

**UNIVERSIDADE DE SANTIAGO DE
COMPOSTELA**



ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

**Identificación y recuento de
nanopartículas en imágenes de
microscopía**

Autor:

Fernando González Salas

Tutores:

Xosé Manuel Pardo López
Victor Leborán Álvarez

Grado en Ingeniería Informática

Junio 2023

Trabajo de Fin de Grado presentado en la Escola Técnica Superior de Enxeñaría de la Universidade de Santiago de Compostela para la obtención del Grado en Inegniería Informática



D. Xosé Manuel Pardo López, Profesor/a del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, y **D. Victor Leborán Álvarez**, Investigador de la Universidad de Santiago de Compostela,

INFORMAN:

Que la presente memoria, titulada *Identificación y recuento de nanopartículas en imágenes de microscopía*, presentada por **D. Fernando González Salas** para superar los créditos correspondientes al Trabajo de Fin de Grado de la titulación de Grado en Ingeniería Informática, se realizó bajo nuestra tutoría en el Departamento de Electrónica y Computación da Universidade de Santiago de Compostela.

Y para que así conste a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a 15 de Junio de 2023:

Titor/a,

Cotitor/a,

Alumno/a,

A handwritten signature in black ink, appearing to read "Fernando Gonzalez Salas". The signature is somewhat stylized and includes a small "a" at the end. Above the signature, the word "Alumno/a," is written in a smaller, straight-line font.

Xosé Manuel Pardo López Victor Leborán Álvarez Fernando Gonzalez Salas

Agradecimientos

Quisiera agradecer a todos los profesores que han colaborado para completar mi formación.

Así como todos los amigos que he conocido en el transcurso de estos cuatro años así como los que ya conocía de antes, pero especialmente a mi familia la cuál siempre me apoyó para acabar esta carrera.

Resumen

La nanotecnología es un campo científico dedicado a investigar y manipular la materia a escala nanométrica. Desde que se descubrieron las nanopartículas a mediados del siglo XX, este campo de las ciencias ha experimentado un avance exponencial en cuanto a técnicas de investigación, a pesar de este avance constante algunas de las actividades de este campo siguen utilizando métodos que requieren de una gran cantidad de recursos.

Este hecho produce una desaceleración en el avance de algunas ramas de investigación, por lo que automatizar algunas de estas actividades puede ser de gran ayuda para los investigadores y podría impulsar los avances en este campo.

Para conseguir esta aceleración, este trabajo busca investigar y desarrollar una herramienta que permita automatizar algunas de las actividades que se realizan en las investigaciones de este campo. Más concretamente este trabajo se centrará en la clasificación y conteo de nanopartículas en imágenes microscópicas.

El objetivo final es desarrollar una herramienta eficiente que permita a investigadores y gente relacionada con este campo automatizar esta actividad, permitiendo ahorrar una gran cantidad de recursos en el proceso.

Para conseguir esto, se investigaron y desarrollaron diferentes técnicas de visión por computador, procesamiento de imágenes y aprendizaje automático, con el objetivo de poder clasificar, contar y medir nanopartículas en imágenes microscópicas. Todo este desarrollo también se acompañó de una interfaz gráfica muy simple que permite a los usuarios utilizar la herramienta de una manera sencilla y cómoda.

Índice general

1. Introducción	1
1.1. Contexto y motivación del problema	1
1.2. Objetivos del trabajo	2
1.3. Análisis del problema y modelo mental del sistema	2
1.4. Estructura de la memoria	4
2. Estado del conocimiento	7
2.1. Métodos de segmentación	8
2.2. Métodos de clasificación	10
3. Materiales	13
3.1. Datasets	13
3.2. Herramientas de desarrollo	14
4. Métodos	15
4.1. Módulo de segmentación	15
4.1.1. Explicación U-Net	15
4.1.2. Implementación U-Net	16
4.1.3. Decodificador de la U-Net	17
4.1.4. Entrenamiento de las implementaciones	18
4.1.5. Limitaciones de la U-Net	20
4.2. Separación de las regiones de interés	21
4.3. Clasificador de nanopartículas	22
4.4. Medición de las nanopartículas	24
4.5. Aumento utilidad de las funcionalidades	26
5. Pruebas	27
5.1. Pruebas módulo de segmentación	27
5.2. Pruebas módulo de clasificación	30
5.3. Pruebas medición nanopartículas	31
5.4. Ejemplos individuales	32
6. Discusión de los resultados	35

7. Conclusiones y posibles ampliaciones	37
A. Manuales técnicos	39
B. Manuales de usuario	41
B.1. Interfaz gráfica	41
B.2. SegmentationModule	41
B.2.1. Definición de la clase	41
B.2.2. Funciones de la clase	42
C. Arquitectura U-Net implementada	47
Bibliografía	53

Índice de figuras

1.1.	Ejemplo de diferentes imágenes de microscopía.	3
1.2.	Modelo mental del sistema.	4
2.1.	Arquitectura del modelo U-Net	9
2.2.	Arquitecturas de redes neuronales para clasificación de imágenes.	11
4.1.	Arquitectura de la red U-Net	16
4.2.	Input, Mask y Prediction usando el modelo de Oxford	18
4.3.	Representación gráfica de IoU	19
4.4.	Imagen de entrada y Segmentación usando el modelo final	20
4.5.	Ejemplo de watershed aplicado a una imagen segmentada	21
4.6.	Proceso de separación de las regiones de interés	22
4.7.	Ejemplos de las diferentes clases de nanopartículas	24
4.8.	Histograma multiclase del tamaño de las nanopartículas	26
5.1.	Prueba de segmentación	27
5.2.	Prueba módulo segmentación	29
5.3.	Prueba opciones preprocesado y postprocesado	30
5.4.	Pruebas módulo clasificación	31
5.5.	Pruebas medición nanopartículas	32
5.6.	Histograma de nanopartículas	33
5.7.	Histograma de nanopartículas	34

X

Índice de cuadros

A.1. Librerías utilizadas en el proyecto	39
--	----

Capítulo 1

Introducción

1.1. Contexto y motivación del problema

Desde el descubrimiento de las nanopartículas, los investigadores de esta área han utilizado gran parte de su tiempo en cuantificar de manera manual el número de nanopartículas que existen en una imagen de microscopía, lo que incrementa de manera exponencial el número de recursos necesarios para esta tarea.

La primera pregunta que surge es el porqué de la necesidad de cuantificar el número de nanopartículas que existen en una imagen. La razón principal, es que los diferentes tipos de nanopartículas tienen un gran número de propiedades interesantes que pueden tener diferentes aplicaciones en una amplia variedad de campos[1], como podrían ser: **en electrónica** utilizándose para la fabricación de pantallas de alta resolución, **en medicina** usándose como agentes de contraste en técnicas de imagenología, como la resonancia magnética (RM) y la tomografía por emisión de positrones (PET) ,**en energías renovables** sirviendo para la fabricación de celdas solares de próxima generación y **en cosméticos** llegándose a utilizar en protectores solares contra los rayos UV.

Una vez que se han expuesto las diferentes utilidades de cuantificar el número de nanopartículas, tenemos que aclarar que la principal motivación de este trabajo es mejorar este proceso para los investigadores del **CIQUS** (*Centro Singular de Investigación en Química Biológica y Materiales Moleculares*), ya que hasta ahora este proceso lo realizaban de manera manual o con herramientas que no se adaptaban a sus necesidades, por lo que automatizar esta actividad podría acelerar diferentes investigaciones.

Para simplificar esta tarea, sería de gran utilidad desarrollar una herramienta que utilice diferentes técnicas de visión por computador. Estas técnicas ya han sido probadas en este ámbito[20] pero desde otro enfoque no tan centrado en la clasificación y medición.

Este nuevo enfoque nos permitiría identificar, cuantificar y medir el tamaño de las diferentes nanopartículas de la imagen con cierta precisión y de manera

semiautomática, lo que permitiría ahorrar tiempo y dejar a los investigadores más liberados de tareas tediosas ganando precisión al mismo tiempo.

Finalmente aclarar que el usuario final de este trabajo van a ser los investigadores del CIQUS, por lo que debemos facilitar el uso de este módulo a estas personas a través de una interfaz gráfica simple.

1.2. Objetivos del trabajo

Para conseguir todo lo previamente expuesto este trabajo consistirá en diferentes objetivos:

1. Revisar el estado de las diferentes técnicas que se utilizan para problemas similares, con el objetivo de seleccionar las más adecuadas para nuestro problema.
2. Obtener las regiones de interés de la imagen, a través de procesos de segmentación, de manera precisa. En nuestro caso, las regiones de interés serán las nanopartículas.
3. Clasificar las nanopartículas en diferentes clases en función de su forma geométrica.
4. Calcular el tamaño de las nanopartículas de manera individual, apoyándonos en el resultado de la segmentación.
5. Desarrollar una interfaz gráfica simple que facilite el uso de este módulo a los investigadores del CIQUS.
6. Evaluar el rendimiento del sistema.
7. Obtener un módulo software funcional que implemente las técnicas mas adecuadas de las investigadas.

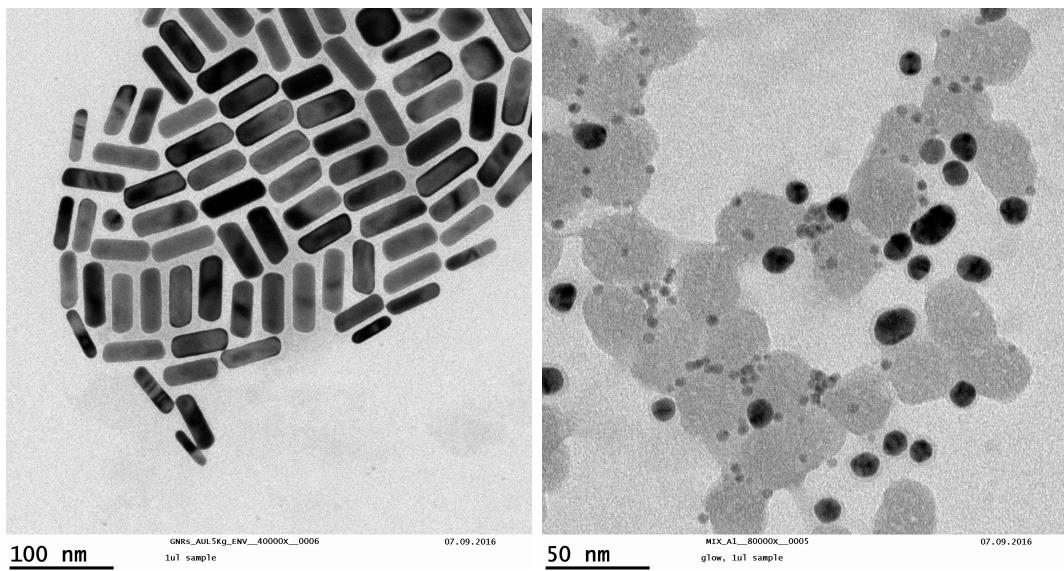
1.3. Análisis del problema y modelo mental del sistema

Una vez aclarados los objetivos principales, debemos tener en cuenta que la mayoría de las imágenes que se van a utilizar son parecidas a las imágenes de la Figura 1.1. Estas imágenes, serán normalmente codificadas en escala de grises en los siguientes formatos:

1. **Formato TEM:** TEM (Transmission Electron Microscopy) es una técnica de microscopía que utiliza un haz de electrones para obtener imágenes detalladas de muestras ultrafinas.

2. Formato SEM: SEM (Scanning Electron Microscopy) es una técnica de microscopía que utiliza un haz de electrones para generar imágenes en 3D de la superficie de una muestra.

La primera problemática se puede observar en la Figura 1.1a, donde se puede apreciar que existen diferentes tipos de nanopartículas en la misma imagen, tal y como se dijo en los objetivos del trabajo sería de gran utilidad que nuestro sistema sea capaz de clasificar las nanopartículas en diferentes clases en función de su forma geométrica.



(a) Imagen con nanopartículas multiclase.

(b) Imagen con problemáticas.

Figura 1.1: Ejemplo de diferentes imágenes de microscopía.

Otra problemática a tratar sería la de analizar nanopartículas muy próximas tal y como se ve en la Figura 1.1b, la dificultad de realizar esto de manera manual es muy elevada, pero sería de gran utilidad que nuestra herramienta sea capaz de solventar este problema, aunque sea de manera parcial. También se puede apreciar que el fondo de la imagen es muy parecido al de algunos elementos de esta, lo que dificulta la tarea de identificarlas y que algunas nanopartículas tienen bordes irregulares.

En todas las etapas de este proceso se usarán técnicas de **vision por computador** y **deep learning**. Donde en la primera etapa la imagen pasa por un proceso de **segmentación**, en el cual se identifiquen las diferentes regiones significativas de la imagen, en nuestro caso estas regiones serán el fondo y las nanopartículas. De este modo obtendremos los datos que se usarán para la etapa de **clasificación**, que nos permitirá clasificar de manera sistemática e individual las regiones de interés.

Finalizado este paso, la idea es realizar una medida del tamaño de cada región segmentada de la imagen, con el objetivo de mostrarle al usuario diferentes estadísticas con respecto a las nanopartículas localizadas.

Con el objetivo de conocer la precisión del sistema, obtendremos métricas que nos indiquen el número de partículas que este haya sido capaz de clasificar de manera correcta, pero la precisión de cada uno de los módulos que componen el sistema se medirá de manera individual con diferentes medidas.

Una vez descrita la motivación del trabajo y la forma de encontrar una solución, el proceso de tratamiento de una imagen descrito anteriormente será similar al mostrado en la Figura 1.2.

