UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería de la Salud

Segmentación de Imágenes de Úlceras Por Presión mediante Aprendizaje Profundo

Pressure Ulcer Image Segmentation using Deep Learning algorithms

Realizado por
Pablo Aguilar Aldana

Tutorizado por
Rafael Marcos Luque Baena
Francisco Javier Veredas Navarro

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, Junio de 2022

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DE LA SALUD

Segmentación de Imágenes de Úlceras Por Presión mediante Aprendizaje Profundo

Pressure Ulcer Image Segmentation using Deep Learning algorithms

Realizado por
Pablo Aguilar Aldana

Tutorizado por
Rafael Marcos Luque Baena
Francisco Javier Veredas Navarro

Departamento
Lenguaje y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2022

Fecha defensa: julio de 2022

Abstract

Pressure ulcers are injuries to the skin caused by pressure or friction over a long period of time. These ulcers present five tissues, skin, periulcer, granulation, slough and necrotic, depending on the severity of the wound. Diagnosis is difficult to make as it is sometimes difficult to know exactly which tissues are present in the wound.

This work proposes the design and use of a simple and dynamic web application to solve and speed up these problems with diagnosis, so that health personnel can enter an image of an ulcer in the application so that it is returned, fully segmented into the different tissues present in the ulcer, by the application itself.

This application will have implemented a segmentation model developed using convolutional networks and deep learning methods (Deep Learning). This model has been selected after an exhaustive study of results among the different possibilities found in the set of PyTorch architectures. The architecture chosen was the famous Unet, known for its successful track record in dealing with medical images, with a pre-training with imagenet (dataset) as it has obtained the best result for each tissue with respect to the other segmentation networks. This architecture was the only one capable of detecting the necrotic stage, as this is, in most cases, the minority stage in this type of wounds.

At the end of this document, future improvements and an installation manual are proposed to achieve the full functionality of the application on any device.

Keywords: ulcers, segmentation, deep learning, tissues.

Resumen

Las úlceras por presión son lesiones producidas en la piel por una presión o fricción sufridas durante un gran periodo de tiempo. Estas úlceras presentan cinco tejidos, piel, periúlcera, granulación, esfacelos y necrótico, que dependerán de la gravedad de la herida. Su diagnóstico resulta complicado de concretar ya que en ocasiones es difícil de saber con exactitud que tejidos están presentes en la herida.

Este trabajo propone el diseño y uso de una aplicación web sencilla y dinámica para solventar y agilizar estos problemas con el diagnóstico, de modo que el personal sanitario pueda introducir una imagen de una úlcera en la aplicación para que esta sea devuelta, totalmente segmentada en los distintos tejidos que presente la úlcera, por la propia aplicación.

Esta aplicación tendrá implementado un modelo de segmentación desarrollado mediante redes convolucionales y métodos de aprendizaje profundo (*Deep Learning*). Este modelo ha sido seleccionado tras un estudio exhaustivo de resultados entre las distintas posibilidades encontradas en el conjunto de arquitecturas de PyTorch. La arquitectura escogida ha la famosa Unet, conocida por su exitoso recorrido en el trato con imágenes médicas, con un preentrenamiento con imagenet (conjunto de datos) ya que ha obtenido el mejor resultado para cada tejido con respecto a las demás redes de segmentación. Esta arquitectura ha sido la única capaz de detectar el estadio necrótico ya que este, en el mayor de los casos, es el estadio minoritario es este tipo de heridas.

Al final de este documento se proponen mejoras futuras y un manual de instalación para conseguir toda la funcionalidad de la aplicación en cualquier dispositivo.

Palabras clave: úlcera, segmentación, aprendizaje profundo, tejidos.

Índice

1 Introducción

1.1. Motivación

Las úlceras por presión son un problema que cada vez está más presente en nuestra sociedad, ya que principalmente aparecen en personas con cierto sedentarismo, ya sea por enfermedad, malos hábitos, o por la dificultad al moverse que provoca la edad. Desgraciadamente cada vez el sedentarismo es una corriente más común y más habitual en la sociedad actual, por lo que este problema incrementa su importancia [1].

El principal inconveniente que se les presenta a los sanitarios, con este tipo de úlceras, se da a la hora de planificar un tratamiento adecuado. Este tratamiento dependerá de muchos factores como pueden ser la edad del paciente, estado físico, patologías previas... pero principalmente dependerá de los distintos estadios que presente la úlcera. Esto es algo difícil de identificar por el personal clínico, ya que es un proceso laborioso y que requiere mucho tiempo, por lo tanto, dinero.

Dada esta situación resultó interesante la idea del trabajo, ya que gracias a la inteligencia artificial se podría acelerar muchísimo todo el procedimiento de segmentación en sí de la úlcera y, como consecuencia, se recetarían tratamientos más rápidamente. Además este enfoque se podría utilizar para el diagnóstico de muchas otras heridas, como úlceras vasculares, quemaduras.... Es innegable que la inteligencia artificial estará cada vez más inmersa en nuestras vidas, incluso en nuestro día a día.

En este contexto existen aplicaciones móvil como por ejemplo GuiaUPP (figura 1), que consiste en una aplicación que sirve de ayuda al personal clínico a la hora de clasificar, diagnosticar y tratar las úlceras por presión, pero todo de una forma más teórica.

Figura 1: App GuiaUPP.

Nuestra solución propone una aplicación web en la cual el usuario, en este caso el sanitario, pueda introducir una imagen de una úlcera, por ejemplo, sacada con el móvil, y automáticamente gracias al modelo de segmentación desarrollado, consiga una segmentación de la imagen en función de los distintos tejidos presentes en la UPP. Esto agilizaría mucho el proceso de diagnóstico y en consecuencia aumentaría la rapidez en la determinación de un tratamiento ajustado y adecuado para cada paciente además de abaratarse costes.

Por otro lado, en cuanto a la motivación personal, la ingeniería de la salud, se podría definir como el nexo de unión que existe entre la revolucionaria tecnología y el ámbito sanitario. Por ello este trabajo de fin de grado (TFG) resultó realmente interesante. El desarrollo del mismo podría decir que ha sido un ”reto” ya que es un trabajo quizás más orientado a la mención en Bioinformática, no a la propia Biomedicina, esto es uno de los principales motivos por los que se ha elegido este proyecto, ya que complementa muy bien el perfil de ingeniero biomédico, abriendo otro abanico de posibilidades y adquiriendo una base para futuros estudios más orientados a la informática, inteligencia artificial, ciencia de datos…

1.2. Objetivos

Este Trabajo de Fin de Grado (TFG) tiene como objetivo presentar una solución ante el alto coste y gran dificultad de diagnosticar y recetar un tratamiento adecuado a las ulceras por presión (UPP). Una Aplicación Web donde el sanitario puede subir una imagen de la úlcera que quiere estudiar y automáticamente obtenga una segmentación de los distintos estadios de la herida en cuestión. De forma que la interfaz será lo más sencilla, fácil de usar e intuitiva posible para evitar problemas. De forma que el profesional sanitario podrá acceder a la aplicación sin identificación e importar la imagen deseada.

Por lo tanto, tenemos dos objetivos claros:

- **Modelo de Segmentación**: Encontrar un modelo de segmentación con un alto rendimiento en la tarea de segmentación de los distintos tipos de tejidos presentes en las úlceras por presión.
- **App Web**: Desarrollo de una aplicación web intuitiva y fácil de usar que integre el modelo de segmentación mencionado anteriormente de manera que segmente la imagen que se introduzcan en la interfaz.

Tras estos dos objetivos más "técnicos.º en lo que a funciones se refiere, también se pretende alcanzar una serie de objetivos mediante la elaboración de este proyecto.

- **Eficiencia de la aplicación web**: Fluidez en la interfaz de la aplicación con el usuario que la está ejecutando.
- **Eficiencia del modelo**: Detección efectiva de los distintos estadios por parte del modelo de segmentación. Escogiendo el mejor de todas las pruebas previamente realizadas.

1.3. Fases del proyecto

En esta sección se detallan las fases de trabajo que se han seguido para la realización del proyecto. De modo que ha habido una documentación previa al desarrollo del código como se especifica en el Anteproyecto presentado en Febrero 2022.

1. **Estudio de redes de segmentación**: Fase de documentación inicial sobre funcionamiento de redes convolucionales. Estudio de los distintos algoritmos de Pytorch de segmentación, viendo sus aplicaciones.
2. **Preparación de datos**: A partir de unos ficheros de Matlab e imágenes obtenemos un conjunto de datos formado por 113 imágenes de úlceras con distintos estadios con sus respectivas 113 segmentaciones (soluciones).
3. **Ajuste de la arquitectura**: Ajuste de los pesos (pre-entrenamiento) de cada arquitectura, hiperparámetros y tamaños de imágenes.
4. **Entrenamiento inicial de los modelos**: Entrenamiento de los modelos previamente ajustados, tratando de optimizar al máximo los resultados obtenidos.
5. **Comparación de resultados obtenidos**: Comparar los resultados que hemos obtenido partiendo de algoritmos de segmentación y arquitecturas diferentes para la elección del mejor modelo para implementarlo en la aplicación.

6. **Desarrollo App**: Integración del modelo de aprendizaje computacional a una aplicación para que el usuario, el sanitario, pueda ejecutar el modelo introduciendo las imágenes que desee segmentar.

7. **Documentación**: Desarrollo de la memoria del trabajo de fin de grado.

1.4. Tecnologías usadas

En esta sección del documento se resumen las tecnologías que dotan de eficiencia al proyecto que se han utilizado para el desarrollo tanto de la aplicación como del modelo de segmentación.

1.4.1. Python

Figura 2: logo Python.

Python es el tercer lenguaje de programación más utilizado en el mundo y sin duda alguna es uno de los lenguajes más importantes del momento, debido a su rápido desarrollo y versatilidad de uso [2].
Su creador fue Guido Van Rossum, un informático holandés que diseñó este lenguaje y fue encargado de además de diseñarlo, pensar y definir las vías posibles de evolución del mismo. En un primer momento, este fue creado para darle una continuidad a ABC (lenguaje de programación de los 80s que se creó como alternativa a BASIC).

La primera versión de Python se publicó en Enero de 1994, tras este lanzamiento ha habido numerosas actualizaciones y cada vez ha ido abarcando más campos de aplicación y desarrollando sus características, por lo tanto se define actualmente como un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código fácil de interpretar y entender de modo que sea un lenguaje multiparadigma (permite crear programas usando más de un estilo de programación), capaz de aportar soluciones a diversos problemas diferentes como por ejemplo orientación a objetos, programación funcional y programación imperativa [3].
Es importante señalar que, al igual que JavaScript, Python es un lenguaje de tipado dinámico, es decir, el intérprete asigna a sus variables un tipo según lo que representen durante el tiempo

de ejecución [4].

1.4.2. PyTorch

Figura 3: logo PyTorch.

Pytorch es un entorno para *machine learning* basado en el lenguaje Python, también se denomina frameworks, estos frameworks permiten utilizar modelos muy complejos en pocas lineas de código.
Los frameworks permiten dar un marco y herramientas para facilitar la programación, por lo general, están desarrollados en " *open source* ", esto quiere decir que cualquier persona podría acceder y modificar el código. Con esto e consigue mantener la fiabilidad, transparencia y un mantenimiento continuo del código [5].
PyTorch se lanzó de forma oficial en 2016 por un equipo de investigación de Facebook (ahora Meta), se basa en la antigua biblioteca Torch y desde su creación se ha desarrollado en *open source*. Su principal objetivo es conseguir de una forma fácil y eficaz la implementación y entrenamiento de modelos de *Deep Learning* (explicado más adelante). Actualmente un 17 % de los desarrolladores de Python utilizan esta tecnología, entre ellos empresas como Tesla, Uber, etc.
Algunas de las ventajas que obtenemos al usar PyTorch son la siguientes:

- **Pytorch y Python**: Es cierto que la gran mayoría de trabajos de Inteligencia Artificial (IA) y *Machine Learning* se efectúan con ayuda de Python. Python y PyTorch son muy compatibles, lo que hace que los desarrolladores de Python tengan facilidad para familiarizarse con Pytorch y viceversa [5].
- **Fácil de aprender**: Sintaxis fácil e intuitiva [5].
- **Comunidad activa**: Este framework se ha desarrollado muy rápidamente construyendo así una gran cantidad de recursos en la red como vídeos o artículos, además de una documentación muy organizada y útil para principiantes [5].
- **Depuración fácil**: PyTorch está profundamente integrado en Python, tanto que algunas herramientas de depuración de Python se pueden usar fácilmente en PyTorch [5].

Más adelante explicaremos por qué hemos elegido Pytorch y no otras alternativas como TensorFlow o Keras.

1.4.3. Jupyter Notebook

Figura 4: logo Jupyter.

Jupyter Notebook es una aplicación cliente-servidor lanzada en 2015 por Proyecto Jupyter. Esta aplicación permite crear y compartir documentos, de modo que las celdas de sus archivos albergan texto en formato *Markdown* y código. Los documentos creados en Jupyter se pueden exportar en formatos HTML, PDF, Markdown y Python [6].

Los dos componentes principales de Jupyter Notebook son un conjunto de núcleos (Kernel) y el Dashboard. Cada núcleo o kernel es un motor de ejecución para un lenguaje que se encarga de procesar las solicitudes y devolver las respuestas apropiadas. El kernel por defecto es IPython, un intérprete de líneas de comandos que permite trabajar con Python. Además de este kernel, Jupyter cuenta con más de 50 kernels que le permiten trabajar con otros lenguajes de programación como pueden ser C++, JavaScript, php... [6].

Esta herramienta es muy útil porque lo usuarios pueden programar, documentar y ejecutar código además de visualizar los resultados en un mismo flujo. Algunos de los principales usos que tiene Jupyter Notebook son los siguientes [6]:

- **Depuración de datos:** Distingue entre los datos importantes y los que no lo son en un análisis de BigData.
- **Modelización estadística:** Método matemático que estima la probabilidad de distribución de una característica específica.
- ***Creación y entrenamiento de aprendizaje automático:*** Esta es nuestra aplicación de interés, diseño, programación y entrenamiento de modelos basados en aprendizaje automático.
- **Visualización de datos:** Representación gráfica de datos para visualizar tendencias.

1.4.4. Anaconda

Figura 5: logo Anaconda.

Anaconda es una distribución de los lenguajes de programación Python y R. Posee más de 720 paquetes de código abierto. La gran ventaja que presenta este software es la simplificación de la gestión e implementación de los paquetes [7].
Es muy útil ya que muchos paquetes científicos dependen de versiones específicas de otros paquetes, conda actúa como administrador de paquetes y como un administrador de entorno. Esto hace que los usuarios estén seguros de que cada versión de cada paquete tenga todas las dependencias que quiere y funcione correctamente [7].
Entre sus características principales podemos destacar algunas: es multiplataforma, por lo tanto puede instalarse en Linux, macOS y Windows, permite instalar y administrar paquetes fácilmente, Anaconda Navigator es una interfaz gráfica de usuario GUI bastante sencilla pero con un potencial enorme, etc [7].
Algunas de las aplicaciones que se encuentran de forma predeterminada en el navegador son:

- Jupyter Lab
- Jupyter Notebook
- Spyder
- Datalore
- Pycharm
- Glueviz
- RStudio

Principalmente de nuestro interés será **Jupyter Notebook**, explicado anteriormente.

1.4.5. Flask

Figura 6: logo Flask.

Para definir Flask, previamente debemos definir qué es un Framework.
Los Frameworks son herramientas que nos dan un esquema de trabajo y una serie de utilidades y funciones que nos facilita y nos abstrae de la construcción de softwares en general. Estos Frameworks están asociados a lenguajes de programación y el más conocido en el mundo de Python es Django, pero Flask es muy competente pese a no tener una curva de aprendizaje tan elevada como Django.[8]
Flask es un microFramework escrito en Python y concebido para facilitar el desarrollo de aplicaciones web bajo el patrón MVC (manera de trabajar que permite diferenciar y separar lo que es el modelo de datos, la vista y el controlador). La palabra micro hace referencia en que con este framework podemos crear aplicaciones web funcionales, pero si en algún momento es necesaria una nueva funcionalidad se deberán instalar una serie de extensiones (*pluggins*) [8].
¿Por qué usar Flask?
A continuación se enumeran una serie de razones por las que se ha optado por usar este Framework para el desarrollo de la aplicación:

1. Flask es un ”micro” Framework, para desarrollar una App básica puede ser muy conveniente.

2. Incluye un servidor Web de desarrollo: Las aplicaciones se pueden correr en servidor de manera sencilla para comprobar los resultados que se van obteniendo.

3. Es compatible con Python3.

4. Buen manejo de rutas.

5. Es compatible con wsgi, un protocolo que utiliza servidores web para servir a las páginas web escritas en Python.

6. Buena documentación.

1.4.6. HTML

HTML es el lenguaje con el que se define el contenido de las páginas web. Con el se pueden definir el texto, imágenes, listas, vídeos y otros elementos que conforman una página web. Es un lenguaje de marcación de elementos para la creación de documentos hipertexto, fácil de aprender, lo que permite que cualquier persona poco iniciada en el mundo de la programación pueda enfrentarse a la tarea de crear una web [9].
Para el desarrollo de este software se ha utilizado *Visual Studio Code* como editor de avanzado de código. Este archivo de texto tiene una característica peculiar, tiene que estar guardado con la extensión .html o .htm, no con .txt. Este lenguaje tiene muchas etiquetas de modo que por ejemplo con el comando <B>podremos resaltar el texto que queremos que esté en negrita, así hay muchas etiquetas más. En resumen, HTML no es más que una serie de etiquetas que se utilizan para definir el contenido del documento y algún estilo básico.

1.4.7. CSS

"Hojas de Estilo en Cascada" (*Cascading Style Sheets*). Básicamente es un lenguaje que maneja el diseño y la presentación de las páginas web, es decir, cómo lucen cuando un usuario las visita. Funciona en bloque junto con el lenguaje HTML, que se encarga más del contenido básico de las páginas web [10].
La utilidad de CSS se basa en que puedes crear reglas para decirle a tu sitio web como quieres mostrar la información y puedes guardar los comandos para elementos de estilo separados de los que configuran el contenido.
Algunas de las ventajas de usar CSS son las siguientes:

- Optimiza la edición
- Facilita la accesibilidad del usuario
- Promueve la creatividad
- Prioriza la limpieza del código

Diferencias HTML y CSS

HTML da estructura al contenido de un sitio web, significa "lenguaje de marcas de hipertexto" y hace referencia al código que define el significado de las instrucciones dadas a una plataforma computacional.
CSS llega para hacer más eficiente el uso de HTML gestionando los datos relacionados con el diseño visual de las plataformas y CSS es uno de los lenguajes más importantes utilizados para ordenar las instrucciones referentes a al diseño de la web, de manera que hace la página web más atractiva [10].

En resumen HTML estructura el contenido de un sitio web mientras que CSS se usa para darle una presentación más atractiva a la página para el usuario que la utilice.
En la figura 7 se muestra un esquema que se seguirá para el desarrollo de la aplicación.

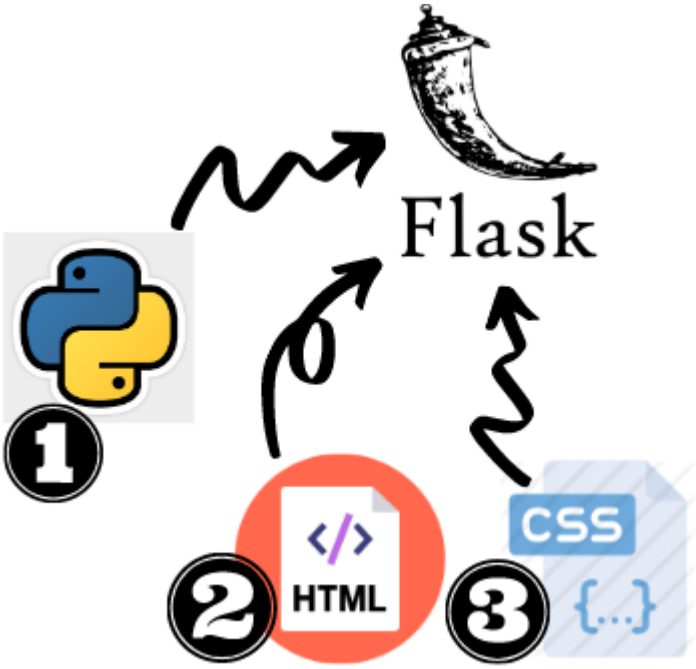

Figura 7: Formantes App.

