



Segmentation of a laser trace using Deep Convolutional Neural Networks

A Degree Thesis Submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya

In partial fulfilment of the requirements for the degree in Bachelor's
degree in Telecommunications Technologies and Services Engineering

Author

Joan Manel Cárdenas Palenzuela

Advisors

Jan Kybic

Barcelona, September 2022

Segmentation of a laser trace using Deep Convolutional Neural Networks

ABSTRACT

The Deep Neural Networks have shown some excellent results in the problems of classification and segmentation of images. One of the areas where this tool has been applied the most is biomedical image processing.

From all the Deep Neural Networks, in this thesis we will study the use of the Convolutional Neural Networks (CNN) and given the fact that the issue raised in this thesis is a segmentation problem, we will stand within the framework of the U-net, a model of architecture of Convolutional Neural Networks with extraordinary results in the field of semantic segmentation of biomedical images.

In the first part, we will use the project created by the department of Biomedical Imaging Algorithms of the Czech technical university (CTU) to get in touch with the architecture of the model, optimize it and obtain the best results possible decreasing the computational cost and the weights of its operations. Finally, we will build our own model with less complexity in order to implement some of the improvements that were not able to be introduced in the first part.

Segmentació d'una traça làser mitjançant Xarxes Neuronals Convolucionals Profundes

RESUM

Les Xarxes Neuronals Profundes han demostrat excel·lents resultats en els problemes de classificació i segmentació d'imatges. Un dels sectors on més s'ha aplicat aquest recurs és en el processament d'imatges biomèdiques.

De totes les Xarxes Neuronals Profundes, en aquesta tesi s'estudiarà l'ús de les Xarxes Neuronals Convolucionals (CNN) i, donat que la qüestió que ens plantejarem serà de segmentació, ens situarem en el marc de U-net, un model d'arquitectura de Xarxes Neuronals Convolucionals amb extraordinaris resultats en l'àmbit de la segmentació semàntica d'imatges biomèdiques.

En una primera fase, utilitzarem el projecte creat pel departament de Biomedical Imaging Algorithms de la Escola Tècnica Txeca de Praga (CTU), per familiaritzar-nos amb el model arquitectònic, optimitzar-lo i obtenir els millors resultats possibles disminuint el cost computacional i el volum que ocupen les seves operacions. Finalment, construirem el nostre propi model de menor complexitat per poder implementar alguna de les millores que no s'han pogut introduir en la primera fase.

Segmentación de una traza láser utilizando Redes Neuronales Convolucionales Profundas

RESUMEN

Las Redes Neuronales Profundas han demostrado unos excelentes resultados en los problemas de clasificación y segmentación de imágenes. Uno de los sectores donde más se ha aplicado este recurso es en el de procesado de imágenes biomédicas.

De entre todas las Redes Neuronales Profundas, en esta tesis estudiaremos el uso de Redes Neuronales Convolucionales (CNN) y dado que la cuestión planteada es un problema de segmentación, nos situaremos dentro del marco de U-net, un modelo de arquitectura de Redes Neuronales Convolucionales con extraordinarios resultados en el ámbito de la segmentación semántica de imágenes biomédicas.

En una primera fase, utilizaremos el proyecto creado por el departamento de *Biomedical Imaging Algorithms* de la Universidad Técnica de la República Checa (CTU) para familiarizarnos con el modelo arquitectónico, optimizarlo y obtener los mejores resultados posibles disminuyendo el coste computacional y el tamaño que ocupan sus operaciones. Finalmente, construiremos nuestro propio modelo de menor complejidad para implementar algunas de las mejoras que no se han podido introducir en la primera fase.

Dedicatoria

A Martina

Agradecimientos

Me gustaría agradecer a la CTU de Praga, por darme la oportunidad de vivir la experiencia de un Erasmus allí y poder realizar este proyecto.

También agradecer al departamento de *Biomedical Imaging Algorithms* y especialmente a mi supervisor Jan Kybic por ayudarme y facilitarme el acceso a los servicios de la CTU.

Por último a Cristina , a mi familia y sobre todo a Martina por su apoyo incondicional y su ayuda para realizar esta tesis.

Historial de revisión y registro de aprobación

Revisión	Fecha	Propósito
0	20/08/2022	Document creation
1	29/09/2022	Document revision
2	03/10/2022	Document revision
3	11/09/2022	Document approbation

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Joan Manel Cárdenas Palenzuela	joan.manel.cardenas@estudiantat.upc.edu
Jan Kybic	kybic@fel.cvut.cz
Veronica Vilaplana Besler	veronica.vilaplana@upc.edu

Written by:		Reviewed and approved by:	
Date	03/10/2022	Date	11/10/2022
Name	Joan Manel Cárdenas Palenzuela	Name	Jan Kybic
Position	Project Author	Position	Project Supervisor

Tabla de contenidos

Abstract	1
Resum	2
Resumen	3
Dedicatoria	4
Agradecimientos	5
Historial de revisión y registro de aprobación	6
Tabla de contenidos	7
Listado de figuras	10
Listado de tablas	12
1. Introducción	13
1.1. Objetivo	13
1.2. Requisitos y especificaciones	15
1.3. Metodología y procedimiento	15
1.4. Work plan y Diagrama de Gantt	16
1.5. Incidencias	19
2. State of the art de la tecnología utilizada o aplicada en esta tesis	21
2.1. Aprendizaje profundo	21
2.2. Convolutional Neural Networks	21
2.3. <i>Image segmentation</i>	22
2.3.1. Segmentación Semántica	23
3. Metodología y procedimientos	24

3.1. Softwares y Librerías	24
3.2. Arquitectura del modelo	25
3.2.1. U-net	25
3.3. Pre-procesado	27
3.3.1. Dataset	27
3.3.2. Ecualización del histograma	28
3.3.3. Recorte de las imágenes	28
3.4. Hiperparámetros	29
3.4.1. Batch size	29
3.4.2. Epochs	30
3.4.3. Optimizer	30
3.5. Métricas de evaluación	32
3.5.1. Accuracy	32
3.5.2. Índice de Jaccard	33
3.5.3. Dice coefficient	33
3.5.4. Relative Area Error	34
3.6. Loss functions	34
3.6.1. Binary cross entropy	34
3.6.2. Dice coefficient loss	35
4. Resultados	36
4.1. Configuraciones de las simulaciones	36
4.2. Simulación uno	37
4.3. Simulaciones dos, tres y cuatro	39
4.4. Simulaciones cinco, seis y siete	46

4.5. Simulaciones ocho y nueve	52
4.6. Simulación diez	56
4.7. Sumario de resultados	59
5. Presupuesto	62
6. Conclusiones y futuras vías de desarrollo	63
Bibliografía	65
Appendices	68
Glosario	74

Listado de figuras

1.1. Ejemplo de dos imágenes donde se captura el impacto del rayo láser	14
1.2. Diagrama de Gantt	19
2.1. Izquierda: CNN; Derecha: FCN	21
2.2. Esquema general FCN	22
2.3. Diferencia de características que se tienen en cuenta en las distintas capas de una CNN	22
2.4. <i>Instance segmentation</i> y <i>semantic segmentation</i>	23
3.1. Estructura general del modelo U-Net	25
3.2. Funcionamiento de la activación ReLU	26
3.3. Funcionamiento del MaxPooling	26
3.4. Ejemplos de imágenes del conjunto de datos	27
3.5. Ejemplo de las tres componentes YUV del espacio de color YUV	28
3.6. Imagen antes y después de ecualizar su histograma, y después de normalizar	28
3.7. Evolución del <i>learning rate</i> según el tamaño del gradiente de descenso	31
3.8. <i>Stochastic gradient descent</i>	31
3.9. Clasificación de los píxeles resultantes de una simulación	32
3.10. Ecuación IoU	33
4.1. Segmentación ideal	37
4.2. Resultados simulación 1	38
4.3. Gráfica de la <i>accuracy</i> simulación 1	38
4.4. Resultados simulación 2	42
4.5. Resultados simulación 3	43
4.6. Gráfica de la <i>accuracy</i> simulación 3	44
4.7. Gráfica de la <i>loss function</i> simulación 3	44
4.8. Resultados simulación 4	45

4.9. Gráfica de la <i>accuracy</i> simulación 4	45
4.10. Gráfica de la <i>loss function</i> simulación 4	46
4.11. Gráfica del IoU y de la <i>loss function</i> del batch size 8	47
4.12. Gráfica del IoU y de la <i>loss function</i> del batch size 16	47
4.13. Gráfica del IoU y de la <i>loss function</i> del batch size 64	47
4.14. Resultados simulación 5	49
4.15. Resultados simulación 6	50
4.16. Gráfica del IoU simulación 6	50
4.17. Gráfica de la <i>loss function</i> simulación 6	51
4.18. Resultados simulación 7	51
4.19. Gráfica del IoU simulación 7	52
4.20. Gráfica de la <i>loss function</i> simulación 7	52
4.21. Resultados simulación 8	54
4.22. Gráfica del <i>dice coefficient</i> simulación 8	54
4.23. Gráfica de la <i>loss function</i> simulación 8	55
4.24. Resultados simulación 9	55
4.25. Gráfica del <i>dice coefficient</i> simulación 9	56
4.26. Gráfica de la <i>loss function</i> simulación 9	56
4.27. Resultados simulación 10	58
4.28. Gráfica del IoU simulación 10	59
4.29. Gráfica de la <i>loss function</i> simulación 6	59
A.1. Ejemplo de la colección de imágenes de <i>ground truth segmentation boundary</i>	68
A.2: Ejemplo ecualización histograma	69
A.3. Gráficas de evolución de las métricas de evaluación y la <i>loss function</i>	73
A.4. Comparativa gráficas de <i>cross entropy loss</i>	73

Listado de tablas

4.1. Simulación 1	37
4.2. Simulación 2	40
4.3. Simulación 3	40
4.4. Simulación 4	41
4.5. Simulación 5	48
4.6. Simulación 6	48
4.7. Simulación 7	49
4.8. Simulación 8	53
4.9. Simulación 9	53
4.10. Simulación 10	57
4.11. Resultados de las simulaciones	61
A.1. Dice score and relative area error at 9*	69
A.2. Resumen de las métricas modificadas en las seis simulaciones	71
A.3. Resumen del espacio de memoria necesario en las seis simulaciones	71
A.4. Resumen de las métricas resultantes en las seis simulaciones	72
A.5. Resultados de las métricas de evaluación durante diferentes repeticiones	74

1

Introducción

En el ámbito de la biomedicina, el diagnóstico a partir de imágenes médicas no sólo depende de la definición inicial de la imagen sino también de la interpretación de la misma. Si bien la obtención de imágenes médicas ha mejorado mucho a lo largo de los años gracias a los avances tecnológicos, no ha sido hasta estos últimos años que el proceso de interpretación de imágenes biomédicas se ha visto beneficiado de estos avances. A pesar de que la introducción de maquinaria encargada de realizar la parte analítica del diagnóstico, este queda sujeta a la ética, es cierto que la introducción de dichas máquinas como soporte artificial a la interpretación realizada por un profesional es un gran avance y puede favorecer a nuevos planteamientos que no podrían producirse de otra manera.

1.1. Objetivo

Dentro del marco de la inteligencia artificial, en esta tesis utilizamos un experimento dónde se efectuaron impactos de un rayo láser sobre diferentes muestras a nivel microscópico. Estos, se capturaron en imágenes que fueron enviadas al departamento de *Biomedical Imaging Algorithms* de la Universidad Técnica de la República Checa (CTU) con el objetivo de diferenciar qué partes de la imagen eran impactos del rayo láser y qué partes eran muestras del fondo. A título ilustrativo, un ejemplo de esta técnica de diferenciación lo encontramos en la FIGURA 1.1. En la primera fila de esta, apreciamos el impacto del rayo láser marcado en el centro de las imágenes A y B, representados de color verde en la segunda fila de la figura. Por otro lado, podemos observar como hay muchos elementos en el fondo, algunos marcados de color azul en la segunda fila. Estos elementos suponen un problema para ser diferenciados con el impacto del rayo. Para resolver esta cuestión, Jan Hering y Jan Kybic, miembros de este departamento, realizaron un algoritmo capaz de segmentar las imágenes y diferenciar entre impacto y fondo.

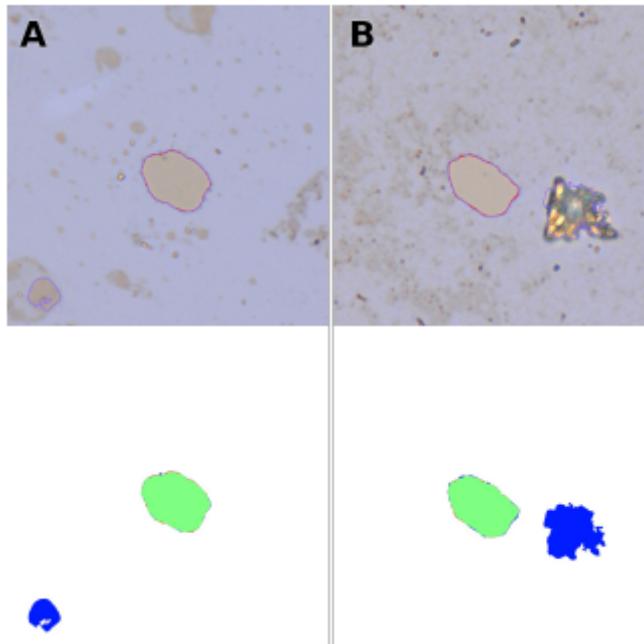


FIGURA 1.1: Ejemplo de dos imágenes donde se captura el impacto del rayo láser

El objetivo inicial de esta tesis era la optimización del código desarrollado por Jan Kybic y Jan Hering minimizando la cantidad de memoria necesaria para realizar este propósito.

Como comentaremos más adelante en esta tesis, finalmente decidimos realizar un código desde cero debido a las dificultades que presentaba el original. Es sobre este donde se realizan todas las optimizaciones pertinentes. De esta manera, los objetivos finalmente se fijaron en los siguientes:

- ❖ Estudiar el uso de las técnicas *Deep Learning* en la clasificación y segmentación aplicado a imágenes biomédicas.
- ❖ Analizar la arquitectura del modelo de CNN U-net utilizando el ya creado por Jan Kybic y Jan Hering y aprender a diseñar uno propio desde cero.
- ❖ Aprender el lenguaje de programación Python así como algunas de las librerías de Redes Neuronales Profundas más importantes como son Pytorch, utilizada tanto para el modelo del departamento de Biomedical Imaging Algorithms o Keras, como para el modelo creado desde cero para este proyecto.
- ❖ Aplicar métodos y algoritmos de optimización de hiperparámetros de los modelos de segmentación.

1.2. Requisitos y especificaciones

El primer instrumento necesario para este proyecto es el lenguaje de programación Python, ya que los ficheros realizados y estudiados en esta tesis son de Python prácticamente en su totalidad. De la misma manera, Pytorch, TensorFlow y Keras son las tres librerías de Python que más se han usado.

Por otro lado, ha sido esencial adquirir unos conocimientos avanzados de Machine Learning (ML) [9] y de las Redes Neuronales Profundas (DNN), ya que hemos utilizado las librerías de Python para poder codificar los conceptos relacionados con ML y DNN.

Finalmente, dado que este proyecto se ha realizado en el marco de la CTU y que inicialmente la memoria necesaria para ejecutar la segmentación era muy elevada, hemos requerido el uso de GPUs de gran potencia, más específicamente las proporcionadas por la propia CTU. Por este motivo, hemos necesitado unos conocimientos de protocolos de acceso remoto, SSH. En este caso aplicaciones como PuTTY o WinSCP han sido de gran importancia.

1.3. Metodología y procedimiento

Este proyecto se orientó inicialmente a desarrollar el modelo de Jan Hering y Jan Kybic [26], concretamente, se planteó la posibilidad de introducir unas optimizaciones en los parámetros de entrenamiento y en el modelo.

Esta forma de trabajar no fue realizable como comentaremos en el apartado de incidencias. Ante estas dificultades que no permitían avanzar en el sentido que nos habíamos propuesto, redefinimos la estrategia para, utilizando como base el modelo creado por Jan Hering y Jan Kybic, ir más allá. En este sentido, la metodología que hemos seguido finalmente es:

Una primera fase de aprendizaje y estudio del código de Jan Hering y Jan Kybic con un seguido de simulaciones y pruebas donde intentábamos obtener resultados coherentes para poder realizar una optimización de su modelo.

Una segunda fase donde realizamos un modelo nuevo teniendo en cuenta todas las dificultades encontradas en el primer modelo y adaptándolo para no volver a caer en las mismas situaciones. [28]

Dado que, en todo caso, el modelo de Jan Hering y Jan Kybic, es el punto de partida, es necesario exponer, aunque sea brevemente, los elementos más relevantes del abstract [26] de Jan Hering y Jan Kybic, a los efectos de nuestro trabajo posterior. Es por eso que en los apéndices de esta tesis encontramos un sumario del proyecto inicial juntamente con análisis de las simulaciones que realizamos.

1.4. Work plan y Diagrama de Gantt

Es obvio que con el forzado cambio de metodología también el Work plan se vió alterado. Crear un modelo desde cero dista claramente de la introducción de modificaciones en un código existente. Pero sin el primer intento de mejora fallido, no se hubiera visto la necesidad de proponer un modelo nuevo desde cero para mejorar la tecnología utilizada. Por ello, se reportan los dos Work plan.

WP1. Documentación

Proyecto: Documentación	WP ref: WP-1	
Constituyente mayor: Creación de documentos	Hoja 1 of 9	
Descripción breve: Elaboración de los tres documentos entregables en referencia a esta tesis, a parte de los informes presentados regularmente al supervisor.	Fecha de inicio: 07/02/22 Fecha de fin: 28/09/22	
Tarea T1: Project Plan and Work Plan	Entregables:	Fechas:
Tarea T2: Project Critical Review		
Tarea T3: Final Report		

WP2. Investigación y adaptación al proyecto base

Proyecto: Investigación y adaptación al proyecto base.	WP ref: WP-2	
Constituyente mayor: Estudio y aprendizaje	Hoja 2 of 9	
Descripción breve: Leer documentos relacionados con la temática que se nos ha planteado en esta tesis para poder entender el proyecto que se nos facilitó inicialmente.	Fecha de inicio: 10/02/22 Fecha de fin: 30/03/22	
Tarea T1: Lecturas relacionadas con CNN	Entregables:	Fechas:
Tarea T2: Aprendizaje sobre U-net		
Tarea T3: Lectura y análisis del report del proyecto base realizado por Jan Hering y Jan Kybic		
Tarea T4: Lecturas relacionadas con Python y Pytorch		

WP3. Ajustes y experimentación con el proyecto base

Proyecto: Ajustes y experimentación con el proyecto base	WP ref: WP-3	
Constituyente mayor: Telemática y experimentación	Hoja 3 of 9	
Descripción breve: Integración del código del proyecto base en los servidores de la CTU y adaptación de los requisitos del proyecto a las especificaciones de los servidores. Una vez hecha la integración, empezar a ejecutar el código para realizar experimentos	Fecha de inicio: 01/03/22 Fecha de fin: 23/04/22	

Tarea T1: Integrar el proyecto en los servidores para poderlo ejecutar Tarea T2: Realizar experimentos con diferentes valores para poder llevar a cabo un análisis de los resultados.	Entregables:	Fechas:
--	--------------	---------

WP4. Análisis de resultados

Proyecto: Análisis de resultados Constituyente mayor: Analítico Descripción breve: A medida que vamos realizando experimentos, analizaremos los resultados de estos. Determinaremos si los resultados tienen lógica según lo previsto o si por lo contrario han surgido eventos inesperados Tarea T1: Analizar los resultados de cada experimento Tarea T2: Comparar los resultados de cada experimentos con las hipótesis realizadas antes de estos y con los otros experimentos. Tarea T3: En caso de incongruencias, determinar el motivo de estas.	WP ref: WP-4 Hoja 4 of 9 Fecha de inicio: 15/03/22 Fecha de fin: 05/05/22 Entregables: Fechas:
---	--

WP5. Exploración de nuevas vías de desarrollo

Proyecto: Exploración de nuevas vías de desarrollo Constituyente mayor: Investigación Descripción breve: Debido a la complejidad del código y a la gran cantidad de resultados incoherentes, exploramos alternativas para poder alcanzar los objetivos. Tarea T1: Debatir si es mejor intentar avanzar con el proyecto base o empezar un modelo desde cero. Tarea T2: Explorar diferentes modelos y analizar su posible adecuación al problema de segmentación planteado en esta tesis.	WP ref: WP-5 Hoja 5 of 9 Fecha de inicio: 01/05/22 Fecha de fin: 29/05/22 Entregables: Fechas:
--	--

WP6. Creación de modelo propio

Proyecto: Creación de modelo propio Constituyente mayor: Programación Descripción breve: Una vez hemos decidido crear un nuevo modelo, ensamblar desde cero utilizando todos los conocimientos adquiridos durante los últimos meses. Tarea T1: Construir un modelo U-net totalmente funcional con unos resultados razonablemente buenos.	WP ref: WP-6 Hoja 6 of 9 Fecha de inicio: 25/06/22 Fecha de fin: 30/07/22 Entregables: Fechas:
---	--

WP7. Experimentación y optimización del modelo

Proyecto: Experimentación y optimización del modelo	WP ref: WP-7	
Constituyente mayor: Programación y desarrollo	Hoja 7 of 9	
Descripción breve: Testeamos el nuevo modelo, y a medida que vamos obteniendo resultados favorables, vamos introduciendo nuevos parámetros a optimizar.	Fecha de inicio: 20/07/22 Fecha de fin: 05/09/22	
Tarea T1: Probar el funcionamiento del nuevo modelo, comprobar con diferentes simulaciones si los resultados obtenidos son coherentes	Entregables:	Fechas:
Tarea T2: Almacenar todos los resultados para su posterior análisis.		

