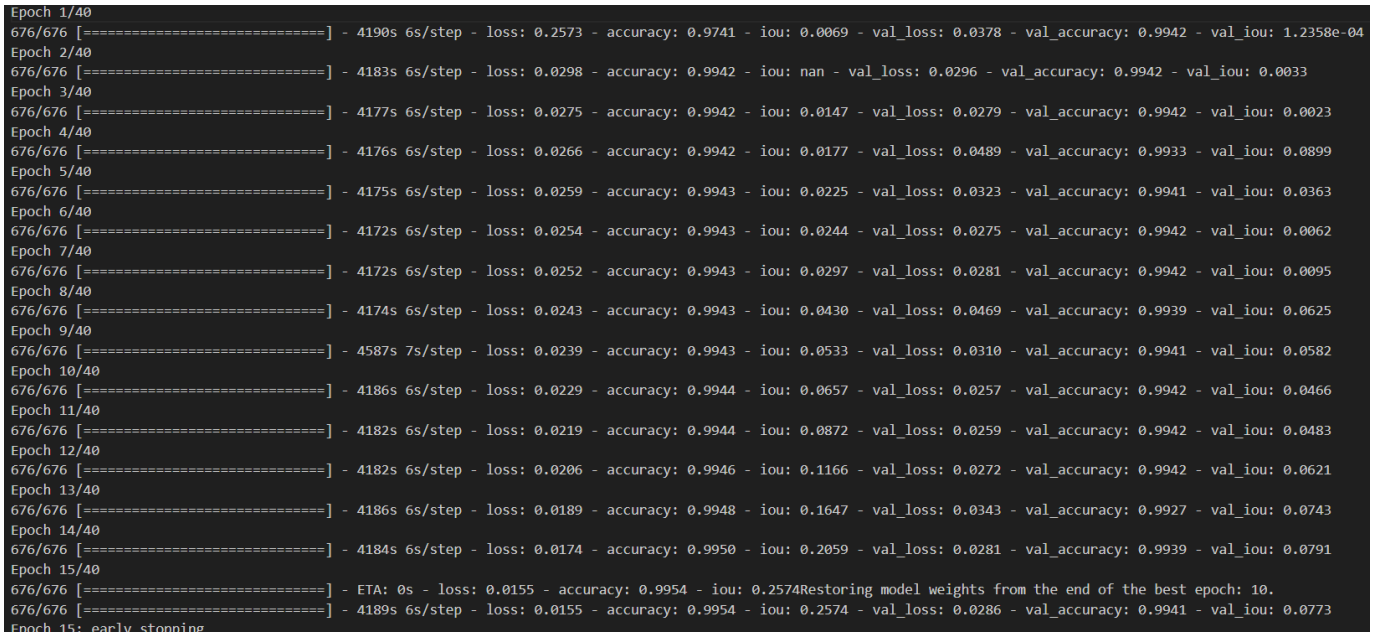


Ilustración 13 - Épocas con sus parámetros del modelo de entrenamiento. Fuente: (Visual Studio Code, María González, 2023)

Mientras se está ejecutando el entrenamiento (Ilustración 15) sale por pantalla unas estadísticas para monitorear el proceso.

- **Epoch (época):** Iteraciones completas del conjunto de entrenamiento. Se han realizado 15, aunque estaba programado para tener 40, esto se debe a que se detuvo de forma anticipada en la época 15 debido a la técnica de *early stopping*, que detuvo el entrenamiento al comprobar que el modelo no mejoraba después de 5 épocas para evitar el sobreajuste y ahorrar tiempo.
- **Loss (pérdida):** Representa la diferencia entre las predicciones del modelo y los valores reales, para saber que está mejorando hay que fijarse que se vaya reduciendo en cada época, lo que indica como es este caso que está mejorando.
- **Accuaracy (precisión):** Métrica que muestra el porcentaje de predicciones correctas del modelo. En este caso empieza con un 97.41% en la primera época y mejora durante todo el proceso hasta pararse en la época 15 con un 99.54%.
- **IoU (Intersection over Union):** Métrica que mide la superposición entre las áreas predichas y las reales. Cuando el valor es 1 o cercano significa que la coincidencia es perfecta, mientras que los valores cercanos al 0 indican que no hay ningún tipo de precisión. En las primeras épocas es bastante bajo, siendo 0.0069 lo que es bastante malo, pero al ser el principio del entrenamiento no pasa nada porque se puede ver una mejoría hasta la última época parando en 0.2574.
- **Val_loss, val_accuaracy, val_iou:** Métricas calculadas en el conjunto de validación, es decir, datos no vistos por el modelo durante el entrenamiento. Estos valores son cruciales para mostrar cómo se comporta el modelo con datos nuevos. En general, se busca que sean consistentes y que no haya gran diferencia entre ellos porque podría significar un signo de desajuste.

El comportamiento del modelo a lo largo de las épocas refleja una curva de aprendizaje positiva y ascendente, es especial en las primeras diez iteraciones. La métrica de pérdida, indica que se desvía de la predicción del modelo de los valores reales, descendiendo de forma notable desde la primera época. Esto sugiere que el modelo ha optimizado los pesos de la forma correcta.

La precisión desde su inicio ha superado el 97%, este nivel tan alto indica que el modelo es capaz de identificar y clasificar correctamente la mayoría de los píxeles en las imágenes, pero hay que tener en cuenta que no al tener una alta precisión signifique que las las áreas segmentadas se superpongan perfectamente a las reales.

El IoU en la segunda época muestra un valor de ‘NaN’, lo que significa que no es numérico, esto puede deberse a varios motivos, pero uno de los más comunes suele ser que el denominador ha sido 0, es decir, es posible que durante esa única época el modelo no haya podido predecir de forma correcta ningún área de interés. Esta situación es una clara indicación de que el modelo tuvo dificultades en esa época específicamente, pero a partir de esa a mostrado una clara

mejoría y alta precisión. También hay que tener en cuenta que ha acabado con un valor cercano al 1 pero que el modelo aun siendo capaz de identificar zonas de interés con una alta precisión, todavía enfrenta desafíos en ciertas predicciones.

La estabilidad de las métricas de validación sugiere que el modelo es capaz de generalizar sus predicciones a nuevos datos, lo que es una señal positiva.

```
# Visualizar la gráfica de pérdida
train_loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(12, 6))
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper right')
plt.show()
```

Código 12 - Ajuste de los parámetros de la gráfica de pérdida. Fuente: (Visual Studio Code, María González, 2023)

Por último, se genera una gráfica de pérdida (Código 12), en la cual es una medida del error entre las predicciones del modelo y las verdaderas etiquetas y evoluciona a lo largo del tiempo o las épocas (Courville, 2016).