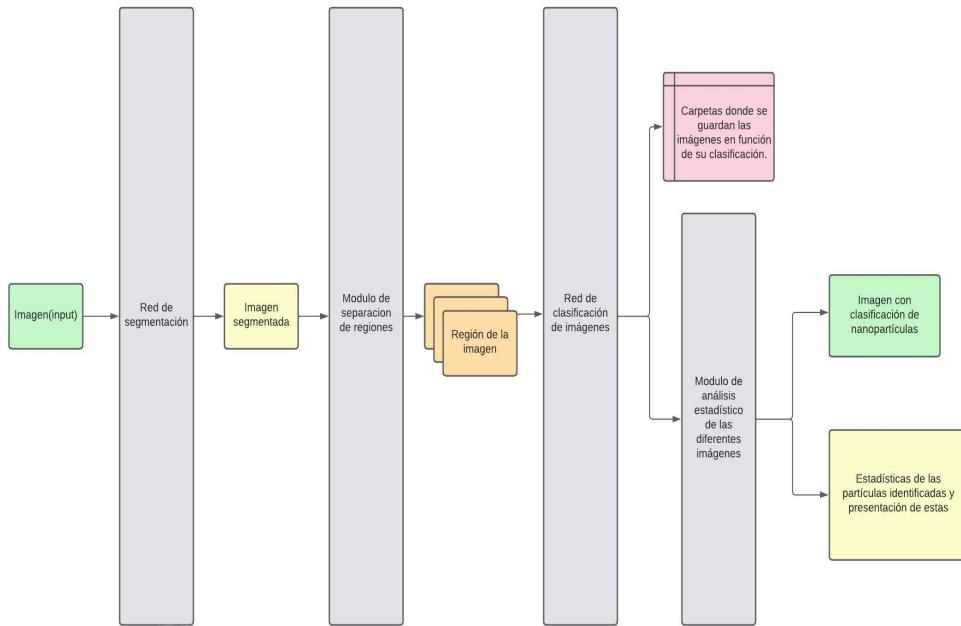


Figura 1.2: Modelo mental del sistema.

1.4. Estructura de la memoria

Este *Trabajo de Fin de Grado* se puede ubicar dentro del **Tipo A** según el *Regulamento do Traballo de Fin de Grao* del Grado en Ingeniería Informática de la USC. Es decir, se desarrolla una idea de sistema informático buscando contribuir a las técnicas de la informática existentes.

Una vez descrito el contexto del trabajo, a partir de esta Introducción que representa el **Capítulo 1**, el resto de la memoria se estructura de la siguiente manera:

- En el **Capítulo 2** se centra en mostrar el **estado del arte** de las técnicas que se utilizan en problemas similares al nuestro. Presentando con más detalles los problemas abordados así como las técnicas.
- En el **Capítulo 3** se describen los materiales utilizados para la realización del trabajo, así como las herramientas utilizadas.
- En el **Capítulo 4** se describe la metodología que se ha seguido para la realización del trabajo, así como las herramientas utilizadas.
- En el **Capítulo 5** se muestran los resultados obtenidos en cada una de las etapas del módulo software, estos resultados son cuantificados con diferentes métricas.
- En el **Capítulo 6** se muestra una discusión de los resultados obtenidos en las pruebas.
- En el **Capítulo 7** se muestran las conclusiones obtenidas a lo largo del trabajo, así como proponer posibles ampliaciones y mejoras.

Capítulo 2

Estado del conocimiento

En este trabajo, como ya se indicó en la Introducción, se pretende desarrollar una herramienta que sea capaz de identificar, localizar, clasificar y medir el tamaño de las partículas de una imagen de microscopía de manera automática. Para el problema de la identificación y clasificación para objetos de tipo general existen diferentes soluciones basadas en inteligencia artificial, basadas en redes convolucionales que permiten llevar a cabo ambas tareas al mismo tiempo, se pueden destacar diferentes tipos de modelos:

- **SSD (Single Shot Detector)**[4]: SSD utiliza una red neuronal convolucional para extraer características de la imagen y luego utiliza múltiples capas de detección a diferentes escalas y resoluciones para detectar objetos de diferentes tamaños. Estas capas de detección, llamadas "heads", están conectadas a la red convolucional base y predicen simultáneamente las clases de objetos y las coordenadas de las cajas delimitadoras asociadas.
- **R-CNN (Region Based Convolutional Neural Networks)**[5]: Es una arquitectura de detección de objetos que consta de tres etapas principales: propuesta de regiones, extracción de características y clasificación. En primer lugar, se generan propuestas de regiones que podrían contener objetos utilizando algoritmos de búsqueda selectiva. Luego, se extraen características de cada región utilizando una red neuronal convolucional (CNN). Finalmente, se clasifica cada región utilizando un clasificador separado.
- **YOLO (You Only Look Once)**[6]: Es una arquitectura de detección de objetos que se basa en una única pasada a través de una red neuronal convolucional. En lugar de generar propuestas de regiones, YOLO divide la imagen en una cuadrícula y predice simultáneamente las cajas delimitadoras y las clases de objetos dentro de cada celda de la cuadrícula. Esto permite una detección de objetos en tiempo real, pero puede tener dificultades para detectar objetos pequeños o muy superpuestos.

A pesar de que estos modelos simplificarían la tarea de detección y clasificación de las partículas, no podemos considerarlos viables para nuestro problema debido a que son modelos muy pesados y requieren de un dataset de entrenamiento muy grande para poder obtener resultados aceptables.

Esta limitación en el tamaño del dataset de entrenamiento es un problema muy importante, ya que en el caso de las imágenes de microscopía, los datasets son muy escasos y la mayoría de los que existen no son de acceso público.

Por lo tanto, se ha decidido resolver de manera separada los problemas de segmentación y clasificación de las partículas, ya que de esta manera existen más posibilidades de obtener buenos resultados con modelos más ligeros y que no requieran una gran cantidad de datos.

2.1. Métodos de segmentación

A la hora de segmentar imágenes, disponemos de una mayor variedad de métodos a pesar de que sigamos teniendo el problema de la escasez de datasets de entrenamiento. Algunos de los métodos más utilizados en la actualidad son los siguientes:

- **Métodos clásicos de segmentación**[2]: Se refiere a técnicas y algoritmos tradicionales utilizados para dividir una imagen en regiones o componentes más pequeños. Estos métodos generalmente se basan en propiedades de los píxeles de la imagen, como su intensidad, textura o gradiente, para determinar los límites o áreas de interés en la imagen. Entre estas técnicas destacan algunas como la **umbralización**, la **detección de bordes** o el **filtrado y segmentación basada en regiones**.
- **Métodos basados en deep learning**[7]: Estas técnicas utilizan redes neuronales profundas para aprender características y patrones automáticamente, lo que permite obtener segmentaciones precisas y de alta calidad. Los métodos de segmentación basados en deep learning pueden beneficiarse de arquitecturas especializadas, como **U-Net**[9] o **Mask R-CNN**[8], que están diseñadas específicamente para abordar la tarea de segmentación.

Conociendo el contexto de nuestro problema, en el que usaremos imágenes con formato STEM[3], el modelo de segmentación que mejor se adapta a nuestras necesidades es el de U-Net, ya que está diseñado para segmentar imágenes en el ámbito de la biomedicina y microscópicas. Otro de los motivos por los que se ha elegido este modelo es que es un modelo muy ligero, ya que tiene menos de 1 millón de parámetros entrenables, lo que nos permite entrenarlo con un dataset de entrenamiento pequeño.

La arquitectura de U-Net se basa en un diseño en forma de U, que consta de dos partes principales: el camino de codificación (downsampling) y el camino de decodificación (upsampling).

El camino de codificación se encarga de extraer características relevantes de la imagen y reducir su tamaño. Consiste en una serie de capas de convolución seguidas de capas de agrupación (pooling) que disminuyen la resolución espacial pero aumentan la profundidad de características. Esto permite capturar características de alto nivel y contextuales en la imagen.

El camino de decodificación se encarga de reconstruir la segmentación detallada utilizando la información contextual y las características extraídas en el camino de codificación. Está compuesto por capas de convolución transpuesta o de aumento (upsampling) que incrementan la resolución espacial de las características. Además, se fusionan características del camino de codificación correspondientes a la misma escala espacial mediante conexiones de salto (skip connections), que permiten combinar información de distintos niveles de abstracción y conservar detalles finos en la segmentación final.

En la parte final de la red, se utiliza una capa de convolución con una función de activación apropiada, como la función sigmoide, para generar una máscara de segmentación binaria. Esta máscara asigna a cada píxel de la imagen un valor entre 0 y 1, donde los valores cercanos a 1 representan la presencia del objeto segmentado y los valores cercanos a 0 representan el fondo.

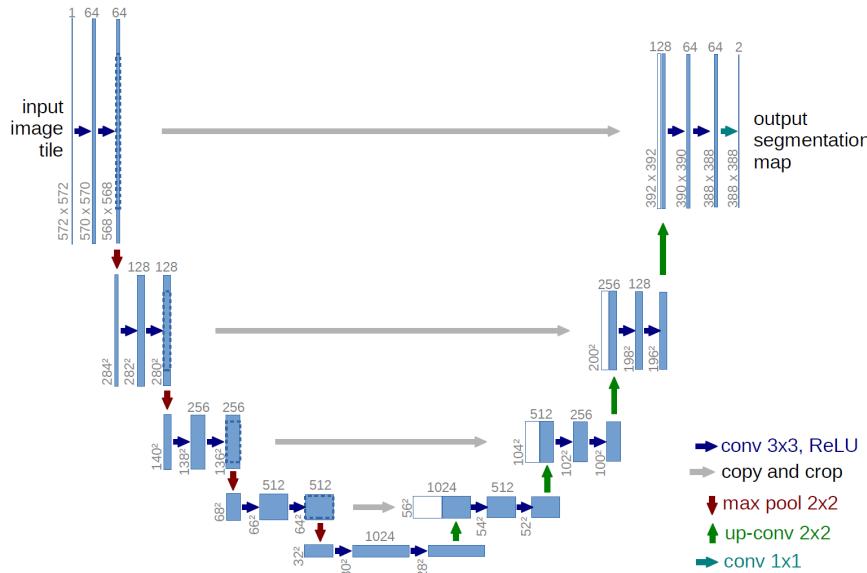


Figura 2.1: Arquitectura del modelo U-Net

A las imágenes segmentadas obtenidas por este modelo, se le aplicará un postprocesado para mejorar los resultados obtenidos, ya que el modelo U-Net no es capaz de segmentar correctamente las partículas que se encuentran muy juntas entre sí. Este postprocesado utilizará algunos principios de los métodos clásicos de segmentación, como la umbralización.

2.2. Métodos de clasificación

Una vez que se obtienen los cuadros delimitadores (bounding boxes) con máscaras de cada partícula, es necesario clasificarlas en función de su forma. Para abordar este problema en la actualidad, se suelen utilizar diferentes modelos que pueden cumplir con esta función con un alto porcentaje de acierto, entre los que destacan los siguientes:

- **Redes neuronales convolucionales**[10]: Las redes neuronales convolucionales (CNN) son un tipo de red neuronal artificial que se utiliza principalmente para procesamiento de imágenes y reconocimiento de patrones. Normalmente están compuestas por capas de convolución que extraen características de la imagen, capas de agrupación que reducen el tamaño de la imagen y capas totalmente conectadas que se encargan de realizar las predicciones ,el número de estas capas puede tener una gran influencia en el resultado final[12], un ejemplo de este tipo de red sería el modelo **VGG-16**[11] que se puede ver en la figura 2.2a.
- **Redes densamente conectadas**: Las redes neuronales densamente conectadas (DNN) son un tipo de red neuronal artificial en la que cada neurona de una capa está conectada a todas las neuronas de la capa anterior. Un ejemplo de este tipo de red[22] podría ser el modelo denso usado para clasificar el dataset MNIST que se puede ver en la figura 2.2b.

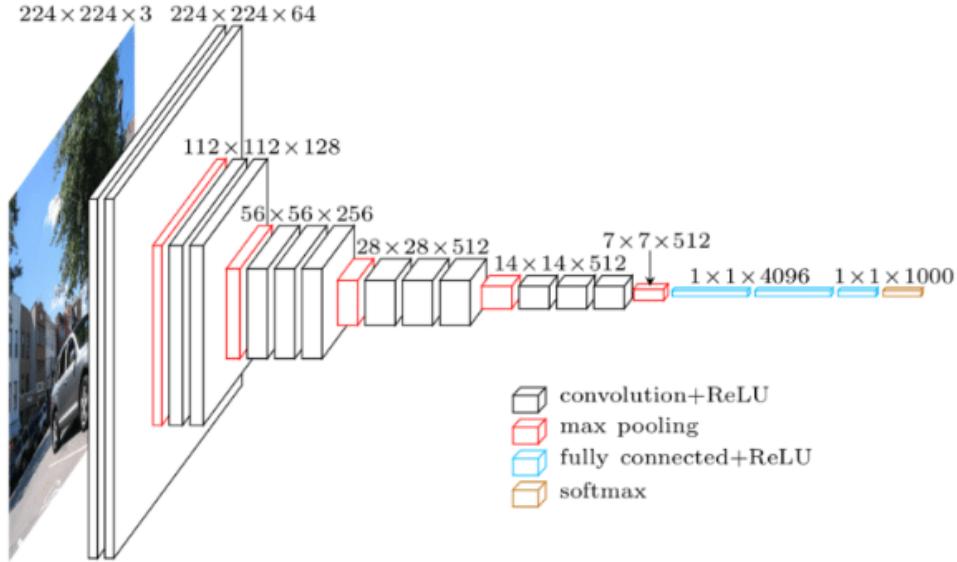
Las redes convolucionales (CNN) son ventajosas para la clasificación de imágenes debido a su capacidad para extraer automáticamente características relevantes y su eficiencia computacional. Pueden identificar patrones visuales sin necesidad de indicar explícitamente qué buscar y son más rápidas y fáciles de entrenar que las redes densamente conectadas. Sin embargo, requieren grandes cantidades de datos de entrenamiento y pueden ser difíciles de interpretar debido a su complejidad. Este tipo de redes ya han sido probadas en el campo de la nanotecnología[21] y han demostrado que pueden alcanzar una precisión en la clasificación muy alta.