WP8. Resultados y análisis

Proyecto: Resultados y análisis	WP ref: WP-8	
Constituyente mayor: Comparación de resultados	Hoja 8 of 9	
Descripción breve: Realizamos una comparación entre los diferentes experimentos que hemos realizado para determinar cuáles han sido los hiperparámetros y las condiciones que nos han llevado a obtener los mejores resultados.	Fecha de inicio: 09/08/19 Fecha de fin: 20/09/22	
Tarea T1: Comparación y estudio entre las simulaciones realizadas con diferentes condiciones	Entregables:	Fechas:
Tarea T2: Parametrizar el máximo número de variables para poder determinar de mejor forma como se obtienen los resultados óptimos.		

WP9. Entrega de documentos

Proyecto: Entrega de documentos	WP ref: WP-9	
Constituyente mayor: Documentación	Hoja 9 of 9	
Descripción breve:	Fecha de inicio: 03/08/22 Fecha de fin: 28/09/19	
Tarea T1: Supervisor's assessment	Entregables:	Fechas:
Tarea T2: Documentation delivery		
Tarea T3: Project Presentation		

Diagrama de Gantt

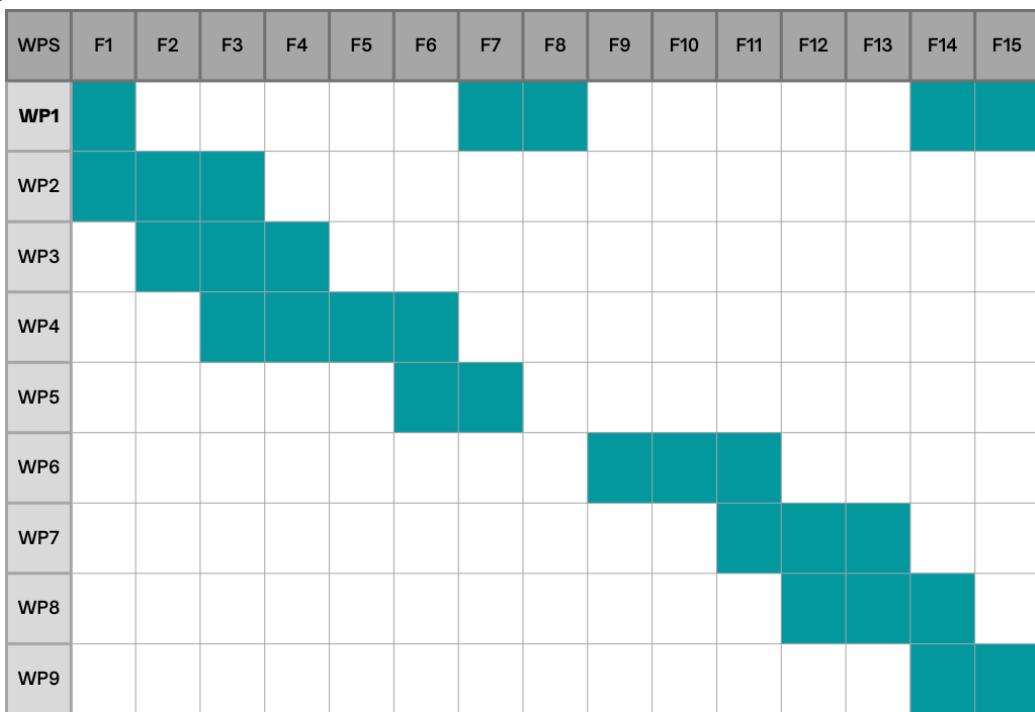


FIGURA 1.2: Diagrama de Gantt

1.5. Incidencias

El primer contratiempo fue debido al volumen de memoria necesaria para realizar la segmentación con el código base de Jan Hering y Jan Kybic. Al tratarse de una segmentación de gran precisión, el tamaño total que ocupaba el modelo era 1408494302.51 MB, con 13,240,162 parámetros entrenables. Debido a esto, la segmentación se debía realizar en un servidor con una GPU, más potente que un ordenador convencional. Es ahí donde se hizo uso de los servidores de la CTU para poder hacer la segmentación mediante una conexión SSH desde las aplicaciones de PuTTY y WinSCP. Una vez lograda la conexión, incluso con los servidores de la CTU, algunas simulaciones no se podían realizar debido al volumen que ocupaban.

Por otro lado, al conectarse a los servidores de la CTU, estos tenían versiones de librerías y packages por defecto y la mayoría de ellas no eran las mismas que las reflejadas en los requisitos del proyecto de Jan Hering y Jan Kybic. El problema surgió cuando al intentar cambiar la versión de las librerías, algunas de las que se necesitaban para el proyecto no se encontraban disponibles y no se podían descargar ya que no disponíamos de los permisos para hacerlo. Por este motivo, muchas de las simulaciones realizadas sin modificar ningún parámetro del código tenían unos resultados muy diferentes a los obtenidos según refleja el abstract [26] de Jan Hering y Jan Kybic.

Pese a este hecho continuamos realizando simulaciones e intentando buscar la manera de conseguir unos resultados similares a los obtenidos por Jan Hering y Jan Kybic. Estas simulaciones se encuentran comentadas en el apéndice de esta tesis.

Estas incidencias nos llevaron a la conclusión de que la metodología planteada no era viable. En la segunda fase de creación de un código desde cero, la experiencia previa nos permitió evitar este tipo de problemas. No surgieron otras por el camino y se pudo concluir el proyecto.

2

State of the art de la tecnología utilizada o aplicada en esta tesis

2.1. Aprendizaje profundo

El aprendizaje profundo (Deep Learning, DL) con redes neuronales es una de las ramas más prometedoras de la inteligencia artificial. Recientemente se ha visto un gran incremento en el uso de este ya que aprovecha al máximo el gran poder computacional de los ordenadores. DL se basa, a muy alto nivel, en realizar tareas de obtención de las características necesarias de una muestra para realizar un razonamiento y extraer todo tipo de información.

Esta tecnología se usa comúnmente en aplicaciones como traducción automática, conducción autónoma, reconocimiento del habla, imagen... [8] Para esta última, una técnica muy extendida y con excelentes resultados es la de Redes Neuronales Convolucionales (Convolutional Neural Networks, CNN).

2.2. Convolutional Neural Networks

Las CNN, son redes neuronales artificiales estructuradas en un sistema de nodos interconectados formados por múltiples capas de neuronas que se usan especialmente en tareas de computer vision.

Un tipo de red concreta dentro de las CNN es la red neuronal totalmente conectada (Fully Convolutional Network, FCN). En la FIGURA 2.1 podemos ver la diferencia entre una red neuronal (izquierda) y una FCN (derecha).

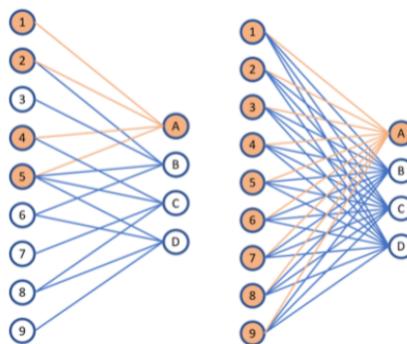


FIGURA 2.1: Izquierda: CNN; Derecha: FCN

Las FCN tienen el problema de que al estar totalmente conectadas, el tamaño de memoria necesario es mucho mayor que para una CNN. Sin embargo, la ventaja

respecto a las CNN es que se pueden usar de forma mucho más general. Imaginemos por ejemplo que todas las imágenes de entrada tuvieran los objetos a segmentar en el mismo lugar y con una forma muy parecida, en este caso sí que podríamos utilizar una CNN, pero en el caso contrario, si las imágenes de entrada tuvieran los objetos de diferentes formas, texturas y situaciones espaciales, los resultados que obtendremos con una CNN serían mucho peor que con una FCN.

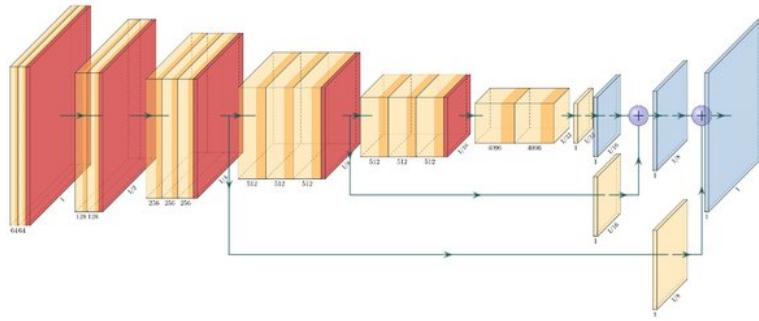


FIGURA 2.2: Esquema general FCN

2.3. Image segmentation

La segmentación de una imagen consiste en la partición de ésta en múltiples segmentos para obtener diferentes características. Si nos fijamos en la estructura general de las FCN (FIGURA 2.2), comparándola con la FIGURA 2.3 donde se utiliza una FCN para la segmentación de imágenes, veremos que la estructura es idéntica y que en cada una de las capas de la FCN utilizamos filtros que ayudan a caracterizar la imagen. En la FIGURA 2.3 se muestran: en la primera capa 96 filtros, cada uno de ellos distinto y estrictamente utilizado para comparar los bordes de los elementos de la imagen; en la capa Conv 3, podemos apreciar 9 filtros, esta vez enfocados en las diferentes texturas.

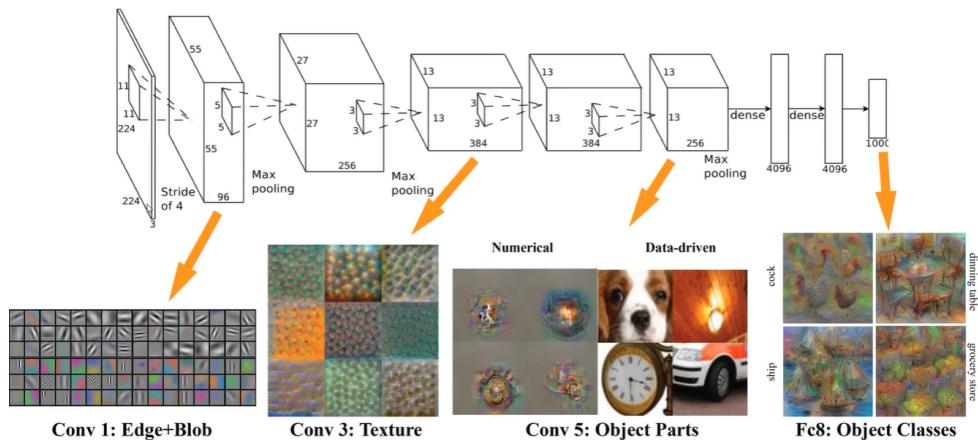


FIGURA 2.3: Diferencia de características que se tienen en cuenta en las distintas capas de una CNN

2.3.1. Segmentación Semántica

Dentro de la segmentación de una imagen, podemos diferenciar entre *instance segmentation* y *semantic segmentation* (SS). La primera trata múltiples objetos dentro de una sola categoría como diferentes entidades, mientras que la segunda, identifica objetos dentro de una misma categoría con la misma entidad. En la FIGURA 2.4 se puede ver un ejemplo.

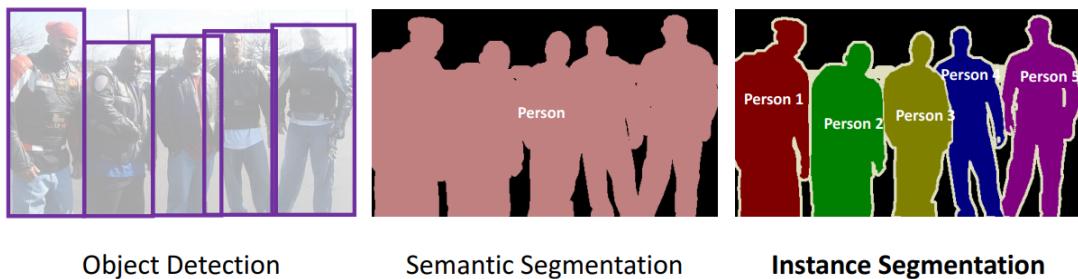


FIGURA 2.4: *Instance segmentation* y *semantic segmentation*

En esta tesis, nos centraremos únicamente en la detección de categorías (impactos de rayos láser), es por ello que trabajaremos con SS, pero en trabajos relacionados con la clasificación, se obtienen igualmente excelentes resultados trabajando con *instance segmentation* en CNN. [4][14]

3

Metodología y procedimientos

En esta sección se comentarán todas las especificaciones de este proyecto, desde las librerías y el software hasta las *loss functions* utilizadas, pasando por los detalles del entrenamiento y la evaluación de los procesos.

3.1. Softwares y Librerías

En el proyecto del departamento de *Biomedical Imaging Algorithms* de la CTU, se utilizaba mayoritariamente Pytorch, una librería de código abierto de aprendizaje automático. Esta librería es usada muy frecuentemente para aplicaciones de visión artificial (*computer vision*, CV), su interfaz más utilizada es la de Python. Sin embargo también tiene interfaz en C++ aunque esta no es tan pulida como la primera.[7]

Para el nuevo modelo que hemos creado, utilizamos Keras, que es otra librería de código abierto de aprendizaje automático para redes neuronales. Esta es capaz de ejecutarse sobre TensorFlow, CNTK o Theano entre otros. Nosotros ejecutaremos sobre TensorFlow. El motivo que nos ha llevado a utilizar esta Keras es el hecho de que esta ofrece más opciones de desarrollo y resulta más fácil exportar el modelo de entrenamiento. Además, su arquitectura es mucho más sencilla y estructural, y la documentación es mucho más extensa que para Pytorch.[18]

TensorFlow fué desarrollado por Google para sistemas capaces de construir y entrenar redes neuronales. En TensorFlow los cálculos se realizan con tensores, los tensores son vectores o matrices de n-dimensiones que representan todo tipo de datos. En nuestro caso, los píxeles de las imágenes.

A parte de Keras y TensorFlow, también cabe destacar otras librerías:

OpenCV es una librería de *computer vision* desarrollada por Intel, en este proyecto la utilizaremos mayoritariamente para el análisis de las imágenes y para modificar las estructuras de éstas (invertir las máscaras, cambiar el espacio de color...). Es importante remarcar que OpenCV trabaja con el espacio de color BRG, en vez del habitual RGB.

Numpy es otra librería de Python y la utilizaremos para la conversión de imágenes en matrices y vectores de grandes dimensiones para poder operar con estos.

La librería de código abierto para Python Pillow la utilizaremos para el manejo de las imágenes de entrada.

Para la creación de gráficas con los resultados de las simulaciones, utilizaremos `Matplotlib`, una librería de generación de gráficos a partir de `arrays` o listas.

Finalmente, la librería para aprendizaje automático de software libre `Scikit-learn`, la usaremos para dividir las imágenes de entrada en grupos para entrenamiento y validación.

3.2. Arquitectura del modelo

La arquitectura utilizada tanto en el modelo del proyecto del departamento de *Biomedical Imaging Algorithms* de la CTU, como para el modelo creado desde cero es *U-net*.[1][6]

3.2.1. U-net

U-net es una CNN simple a la vez que eficaz. Debe su nombre a la forma de U que crea el diseño de su modelo (FIGURA 3.1). Se utiliza para la segmentación semántica, de esta manera cada píxel de la imagen de entrada se clasifica en una clase (impacto del rayo o fondo de la imagen), como hemos visto en el apartado dos.

Fué introducida por Olaf Ronnenberg en el año 2015 para la segmentación semántica de imágenes biomédicas, que es el mismo tema que se trata en esta tesis.

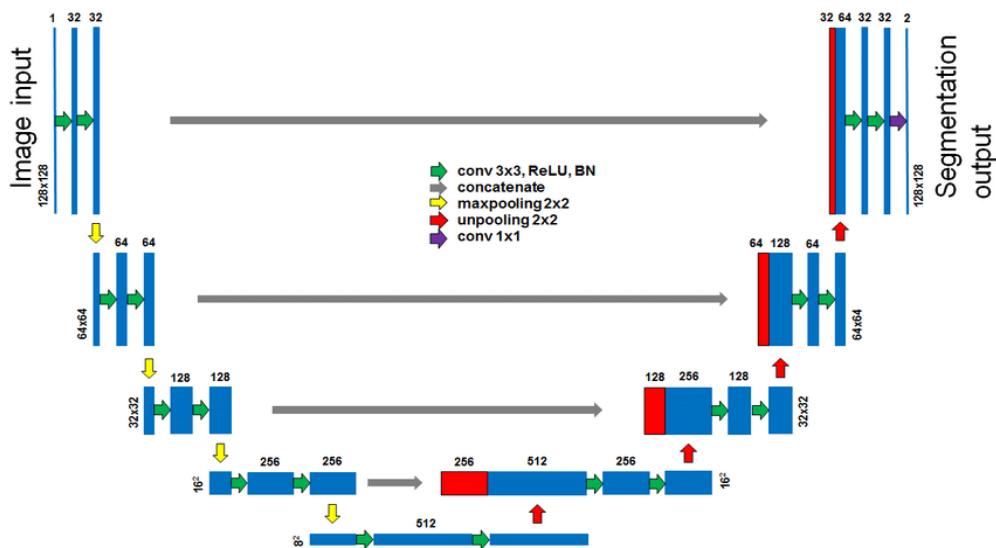


FIGURA 3.1: Estructura general del modelo U-Net

El modelo *U-net* representado en la FIGURA 3.1, consiste en la ruta de codificación (*encoder path* o *contracting path*) correspondiente a la parte izquierda del modelo y la ruta de decodificación (*decoder path* o *expansive path*) correspondiente a la parte derecha del modelo. El *encoder path* está formado por una serie de *contracting layers*.

En la primera ruta se pueden diferenciar tres iteraciones:

- ❖ 2D 3x3 convolución, con 16 canales de salida. En la FIGURA 3.1, podemos observar como hay 32 canales de salida, finalmente utilizamos 16 canales en nuestro modelo. Pero se puede montar con el número de canales que se quiera, siempre y cuando se mantenga la misma estructura, la función de activación ReLU (FIGURA 3.2), encargada de activar o no a cada una de las neuronas involucradas en el proceso. Utilizamos zero-padding para llenar la matriz de convolución y que el *output* tenga el mismo tamaño que la entrada. Finalmente inicializamos el *kernel* con una distribución gaussiana truncada, centrada alrededor de 0. [19]
- ❖ 2D 2x2 MaxPooling. Sirve para reducir el tamaño de la imagen manteniendo la información más relevante. Pasamos una ventana a lo largo de toda la imagen y nos quedamos con el valor máximo dentro de esa ventana. FIGURA 3.3.
- ❖ Dropout después de cada iteración, se introduce para asegurarnos de que no aparezca *overfitting* en el modelo.