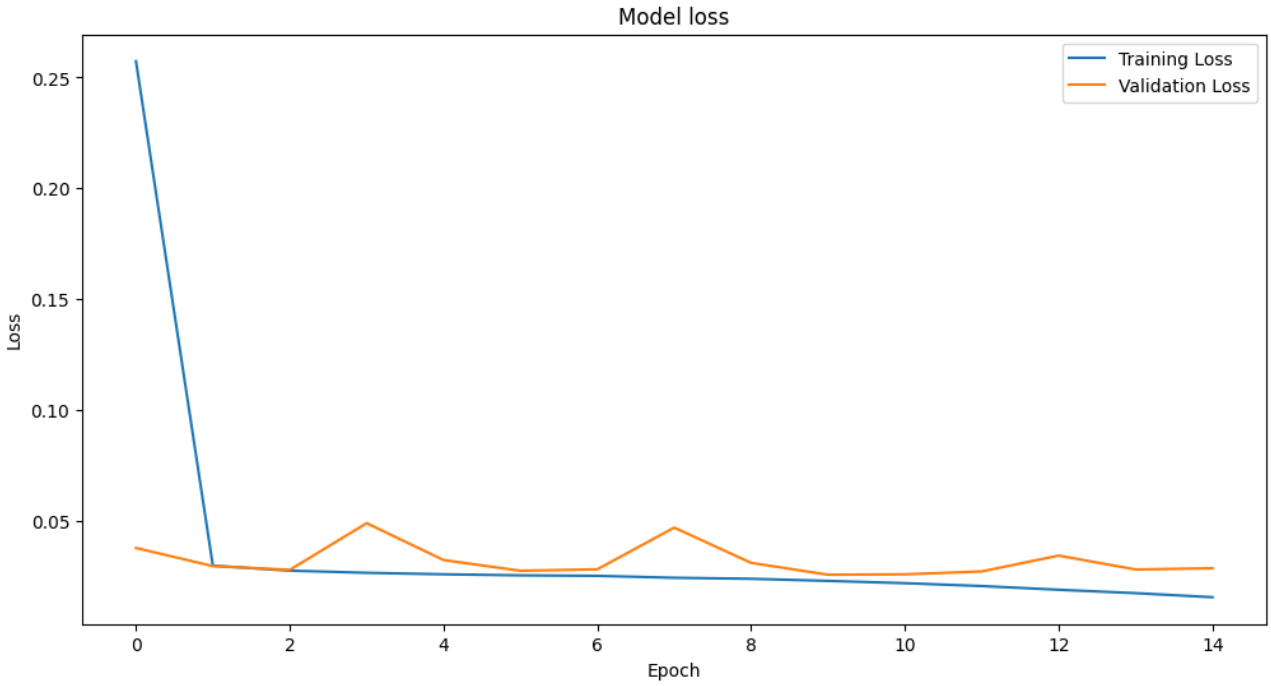


Ilustración 14 - Visualización gráfica de pérdidas. Fuente: (Visual Studio Code, María González, 2023)

En la gráfica (Ilustración 14), se observa que la línea azul perteneciente al entrenamiento empieza con valores muy altos de pérdida, pero según van avanzando las épocas va mejorando. Y la línea naranja perteneciente a la validación empieza en valores cercanos al 0.05 de pérdida y durante el resto de las épocas hay pequeños picos que no superan los 0.05 puntos de pérdida.

Al final la gráfica muestra un modelo de aprendizaje rápido de los datos de entrenamiento, logrando un buen equilibrio entre el entrenamiento y la validación, lo que sugiere una generalización adecuada. Sin embargo, todavía podría mejorar en potencial de términos de estabilidad en datos no vistos.

3.5.3. Validación

Fase del modelo en la cual se valida el rendimiento del modelo con un conjunto de datos, en este caso el conjunto de prueba que se está utilizando se definió al principio de la fase de entrenamiento, siendo este el 20% de los datos. Esta fase es esencial para saber cómo funcionaría el modelo en el "mundo real", es decir con datos que no se han visto durante el entrenamiento.

```
val_loss, val_acc, val_iou = model.evaluate(X_val, y_val, verbose=1)
print(f'Validation Accuracy: {val_acc * 100:.2f}%')
print(f'Validation Loss: {val_loss:.4f}')
print(f'Validation IOU: {val_iou:.4f}')
print("Fin de la evaluacion del modelo")

# --------- PREDICCIÓN DE MÁSCARAS -----------
predictions = model.predict(X_val)
print("Fin de la prediccion de mascaras")

# Convertir las predicciones de probabilidades a etiquetas de clase
predicted_masks = np.argmax(predictions, axis=-1)
print("Fin de la conversion de probabilidades a etiquetas de clase")
```

Código 13 - Fase de validación, junto a la predicción de máscaras. Fuente: (Visual Studio Code, María González, 2023)

La validación del modelo (Código 13), utiliza el método *evaluate,* con el cual se evalúa el modelo en un conjunto definido como *X_val* y *y_val*, haciendo que se devuelven las métricas que se definieron a la hora de compilar el modelo.

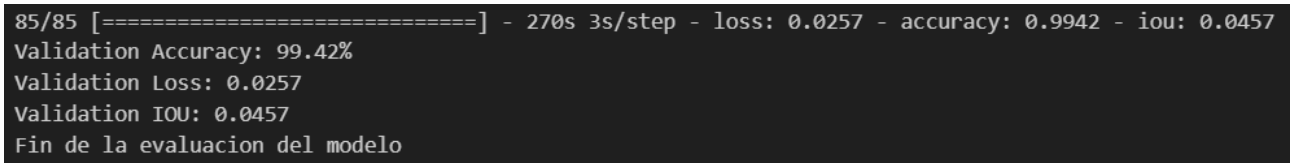


Ilustración 15 - Output resultante con sus parámetros de la validación del modelo. Fuente: (Visual Studio Code, María González, 2023)

El output resultante (Ilustración 15) proporciona la siguiente información:

- **85/85 – 270s 3s/step:** Existen 85 batches (épocas) de datos que se han completado, tardando en total 270 segundos y 3 segundos por cada batch.
- **Loss – 0.0257:** Pérdida promedio calculada en el conjunto de validación, se utiliza para saber si se ha comportado de manera correcta o no el modelo. Una pérdida de 0.0257 es bastante baja, lo que sugiere que el modelo se ha ajustado bastante bien a los datos de validación.
- **Accuracy – 0.9942 o Validation Accuracy – 99.42%:** La precisión es una medida que se indica en porcentaje y se refiere a las predicciones correctas que se hayan predicho. Una precisión del 99.42% es muy buena y significa que el modelo está haciendo predicciones muy precisas.
- **iou – 0.0457:** Mide si se superponen de manera correcta las áreas predichas por el modelo en las imágenes segmentadas, el valor es algo bajo lo que sugiere que el modelo está clasificando correctamente la mayoría de los píxeles, como indica la alta precisión, pero que algunas áreas predichas no son semejantes a las reales.

```python
# --------- VISUALIZACIÓN DE LAS PREDICCIONES -----------
def plot_sample(x, y, prediction):
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

    ax1.imshow(x, cmap='gray')
    ax1.set_title('Image')

    ax2.imshow(y, cmap='gray')
    ax2.set_title('True Mask')

    ax3.imshow(prediction, cmap='gray')
    ax3.set_title('Predicted Mask')

# Mostrar 5 ejemplos al azar
indices = np.random.choice(len(X_val), 15)
for i in indices:
   plot_sample(X_val[i], y_val[i], predicted_masks[i])

print("Fin de la muestra de ejemplos al azar")
```

Código 14 - Parámetros para visualizar las predicciones de la validación. Fuente: (Visual Studio Code, María González, 2023)

Se visualizarán las predicciones (Código 14) realizadas al modelo. Para ello se declara una función llamada *plot_sample* que recibe argumentos de la imagen (*x*), la máscara verdadera (*y*) y la máscara predicha (*prediction*). Se seleccionan un conjunto de ejemplos al azar y se muestran los resultados.