1.5. Úlceras Por Presión (UPP)

Las úlceras por presión (UPP) son lesiones en la piel producidas por presión o fricción entre planos duros, provocando un bloqueo sanguíneo, una isquemia, que produce una degeneración de los tejidos. Este tipo de heridas o lesiones se producen en personas con movilidad reducida por diferentes causas: accidente, estados de coma, parálisis total o parcial, dificultad para hacer cambios posturales, etc. [11][1]. Son catalogadas como un problema de salud de primer orden, ya que tienen una gran incidencia y un elevado coste de tratamiento, además las personas que las padecen sufren una descenso en la calidad de vida muy significativo.
Las úlceras por presión afectan a diferentes tejidos, estos son los siguientes:

- **Piel**: tejido sano que rodea a la úlcera [12].
- **Periúlcera**: Indica la etapa de cicatrización de la úlcera, suele rodear a la herida [12].
- **Granulación**: Afecta al tejido conectivo fibroso, este tiene un color rojo brillante o rosa oscuro y sirve como indicador de crecimiento del tejido nuevo durante el proceso de cicatrización de la herida [13] [12].

- **Esfacelos**: Alto contenido de bacterias, amarillo, blanquecino o verdoso.
- **Necrótico** o **Necrosis**: Es el peor estado, colores oscuros incluso negros. Existe un bloqueo sanguíneo y la muerte del tejido [12].

En la actualidad los sanitarios se guían por los colores de cada estadio para su identificación. Como se ha comentado anteriormente, las úlceras por presión aparecen cuando hay una conducta sedentaria severa y prolongada, ya sea de forma voluntaria o forzada por recuperación post-operatoria, incluso se han visto casos de úlceras que se han formado por corsés ortopédicos. En la figura 8 se muestra un esquema de las zonas de aparición más frecuente de las úlceras por presión.

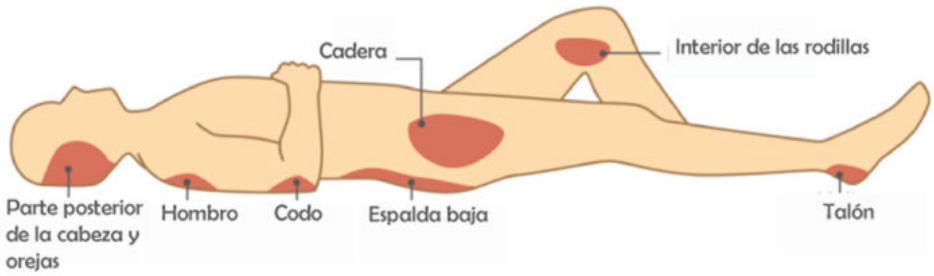

Figura 8: Zonas más comunes de las úlceras por presión (UPP).

2 Antecedentes

2.1. Inteligencia Artificial (IA)

La Inteligencia Artificial (IA) es la simulación de procesos de seres inteligentes (humanos o no) por parte de máquinas, especialmente sistemas informáticos. Estos procesos incluyen el aprendizaje, el razonamiento y la autocorrección. Las aplicaciones más comunes y más desarrolladas de esta tecnología incluyen sistemas expertos, reconocimiento de voz y visión artificial [14] [15].

Estas tecnologías se usan actualmente par ayudar a los humanos a beneficiarse de mejoras significativas y disfrutar de una mayor eficiencia en casi todos los ámbitos de la vida. Como todo, la IA y más específicamente, su desarrollo, tiene desventajas o riesgos, lo que hace que debamos de estar atentos al desarrollo y así ser capaces de prevenir posibles desventajas directas o indirectas.

Algunos de los campos que más se están desarrollando dentro de la IA son los siguientes:

- Reconocimiento de imágenes estáticas, clasificación y etiquetado: Este es nuestro ámbito de aplicación y sobre todo puede tener una buena compatibilidad con la industria médica, para detectar patologías [14].

- Mantenimiento Predictivo: Técnica que utiliza herramientas y técnicas de análisis de datos para detectar anomalías en el funcionamiento, procesos o posibles defectos en las máquinas/equipos, de modo que se puedan solucionar antes de que sobrevenga el fallo general [14] [16].

- Detección y clasificación de objetos: Esto se está usando y desarrollando intensamente en la industria automovilística para leer las señales de tráfico o detectar peatones [14].

- Protección para amenazas de seguridad cibernética [14].

Otro de los principales beneficios de la IA es que permitirá que las máquinas y robots realicen tareas que los humanos consideran difíciles, aburridas o peligrosas, en algunos casos se podrá realizar aquello que hasta entonces para el ser humano parecía imposible [14].

2.2. Machine Learning

Uno de los principales conceptos que se debe introducir es el *Machine Learning*, este se define como una rama de la inteligencia artificial (IA) y la informática que se centra en el uso de datos y algoritmos para imitar la forma en la que aprende la inteligencia natural, mejorando gradualmente su precisión [17].

El *Machine Learning* es un componente clave en campos como la inteligencia artificial y la ciencia de datos. Los algoritmos propios de *Machine Learning* se aprenden automáticamente de los datos de entrada (autoaprendizaje) [17].

En resumen podríamos decir que los algoritmos de *Machine Learning* aprenden a realizar una tarea de forma autónoma, mejorando sus resultados con el tiempo.

Podemos definir 4 etapas en el desarrollo de un modelo de *Machine Learning*:

1. **Selección y preparación de datos de entrenamiento**: Se escoge un conjunto de datos que sirve para ”alimentar” a ese algoritmo de *Machine Learning* para que aprenda a resolver un problema específico. Podríamos clasificar el conjunto de datos de entrenamiento de la siguiente manera [17]:
    - Etiquetados: De manera que el modelo aprenderá las características que debe identificar.
    - No etiquetados: Será el modelo quien decida qué características se deben detectar y extraer.
2. **Selección del algoritmo**: Este algoritmo dependerá del tipo de problema para el que haya que resolver y del tipo de datos sobre los que se ejecute [17].
3. **Entrenamiento**: El algoritmo se ejecuta repetidamente sobre el conjunto de datos de entrenamiento, donde se comparará la salida de este real con lo que debería de haber devuelto. **Este sería el modelo de *Machine Learning*** [17].
4. **Uso y Mejora**: Se utiliza el modelo entrenado para resolver el problema para el que ha sido utilizado [17].

Es importante introducir los tipos de *Machine Learning*:

1. Aprendizaje Supervisado: Este es el tipo de algoritmo que se usa en este proyecto, de modo que los datos han sido etiquetados previamente como tendría que ser categorizada la nueva información, por lo tanto este método requiere de la intervención humana para el etiquetado de datos [14].
2. Aprendizaje No Supervisado: Es este método es el propio algoritmo el que elige que características, encuentran una manera de clasificación por ellos mismos [14].

3. Aprendizaje de refuerzo: Este tipo de algoritmos aprenden por experiencia, de modo que hay que darles un refuerzo ”positivo” cada vez que aciertan [14].

En la figura 9 se muestran los tipos de aprendizaje automático que existen.

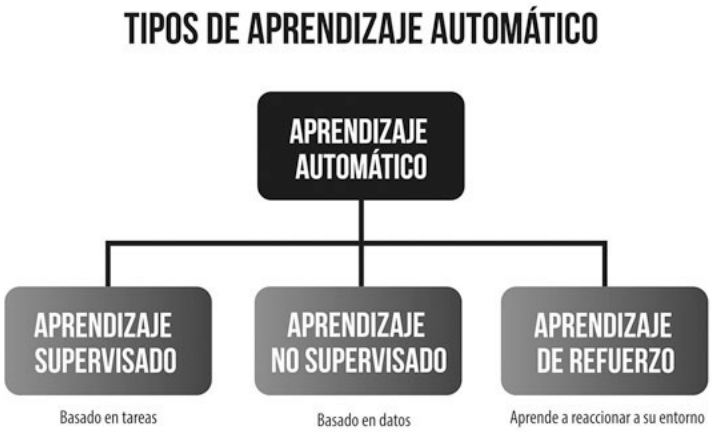

Figura 9: Tipos de Machine Learning.

2.2.1. Deep Learning

El *Deep Learning* (Aprendizaje profundo) es una de las aplicaciones más poderosas y de mayor crecimiento de la inteligencia artificial, este se trata de un subcampo del *Machine Learning* (Aprendizaje automático) que se utiliza para resolver problemas muy complejos y que normalmente implican grandes cantidades de datos. Este aprendizaje profundo de redes neuronales que se organizan en capas, la característica principal de este tipo de aprendizaje es la existencia de capas ”internas” o ”profundas” que hacen posible la resolución de problemas más complejos [18].

En la figura 10 se puede apreciar visualmente la diferencia entre una red neuronal simple con una basada en aprendizaje profundo.

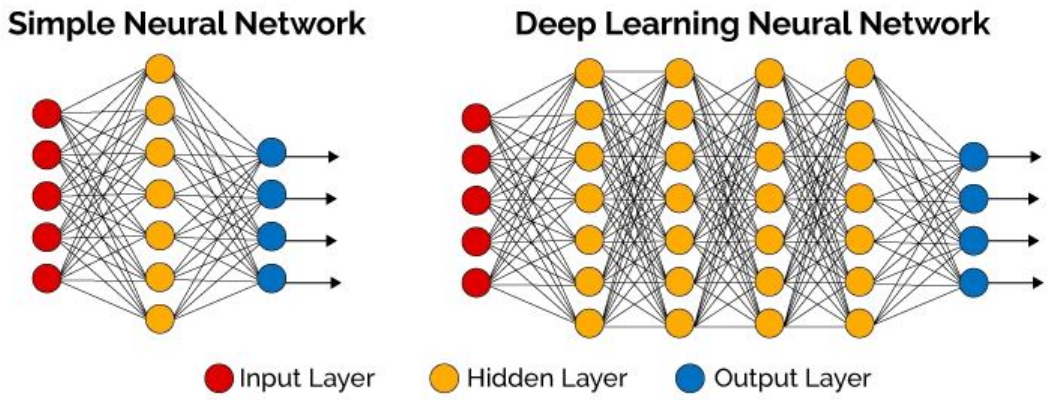

Figura 10: Comparación de red neuronal simple con una basada en deep learning.

El *Deep Learning* es principalmente usado para la detección de objetos, predicciones y traducciones de texto. Lo que diferencia a una red profunda de una que no lo es, es principalmente el número de capas sobre las que esta trabaja, además el aprendizaje profundo implica otra serie de mecanismos además del *back propagation*, como son las técnicas de regulación (*drop out*) y tipo de unidades (*RELU*) [19].

2.3. Redes Neuronales

Todas las actividades de investigación en el estudio de redes neuronales artificiales se basan en una búsqueda de emular la forma de procesamiento de datos que tiene el cerebro humano, algo que es completamente distinto al funcionamiento de un computador digital [20].

El cerebro humano es un sistema altamente complejo y que a día de hoy no se ha conseguido comprender su funcionamiento completamente, y podríamos definir su funcionamiento como complejo, no lineal y paralelo. Es capaz de realizar varias operaciones al mismo tiempo, algo diferente a los computadores que la mayoría trabajan de forma secuencial [20].

El elemento básico de un sistema neuronal biológico es la neurona, millones de neuronas se organizan en capas donde están conectadas unas con otras a través del axón y sus ramificaciones, estas neuronas van procesando información e interpretándola siendo capaces de generalizar conceptos a partir de casos particulares [20].

Por lo tanto, son tres los conceptos claves que se pretenden emular:

1. Procesamiento paralelo: Capaz de procesar datos de distinta forma al mismo tiempo [20].

2. Memoria distribuida: En las redes neuronales biológicas la información no se guarda en una posición de memoria definida como ocurre en los computadores, si no que esta está distribuida por la red neuronal de modo que si aparece algún fallo o deterioro en el funcionamiento de una región no se pierda toda la información [20].

3. Adaptabilidad al entorno: Esta cualidad hace posible generalizar conceptos desde casos particulares, posibilita el aprendizaje [20].

La arquitectura de las redes neuronales hace referencia al patrón de conexión de una red neuronal. Estas redes neuronales pasan la información de manera direccional, es decir, de las neuronas pre-sinápticas a las post-sinápticas, las neuronas se agrupan en unidades que denominamos capas, por lo tanto una red neuronal se compondrá de una o más capas [20].

Podemos clasificar estas capas en tres grupos distintos:

1. Capas de entrada: donde se encuentran las neuronas que reciben los datos iniciales.

2. Capas ocultas: Son las capas que no tienen una conexión directa con el entorno, aporta a la red neuronal grados de libertad para representar determinadas características.

3. Capas de salida: Son las neuronas que proporcionan una respuesta de la red neuronal.

En la Figura 11 se muestra un esquema de la distribución de capas en una red neuronal (profunda).

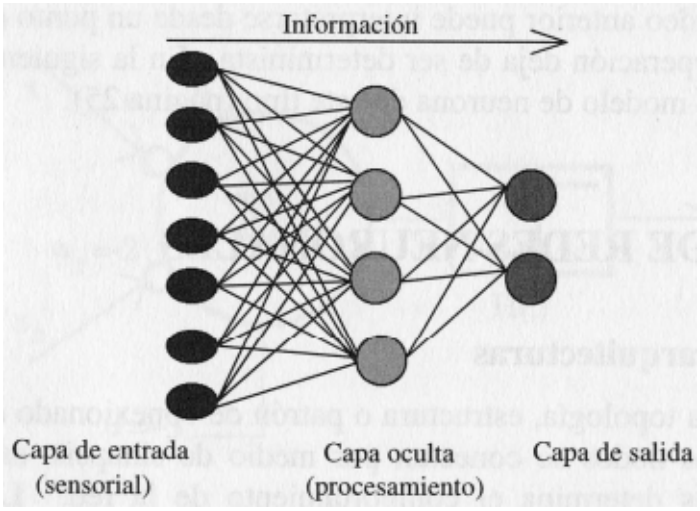

Figura 11: Arquitectura de una red neuronal con tres capas.

2.3.1. Redes Convolucionales

Las redes convolucionales se asemejan un poco más al comportamamiento y procesamiento de datos del cerebro humano, pongamos el ejemplo de la visión, de manera que en primer lugar se distinguirán formas simples, después bordes y colores de esas formas y por último qué es exactamente, llegando a simular en cierto modo la visión humana [21].
La aplicación de las redes neuronales convolucionales son especialmente efectivas para tareas de visión artificial, segmentación de imágenes y clasificación. Consisten en múltiples capas

llamadas filtros, donde en un primer momento se extraen las características de interés, después se intentan mejorar y por último se evalúan [21].
Las redes neuronales convolucionales son capaces de aprender a reconocer una gran diversidad de objetos dentro de una misma imagen pero para lograr esto es necesario una fase previa de ”entrenamiento” con una gran cantidad de ”muestras” para que sea capaz de captar las características únicas de cada objeto y generalizarlo, esto es lo que se denomina como aprendizaje de un algoritmo [21].
Una imagen está formada por un conjunto de píxeles, estos píxeles son tomados como valor de entrada por la red. Los píxeles tienen un valor entre 0 y 255 por lo que se normalizan para que este valor se mueva entre 0 y 1, esto ocurre cuando tenemos imágenes de solo 1 canal (Escala de grises), si tenemos imágenes a color tendremos 3 çapasüna para el rojo, otra para el verde y otra para el azul (RGB) [21]. (Ejemplo en la figura 12)

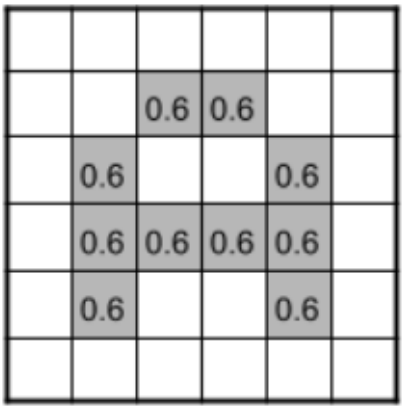

Figura 12: Ejemplo de imagen de entrada normalizada.

A esta imagen de entrada (figura 12) se le aplicarán una serie de filtros, por los cuales se completará una matriz de menor tamaño. El proceso de generar esta matriz sería la primera convolución de este ejemplo. En la figura 13 se puede apreciar el funcionamiento de la aplicación de filtros.

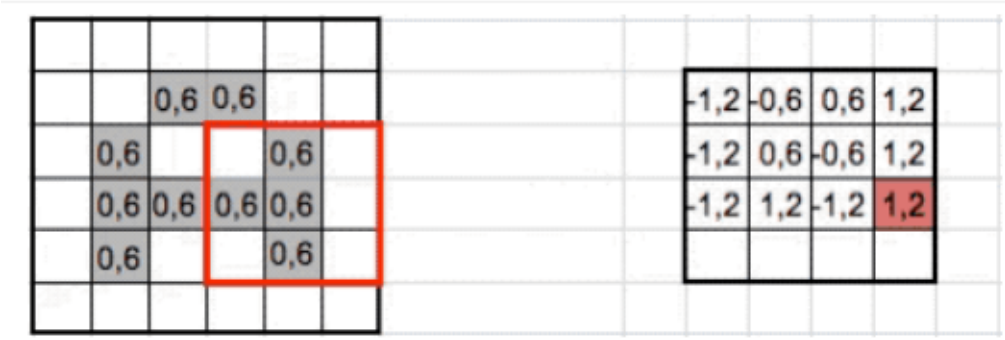

Figura 13: Aplicación de filtro.

Tras esta aplicación tenemos una matriz resultante más pequeña (que generaliza la imagen por zonas) a la cual le aplicaríamos una función de activación que determinaría el mapa de detección de características, Una de las funciones más usadas de activación es la función ReLu (*Rectifier Linear Unit*, $f(x) = max(0, x)$) la cual da el valor 0 a todos los píxeles (de la nueva matriz) con valores negativos [21]. La figura 14 muestra un esquema general de esta transformación.

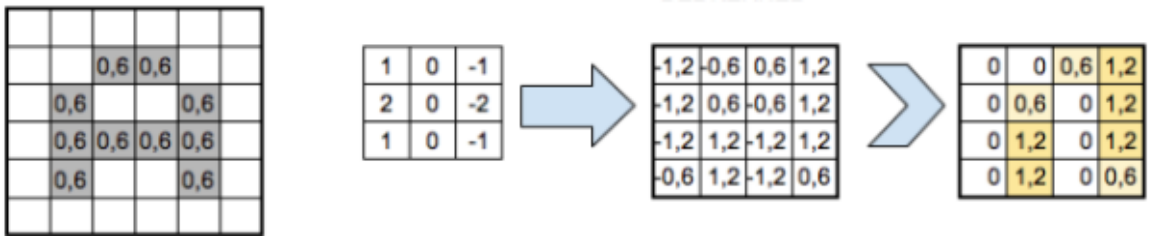

Figura 14: Esquema general.

2.4. Clasificación, Detección y Segmentación

Para la correcta comprensión de este trabajo es necesario diferenciar claramente tres conceptos que con cierta frecuencia se entienden como sinónimos pero que realmente no lo son. Estos tres conceptos son explicados detenidamente a continuación.

2.4.1. Clasificación de imágenes

Consiste en decir si una imagen pertenece al grupo x o al grupo y, si es o no es de una determinada clase.
Supongamos el ejemplo de clasificar las imágenes entre imágenes de gatos y de perros. Para un correcto entrenamiento del modelo se le deberían proporcionar una gran cantidad de imágenes de perros y de gatos, por separado. A partir de estas imágenes el algoritmo conseguirá

detectar características únicas en las imágenes de gato y de perro, de modo que será capaz de clasificar una imagen como perteneciente al conjunto de gatos aunque el gato sea de un color u otro, o incluso este de una forma u otra, lo mismo para las imágenes de los perros [22].
Estas redes neuronales tienen una gran cantidad de capas ocultas (convoluciones) que van aplicando filtros a la imagen para conocer cada vez formas y características más complejas como hemos comentado anteriormente. A la última capa (convolución) pasará por un proceso de *subsampling*, donde deberán prevalecer las características más importantes que se han detectado en cada filtro, esta última capa es la capa que conecta con la capa de salida [22].
Esta capa de salida tendrá dos neuronas (gatos y perros) y será de la forma [0,1] para las imágenes de gatos y [1,0] para las de perros, ya que este sería un problema de clasificación binaria (una clase u otra). Por lo tanto si proporciona una salida de [0,2 0,8] quiere decir que la imagen tiene un 20 % de probabilidades de ser del conjunto de imágenes de perro y un 80 % de las de gato, por lo que se catalogará como imagen de gato [22] [23].

2.4.2. Detección de objetos

Como su nombre indica consiste en detectar algún objeto determinado en las imágenes, muy usado actualmente en automóviles con conducción autónoma.
Para detectar un objeto es necesario primero localizar un objeto en una imagen y posteriormente declarar que un objeto pertenece a la clase buscada. Este tipo de problemas se afrontan mediante redes neuronales convolucionales, donde se requiere que el modelo detecte una cantidad limitada o específica de objetos en una imagen. Este tipo de algoritmos tienen una serie de características que lo diferencian [24]:

1. Detecta múltiples objetos.

2. Da la posición del objeto que detecta, ya sea por rectángulos, indicando su centro, contorno…

3. Detectar rápidamente, ya que por ejemplo en la industria automovilística este tipo de algoritmos tiene que poder detectar en ”tiempo real”, vídeo, un objeto lo más rápido posible para que el vehículo en este caso haga sus correspondientes acciones.