Las redes densamente conectadas (DNN) son flexibles y pueden adaptarse a diversos problemas de clasificación. Permiten una interpretación más clara de los resultados, pero requieren una mayor capacidad computacional y tiempo de entrenamiento. También son sensibles a las traslaciones y variaciones de escala en las imágenes, a menos que se utilicen técnicas adicionales. En este tipo de redes la precisión viene normalmente influenciada por la cantidad de capas que tenga la red y por la cantidad de neuronas que tenga cada capa, aunque no necesariamente mejorará la precisión de la red al aumentar estos parámetros por lo que encontrar el número óptimo de capas y neuronas es un proceso de prueba y error que puede consumir bastante tiempo. Como en el caso anterior, este tipo de redes ya han

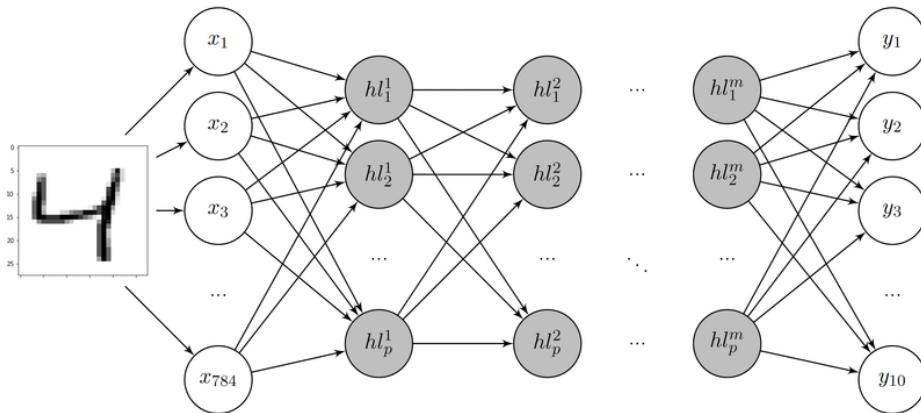
sido probadas en el campo de la nanotecnología[23] y han demostrado que pueden alcanzar una precisión en la clasificación satisfactoria.

Estos dos conjuntos de modelos son los más utilizados en la actualidad para la clasificación de imágenes, pero en este trabajo se ha decidido utilizar las redes densamente conectadas ya que son más sencillas de implementar y requieren de menos recursos computacionales para su entrenamiento.

Además, en este problema las diferentes nanopartículas se van a clasificar en función de su forma geométrica, por lo que no es necesario utilizar modelos tan complejos como las CNN.



(a) Arquitectura de la red VGG16 [11].



(b) Arquitectura de una red DNN entrenada con los datos del MNIST[22].

Figura 2.2: Arquitecturas de redes neuronales para clasificación de imágenes.

Capítulo 3

Materiales

3.1. Datasets

Con el objetivo de desarrollar este proyecto fue necesario realizar diferentes entrenamientos de los diferentes modelos. Debido a esto, ha sido necesario utilizar diferentes datasets entro los que se incluyen:

- Dataset PETS: Dataset creado por la Universidad de Oxford [13]¹ que contiene una gran variedad de imágenes de 37 categorías diferentes. Este dataset se caracteriza por tener una gran cantidad de imágenes con diferentes condiciones de iluminación y diferentes fondos. Las imágenes se caracterizan por tener un ground truth que incluye máscaras de segmentación lo que permite entrenar modelos neuronales con este dataset.
- Dataset Nanoparticles Github: Dataset creado por el usuario de Github *anilgurses*² que contiene imágenes de nanopartículas segmentadas en dos clases. Este dataset es más limitado que el anterior pero contiene imágenes que pertenecen al contexto de nuestro problema, lo que permite entrenar modelos neuronales para la segmentación de nanopartículas.
- Dataset MNIST [14]³: Dataset que contiene imágenes de dígitos escritos a mano, este dataset es muy utilizado para el aprendizaje de redes neuronales. Este dataset a pesar de que ya es muy usado permite entrenar modelos neuronales para la clasificación de números. Lo que puede usarse para detectar las escalas automáticamente.
- Dataset CIQUS: Imágenes de nanopartículas en formato TIF que se han utilizado para el entrenamiento de los modelos neuronales de clasificación. Estas imágenes son un conjunto de unas 100 imágenes, propiedad del CIQUS,

¹Dataset disponible en <https://www.robots.ox.ac.uk/~vgg/data/pets/>

²Dataset disponible en <https://github.com/anilgurses/TEM-Nano-Particle-Cell-Dataset>

³Dataset disponible en <http://yann.lecun.com/exdb/mnist/>

que en ellas se pueden encontrar una colección de diferentes nanopartículas que se han utilizado para el entrenamiento de estos modelos.

3.2. Herramientas de desarrollo

Hardware

Para el desarrollo de este proyecto se ha utilizado un ordenador portátil **Dell Inspiron 16 5630** el cuál tiene unas especificaciones que permitieron el desarrollo de este proyecto sin ningún problema.

Este ordenador tiene un procesadore **Intel Core i7-1360P** con una frecuencia de 1.60 GHz, 16 GB de memoria RAM, una tarjeta gráfica **Intel Iris Xe** y un almacenamiento de 1 TB SSD. Estos elementos junto con el sistema operativo **Windows 11** permitieron realizar todo el desarrollo y pruebas de manera eficiente.

Herramientas software

En cuanto al software con el que se desarrolló este proyecto, se incluyen herramientas como **Github** para el control de versiones, **Visual Studio Code** como editor de código fuente, **Jupyter Notebook** para el desarrollo de los modelos neuronales y **Docker** para la creación de contenedores que permitan ejecutar la herramienta de manera sencilla y ,finalmente, para el entrenamiento de las redes neuronales se utilizó **Google Colab** debido a la accesibilidad que esta herramienta tiene a tarjetas gráficas.

Librerías

Para codificar este trabajo se ha utilizado **Python** en su versión 3.10.10. Esta elección es debida a que normalmente el trabajo con redes neuronales se realiza en este lenguaje de programación y la documentación de las diferentes librerías son muy amplias y fáciles de entender.

Para el desarrollo de este proyecto se han utilizado diferentes librerías que permiten el desarrollo de este proyecto de manera eficiente. Entre las que se incluyen algunas como **Tensorflow** para el desarrollo de redes neuronales, **OpenCV** para facilitar la manipulación de imágenes, **PySimpleGUI** que se utilizó para el desarrollo de la interfaz gráficas debido a su simplicidad, **Matplotlib** para la visualización de los datos obtenidos en el proceso, **Numpy** que permite trabajar con matrices y realizar cálculos algebraicos de manera simple y **Segmentation-models** para el uso de modelos neuronales prediseñados para la segmentación de imágenes.

Capítulo 4

Métodos

En este capítulo nos centraremos en la descripción de los diferentes métodos usados para la realización de este proyecto, explicando el funcionamiento de cada uno de ellos.

4.1. Módulo de segmentación

Para llevar a cabo la segmentación de las imágenes, se ha realizado un estudio de las diferentes alternativas que existen, tal y como se indicó en el Capítulo 2. Finalmente se ha optado por utilizar la red neuronal denominada **U-Net** [9] debido a que tiene algunas características que se adaptan a la perfección a nuestro problema. Estas son:

- Arquitectura ligera que obtiene buenos resultados con datasets limitados.
- Testeadas en la segmentación de imágenes biomédicas.
- Facilidad de implementación.

A continuación se explicará la arquitectura de la red U-Net, así como la implementación utilizada en este proyecto y los motivos de las diferentes decisiones tomadas en cuanto a la implementación.

4.1.1. Explicación U-Net

Este tipo de red tiene una estructura de U, de ahí su nombre, y está formada por dos partes: la primera de **codificación** y la segunda de **decodificación** tal y como se ve en la figura 4.1.

La parte de codificación está formada por una serie de **capas convolucionales** que se encargan de extraer las características de la imagen de entrada y capas de **max pooling** que son las que comprimen la imagen a lo largo de los diferentes bloques codificadores hasta llegar a la decodificación.

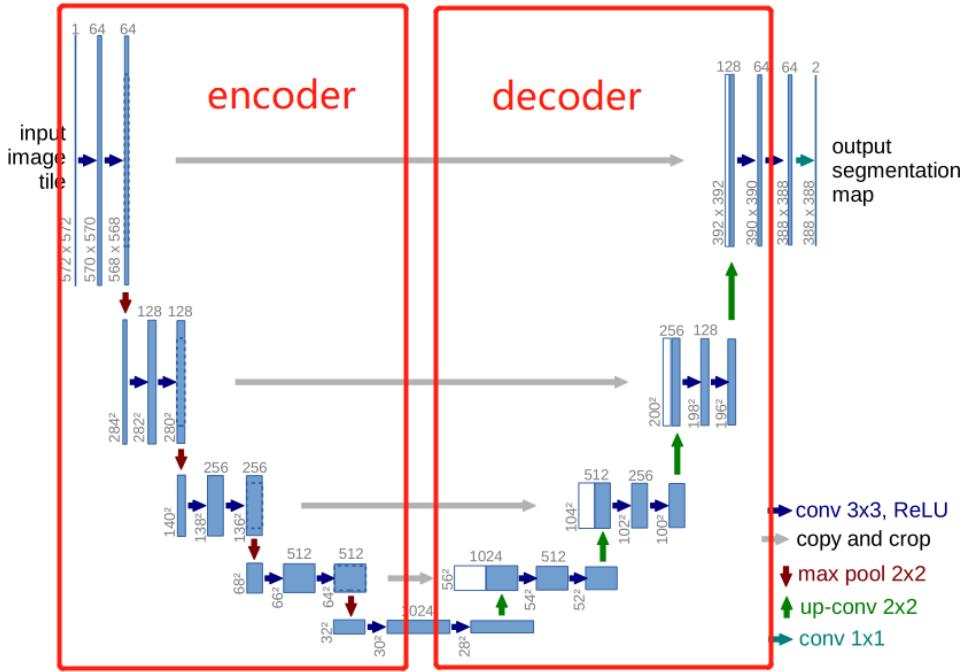


Figura 4.1: Arquitectura de la red U-Net

Estas capas se van encadenando de forma que la salida de una capa es la entrada de la siguiente.

La parte de decodificación está formada por una serie de **capas convolucionales** que se encargan de reconstruir la imagen segmentada a partir de las características extraídas en la parte de codificación y capas de **up sampling** que se encargan de aumentar la resolución de la imagen, estas capas se van encadenando de forma que la salida de una capa es la entrada de la siguiente.

Como se puede observar en la imagen, esta arquitectura es simétrica debido a que cada bloque codificador tiene conexiones con el bloque situado a la misma profundidad en la parte decodificadora.

Estas **skip connections** permiten combinar información de distintos niveles de abstracción y conservar detalles finos en la segmentación final, ya que mejora la reconstrucción de la imagen segmentada en el bloque decodificador.

4.1.2. Implementación U-Net

Para la implementación de la red U-Net se ha utilizado la librería Tensorflow y Keras. El resumen de este modelo se puede observar en el apéndice C.

Esta implementación se puede observar que es algo pesada con respecto a la arquitectura presentada en el trabajo original, estos cambios vienen dados por la introducción de diferentes tipos de capas para mejorar el rendimiento de la red neuronal y se explicarán en secciones posteriores.

En este trabajo se podría haber usado un modelo con algunas variaciones[15] como por ejemplo el uso de modelos como el **U-Net++**[16] o **EFF-UNet**[17] ya que estos modelos han demostrado ser ligeramente mejores que el modelo original de la U-Net, pero por la limitación de tiempo y recursos no se ha podido implementar.

Codificador de la U-Net

En cuanto al modelo implementado manualmente, las capas elegidas para el codificador son capas convolucionales de 3x3 con un stride de 1 y un padding de *same* para que la imagen no se vaya reduciendo a lo largo de las capas de la misma profundidad.

El número de filtros de los diferentes bloques va en función de la profundidad del bloque en la red, tal y como se puede ver en el apéndice C.

Con respecto a la implementación original[9] la utilizada se caracteriza por tener un nivel de profundidad más y un número de filtros mayor en cada bloque, esto se debe a que en las pruebas realizadas se ha observado que con un número de filtros mayor se obtienen mejores resultados. El bloque codificador como se puede observar recibe una imagen en formato RGB y va reduciendo su tamaño a la mitad cada vez que pasa por un bloque convolucional, hasta que llega al cuello de botella.

Al final de cada capa convolucional se añade una capa de BatchNormalization para normalizar los datos, con el objetivo de evitar el overfitting y mejorar el rendimiento de la red neuronal.

El bloque continúa con una capa de maxPooling que multiplica las regiones de la imagen por un kernel 2x2 para contraer la imagen a la mitad de tamaño. La capa final del bloque es una de Dropout que evita el overfitting. En función de la profundidad el número de filtros que se aplican varía, siendo 64 para la primera capa, 128 para la segunda, 256 para la tercera y 512 para la cuarta.

Para el cuello de botella se han elegido capas convolucionales de 3x3 con un stride de 1 y un padding de *same*, con una profundidad de 1024 filtros.

4.1.3. Decodificador de la U-Net

Para el decodificador, con el objetivo de aumentar la resolución, se ha elegido usar capas convolucionales traspuestas (*Convolutional2dTranspose*) de 3x3 con un stride de 2 y un padding de *same*, tal y como se puede observar en el apéndice C.