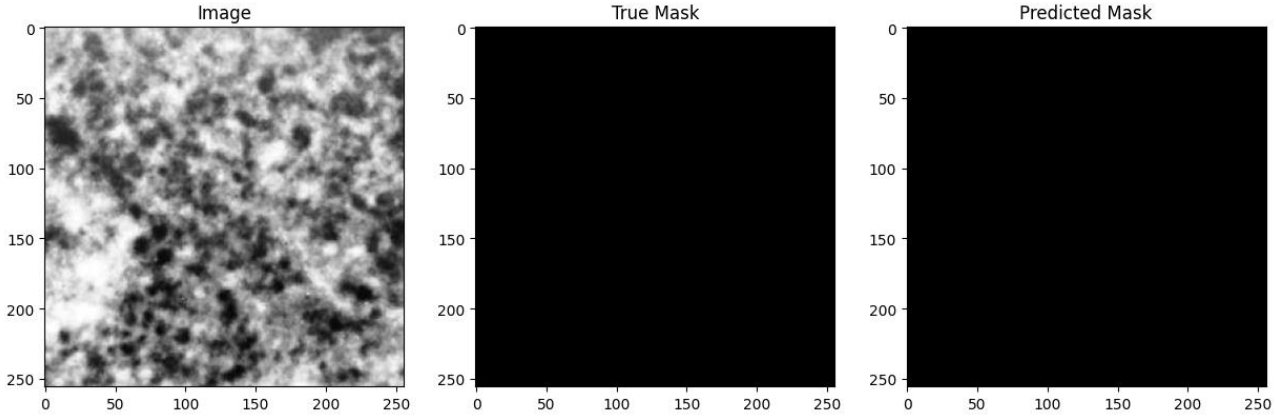


Ilustración 16 - Comparativa de las máscaras de capa reales con las predichas junto a la imagen original, parte 1 de 2. Fuente: (Visual Studio Code, María González, 2023)

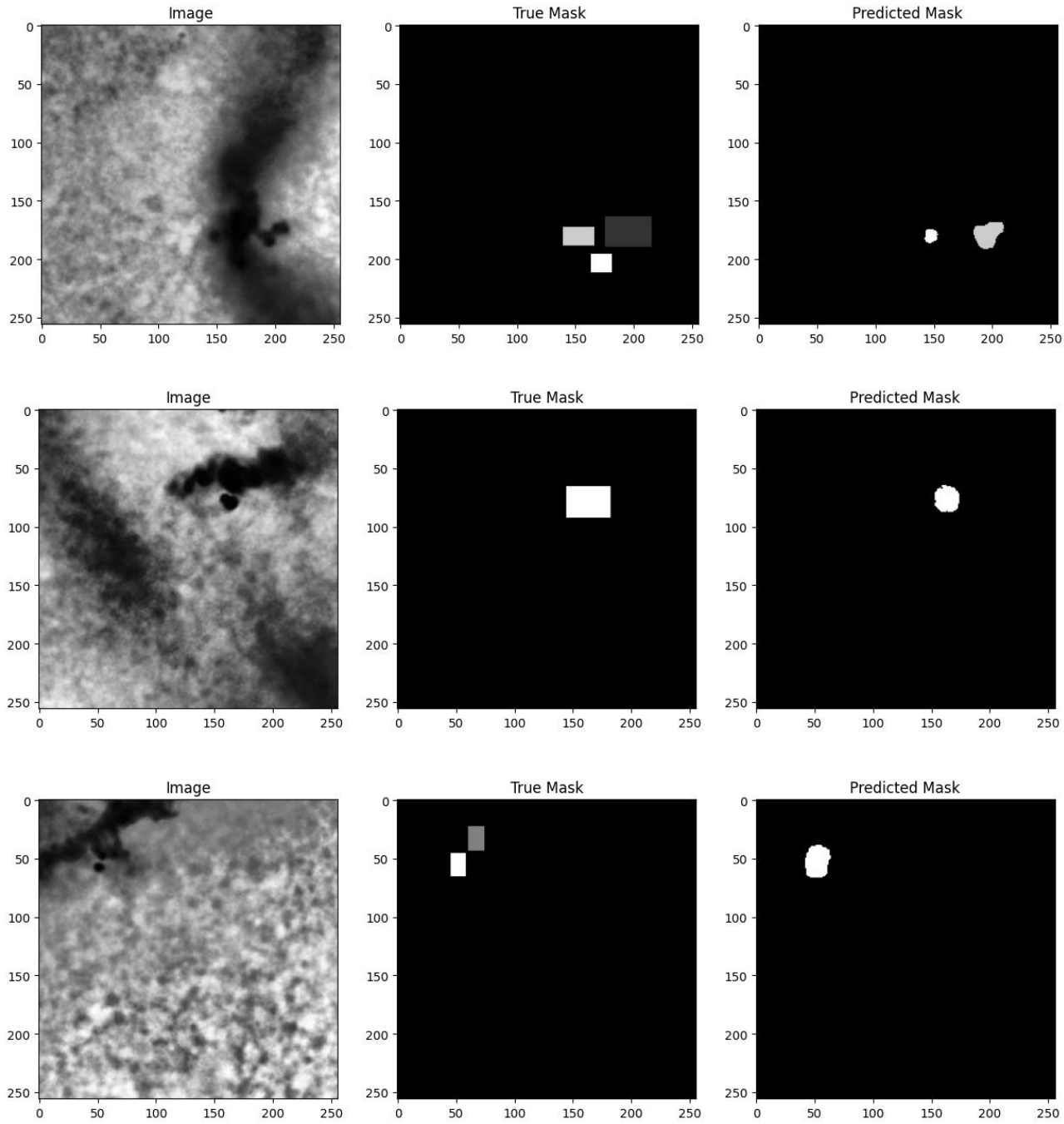


Ilustración 17 - Comparativa de las máscaras de capa reales con las predichas junto a la imagen original, parte 2 de 2. Fuente: (Visual Studio Code, María González, 2023)

Cómo se puedes observar en las Ilustraciones 16 y 17. Cuando mejor predice es cuando solo hay fondo en el resto de los casos sí que predice bastante bien las espinas dendríticas, aunque a veces hay errores esto se puede deber a la propia complejidad del problema.

```
# --------- MAPA DE CALOR DE ERRORES -----------
index = np.random.choice(len(X_val))
sample_image, sample_mask, sample_prediction = X_val[index], y_val[index], predicted_masks[index]

error_map = np.abs(sample_mask - sample_prediction)

plt.imshow(error_map, cmap='hot')
plt.colorbar()
plt.title('Error Heatmap for a Random Image')
plt.show()
```

Código 15 - Ajuste de los parámetros para generar el mapa de calor de errores. Fuente: (Visual Studio Code, María González, 2023)

Y por último los mapas de calor (código 15) que sirven para representar valores individuales de una matriz que se representan mediante colores, son muy útiles para visualizar y comprender la información compleja de manera intuitiva.