En este tipo de algoritmo los datos de entrenamiento deberán tener una clasificación del objeto, si es gato o perro siguiendo el ejemplo anterior, y además la posición de la imagen (X,Y) y el ancho y alto del objeto. Por lo tanto este tipo de problemas a diferencia de los de clasificación tendrán como salida una respuesta como esta [0 1 50 50 47 89] si se trata de un solo objeto [24] [25].
Basándonos en el problema del apartado anterior vemos que una imagen se clasificaba según si aparecía un gato o un perro, pero… ¿Qué pasa si aparecen dos gatos en la misma imagen?

¿Y dos perros? ¿Y un gato y un perro? ... De modo que con el método anterior una vez que el algoritmo detecte un objeto (perro o gato) dará la respuesta para su clasificación, esto no es interesante para este problema [24] [25].
En este caso el algoritmo debe de hacer una iteración por la imagen, de modo que vaya recorriendo la imagen (poco a poco), analizándola por medio de rectángulos por ejemplo, buscando todos los objetos de una determinada clase y una vez que la haya recorrido completamente sin encontrar algún objeto pasaría a recorrer de nuevo la imagen pero buscando otro tipo de objeto (primero perros y luego gatos, por ejemplo). Supongamos un ejemplo (figura 15) [24]:

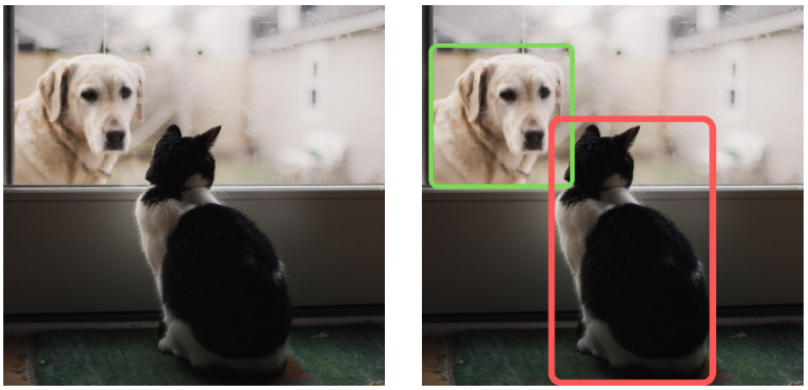

Figura 15: Imagen de entrada y resultado esperado.

Ahora empezaría la iteración para encontrar al primer animal (perro) (figura 16):

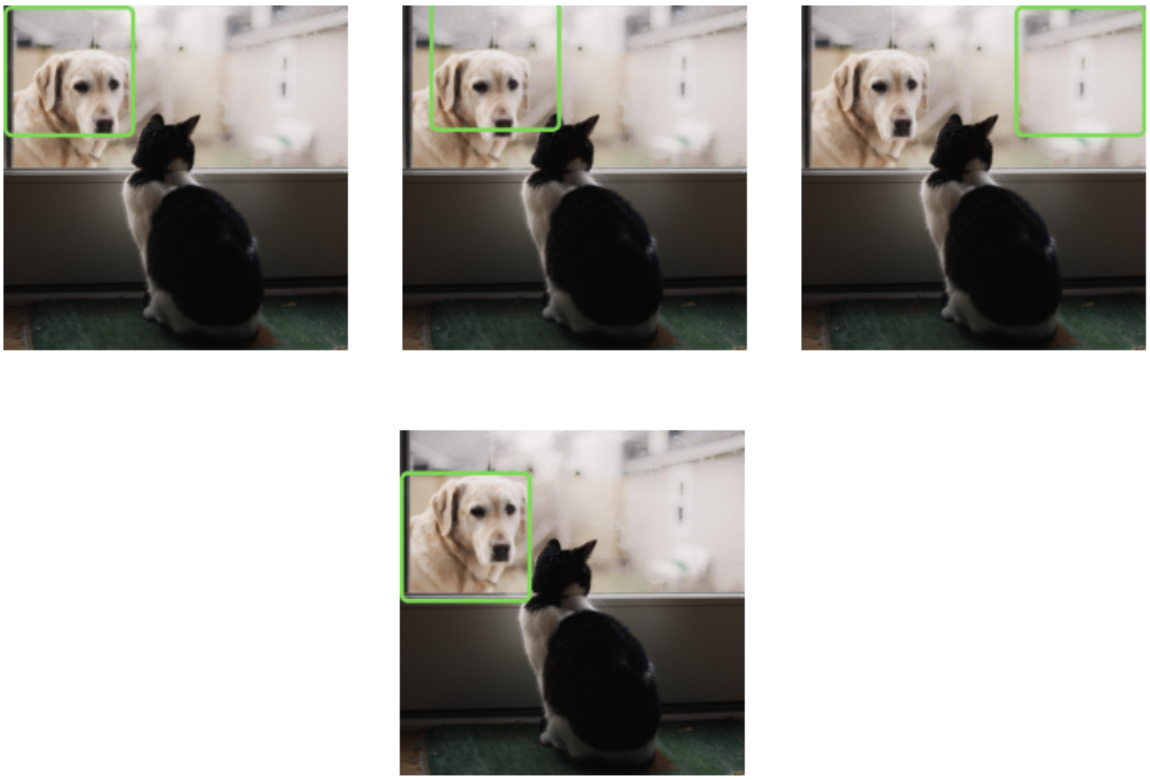

Figura 16: Detección de perro.

Una vez que se hayan detectado todos los objetos ”perro” de la imagen pasamos a otra clase, gato (figura 17).

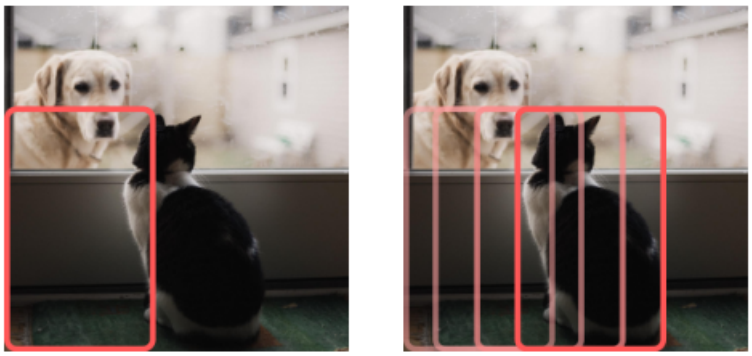

Figura 17: Detección de gato.

Es posible que se detecten dos perros en la misma caja, que un objeto se detecte para dos cajas distintas... para solventar este tipo de problemas se utilizan métricas específicas para este tipo de algoritmos como son mAP, donde se evalúa al mismo tiempo si la clase del objeto es la correcta y si la posición de la caja es la adecuada. Para evitar problemas de solapamiento entre las ”cajas de gato” y ”las cajas de perro” se utilizan otras métricas como son el IoU, métrica

además usada en el tema del proyecto [24] . En la figura 18 se muestran de forma visual los problemas comentados anteriormente.

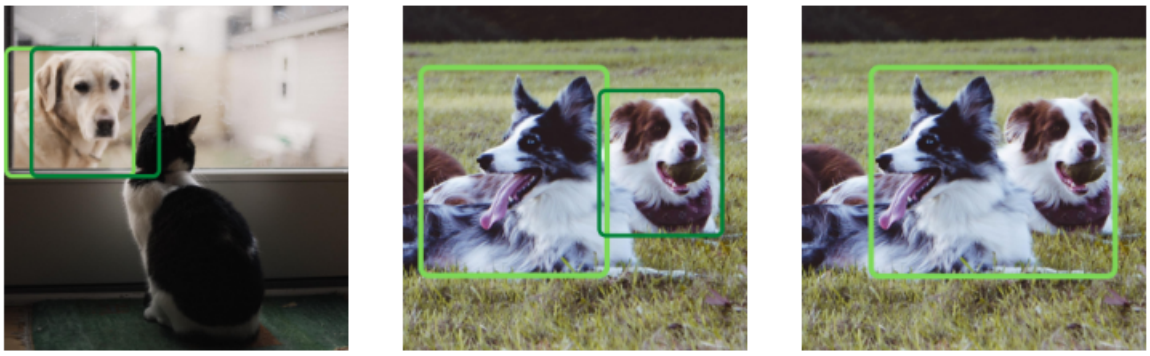

Figura 18: Posibles problemas.

2.4.3. Segmentación de imágenes

Es donde se basa este trabajo y consiste en desglosar una imagen según ciertas características.
La segmentación semántica hace referencia a la clasificación por píxel de una imagen, para esto es necesario que las imágenes de entrada tengan todos sus píxeles clasificados en sus correspondientes máscaras (matriz de la misma dimensión que la imagen correspondiente donde cada valor, cada píxel, hace referencia a la clase a la que pertenece realmente [0,1,2,3...]) de este modo el algoritmo será capaz de hacer una clasificación por píxeles de una imagen tras entrenar con un conjunto de datos de imágenes reales con sus respectivas máscaras, soluciones [26].
Un ejemplo de este tipo de segmentación es el que se da en la conducción autónoma, de manera que se combina con la detección de objetos ya que un automóvil requiere de una buena comprensión de la escena que tiene, diferenciar entre peatones, señales, semáforos, etc. Esto se puede apreciar mejor en la figura 19[26].

Figura 19: Ejemplo de Segmentación en conducción autónoma.

Este tipo de algoritmos como se puede apreciar permite crear segmentos dentro de una misma imagen y atribuir un significado semántico a cada una de estas regiones atribuyéndole a cada píxel el color correspondiente según la clase a la que pertenezca. Este tipo de algoritmos se centran en las redes neuronales convolucionales, donde se ha explicado que las convoluciones se utilizan para sacar las características de una imagen/región, extraer información relativa a los patrones visuales, esta información es más compleja en función de la cantidad de capas de convolución por las que haya pasado, siendo las últimas capas las que capturan el contenido semántico, es decir, la información de la imagen [27].

3
Metodología

En este apartado se detallarán los distintos pasos seguidos para la elaboración del proyecto. Donde en primer lugar se explicará el conjunto de datos utilizado para el entrenamiento, validación y testeo del modelo del segmentación, el aprendizaje del modelo.
Luego se explicará el marco experimental que ha sido aplicado en este proyecto, de manera que se ha intentado que el modelo sea lo más eficiente posible, obteniendo así buenos resultados en el menor tiempo posible, ya que las ejecuciones de los modelos de segmentación desarrollados para este proyecto han sido de horas debido, en gran parte, a la capacidad de tarjeta gráfica de la máquina donde se han ejecutado (Nvidia 1060, 6GB).
Más adelante se expondrán los modelos de segmentación propios de PyTorch y se explicará cuales han sido los seleccionados para el desarrollo y comparativa.
Para terminar se hará una evaluación de los resultados de los modelos entrenados, esta evaluación constará de unos resultados cualitativos (cómo de ”bien” segmenta el modelo) y cualitativos (métricas aplicadas para su evaluación, porcentajes de acierto). En esta sección se terminará justificando por que se ha elegido un determinado modelo.

3.1. Conjunto de datos

Como se ha explicado anteriormente en este documento (véase sección 2.4.3) los modelos de segmentación necesitan para su correcto aprendizaje partir de dos tipos de datos:

- Imágenes de úlceras por presión de un tamaño de (1224x1632).
- Máscaras de estas imágenes con la correspondiente clasificación de la imagen original por estadios. Son de la misma dimensión que la imagen a la que corresponden y cada píxel (posición) de la imagen tiene asignado un valor que se le asociará a un determinado estadio de esta herida.

El conjunto de datos del que se parte está formado por 113 imágenes de úlceras y sus correspondientes 113 máscaras. Un ejemplo sería el siguiente (figura 20):

Figura 20: Imagen y máscara asociada.

Como se ve puede apreciar en la imagen anterior, en la máscara quedan delimitados los bordes de cada tejido, esto ocurre gracias a la función para visualizar estas imágenes que se ha implementado en Python. Si abrimos la imagen correspondiente a la máscara desde el equipo tendremos una imagen en negro, ya que esta imagen es una matriz de números de la misma dimensión de la imagen original, en la cual los valores de píxel van desde el 0 al 5, correspondiente cada valor a un estadio de la úlcera, sabiendo que un píxel de una imagen normal tiene valores entre 0 y 255, los de la máscara son tan próximos a 0 que se muestra cómo una imagen completamente oscura.

3.2. Modelos de Segmentación PyTorch elegidos

Para el desarrollo de este trabajo se han valorado todas las siguientes arquitecturas de PyTorch para la segmentación de imágenes:

- Unet: Red convolucional para la segmentación de imágenes biomédicas. Esta es una red neuronal que se puede entrenar completamente sin un gran número de imágenes. Es una red rápida y eficiente [28].

- Unet++: Mejora de la arquitectura presentada anteriormente Unet, con más peso. En los estudios se ha demostrado que obtienen valores de coeficiente IoU (véase sección 3.3.2) entre 3.9 y 3.4 puntos mayores que usando su antecesora, Unet [29].

- MAnet: Creado como alternativa a Unet. Buenos resultados en el conjunto de datos de MICCAI 2017 LiTS Challenge, obteniendo unos resultados de coeficiente Dice de 0.960 con un error de 0.03, probado en segmentación de imágenes de tumores y de hígados [30].

- Linknet: Esta red neuronal es capaz de aprender sin ningún gran aumento de parámetros, gracias a esto brinda de un gran rendimiento y eficacia en sus ejecuciones. Se ha probado con el conjunto de imágenes CamVid, ejemplo que se ha tomado para realizar este proyecto, y los resultados han sido muy buenos [31].

- FPN: Es un modelo más enfocado a la detección de múltiples objetos dentro de una misma imagen, dada a la relación que tiene con este proyecto también se estudia su posible uso ya que tiene muy buenos resultados en conjuntos de datos como COCO. Uso probado en segmentación semántica [32].

- PSPNet: Es una arquitectura que brinda de excelentes resultados en las tareas de predicción a nivel de píxeles. Esta arquitectura ha sido la mejor usada para el conjunto de datos imagenet 2016, además de conseguir otros records en la precisión según el coeficiente IoU con otros conjuntos de datos [33].

- PAN: Pyramid Attention Network, esta arquitectura combina el mecanismo de atención y un mecanismo de piramide espacial, es una arquitectura más antigua probada con buenos resultados en conjuntos de datos de 2012 [34].

- DeepLabV3: Diseñado para manejar el problema de segmentar objetos en múltiples escalas, usa un método de convolución en cascada [35].

- DeepLabV3+: Amplía a su antecesor DeepLabV3 agregándole un modulo decodificador simple para refinar resultados en la segmentación, sobre todo útil en los límites de los objetos [36].

Viendo la cantidad de posibilidades que hay sería interesante *realizar un estudio comparativo entre todas las redes posibles con un mismo conjunto de datos*, esto serviría como idea de futuro y mejora de este proyecto.
Tras lo explicado anteriormente para el desarrollo de este trabajo se ha decidido que las mejores opciones para hacer el estudio son las siguientes arquitecturas: **Unet**, por su fama como arquitectura y eficacia en el ámbito biomédico, **FPN**, por la capacidad de detección de varios objetos (clases) dentro de una misma imagen, también a modo de ver los resultados en un modelo de segmentación, y por último **PSPNet**, por sus resultados con conocidos conjuntos de datos y por su eficacia a la hora de predecir o clasificar cada píxel de una imagen.

3.3. Marco Experimental

A partir de estos datos de entrada era necesario hacer modificaciones y ajustes para conseguir una configuración óptima del modelo.

3.3.1. K-Fold (Validación Cruzada)

En primer lugar se ha aplicado un método estadístico de validación cruzada (*Cross Validation*). Este método estadístico sirve para evaluar y comparar algoritmos de aprendizaje ejecutados con segmentos de datos diferentes.
Partiendo de esas 113 imágenes comentadas anteriormente era necesario seleccionar imágenes para entrenar el algoritmo, para validarlo en la propia ejecución y dejar otras restantes para evaluar el modelo una vez ha terminado su aprendizaje, de modo que estas últimas sean ”nuevas” para él. La problemática aquí ocurre cuando por ejemplo, en el caso de las úlceras, si se dividen los datos aleatoriamente no se puede asegurar que el modelo esté entrenando con imágenes que tengan todos los estadios posibles, por lo que el modelo no aprenderá bien. En resumen, los resultados del modelo dependen en buena medida de las imágenes con las que se ha entrenado.
Para solventar este problema hemos usado el método estadístico de los K-fold, uno de los métodos de validación cruzada comentados anteriormente.
Con este sistema de división de datos (K-fold) separábamos en un primer momento las imágenes de entrenamiento y de test (evaluación), 90 imágenes de entrenamiento y 23 imágenes de test (33 % del total de las imágenes son de test), para que no siempre sean las mismas imágenes las de entrenamiento y test implementamos un K-Fold con K = 5, de modo que para cada K (0, 1, 2, 3, 4 y 5) selecciona unas 23 imágenes de test que en los k anteriores no había cogido, el resto imágenes formarían el conjunto de imágenes de entrenamiento. Por lo que se entrenará un modelo con cada K. En la figura 21 se presenta una explicación visual de este procedimiento.

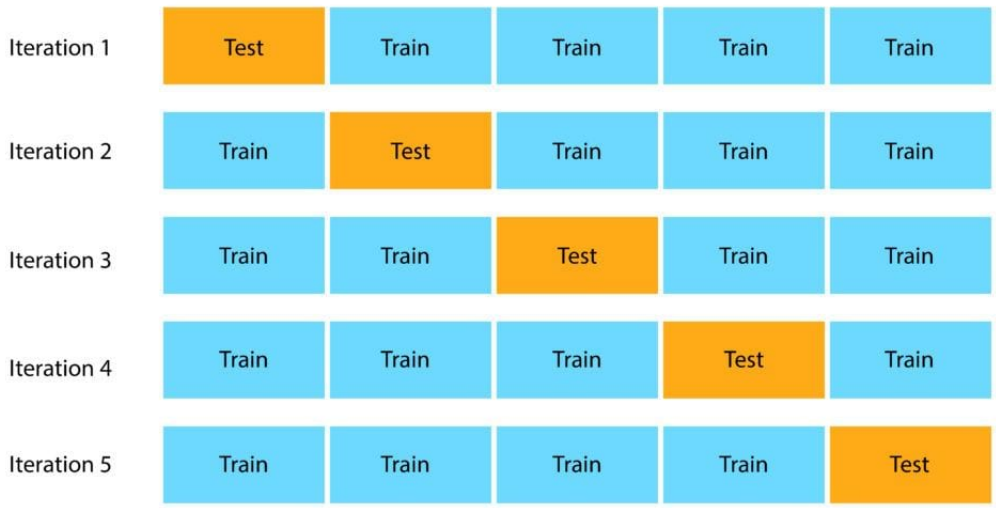

Figura 21: Funcionamiento K-Fold, k = 5.

Después de esta división se hará otra pero exclusivamente en los datos de entrenamiento, que son los que se le van a mandar como argumento al modelo para su aprendizaje. Esta división también será por cada k y será la siguiente, el 33 % de los datos de entrenamiento serán de validación y el resto pertenecerán al conjunto de datos con los que entrenará el modelo para su posterior validación. Por lo tanto haciendo un resumen, por cada entrenamiento tendríamos:

- 60 imágenes (y máscaras) para la fase de entrenamiento.
- 30 imágenes (y máscaras) para la fase de validación.
- 23 imágenes (y máscaras) para la fase de test.

Por lo tanto, por cada ejecución serán entrenamos 5 modelos diferentes (1 por cada K), diferentes porque se han entrenado con conjuntos de datos distintos.

3.3.2. Métricas de entrenamiento

Las métricas por las que el modelo de segmentación ha sido entrenado han sido el coeficiente IoU y el Accuracy.

- **IoU**: El coeficiente de IoU, también conocido como índice de Jaccard, es una medida de similitud entre dos conjuntos de datos del mismo tamaño, cuanto más próximo sea a 1 más similares serán los dos conjuntos de datos en cuestión. Este se calcula de la siguiente forma [37]:

$$\frac{\text{número de similitudes entre ambos conjuntos}}{\text{tamaño de un conjunto}} \tag{1}$$

  Pongamos un ejemplo:

  - Conjunto 1 = [1 ,2 ,3, 5, 9]
  - Conjunto 2 = [3, 4, 5, 6, 7]

  El resultado sería el siguiente:

  - **Número de similitudes entre ambos conjuntos: 2**
  - **Tamaño de conjunto: 5**
  - **IoU = 0.40**

- **Accuracy**: Esta métrica representa el porcentaje total de valores correctamente clasificados por píxel. Esta métrica es más común usarla cuando estamos ejecutando con datos balanceados, es decir, que hay la misma cantidad de valores por cada clase.