Repetimos este proceso para cada una de las diferentes capas que forman el *encoder path*.

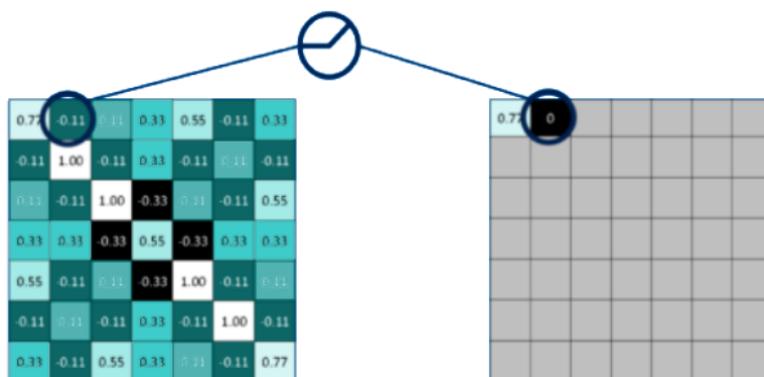


FIGURA 3.2: Funcionamiento de la activación ReLU

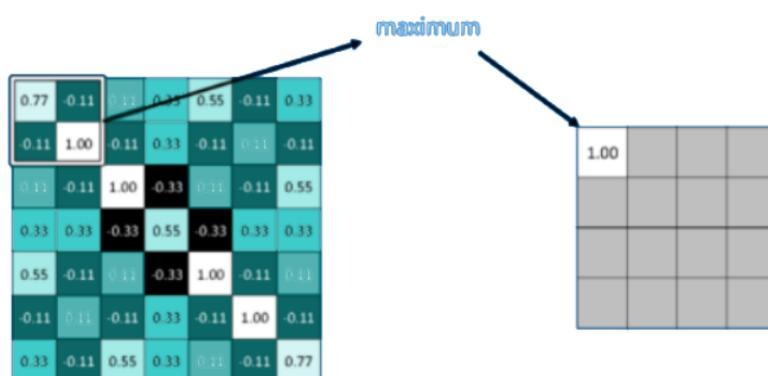


FIGURA 3.3: Funcionamiento del MaxPooling

Para el *decoder path* diferenciamos cuatro diferentes operaciones para cada capa.

- ❖ 2x2 up convolutions. Realizan la función inversa del *maxPooling* del *encoder path*
- ❖ La concatenación. Este paso es esencial y es el que diferencia U-net de otros modelos arquitectónicos de CNNs y es que se realiza la concatenación de la matriz resultante de la 2x2 up convolution con la matriz de igual tamaño del mismo nivel que encontramos en el *encoder path*.
- ❖ 2D 3x3 convolución. Se repite el 2D 3x3 del *encoder path* con la misma intención
- ❖ Dropouts. Se mantienen los Dropouts para que no aparezca *overfitting* en el modelo.

3.3. Pre-procesado

3.3.1. Dataset

El conjunto de datos con el que trabajamos son 640 imágenes de 3200 x 2400 pixeles con componentes en las tres dimensiones RGB.

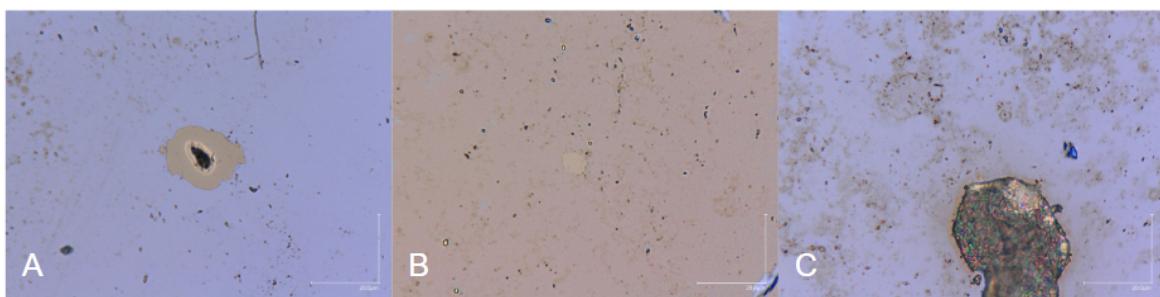


FIGURA 3.4: Ejemplos de imágenes del conjunto de datos

Como podemos apreciar en la FIGURA 3.4, tenemos imágenes con un claro impacto de rayo láser en el centro (FIGURA 3.4, imagen A), otras con el color del fondo diferente (FIGURA 3.4, imagen B) y otras donde aparecen elementos no deseados (FIGURA 3.4, imagen C). Esto es debido a la escala a la que se obtienen estas imágenes. Estamos hablando del orden de micrómetros y al capturar imágenes de gran resolución a una escala tan pequeña, provoca la aparición de interferencias imperceptibles por el ojo humano.

Como veremos en el apartado cuatro de resultados, estas interferencias repercutirán muy negativamente a la segmentación si realizamos el entrenamiento directamente sin aplicar ningún tipo de pre-procesado. Es por esto que aplicaremos una serie de técnicas de pre-procesado para mejorar la calidad de las imágenes de entrada.

3.3.2. Ecualización del histograma

La ecualización del histograma de una imagen es una transformación que se le aplica a la imagen. Realizamos esta transformación para ampliar el rango dinámico de los niveles de la imagen, para acentuar el contraste visual. Normalmente se aplica a imágenes en escala de grises, pero en nuestro caso trabajamos con imágenes RGB. Sin embargo, realizar la ecualización de los tres componentes del espacio de color RGB es mas complejo y los resultados no son tan beneficiosos. Es por este motivo que el primer paso a realizar es convertir las imágenes del espacio de color RGB a YUV, un espacio de color definido en términos de una componente de luma (Y) y dos componentes de crominancia (UV). FIGURA 3.5. La razón por la cual convertimos a este espacio es que la componente de luma es la más relevante y solo representa la escala de luminancia, con lo que se puede interpretar como la representación de la imagen RGB en escala de grises. Realizamos la ecualización del histograma de la componente de luminancia de las imágenes de entrada y una vez realizada esta transformación, volvemos al espacio de color RGB (FIGURA 3.6).

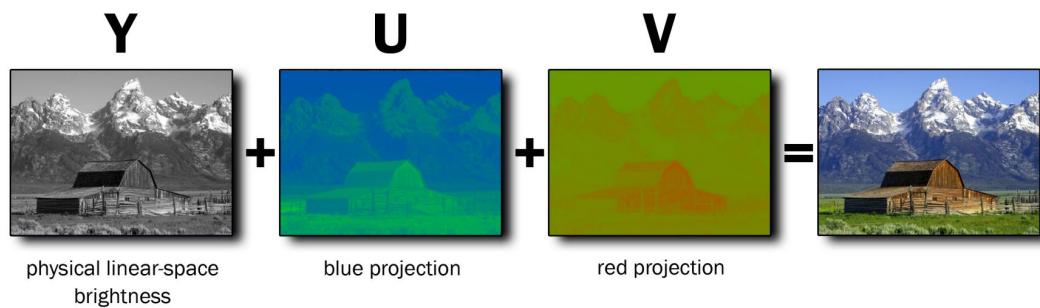


FIGURA 3.5: Ejemplo de las tres componentes YUV del espacio de color YUV

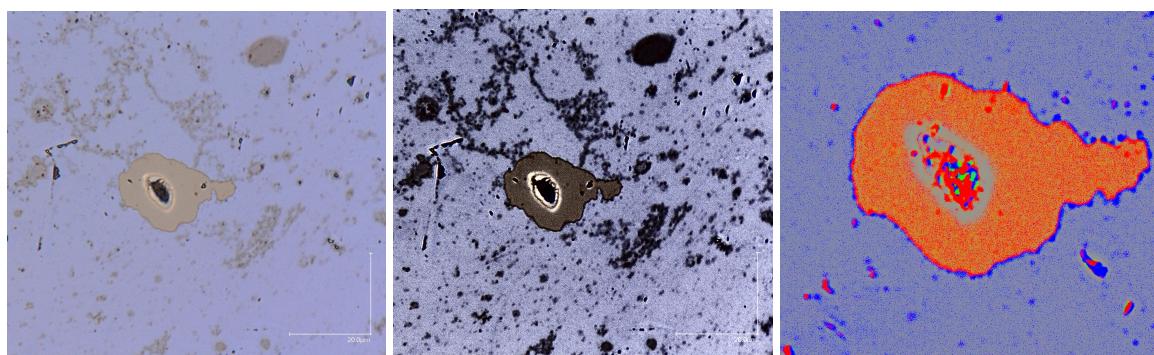


FIGURA 3.6: Imagen antes y después de ecualizar su histograma, y después de normalizar

3.3.3. Recorte de las imágenes

Como hemos mencionado anteriormente, las imágenes son de un tamaño de 3200 x 2400 píxeles. Esta resolución para realizar la segmentación, por muchas transformaciones de pre-procesado que apliquemos, no es muy recomendable, ya

que tal tamaño exige un gran volumen de memoria para poder ser segmentado. Asimismo, si utilizamos imágenes de una resolución tan grande, estaremos dedicando espacio de memoria a poder almacenar las imágenes enteras en vez de dedicarla a utilizar más parámetros de entrenamiento y de mayor precisión para mejorar la segmentación. Siguiendo este razonamiento, inicialmente decidimos realizar un cambio de tamaño y pasar de 3200 x 2400 a 1024 x 1024 píxeles. Pero como veremos en la sección de resultados, realizar un cambio de tamaño así sólo producía una peor segmentación, ya que estábamos perdiendo demasiada resolución sólo a expensas de ocupar menos espacio de memoria.

Como podemos observar en las imágenes de las figuras ssa o 44, el impacto del rayo láser siempre se encuentra en el centro. De la misma manera, apreciamos como en las partes exteriores de las imágenes siempre aparecen elementos no deseados a modo de interferencia. Es por ello que finalmente, realizamos un recorte del centro de la imagen. De esta manera, obtenemos imágenes de tamaño 1024 x 1024 píxeles (ya que era el tamaño especificado en el recorte), sin reducir la calidad del centro de éstas. Así, muchos de los objetos que aparecían en las partes exteriores serán eliminados y podremos realizar una segmentación manteniendo la calidad del centro y sin aumentar el volumen de memoria necesaria.

3.4. Hiperparámetros

Una vez definido el modelo, éste tiene unos parámetros que debemos considerar para optimizarlos lo mejor posible y obtener los mejores resultados. Estos parámetros modifican el comportamiento del modelo en cada capa y repercuten en el tiempo que dura una simulación y en el volumen de memoria necesaria para hacerla. Cada problema de segmentación tiene sus propios valores de hiperparámetros óptimos y desgraciadamente no hay una fórmula concreta para averiguar qué valor de cada hiperparámetro es el óptimo para un modelo. Además, cabe destacar que los hiperparámetros, como veremos en esta sección, están relacionados entre sí. Y modificar el valor de uno puede alterar a otros. Es por eso que al tratar de obtener los valores óptimos, hay que intentar parametrizarlos todos a la vez.

3.4.1. Batch size

El *batch size* se define como el número de muestras que se propagan en la red antes de actualizar los parámetros del modelo de nuevo. El número de muestras en un *forward/backward pass*.

En el momento de escoger entre un *batch size* grande o uno pequeño, cabe remarcar que uno de grandes dimensiones necesita grandes cantidades de memoria y si bien es cierto que realiza la segmentación más rápidamente, no siempre converge tan rápido y puede llegar a introducir una degradación en la calidad del modelo. Por otro

lado, un *batch size* pequeño puede producir menos precisión en cada paso debido a que la muestra del *batch* puede no ser lo suficientemente representativa del total. Sin embargo, pese a que realiza la segmentación de forma más lenta, suele converger mucho más rápido. Esto lo trataremos más adelante en el punto cuatro, donde veremos las diferencias entre el tiempo necesario para la simulación y cuánto tarda en converger dependiendo del tamaño del *batch*.

3.4.2. Epochs

El número de *epochs* hace referencia al número de veces que cada imagen ha sido utilizada durante el entrenamiento.

Cada *epoch* tiene un número de iteraciones que se define como:

$$\text{Iterations} = \frac{\text{Training Images}}{\text{Batch size}} \quad (3.1)$$

Como hemos mencionado anteriormente, los hiperparámetros están relacionados entre ellos y en este caso, el número de *epochs* se puede relacionar con el *batch size*.

$$\text{Epochs} = \frac{\text{Batch size} \times \text{Iterations}}{\text{Training Images}} \quad (3.2)$$

De esta manera, la conclusión que podemos obtener es que no sirve de nada tratar de optimizar el número de *epochs* si no se tiene en cuenta el *batch size*. Como veremos en la sección de resultados, lo idóneo es intentar parametrizar ambos para obtener el conjunto que mejor se adapte.

3.4.3. Optimizer

Los *optimizers* son algoritmos utilizados para minimizar la función de error (*loss function*, que veremos al final del punto tres) y para maximizar la eficiencia. Son funciones matemáticas que dependen de los parámetros del modelo. [2][24]

Hay dos factores muy importantes para definir un *optimizer*.

- ❖ *Learning rate*: El *learning rate* determina cómo de grandes o pequeños son los pasos del gradiente de descenso en la dirección del mínimo local de la función. De este modo, determina cómo de rápido o lento llegaremos a los pesos óptimos. FIGURA 3.7

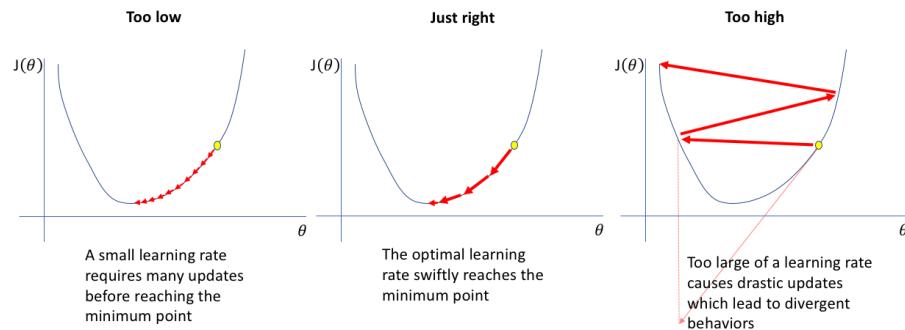


FIGURA 3.7: Evolución del *learning rate* según el tamaño del gradiente de descenso

- ❖ **Momentum:** El *momentum* es un método que ayuda a acelerar los vectores del gradiente de descenso en la dirección del mínimo local, consiguiendo una convergencia más rápida. FIGURA 3.8

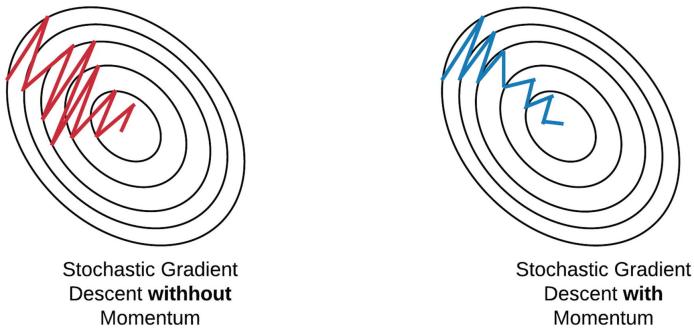


FIGURA 3.8: Stochastic gradient descent

En nuestro modelo solo utilizaremos el *optimizer Adam*, mientras que en el modelo de Jan Hering y Jan Kybic se utiliza el *Stochastic gradient descent* (SGD). A continuación haremos una breve descripción de ambos.

- ❖ El nombre del *optimizer Adam* deriva de *adaptive moment estimation* y utiliza estimaciones del primer y segundo momento del gradiente para adaptar el *learning rate* para cada peso del modelo. La gran ventaja que introduce el *optimizer Adam* es que es un buen punto de partida, no requiere de ningún cambio en el *momentum* o en el *learning rate* (que por defecto es 0.001) para obtener grandes resultados de forma general, cosa que con los optimizers anteriores al Adam (AdaGrad o RMSProp) los resultados de partida eran muy negativos.
- ❖ El *optimizer SGD* es un método iterativo de optimización. Si bien es cierto que dentro de los optimizers adaptativos, el Adam es un buen punto de partida, se ha demostrado que el *optimizer SGD* obtiene mejores resultados generales. [3]

3.5. Métricas de evaluación

Para entrenar el modelo, necesitamos definir las métricas que juzgarán la actuación de este después de cada iteración y qué función se ha de seguir para optimizar los resultados. En primer lugar se describirán las diferentes métricas utilizadas para entrenar el modelos, y posteriormente se explicarán las funciones utilizadas para optimizar los resultados. Cabe remarcar que, como pasa con los hiperparámetros, hay una infinidad de opciones y no hay una norma establecida para escoger métricas. Es cierto que dependiendo del problema hay unas que, por lo general, suelen obtener los mejores resultados pero eso no tiene porqué ser siempre así. Igualmente, se puede utilizar más de una métrica para una misma simulación. [15]

Para entender en qué se basan las métricas de evaluación, primero hemos de estudiar cómo clasificar los píxeles resultantes de una simulación. En esta tesis, los píxeles sólo pueden corresponder a dos clases, o bien pertenecen al fondo o son parte del impacto del rayo láser.

Podemos hablar de cuatro opciones a la hora de evaluar las predicciones de los píxeles, Verdadero Positivo (True Positive, TP), Verdadero Negativo (True Negative, TN), Falso Positivo (False Positive, FP) y Falso Negativo (False Negative, FN). Utilizando la FIGURA 3.9 como ejemplo, TP hace referencia a los píxeles que se clasificaron como impacto del rayo láser y tras la segmentación, se han predicho como verdaderos impactos del rayo láser. TN hace referencia a los píxeles de la imagen que se clasificaron como fondo y tras la segmentación se han predicho como verdadero fondo. Los píxeles que pertenecían al fondo de la imagen pero que tras la segmentación se han predicho como parte del impacto del láser se clasifican como FP. Y finalmente, los píxeles que realmente pertenecían al impacto del rayo láser pero tras la predicción se han marcado como fondo son los FN.



FIGURA 3.9: Clasificación de los píxeles resultantes de una simulación

3.5.1. Accuracy

La accuracy es una métrica muy simple que calcula el ratio de píxeles correctamente clasificados tras la segmentación en comparación al número total de píxeles.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.3)$$

3.5.2. Índice de Jaccard

El índice de Jaccard, también denominado intersección sobre la unión (IoU), es una de las métricas de evaluación más comunes en segmentación. Se define como la intersección entre el área predecida ($TP+FP$) y el área *Ground Truth Mask* dividida entre la unión de estas dos. FIGURA 3.10 [16][17]

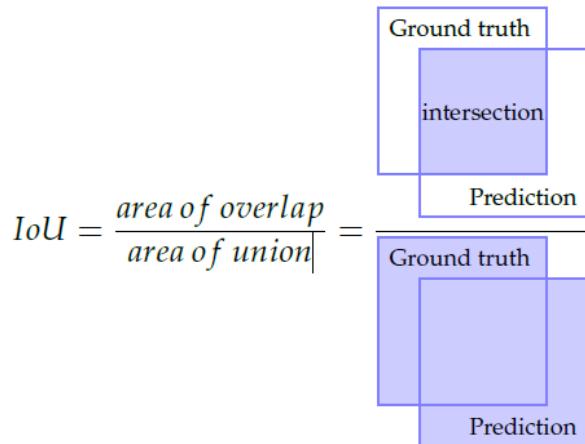


FIGURA 3.10: Ecuación IoU

3.5.3. Dice coefficient

El Dice coefficient o también llamado F1-Score se calcula a partir de dos métricas, Precisión y Recall.