Se concatenan las salidas de los bloques codificadores con los inputs de los bloques decodificadores de la misma profundidad, con el objetivo de que no se pierda información a lo largo del proceso. Por este motivo la primera capa del decodificador tiene una profundidad de 512 filtros, la segunda de 256, la tercera de 128 y la cuarta de 64 siguiendo un orden inverso con respecto al bloque

codificador. De esta forma, se consigue que la red pueda reconstruir la imagen segmentada a partir de las características extraídas en la parte de codificación.

A los bloques decodificadores, se añaden capas de Dropout para evitar el overfitting, las cuales son seguidas de capas convolucionales para obtener características más complejas.

Finalmente, la capa de salida produce una imagen de tres canales(uno por cada clase del dataset PETS) que aplica padding y con una función de activación softmax para obtener la probabilidad de cada pixel de pertenecer a una clase. Estas probabilidades servirán para formar la máscara segmentada final.

4.1.4. Entrenamiento de las implementaciones

Para entrenar los diferentes modelos se aplicará la técnica de **Transfer Learning** [18] que consiste en utilizar un modelo ya entrenado con un dataset muy grande de imágenes variadas y aplicarle un entrenamiento con un dataset más pequeño que se adapte a nuestro problema. De esta forma, se consigue que el modelo aprenda con pocos datos del problema a resolver.

Debido a la limitación de tamaño en el dataset de nanopartículas de Github se utilizará transfer learning desde el dataset PETS [13] para entrenar los diferentes modelos y se utilizará el dataset de nanopartículas para realizar el fine-tunning.

Una vez completado el primer entrenamiento de los modelos, se pueden obtener resultados como en la figura 4.2, donde se puede observar el input, la máscara y la predicción del modelo de Oxford. Se obtienen tres tipos de clases diferentes ya que el dataset está definido para segmentar el fondo, los animales y los píxeles indeterminados, en nuestro caso solo predeciremos dos clases, el fondo y las nanopartículas por lo que será necesario realizar un preprocessado de los datos para obtener una máscara binaria.

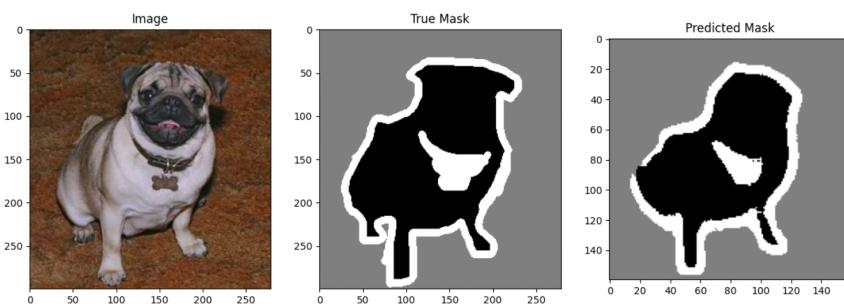


Figura 4.2: Input, Mask y Prediction usando el modelo de Oxford

A continuación, se realizó el proceso de Transfer Learning a todos los modelos implementados. Estos entrenamientos fueron realizados con el dataset de

nanopartículas, utilizando imágenes de tamaño 256x256 píxeles para agilizar el fine-tuning, y se compararon los resultados obtenidos por los diferentes modelos.

Durante el proceso de fine-tuning se empleó el método denominado **data augmentation** con el objetivo de aumentar el conjunto de datos de entrenamiento. Para implementar este procedimiento aplicamos rotaciones, aumentos de brillo, aumentos de contraste, aumentos de saturación, aumentos de la escala y desplazamientos aleatorios en las imágenes del conjunto de datos. Estas modificaciones se realizan de una manera totalmente aleatoria para tener un dataset más variado y así conseguir un mejor entrenamiento. Las transformaciones aleatorias tienen que ser las mismas para las imágenes y las máscaras para que no se pierda la relación entre ellas.

Durante el entrenamiento de los modelos, se utilizó como función de pérdida la denominada **Categorical CrossEntropy** que se define como:

$$\text{CategoricalCrossEntropy} = - \sum_{i=1}^C y_i \log(p_i) \quad (4.1)$$

Donde y_i es la etiqueta real de la clase i y p_i es la probabilidad de la clase i . Y las medidas de precisión utilizadas fueron **Accuracy** y **Intersection over Union (IoU)** que se definen como:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.2)$$

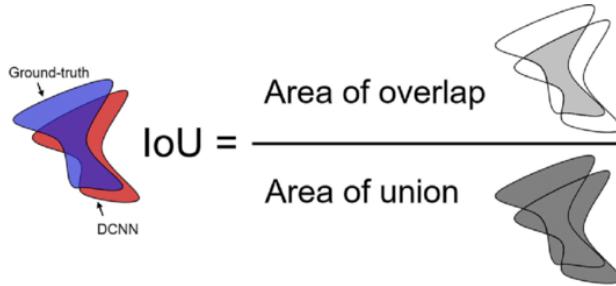


Figura 4.3: Representación gráfica de IoU

$$\text{IoU} = \frac{TP}{TP + FP + FN} = \frac{\text{Area de intersección}}{\text{Area de unión}} \quad (4.3)$$

Donde TP es el número de verdaderos positivos, TN es el número de verdaderos negativos, FP es el número de falsos positivos y FN es el número de falsos negativos. Todas estas medidas toman como referencia los píxeles que tienen que ser clasificados como partículas.

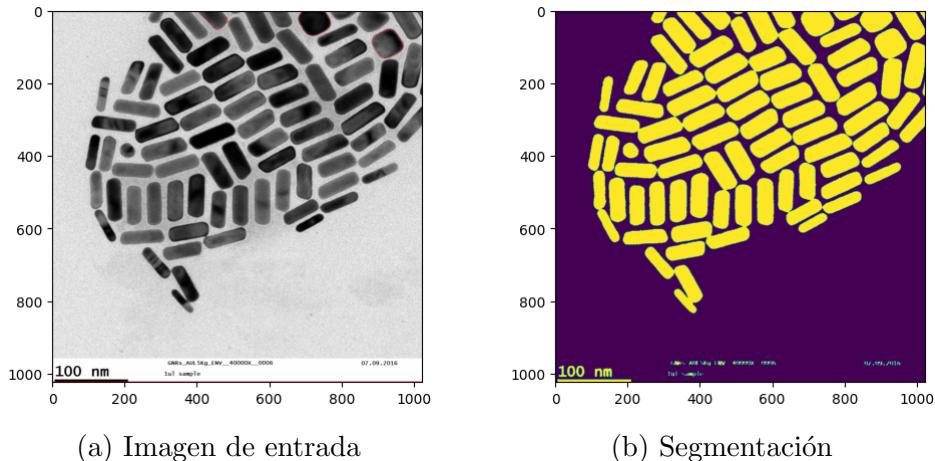


Figura 4.4: Imagen de entrada y Segmentación usando el modelo final

Tras realizar los entrenamientos con los datos aumentados se puede observar la predicción que se realiza de una imagen con el modelo final tal y como se puede ver en la Figura 4.4. Este modelo obtiene un **IoU**(Ecuación 4.3) de cerca del 75 % así como una **precisión**(Ecuación 4.2) cercana al 94 %.

4.1.5. Limitaciones de la U-Net

A pesar de que las imágenes segmentadas obtenidas por los diferentes modelos son bastante buenas, se puede observar que en algunas imágenes la segmentación de diferentes nanopartículas se tocan entre si. Esto se debe a que el dataset de nanopartículas no es lo suficientemente grande y a que la U-Net tiene esta limitación por su arquitectura.

Para mitigar esta limitación se ha realizado un proceso de **post-procesado** de las imágenes segmentadas. Este proceso consiste en aplicar un filtro de **erosión** y **dilatación** a las imágenes segmentadas para separar el mayor número posible de nanopartículas que se toquen entre si.

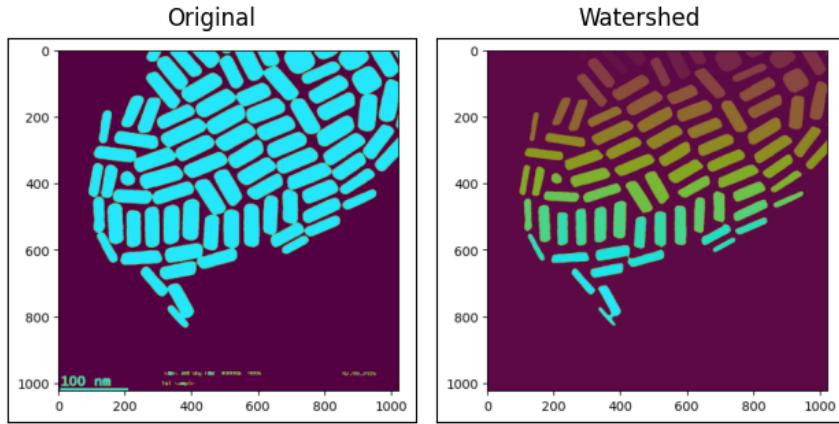


Figura 4.5: Ejemplo de watershed aplicado a una imagen segmentada

Este proceso se denomina **transformada watershed**[19] y, como se explicó antes, mitiga este problema pero con el inconveniente de que los objetos que se van a medir no son los mismos que los que se han segmentado, por lo que el tamaño de las nanopartículas finales será algo menor que el real tal y como se puede observar en la figura 4.5.

Otra limitación de la U-Net es que en algunos casos si los colores del fondo o de las nanopartículas son muy parecidos, esta arquitectura tiene dificultades a la hora de segmentar correctamente las nanopartículas.

Para mitigar este problema se ha aplicado un **preprocesado** de las imágenes, que consiste en el aumento del contraste de las imágenes y la eliminación del ruido de las imágenes, con el objetivo de poder aumentar la distingibilidad de las nanopartículas con el fondo en imágenes como la de la Figura 1.1b.

4.2. Separación de las regiones de interés

Una vez que el proceso de segmentación ha finalizado y obtenemos una máscara similar a la de la figura 4.5, es necesario aplicar un **postprocesado** a esta para separar las diferentes regiones de interés para poder clasificar y medir cada objeto de manera individual.

Para realizar esta separación nos hemos apoyado en la función **connectedComponentsWithStats** de OpenCV que devuelve una lista de los contornos de cada región de interés. Al aplicar esta función podemos obtener máscaras

individuales para cada partícula.

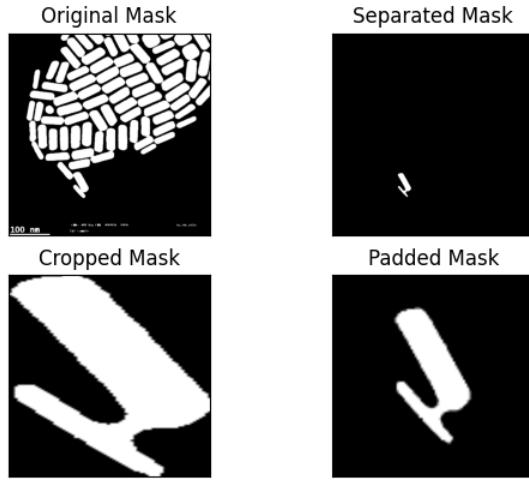


Figura 4.6: Proceso de separación de las regiones de interés

A esta máscara individual se le aplica un mecanismo de detección de **bounding boxes** para obtener la región de la imagen correspondiente a una nanopartícula.

Una vez obtenida el **bounding box** del objeto, se recortará la región de la máscara correspondiente a este. Esto se logra a través de la función **connectedComponentsWithStats** de OpenCV tal.

Una vez tenemos aislado el bounding box de una nanopartícula, haremos que la imagen sea cuadrada. Para conseguirlo añadimos padding a la imagen hasta que sea cuadrada para evitar que la partícula se deforme cuando tengamos que adaptar el tamaño de la imagen al tamaño que recibe como input la red de clasificación. Haciendo este proceso mejoramos el rendimiento de la red neuronal de clasificación. Todas estas transformaciones se pueden apreciar en la figura 4.6.

4.3. Clasificador de nanopartículas

Una vez que tenemos las imágenes de las nanopartículas separadas y con el padding añadido, podemos pasar a clasificarlas. Para realizar esto se usará una red neuronal densamente conectada del Listing 4.1 con 7 capas ocultas y una capa de salida con 5 neuronas, una para cada tipo de nanopartícula.

Esta red recibirá una imagen de tamaño 100x100 píxeles, con el objetivo de obtener una mayor precisión sin deformar la nanopartícula aislada, en escala de grises y devolverá un vector de 5 posiciones con la probabilidad de que la imagen pertenezca a cada una de las clases.

```

1 -----
2 Layer (type)           Output Shape      Param #
3 -----
4 flatten_31 (Flatten)     (None, 10000)       0
5
6 dense_248 (Dense)       (None, 2048)        20482048
7
8 dense_249 (Dense)       (None, 512)         1049088
9
10 dense_250 (Dense)      (None, 256)         131328
11
12 dense_251 (Dense)      (None, 256)         65792
13
14 dense_252 (Dense)      (None, 128)         32896
15
16 dense_253 (Dense)      (None, 128)         16512
17
18 dense_254 (Dense)      (None, 64)          8256
19
20 dense_255 (Dense)      (None, 5)           325
21 -----
22 Total params: 21,786,245
23 Trainable params: 21,786,245
24 Non-trainable params: 0

```

Listing 4.1: Arquitectura del modelo usado para clasificación

Todas neuronas excepto las de la última capa tendrán todas la función de activación **relu**, esta función se caracteriza por estar definida como $f(x) = \max(0, x)$, es decir, si el valor de entrada es negativo, la salida será 0, en caso contrario, la salida será el mismo valor de entrada. En la última capa la función de activación usada es la denominada **softmax** que se define como $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$, esta función se usa para obtener la probabilidad de que una imagen pertenezca a cada una de las clases y tiene como característica que la suma de todas las probabilidades es 1.