Estas métricas tienen umbral, *Threshold*, que permite la clasificación de los píxeles como pertenecientes a una determinada clase, en el caso de que la métrica en cuestión supere este umbral. En función de este umbral el modelo puede tener unos resultados u otros.

3.3.3. Early Stopping

Dada la gran cantidad de tiempo que lleva la ejecución de un modelo de segmentación es necesaria la aplicación de una clase que pare el modelo cuando ya no mejore.
En este proyecto se ha implementado una función de parada temprana cuando el modelo lleve 20-30 épocas (depende de la ejecución) sin mejorar. Con esto evitaríamos tener problemas de sobreentrenamiento o no convergencia de la red.
Esta función se implementa por medio de *callbacks*, de modo que se llame al finalizar cada época para comprobar a través de la métrica del IoU si el modelo ha mejorado o no, en caso negativo se le sumará 1 al contador de esta clase, cuando llegue al límite (paciencia) está hará que el modelo termine de ejecutar y pase al siguiente (si es que lo hay).

La figura 22 muestra una gráfica donde se puede apreciar que la función de parada temprana hace que se detenga la ejecución, antes de que siga entrenando sin mejorar.

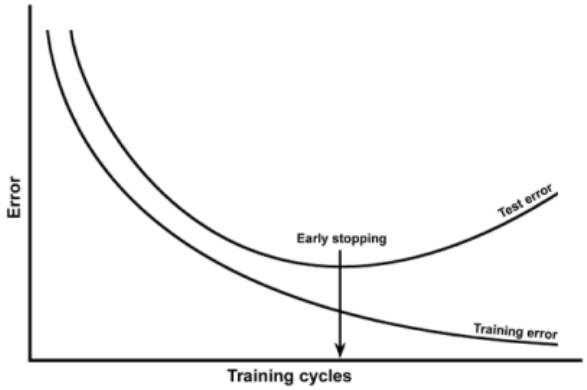

Figura 22: Gráfica de Early Stopping.

3.4. Resultados obtenidos

Esta parte del documento representará el grueso del mismo, donde se recogen los estudios realizados con cada arquitectura y los resultados de las mismas.

Es importante aclarar que las distintas arquitecturas de segmentación han sido preentrenadas con el mismo conjunto de datos, en el caso de que existiera ese pre-entrenamiento, además, todas han sido probadas con dos *encoders* (redes neuronales) diferentes, ya que eran los que mejor funcionaban y los demás eran demasiado grandes.

Al final de esta sección se muestran dos tablas comparativas de los resultados (13 y 14) que se han obtenido para las 12 configuraciones distintas a modo de resumen, con la elección final del modelo para la implementación en la App Web. *se_resnext50_32x4d* y *se_resnext101_32x4d*.

Todas las arquitecturas mencionadas a continuación han sido ejecutadas bajo un marco experimental común, donde las propiedades principales eran:

- Se han entrenado durante un máximo de 60 épocas (iteraciones) del modelo de segmentación.
- Se ha implementado una función de EarlyStopping con una paciencia de 30 épocas, de modo que cuando pasan 30 épocas sin mejorar el modelo tomaba como resultado el último modelo que mejoró las métricas impuestas.
- Se ha ejecutado con la implementación de Kfold con K = 5.
- La activación del los modelos será con la función ”softmax2d”, esta función calculará las probabilidades de cada clase sobre todas las clases posibles.
- Todas las configuraciones se han entrenado en GPU, ”cuda”.

3.4.1. Feature Pyramid Network (FPN)

Esta arquitectura ha sido probada en 4 notebooks distintos con las siguientes combinaciones:

- se_resnext50_32x4d + imagenet como conjunto de datos de preentrenamiento.
- se_resnext50_32x4d sin preentrenamiento.
- se_resnext101_32x4d + imagenet.
- se_resnext101_32x4d sin preentrenamiento.

Es importante apuntar que los entrenamientos de aquellos modelos que no tenían un preentrenamiento previo llevaban más tiempo y por lo general se obtenían peores resultados, ya que el modelo de segmentación debía de partir desde 0.

Para cada combinación se ha aplicado el marco experimental explicado en el apartado 3.3 de este mismo proyecto.

Gracias a la implementación de *K-Folds*, al finalizar la ejecución se comparaban los resultados de los 5 modelos resultantes para una misma configuración del modelo, en esta sección se presenta una media de la evaluación de estos 5 modelos y la matriz de confusión del modelo ”K” con mejor puntuación.

se_resnext50_32x4d + imagenet

El modelo ha sido entrenado con una métrica de iou con un threshold (umbral) de 0.48.

La configuración del modelo es la siguiente:

```
ENCODER = 'se_resnext50_32x4d' # O se_resnext101_32x4d
ENCODER_WEIGHTS = 'imagenet' # O sin preentrenamiento
CLASSES = ['no asignado','piel','periulcera','granulación','
   esfacelos','necrótico']
ACTIVATION = 'softmax2d'
DEVICE = 'cuda'
model = smp.FPN(encoder_name=ENCODER, encoder_weights=
   ENCODER_WEIGHTS, classes=len(CLASSES), activation=ACTIVATION
   )
```

Siendo el argumento "*Classes*" los distintos tejidos que presentan las úlceras.

La evaluación del coeficiente IoU (*Intersection Over Union*) y *Pixel Accuracy* por cada clase se muestran en la tabla 1:

	IoU coefficent	*Pixel Accuracy*
Piel	0.6944±0.0932	0.7732±0.0745
Periúlcera	0.4548±0.1012	0.7925±0.0356
Granulación	0.3535±0.063	0.6222±0.0498
Esfacelos	0.3001±0.0429	0.4025±0.0581
Necrótico	0	0

Cuadro 1: Tabla de resultados

Lamentablemente la clase necrótico no ha sido detectada pese a conseguir una puntuación máxima de coeficiente IoU de 0.7148. Apreciar también los buenos resultados obtenidos para el píxel accuracy, acierto por cada píxel.

Ahora pasamos a hacer una evaluación de esta configuración a través de imágenes, de forma que se aprecie de forma visual (figuras 23 y 24) .

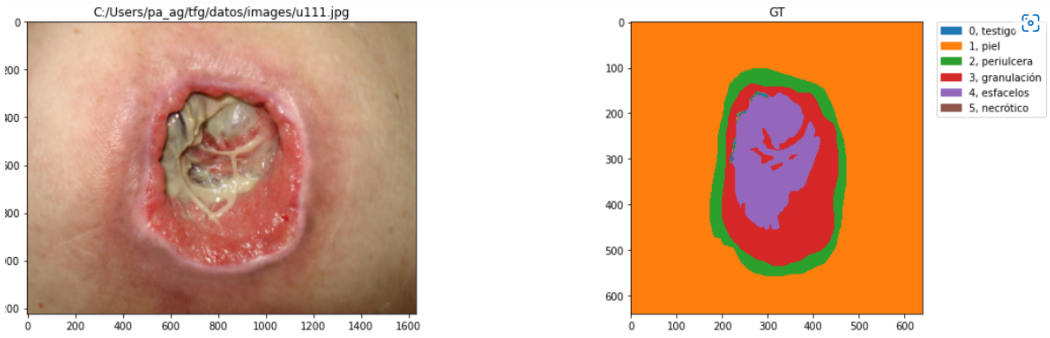

Figura 23: Imagen original - Máscara perfecta.

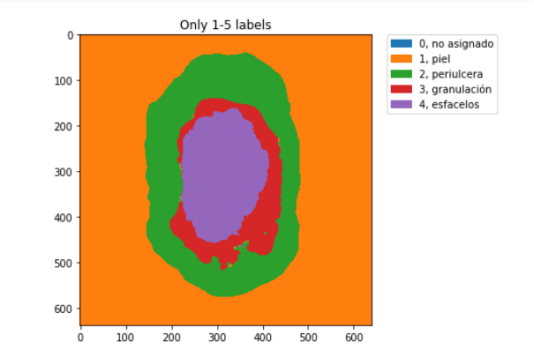

Figura 24: Predicción del modelo.

Se puede apreciar que generaliza en buena medida los esfacelos, aun así dando un esbozo de la zona afectada por estos. También se delimita y aprecia bien los contornos de los tejidos.

Ahora presentamos la matriz de confusión (figura 25) de los resultados. Esta matriz está compuesta por 6 filas y 6 columnas, donde cada columna/fila representa una clase (incluyendo a la clase ”no asignado”). Una matriz de confusión perfecta tendría una diagonal compuesta por 1, esto sería señal de que todos las clases se han detectado a la perfección.

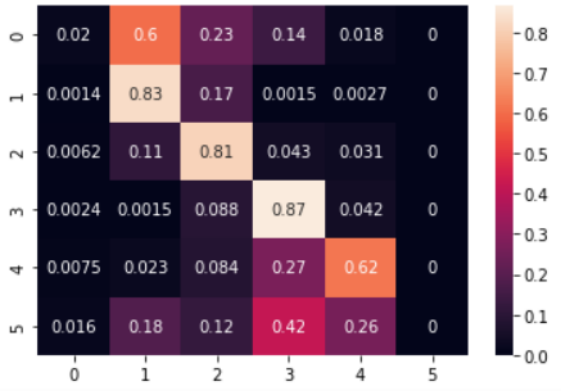

Figura 25: Matriz de confusión.

Vemos que la mayoría de los píxeles de clase ”no asignado” se confunden con la piel, lo cual es buena señal, en cambio, los píxeles de la clase ”necrótico” se confunden con el resto de tejidos presentes.

se_resnext50_32x4d sin preentrenamiento

En esta configuración probamos con el mismo encoder pero sin preentrenar los datos, a priori se entiende que el entrenamiento será más lento y peor, ya que le costará más a la red llegar a entender las imágenes.

Es importante marcar que el valor máximo de coeficiente IoU alcanzado durante todo el entrenamiento, los 5 modelos, ha sido de 0.63, hemos visto un descenso en este coeficiente posiblemente motivado por la dificultad que se le presenta a al red para entender los datos bien sin preentrenamiento.

Como en la anterior arquitectura, en la tabla 2 presentamos los datos de coeficiente de IoU y de Pixel Acurracy.

	IoU coefficent	*Pixel Accuracy*
Piel	0.6812±0.0771	0.7922±0.0416
Periúlcera	0.4325±0.2560	0.7879±0.3164
Granulación	0	0
Esfacelos	0.3257±0.0474	0.6084±0.0781
Necrótico	0	0

Cuadro 2: Tabla de resultados

Este modelo no ha sido capaz de detectar la clase de granulación además de la clase necrótico. Evidentemente no ha obtenido los resultados necesarios para implementarlo en la aplicación. Aún no detectando las clases mencionadas, ha obtenido muy buenos resultados en la clasificación de las clases esfacelos, periúlcera y piel.

Ahora pasamos a una evaluación cualitativa de este modelo (figura 26).

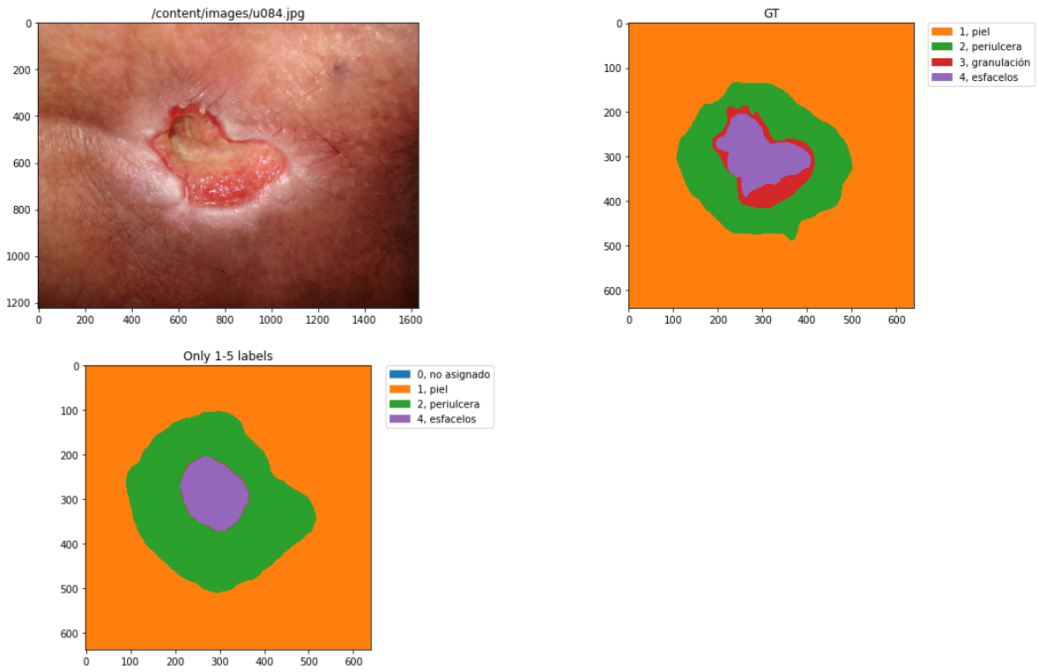

Figura 26: Imagen original - Máscara perfecta (GT) - Predicción.

El modelo como se ha comentado anteriormente no es capaz de detectar algunas clases, además no marca los contornos de los distintos estados de forma efectiva pero si detecta donde se encuentra la mayoría de clase esfacelos, que sería el punto de mayor preocupación del experto clínico.

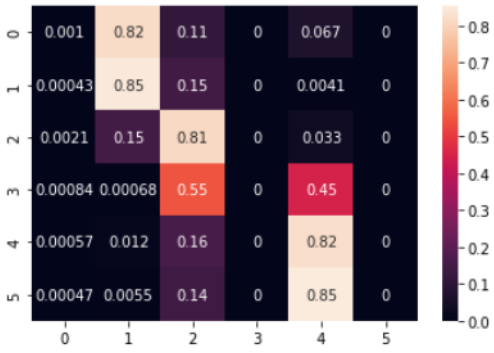

Figura 27: Matriz de confusión.

En su matriz de confusión (figura 27) se puede apreciar la buena puntuación obtenida para las clases piel, periúlcera y esfacelos.

se_resnext101_32x4d + imagenet

En esta prueba cambiamos el encoder de la arquitectura a uno con mayor peso, esta red se espera que de mejores resultados. Obteniendo una puntuación máxima de IoU de 0.7, superando así a los anteriores.

El procedimiento es el mismo que para los anteriores modelos. Resultados en la tabla 3.

	IoU coefficent	*Pixel Accuracy*
Piel	0.7105±0.0524	0.8146±0.0489
Periúlcera	0.4732±0.0456	0.7025±0.0393
Granulación	0.3730±0.0488	0.7122±0.0412
Esfacelos	0.3895±0.0301	0.4925±0.0348
Necrótico	0	0

Cuadro 3: Tabla de resultados

Hasta el momento estos han sido los mejores resultados obtenidos, aun sin detectar la clase necrótico.

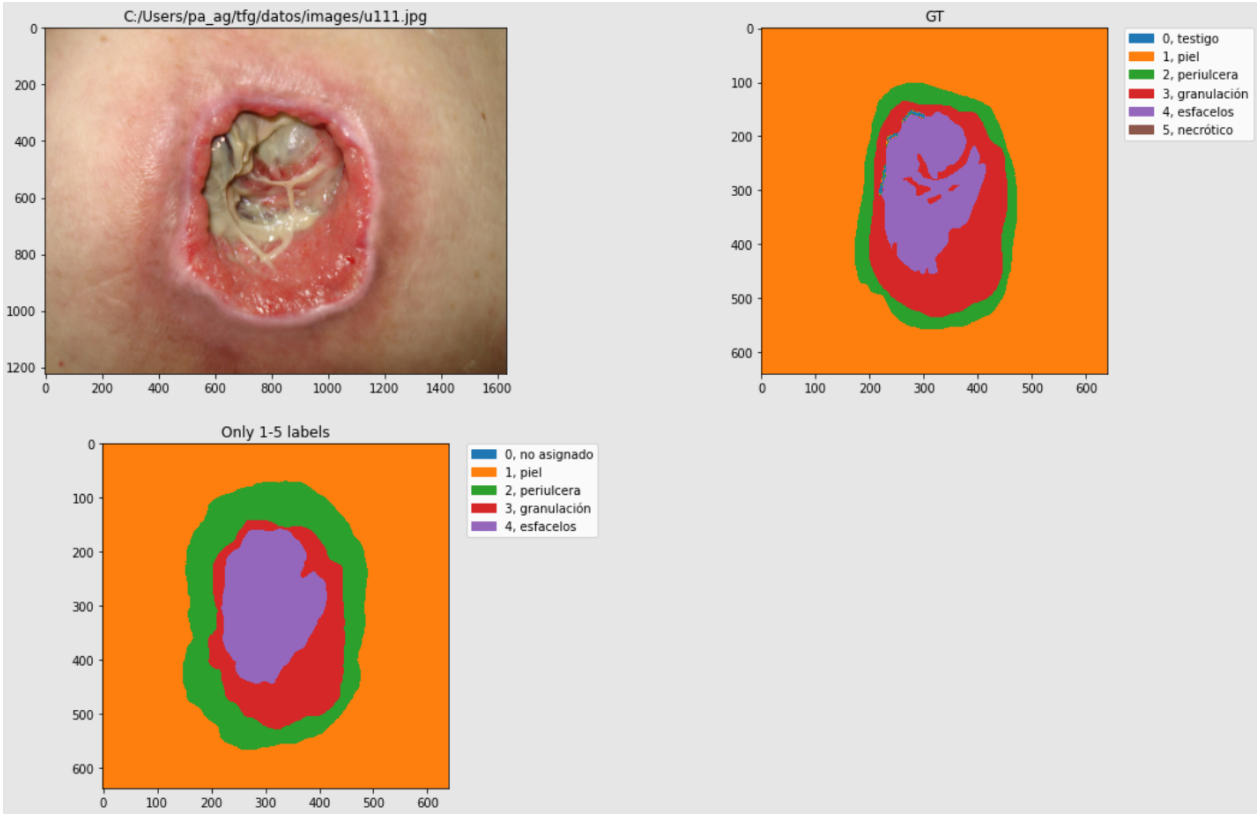

Figura 28: Imagen original - Máscara perfecta (GT) - Predicción.

En la figura 28 apreciamos que el tejido necrótico no se detecta (ya que no sale en la leyenda) pero se delimitan muy bien los bordes de los tejidos y la parte de esfacelos se redondea completamente con un resultado muy similar al GT.

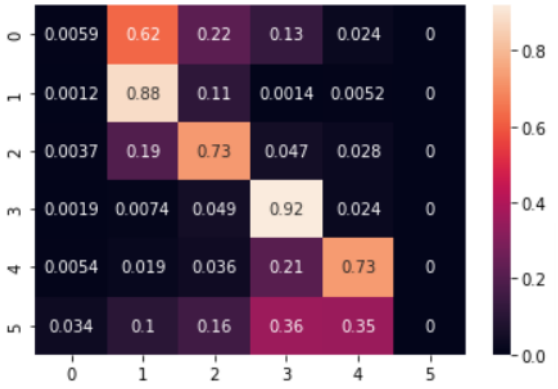

Figura 29: Matriz de confusión.

En su matriz de confusión (figura 29) podemos ver que la mayoría de clase no asignado la asocia a piel y el necrótico se confunde con las clases esfacelos y granulación. Aun así el resultado de la diagonal es próximo a 1, siendo la clase granulación la mejor detectada.

se_resnext101_32x4d + sin preentrenamiento

En el entrenamiento de los modelos con esta configuración se llegó a un valor máximo de IoU de 0.713, superando las puntuaciones de los modelos anteriores, pero como se ha comentado anteriormente, los resultados obtenidos con redes sin un preentrenamiento (imagenet), por lo general, dan peores resultados.

Resultados para el coeficiente IoU y Pixel Accuracy en la tabla 4:

	IoU coefficent	*Pixel Accuracy*
Piel	0.7015 ± 0.0589	0.7845 ± 0.0326
Periúlcera	0.4544 ± 0.0503	0.7395 ± 0.0489
Granulación	0.3308 ± 0.0475	0.4526 ± 0.0805
Esfacelos	0.4059 ± 0.0262	0.5512 ± 0.0157
Necrótico	0	0

Cuadro 4: Tabla de resultados

Como dato a destacar, si comparamos los resultados de esta red con la misma configuración pero con preentrenamiento, es decir, justo la anterior, vemos que no difieren los resultados prácticamente, incluso llegando a clasificar (cuantitativamente) la clase esfacelos mejor.