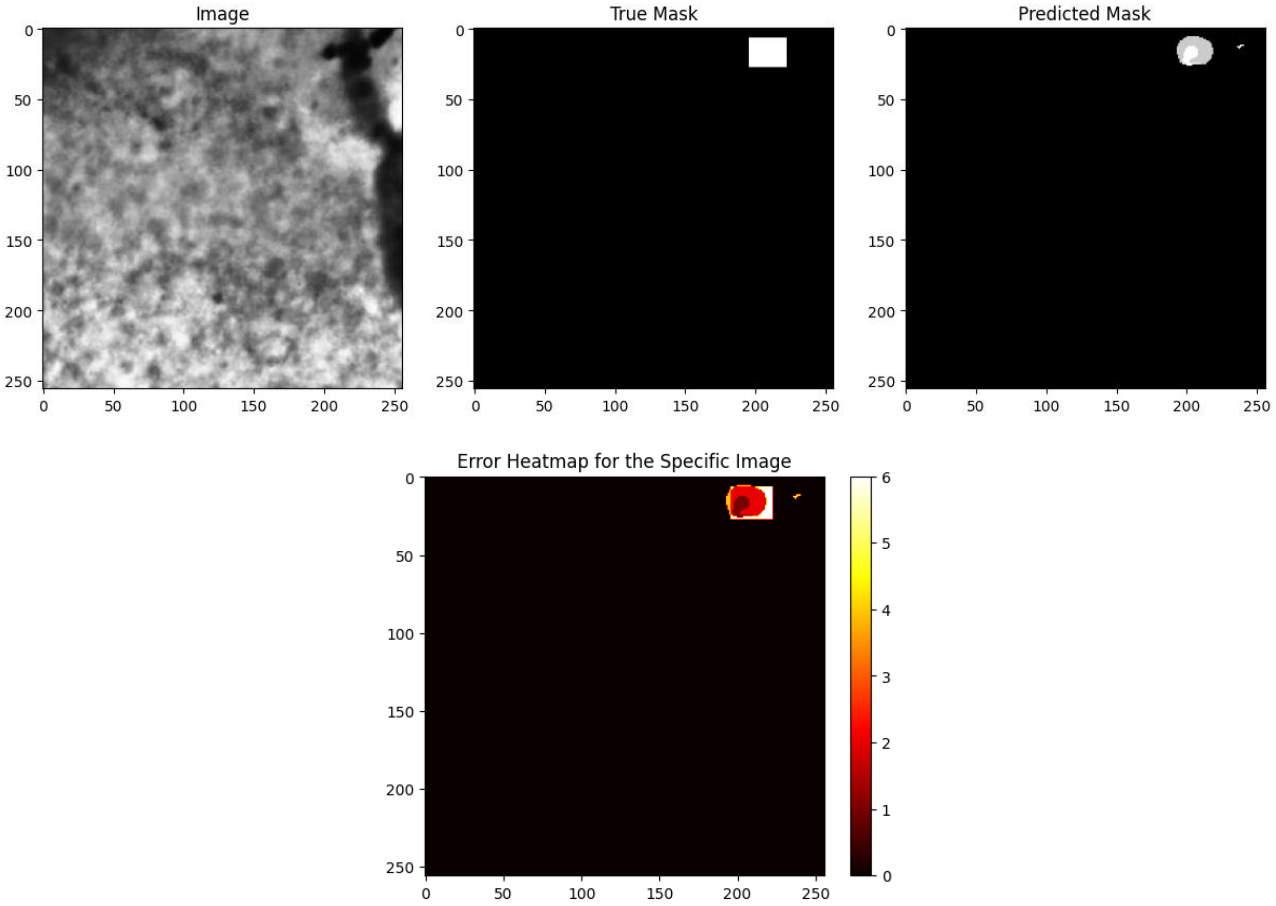


Ilustración 18 - Comparativa de las predicciones verdaderas y las del modelo con el mapa de calor de errores. Fuente: (Visual Studio Code, María González, 2023)

Como se puede ver en la ilustración 18, se muestra la imagen real, la máscara verdadera y la máscara predicha. En el mapa de calor los valores más cercanos al negro que es 0, serían los valores reales y que coinciden de forma correcta con la máscara de capa verdadera, como se puede observar en la ilustración tiene un porcentaje de exactitud muy alto.

4. Evaluación de los resultados

Por último, se evalúa el rendimiento del modelo con datos completamente nuevos para obtener una estimación imparcial de su eficiencia. Los datos que se utilizan son un conjunto de prueba que tiene que estar etiquetado y tener el mismo formato con el que se entrenó y se validó el modelo, pero con datos nuevos y que no se hayan utilizado en ninguna de las dos fases.

Como resultado se obtiene el rendimiento del conjunto de prueba y se considera si ha tenido un buen desempeño del modelo para situaciones del mondo real.

Lo primero es utilizar la misma función de métrica IoU que se utilizó en el entrenamiento (Código 8), para que se reconozcan las mismas dimensiones y parámetros guardados en el entrenamiento.

```
# ------ IMPORTACION DEL MODELO CARGADO ------
model_path = "unet_model.h5"
model = load_model(model_path, custom_objects={'iou': iou})

# ------ CARGAR LAS IMAGENES Y MASCARAS DE CAPA ------
X_test = np.load('processed_images_resultados.npy')
y_test = np.load('processed_masks_resultados.npy')

# Agrega las dimensiones de canales
X_test = X_test.reshape((90, 256, 256, 1))
y_test = y_test.reshape((90, 256, 256, 1))

# Verifica las dimensiones de los datos cargados
print(f"Dimensiones de X_test: {X_test.shape}")
print(f"Dimensiones de y_test: {y_test.shape}")
```

```
Dimensiones de X_test: (90, 256, 256, 1)
Dimensiones de y_test: (90, 256, 256, 1)
```

Código 16 - Importación del modelo entrando y un conjunto de 90 imágenes y máscaras etiquetadas no utilizadas en el entrenamiento. Fuente: (Visual Studio Code, María González, 2023)

Después se cargará el modelo U-Net (Código 16) ya entrenado, las nuevas imágenes y máscaras de capa, como son en escala de grises hay que añadir una dimensión y se verifica que tienen el mismo formato de entrada de datos que tenía el modelo.

```
# ------ EVALUACION DEL MODELO ------
test_loss, test_accuracy, test_iou = model.evaluate(X_test, y_test)
print("Perdida en Test:", test_loss)
print("Precision en Test:", test_accuracy)
print("IoU en Test:", test_iou)
```

```
3/3 [==============================] - 20s 4s/step - loss: 0.0277 - accuracy: 0.9930 - iou: 0.0561
Perdida en Test: 0.02765035070478916
Precision en Test: 0.9929921627044678
IoU en Test: 0.0561445951461792
```