- ❖ Precisión hace referencia a la porción de detecciones positivas que se predicen de forma correcta.

$$Precision = \frac{TP}{TP + FP} \quad (3.4)$$

- ❖ Recall se identifica como la porción de píxeles que se deberían predecir como positivos que realmente se predicen como tal.

$$Recall = \frac{TP}{TP + FN} \quad (3.5)$$

El Dice coefficient es la métrica que une la precisión y el recall, y se calcula como la media armónica entre ambas. De esta manera nos aseguramos que tengan el mismo peso a la hora de obtener un resultado.

$$Dice coefficient = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3.6)$$

3.5.4. Relative Area Error

Esta métrica no la utilizamos en nuestro modelo, sin embargo se utiliza en el modelo de Jan y por eso haremos una breve explicación. Se define el error relativo como la relación entre el error absoluto de una medida y la propia medida.

$$RE_{area} = \frac{|Y| - |\hat{Y}|}{|Y|} \quad (3.7)$$

3.6. Loss functions

Las Funciones de pérdida (*loss functions*) se utilizan para entrenar los modelos. Son las funciones de probabilidad que cuantifican las similitudes entre las predicciones y los valores reales. Estas funciones son decrecientes y su valor mínimo es el cero ya que un cero equivale a que las diferencias entre las predicciones y los valores reales son nulas.

Las métricas y los hiperparámetros definidos en las secciones anteriores sirven para optimizar los pesos y así minimizar la loss function.

Hay muchas *loss functions* dependiendo del tipo de problema al que nos enfrentemos. Para la segmentación semántica de imágenes biomédicas, la más utilizada es la *cross entropy* pero en este proyecto hemos estudiado dos, la *binary cross entropy* y la *dice coefficient loss*. [11]

3.6.1. Binary cross entropy

Esta *loss function* compara cada una de las probabilidades de las predicciones con el valor real al que corresponden. Al tratarse de probabilidades y ser una función binaria, los valores pueden corresponder a cero o a uno, cada uno representa una de las dos clases (fondo de la imagen, impacto del rayo láser).

La ecuación 3.8 caracteriza la *binary cross entropy*

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)) \quad (3.8)$$

Donde y corresponde a la predicción y $p(y)$ es la probabilidad de la predicción de corresponder a una de las dos clases para todos los píxeles N . A esta probabilidad se le aplica un logaritmo, de esta manera la *binary cross entropy* se presentará en escala logarítmica. Además tiene el símbolo menos ya que las *loss functions* tienen que ser decrecientes.[23]

3.6.2 Dice coefficient loss

Como veremos en el apartado cuatro, para tratar de analizar si alguna otra loss function se adapta mejor a nuestro modelo que la *binary cross entropy*, analizaremos la opción de utilizar una de las métricas de evaluación como loss function. En concreto hallamos estudios realizados sobre la opción de utilizar el Dice coefficient como loss function. [4][21].

En resumen, la lógica detrás de esta opción es que al necesitar una función decreciente para la loss function, sabemos que cualquiera de las métricas de evaluación genera una función creciente y con un crecimiento exponencial. De esta manera, si invertimos los resultados de las métricas de evaluación, tendremos una función logarítmica decreciente. Los resultados de esta prueba los tenemos en la simulación nueve del apartado cuatro.

4

Resultados

En este apartado estudiaremos todos los pasos necesarios para desarrollar las simulaciones de entrenamiento del modelo y analizaremos los resultados obtenidos en cada una de ellas, utilizando todos los parámetros explicados anteriormente, avanzaremos por las simulaciones modificando los hiperparámetros, las métricas, las loss functions... Recordando que el objetivo principal es realizar la segmentación semántica de las imágenes del dataset para obtener las regiones del impacto del rayo láser.

Primero explicaremos algunas de las simulaciones realizadas y expondremos los resultados obtenidos en cada una de ellas veremos cómo afectan estos con respecto a las simulaciones anteriores.

Finalmente discutiremos cuáles han sido las opciones que nos han llevado a los mejores resultados.

4.1. Configuraciones de las simulaciones

Para realizar las simulaciones, primero hemos de dividir las 640 imágenes del dataset en imágenes de entrenamiento (*train*) e imágenes de evaluación (*testing*). Tras múltiples pruebas, decidimos dedicar un 90% al entrenamiento (576 imágenes) y el otro 10% a la evaluación (64 imágenes). El proceso que se sigue para entrenar y evaluar es el siguiente:

Por cada iteración, el modelo estudia las imágenes de entrenamiento, estas imágenes se comparan con sus máscaras, que son la representación binaria de las imágenes de entrada y sirven para determinar la región a segmentar. Una vez realizadas todas las iteraciones dentro de un epoch, con el entrenamiento que ha realizado el modelo, intenta predecir las imágenes de evaluación, si la predicción es buena o no dependerá de lo mucho que se parezca esta a la propia máscara.

Así pues, la segmentación ideal sería una predicción donde todos los píxeles de la imagen predicha (FIGURA 4.1 Imagen C) pertenecieran a la misma clase que los de su máscara (FIGURA 4.1 Imagen B).

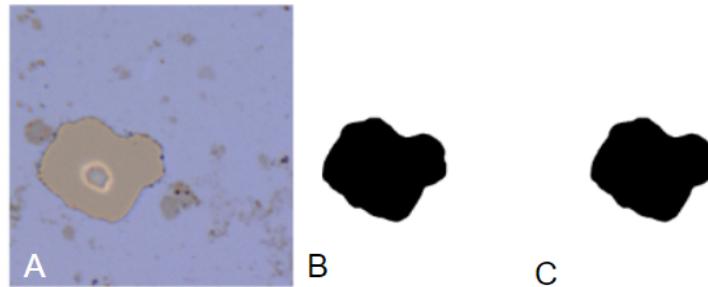


FIGURA 4.1: Segmentación ideal

4.2. Simulación uno

Tras realizar simulaciones de prueba con una parte de la muestra de 640 imágenes, para la primera simulación (TABLA 4.1) con todas las imágenes, los parámetros con los que entrenaremos el modelo son los siguientes:

Simulación nº	1
Epochs	20
Batch size	16
Imagen	Sin recorte Cambio de tamaño: 1.024x1.024 píxeles de resolución Normalización: no Ecualización del histograma: no
Métrica de evaluación	Accuracy
Loss function	Binary cross entropy
Optimizer	Adam

TABLA 4.1: Simulación 1

Como podemos observar en la FIGURA 4.2, los resultados no son nada satisfactorios, se puede intuir una pequeña relación entre las predicciones y las máscaras de referencia pero es muy poco significativa. Esto es debido al hecho de utilizar el cambio de tamaño en vez del recorte, lo que hace que mantengamos las relaciones entre la cantidad de fondo y el tamaño del impacto del rayo láser. En muchas ocasiones este es tan solo una pequeña porción de píxeles en el centro y con la simplicidad del modelo que se ha utilizado en esta simulación, este no ha sido capaz de entrenar correctamente y los resultados han sido muy negativos.

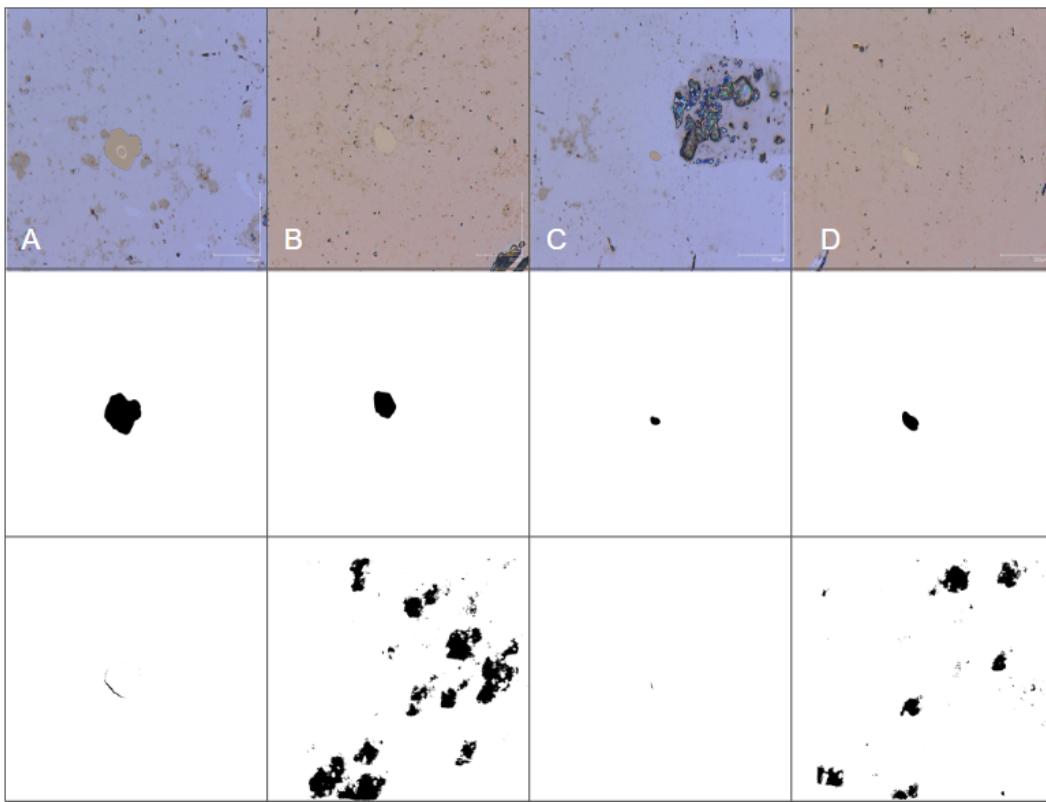


FIGURA 4.2: Resultados simulación 1. La Columna A representa el primer ejemplo.
Lo mismo para B, C y D.

Sin embargo, si nos fijamos en la FIGURA 4.3, podemos observar que obtenemos unos resultados de más del 98%, esto es debido a que la *accuracy* no es la métrica óptima para este tipo de problemas, dado que esta lo que mide es la similitud entre todos los píxeles de la imagen predecida y de la máscara. Como los resultados predecidos son prácticamente blancos en su totalidad, estos se parecen mucho a las máscaras, donde predomina el fondo de este color. En las siguientes simulaciones, trataremos este problema utilizando las otras métricas mencionadas en el apartado tres donde el peso del objeto a segmentar es mucho más relevante que los píxeles que pertenecen al fondo.



FIGURA 4.3: Gráfica de la *accuracy* simulación 1

4.3. Simulaciones dos, tres y cuatro

En este grupo de simulaciones, nos centraremos en la mejora del pre-procesado ya que según los resultados obtenidos en la primera, el modelo parece estar funcionando pero la complejidad de las imágenes de entrada aparenta ser demasiado elevada para nuestro modelo. En este punto estudiamos varias vías:

1. Dividir la imagen en paquetes y solo segmentar los paquetes en el que aparezcan segmentos del impacto del rayo láser. Desechamos esta idea porque pese a que se han hecho estudios [5] donde se han obtenido excelentes resultados realizando estos paquetes, el volumen de memoria necesaria es demasiado elevado y siguiendo con el objetivo de obtener un modelo con buenos resultados pero sin la necesidad de una gran capacidad de memoria, concluimos que hay otras opciones con las que este se podía cumplir mejor. Además, si recordamos lo mencionado anteriormente en esta tesis, nuestras imágenes solo tienen un impacto en la zona central, eso quiere decir que en el resto de la imagen no es necesario hacer segmentación. La idea de dividir la imagen en paquetes, es muy acertada en el caso de que en la imagen haya múltiples objetos a segmentar, pero como en nuestro caso no es así, se trata de una sobrealmimentación del modelo que nos podría llevar a grandes resultados, pero excesivamente complejo.
2. Aumentar las capacidades del modelo, de la misma manera que en el punto anterior, para segmentar una imagen de gran tamaño donde para la gran mayoría de esta no se requiere una segmentación es un gasto innecesario de recursos y volumen de memoria. Además, a diferencia del punto anterior, no es seguro que aumentando las capacidades del modelo de segmentación sin realizar ninguna mejora a las imágenes de entrada vayamos a conseguir mejores resultados.
3. Finalmente la vía que decidimos aplicar es recortar el tamaño de la imagen y normalizarla. A diferencia de la simulación uno donde realizamos un cambio de tamaño, al realizar un recorte, no mantenemos las proporciones de fondo/impacto y este último pasa a ser parte mayoritaria de las imágenes de entrada.

Tras realizar el recorte de la imagen, realizamos la normalización tanto de la imagen de entrada como de su máscara. Este paso solo lo realizamos para intentar mejorar un poco más el contraste y que se acentúe el contorno del impacto del rayo láser y para que el tipo de variables con el que trabajará el modelo sean *floats* en vez de *integers*.

En el caso de la máscara es menos complejo, al tratarse de una imagen binaria en escala de grises, dividimos todos sus píxeles por 255, de esta manera el rango pasa de

(0,255) a (0,1). En el caso de las imágenes de entrada, las dividimos por el valor máximo de píxel de la imagen.

En las simulaciones dos (TABLA 4.2), tres (TABLA 4.3) y cuatro (TABLA 4.4) como nos centramos en mejorar el pre-procesado, muchos de los parámetros con los que las entrenamos son los mismos que la primera simulación:

Simulación nº	2
Epochs	20
Batch size	16
Imagen	Recorte: 1.024x1.024 píxeles de resolución Normalización: no Ecualización del histograma: no
Métrica de evaluación	Accuracy
Loss function	Binary cross entropy
Optimizer	Adam

TABLA 4.2: Simulación 2

Simulación nº	3
Epochs	20
Batch size	16
Imagen	Recorte: 1.024x1.024 píxeles de resolución Normalización: si Ecualización del histograma: no
Métrica de evaluación	Accuracy
Loss function	Binary cross entropy
Optimizer	Adam

TABLA 4.3: Simulación 3

Simulación nº	4
Epochs	50

Simulación nº	4
<i>Batch size</i>	16
Imagen	Recorte: 1.024x1.024 píxeles de resolución Normalización: si Ecualización del histograma: no
Métrica de evaluación	Accuracy
Loss function	<i>Binary cross entropy</i>
Optimizer	<i>Adam</i>

TABLA 4.4: Simulación 4

En la primera fila de la FIGURA 4.4 tenemos las imágenes de entrada, en la segunda, las imágenes después de recortar el centro, estas son con las que entrenamos en la experiencia dos. En la tercera fila tenemos las máscaras, que sirven de referencia para lo que sería el resultado ideal. En la cuarta fila tenemos los resultados de la predicción.

Como podemos apreciar, los resultados son mucho mejores que en la simulación uno. Se empieza a ver el contorno de los impactos del rayo láser en los ejemplos de la FIGURA 4.4-A, FIGURA 4.4-B y FIGURA 4.4-D aunque seguimos estando muy lejos de unos resultados decentes. Además, en los ejemplos B, C y D han aparecido algunos píxeles predecidos como FP.

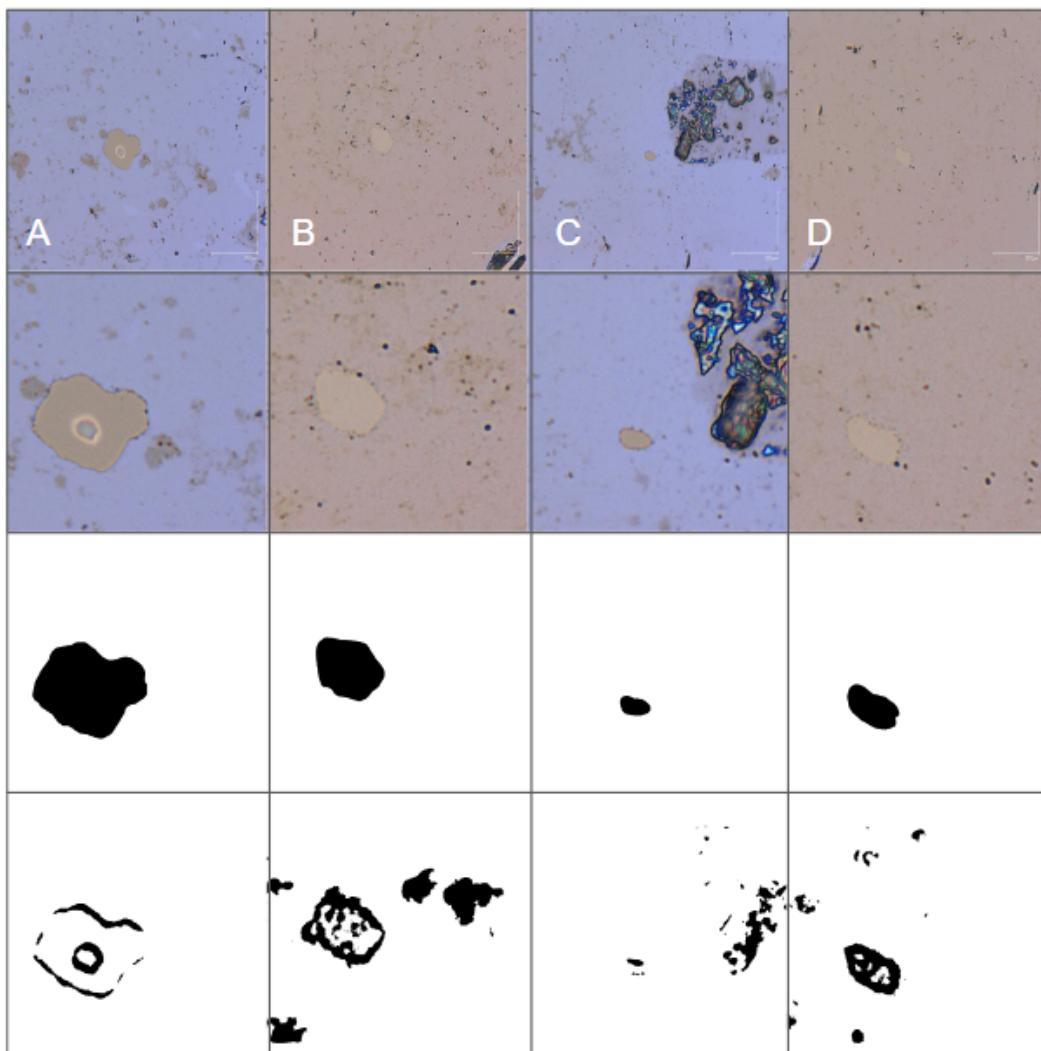


FIGURA 4.4: Resultados simulación 2

En la FIGURA 4.5 analizamos algunos ejemplos de la experiencia 3. Gracias a la normalización de las imágenes, sin modificar nada más respecto a la experiencia dos, las predicciones son mucho mejores. La mayoría de impactos tienen gran parte de su interior marcados correctamente y ya no solo tenemos los contornos definidos. También es cierto que al normalizar las imágenes, pese a que a niveles generales se han mejorado las predicciones, hay algunas imágenes concretas donde la normalización ha provocado resultados mucho peores. Este inconveniente lo resolveremos en las siguientes simulaciones.