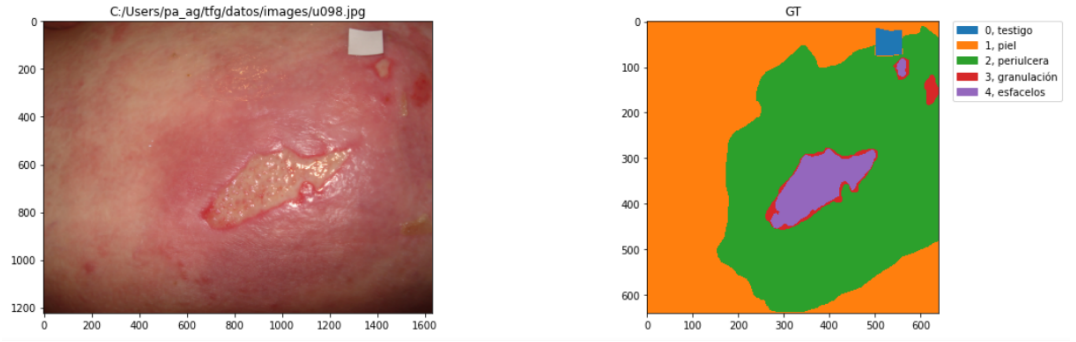

Figura 30: Imagen original - Máscara Perfecta.

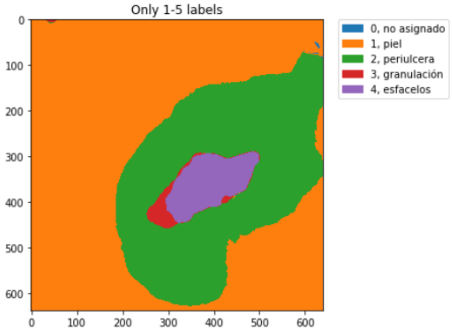

Figura 31: Predicción del modelo.

En las imágenes anteriores (figuras 31 y 30) vemos que se consiguen unos resultados similares al GT no detectando la etiqueta blanca de ”control” y asignándola como piel. Con esto podemos intuir que esta clase ”no asignado” se confunde con la clase ”piel”.

Figura 32: Matriz de confusión.

En la matriz de confusión (figura 32) vemos que se han obtenido unos resultados mínimamente peores que en la anterior red, esto nos hace pensar que la clave para la obtención de resultados exitosos se base en la aplicación de un codificador (encoder) adecuado.

Por lo tanto la mejor configuración obtenida para la red **FPN** ha sido **se_resnext101_32x4d** con preentrenamiento, **imagenet**.

3.4.2. UNET

Ahora pasamos a la arquitectura de modelos, en un principio, más prometedora, ya que es conocida por su buen acierto en la segmentación de imágenes biomédicas, el marco experimental común será el mismo que el usado con la anterior arquitectura (véase sección 3.4.1). Damos paso a su implementación.

```
ENCODER = 'se_resnext50_32x4d' # O se_resnext101_32x4d
ENCODER_WEIGHTS = 'imagenet' # O sin preentrenamiento
CLASSES = ['no asignado','piel','periulcera','granulación','
   esfacelos','necrótico']
ACTIVATION = 'softmax2d'
DEVICE = 'cuda'
model = smp.Unet(encoder_name=ENCODER, encoder_weights=
   ENCODER_WEIGHTS, classes=len(CLASSES), activation=ACTIVATION
   )
```

se_resnext50_32x4d + imagenet

Todas las pruebas se han ejecutado con el mismo marco experimental para concluir con un mejor modelo en igualdad de condiciones, por lo que esta primera prueba determinará como mejor modelo aquel que obtenga el mejor valor de IoU (más próximo a 1), siendo este de 0.71.

Pasamos a ver una media de los resultados en la evaluación tanto del IoU, como del Píxel accuracy en la tabla 5.

	IoU coefficent	*Pixel Accuracy*
Piel	0.7127 ± 0.0425	0.8247 ± 0.0246
Periúlcera	0.5466 ± 0.0215	0.7934 ± 0.0246
Granulación	0.5043 ± 0.0324	0.7036 ± 0.0378
Esfacelos	0.4269 ± 0.0289	0.5256 ± 0.0345
Necrótico	0	0

Cuadro 5: Tabla de resultados

Para esta primera prueba podemos observar que el resultado ha sido mucho más igualitario en cada uno de los 5 modelos ejecutados, ya que la desviación típica entre estos es menor. Aún con esta arquitectura seguimos sin poder obtener datos de necrótico.

Pasamos a ver un análisis más cualitativo de los resultados de esta primera configuración:

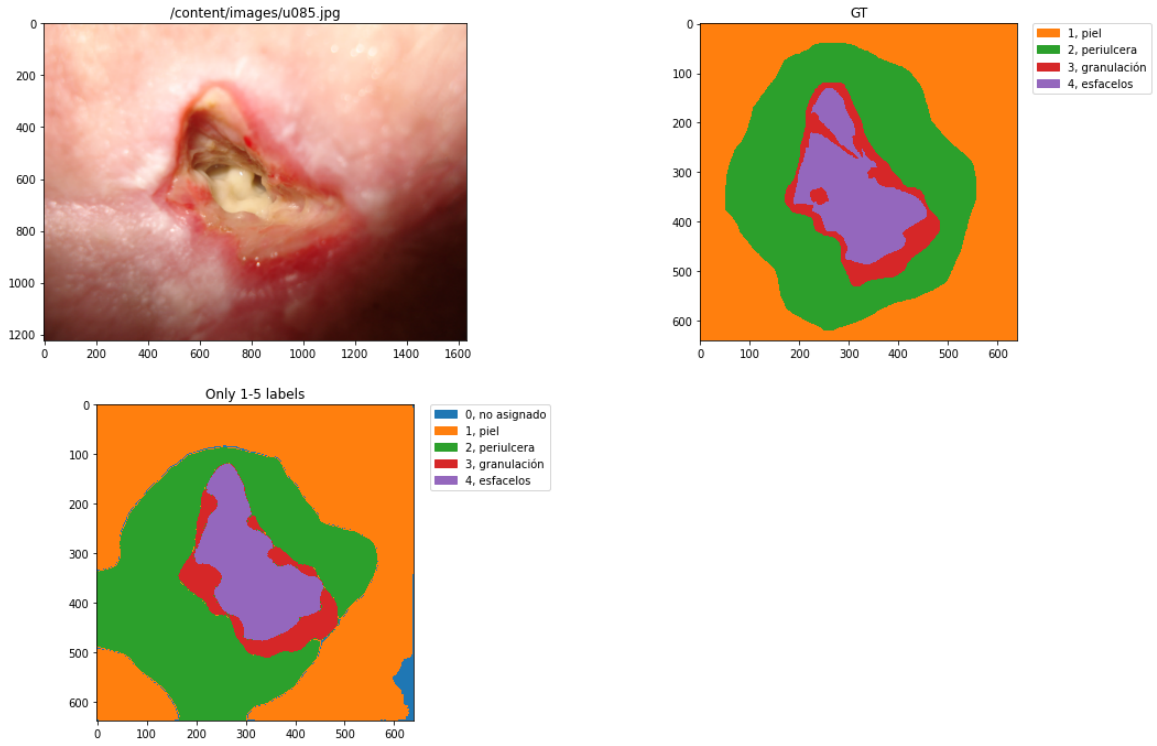

Figura 33: Imagen original - Máscara perfecta (GT) - Predicción.

Podemos ver en la figura 33 que hemos obtenido muy buenos resultados, obteniendo casi a la perfección todos los tejidos que se presentan en la herida.

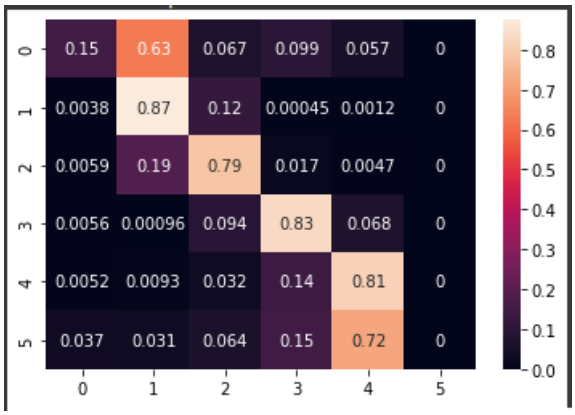

Figura 34: Matriz de confusión.

También se puede apreciar en la figura 34 que muestra la matriz de confusión que se ha obtenido un acierto en la clasificación de cada píxel equitativo para todas las clases menos para la clase necrótico, confundiéndose esta última con su clase antecesora, esfacelos.

se_resnext50_32x4d sin preentrenamiento

Pasamos a la siguiente configuración, de igual forma que para **FPN** se prueba la misma configuración anterior pero sin preentrenamiento. Para nuestra sorpresa se ha conseguido un valor máximo de IoU de 0.756, superando a la prueba anterior con preentrenamiento.

Los resultados obtenidos en la evaluación del modelo han sido los que se muestran en la tabla 6:

	IoU coefficent	*Pixel Accuracy*
Piel	0.8496 ± 0.0289	0.8904 ± 0.0189
Periúlcera	0.5954 ± 0.0211	0.8391 ± 0.0389
Granulación	0.6052 ± 0.0389	0.6948 ± 0.0263
Esfacelos	0.4238 ± 0.0238	0.5148 ± 0.0175
Necrótico	0	0

Cuadro 6: Tabla de resultados

Se ha conseguido un buen resultado en los 5 modelos, ya que la desviación es pequeña, aunque ninguno es capaz de detectar el necrótico, pasamos a una valoración cualitativa del modelo.

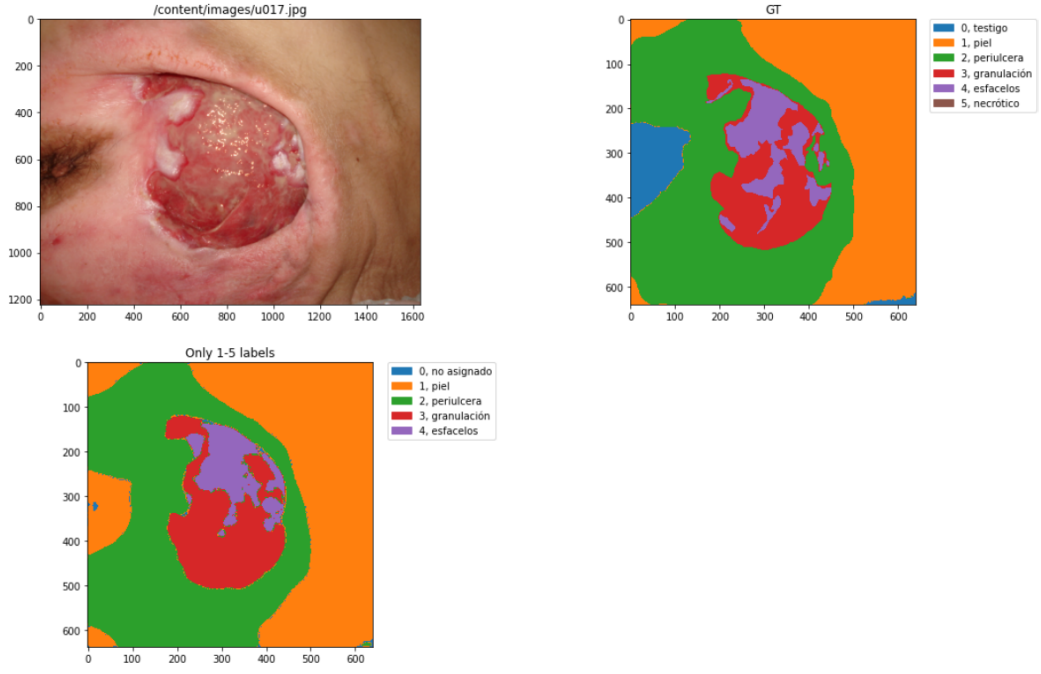

Figura 35: Imagen original - Máscara perfecta (GT) - Predicción.

Como se puede observar en la figura 35 se ha conseguido replicar el GT incluso denotando partes que parecen esfacelos y se no indicaban en el GT.

Figura 36: Matriz de confusión.

La matriz de confusión (figura 36) es muy similar a la anterior, teniendo en su diagonal valores muy próximos a 1. Este sería un buen modelo para la implementación en la aplicación, demostrando que el preentrenamiento puede no ser decisivo en los resultados, dependiendo de la arquitectura.

se_resnext101_32x4d + imagenet

Esta configuración era la más esperanzadora del proyecto ya que juntaba todos los condicionantes para hacer un modelo de segmentación exitoso, una red neuronal apropiada, unet, un encoder de más peso y un preentrenamiento. Esta configuración consiguió un máximo de 0.74 como coeficiente IoU. Los resultados se muestran en la tabla 7.

	IoU coefficent	*Pixel Accuracy*
Piel	0.8336 ± 0.0358	0.8756 ± 0.0378
Periúlcera	0.5875 ± 0.0378	0.8754 ± 0.0278
Granulación	0.6753 ± 0.0389	0.8442 ± 0.0286
Esfacelos	0.5045 ± 0.0388	0.6863 ± 0.0357
Necrótico	0.1189 ± 0.0523	0.1585 ± 0.0614

Cuadro 7: Tabla de resultados

Vemos que es la primera red que ha sido capaz de detectar tejido necrótico, por lo tanto hasta este momento este será el modelo que se implementará en la aplicación. Veamos el resultado.

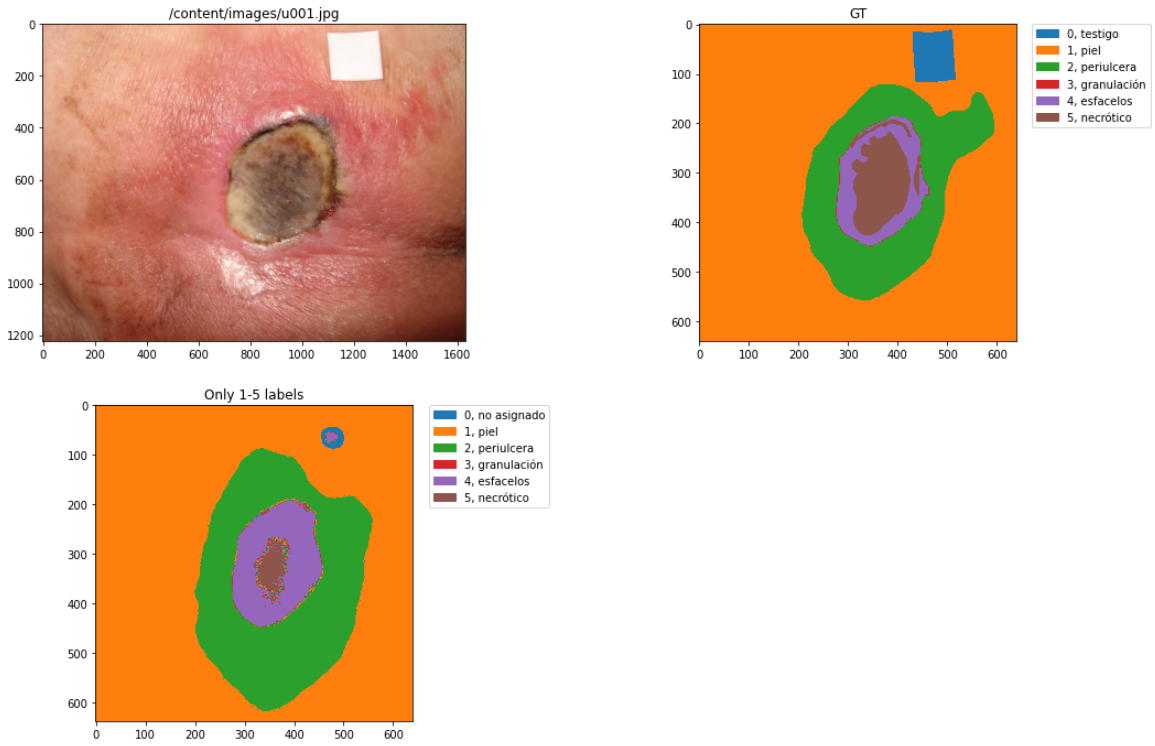

Figura 37: Imagen original - Máscara perfecta (GT) - Predicción.

En la figura 37 se aprecia que existe esa detección de la clase necrótico, es cierto que no de forma perfecta pero esto si le daría un indicio a los médicos para poder diagnosticar en función de la aparición o no de esta clase.

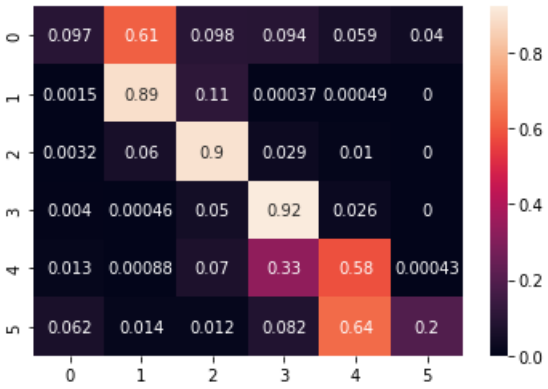

Figura 38: Matriz de confusión.

Vemos que en la matriz de confusión (figura 38) que es cierto que la clase esfacelos no es tan bien detectada como en otras ocasiones, a cambio de esto conseguimos la aparición de necrótico en nuestros resultados.

se_resnext101_32x4d + sin preentrenamiento

Tras los resultados conseguidos en la anterior configuración de esta se esperaba que pudiese mejorar, como había ocurrido con se_resnext50_32x4d sin preentrenamiento. Se ha conseguido alcanzar una puntuación del IoU de 0.746.

Los resultados de esta configuración se muestran en la tabla 8.

	IoU coefficent	*Pixel Accuracy*
Piel	0.7475 ± 0.0572	0.8375 ± 0.0127
Periúlcera	0.5642 ± 0.0458	0.7625 ± 0.0368
Granulación	0.4983 ± 0.0581	0.6114 ± 0.0257
Esfacelos	0.4835 ± 0.0251	0.6053 ± 0.0147
Necrótico	0	0

Cuadro 8: Tabla de resultados

En este caso volvemos a dejar a un lado la clase necrótico, por lo que todo apunta a que la anterior configuración será la implementada. Hagamos un repaso visual de los resultados:

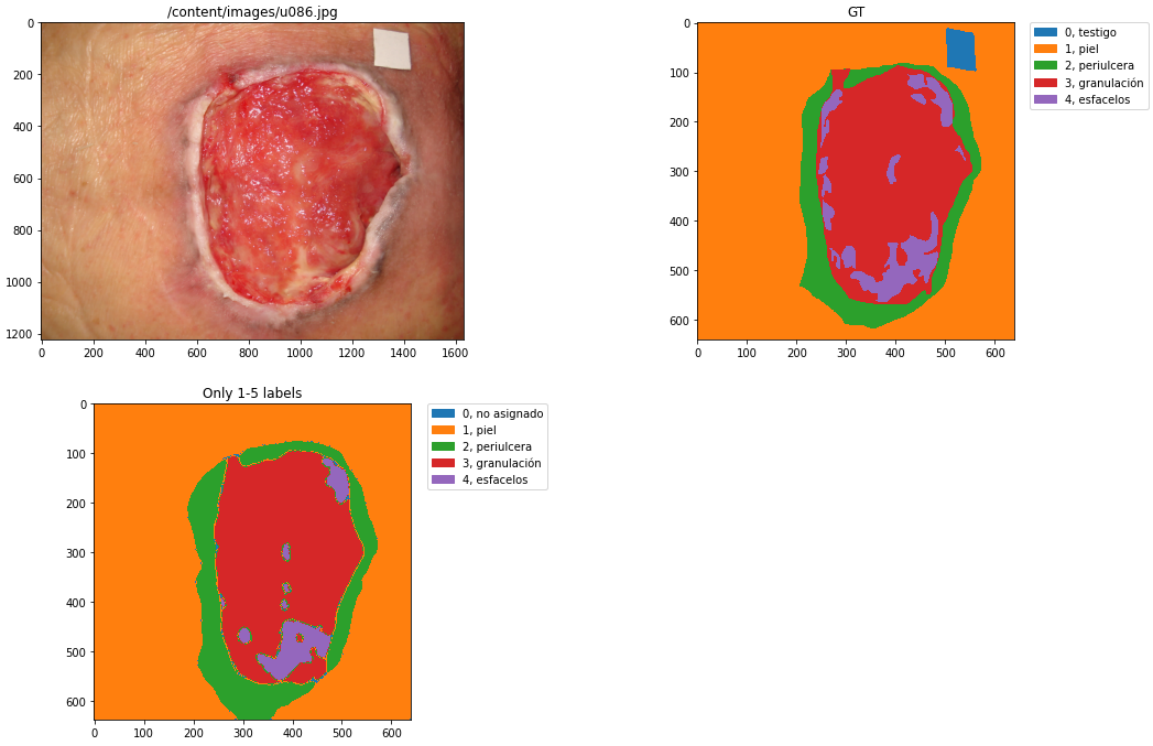

Figura 39: Imagen original - Máscara perfecta (GT) - Predicción.