Código 17 - Evaluación y outputs del modelo entrenado. Fuente: (Visual Studio Code, María González, 2023)

Se evalúa el modelo (Código 17). Primero se evalúa la pérdida, precisión y métrica de IoU. Los valores obtenidos son bastantes buenos y se imprime los resultados de estos valores.

```
# ------ VISUALIZACION DE PREDICCIONES ------
n_images = 10  # Número de imágenes a mostrar

# Definir colores para la leyenda
patch1 = mpatches.Patch(color='gray', label='Imagen Original')
patch2 = mpatches.Patch(color='yellow', label='Bajo Valor de Predicción (e.g., Fondo)')
patch3 = mpatches.Patch(color='lightblue', label='Valor Medio de Predicción')
patch4 = mpatches.Patch(color='purple', label='Alto Valor de Predicción (e.g., Espinas)')

fig, axes = plt.subplots(n_images, 2, figsize=(10, 5*n_images))  # 2 columnas

for i in range(n_images):
    # Mostrar imagen original superpuesta con predicción
    axes[i, 0].imshow(X_test[i, :, :, 0], cmap='gray')
    axes[i, 0].imshow(predictions[i, :, :, 0], alpha=0.5)
    axes[i, 0].set_title('Original + Predicción')
    axes[i, 0].axis('off')

    # Mostrar imagen original superpuesta con máscara verdadera
    axes[i, 1].imshow(X_test[i, :, :, 0], cmap='gray')
    axes[i, 1].imshow(y_test[i, :, :, 0], alpha=0.5)
    axes[i, 1].set_title('Original + Máscara Verdadera')
    axes[i, 1].axis('off')

# Colocar la leyenda en el último subplot
axes[-1, -1].legend(handles=[patch1, patch2, patch3, patch4], loc='lower right')
plt.tight_layout()
plt.show()
```

Código 18 - Ajuste de los parámetros para la visualización de predicciones del modelo entrenado. Fuente: (Visual Studio Code, María González, 2023)

También se muestran las predicciones (código 18), para ello toma una variable que almacena el número de imágenes que se quieren mostrar, en este caso son 10. A continuación, se genera una leyenda para entender los colores que se van a mostrar de una forma más clara y se muestran los resultados en dos columnas; en la primera columna estará la imagen original con las predicciones superpuestas y en la segunda columna la imagen original con la máscara verdadera superpuestas para poder hacer una comparación visual más intuitiva.

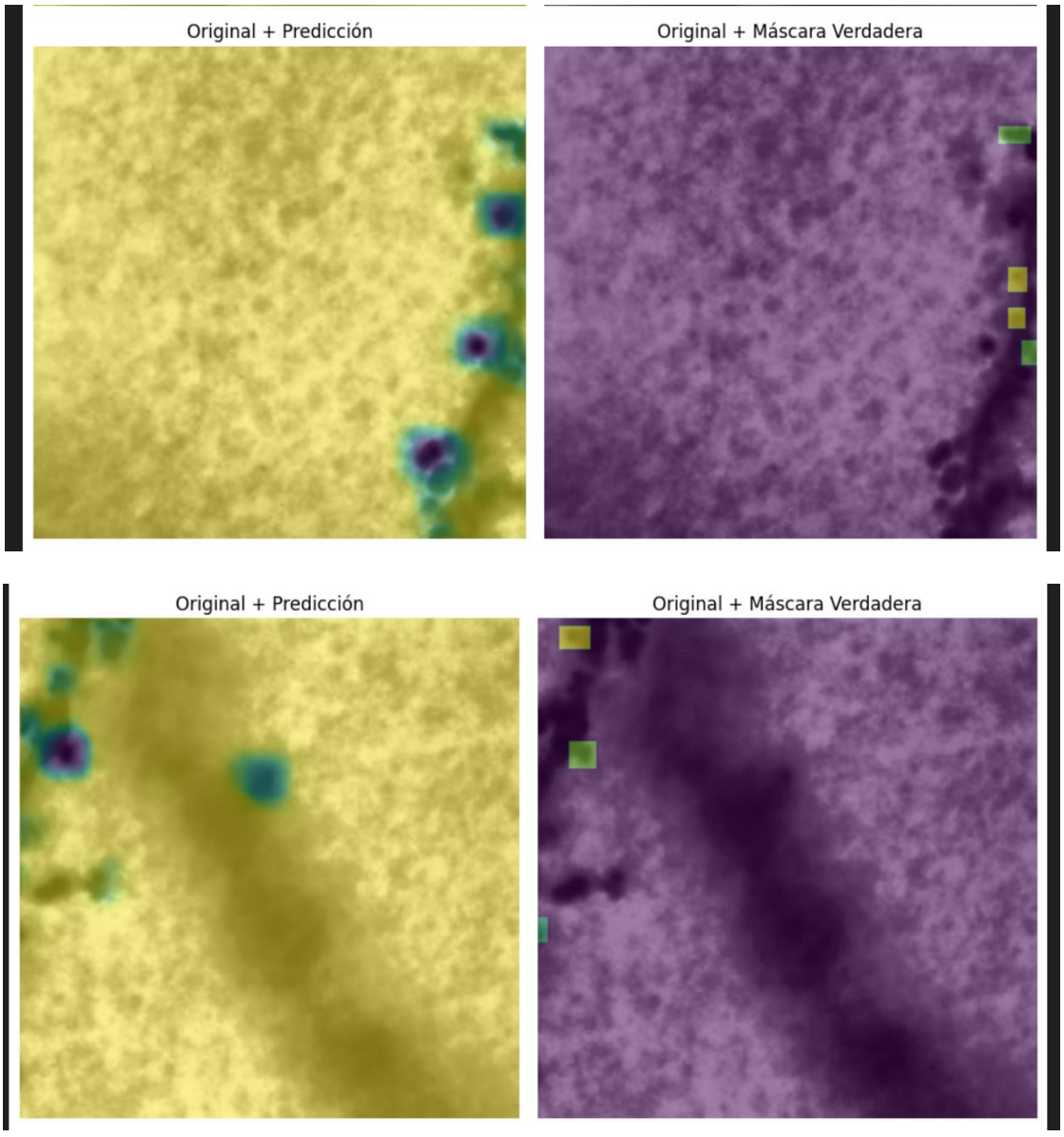


Ilustración 19 - Visualización comparativa de la predicción de espinas dendríticas y las máscaras reales del modelo entrenado, parte 1 de 2. Fuente: (Visual Studio Code, María González, 2023)