Las clases que se van a identificar hacen referencia a la forma geométrica de la nanopartícula, estas clases son:**Bipirámides, Rectangulares, Hexagonales, Cuadradas y Circulares**.

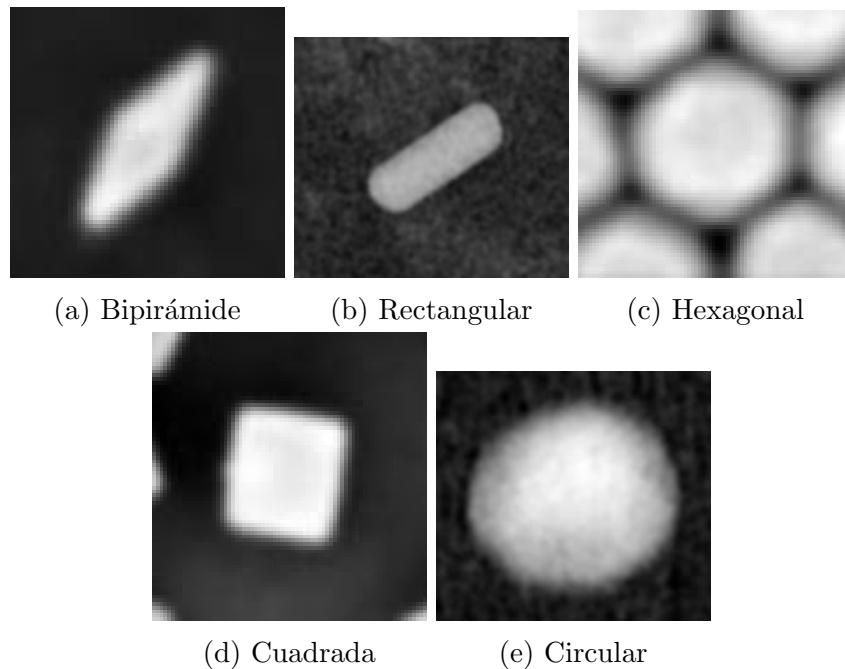


Figura 4.7: Ejemplos de las diferentes clases de nanopartículas

Para entrenar este modelo, se han usado aproximadamente 120 imágenes de cada tipo de nanopartícula, 600 máscaras se han usado en total al recortar de forma manual y usar data augmentation en las secciones de las imágenes originales que contuvieran ese tipo de partículas.

Una vez que obtenemos todas las nanopartículas de una imagen, se guardan en diferentes carpetas en función del tipo de nanopartícula que sea. Este proceso se realizó manualmente y se denomina como **etiquetado de datos**.

Para cuantificar lo bien que funcionan los modelos de clasificación, se utilizó la métrica **accuracy** (*Ecuación 4.2*) que se define como el número de predicciones correctas sobre el número total de predicciones realizadas.

4.4. Medición de las nanopartículas

Una vez que ya hemos definido todo el proceso para localizar y clasificar las nanopartículas, es necesario medir su tamaño. Para conseguir este objetivo, es necesario contar el número de píxeles que ocupa cada nanopartícula en la imagen.

Una vez que tenemos el número de píxeles que ocupa una nanopartícula, es necesario convertirlo a la escala correspondiente.

Para ello se ha desarrollado una funcionalidad que permite leer automáticamente las escalas de imágenes del CiQUS y convertir el número de píxeles a la escala correspondiente. En caso de que no se pueda leer la escala, se puede introducir manualmente el ratio de conversión a través de la interfaz gráfica y si no se

introduce ningún valor, se asume que el ratio de conversión es 1 y el tamaño de las nanopartículas se medirá en píxeles.

El proceso seguido para realizar esta funcionalidad es el siguiente:

1. Se lee la imagen.
2. Mediante el uso de procesos de umbralización, erosión y dilatación se obtiene la región de la imagen que contiene la escala.
3. Filtramos los elementos de la región que correspondan a los números de la escala, esto se puede saber conociendo la proporción del tamaño con respecto al tamaño de la escala.
4. Se utiliza una red neuronal de **Kaggle**¹ para reconocer los números de la escala, obteniendo el valor numérico de esta.
5. Se obtiene el área real correspondiente a cada píxel.

Una vez que tenemos el ratio de conversión, podemos obtener el área de cada nanopartícula. Esto se realiza multiplicando el número de píxeles que ocupa la nanopartícula por el ratio de conversión.

Una vez que tenemos el área de cada nanopartícula, podemos obtener un histograma multiclasa del tamaño de las nanopartículas, tal y como se puede observar en la figura 4.8, donde podemos observar por un lado la imagen que se ha usado para obtener el histograma y por otro lado los diferentes histogramas que representan el tamaño y la cantidad de nanopartículas de cada clase que existen en la imagen.

¹<https://www.kaggle.com/code/arunrk7/digit-recognition-using-cnn-99-accuracy>

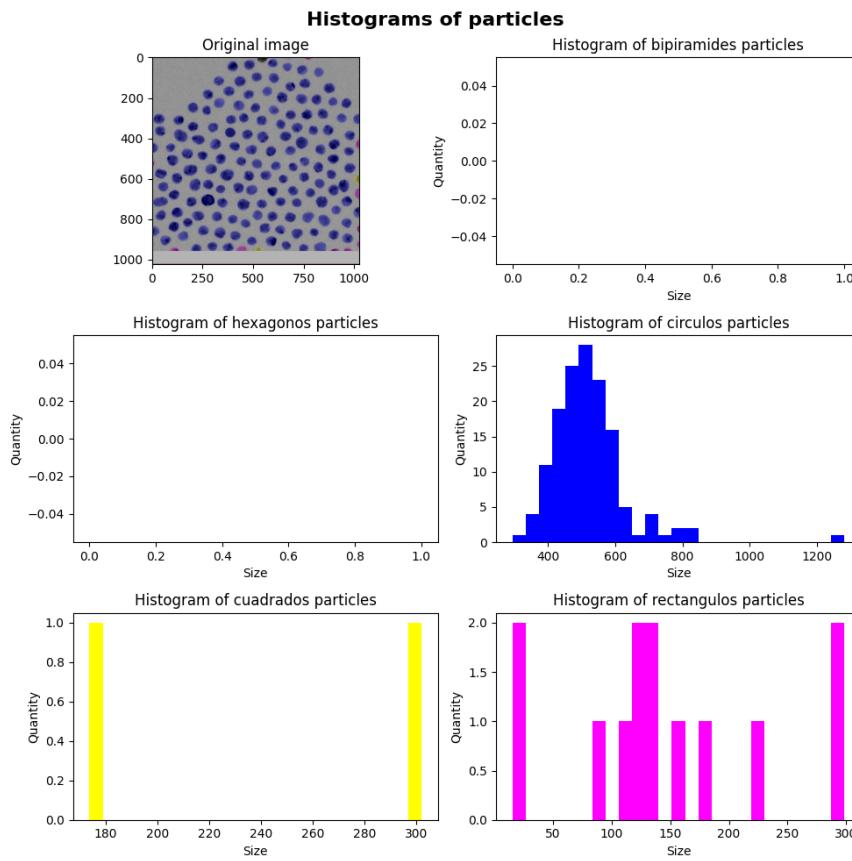


Figura 4.8: Histograma multiclasé del tamaño de las nanopartículas

4.5. Aumento utilidad de las funcionalidades

Como se comentó en la sección anterior, la funcionalidad de leer las escalas automáticamente solo funciona con las imágenes del Citius.

Para permitir la medición de nanopartículas en imágenes con otros formatos, se ha desarrollado una interfaz gráfica que permite al usuario introducir la escala de manera manual, así como la elección de otras opciones como:

- **Watershed:** Permite activar o desactivar el algoritmo de Watershed.
- **Denoise:** Permite aplicar eliminación de ruido a las imágenes.
- **Aumento contraste:** Permite aumentar el contraste de las imágenes.
- **Tamaño de la imagen:** Permite seleccionar el tamaño de la imagen.

Esta interfaz gráfica se ha desarrollado con la librería **PyQt5** de Python. Esta permite el uso de todas las funcionalidades desarrolladas en este proyecto con una mayor variedad de imágenes.

Capítulo 5

Pruebas

5.1. Pruebas módulo de segmentación

En estas pruebas se evaluará el comportamiento del módulo de segmentación sobre un conjunto de imágenes del repositorio de Github usado para el entrenamiento de los modelos¹, que se encarga de la extracción de las características de las imágenes.

En la figura 5.1 se pueden ver los tipos de imágenes que forman parte del proceso de evaluación. En la imagen 5.1a se puede ver la imagen original y en la imagen 5.1b se puede ver la imagen segmentada, estas imágenes son las que se deberían obtener del módulo de segmentación.

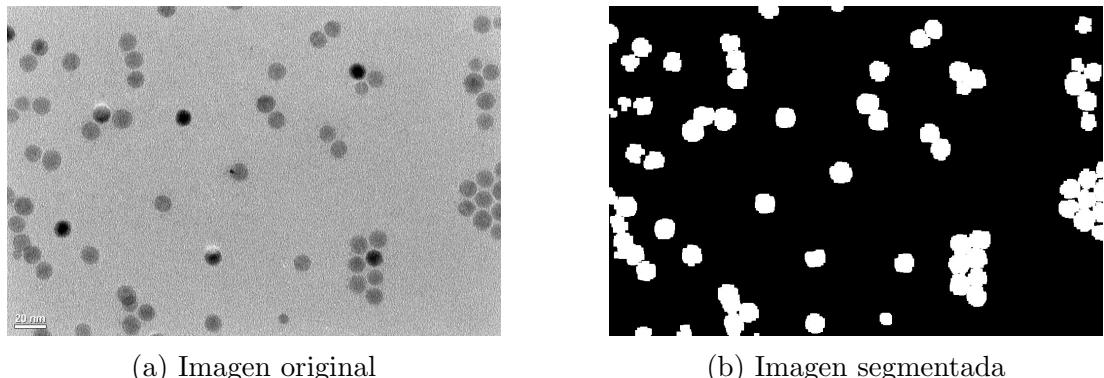


Figura 5.1: Prueba de segmentación

Una vez que conocemos los tipos de imágenes que se van a utilizar, se procede a realizar las pruebas.

Las pruebas tendrán como objetivo evaluar los distintos modelos y medidas mencionados en el Capítulo 4, y contrastar los resultados obtenidos con los resultados esperados para determinar cuál de ellos es el más efectivo.

¹Dataset disponible en <https://github.com/anilgurses/TEM-Nano-Particle-Cell-Dataset>

Las métricas que se utilizarán para evaluar los modelos son las siguientes:

- **Accuracy:** La precisión de este modelo se va a medir a nivel de píxel y se calcula como el número de píxeles correctamente clasificados sobre el número total de píxeles. Ecuación 4.2.
- **Intersection over Union (IoU):** El IoU de este modelo se va a medir con respecto a los píxeles correspondientes a las nanopartículas de la máscara. Ecuación 4.3.

Los modelos que vamos a evaluar son tres modelos diferentes:

- **Modelo de Oxford:** Este modelo es el modelo que se encuentra en un notebook de Google Colab², este modelo es el más ligero de los tres y el que menos tiempo tarda en realizar las predicciones. Se caracteriza por tener una menor profundidad que los otros modelos y por tener 32 filtros convolucionales en la capa menos profunda y 256 en la más profunda. Es el modelo más próximo al de la arquitectura original.
- **Modelo de la librería:** Este modelo es el modelo que se encuentra en la librería de Python **segmentation-models**³, este modelo es el más pesado de los tres y el que más tiempo tarda en realizar las predicciones. Se caracteriza por tener en la etapa codificadora el modelo VGG-16, por tener una mayor profundidad que el modelo implementado por Oxford y por tener un mínimo de 64 filtros convolucionales en la capa menos profunda ligera y 1024 en la más profunda.
- **Modelo implementado:** Este modelo es el modelo que hemos implementado nosotros, este modelo es algo más ligero que el anterior pero más pesado que el de Oxford. Se caracteriza por tener el mismo esquema que el modelo implementado por Oxford pero con una mayor profundidad y por tener un mínimo de 64 filtros convolucionales en la capa menos profunda y 1024 en la más profunda.

Estos modelos al ser redes convolucionales pueden tener de entrada imágenes de cualquier tamaño, por lo que podemos hacer un análisis de cómo afecta el tamaño a las diferentes métricas.

Una vez realizadas las pruebas se obtendrán los valores de las métricas mencionadas anteriormente para cada modelo y se representarán en la Figura 5.2.

²Notebook disponible en <https://rb.gy/m1djm>

³Librería disponible en <https://pypi.org/project/segmentation-models/>

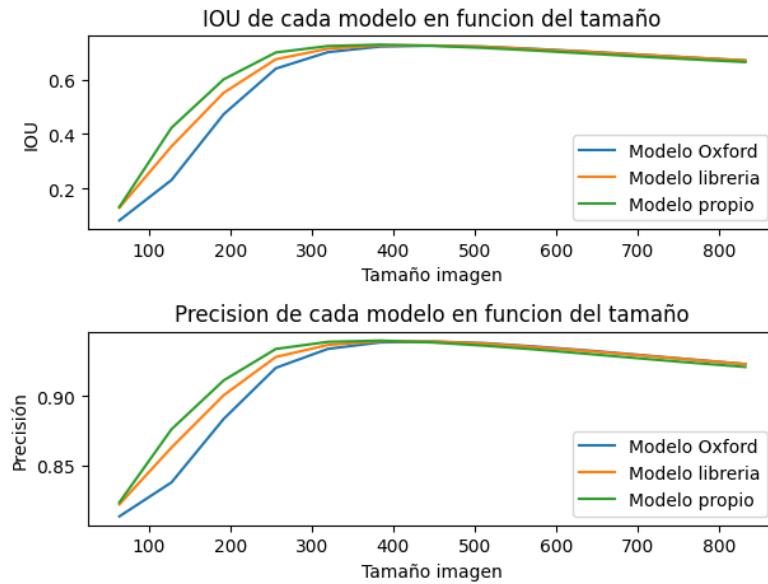


Figura 5.2: Prueba módulo segmentación

Como se puede observar en la figura anterior, el modelo de segmentación implementado por Oxford es el que peor resultados obtiene, esto es debido a que este modelo es muy ligero y no tiene tantos parámetros como los otros a pesar de esto, el modelo implementado por Oxford es el que menos tiempo tarda en realizar las predicciones.