En la figura 39 se aprecia que para tratarse de una red sin preentrenamiento se han conseguido muy buenos resultados, no mostrando prácticamente la clase no asignado, las zonas con más esfacelos se detectan y los bordes entre las distintas clases están bien diferenciados.

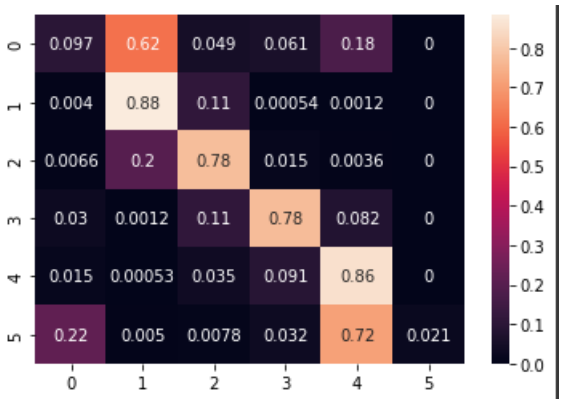

Figura 40: Matriz de confusión.

En la matriz de confusión (figura 40) se aprecia una buena diagonal y se demuestra que la clase no asignado y necrótico se confunden con las demás, en su mayoría con piel y esfacelos respectivamente.

3.4.3. Pyramid scene parsing network (PSPNet)

Por último pasamos a la arquitectura de modelo PSPNet, ciertamente ha sido la que peor resultados ha dado aun siguiendo el mismo marco experimental que las demás. Aún así, hagamos un breve repaso de las pruebas que se han hecho con esta arquitectura y sus resultados.

Su implementación es la siguiente:

```
ENCODER = 'se_resnext50_32x4d' # O se_resnext101_32x4d
ENCODER_WEIGHTS = 'imagenet' # O sin preentrenamiento
CLASSES = ['no asignado','piel','periulcera','granulación','
   esfacelos','necrótico']
ACTIVATION = 'softmax2d'
DEVICE = 'cuda'
model = smp.PSPNet(encoder_name=ENCODER, encoder_weights=
   ENCODER_WEIGHTS, classes=len(CLASSES), activation=ACTIVATION
   )
```

se_resnext50_32x4d + imagenet

Esta primera prueba consiguió un máximo total de 0.6533 como valor de coeficiente de IoU. Algo que se aleja de lo que vimos con la anterior arquitectura, Unet.

En la tabla 9 vemos cómo ha sido la evaluación de esta primera configuración.

	IoU coefficent	*Pixel Accuracy*
Piel	0.7725 ± 0.0388	0.8878 ± 0.0452
Periúlcera	0.3945 ± 0.0327	0.5345 ± 0.0388
Granulación	0.4821 ± 0.0432	0.6457 ± 0.0463
Esfacelos	0.4545 ± 0.0375	0.4058 ± 0.0368
Necrótico	0	0

Cuadro 9: Tabla de resultados

Esta arquitectura tampoco detecta la clase necrótico pero si consigue un buen resultado cuantitativo en el resto de clases.

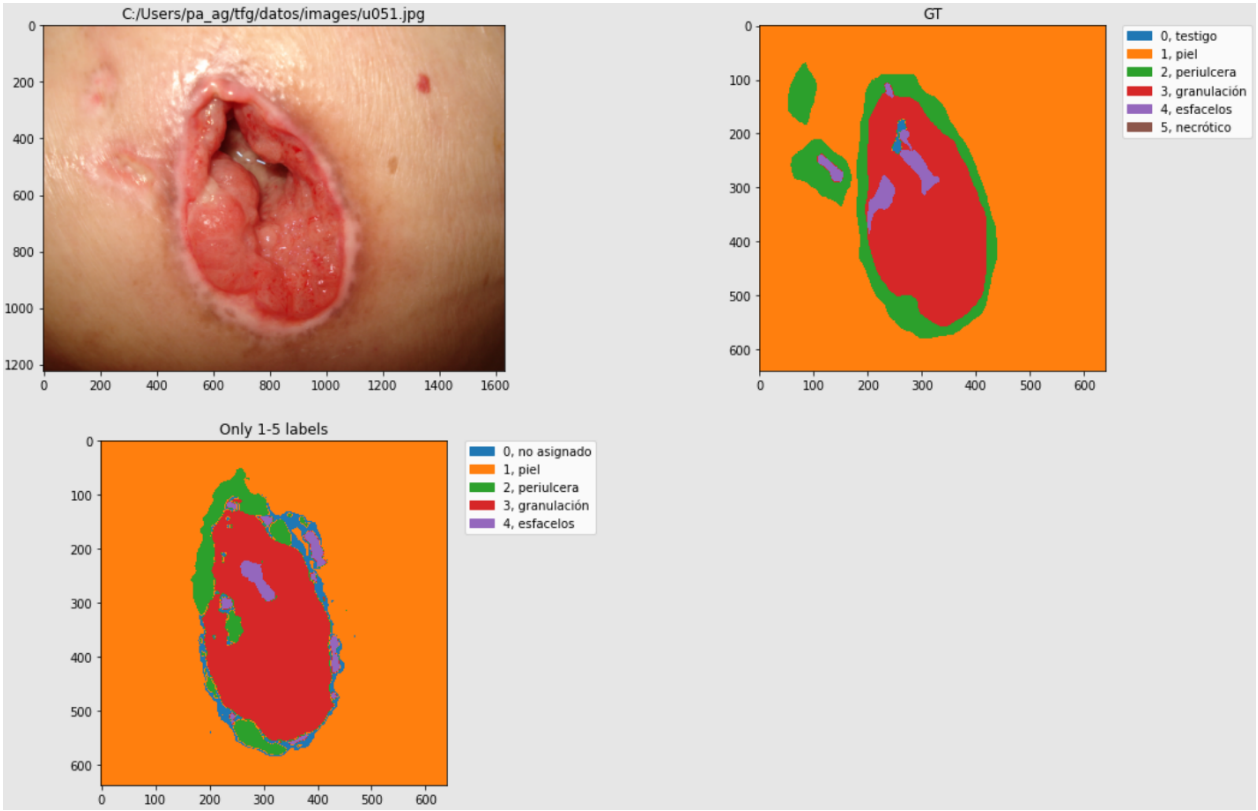

Figura 41: Imagen original - Máscara perfecta (GT) - Predicción.

En la imagen 41 se aprecia una distorsión el la periúlcera incluso mostrando partes de la clase no asignado.

Figura 42: Matriz de confusión.

En la matriz de confusión (figura 42) se demuestra que no detecta bien las clases periúlcera y esfacelos, dejando a un lado el necrótico. Teniendo en cuenta que esta configuración lleva un preentrenamiento (imagenet) se espera que los demás resultados sean similares o peores.

se_resnext50_32x4d sin preentrenamiento

La puntuación máxima alcanzada para el IoU ha sido 0.68. Desglosemos estos resultados (tabla 10):

	IoU coefficent	*Pixel Accuracy*
Piel	0.7882 ± 0.0262	0.8544 ± 0.0256
Periúlcera	0.4533 ± 0.0135	0.6752 ± 0.0345
Granulación	0.5580 ± 0.0345	0.8410 ± 0.0465
Esfacelos	0.5537 ± 0.0258	0.7512 ± 0.0289
Necrótico	0	0

Cuadro 10: Tabla de resultados

No se ha conseguido detectar la clase necrótico, aun así las clases granulación y piel han sido detectadas de manera muy eficaz.

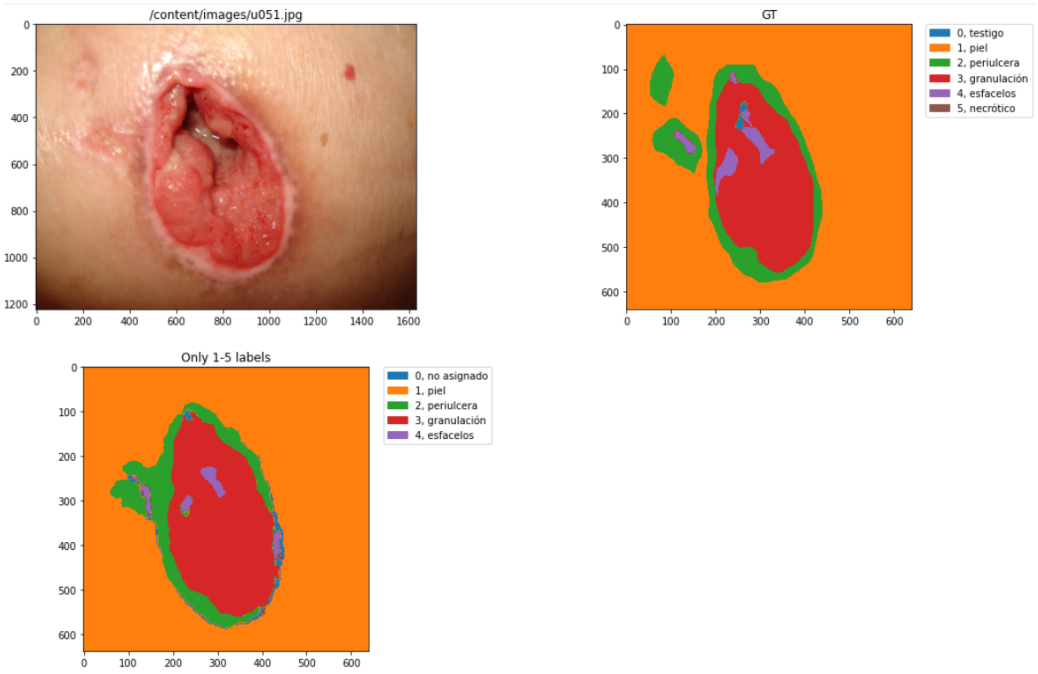

Figura 43: Imagen original - Máscara perfecta (GT) - Predicción.

Como podemos apreciar en la figura 43, aparece la clase no asignado bordeando a la úlcera en si, no detecta esos colores claros de la periúlcera y lo lleva a confusión.

Figura 44: Matriz de confusión.

En la matriz de confusión (figura 44) podemos ver como la clase esfacelos casi en el 50 % de las ocasiones se predice mal y la clase no asignado se reparte equitativamente entre las clases esfacelos, piel y granulación.

se_resnext101_32x4d + imagenet

Tal y como hemos visto hasta ahora, esta configuración ha sido la que mejores resultados nos ha venido dando independientemente de la red escogida, en este caso la puntuación máxima alcanzada ha sido de 0.75, este es uno de los resultados más altos. Veamos como se puede

interpretar en la tabla 11.

	IoU coefficent	*Pixel Accuracy*
Piel	0.7675 ± 0.0310	0.8700 ± 0.0269
Periúlcera	0.5455 ± 0.0328	0.7187 ± 0.0367
Granulación	0.3676 ± 0.0354	0.5275 ± 0.0375
Esfacelos	0.4556 ± 0.0278	0.5035 ± 0.0357
Necrótico	0	0

Cuadro 11: Tabla de resultados

Esta configuración tampoco ha sido capaz de detectar el necrótico, si es cierto que los resultados fueron muy similares independientemente de los datos de entrenamiento, validación o test que tuvieran.

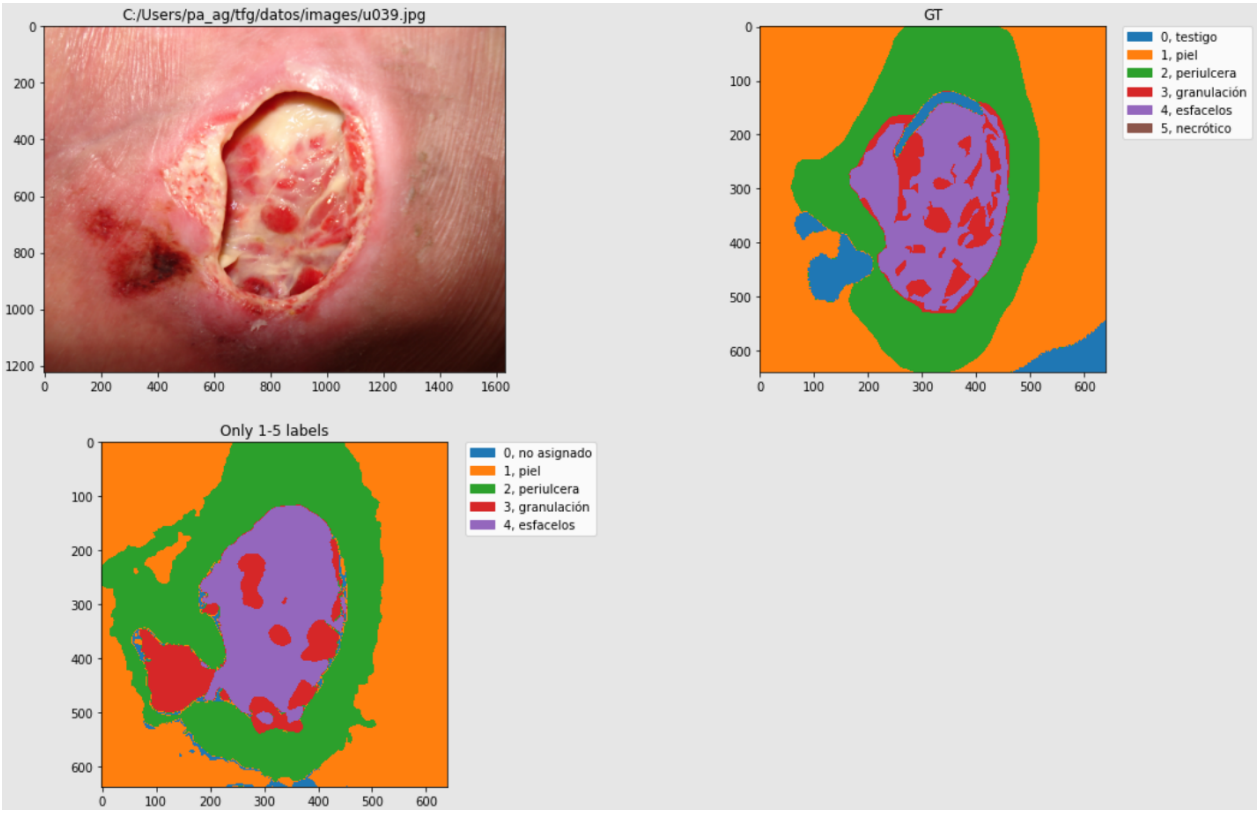

Figura 45: Imagen original - Máscara perfecta (GT) - Predicción.

Vemos en la figura 45 que si generaliza en buena medida los esfacelos. Cabe destacar que la herida que se muestra en la parte izquierda de la imagen, el GT lo cataloga como testigo pero nuestro modelo de segmentación, al tener un tono rojizo lo muestra como estado de granulación.

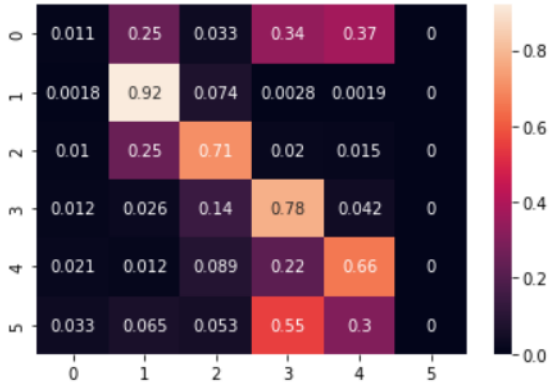

Figura 46: Matriz de confusión.

En la matriz de confusión (figura 46), apreciamos que los tejidos que pero ha podido reconocer el modelo han sido periúlcera y esfacelos, además del necrótico que ha sido inexistente.

se_resnext101_32x4d + sin preentrenamiento

Esta ha sido la última prueba que se ha realizado para este trabajo, donde se ha conseguido un coeficiente de IoU de 0.70.

Los resultados que se han obtenido son los que se presentan en la tabla 12:

	IoU coefficent	*Pixel Accuracy*
Piel	0.7788 ± 0.0487	0.8625 ± 0.0427
Periúlcera	0.5636 ± 0.0366	0.6759 ± 0.0375
Granulación	0.5575 ± 0.0365	0.6025 ± 0.0327
Esfacelos	0.3625 ± 0.0275	0.5875 ± 0.0285
Necrótico	0	0

Cuadro 12: Tabla de resultados

Como las demás configuraciones, no ha sido capaz de aprender a identificar el necrótico.

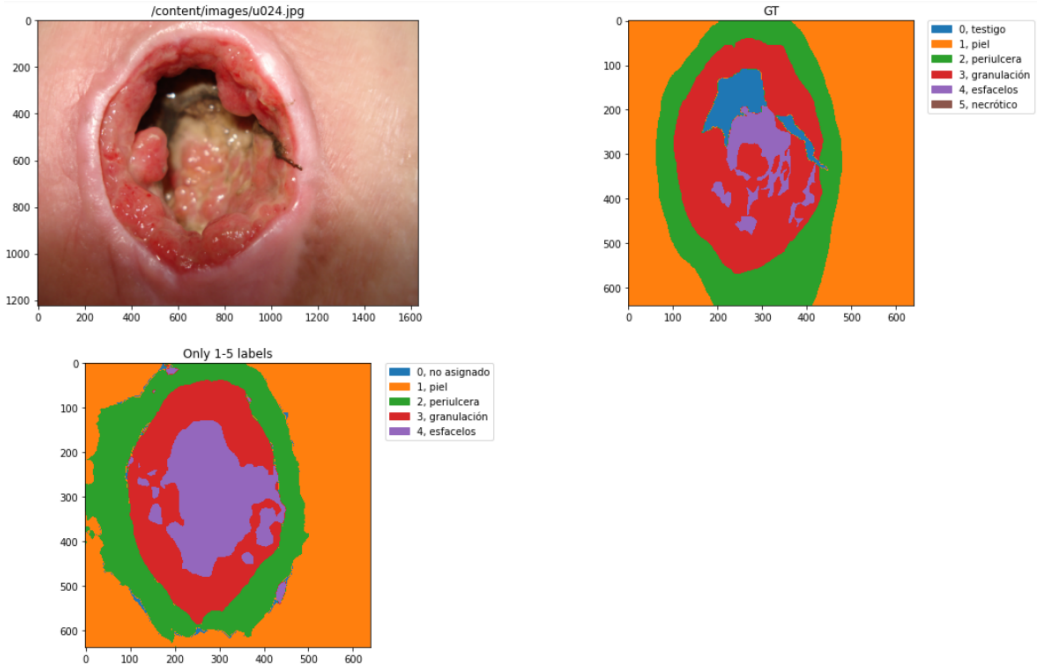

Figura 47: Imagen original - Máscara perfecta (GT) - Predicción.

En la figura 47 se puede apreciar que los resultados han sido buenos a pesar de no detectar necrótico y no tener preentrenamiento. Podríamos concluir que aquellas redes que utilizan como *encoders* se_resnext101_32x4d consiguen mejores resultados. Esta característica parece más decisiva que el hecho de tener un preentrenamiento.

Figura 48: Matriz de confusión.

En la matriz de confusión (figura 48) se aprecian resultados similares a las demás configuraciones exceptuando la confusión de necrótico, esta vez no con la clase esfacelos si no con la granulación.

3.4.4. Resumen de resultados

En esta sección se presentan dos tablas (13 y 14) donde se resumen los resultados de los modelos que se han entrenado, la primera para los valores de IoU, y la segunda para PixelAccuracy. Señalando también que la arquitectura escogida para la implementación en la aplicación ha sido la compuesta por Unet, se_resnext101_32x4d con preentrenamiento Imagenet, ya que ha sido el único modelo capaz de identificar el tejido necrótico.