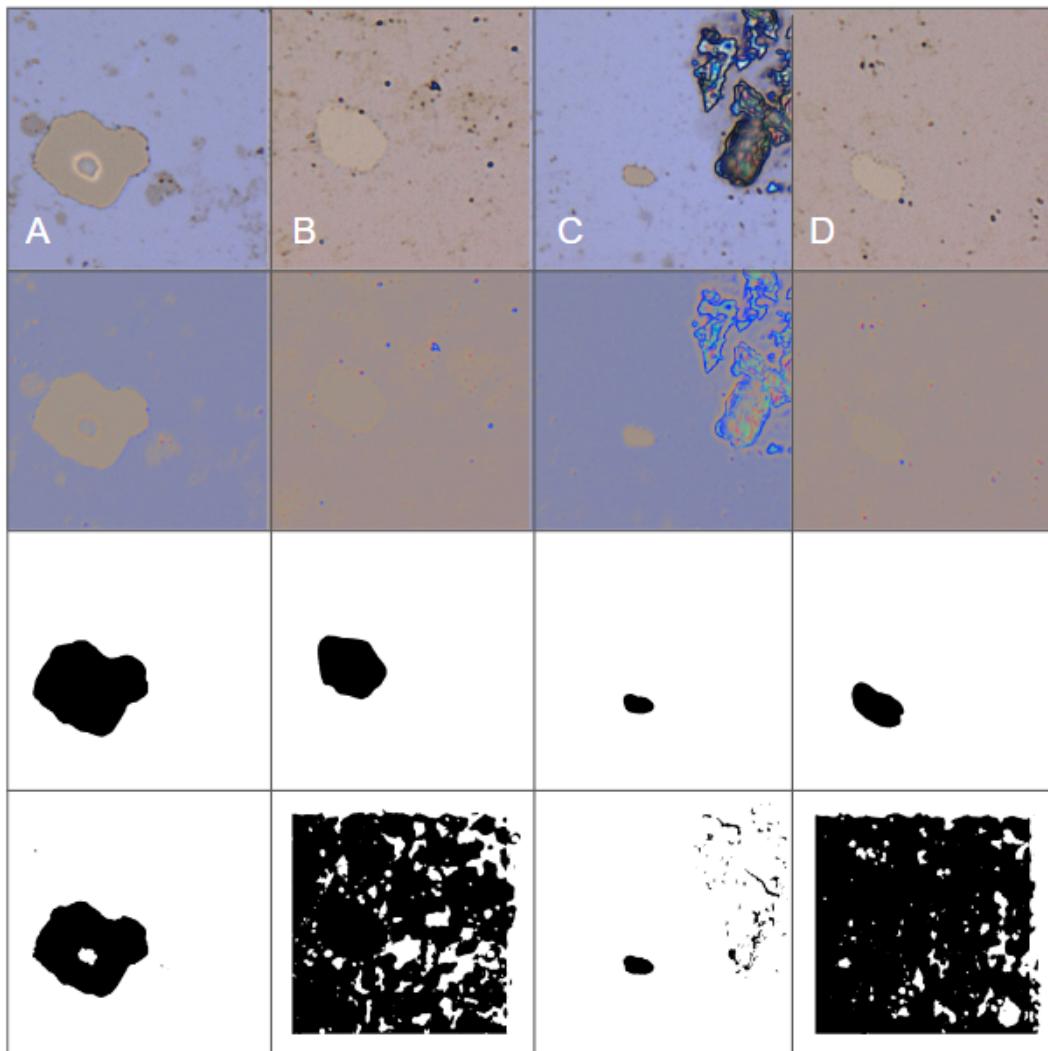


FIGURA 4.5: Resultados simulación 3

Pese a que en las simulaciones dos y tres nos estamos centrando en el pre-procesado, en la FIGURA 4.6 podemos ver la evolución de la accuracy, y en la FIGURA 4.7 la loss function durante los 20 epochs de la experiencia tres. Dejando de lado lo comentado en la experiencia uno sobre el motivo de los excelentes resultados de la accuracy, otro punto a tener en cuenta son los picos que aparecen en la loss function en los epochs 12 y 15. Estos muy probablemente son debidos a una mala selección del learning rate del optimizer. Pese a que por ahora no afectan significativamente, en las siguientes simulaciones también lo tendremos en cuenta para mejorar los resultados y asegurarnos que no produzcan mayores inconvenientes.

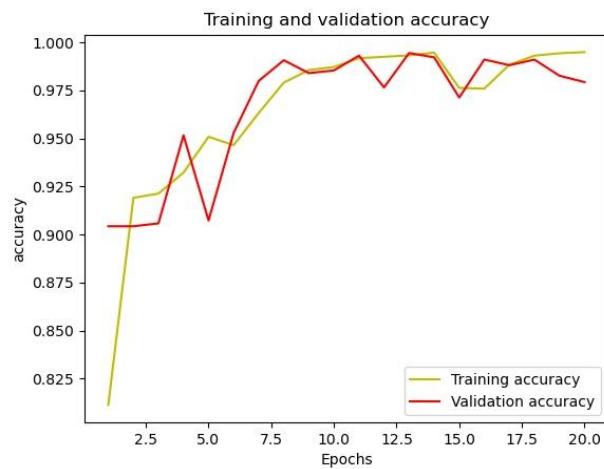


FIGURA 4.6: Gráfica de la *accuracy* simulación 3

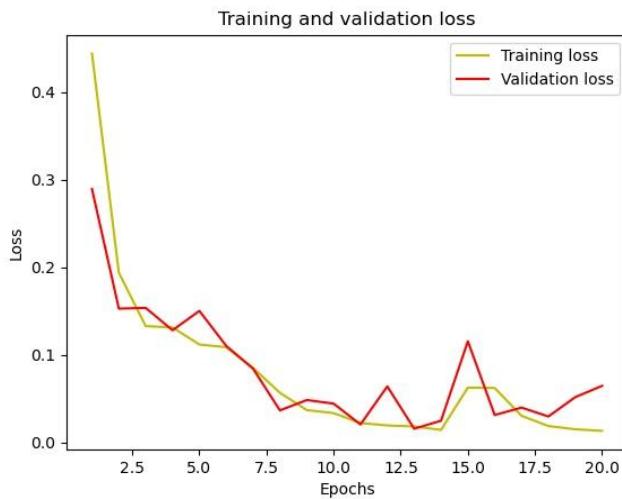


FIGURA 4.7: Gráfica de la *loss function* simulación 3

Observando las gráficas de la evolución de la *loss function* y la *accuracy* durante los *epochs* de las simulaciones anteriores, vemos que no llegan a converger totalmente probablemente debido a que no se han utilizado suficientes *epochs* para el entrenamiento. Así pues, para la simulación cuatro utilizamos los mismos parámetros que en las anteriores pero esta vez con 50 *epochs*.

Si observamos la FIGURA 4.8, podemos apreciar que pese a que realmente el número de *epochs* utilizado en las anteriores simulaciones no permitía llegar a la convergencia, la diferencia entre esta simulación y las anteriores no es tan grande ya que las únicas mejoras que apreciamos es que en la primera imagen de ejemplo todo el impacto se ha predicho correctamente. Por este motivo deducimos que el problema con algunas de las imágenes como en la FIGURA 4.8-B o la FIGURA 4.8-D no está relacionado con el número de *epochs*, sino con los parámetros que utilizamos

para entrenar. Es por esto que en las siguientes simulaciones pasamos a modificar los parámetros de entrenamiento.

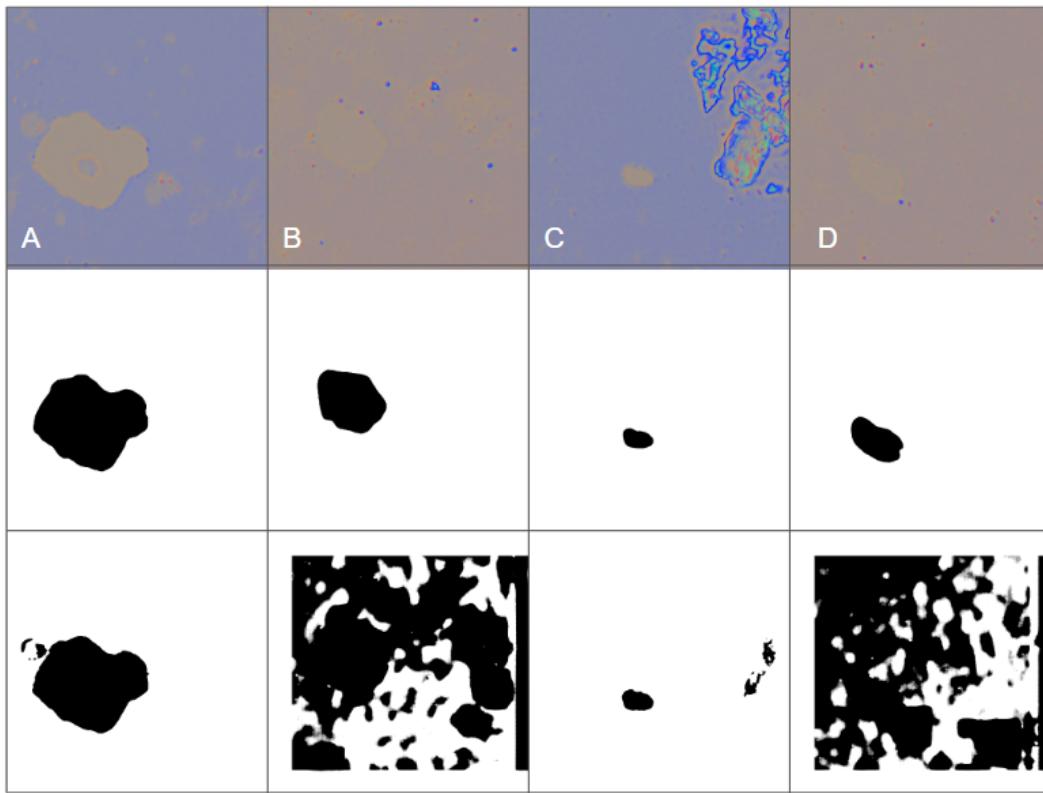


FIGURA 4.8: Resultados simulación 4

En la FIGURA 4.9 y la FIGURA 4.10 se ve más claramente como los datos de entrenamiento si que llegan a una convergencia, mientras que los de validación fluctúan bastante. Además en esta simulación tenemos un ejemplo aún más claro de los picos mencionados en las simulaciones anteriores debidos al *learning rate* escogido.

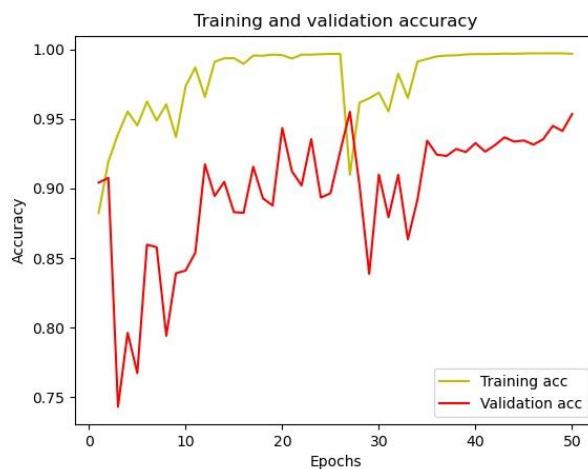


FIGURA 4.9: Gráfica de la accuracy simulación 4

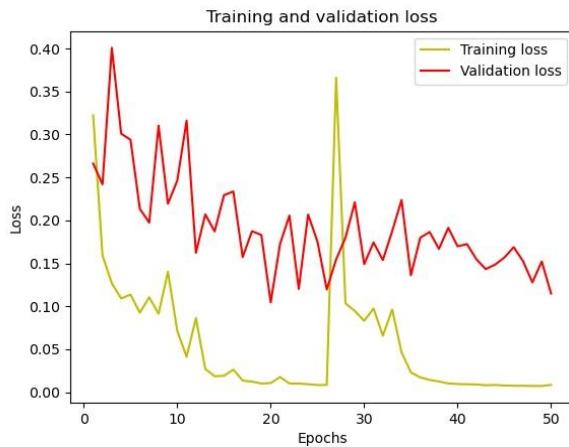


FIGURA 4.10: Gráfica de la loss function simulación 4

4.4. Simulaciones cinco, seis y siete

En las simulaciones cinco, seis y siete tratamos el tema de las métricas de evaluación además de aplicar inversión en las máscaras y modificar el tamaño del *batch*.

Como hemos mencionado anteriormente, la *accuracy* no es una buena métrica a la hora de evaluar segmentación semántica. Al hacer un cómputo sobre todos los píxeles de la imagen, los pertenecientes al fondo suponen una mayoría y condicionan notoriamente los resultados. Es por esto que el principal cambio que introduciremos es utilizar el Índice de Jaccard (IoU) como métrica de evaluación.

Hasta ahora estábamos utilizando siempre un *batch size* de 16, este es un buen punto de partida ya que no es ni extremadamente pequeño (con lo que no sería representativo y entrenaría el modelo de forma muy lenta) ni extremadamente grande (que provocaría que no convergiera rápidamente y añadiese una degradación en la calidad del modelo pese a realizar el entrenamiento mucho más rápido). En este punto parametrizamos el *batch size* para comprobar resultados con diferentes valores y poder verificar si realmente 16 era la mejor opción o si otros tamaños eran más óptimos. En la FIGURA 4.12, se ve claramente como la mejor opción para el *batch size* es 16. Mientras que para 8 (FIGURA 4.11) se obtienen unos valores similares a los de 16, la simulación dura 8 horas aproximadamente, eso son dos horas más que para el tamaño 16 y casi cuatro para 32. Así que sumado a que los resultados no son tan buenos, descartamos la opción de un *batch size* de 8. Por otro lado, la simulación con un tamaño de 64 (FIGURA 4.13) tardó la mitad de la de 8, lo que la hace más rápida que la de 16. Pero los resultados son los peores de todos y añade bastante volumen de memoria necesario para entrenar al modelo.

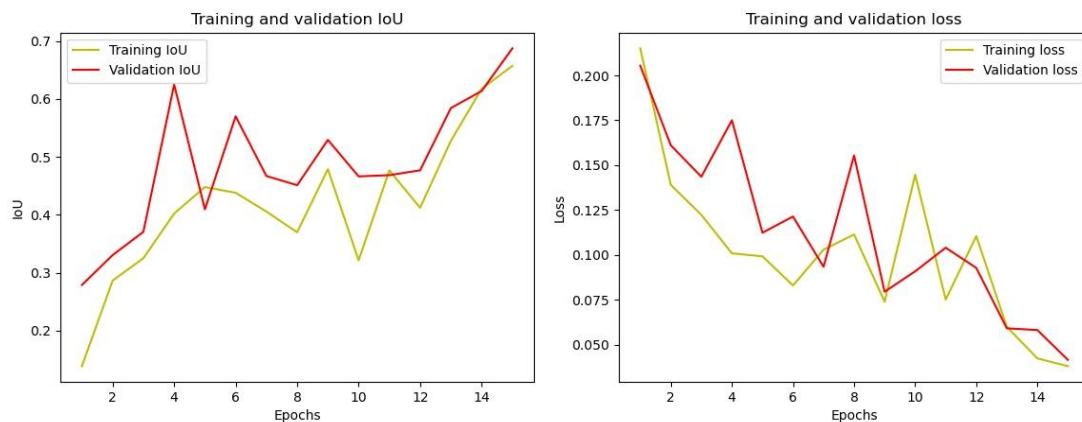


FIGURA 4.11: Gráfica del IoU y de la loss function del batch size 8

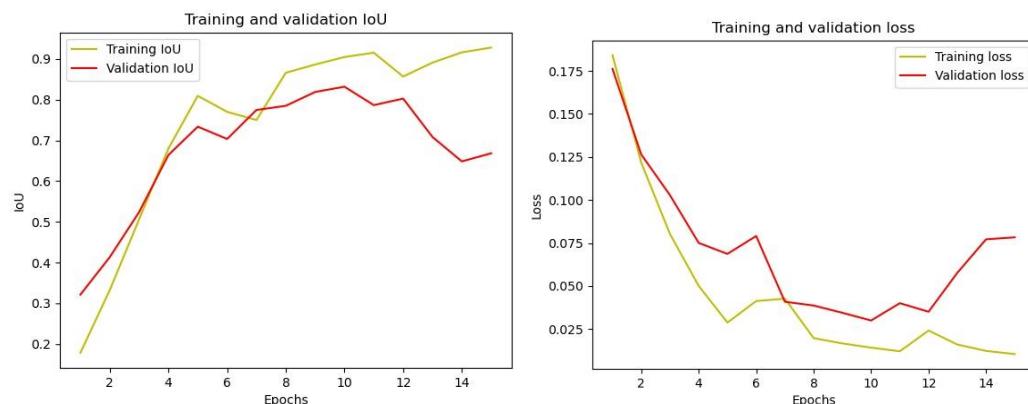


FIGURA 4.12: Gráfica del IoU y de la loss function del batch size 16

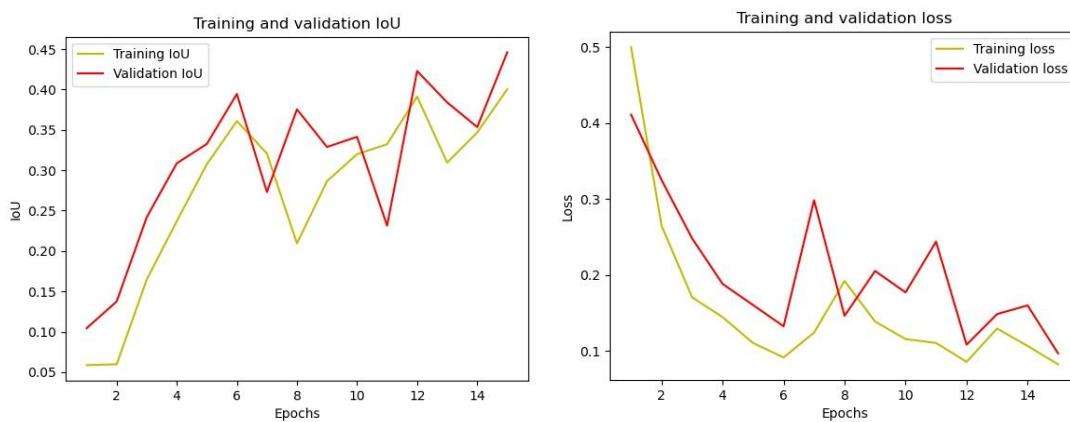


FIGURA 4.13: Gráfica del IoU y de la loss function del batch size 64

Así pues reafirmamos que 16 era una gran opción para definir el batch size, sin embargo, al obtener resultados muy similares y no provocar un gran aumento en el volumen de memoria necesario, empezamos a utilizar 32 ya que acelera el proceso de entrenamiento.

Los parámetros para el entrenamiento del modelo en las simulaciones cinco (TABLA 4.5), seis (TABLA 4.6) y siete (TABLA 4.7) son los siguientes:

Simulación nº	5
Epochs	15
Batch size	32
Imagen	Recorte: 1.024x1.024 píxeles de resolución Normalización: si Ecualización del histograma: no
Métrica de evaluación	IoU
Loss function	<i>Binary cross entropy</i>
Optimizer	<i>Adam</i>

TABLA 4.5: Simulación 5

Simulación nº	6
Epochs	15
Batch size	16
Imagen	Recorte: 1.024x1.024 píxeles de resolución Normalización: si Ecualización del histograma: no
Métrica de evaluación	IoU
Loss function	<i>Binary cross entropy</i>
Optimizer	<i>Adam</i>

TABLA 4.6: Simulación 6

Simulación nº	7
Epochs	40
Batch size	32
Imagen	Recorte: 1.024x1.024 píxeles de resolución Normalización: si

Simulación nº	7
	Ecuallización del histograma: no
Métrica de evaluación	IoU
Loss function	<i>Binary cross entropy</i>
Optimizer	Adam

TABLA 4.7: Simulación 7

Fijáandonos en la FIGURA 4.14, lo más destacable es que hemos mejorado radicalmente las predicciones en los ejemplos FIGURA 4.14-B y FIGURA 4.14-D con respecto a las simulaciones anteriores, esta mejora radica en el hecho de haber entrenado el modelo con la métrica IoU en vez de la *accuracy*. Es cierto que como pasaba en las simulaciones dos y tres, las predicciones del impacto del láser están vacías por dentro, esto es debido a que no hemos utilizado suficientes epochs en este entrenamiento del modelo. Sin embargo en este caso nuestra intención era mejorar las manchas debidas al uso de la *accuracy*.