Por otro lado el modelo implementado por la librería y por nosotros obtienen resultados muy similares, pero el modelo implementado por la librería tarda más tiempo en realizar las predicciones.

Se puede concluir que a mayor tamaño de imagen los modelos obtienen rendimientos similares y se optó por elegir el modelo implementado por la librería ya que es el que mejores resultados obtiene y el tiempo de predicción es aceptable para cualquier resolución elegida.

Otro fenómeno que se observa es que a medida que el tamaño de las imágenes aumenta, el tiempo de predicción también aumenta, esto es debido a que a mayor tamaño de imagen, mayor número de píxeles y por tanto mayor número de operaciones a realizar.

En cuánto a las opciones de preprocessado que se han mencionado en el Capítulo 4, se puede observar en la Figura 5.3 que el preprocessado que mejores resultados obtiene es la eliminación del ruido, pero que el aumento de contraste empeora el rendimiento. Esto es debido a que en el dataset de prueba las nanopartículas ya son claramente distinguibles, por lo que aumentar el contraste no aporta nada llegando incluso a empeorar el rendimiento. Se puede concluir que el preprocessado se debe aplicar en función del tipo de imágenes que se vayan a utilizar.

En cuánto a las opciones de postprocesado que se han mencionado en el

Capítulo 4, se puede observar en la Figura 5.3 que el postprocesado empeora el IoU que se obtiene, esto ya se conocía de antemano ya que el postprocesado se utiliza para evitar que nanopartículas colindantes se unan en una misma máscara, y para conseguir esto se deben omitir los píxeles que se encuentren en la frontera de dos nanopartículas, por lo que se pierde información y el IoU disminuye.

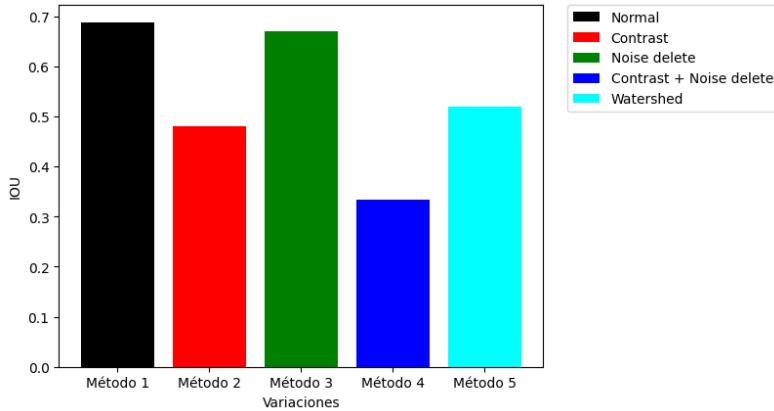


Figura 5.3: Prueba opciones preprocesado y postprocesado

5.2. Pruebas módulo de clasificación

Una vez que se ha realizado la prueba del módulo de segmentación, se procede a realizar la prueba del módulo de clasificación.

Para realizar las pruebas se utilizará la métrica de **accuracy** y se usará a nivel de imagen. Esta métrica se calcula como el número de predicciones correctas sobre el número total de predicciones realizadas (Ecuación 4.2).

El conjunto de datos sobre el que se realizarán estas pruebas es un subconjunto del conjunto de datos de entrenamiento, este subconjunto de datos se obtiene separando manualmente las máscaras de las nanopartículas en función del tipo al que pertenezcan. De este modo se obtiene un conjunto de imágenes de cada tipo de nanopartícula.

Una vez que se tiene el conjunto de datos, se procede a realizar las pruebas sobre el modelo propuesto en el Capítulo 4.

El objetivo de estas pruebas es medir la precisión del método de clasificación utilizado, para mejorar los resultados se realizaron las pruebas con imágenes de diferentes tamaños, así obtendremos el mejor tamaño de imagen para el modelo propuesto. En estas pruebas obtendremos el valor de la métrica **accuracy** y con el mejor valor de esta métrica podremos obtener el tamaño de imagen que mejor funciona para el modelo propuesto. Al ejecutar las pruebas obtenemos los resultados de la figura 5.4.

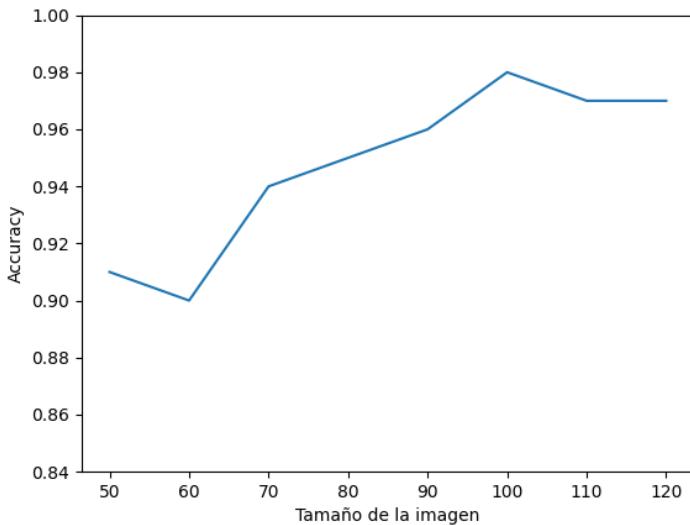


Figura 5.4: Pruebas módulo clasificación

Como se puede observar, el modelo tiene una precisión bastante alta siempre superior al 90 %. Pero a medida que el tamaño de imagen aumenta, la precisión del modelo también aumenta hasta llegar a un valor de 98 % cuando se utilizan imágenes de tamaño 100x100.

Por este motivo, se eligió que el tamaño de entrada del modelo fuera 100x100 ya que es un tamaño de imagen que no es muy grande y que a la vez permite obtener un valor muy alto en la precisión a la hora de clasificar, obteniendo mejores predicciones.

5.3. Pruebas medición nanopartículas

Para demostrar que el método propuesto para medir las nanopartículas es preciso es necesario comprobar que el tamaño medido de las mismas no tiene mucha variación al cambiar el tamaño de la imagen.

Para realizar estas pruebas se usarán una imagen del CIQUS y se medirán las nanopartículas de la imagen, todo esto con diferentes tamaños de imagen. En caso de que el tamaño medio de las nanopartículas no varíe mucho, se puede concluir que el método propuesto es preciso.

Tras realizar las pruebas se puede observar en la Figura 5.5 que la variación de la media del tamaño de las nanopartículas es muy pequeña, por lo que se puede concluir que el método propuesto es preciso. La variación que existe entre tamaños de imagen pequeños es debido a que si la imagen es muy pequeña un gran número de nanopartículas cercanas entre sí se unen en una misma máscara, por lo que el tamaño de la máscara es mayor y el tamaño medido es mayor. Estas

conclusiones son acertadas ya que en el proceso con el que se obtiene el valor de área de cada píxel, explicado en el Capítulo 4, se utiliza un factor de conversión con respecto a la escala de la imagen, por lo que el tamaño de la nanopartícula no depende del tamaño de la imagen si no de la escala de la imagen.

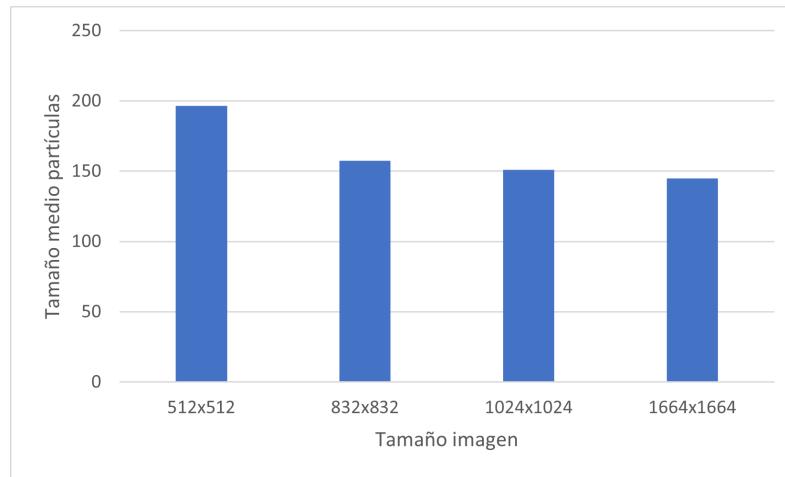


Figura 5.5: Pruebas medición nanopartículas

5.4. Ejemplos individuales

Una vez que se han realizado las pruebas de los módulos de segmentación y clasificación, se procede a realizar las pruebas del experimento conjunto.

Estas pruebas conjuntas consisten en aplicar el modelo de segmentación a las imágenes de prueba y posteriormente aplicar el modelo de clasificación a las nanopartículas obtenidas. Una vez que se han aplicado los dos modelos, se obtiene el resultado final del experimento.

Estos casos de uso se realizan a través de la interfaz gráfica, a través de la cual obtendremos los histogramas de las nanopartículas obtenidas y podremos ver de manera gráfica los errores de la solución propuesta.

Un ejemplo de este resultado final se puede apreciar en la figura 4.8, donde se puede apreciar que las nanopartículas situadas a los bordes de la imagen no se suelen clasificar correctamente, en cambio las demás tienen una clasificación correcta.

Una de las limitaciones que habíamos expuesto en el Capítulo 4 era que el modelo de segmentación no era capaz de detectar nanopartículas que estuvieran muy juntas, este efecto se puede apreciar en la figura 5.6, donde se puede observar que las nanopartículas que están muy juntas no se detectan correctamente y se clasifican como una sola nanopartícula de una clase a la que no pertenecen.

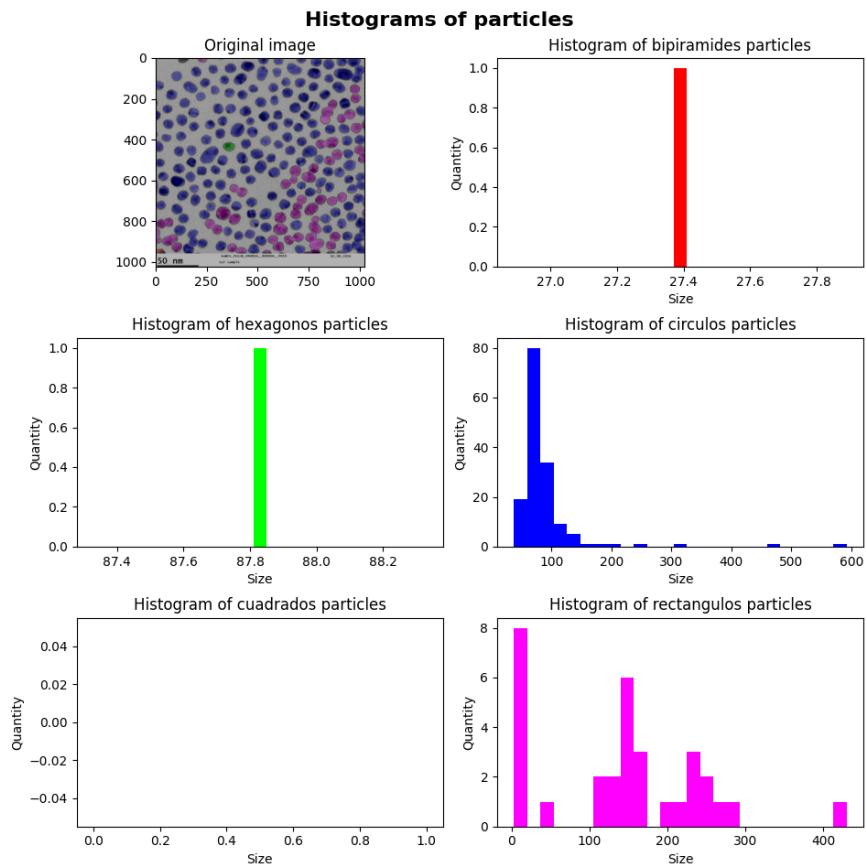


Figura 5.6: Histograma de nanopartículas

Una vez realizadas las pruebas de estos casos de uso, quedaría probar a ver si nuestro modelo es capaz de detectar y clasificar diferentes tipos nanopartículas en la misma imagen, esto se puede ver en la figura 5.7.