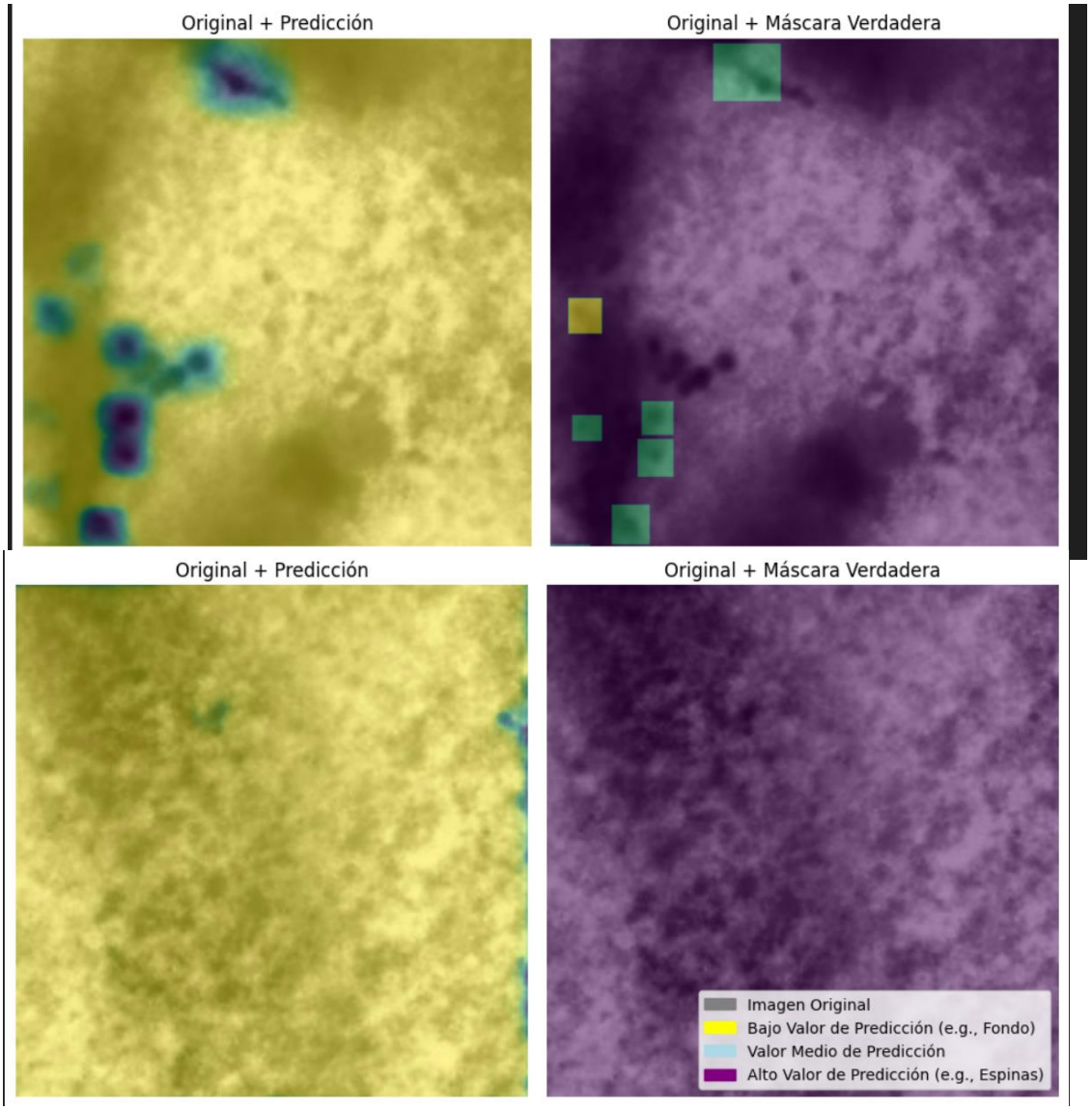


Ilustración 20 - Visualización comparativa de la predicción de espinas dendríticas y las máscaras reales del modelo entrenado, parte 2 de 2. Fuente: (Visual Studio Code, María González, 2023)

Las ilustraciones 19 y 20 muestran con gran exactitud referenciados con los diferentes colores, lo que es fondo, de una posible espina dendrítica y una alta posibilidad de espina dendrítica.

5. Conclusiones

Este proyecto ha sido una profunda exploración técnica y académica en la segmentación automática de espinas dendríticas mediante la aplicación del modelo U-Net. Esta elección metodológica ha sido el resultado de un meticuloso análisis comparativo de diversas herramientas y softwares contemporáneos, ponderando sus respectivas fortalezas y limitaciones, y considerando la propuesta innovadora que U-Net podría representar.

Tras revisar diversos modelos de aprendizaje previamente utilizados en investigaciones similares, observé que muchos de ellos, a pesar de sus logros en determinados ámbitos, no se adaptaban óptimamente a la especificidad y complejidad intrínseca de la segmentación dendrítica. Esta travesía no solo me ha proporcionado las habilidades necesarias en la segmentación de imágenes biomédicas, sino también un enriquecimiento en la comprensión biológica de las neuronas, las conexiones dendríticas y su crucial rol en patologías neurodegenerativas.

Desafíos

Uno de los primeros obstáculos técnicos han sido la laboriosa tarea de etiquetado de grandes conjuntos de imágenes y la selección del formato adecuado para su exportación, donde la precisión era esencial. Posteriormente, al abordar la implementación de la arquitectura del modelo, emergieron desafíos relacionados con la optimización y equilibrio del sistema: era imperativo evitar la generación de un modelo excesivamente profundo para prevenir el sobreajuste y, al mismo tiempo, manejar la alta demanda computacional.

Estas circunstancias me han llevaron a investigar y desarrollar estrategias para equilibrar los parámetros y garantizar que el modelo capturara de forma efectiva las características idiosincráticas de las imágenes.

Logros

El propósito subyacente a lo largo de este proyecto ha sido adentrarme en la especialidad y descubrir sus posibilidades.

Mediante iteraciones y ajustes metodológicos, he logrado confeccionar un modelo con un equilibrio técnico y funcional que satisface su cometido primordial.

Valoración personal

Este proyecto se ha manifestado como una secuencia continua de retos técnicos y académicos.

A través de estos desafíos, he consolidado no solo un repertorio de habilidades técnicas avanzadas, sino también una profunda apreciación por la sofisticación del cerebro y cómo, equipados con la tecnología y conocimiento adecuados, podemos desarrollar herramientas de análisis vanguardistas.

La adopción del modelo U-Net, más allá de una elección técnica, ha resultado ser una piedra angular en mi formación, profundizando mi entendimiento sobre adaptabilidad, optimización y evaluación en contextos reales de investigación.

Cada desafío enfrentado ha fortalecido mi resiliencia y ha cimentado confianza para abordar futuras investigaciones.

5.1. Líneas futuras

La tecnología está en constante evolución y la investigación científica como se observa al inicio de este proyecto tiene una gran trayectoria, abriendo continuamente nuevas formas de análisis y compresión de las espinas dendríticas.

Dentro del tema de la neurociencia y la computación avanzada, se pueden pensar en algunas direcciones prometedoras que puedan potenciar y expandir el alcance del proyecto.

1. Integración con plataformas de microscopía

   La integración directa entre las plataformas de microscopía en tiempo real propone una correlación entre la detección y el análisis inmediato de espinas dendríticas. Es decir, en vez de tener que hacer extensos y tediosos periodos de post-procesamiento y análisis, los investigadores podrían acceder a esa retroalimentación de forma casi inmediata. Garantizando una mejor optimización del tiempo.

2. Análisis temporal

   Las espinas dendríticas cuentan con una gran complejidad al ser estructuras dinámicas y con gran plasticidad sufren transformaciones a lo largo del tiempo.

   Utilizando un enfoque de análisis temporal, se podría rastrear estas espinas durante periodos extendidos como meses o años, pudiendo ofrecer visiones de como se van adaptando y la evolución neuronal.

   Al observar estos cambios, se podría encontrar una correlación entre transformación morfológicas con eventos o estímulos determinados, ofreciendo una mejor comprensión de las funciones neuronales. También con esta perspectiva temporal se podría observar como las neuronas forman y pierden conexiones.

3. Interpretabilidad del mercado

Una parte fundamental de cualquier proceso es entender por qué y cómo los modelos toman decisiones. Utilizando técnicas de **LIME** o **SHAP**, se podría descomponer las decisiones del modelo y determinar qué áreas son las que se tienen más en cuenta para un veredicto u otro.

Estas perspectivas no sólo ayudarían a garantizar la confiabilidad y transparencia del modelo, sino que también podría revelar patrones morfológicos o características dendríticas hasta el momento no conocidas, esenciales para la función o la neurodegeneración neuronal.

6. Bibliografía

(RSNA), R. S., & (ACR), A. C. (s.f.). *Magnetoencefalografía*. Recuperado el 8 de enero de 2023, de Radiologyinfo.org: https://www.radiologyinfo.org/es/info/meg

Adesanya, I. (2021). *UNet architecture breakdown*. Recuperado el 12 de abril de 2023, de Kaggle.com: https://www.kaggle.com/code/prvnkmr/unet-architecture-breakdown

Alberca, A. S. (2022). *La librería Numpy*. Recuperado el 25 de julio de 2023, de Aprende con Alf: https://aprendeconalf.es/docencia/python/manual/numpy/#:~:text=NumPy%20es%20una%20librer%C3%ADa%20de,un%20gran%20volumen%20de%20datos.

Alzheimers. (s.f.). *¿Qué es la demencia frontotemporal?* Recuperado el 21 de agosto de 2023, de Alzheimers.gov: https://www.alzheimers.gov/es/alzheimer-demencias/demencia-frontotemporal

Brain basics: The life and death of a neuron. (20 de marzo de 2023). Recuperado el 22 de marzo de 2023, de National Institute of Neurological Disorders and Stroke: https://www.ninds.nih.gov/health-information/public-education/brain-basics/brain-basics-life-and-death-neuron

Center Director. (19 de marzo de 2015). *Brain architecture.* Recuperado el 20 de marzo de 2023, de Center on the Developing Child at Harvard University: https://developingchild.harvard.edu/science/key-concepts/brain-architecture/

Cireşan, D. C., Giusti, A., Gambardella, L. M., & Schmidhuber, J. (2012). Deep neural networks segment neuronal membranes in electron microscopy images. *NeurIPS*, 9. Recuperado el 17 de febrero de 2023

Courville, I. G. (2016). *Deep Learning.* MIT Press. Recuperado el 27 de agosto de 2023

Cvat.ai. (s.f.). *Documentación*. Recuperado el 18 de julio de 2023, de CVAT.AI: https://opencv.github.io/cvat/docs/

Data Science Team. (2020). *Interpretación de su modelo de aprendizaje profundo por SHAP — Aprendizaje automático —*. Recuperado el 22 de agosto de 2023, de DATA SCIENCE: https://datascience.eu/es/aprendizaje-automatico/interpretacion-de-su-modelo-de-aprendizaje-profundo-por-shap/#:~:text=Qu%C3%A9%20es%20SHAP%3F,ayuda%20de%20los%20valores%20Shapely.

Data Science Team. (2021). *Función de activación Relu — Aprendizaje automático —*. Recuperado el 20 de agosto de 2023, de DATA SCIENCE: https://datascience.eu/es/aprendizaje-automatico/funcion-de-activacion-relu/

DataScientest. (2021). *Convolutional Neural Network : definición y funcionamiento*. Recuperado el 8 de agosto de 2023, de Formation Data Science | DataScientest.com: https://datascientest.com/es/convolutional-neural-network-es#:~:text=El%20Max%2DPooling%20es%20un,reduciendo%20su%20tama%C3%B1o.

datascientest. (2022). *Scikit-Learn : Descubre la biblioteca de Python dedicada al Machine Learning*. Recuperado el 25 de julio de 2023, de Formation Data Science | DataScientest.com: https://datascientest.com/es/scikit-learn-decubre-la-biblioteca-python#:~:text=Es%20una%20biblioteca%20de%20Python,una%20API%20propia%20y%20estandarizada.

DataScientest. (2022). *U-NET: todo lo que tienes que saber sobre la red neuronal de Computer Vision*. Recuperado el 10 de julio de 2023, de Formation Data Science | DataScientest.com: https://datascientest.com/es/u-net-lo-que-tienes-que-saber

DataScientest. (2023). *Matplotlib: todo lo que tienes que saber sobre la librería Python de Dataviz*. Recuperado el 25 de julio de 2023, de Formation Data Science | DataScientest.com: https://datascientest.com/es/todo-sobre-matplotlib#:~:text=%C2%BFQu%C3%A9%20es%20PyPlot%3F,los%20ejes%20de%20un%20gr%C3%A1fico.

Del Ser Lorente, J. (14 de noviembre de 2019). *Explicabilidad e inteligencia artificial*. Recuperado el 22 de agosto de 2023, de Tecnalia: https://www.tecnalia.com/blog/explicabilidad-inteligencia-artificial

IBM. (s.f.). *¿Qué son las redes neuronales convolucionales?* Recuperado el 21 de agosto de 2023, de ibm.com: https://www.ibm.com/es-es/topics/convolutional-neural-networks

ImageJ. (2014). *Introduction, Fiji/Image*. Recuperado el 8 de enero de 2023, de Imagej.net: https://imagej.net/learn/

Junquera, R. (s.f.). *Dendrita de la neurona*. Recuperado el 12 de febrero de 2023, de fisioterapia-online: https://www.fisioterapia-online.com/glosario/dendrita-de-la-neurona

KeepCoding. (2022). *¿Qué es el clustering o agrupamiento en machine learning?* Recuperado el 21 de agosto de 2023, de https://keepcoding.io/: https://keepcoding.io/blog/que-es-clustering-o-agrupamiento/#:~:text=El%20clustering%20o%20agrupamiento%20en%20machine%20learning%20es%20una%20t%C3%A9cnica,CDs%20que%20tienes%20que%20clasificar.

Kipuna. (2023). *Detector de bordes Canny con Python y OpenCV*. Recuperado el 21 de agosto de 2023, de Kipuna Ec: https://kipunaec.com/detector-de-bordes-canny-con-python-y-opencv/

Martins, J. (12 de septiembre de 2022). *Diagrama de Gantt: qué es y cómo crear uno con ejemplos.* Recuperado el 28 de agosto de 2023, de Asana: https://asana.com/es/resources/gantt-chart-basics

MedlinePlus. (s.f.). *Esclerosis lateral amiotrófica (ELA).* Recuperado el 22 de agosto de 2023, de MedlinePlus: https://medlineplus.gov/spanish/ency/article/000688.htm#:~:text=La%20esclerosis%20lateral%20amiotr%C3%B3fica%20o,movimiento%20de%20los%20m%C3%BAsculos%20voluntarios.

miro.com. (s.f.). *Creador de diagramas de Gantt online*. Recuperado el 8 de agosto de 2023, de miro.com: https://miro.com/es/planificacion-estrategica/diagrama-de-gantt/

Mskcc. (2020). Imagen por resonancia magnética funcional (fMRI). *Memorial Sloan Kettering Cancer Center*, 1. Recuperado el 21 de enero de 2023, de Memorial Sloan Kettering Cancer Center.

NeuroPoly. (s.f.). *software — NeuroPoly documentation*. Recuperado el 21 de enero de 2023, de Polymtl.ca: http://neuro.polymtl.ca/software.html

Oxford Instruments. (s.f.). *Neuroscience research - microscopy & imaging solutions - Imaris*. Recuperado el 21 de enero de 2023, de Oxford Instruments: https://imaris.oxinst.com/neuroscience

PyPI. (s.f.). *imageio*. Recuperado el 25 de julio de 2023, de PyPI: https://pypi.org/project/imageio/

Rada, L., Erdil, E., OzgurArgunsah, A., Unay, D., & Cetin, M. (2014). Automatic dendritic spine detection using multiscale dot enhancement filters and SIFT features. *IEEE*, 5. Recuperado el 17 de febrero de 2023

Rada, L., Kilic, B., Erdil, E., Ramiro-Cortés, Y., Israely, I., Unay, D., . . . Argunsah, A. Ö. (2018). Tracking-assisted detection of dendritic spines in time-lapse microscopic images. *Neuroscience*, 189-205. Recuperado el 18 de feberero de 2023

Raida, V. S. (febrero de 2018). Las espinas dendríticas, su función y algunas alteraciones. *Scielo*. Recuperado el 13 de marzo de 2023, de SciELO.

Residuales, B. A. (2020). ¿Qué es microscopio de fluorescencia? *Micro Planet Bioindicacion*, 2. Recuperado el 22 de agosto de 2023, de Bioindicación Aguas Residuales.

Rodrigo, J. A. (abril de 2017). *Máquinas de Vector Soporte (Support Vector Machines, SVMs).* Recuperado el 21 de agosto de 2023, de Cienciadedatos.net: https://cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines

Rongjian Li*, T. Z. (2017). *Deep Learning Segmentation of Optical.* IEEE. Recuperado el 18 de febrero de 2023, de https://par.nsf.gov/servlets/purl/10057753

Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. Recuperado el 2 de agosto de 2023, de arXiv: http://arxiv.org/abs/1505.04597https://arxiv.org/abs/1505.04597

Ruiz, G., Muñoz Pérez, J. A., Bernal, G., & Rubio, L. (s.f.). Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial. *Revista Iberoamericana de Inteligencia Artificial*, 16. Recuperado el 21 de agosto de 2023, de Redalyc.org.

Ruszczycki, B., Szepesi, Z., Wilczynski, G. M., Bijata, M., Kalita, K., Kaczmarek, L., & Wlodarczyk, J. (25 de agosto de 2012). Sampling issues in quantitative analysis of dendritic spines morphology. *BMC bioinformatics*, 213. Recuperado el 5 de agosto de 2023

Sankesara, H. (23 de enero de 2019). *UNet.* Recuperado el 7 de agosto de 2023, de Towards Data Science: https://towardsdatascience.com/u-net-b229b32b4a71

Santos, D., Dallos, L., & Gaona-García, P. A. (20 de diciembre de 2020). Algoritmos de rastreo de movimiento utilizando técnicas de inteligencia artificial y machine learning. *Universidad Distrital Francisco José de Caldas*, 16. Recuperado el 29 de julio de 2023, de scielo.cl.

Tecnología | Uniandes. (2022). *Scikit-image - Tecnología*. Recuperado el 25 de julio de 2023, de Tecnología | Uniandes: https://tecnologia.uniandes.edu.co/scikit-image/

TensorFlow. (s.f.). *TensorFlow Core*. Recuperado el 25 de julio de 2023, de TensorFlow: https://www.tensorflow.org/overview?hl=es-419

Torres, A. (21 de abril de 2023). *Dendritas.* Recuperado el 2 de marzo de 2023, de Kenhub: https://www.kenhub.com/es/library/anatomia-es/dendritas

Uniwebsidad. (s.f.). *10.1. Módulos de sistema*. Recuperado el 25 de julio de 2023, de Uniwebsidad.com: https://uniwebsidad.com/libros/python/capitulo-10/modulos-de-sistema

Valencia Segura, R. K., Colín Barenque, L., & Fortoul van der Goes, T. I. (2018). *Las espinas dendríticas, su función y algunas alteraciones*. Recuperado el 20 de febrero de 2023, de https://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S0026-17422018000100046

Villasenor, G. (2023). *RANSAC – Definición y Explicaciones*. Recuperado el 21 de agosto de 2023, de Portal Contacto Político: https://www.contactopolitico.com.ar/%F0%9F%94%8E-ransac-definicion-y-explicaciones/

Xiao, X., Djurisic, M., Hoogi, A., Sapp, R. W., Shatz, C. J., & Rubin, D. L. (2018). Automated dendritic spine detection using convolutional neural networks on maximum intensity projected microscopic volumes. *Journal of neuroscience methods*, 25-35. Recuperado el 17 de febrero de 2023

Yang, B., Huang, J., Wu, G., & Yang, J. (2021). Classifying the tracing difficulty of 3D neuron image blocks based on deep learning. *Brain informatics*, 9. Recuperado el 29 de agosto de 2023

Yoshikura, T. (10 de agosto de 2023). *What is Brightfield microscopy?* Recuperado el 22 de agosto de 2023, de Olympus-lifescience.com: https://www.olympus-lifescience.com/es/discovery/what-is-brightfield-microscopy/

Zhang, J. (18 de octubre de 2019). *UNet — line by line explanation.* Recuperado el 8 de agosto de 2023, de Towards Data Science: https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5

Zion Market Research. (s.f.). *Global neuroscience market industry size, share, trends, analysis and forecast 2023 - 2030.* Zion Market Research. Recuperado el 25 de enero de 2023, de https://www.zionmarketresearch.com/report/neuroscience-market

Zyprian, F. (9 de marzo de 2023). *CV2 - Guía maestra OpenCV hecha para desarrolladores Python.* Recuperado el 25 de julio de 2023, de Konfuzio: https://konfuzio.com/es/cv2/#:~:text=CV2%20es%20una%20potente%20librer%C3%ADa,y%20el%20filtrado%20de%20im%C3%A1genes.