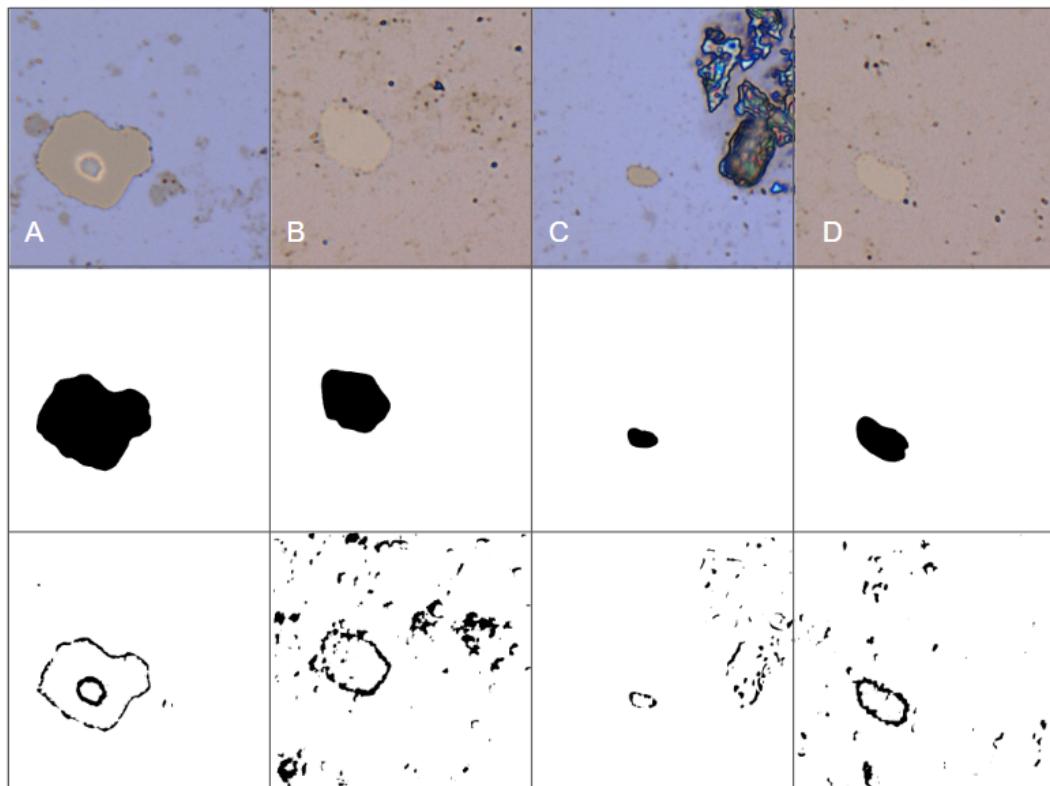


FIGURA 4.14: Resultados simulación 5

En la simulación número seis, introducimos un cambio para mejorar el hecho de que el interior de los impactos quedará vacío y es invertir las máscaras. De esta manera el

fondo será de color negro y el impacto de color blanco. Como vemos en la FIGURA 4.15, el mero hecho de invertir el color de las máscaras ha mejorado significativamente los resultados.

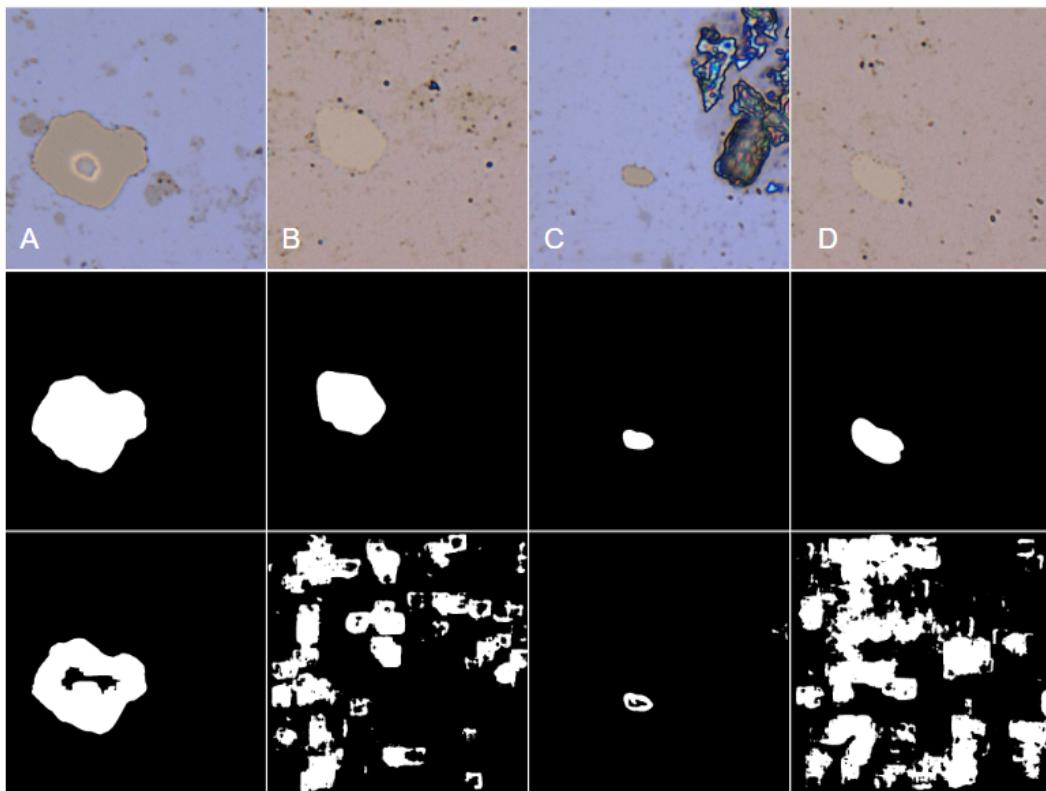


FIGURA 4.15: Resultados simulación 6

En las gráficas de la FIGURA 4.16 y la FIGURA 4.17 podemos ver como los resultados de IoU obtenidos en la simulación seis no son tan altos como los de la *accuracy* en las simulaciones anteriores. No obstante, como hemos explicado, es mucho más representativo un resultado decente en IoU que excepcionales resultados en *accuracy*.

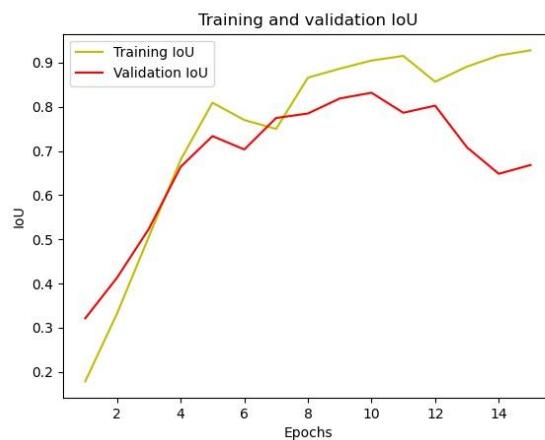


FIGURA 4.16: Gráfica del IoU simulación 6

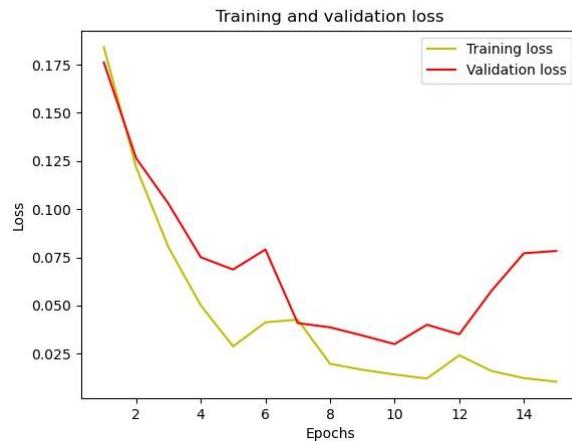


FIGURA 4.17: Gráfica de la loss function simulación 6

En las simulaciones cinco y seis, al utilizar solamente 15 epochs, el modelo no converge totalmente, es por eso que realizamos la simulación siete con los mismos parámetros que la experiencia seis pero con cuarenta epochs.

En la FIGURA 4.18 tenemos los resultados. Por primera vez, en muchas de las imágenes de entrada obtenemos unas predicciones de gran calidad, como podemos ver en los ejemplos de la FIGURA 4.18-A y la FIGURA 4.18-C del ejemplo. Además también hemos reducido la aparición de manchas en imágenes como en la FIGURA 4.18-B y la FIGURA 4.18-D del ejemplo.

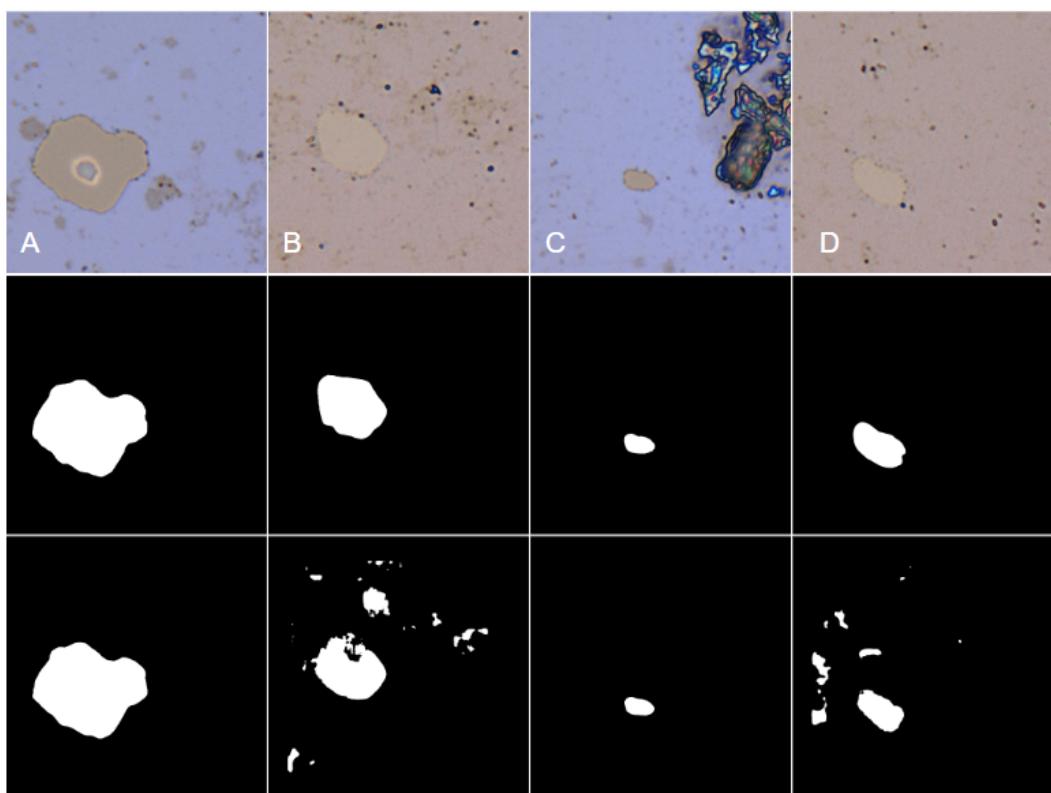


FIGURA 4.18: Resultados simulación 7

Estas mejoras en los resultados de las predicciones (FIGURA 4.19 y FIGURA 4.20) también se ven reflejadas tanto en el entrenamiento como en validación convergen claramente y los valores de IoU que se alcanzan son mucho más cercanos a uno que en las simulaciones anteriores.

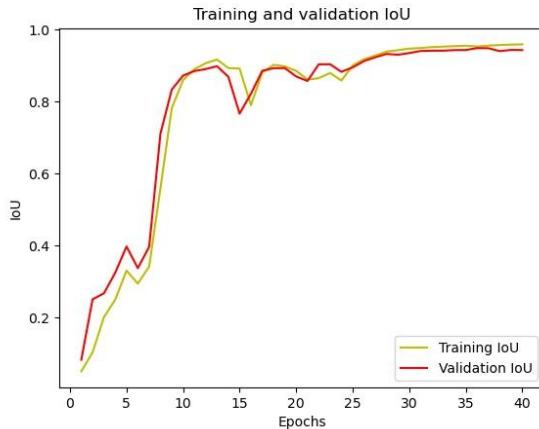


FIGURA 4.19: Gráfica del IoU simulación 7

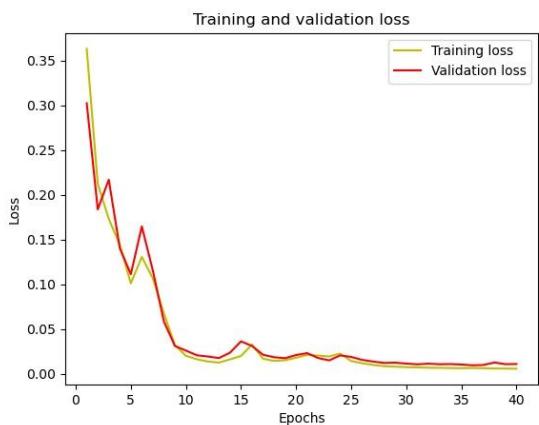


FIGURA 4.20: Gráfica de la loss function simulación 7

4.5. Simulaciones ocho y nueve

Llegados a este punto, dado que los resultados con el IoU no acababan de ser excelentes, decidimos utilizar otra métrica de evaluación, el Dice coefficient. Como este es un resultado directo de la precisión y el recall, suponíamos que podría servir mejor como métrica de evaluación del modelo de segmentación.

Por otro lado, hasta ahora siempre hemos utilizado la *binary cross entropy* como loss function, y si bien es cierto que para modelos de segmentación semántica, esta es la loss function que generalmente da mejores resultados (dado que está pensada expresamente para casos con solo dos grupos de clasificación), parecía que llegaba a

un punto donde no conseguía mejorar más los resultados de las predicciones. Por ese motivo, decidimos utilizar el Dice coefficient como loss function.

En la simulación ocho (TABLA 4.8) y nueve (TABLA 4.9), los parámetros de entrenamiento fueron los siguientes:

Simulación nº	8
Epochs	40
Batch size	32
Imagen	Recorte: 1.024x1.024 píxeles de resolución Normalización: si Ecualización del histograma: no
Métrica de evaluación	Dice coefficient
Loss function	Binary cross entropy
Optimizer	Adam

TABLA 4.8: Simulación 8

Simulación nº	9
Epochs	40
Batch size	32
Imagen	Recorte: 1.024x1.024 píxeles de resolución Normalización: si Ecualización del histograma: no
Métrica de evaluación	Dice coefficient
Loss function	Dice coefficient loss
Optimizer	Adam

TABLA 4.9: Simulación 9

Como observamos en algunos ejemplos de los resultados de la simulación ocho (FIGURA 4.21), solo modificando la métrica de evaluación con respecto a la experiencia siete, hemos conseguido reducir la aparición de manchas en las predicciones, manteniendo el nivel de precisión en las que ya teníamos unos excelentes resultados.

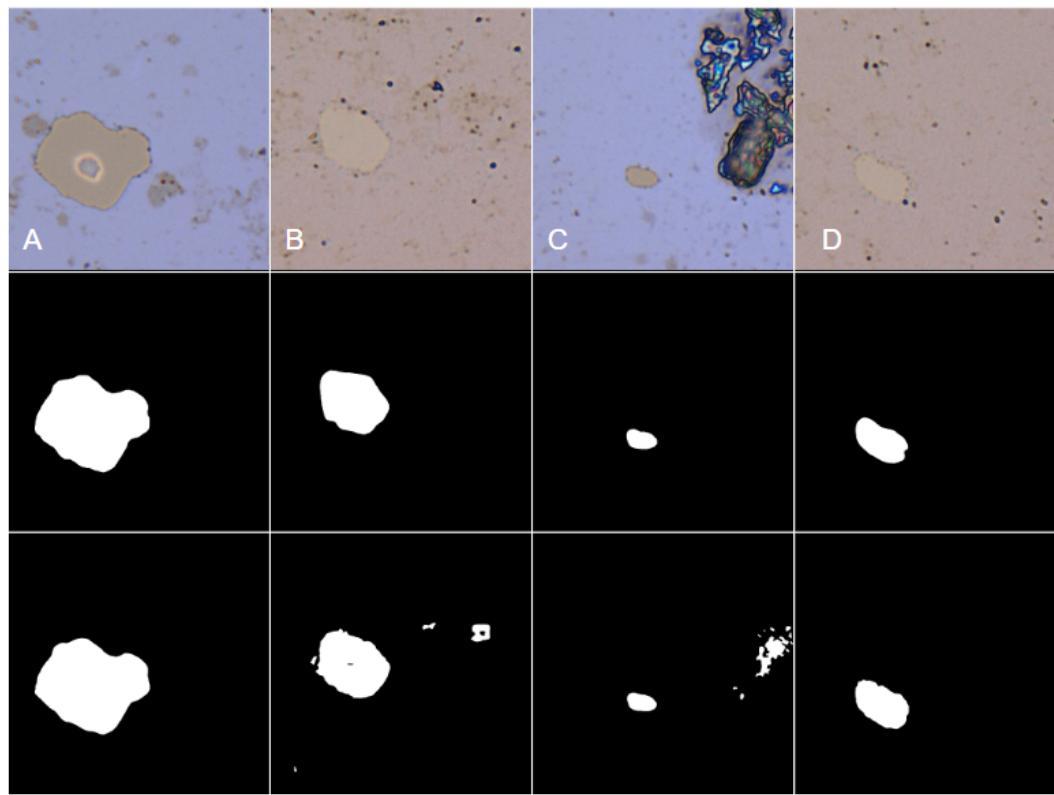


FIGURA 4.21: Resultados simulación 8

Podemos ver como el Dice coefficient (FIGURA 4.22) es mucho más inestable y tarda mucho más en converger, es cierto que esto puede ser mejorado con el *learning rate* y el *momentum* del optimizer.

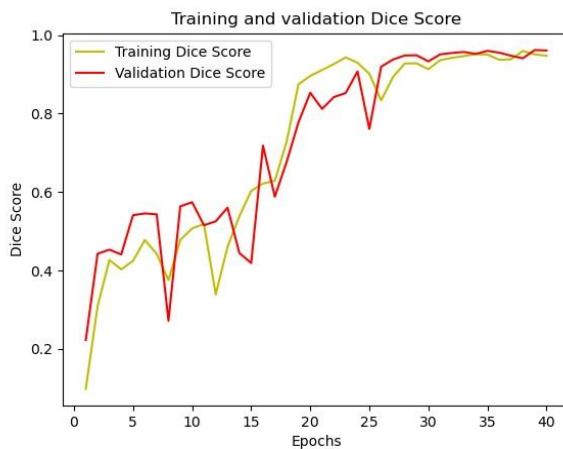


FIGURA 4.22: Gráfica del dice coefficient simulación 8

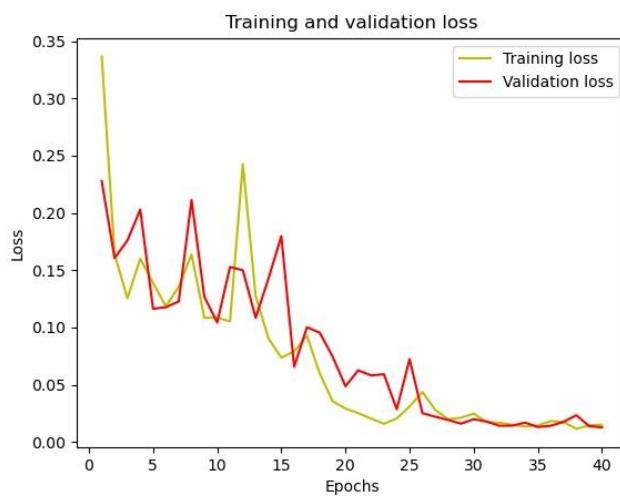


FIGURA 4.23: Gráfica de la loss function simulación 8

Para intentar estabilizar la loss function (FIGURA 4.23), que el modelo converja más rápidamente y que todas las predicciones tengan unos resultados excelentes cambiaremos la loss function para la simulación nueve como hemos mencionado anteriormente.