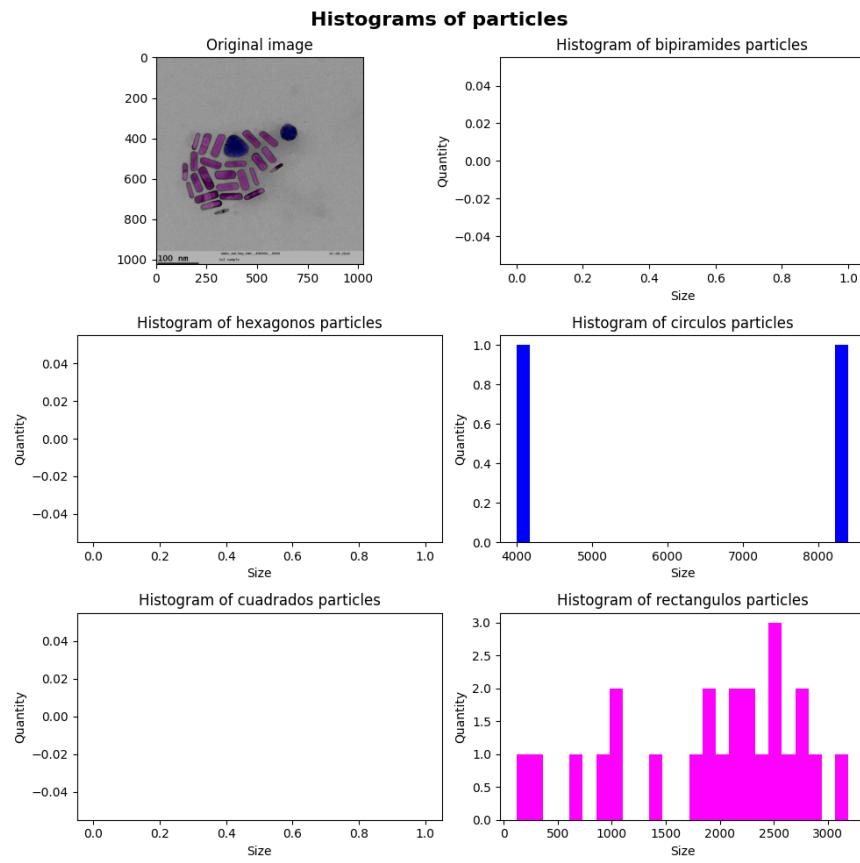


Figura 5.7: Histograma de nanopartículas

Capítulo 6

Discusión de los resultados

Tal y como se introdujo en el Capítulo 2, el objetivo de este trabajo era desarrollar una herramienta software basada en herramientas de visión por computador que pudiera servir como herramienta los investigadores del CIQUS para facilitarles tareas tediosas.

Para ello, se ha desarrollado una herramienta teniendo en cuenta la cantidad de datos y recursos computacionales disponibles.

La solución propuesta es capaz de clasificar nanopartículas en función de su forma geométrica, utilizando para ello un modelo de aprendizaje profundo basado en redes neuronales convolucionales para segmentar, y un modelo de aprendizaje supervisado basado en redes neuronales densas para clasificar.

Esta aproximación permite obtener resultados similares, con menor cantidad de recursos y tiempo, que los que pudieramos haber obtenido en caso de haber utilizado otro tipo de modelos de detección de objetos basados en redes neuronales convolucionales, como los descritos en el Capítulo 2.

En el Capítulo 5 mostramos que el módulo software desarrollado es capaz de clasificar varias clases de nanopartículas en la misma imagen.

A pesar de todos estos resultados positivos obtenidos, el módulo software desarrollado no es capaz de clasificar todas las clases de nanopartículas correctamente. Tal y como se dijo previamente, si las nanopartículas están superpuestas o si su forma no es clara el módulo software no es capaz de clasificarlas correctamente.

Por lo tanto, se puede concluir que el módulo software desarrollado es capaz de clasificar nanopartículas en casos en los que la forma de las nanopartículas es clara y no están superpuestas. En caso contrario, el módulo software tendrá imprecisiones.

Capítulo 7

Conclusiones y posibles ampliaciones

La principal aportación de este trabajo es el desarrollo de un módulo software basado en herramientas del campo de la visión por computador que permite clasificar nanopartículas en función de su forma geométrica. Esta alternativa es capaz de clasificar nanopartículas en casos en los que la forma de las nanopartículas es clara y no están superpuestas. Por lo que la solución podría usarse en casos reales en los que se cumplan estas condiciones.

En este trabajo quedó demostrado que el usar una red neuronal del tipo UNet para segmentar las nanopartículas, ya que son las más usadas en el ámbito biomédico, y una simple red neuronal densa puede ser una alternativa viable para solucionar este problema. A pesar de estos resultados, también hay que destacar que para mejorar el rendimiento del sistema fue necesario aplicar otras operaciones como pueden ser la eliminación de ruido, el aumento del contraste en imágenes o el uso de watershed.

Una de las principales ampliaciones podría ser la utilización de más datos para el entrenamiento de los modelos, para lograr esto sería necesario realizar un etiquetado de datos manual. Esto permitiría mejorar el rendimiento del sistema, ya que se podrían utilizar más imágenes para el entrenamiento.

En caso de que esta ampliación fuera posible se podrían introducir más capas a los modelos utilizados, lo que podría mejorar el rendimiento del sistema.

En conclusión, el módulo software desarrollado es capaz de clasificar nanopartículas en casos en los que la forma de las nanopartículas es clara y no están superpuestas. En caso contrario, el módulo software tendrá imprecisiones, las cuáles se podrían reducir si se dispusiera de más datos para el entrenamiento de los modelos.

Apéndice A

Manuales técnicos

Para reproducir el trabajo realizado en este proyecto, se debe utilizar Python en la versión 3.10.10 y se deben instalar las siguientes librerías:

Librería	Versión	Librería	Versión
numpy	1.21.4	Pillow	9.4.0
opencv-python	4.7.0.62	PyQt5	5.15.9
matplotlib	3.7.1	PyQt5-Qt5	5.15.2
matplotlib-inline	0.1.6	PyQt5-sip	12.11.1
keras	2.11.0	PyQt5Designer	5.14.1
keras-unet-collection	0.1.13	PySimpleGUI	4.60.4
keras-visualizer	3.1.2	pip	23.1.2
visualkeras	0.0.2	scikit-image	0.20.0
tensorflow	2.11.0	scikit-learn	1.2.2
tensorflow-estimator	2.11.0	tensorboard	2.11.2
tensorflow-intel	2.11.0	tensorboard-data-server	0.6.1
tensorflow-io-gcs-filesystem	0.31.0	tensorboard-plugin-wit	1.8.1
visualkeras	0.0.2	segmentation-models	1.0.1
h5py	3.8.0	graphviz	0.20.1
pandas	1.5.3	seaborn	0.12.2

Cuadro A.1: Librerías utilizadas en el proyecto

Apéndice B

Manuales de usuario

Este apéndice incluye todas las funciones creadas que permiten utilizar la aplicación desarrollada, el código de la aplicación se encuentra en el siguiente enlace de OneDrive¹.

B.1. Interfaz gráfica

Para utilizar la interfaz gráfica es necesario intalar las librería descritas en el ReadMe del repositorio de OneDrive y ejecutar el archivo **GUI.py**. Las funciones de la interfaz gráfica son las mismas que las descritas en el Capítulo 4 y se incluyen tambien las funcionaldes de la sección 4.5 de este trabajo, permitiendo que se puedan utilizar de una manera más simple para los usuarios que no tengan conocimientos de programación.

B.2. SegmentationModule

B.2.1. Definición de la clase

En esta clase se encuentran todas las funciones que permiten realizar la segmentación, clasificación y medición de las nanopartículas en imágenes. Para utilizar esta clase, se debe importar el módulo y crear un objeto de la clase SegmentationModule. A esta clase se le pueden pasar los siguientes parámetros:

- **model_path**: Ruta del modelo de segmentación a utilizar. Por defecto, se utiliza el modelo de segmentación de nanopartículas, por defecto es el modelo de UNet expuesto en este trabajo.

¹Enlace al repositorio de OneDrive:

https://nubeusc-my.sharepoint.com/:f/g/personal/fernando_gonzalez_salas_rai_usc_es/Euk9aDRoPtFKmGdWBYS6DIgB4QJ0JugKhyx05hESg899uQ?e=V5LR1r

- **model_classification:** Ruta del modelo de clasificación de nanopartículas a utilizar. Por defecto, se utiliza el modelo de clasificación de nanopartículas expuesto en este trabajo.
- **model_path_numeros:** Ruta del modelo de clasificación de números a utilizar, utilizado para el reconocimiento de la escala. Por defecto, se utiliza el modelo de clasificación de números expuesto en este trabajo.

B.2.2. Funciones de la clase

def classify(self,img):

Esta función se encarga de clasificar las nanopartículas de manera individual. En esta función se realiza las transformaciones necesarias para que el modelo de clasificación pueda clasificar las imágenes. Esta función recibe como parámetros:

- **img:** Matriz numpy que representa la imagen de la nanopartícula a clasificar.

Como output esta función devuelve un entero que representa la clase a la que pertenece la nanopartícula.

def prepare_image_to_segmentation

(self,path,size=(1024,1024),contrast=False,noise=False):

Esta función se encarga de preparar una imagen para que pueda ser segmentada. Esta función recibe como parámetros:

- **path:** Ruta de la imagen a preparar.
- **size:** Tamaño de la imagen a preparar, por defecto es (1024,1024).
- **contrast:** Booleano que indica si se debe aplicar un contraste a la imagen, por defecto es False.
- **noise:** Booleano que indica si se debe aplicar eliminación de ruido a la imagen, por defecto es False.

def obtain_mask(self,i):

Esta función se encarga de obtener la máscara final de la predicción realizada por el modelo de segmentación. Esta función recibe como parámetros:

- **i:** Tensor de numpy que representa la predicción realizada por el modelo de segmentación.

Como output esta función devuelve una matriz de numpy que representa la máscara final de la predicción realizada por el modelo de segmentación.

```
def display_mask(self,i):
```

Esta función se encarga de mostrar la máscara final de la predicción realizada por el modelo de segmentación. Esta función recibe como parámetros:

- **i**: Tensor de numpy que representa la predicción realizada por el modelo de segmentación.

Esta función no devuelve ningún output.

```
def separar_regionescala(self,img,path="./prueba/"):
```

Esta función se encarga de separar las diferentes regiones del área de la escala de una imagen. Una vez realizado esto, se guardan las regiones correspondientes a los números de la escala y se obtiene el valor numérico de esta, para posteriormente obtener el ratio de conversión píxel-área. Esta función recibe como parámetros:

- **img**: Matriz numpy que representa el área de la imagen correspondiente a la escala.
- **path**: Ruta donde se guardarán las imágenes de las diferentes regiones de la escala.

Esta función devuelve el ratio de conversión píxel-área y el valor numérico de la escala.

```
def get_scale_area(self,im,segmented_scale):
```

Esta función se encarga de obtener el área de la escala de una imagen. Esta función recibe como parámetros:

- **im**: Matriz numpy que representa la imagen de la que se quiere obtener el área de la escala.
- **segmented_scale**: Matriz numpy que representa la máscara de la imagen con la escala segmentada.

Esta función devuelve el output de la función separar_regionescala y la imagen original sin la escala.

```
def prepare_image_without_scale
```

```
(self,path,size=(1024,1024),noise=False,contrast=False):
```

Esta función se encarga de preparar una imagen para que pueda ser segmentada y a su vez obtener la escala de la imagen automáticamente como la conversión píxel-área. Esta función recibe como parámetros:

- **path:** Ruta de la imagen a preparar.
- **size:** Tamaño de la imagen a preparar, por defecto es (1024,1024).
- **noise:** Booleano que indica si se debe aplicar eliminación de ruido a la imagen, por defecto es False.
- **contrast:** Booleano que indica si se debe aplicar un contraste a la imagen, por defecto es False.

Esta función devuelve el output de la función get_scale_area y la imagen preparada para ser segmentada sin la región de la escala.

def separar_regiones

(self,img,segmented=None,path=./prueba/,get_original = False):

Esta función se encarga de separar las diferentes regiones(o bounding boxes) correspondiente a las nanopartículas de una imagen. Una vez realizado esto, se guardan las regiones correspondientes a las nanopartículas y se clasifican. Esta función recibe como parámetros:

- **img:** Matriz numpy que representa la máscara final predicha con nuestro modelo de segmentación y tras aplicar el postprocesado en caso de indicarlo.
- **segmented:** Matriz numpy que representa la máscara final predicha sin aplicar el postprocesado.
- **path:** Ruta donde se guardarán las imágenes de las diferentes regiones de la escala.
- **get_original:** Booleano que indica si se desea contar el área de las nanopartículas en la imagen original o en la imagen con el watershed aplicado.

Esta función devuelve una lista con las áreas, en número de píxeles, de las nanopartículas de la imagen y una máscara con las nanopartículas segmentadas por colores en función de su clase.

def watershed(self,predicted_mask,img):

Esta función se encarga de aplicar el algoritmo de watershed a la máscara final predicha por el modelo de segmentación. Esta función recibe como parámetros:

- **predicted_mask:** Matriz numpy que representa la máscara final predicha por el modelo de segmentación.
- **img:** Matriz numpy que representa la imagen original.

Esta función devuelve la imagen original y la máscara final predicha por el modelo de segmentación tras aplicar el algoritmo de watershed.

```
def get_histogram(self,particles_sizes,image,color_mask):
```

Esta función se encarga de obtener el histograma de las áreas de las nanopartículas de una imagen. Esta función recibe como parámetros:

- **particles_sizes**: Matriz que representa los diferentes tamaños por tipo de nanopartícula que existen en la imagen.
- **image**: Matriz numpy que representa la imagen original.
- **color_mask**: Máscara creada, a partir de la máscara predicha, tras la clasificación de las nanopartículas, donde cada clase está representada por un color.

Esta función devuelve el objeto de la clase Figure de matplotlib que representa el histograma multiclasé final de la imagen.

```
def predict_mask(self,image):
```

Esta función se encarga de predecir usando, el modelo de segmentación, la máscara final de una imagen.

Esta función recibe como parámetros:

- **image**: Matriz numpy que representa la imagen a segmentar.

Esta función devuelve un tensor de numpy de 3 canales que tras aplicar la función **obtain_mask** se convertirá en la máscara final.

```
def get_areas(self,white_pixels,ratio):
```

Esta función se encarga de realizar la conversión de píxeles a área de las nanopartículas de una imagen. Esta función recibe como parámetros:

- **white_pixels**: Lista de listas, cada una por clase a predecir, con el número de píxeles blancos de cada una de las nanopartículas de la imagen.
- **ratio**: Ratio de conversión píxel-área.

Esta función devuelve una lista con el valor real de las áreas de las diferentes nanopartículas de la imagen.

Apéndice C

Arquitectura U-Net implementada

Resumen de la implementación de la red utilizada en el proyecto.

```
1 Model: "UNet"
2 -----
3     Layer (type)           Output Shape        Param #
4     Connected to
5     =====
6     input_1 (InputLayer)    [(None, None, None, 0
7         []
8             3)]
9
10    conv2d (Conv2D)         (None, None, None, 1792
11        ['input_1[0][0]']
12            64)
13
14    batch_normalization (BatchNorm (None, None, None, 256
15        ['conv2d[0][0]']
16        alization)           64)
17
18    conv2d_1 (Conv2D)        (None, None, None, 36928
19        ['batch_normalization[0][0]']
20            64)
21
22    batch_normalization_1 (BatchNo (None, None, None, 256
23        ['conv2d_1[0][0]']
24        rmalization)          64)
25
26    max_pooling2d (MaxPooling2D) (None, None, None, 0
27        ['batch_normalization_1[0][0]']
28            64)
29
30    dropout (Dropout)        (None, None, None, 0
31        ['max_pooling2d[0][0]']
32            64)
```

```

25
26 conv2d_2 (Conv2D)           (None, None, None,    73856
27   ['dropout [0] [0]',]
28                                         128)
29
30 batch_normalization_2 (BatchNo (None, None, None,    512
31   ['conv2d_2 [0] [0]',]
32   rmalization)             128)
33
34 conv2d_3 (Conv2D)           (None, None, None,    147584
35   ['batch_normalization_2 [0] [0]',]
36                                         128)
37
38 batch_normalization_3 (BatchNo (None, None, None,    512
39   ['conv2d_3 [0] [0]',]
40   rmalization)             128)
41
42 max_pooling2d_1 (MaxPooling2D) (None, None, None,    0
43   ['batch_normalization_3 [0] [0]',]
44                                         128)
45
46 dropout_1 (Dropout)         (None, None, None,    0
47   ['max_pooling2d_1 [0] [0]',]
48                                         128)
49
50 conv2d_4 (Conv2D)           (None, None, None,    295168
51   ['dropout_1 [0] [0]',]
52                                         256)
53
54 batch_normalization_4 (BatchNo (None, None, None,    1024
55   ['conv2d_4 [0] [0]',]
56   rmalization)             256)
57
58 conv2d_5 (Conv2D)           (None, None, None,    590080
59   ['batch_normalization_4 [0] [0]',]
60                                         256)
61
62 batch_normalization_5 (BatchNo (None, None, None,    1024
63   ['conv2d_5 [0] [0]',]
64   rmalization)             256)
65
66 max_pooling2d_2 (MaxPooling2D) (None, None, None,    0
67   ['batch_normalization_5 [0] [0]',]
68                                         256)
69
70 dropout_2 (Dropout)         (None, None, None,    0
71   ['max_pooling2d_2 [0] [0]',]
72                                         256)
73
74 conv2d_6 (Conv2D)           (None, None, None,    1180160
75   ['dropout_2 [0] [0]',]

```

```

63                                         512)
64
65     batch_normalization_6 (BatchNo  (None, None, None,    2048
66         ['conv2d_6 [0] [0]'])
67         rmalization)           512)
68
69     conv2d_7 (Conv2D)          (None, None, None,    2359808
70         ['batch_normalization_6 [0] [0]'])
71         512)
72
73     batch_normalization_7 (BatchNo  (None, None, None,    2048
74         ['conv2d_7 [0] [0]'])
75         rmalization)           512)
76
77     max_pooling2d_3 (MaxPooling2D) (None, None, None,    0
78         ['batch_normalization_7 [0] [0]'])
79         512)
80
81     dropout_3 (Dropout)        (None, None, None,    0
82         ['max_pooling2d_3 [0] [0]'])
83         512)
84
85     conv2d_8 (Conv2D)          (None, None, None,    4719616
86         ['dropout_3 [0] [0]'])
87         1024)
88
89     batch_normalization_8 (BatchNo  (None, None, None,    4096
90         ['conv2d_8 [0] [0]'])
91         rmalization)           1024)
92
93     conv2d_9 (Conv2D)          (None, None, None,    9438208
94         ['batch_normalization_8 [0] [0]'])
95         1024)
96
97     batch_normalization_9 (BatchNo  (None, None, None,    4096
98         ['conv2d_9 [0] [0]'])
99         rmalization)           1024)
100

```

```
101    conv2d_10 (Conv2D)          (None, None, None,      4719104
102        ['dropout_4[0][0]',]
103
104    conv2d_11 (Conv2D)          (None, None, None,      2359808
105        ['conv2d_10[0][0]',]
106
107    conv2d_transpose_1 (Conv2DTran (None, None, None,     1179904
108        ['conv2d_11[0][0]',]
109        pose)                  256)
110
111    concatenate_1 (Concatenate) (None, None, None,      0
112        ['conv2d_transpose_1[0][0]',]
113
114    dropout_5 (Dropout)        (None, None, None,      0
115        ['concatenate_1[0][0]',]
116
117    conv2d_12 (Conv2D)          (None, None, None,      1179904
118        ['dropout_5[0][0]',]
119
120    conv2d_13 (Conv2D)          (None, None, None,      590080
121        ['conv2d_12[0][0]',]
122
123    conv2d_transpose_2 (Conv2DTran (None, None, None,     295040
124        ['conv2d_13[0][0]',]
125        pose)                  128)
126
127    concatenate_2 (Concatenate) (None, None, None,      0
128        ['conv2d_transpose_2[0][0]',]
129
130    dropout_6 (Dropout)        (None, None, None,      0
131        ['concatenate_2[0][0]',]
132
133    conv2d_14 (Conv2D)          (None, None, None,      295040
134        ['dropout_6[0][0]',]
135
136    conv2d_15 (Conv2D)          (None, None, None,      147584
137        ['conv2d_14[0][0]',]
138
139
```

```

137     conv2d_transpose_3 (Conv2DTran  (None, None, None,    73792
138         ['conv2d_15 [0] [0]'])
139         spose)                      64)
140
140     concatenate_3 (Concatenate)   (None, None, None,    0
141         ['conv2d_transpose_3 [0] [0]',           128)
141         'batch_normalization_1 [0] [0]')
142
142     dropout_7 (Dropout)        (None, None, None,    0
143         ['concatenate_3 [0] [0]'])
144         128)
145
146     conv2d_16 (Conv2D)          (None, None, None,    73792
147         ['dropout_7 [0] [0]'])
148         64)
148
149     conv2d_17 (Conv2D)          (None, None, None,    36928
150         ['conv2d_16 [0] [0]'])
151         64)
151
152     conv2d_18 (Conv2D)          (None, None, None,    1731
153         ['conv2d_17 [0] [0]'])
154         3)
154
155 =====
156 Total params: 34,530,883
157 Trainable params: 34,522,947
158 Non-trainable params: 7,936
159 -----

```

Listing C.1: Arquitectura implementada de la red U-Net

Bibliografía

- [1] Khan, Y., Sadia, H., Ali Shah, et al. (2022). Classification, Synthetic, and Characterization Approaches to Nanoparticles, and Their Applications in Various Fields of Nanotechnology: A Review. *Catalysts*, 12, 1386. <https://doi.org/10.3390/catal12111386>
- [2] Nida M. Zaitoun, Musbah J. Aqel. (2015). Survey on Image Segmentation Techniques. *Procedia Computer Science*, Vol. 65, . 797-806, <https://doi.org/10.1016/j.procs.2015.09.027>.
- [3] Xiaonan Luo, Aakash Varambhia, Weixin Song, Dogan Ozkaya, Sergio Lozano-Perez, Peter D. Nellist. (2022). High-precision atomic-scale strain mapping of nanoparticles from STEM images, *Ultramicroscopy*, Vol.239, 113561, <https://doi.org/10.1016/j.ultramic.2022.113561>.
- [4] Liu, W. et al. (2016). SSD: Single Shot MultiBox Detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds) *Computer Vision – European Conference on Computer Vision(ECCV) 2016. Lecture Notes in Computer Science*, vol 9905. https://doi.org/10.1007/978-3-319-46448-0_2.
- [5] Girshick, Ross & Donahue, Jeff & Darrell, Trevor & Malik, Jitendra. (2013). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition(CVPR)*. <https://doi.org/10.1109/CVPR.2014.81>.
- [6] Redmon, Joseph & Divvala, Santosh & Girshick, Ross & Farhadi, Ali. (2016). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition(CVPR)*. 779-788. <https://doi.org/10.1109/CVPR.2016.91>.
- [7] Garcia-Garcia, Alberto & Orts, Sergio & Oprea, Sergiu & Villena Martínez, Víctor & Rodríguez, José. (2017). A Review on Deep Learning Techniques Applied to Semantic Segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition(CVPR)*. <https://doi.org/10.48550/arXiv.1704.06857>

- [8] He, Kaiming & Gkioxari, Georgia & Dollar, Piotr & Girshick, Ross. (2017). Mask R-CNN. *International Conference on Computer vision (ICCV)*. 2980-2988. <https://doi.org/10.1109/ICCV.2017.322>.
- [9] Ronneberger, Olaf & Fischer, Philipp & Brox, Thomas. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. LNCS. 9351. 234-241. https://doi.org/10.1007/978-3-319-24574-4_28.
- [10] Sharma, Neha & Jain, Vibhor & Mishra, Anju. (2018). An Analysis Of Convolutional Neural Networks For Image Classification. *Procedia Computer Science*. 132. 377-384. <https://doi.org/10.1016/j.procs.2018.05.198>.
- [11] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations (ICLR)*, 1–14. <https://doi.org/10.48550/arXiv.1409.1556>
- [12] S.H. Shabbeer Basha, Shiv Ram Dubey, Viswanath Pulabaigari, Snehasis Mukherjee. (2020). Impact of fully connected layers on performance of convolutional neural networks for image classification. *Neurocomputing*, Vol. 378. 112-119, ISSN 0925-2312, <https://doi.org/10.1016/j.neucom.2019.10.008>.
- [13] O. M. Parkhi, A. Vedaldi, A. Zisserman, C. V. Jawahar. (2012). Cats and Dogs. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3498-3505, <https://doi.org/10.1109/CVPR.2012.6248092>.
- [14] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6), 141–142, <https://doi.org/10.1109/MSP.2012.2211477>.
- [15] N. Siddique, S. Paheding, C. P. Elkin and V. Devabhaktuni, (2021), U-Net and Its Variants for Medical Image Segmentation: A Review of Theory and Applications, in *IEEE Access*, vol. 9, . 82031-82057, <https://doi.org/10.1109/ACCESS.2021.3086020>.
- [16] Zhou, Zongwei & Rahman Siddiquee, Md Mahfuzur & Tajbakhsh, Nima & Liang, Jianming. (2018). UNet++: A Nested U-Net Architecture for Medical Image Segmentation: 4th International Workshop, DLMIA, and 8th International Workshop, ML-CDS, Held in Conjunction with MICCAI, Proceedings. https://doi.org/10.1007/978-3-030-00889-5_1.
- [17] B. Baheti, S. Innani, S. Gajre and S. Talbar. (2020). EffUNet: A Novel Architecture for Semantic Segmentation in Unstructured Environment, *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, . 1473-1481, <https://doi.org/10.1109/CVPRW50498.2020.00187>.

- [18] F. Zhuang et al. (2021). A Comprehensive Survey on Transfer Learning, in Proceedings of the IEEE, vol. 109, no. 1, . 43-76, <https://doi.org/10.1109/JPROC.2020.3004555>.
- [19] Laurent Najman, Michel Schmitt. (1994). Watershed of a Continuous Function. Signal Processing, 38 (1), .99-112. [https://doi.org/10.1016/0165-1684\(94\)90059-0](https://doi.org/10.1016/0165-1684(94)90059-0).
- [20] M. F. Liz, A. V. Nartova, A. V. Matveev and A. G. Okunev. (2020). Using Computer Vision and Deep Learning for Nanoparticle Recognition on Scanning Probe Microscopy Images: Modified U-net Approach, Science and Artificial Intelligence conference (S.A.I.ence). 13-16, <https://doi.org/10.1109/S.A.I.ence50533.2020.9303184>.
- [21] Zelenka, Claudio & Kamp, Marius & Strohm, Kolja & Kadoura, Akram & Johny, Jacob & Koch, Reinhard & Kienle, Lorenz. (2023). Automated classification of nanoparticles with various ultrastructures and sizes via deep learning. Ultramicroscopy. 246. 113685. <https://doi.org/10.1016/j.ultramic.2023.113685>.
- [22] Pérez-Bosch, Emilio & Romero-Zaliz, Rocío & Pérez, Eduardo & Kalishettyhalli Mahadevaiah, Mamathamba & Reuben, John & Schubert, Markus & Jiménez-Molinos, F. & Roldan, Juan & Wenger, Ch. (2021). Toward Reliable Compact Modeling of Multilevel 1T-1R RRAM Devices for Neuromorphic Systems. Electronics. 10. 645. <https://doi.org/10.3390/electronics10060645>.
- [23] Ciloglu, F.U., Caliskan, A., Saridag, A.M. et al. (2021). Drug-resistant *Staphylococcus aureus* bacteria detection by combining surface-enhanced Raman spectroscopy (SERS) and deep learning techniques. Sci Rep 11, 18444. <https://doi.org/10.1038/s41598-021-97882-4>.