Arquitectura	**Clases**	**se_resnext50_32x4d**		**se_resnext101_32x4d**	
		Imagenet	*None*	***Imagenet***	*None*
FPN	Piel	0.77±0.07	0.79±0.04	0.81±0.04	0.78 ± 0.03
	Periúlcera	0.77±0.07	0.78±0.31	0.70±0.03	0.73 ± 0.04
	Granulación	0.62±0.04	-	0.71±0.04	0.45 ± 0.08
	Esfacelos	0.40±0.05	0.60±0.07	0.49±0.03	0.55 ± 0.01
	Necrótico	-	-	-	-
Unet	Piel	0.82 ± 0.02	0.89 ± 0.02	**0.87 ± 0.04**	0.83 ± 0.01
	Periúlcera	0.79 ± 0.02	0.84 ± 0.03	**0.87 ± 0.02**	0.76 ± 003
	Granulación	0.70 ± 0.03	0.69 ± 0.02	**0.84 ± 0.02**	0.61 ± 0.02
	Esfacelos	0.52 ± 0.03	0.51 ± 0.01	**0.68 ± 0.03**	0.60 ± 0.01
	Necrótico	-	-	**0.15 ± 0.06**	-
PSPNet	Piel	0.88 ± 0.04	0.85 ± 0.02	0.87 ± 0.02	0.86 ± 0.04
	Periúlcera	0.53 ± 0.03	0.67 ± 0.03	0.71 ± 0.03	0.67 ± 0.03
	Granulación	0.64 ± 0.04	0.84 ± 0.04	0.52 ± 0.03	0.60 ± 0.03
	Esfacelos	0.40 ± 0.03	0.75 ± 0.02	0.50 ± 0.03	0.58 ± 0.02
	Necrótico	-	-	-	-

Cuadro 13: Tabla comparativa de los resultados de *Pixel Accuracy* por clases

Arquitectura	**Clases**	**se_resnext50_32x4d**		**se_resnext101_32x4d**	
		Imagenet	*None*	***Imagenet***	*None*
FPN	Piel	0.69±0.09	0.68±0.07	0.71±0.05	0.70 ± 0.05
	Periúlcera	0.45±0.10	0.43±0.25	0.47±0.04	0.45 ± 0.05
	Granulación	0.35±0.06	-	0.37±0.04	0.33 ± 0.04
	Esfacelos	0.30±0.04	0.32±0.04	0.38±0.03	0.40 ± 0.02
	Necrótico	-	-	-	-
Unet	Piel	0.71 ± 0.04	0.84 ± 0.02	**0.83 ± 0.03**	0.74 ± 0.05
	Periúlcera	0.54 ± 0.02	0.59 ± 0.02	**0.58 ± 0.03**	0.56 ± 0.04
	Granulación	0.50 ± 0.03	0.60 ± 0.03	**0.67 ± 0.03**	0.49 ± 0.05
	Esfacelos	0.42 ± 0.02	0.42 ± 0.02	**0.50 ± 0.03**	0.48 ± 0.02
	Necrótico	-	-	**0.11 ± 0.0523**	-
PSPNet	Piel	0.77 ± 0.03	0.78 ± 0.02	0.76 ± 0.03	0.77 ± 0.04
	Periúlcera	0.39 ± 0.03	0.45 ± 0.01	0.54 ± 0.03	0.56 ± 0.03
	Granulación	0.48 ± 0.04	0.55 ± 0.03	0.36 ± 0.03	0.55 ± 0.03
	Esfacelos	0.45 ± 0.03	0.55 ± 0.02	0.45 ± 0.02	0.36 ± 0.02
	Necrótico	-	-	-	-

Cuadro 14: Tabla comparativa de los resultados de IoU *(Intersection Over Union)* por clases

4

Aplicación Web

En este apartado se detallará como se ha estructurado la aplicación a través de Flask, Css y HTML y se explicará además el funcionamiento de la misma a través de imágenes para su mayor comprensión.
Como introducción es necesario entender que la aplicación sigue una arquitectura Cliente-Servidor, donde el servidor es el que proporciona la mayoría de recursos y funcionalidad de la aplicación web y el cliente es el que en menor parte comprueba la validez de la misma, introduce datos y la testea. El esquema cliente- servidor es el que se muestra en la figura 49.

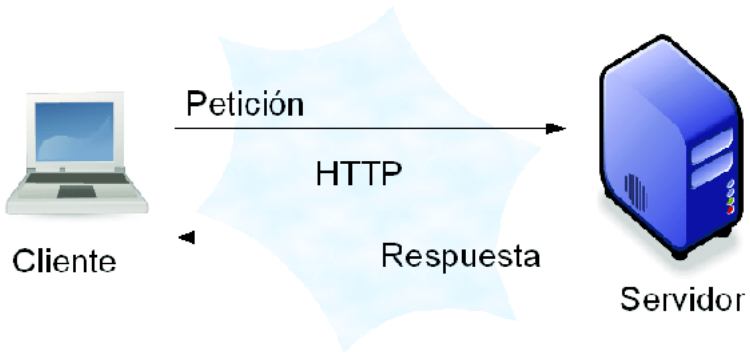

Figura 49: Modelo Cliente-Servidor.

Este tipo de arquitectura es el más usado en el diseño de las aplicaciones web, dada la fácil interacción con el cliente y la conexión con bases de datos. Las aplicaciones web están en continuo desarrollo y crecimiento, en gran parte gracias a la fácil accesibilidad que tienen hoy en día, de modo que se puede conectar y usar cualquier la aplicación web estando conectado a una red de Internet o una Intranet.

4.1. Estructura

La aplicación web está formada por 2 plantillas .html, una hoja de estilos CSS, un *Script* de Python, donde se declaran las funciones, importan librerías y se ejecuta la aplicación y por

último un pequeño código de JavaScript que tiene una función que detallaremos más adelante.

4.1.1. Primer paso: Subida de imagen

Las plantillas HTML son las que dotan de forma y estructura, por así decirlo, a una web, esta aplicación como se comenta anteriormente está formada por dos plantillas, una plantilla ”index.html” y otra ”upload.html”.
En la primera plantilla el usuario puede seleccionar un archivo de su máquina (imagen en formato jpg, png o bmp) y subirla para su posterior procesamiento. El usuario no tendrá que esperar a que su imagen se segmente para ver qué imagen realmente a escogido porque automáticamente antes de segmentar la imagen seleccionada tendrá una vista previa de la imagen en cuestión, pudiendo seleccionar otra si así lo desea. Es aquí donde aparece el pequeño código de Java, siendo el siguiente (figura 50):

```
</script>

<script type = "text/javascript">
    function showPreview(event) {
        if(event.target.files.length > 0){
            var src = URL.createObjectURL(event.target.files[0]);
            var preview = document.getElementById("file-ip-1-preview");
            preview.src = src;
            preview.style.display = "block";
        }
    }
</script>
```

Figura 50: Código JavaScript para la previsualización de una imagen.

Donde se recoge el recurso del cliente y se muestra en pantalla.
Después el usuario puede seleccionar el botón que ejecuta todo el modelo de segmentación desarrollado, una vez pulse el botón este recurrirá a una acción donde se encuentra la función (upload) que recoge el archivo mandado como POST, es decir, que no se muestra en URL, al servidor, y este lo procesa con el modelo de segmentación. En las figuras 51 y 52 se muestra cómo ha sido diseñado tanto la apariencia del botón como su funcionalidad respectivamente.

```
<form action="/upload" enctype="multipart/form-data" method="post" id="botonesform">

    <label for="file-ip-1">Selecciona Imagen</label>
    <input type="file" src="file-ip-1" alt="imagen" name="archivo" id="file-ip-1" accept="image/*" onchange="showPreview(event)">
    <button class="boton" type="submit" id="segmentar" disabled="" onclick="mostrar()">Segmentar imagen

    </button>
</form>
```

Figura 51: Botón de segmentar en HTML.

```python
@app.route('/upload', methods = ['POST'])
def upload_image():

    if request.method == 'POST':

        f = request.files['archivo']

        filename = secure_filename(f.filename)
```

Figura 52: Recogida de archivo (imagen) en la acción ”upload”.

4.1.2. Segundo paso: Segmentar imagen

Tras pulsar el botón ”Segmentar” y recoger el archivo de la imagen la función ”upload” comienza la preparación de la imagen para pasarla al modelo de segmentación y que este muestre el resultado. Muestro el código para su explicación (figura 53):

```python
    if f and allowed_file((filename)):

        path = app.config['UPLOAD_FOLDER']+'/'+ filename
        f.save(path)

        imagen_ini = ["C:/Xampp/htdocs/Aplicacion/static/images/"+filename]


        imagen_dataset = Dataset(
            imagen_ini,
            augmentation=get_validation_augmentation(),
            preprocessing=get_preprocessing(preprocessing_fn),
            classes=CLASSES,
            )
        print(type(imagen_dataset[0]))
        imagen_dataset[0]
        x_tensor = torch.from_numpy(imagen_dataset[0]).to('cpu').unsqueeze(0)
        pr_mask = model.predict(x_tensor)
        pr_mask = (pr_mask.squeeze().cpu().numpy().round())

        mask_def = unificar(pr_mask)

        mask_predict = mask_def.astype(int)

        path_seg = app.config['SEG_FOLDER']+'/'+ filename
        visualizar(mask_predict, path_seg)

        return render_template('subida.html', img = filename)
    return render_template('index.html')
```

Figura 53: Código para mostrar la imagen segmentada.

El funcionamiento es el siguiente:

1. Se comprueba que el fichero tiene una extensión permitida: .jpg, png, bmp, JPG, PNG.

2. Se guarda el archivo en un directorio que recogerá las imágenes previas a la segmentación.

3. Comienza la transformación de la imagen para su segmentación, este proceso es igual que para las imágenes de testeo que se usaban para evaluar los modelos en la sección Metodología, Resultados.

4. Una vez que se segmenta, se aplica una función para visualizar la segmentación por colores con su correspondiente leyenda para la correcta compresión del usuario (Se muestra un ejemplo de la imagen devuelta en la figura 54).

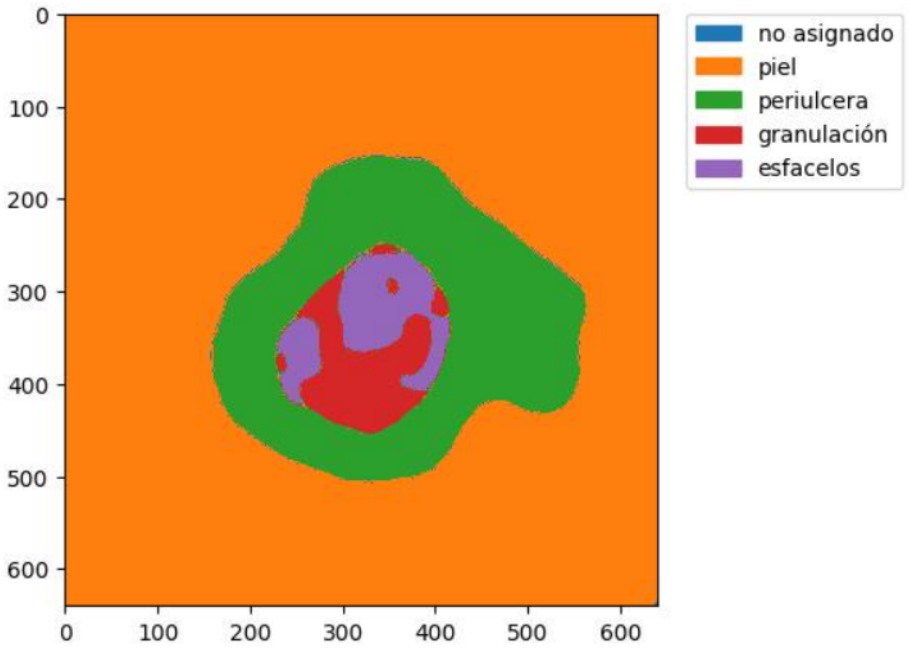

Figura 54: Ejemplo de imagen visualizada.

Para evitar problemas de mandar dos veces la misma imagen se ha implementado en el código una que si se sube una imagen con el nombre de otra que ya se ha subido anteriormente, esta se sobre escribirá quedando la imagen más reciente guardada.

4.1.3. Tercer paso: Volver a segmentar

Para habilitar esta función se ha creado un botón con el texto ”Segmentar otra imagen” enlazado a una acción, de forma que el usuario al seleccionar este botón activa la función ”reset”, esta función lo envía a la pantalla de inicio (figura 55).

4.2. Funcionamiento

Tras la explicación del código pasamos a una explicación más ”visual” de la aplicación para ver lo que puede hacer el usuario y toda la funcionalidad de la aplicación, siguiendo los mismos pasos comentados en la sección anterior.

4.2.1. Primer paso: Subida de imagen

: La figura 55 muestra el inicio de la aplicación, donde el usuario puede seleccionar una imagen para su segmentación. En esta primera pantalla el usuario puede únicamente seleccionar la imagen, ya que le aparecerá un botón deshabilitado para segmentar, que solo se activará cuando este seleccione una imagen.

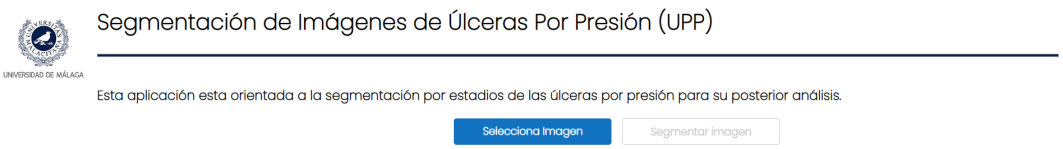

Figura 55: Pantalla de inicio de la aplicación.

Como se puede apreciar en la figura 55, el usuario ve dos botones, uno para seleccionar la imagen, y otro **deshabilitado** para segmentar la imagen. Cuando el usuario pinche en la opción "Seleccionar imagen" se le abrirá una ventana para navegar por su equipo y seleccionar un archivo (figura 56).

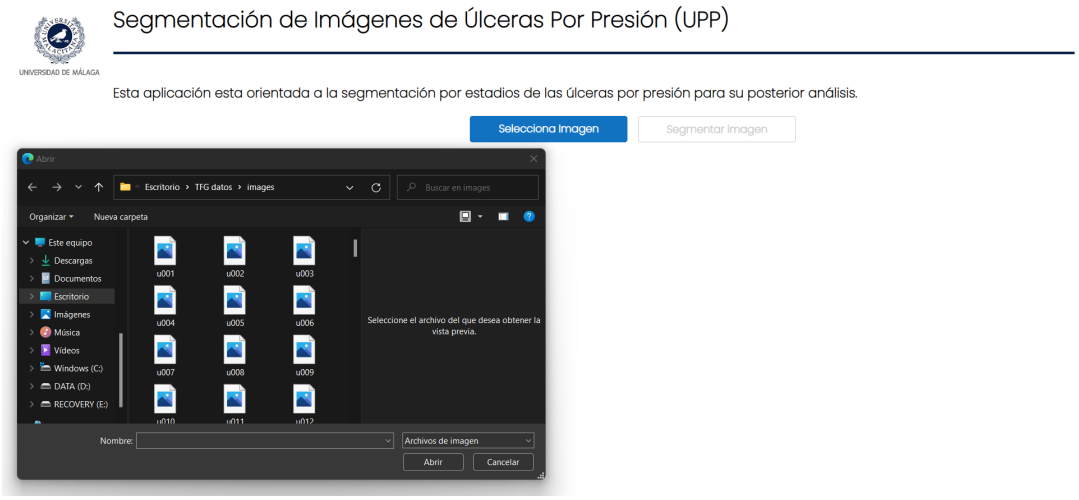

Figura 56: Selección de archivo.

Una vez que señale el archivo y seleccione la opción ”Abrir” se mostrará la imagen por pantalla (figura 57).

Figura 57: Previsualización de imagen.

Como se ha comentado anteriormente, el usuario que está utilizando la aplicación siempre podrá seleccionar otra imagen si así lo desea, de modo que podrá seleccionar otra imagen y visualizarla (figuras 58 y 59) .

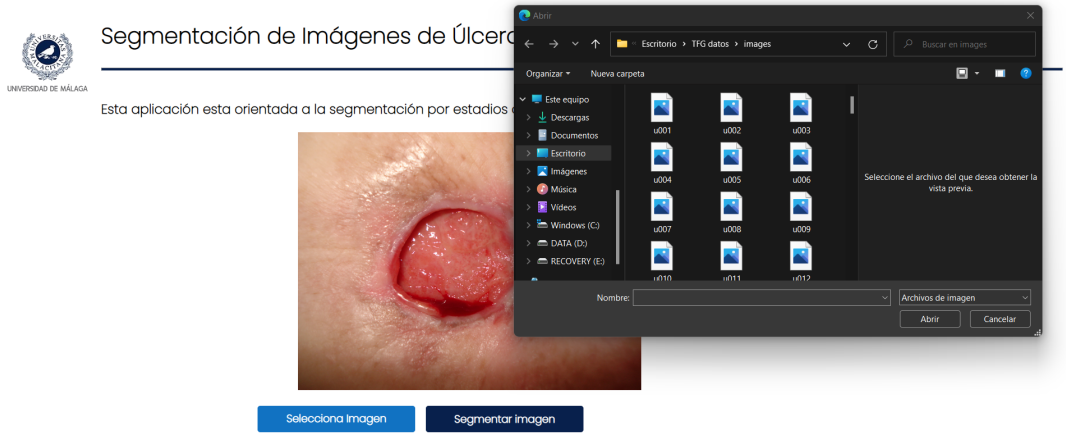

Figura 58: Selección de otra imagen.

Figura 59: Cambio de imagen.

4.2.2. Segundo paso: Segmentar imagen

En este paso el usuario una vez que elija la imagen que desee segmentar, se habilita el botón ”Segmentar Imagen” que el usuario puede accionar y empezará la carga del modelo de segmentación y el procesamiento de la imagen solicitado. Para hacer más ”ameno” la espera del cliente mientras se desarrolla todo el proceso se le muestra por pantalla un icono de ”loader” (figura 60), así el usuario sabrá que no se ha quedado colgada la página y por ende su imagen estará procesándose.

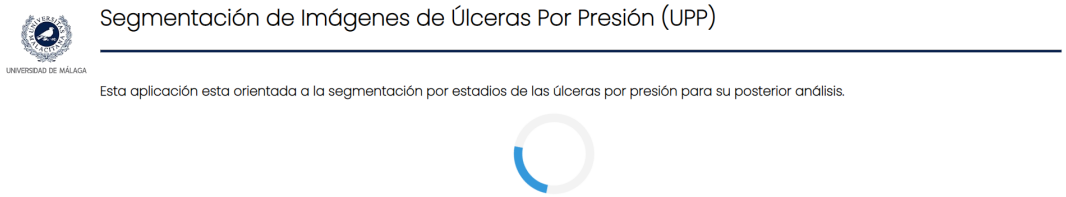

Figura 60: Procesamiento de la imagen.

Una vez terminado el proceso se mostrarán por pantalla las dos imágenes, la primera la imagen original y al lado la imagen segmentada con su correspondiente leyenda, como se

muestra en la figura 61.

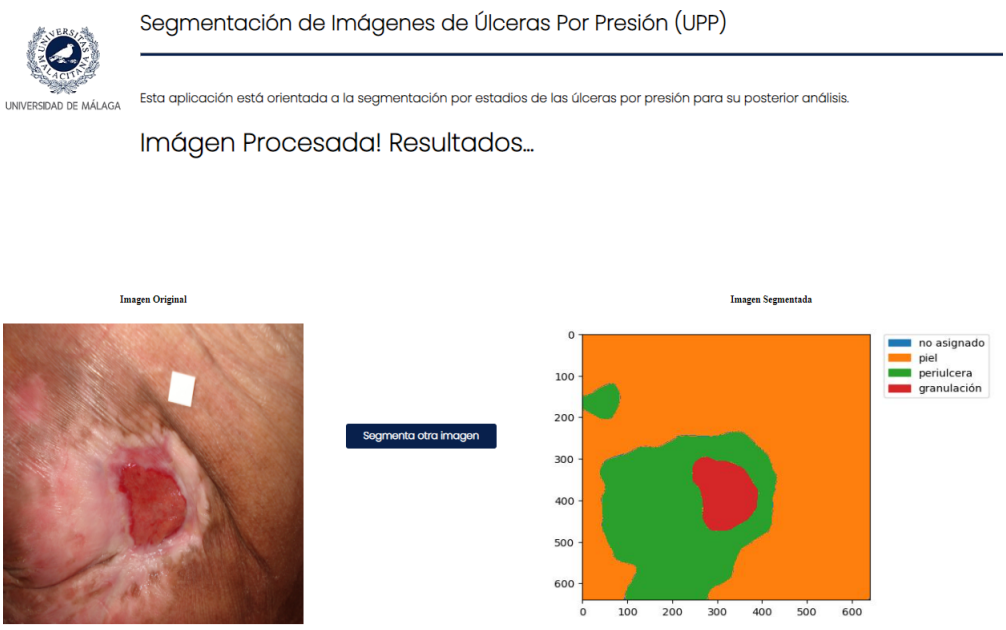

Figura 61: Resultados.

4.2.3. Tercer paso: Volver a segmentar

El cliente podrá seleccionar el botón ”Segmentar otra imagen” para volver de nuevo a la pantalla de inicio (figura 55).

5

Conclusions and Futures Lines of Research

5.1. Conclusions

The aim of this work was to create a useful and effective tool for the staging of pressure ulcers. Given that the design of a software tool such as this would speed up the diagnostic process to a large extent, as the clinical expert would have the tool at his disposal so that through an image (a photo) he could obtain the complete segmentation of the wound.

The first conclusion I can draw from this work is that I have learnt a lot in this field, as the Biomedicine degree does not go as far into the field of computer science.

As for the complexity of the development of the project, the most laborious task has been the development of a first complete code booklet, as at the beginning there were library incompatibilities and problems when visualising the data. Once this problem was understood and solved, we were able to start optimising the training model based on early stopping functions and adjusting the evaluation and training metrics. Once we developed that first notebook the task was based on comparing results and varying certain values to see the behaviour/response of the model.