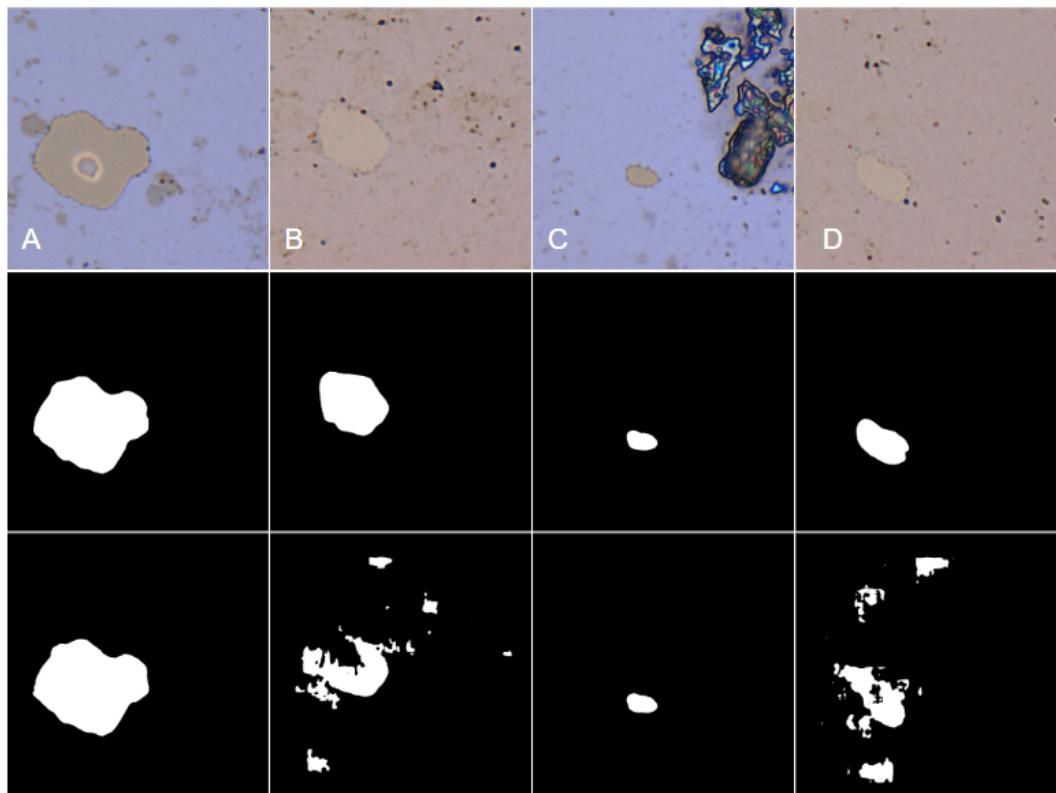


FIGURA 4.24: Resultados simulación 9

Los resultados obtenidos en esta experiencia (FIGURA 4.24) son peores que en la anterior. Aparece una mayor cantidad de manchas en las predicciones y las zonas del

impacto en las predicciones donde aparecen las manchas no están tan bien predecidas como en la anterior. Así pues, pese a que con esta loss function el modelo es más estable y converge muchísimo más rápido (FIGURA 4.25 y FIGURA 4.26), no podemos concluir que sea una mejor opción que la *binary cross entropy*.

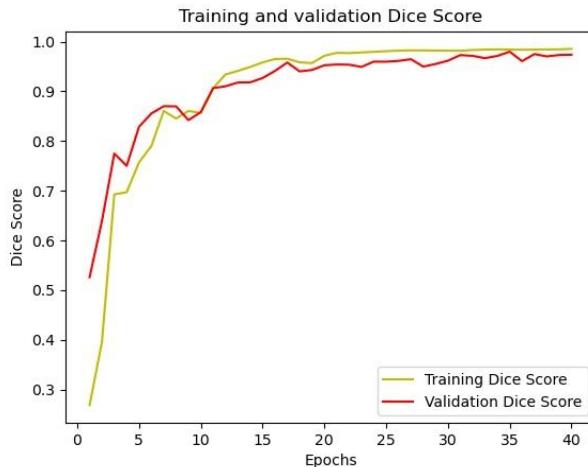


FIGURA 4.25: Gráfica del dice coefficient simulación 9

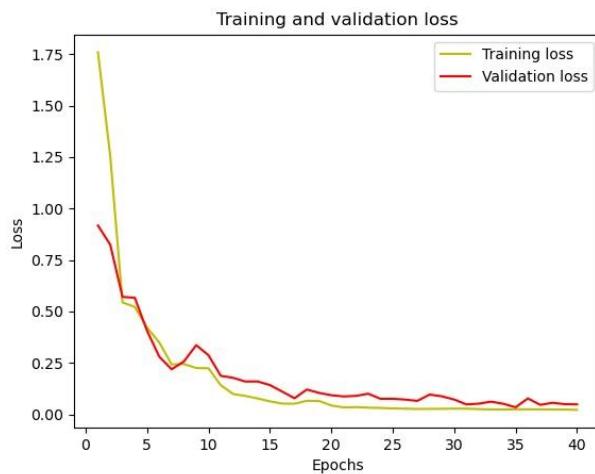


FIGURA 4.26: Gráfica de la loss function simulación 9

4.6. Simulación diez

Pese a utilizar diferentes batch sizes, número de epochs, métricas de evaluación y loss functions, no conseguíamos hacer desaparecer las manchas en los resultados de las predicciones. Es por ese motivo que decidimos volver atrás y analizar nuevamente el pre-procesado. Como podemos observar en las imágenes de las figuras de las simulaciones anteriores, al final, los mayores problemas no venían debido a los artefactos interferencia en parte de la imagen o a los diferentes tamaños y formas de los impactos del láser a lo largo de las imágenes de entrada sino que eran debidos a el

cambio de color en el fondo de la imagen. Esto provocaba que en las imágenes con un fondo de color muy parecido al del impacto, aparecieran las manchas. Por este motivo decidimos aplicar la técnica de ecualización de histograma de la imagen, para ampliar el rango dinámico de los niveles y acentuar el contraste.

En esta simulación (TABLA 4.10) los parámetros de entrenamiento fueron los siguientes:

Simulación nº	10
Epochs	40
Batch size	32
Imagen	Recorte: 1.024x1.024 píxeles de resolución Normalización: si Ecualización del histograma: si
Métrica de evaluación	IoU
Loss function	Binary cross entropy
Optimizer	Adam

TABLA 4.10: Simulación 10

Los resultados obtenidos (FIGURA 4.27) al realizar la ecualización son excelentes, ya en las imágenes de entrada observamos como la ecualización ha resaltado los impactos del rayo láser y eso provocará la desaparición de las manchas en las predicciones.

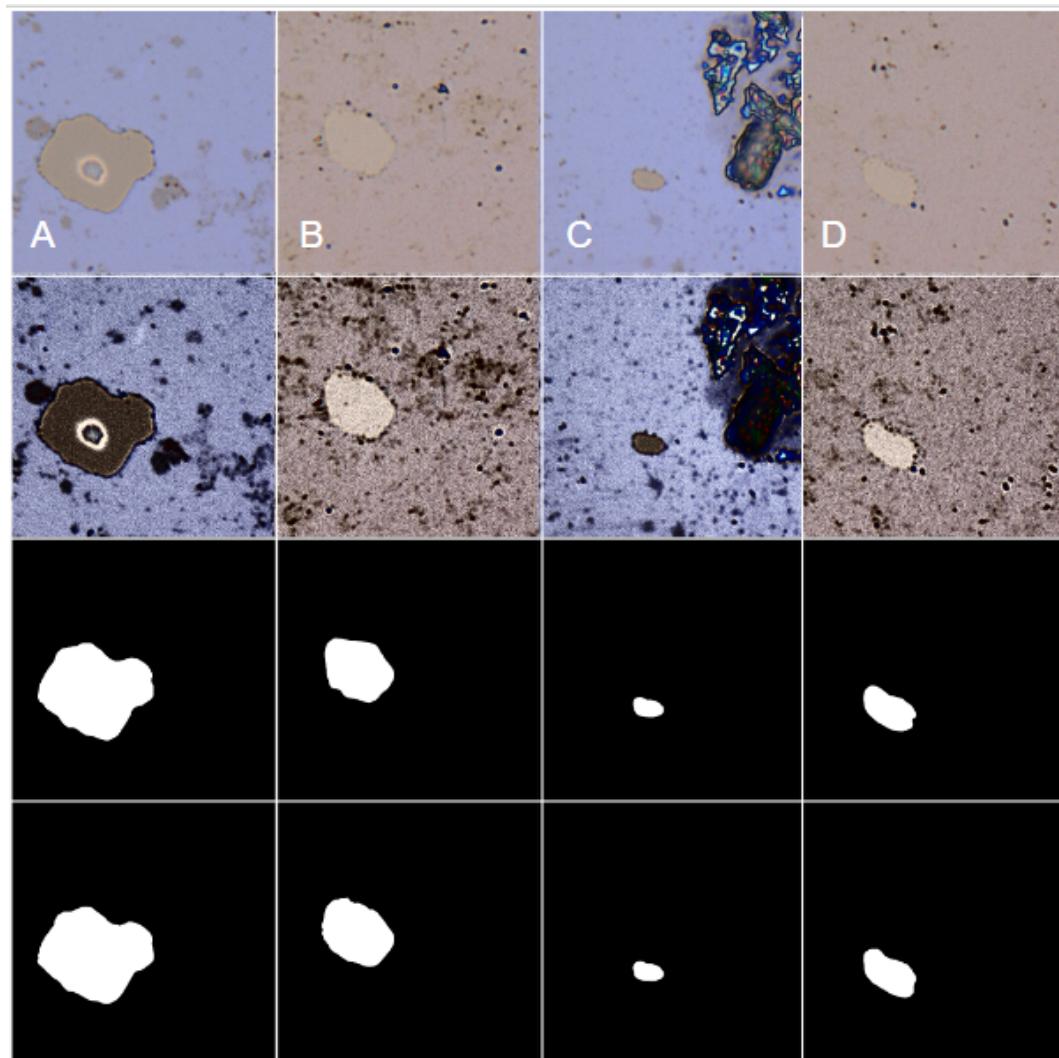


FIGURA 4.27: Resultados simulación 10

Como podemos observar en las gráficas de la FIGURA 4.28 y la FIGURA 4.29 se aprecia un pico alrededor del epoch 10, pero tras realizar más simulaciones variando el *learning rate*, es cierto que al variar el *learning rate* por defecto (0.001) con un *learning rate* menor (0.0001) estos picos no se veían tan pronunciados y con un *learning rate* mayor (0.01) los picos eran más pronunciados y aparecían nuevos, a la hora de evaluar las predicciones, todas llegaban a los mismos resultados.

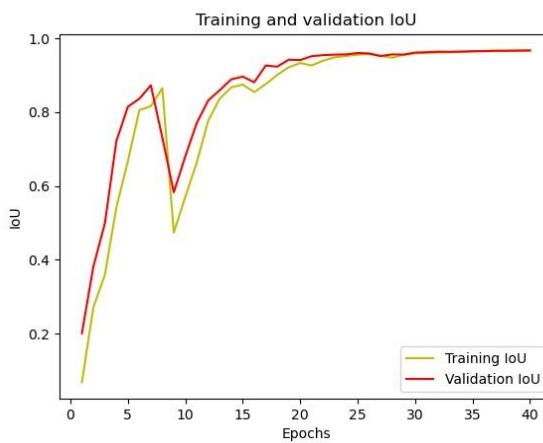


FIGURA 4.28: Gráfica del IoU simulación 10

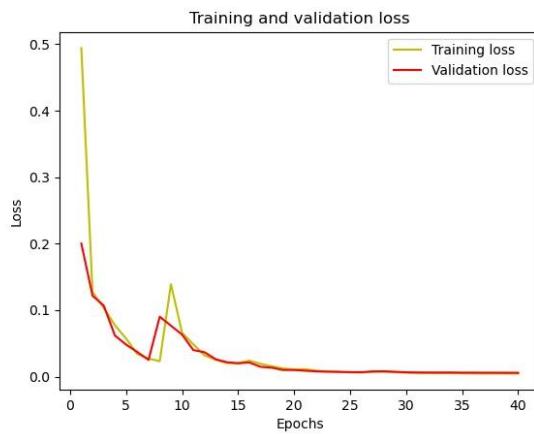


FIGURA 4.29: Gráfica de la loss function simulación 6

4.7. Sumario de resultados

En este apartado analizaremos los resultados numéricos de todas las simulaciones (TABLA 4.11) y veremos como lo representado en las gráficas de las métricas de evaluación y loss function y las imágenes de ejemplos concuerdan con los resultados numéricos obtenidos.

En las cuatro primeras simulaciones podemos ver como la *accuracy* es muy elevada, tanto en *Training* como en *Validation*, pero la *loss function* es la más elevada de todas las simulaciones. Esto demuestra lo mencionado anteriormente que pese a que los niveles de *accuracy* sean muy elevados, estos no son representativos a la hora de afrontar un problema de segmentación. Además, si nos fijamos en la *accuracy* de la *Validation*, podremos observar que pese a que objetivamente los resultados de la simulación uno son los peores, tiene la *accuracy* más alta y esto es debido a que como realizamos el corte de las imágenes en las simulaciones dos, tres y cuatro en vez de el cambio de tamaño en la uno, la región del impacto del rayo láser pasa a representar

gran parte de las imágenes y en la uno, como el impacto era una región muy pequeña, la mala predicción de esta no influenciaba prácticamente en el cómputo global

Una vez pasamos a las simulaciones cinco seis y siete, comprobamos que pese a obtener resultados de IoU más bajos que con la *accuracy*, esta es mucho más representativa y funciona mucho mejor como métrica de evaluación. Es lógico que al utilizar tan pocos *epochs* en las simulaciones cinco y seis los resultados parecen bastante negativos, pero en estas dos simulaciones no buscábamos mejorar la *loss function*, sino verificar que la *accuracy* no es una buena opción para la segmentación semántica y comprobar de qué modo afectaría el cambio de *batch* y la inversión de las máscaras. En la simulación siete por lo contrario, sí que buscábamos los mejores resultados de la *loss function* y las métricas de evaluación y se puede ver claramente en la tabla como los resultados son bastante buenos.

Apreciamos también que entre la simulación siete y ocho, no hay prácticamente diferencias en los resultados, demostrando así que la IoU y el Dice coefficient son ambas igualmente válidas para segmentación semántica.

En la simulación nueve, se ve reflejado en los resultados numéricos lo mencionado anteriormente y es que al utilizar el Dice coefficient como *loss function* no hemos obtenido unas predicciones tan precisas.

Finalmente en la simulación diez, obtenemos los mejores resultados de todos.

		Loss	Accuracy	IoU	Dice score
Simulación 1	Training	0.0201	0.9824	-	-
	Validation	0.0872	0.9867	-	-
Simulación 2	Training	0.0187	0.9762	-	-
	Validation	0.0772	0.9685	-	-
Simulación 3	Training	0.0134	0.9950	-	-
	Validation	0.0648	0.9793	-	-
Simulación 4	Training	0.0113	0.9930	-	-
	Validation	0.0525	0.9625	-	-
Simulación 5	Training	0.0303	-	0.6570	-
	Validation	0.0416	-	0.6960	-

Simulación 6	<i>Training</i>	0.0144	-	0.9122	-
	<i>Validation</i>	0.0769	-	0.7103	-
Simulación 7	<i>Training</i>	0.0056	-	0.9595	-
	<i>Validation</i>	0.0109	-	0.9438	-
Simulación 8	<i>Training</i>	0.0149	-	-	0.9476
	<i>Validation</i>	0.0126	-	-	0.9609
Simulación 9	<i>Training</i>	0.0212	-	-	0.9853
	<i>Validation</i>	0.0481	-	-	0.9736
Simulación 10	<i>Training</i>	0.0092	-	0.9675	-
	<i>Validation</i>	0.0087	-	0.9655	-

TABLA 4.11: Resultados de las simulaciones

5

Presupuesto

6

Conclusiones y futuras vías de desarrollo

Como hemos podido comprobar en los resultados de nuestro proyecto y en los del proyecto de Jan Hering y Jan Kybic, la aplicación de técnicas de DL en el ámbito de la biomedicina es muy prometedora y tiene muchas aplicaciones. Más concretamente en el tópico de esta tesis, hemos podido comprobar como los modelos de CNN son excelentes herramientas para afrontar la segmentación de imágenes biomédicas.

No se pudieron llevar a cabo los objetivos iniciales de utilizar el proyecto realizado por Jan Hering y Jan Kybic para optimizar sus parámetros y reducir la cantidad de memoria necesaria para la simulación de su modelo.

Sin embargo, el hecho de intentar encontrar los motivos que producían las diferencias entre los resultados obtenidos por Jan y los nuestros, ya generó un aprendizaje en técnicas relacionadas con DL, arquitecturas de modelos CNN y segmentación de imágenes.

Desde el momento en que decidimos crear nuestro propio modelo desde cero, consideramos que muchas de las nociones de DL que durante la primera parte no se habían tratado en profundidad, al tener que crear el modelo, se estudiaron con mucho más detalle. Por ejemplo, muchos de los parámetros necesarios para entrenar modelos, no solo los teníamos que conocer, como en la primera parte, sino que debíamos ser capaces de analizar por qué motivos unos eran mejor que otros y aplicarlos en nuestro modelo. Siguiendo esta misma idea, en el tópico de segmentación de imágenes, si bien es cierto que al analizar el proyecto de Jan Hering y Jan Kybic sabíamos que se realizaban unas técnicas de pre-procesado, no ha sido hasta que las hemos utilizado nosotros que hemos sido capaces de ver hasta qué punto pueden influir en los resultados. Y de la misma manera, con las métricas de evaluación y las *loss functions*, donde no solo las hemos tenido que entender sino que debíamos ser capaces de evaluarlas y analizarlas.

Por otro lado, otro de los objetivos que nos habíamos marcado era aprender el lenguaje de programación Python ya que antes de realizar este proyecto, nuestros conocimientos en este eran prácticamente nulos. En la primera parte, al solo analizar el proyecto ya creado, no nos fue muy necesario conocimientos de Python, sin embargo, para crear el modelo, tuvimos que aprender mucha sintaxis de este lenguaje.

Finalmente, nos podemos dar cuenta de que el hecho de haber creado un modelo desde cero, si bien es cierto que ha provocado no poder profundizar demasiado en el proyecto de Jan Hering y Jan Kybic, sí que ha servido para alcanzar mucho mejor el resto de objetivos iniciales que nos propusimos.

Pensando en futuras vías de desarrollo, por desbordar del ámbito estricto de este trabajo no hemos profundizado en la temática de los optimizers. Es cierto que con el Adam hemos llegado a los objetivos marcados inicialmente, pero es muy probable que con un estudio más profundo de los optimizers, lográramos alcanzar unos resultados óptimos en este tópico.

Otro punto que resultaría bastante interesante realizar es una optimización mucho más exhaustiva del volumen de memoria necesario. En esta tesis siempre hemos intentado reducir al máximo la cantidad de memoria necesaria, pero se puede llegar a reducir mucho más. Una aplicación que tendría el lograr una gran reducción de memoria es el hecho de poder realizar simulaciones en tiempo real.

Este dato por sí sólo permite percibir que esta investigación aborda un espectro de mejoras en el diagnóstico médico con imágenes que es susceptible de transferencia en el ámbito empresarial. Además, el estudio realizado se enmarca en una línea de investigación con impacto social, pues participa en el objetivo 9 (Desarrollo de dispositivos electrónicos más eficientes) de los Objetivos de las Naciones Unidas para un desarrollo sostenible. Y finalmente, el objeto de este trabajo de final de grado enlaza con la iniciativa IMPaCT que es la Infraestructura de Medicina de Precisión asociada a la Ciencia y la Tecnología, del Ministerio de Educación y Ciencia. [25]

Bibliografía

- [1] O. Ronneberger, P. Fischer y T. Brox, "U-Net: Convolutional networks for biomedical image segmentation", Vision, Vol.9351, n.º 234-241, 2015. [En línea]. Disponible: <https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>
- [2] S. Ruder, "An overview of gradient descent optimization algorithms", NUI Galaway, Dublin, 2017. [En línea]. Disponible:
<https://arxiv.org/pdf/1609.04747.pdf>
- [3] A. Gupta, R. Ramanath, J. Shi y S. S. Keerthi, "Adam vs. SGD: Closing the generalization gap on image classification", California, 2021. [En línea]. Disponible: <https://opt-ml.org/papers/2021/paper53.pdf>
- [4] S. Jadon, "A survey of loss functions for semantic segmentation", 2006.14822v4 [eess.IV], septiembre de 2020. [En línea]. Disponible: <https://arxiv.org/pdf/2006.14822.pdf#:~:text=Dice%20Coefficient%20is%20a%20widely,in%20achieving%20the%20optimal%20results>
- [5] V. Chiluka. "Patchify in python - to extract patches from large images". Python Programs. <https://python-programmes.com/patchify-in-python-to-extract-patches-from-large-images/>.
- [6] S. Bhattacharjee. "Python_for_microscopists". <https://www.researchgate.net/profile/Sreenivas-Bhattacharjee>.
- [7] E. Stevens, L. Antiga y T. Viehmann, Deep Learning With PyTorch. New York: MANNING, 2020. [En línea]. Disponible: https://isip.piconepress.com/courses/temple/ece_4822/resources/books/Deep-Learning-with-PyTorch.pdf
- [8] N. Navab, J. Hornegger, W. M. Wells y A. F. Frangi, Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. Munich: Springer, 2015. [En línea]. Disponible: <https://link.springer.com/content/pdf/10.1007/978-3-319-24574-4.pdf>
- [9] "Modelos de detección de objetos | aprende machine learning". Aprende Machine Learning. <https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/>.
- [10] Cai S, Tian Y, Lui H, Zeng H, Wu Y, Chen G. " Dense-UNet: a novel multiphoton in vivo cellular image segmentation model based on a convolutional neural network." Quant Imaging Med Surg, 2020. [En línea]. Disponible: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7276369/#:~:text=Dense%2DUNet%20adds%20a%20dense,superiority%20of%20the%20deep%20CNN>
- [11] N. Ibtehaz y M. S. Rahman, "MultiResUNet : Rethinking the U-Net architecture for multimodal biomedical image segmentation", Neural Networks, Volume 121, pp. 74–87, 2020.

- [12] Zhou Z, Siddiquee MMR, Tajbakhsh N, Liang J. "UNet++: A Nested U-Net Architecture for Medical Image Segmentation." Deep Learn Med Image Anal Multimodal Learn Clin Decis Support. 2018.
- [13] L. Ding, K. Zhao, X. Zhang, X. Wang y J. Zhang, "A lightweight u-net architecture multi-scale convolutional network for pediatric hand bone segmentation in x-ray image", IEEE Access, vol. 7, pp. 68436–68445, 2019. [En línea]. Disponible: <https://doi.org/10.1109/access.2019.2918205>
- [14] "Beginner's guide to semantic segmentation". V7 - AI Data Platform for Computer Vision. <https://www.v7labs.com/blog/semantic-segmentation-guide>.
- [15] E. Tiu, "Metrics to evaluate your semantic segmentation model", Stanford ML Group, Boston, 2019. [En línea]. Disponible: <https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2>
- [16] "Intersection over union iou in object detection segmentation". LearnOpenCV - OpenCV, PyTorch, Keras, Tensorflow examples and tutorials. <https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/#IoU-in-Image-Segmentattion>.
- [17] "U-Net and iou for object detection in image processing ★ code A star". Code A Star. <https://www.codeastar.com/u-net-object-detection-iou/>.
- [18] "Keras documentation: Keras API reference". Keras: the Python deep learning API. <https://keras.io/api/>.
- [19] A. Chazareix. "About convolutional layer and convolution kernel". Sicara | Experts en Data et IA. <https://www.sicara.fr/blog-technique/2019-10-31-convolutional-layer-convolution-kernel#:~:text=A%20common%20choice%20is%20to,:%203,%201%20by%20color>.
- [20] "PyTorch model summary - detailed tutorial - python guides". Python Guides. <https://pythonguides.com/pytorch-model-summary/>.
- [21] A. Anis. "How to build custom loss functions in keras for any use case | cnvrg.io". cnvrg. <https://cnvrg.io/keras-custom-loss-functions/>.
- [22] bigironsphere. "Loss function library - keras & pytorch". Kaggle: Your Machine Learning and Data Science Community. <https://www.kaggle.com/code/bigironsphere/loss-function-library-keras-pytorch/notebook>.
- [23] D. Godoy, "Understanding binary cross-entropy / log loss: A visual explanation", Towards Data Science, 2018. [En línea]. Disponible: <https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>
- [24] Musstafa. "Optimizers in deep learning". Medium. <https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0>.
- [25] "Objetivos de desarrollo sostenible". Naciones Unidas. <https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/>.

- [26] J. Kybic. "Laserbeam/Biomedical_Imaging_Algorithms_report.pdf at main · manelcardenas/Laserbeam". GitHub. https://github.com/manelcardenas/Laserbeam/blob/main/report/Biomedical_Imaging_Algorithms_report.pdf.
- [27] J. Hering. "Laserbeam". GitLab. <https://gitlab.fel.cvut.cz/biomedical-imaging-algorithms/laserbeam>.
- [28] J. M. Cardenas. "GitHub - manelcardenas/Laserbeam". GitHub. <https://github.com/manelcardenas/Laserbeam>.

Appendices

En esta sección haremos un breve resumen de los puntos más importantes del abstract de Jan Hering y Jan Kybic juntamente con los resultados que obtuvimos nosotros al trabajar con su modelo.

El dataset es el mismo con el que trabajamos en nuestro modelo (640 imágenes). Sin embargo, ellos dividieron 640 imágenes en 8 grupos, con una dificultad de segmentación creciente del grupo 1 al 8. Cada grupo consta de 80 imágenes. En la FIGURA A.1 podemos observar las diferentes dificultades que presentan las imágenes. El fondo cambia de color azul (FIGURA A-1, imagen A) a ocre (FIGURA A-1, imagen B). Aparecen elementos en el fondo (FIGURA A-1, imagen C) o imágenes sin impactos detectables (FIGURA A-1, imagen D)

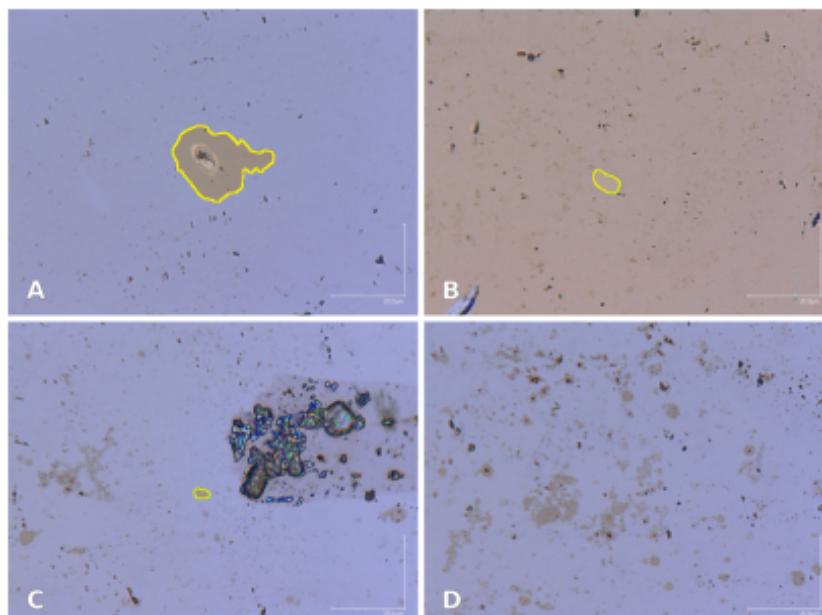


FIGURA A.1: Ejemplo de la colección de imágenes

El modelo de Jan Hering y Jan Kybic empieza realizando una ecualización del histograma de las imágenes de entrada para eliminar regiones *False Positive* (FP) que aparecían en los resultados de la segmentación.

En la FIGURA A.2 podemos ver en la primera fila una imagen del grupo 8 del dataset, a la derecha está la imagen sin ecualización de histograma, y a la izquierda está la misma imagen pero con el histograma ecualizado. En la segunda fila tenemos los resultados de la segmentación sin la ecualización (derecha), y con ella (izquierda).

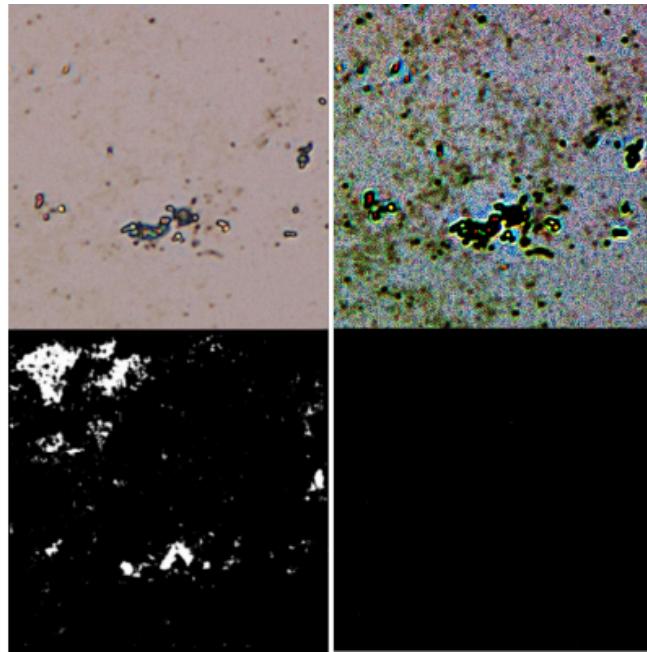


FIGURA A.2: Ejemplo ecualización histograma

Esta idea nos ha sido de gran utilidad en la realización de nuestro modelo, ya que nos enfrentamos al mismo problema y utilizamos la misma técnica de pre-procesado para resolverlo.

Para la arquitectura, el código Jan Hering y Jan Jybic también utiliza un modelo U-net. Para las métricas de evaluación, ellos aplican dos al mismo tiempo para entrenar el modelo, el Dice Score y la *relative area error*.

Con el modelo montado como hemos explicado, los resultados obtenidos con métricas para los diferentes grupos del dataset se muestran en la TABLA A.1.

Var	Group	n	Min	Median	Mean	Maz	SD
Relative error	F-scan 01	11	-0.11	-0.002	0.001	0.043	0.015
	F-scan 02	17	-0.025	-0.010	-0.010	0.008	0.008
	F-scan 03	22	-0.016	-0.005	-0.004	0.010	0.007
	F-scan 04	20	-0.049	-0.001	-0.002	0.039	0.017
	F-scan 05	11	-0.044	-0.002	0.088	0.823	0.254
	F-scan 06	18	-0.018	-0.002	0.004	0.103	0.028
	F-scan 07	12	-0.003	-0.044	0.086	0.40.9	0.119
	F-scan 08	14	-0.136	-0.502	0.567	1.000	0.423
	all	125	-0.136	-0.001	0.078	1.000	0.239

Dice score	F-scan 01	11	0.969	0.990	0.988	0.992	0.006
	F-scan 02	17	0.977	0.992	0.989	0.994	0.004
	F-scan 03	22	0.985	0.990	0.990	0.993	0.002
	F-scan 04	20	0.962	0.986	0.984	0.991	0.007
	F-scan 05	11	0.700	0.988	0.952	0.992	0.089
	F-scan 06	18	0.940	0.987	0.983	0.992	0.013
	F-scan 07	12	0.823	0.956	0.941	0.978	0.050
	F-scan 08	14	0.000	0.629	0.491	0.909	0.404
	all	125	0.000	0.988	0.924	0.994	0.205

TABLA A.1: Dice score and relative area error at 9*

Estos resultados son bastante parecidos a los que obtenemos con nuestro modelo al utilizar la métrica de Dice Score, así pues podemos comprobar como nuestro modelo realiza la segmentación de forma bastante satisfactoria. Sin embargo, como veremos a continuación, al realizar las simulaciones con el modelo de Jan Hering y Jan Kybic, los resultados no se correspondían con los mostrados en la TABLA A.1.

Vamos a analizar seis simulaciones, estudiaremos las diferencias entre los resultados obtenidos en estas y los teóricos mostrados anteriormente, a continuación discutiremos las anomalías encontradas.

1. En la simulación uno utilizamos exactamente los valores predeterminados del código de Jan Hering.
2. En la segunda simulación, solo se intenta reducir drásticamente los parámetros de entrenamiento para ver si realmente afectan a las medidas de la simulación.
3. Tercera simulación, esta vez, se mantienen la mayoría de parámetros de la simulación dos, pero se intenta mejorar los resultados.
4. Simulación cuatro, se modifican los parámetros de entrenamiento sin llegar a los originales para mejorar las medidas pero reducir costes de la simulación.
5. Tras los sorprendentes resultados de la simulación cuatro, esta simulación es un intento de optimización de los parámetros de la simulación cuatro para confirmar que realmente mejoramos las medidas de la simulación uno utilizando mucho menos espacio y tiempo de simulación.
6. Simulación seis, solo para asegurar que las métricas tienen sentido y que no se dan resultados aleatoriamente sino que tienen coherencia (para ser así, SIM. 6 ha de dar unos resultados muy malos en las tres métricas)

En la TABLA A.2 vemos un resumen de las métricas que modificamos en las seis simulaciones. El término *Unet_depth* que aparece en la tabla es el número de capas descendientes del modelo, siendo cuatro el valor original. Al reducir este número

siempre se deberían obtener mucho peores resultados ya que estaríamos omitiendo una gran cantidad de capas convolucionales.

	Num. Epochs	Unet_depth	unet_batch_size
SIM. 1	40	4	32
SIM. 2	20	2	16
SIM. 3	30	2	16
SIM. 4	20	4	16
SIM. 5	25	4	16
SIM. 6	5	2	4

TABLA A.2: Resumen de las métricas modificadas en las seis simulaciones

En la siguiente TABLA A.3 podemos observar las diferencias en espacio de memoria necesario para los diferentes indicadores de las simulaciones.

	Total Parameters	Input size (MB)	F/B pass size (MB)	Parameters size(MB)	Total Size(MB)
SIM. 1	13,240,162	12	11408494240.00	50.51	11408494302.51
SIM. 2	196,338	12	2684349264.00	0.75	2684349276.75
SIM. 3	196,338	12	2684349264.00	0.75	2684349276.75
SIM. 4	3,312,562	12	2852120392.00	12.64	2852120416.64
SIM. 5	3,312,562	12	2852120392.00	12.64	2852120416.64
SIM. 6	12,606	12	167770824.00	0.05	167770836.05

TABLA A.3: Resumen del espacio de memoria necesario en las seis simulaciones

Como podemos observar, lo primero que nos impacta es el volumen de memoria que requiere el modelo. Además, pese a simplificarlo al máximo en las simulaciones SIM. 2 y SIM. 6 sigue ocupando unas cantidades extremadamente grandes.

En esta tabla A.4 encontramos las diferentes métricas resultantes de las simulaciones.

	Duración de la simulación	Loss function	Dice score	Relative area error
SIM. 1	3h 29m	0.01761488	0.98836979	0.021422877
SIM. 2	1h 3m	0.02169229	0.98363192	0.071572738
SIM. 3	1h 40m	0.01892206	0.98620181	0.040633276
SIM. 4	1h 41m	0.01799988	0.99231160	0.013237981
SIM. 5	1h 13m	0.01508146	0.99233396	0.016249639
SIM. 6	11m	0.15791340	1.0	0.999936191

TABLA A.4: Resumen de las métricas resultantes en las seis simulaciones

Si comparamos los resultados de la TABLA A.4 con los de la TABLA A.1 podemos observar como en principio el Dice score y la Relative Area Error de la simulación uno con los resultados expuestos en el abstract de Jan Hering cuadran. Sin embargo, no tiene ningún tipo de sentido que al utilizar menos parámetros de entrenamiento los resultados sean mejores y más sabiendo que lo que se ha hecho es, por ejemplo, reducir el número de capas descendentes (*Unet_depth*) como en la SIM. 2.

Otra incongruencia muy importante que encontramos fue las diferencias al repetir una misma simulación. En la TABLA A.5 encontramos los resultados de las métricas de evaluación durante diferentes repeticiones. Como podemos observar, el tiempo necesario para cada repetición varía bastante y además, pese a que en este tipo de problemas siempre se introduce una pequeña variación, en este caso tenemos un factor de aleatoriedad. Podemos garantizar que esto no se debe a que el modelo no converge ya que en todas las simulaciones las gráficas de evolución de las métricas de evaluación y la loss function se parecen bastante a las de la FIGURA A.3.

		Simulation duration	Loss function	Dice score	Relative area error
Rep. 1	test_training	3h 29m	0.05217151	0.90499879	0.13562332
	evaluate_contour		0.01761488	0.98836979	0.02142287
Rep. 2	test_training	2h 50m	0.04780265	0.88397025	0.10490719
	evaluate_contour	-	0.01856321	0.96323423	0.03349234
Rep. 3	test_training	3h 21m	0.04542753	0.90999409	0.11627099

	<i>evaluate_contour</i>	-	0.01712891	0.98911050	0.02273672
Rep. 4	<i>test_training.py</i>	4h 33m	0.07605184	0.84257193	0.18132351
	<i>evaluate_contour</i>	-	0.02682607	0.98490507	0.05142706
Rep. 5	<i>test_training.py</i>	2h 37m	0.04234112	0.91532312	0.11612331
	<i>evaluate_contour</i>	-	0.012444899	0.99173702	0.01738719
Rep. 6	<i>test_training.py</i>	2h 37m	0.04039642	0.92530803	0.13470823
	<i>evaluate_contour</i>	-	0.01363707	0.99156523	0.01411260
Rep. 7	<i>test_training.py</i>	2h 36m	0.05689967	0.88107786	0.09917258
	<i>evaluate_contour</i>	-	0.01456118	0.99044876	0.01620476

TABLA A.5: Resultados de las métricas de evaluación durante diferentes repeticiones

Muchos de estos problemas se pueden llegar a solventar, pero el inconveniente con el proyecto de Jan Hering y Jan Kybic es que al intentar analizar el origen de estos problemas, el proceso se volvía muy complejo y muchas veces el modificar un aspecto del código o no provocaba ninguna alteración pese al cambio o provocaba unos cambios que no se correspondían con lo que se había modificado.

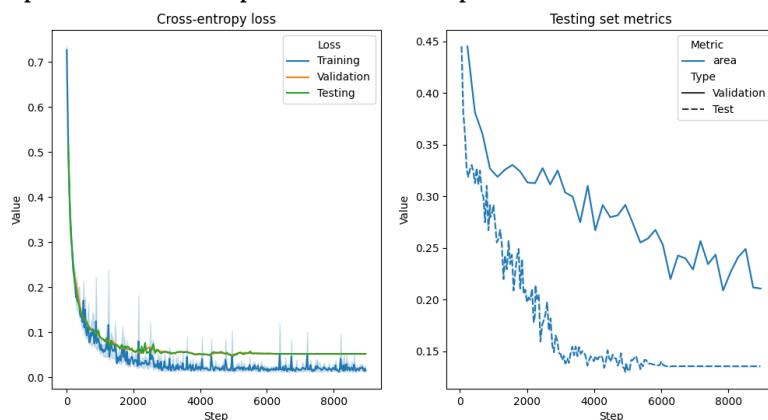


FIGURA A.3: Gráficas de evolución de las métricas de evaluación y la loss function

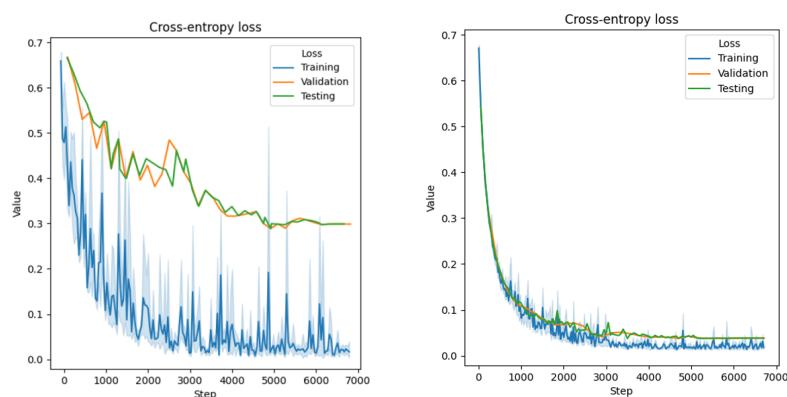


FIGURA A.4: Comparativa gráficas de cross entropy loss

Glosario

ML Machine Learning

DL Deep Learning

CV Computer Vision

DNN Deep Neural Network

CNN Convolutional Neural Network

FCN Fully Convolutional Network

CTU Czech Technical University

GPU Graphics Processing Unit

SSH Secure SHell

SS Semantic Segmentation

RGB/BRG Red Green Blue/Blue Red Green colour model

ReLU Rectified Linear Unit

SGD Stochastic Gradient Descent

TP True Positive

TN True Negative

FP False Positive

FN False Negative

IoU Intersection over Union