As for the results of the segmentation model, we can say that they have been very satisfactory, even more than expected, since we started with 113 images and I thought at first that algorithms based on deep learning (deep learning) needed huge amounts of data. The best configuration was the Unet architecture, sext10132x4d and Imagenet as pre-training. This configuration was the only one with which necrotic detection was possible, and for this reason, despite obtaining better results for "sphacels" in other configurations, this option was chosen for implementation in the AppWeb.

The aforementioned configuration has achieved very good overall results, achieving an average IoU (*Intersection Over Union*) coefficient among all classes of 0.54 ± 0.03 and an average overall *Pixel Accuracy* of 0.70 ± 0.04.

On the other hand, regarding the development of the web application, I found the implementation of both the python code and the html code very enjoyable, achieving a very nice interface for the client. I think that this application has much more potential and a function could be implemented that, for example, returns a percentage by stages of the segmented image, so that if a percentage, for example of necrotic, is so small that in the image that cannot be seen with the naked eye, it can be seen in numerical form, with a histogram, with a bar chart, etc.

In short, I think that both the first part of the code, training and comparison and the second part of the design of a dynamic application have obtained very good results, providing an effective solution to the problem tackled at the beginning of the project.

5.2. Future lines of Research

As future lines of this project, it is proposed to obtain better results in the segmentation models, for which the following conditions must be met:

- Automatic hyperparameter adjustment through "*RandomSearch*" or "*GridSearch*, in this way the result of the segmentation model will be further optimized since we would find the configuration of the metrics that would give the best results.
- Run the code in an environment with higher GPU capacity, either locally or remotely. This will serve to make the executions faster, make a more efficient data increase, increasing the size of the images with which the model trains.
- Increase the data set, especially images that contain a greater amount of necrotic, since, being the minority stage in the vast majority of images or even not appearing at all, the model has not been able to identify it.

Another of the future lines could be the study and comparison of all the segmentation models that PyTorch has, in all its versions (normal and improved), in order to determine which architecture would work more effectively in a field of application such as the segmentation of pressure ulcers.

And finally, the aspect to be improved would be the detection of the control class in a training phase, but the non-display of this class in the test images, since it is a class that does not provide clinical value or help with diagnosis.

6

Conclusiones y Líneas Futuras

6.1. Conclusiones

El objetivo de este trabajo era crear una herramienta útil y eficaz a la hora de segmentar por estadios las úlceras por presión. Dado que el diseño de una herramienta software como esta agilizaría en buena medida el proceso de diagnóstico, ya que el experto clínico tendría a su disposición la herramienta para que a través de una imagen (una foto) consiga la segmentación completa de la herida.

La primera conclusión que puedo sacar después de la elaboración de este trabajo es el aprendizaje que he obtenido en este campo ya que en la mención Biomedicina no se profundiza tanto en el campo de las ciencias de la computación.

En cuanto a la complejidad del desarrollo del proyecto, la tarea más laboriosa ha sido el desarrollo de un primer cuaderno de código completo, ya que al principio había incompatibilidad de librerías y problemas al visualizar los datos. Una vez comprendido este problema y solucionado se pudo empezar a optimizar el modelo de entrenamiento a partir de funciones de parada temprana y ajustando las métricas de evaluación y entrenamiento. Una vez que desarrollamos ese primer cuaderno la tarea se basaba en comparar resultados y variar ciertos valores para ver el comportamiento/respuesta del modelo.

En cuanto a los resultados del modelo de segmentación podemos decir que han sido muy satisfactorios, incluso más de lo esperado, ya que partíamos de 113 imágenes y pensaba en un primer momento que los algoritmos basados en aprendizaje profundo (*Deep Learning*) necesitaban enormes cantidades de datos. Siendo la mejor configuración la formada por la arquitectura Unet, se_resnext101_32x4d e Imagenet como preentrenamiento. Esta configuración ha sido la única con la que ha sido posible la detección de necrótico, por este motivo, pese a obtener mejores resultados para " esfacelos " en otras configuraciones, se ha optado por esta opción para su implementación en la AppWeb.

La configuración comentada anteriormente ha conseguido unos resultados generales muy buenos, consiguiendo una media de coeficiente IoU (*Intersection Over Union*) entre todas las

clases de 0.54 ± 0.03 y una media de *Pixel Accuracy* general de 0.70 ± 0.04.

Por otro lado en cuanto el desarrollo de la aplicación Web, me ha resultado muy amena la implementación tanto del código python como la del código html, consiguiendo una interfaz muy agradable para el cliente. Creo que esta aplicación tiene mucho más potencial y se podría implementar alguna función que por ejemplo, devuelva un porcentaje por estadios de la imagen segmentada, de modo que si un porcentaje, por ejemplo de necrótico, es tan pequeño que en la imagen que no se puede apreciar a simple vista, este se aprecie en forma numérica, con un histograma, con un diagrama de barras, etc.

En definitiva, tanto la primera parte de código, entrenamiento y comparación como la segunda parte de diseño de una aplicación dinámica pienso que han obtenido resultados muy buenos, aportando una eficaz solución al problema abordado al comienzo del proyecto.

6.2. Líneas Futuras

Como lineas futuras de este proyecto se plantea la obtención de mejores resultados en los modelos de segmentación, para ello se deberán cumplir las siguientes condiciones:

- Ajuste de hiperparámetros automático mediante "*RandomSearch*" o "*GridSearch*, de esta forma se optimizará aún más el resultado del modelo de segmentación ya que encontraríamos la configuración de las métricas que dieran los mejores resultados.

- Ejecutar en un entorno con mayor capacidad de GPU, ya sea de forma local o remota. Esto servirá para hacer las ejecuciones más rápidas, hacer un aumento de datos más eficaz, aumentando el tamaño de las imágenes con las que entrena el modelo.

- Aumentar el conjunto de datos, especialmente imágenes que contengan más cantidad de necrótico, ya que, al ser el estadio minoritario en la gran mayoría de imágenes o incluso ni aparecer, el modelo no ha podido conseguir identificarlo.

Otra de las lineas futuras podría ser el estudio y comparación de todos los modelos de segmentación que tiene PyTorch, en todas sus versiones (normales y mejoradas), para así determinar que arquitectura trabajaría de forma más eficaz en un ámbito de aplicación como el de la segmentación de úlceras por presión.
Y por último aspecto a mejorar sería detección de la clase testigo en una fase de entrenamiento, pero la no visualización de esta clase en las imágenes de test, ya que es una clase que no aporta valor clínico ni ayuda al diagnóstico.

Referencias

[1] R. M. G. Blázquez R. P. y Rubio. "Prevención y tratamiento de las Úlceras por Presión. Revista Clínica de Medicina de Familia". En: *Revista Clínica de Medicina de Familia* 6 (2007), págs. 284-290.

[2] *¿Qué es Python?* URL: https%20:%20//pythones.net/que%20%E2%88%92%20es%20%E2%88%92%20python%20%E2%88%92%20y%20%E2%88%92%20sus%20%E2%88%92%20caracteristicas/.

[3] *La historia de Python. Las versiones de un lenguaje único.* URL: https://www.tokioschool.com/noticias/historia-python/.

[4] *Tipado dinámico.* URL: https://developer.mozilla.org/es/docs/Glossary/Dynamic-typing.

[5] *PyTorch: saber todo sobre el marco de trabajo de Deep Learning de Facebook.* URL: https://datascientest.com/es/pytorch-saber-todo-sobre-el-marco-de-trabajo-de-deep-learning-de-facebook.

[6] *Jupyter Notebook: documentos web para análisis de datos, código en vivo y mucho más.* URL: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/jupyter-notebook/.

[7] *¿Qué es Anaconda?* URL: https://eiposgrados.com/blog-python/que-es-anaconda/.

[8] *¿Qué es Flask?* URL: https://openwebinars.net/blog/que-es-flask/.

[9] *HTML.* URL: https://desarrolloweb.com/articulos/que-es-html.html.

[10] *CSS.* URL: https://blog.hubspot.es/website/que-es-css.

[11] R. M. Veredas. Francisco J. y Luque-Baena. "Wound image evaluation with machine learning. Neurocomputing". En: *Neurocomputing* 164 (2015), págs. 112-112.

[12] L. Morente. "Valoración de Úlceras Por Presión Mediante Tecnologías de La Información Y La Comunicación". En: *Universidad de Málaga* 1 (2012), pág. 1.

[13] C. Vidal P. Sepúlveda. "ÚLCERAS POR PRESIÓN." En: *Ulcers* 14 (2019), pág. 2.

[14] *Inteligencia artificial.* URL: https://static0planetadelibroscom.cdnstatics.com/libros_contenido_extra/40/39308_Inteligencia_artificial.pdf.

[15] *Inteligencia artificial.* URL: https://www.computerweekly.com/es/definicion/Inteligencia-artificial-o-IA.

[16] *Mantenimiento predictivo.* URL: https://www.iberdrola.com/innovacion/mantenimiento-predictivo.

[17] *Machine Learning.* URL: https://www.ibm.com/es-es/cloud/learn/machine-learning.

[18] LeCun Y. Bengio Y. Hinton G. "Deep Learning." En: *Nature* 7553 (2015), págs. 436-444.

[19] *Deep Learning ¿Qué es y por qué es importante?* URL: https://www.sas.com/es_es/insights/analytics/deep-learning.html.

[20] *Redes Neuronales, Universidad Politécnica de Madrid.* URL: https://www.researchgate.net/profile/Pedro-Larranaga/publication/268291232_Tema_8_Redes_Neuronales/links/55b7b5c408ae9289a08c0c68/Tema-8-Redes-Neuronales.pd.

[21] *Redes Neuronales Convolucionales.* URL: https://www.juanbarrios.com/redes-neurales-convolucionales.

[22] *¿Cómo funcionan las Convolutional Neural Networks?* URL: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/.

[23] *Reconocimiento y clasificación de imágenes con Deep Learning.* URL: https://www.campusmvp.es/recursos/post/reconocimiento-y-clasificacion-de-imagenes-con-deep-learning.aspx.

[24] *Modelos de Detección de Objetos.* URL: https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/.

[25] *Clasificación de imágenes, detección de objetos, segmentación semántica, segmentación de instancias y otros conceptos.* URL: https://programmerclick.com/article/2299928720/.

[26] *Segmentación semántica usando PyTorch.* URL: https://programmerclick.com/article/38211777380/.

[27] *SEGMENTACIÓN DE IMÁGENES MEDIANTE OPERACIONES MORFOLÓGICAS EN PYTHON.* URL: https://es.acervolima.com/segmentacion-de-imagenes-mediante-operaciones-morfologicas-en-python/.

[28] Ronneberger O. Fischer P. Brox T. "U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention." En: *Springer* 1 (2015), págs. 234-241.

[29] Zongwei Zhou Md Mahfuzur Rahman Siddiquee Nima Tajbakhsh y Jianming Liang. "UNet++: A Nested U-Net Architecture for Medical Image Segmentation." En: *Springer* 7553 (2015), págs. 436-444.

[30] Fan Tongle y col. “MA-Net: A Multi-Scale Attention Network for Liver and Tumor Segmentation”. En: *IEEE Access* 8 (2020), págs. 179656-179665.

[31] Chaurasia A. Culurciello E. “Linknet: Exploiting encoder representations for efficient semantic segmentation.” En: *IEEE* 7553 (2017), págs. 1-4.

[32] Tsung-Yi Lin y col. “Feature pyramid networks for object detection”. En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, págs. 2117-2125.

[33] Hengshuang Zhao y col. “Pyramid scene parsing network”. En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, págs. 2881-2890.

[34] Hanchao Li y col. “Pyramid attention network for semantic segmentation”. En: *arXiv preprint arXiv:1805.10180* (2018).

[35] Liang-Chieh Chen y col. “Rethinking atrous convolution for semantic image segmentation”. En: *arXiv preprint arXiv:1706.05587* (2017).

[36] Liang-Chieh Chen y col. “Encoder-decoder with atrous separable convolution for semantic image segmentation”. En: *Proceedings of the European conference on computer vision (ECCV)*. 2018, págs. 801-818.

[37] *Segmentación semántica usando PyTorch*. URL: https://statologos.com/jaccard-similaridad/.

Apéndice A
Manual de Instalación

En este apartado se detallan tanto los programas que se necesitan instalar como las librerías necesarias con su correspondiente versión para ejecución de este proyecto desde 0. Este código se puede ejecutar desde cualquier entorno de programación que permita Python como lenguaje, aquí detallaremos los pasos a seguir para ejecutarlo de manera que haya que instalar la menor cantidad de programas y librerías posibles.

Este trabajo se divide en dos partes claramente diferenciadas:

- Modelo de segmentación
- Aplicación Web

Por lo tanto es necesario marcar las instrucciones de instalación para cada una por separado.

A.1. Modelo de segmentación

A.1.1. Google Colaboratory

Google colaboratory (Google Colab) es un producto de Google Research. Esta extensión de google permite a cualquier persona con una cuenta de Google escribir y ejecutar código en Python en el propio navegador, sin necesidad de instalar nada. Este entorno es especialmente utilizado para formación académica, aprendizaje automático, análisis de datos. Además Google Colab trabaja como servicio alojado de Jupyter de modo que en su interfaz se podrá programar en forma de Notebook, de modo que se podrá escribir texto y código (en celdas diferentes) pero dentro del mismo cuaderno.
Gracias a Google Colab no necesitamos instalar librerías en nuestra máquina. Por lo tanto las librerías que tenemos que importar son las que se muestran en la tabla 15:

Librería	Versión
Python	3.7.13
Torch	1.11.0+cu113
Segmentation_models_pytorch	0.2.1
Matplotlib	3.2.2
Cv2	4.1.2
Numpy	1.21.6
Sklearn	1.0.2
Seaborn	0.11.2
Tensorflow	2.8.0
Albumentations	1.2.0

Cuadro 15: Librerías/versiones necesarias.

Otras librerías que se usan ya vienen instaladas en la versión de Python:

- Glob
- Statistics
- Random
- Shutil

La librería de Torch versión 1.11.0+cu113 es la que posibilita que se entrenen los modelos con la GPU, de esta forma las modelos se entrenarán más rápido y se conseguirá más capacidad para hacer aumento de datos si fuera necesario.
Existe un conflicto entre dos de las librerías que se deben importar para la ejecución del código, *albumentations* y *Open CV*. Ya que albumentations instala un paquete de OpenCV que no satisface nuestras necesidades y debe de depender de el.
Por lo tanto, debemos obligar a albumentations a usar la versión de open cv ya instalada. De forma que primero hay que importar la librería de open cv a través del comando:

```
import cv2
```

Y después deberíamos instalar la librería albumentations ya que esta no está instalada de forma predeterminada en Google Colab de la siguiente forma:

```
pip install -U albumentations --no-binary qudida,albumentations

import albumentations
```

De esta forma la librería albumentations dependerá de la versión de open cv (cv2) instalada en el entorno.
Con esto ya estaría el entorno listo para ejecutar el código del modelo de segmentación.

A.2. Aplicación

Para la ejecución del código que conforma la aplicación web se partirá desde la consola del equipo, previamente es necesaria que tengamos instaladas unas versiones de Python y pip recientes, en los siguientes enlaces está la descarga y la forma en como se debe instalar.

- Python: https://www.python.org/downloads/
- Pip: https://pypi.org/project/pip/

A.2.1. Xampp

Para el inicio de la aplicación también es necesaria la instalación de Xampp, un paquete de software libre con el sistema de gestiñon de base de datos MySQL y el servidor web de Apache. Su manual de instalación se puede encontrar en el siguiente enlace:

- Xammp: https://www.apachefriends.org/download.html

Activación SQL y Apache

Una vez que está instalada se debe de abrir el panel de control y activar MySQL y Apache.
Al activar Apache habrá un problema de conexión y debemos de darle al botón "config" en la linea de Apache y a "Apache httpd.conf". La figura 62 muestra los pasos a seguir visualmente.

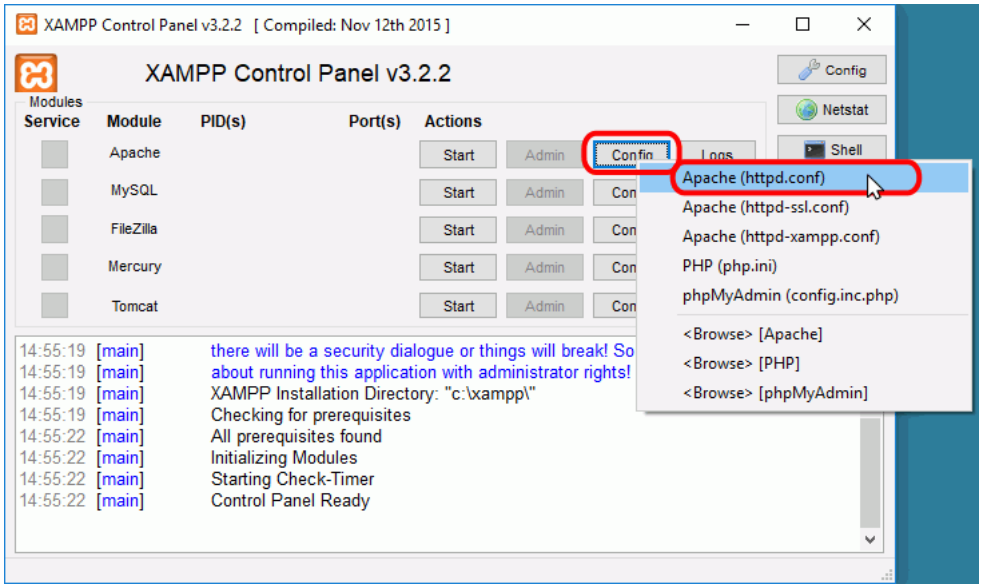

Figura 62: Panel de control Xampp.

En el bloc de notas que se abre tenemos que cambiar lo siguiente:

- *Listen 80 ->Listen 8080*

- *ServerName localhost:80 ->ServerName localhost:8080*

Por último guardaremos los cambios y activaremos de nuevo Apache.

A.2.2. Ejecución

Para la ejecución del código de la aplicación hay que descargarse el zip completo de la entrega del trabajo de fin de grado, y descomprimirlo en un directorio.
El esquema que sigue la aplicación hecha con Flask es el que se muestra en la figura 63:

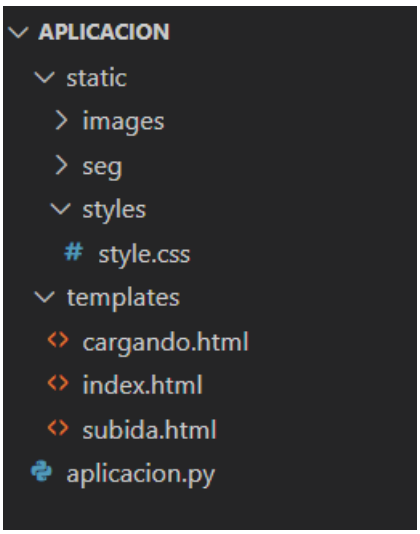

Figura 63: Esquema Flask App.

De modo que en la primera carpeta estará el script de python, en la carpeta ”templates”se guardan los archivos .html, y en la carpeta ”Static/Styles” se guarda el archivo .css.

Librerías necesarias para la ejecución

Las librerías necesarias para la correcta ejecución de la aplicación son las que se muestran en la tabla 16.

Librería	Versión
Python	3.7.13
Torch	1.11.0+cu113
wekzeug	2.1.2
Flask_restful	0.3.8
Flask	2.1.2
Segmentation_models_pytorch	0.2.1
Matplotlib	3.2.2
Cv2	4.1.2
Numpy	1.21.6
Sklearn	1.0.2
Seaborn	0.11.2
Tensorflow	2.8.0
Albumentations	1.2.0

Cuadro 16: Librerías/versiones necesarias.

Para la instalación de todas estas librerías con sus versiones es necesario ejecutar el siguiente comando:

```
cd C:/Xampp/htdocs/Aplicacion (directorio donde está el archivo
    requeriments.txt)
pip install -U requeriments.txt
```

O en su defecto:

```
cd C:/Xampp/htdocs/Aplicacion (directorio donde está el archivo
    requeriments.txt)
py -m install requeriments.txt
```

Ejecución por consola

Para ejecutar la aplicación debemos de abrir una ventana de comando y usar el siguiente código.

```
cd ''Directorio''
```

Siendo ”Directorio” el directorio donde se haya descargado el archivo del proyecto, y específicamente la carpeta **Aplicación**.
Por ejemplo:

```
cd C:/Xampp/htdocs/Aplicacion
```

El siguiente comando será el siguiente:

```
py aplicacion.py
```

Este último comando ejecutará la aplicación y devolverá una dirección http, la cual debemos de copiar y pegar en nuestro navegador.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA