Trabajo Fin de Máster
Máster Universitario en Ingeniería de Telecomunicación

# Aprendizaje profundo para la segmentación de lesiones pigmentadas de la piel

Autora: Rocío Méndez Hernández
Tutoras: Begoña Acha Piñero
Mª del Carmen Serrano Gotarredona

**Dpto. Teoría de la Señal y Comunicaciones**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2019

Trabajo Fin de Máster
Máster Universitario en Ingeniería de Telecomunicación

# Aprendizaje profundo para la segmentación de lesiones pigmentadas de la piel

Autora:
Rocío Méndez Hernández

Tutoras:
Begoña Acha Piñero
Catedrática de Universidad

Mª del Carmen Serrano Gotarredona
Catedrática de Universidad

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Máster: Aprendizaje profundo para la segmentación de lesiones pigmentadas de la piel

Autora: Rocío Méndez Hernández

Tutoras: Begoña Acha Piñero
Mª del Carmen Serrano Gotarredona

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

*A todos los que habéis creído en mí*
*Y a los que no*

# Agradecimientos

Muchas gracias a mis padres, pues a todo lo que me habéis dado le debo el estar aquí y ahora. Gracias por el esfuerzo y el empeño que siempre habéis puesto en la educación de vuestros hijos, y por apoyarme en mis decisiones. A mis hermanos, quienes con frecuencia se burlan de mí porque, según ellos, "me paso el día estudiando", también os doy las gracias, porque los momentos de distracción os los debo a vosotros. Gracias a Alfonso por haber combinado los dos; por pasarte el día estudiando conmigo y por los momentos de descanso.

Quiero agradecer a los docentes de la escuela el papel que han jugado en la formación de esta generación de telecos y de mí en particular, y a mis compañeros de carrera por su ayuda y colaboración en esta hazaña que es estudiar teleco; todos habéis contribuido a que llegue este momento. En especial, gracias a mis compañeros del Máster de Teleco, y muy especialmente a los que os considero mis amigos. A lo largo de estos siete años ha habido etapas más duras que este proyecto, sin embargo nada ha sido tan personal y autónomo como este Trabajo Fin de Máster, y constituye lo que considero un gratificante final a mi paso por la escuela.

Finalmente, gracias a mis tutoras, Begoña y Carmen, por introducirme en el mundo del *deep learning* y haber despertado en mí el interés por este campo, así como por vuestra guía y orientación durante este proyecto.

*Rocío Méndez Hernández.*

*Sevilla, 2019.*

# Resumen

La imagen médica es un campo donde han confluido diversas disciplinas como la medicina, la ingeniería electrónica, la física y la informática, estas tres últimas muy relacionadas con la ingeniería de telecomunicaciones. Su análisis forma parte de lo que se conoce como sistemas de diagnóstico asistido por ordenador (CAD: *Computer Aided Diagnosis*), y ayudan a los profesionales de la medicina a interpretar toda la información disponible, a resaltar lo más relevante y en definitiva a alcanzar un diagnóstico más completo y óptimo.

En este trabajo fin de máster se propone un método automático de análisis de imágenes digitales de lesiones en la piel (imágenes dermatoscópicas) mediante técnicas de aprendizaje profundo o *deep learning*. En concreto, el proyecto se centrará en la segmentación de la lesión dentro de la imagen, con el objetivo de posteriormente poderla clasificar en las diferentes enfermedades que podría suponer, entre ellas el melanoma. El cáncer de piel es una enfermedad en la que la prevención no es sólo fundamental, sino posible. Al tratarse de lesiones visibles en la superficie de la piel, los pacientes pueden revisárselas con frecuencia y acudir a dermatólogos expertos a que las inspeccionen, pudiendo éstos hacer usos de herramientas de ayuda al diagnóstico.

Para llevar a cabo la segmentación, se utilizará una modificación de la red neuronal *U-Net* [1] propuesta en 2015 por los investigadores de la Universidad de Friburgo Ronneberger, Fischer, y Brox, una red con muy buen desempeño en diferentes aplicaciones de segmentación biomédica. De hecho, la *U-Net* superó el rendimiento del método ganador del concurso de segmentación de imágenes microscópicas de estructuras neuronales del ISBI (*International Symposium on Biomedical Imaging*) de 2012 [2], el cual fue una red neuronal con ventana deslizante propuesta por Ciresan et al. [3]. Además, ganó el concurso de rastreo de células del ISBI en 2015 [4] por un amplio margen.

# Abstract

Medical imaging is a field in which various disciplines such as Medicine, Electronic Engineering, Physics and Information Technology have come together, the latter three being closely related to Telecommunications Engineering. Its analysis is part of what is known as computer aided diagnosis (CAD) systems, and they help medical professionals to interpret all available information, highlight the most relevant features and, in short, to achieve a more complete and optimal diagnosis.

This master's thesis proposes an automatic method for the analysis of digital images of skin lesions (dermoscopic images) based on deep learning techniques. Specifically, the project will focus on the segmentation of the lesion within the image, with the aim of later classifying it into different diseases, including melanoma. Skin cancer is a disease in which prevention is not only fundamental, but also possible. As they are visible lesions on the surface of the skin, patients can check them frequently and have them inspected by expert dermatologists, who can use computer diagnosis tools.

In order to carry out the segmentation, a modification of the *U-Net* neural network [1] will be used, which was proposed in 2015 by Ronneberger, Fischer, and Brox, researchers from the University of Freiburg. This network achieved very good performance in different biomedical segmentation applications. In fact, it outperformed the prior best method of the ISBI (International Symposium on Biomedical Imaging) challenge (2012) for segmentation of microscopic images of neural structures [2], which was a sliding-window convolutional network proposed by Ciresan et al. [3]. Furthermore, the *U-Net* won the 2015 ISBI cell tracking challenge [4] by a wide margin.

# Índice

# 1 INTRODUCCIÓN

*Los avances tecnológicos no pueden suceder sin científicos o ingenieros. El desafío de la sociedad es equiparar a las suficientes personas, con las habilidades correctas y formas de pensar, que lleguen a trabajar en los problemas más importantes.*

– Eric Schmidt –

Los avances tecnológicos llevan revolucionando el mundo de la medicina desde finales del siglo XIX [5]. El descubrimiento de los rayos X, así como del electrocardiograma, supusieron un cambio en la concepción de la medicina y en el diagnóstico de pacientes. Desde entonces, múltiples áreas de la ingeniería y las telecomunicaciones han dedicado esfuerzos al desarrollo de tecnologías médicas, ayudando tanto en el diagnóstico, como en el tratamiento o en la prevención de enfermedades. Son invenciones como el marcapasos, la endoscopia, el respirador artificial, la ecografía, la resonancia magnética, etc. Frecuentemente, el objetivo de estas innovaciones tecnológicas es la obtención de imágenes médicas, cuyo análisis es sin duda una herramienta fundamental de diagnóstico en la actualidad.

Aún más, estos avances han supuesto, además de la dedicación de ingenieros electrónicos, eléctricos o de telecomunicaciones, la aparición de nuevas carreras dedicadas exclusivamente a la aplicación de la ingeniería al campo médico, como pueden ser la Ingeniería de la Salud o la Ingeniería Biomédica. Tampoco se puede olvidar el papel que tienen en la medicina actual la historia clínica computarizada o los softwares de ayuda al diagnóstico, posibles gracias al progreso meteórico de la computación en los últimos 30 años.

En la actualidad se sigue innovando a pasos agigantados en el ámbito de la sanidad, en el que se han involucrado industrias como la robotización, la realidad virtual, la impresión 3D o la inteligencia artificial. En esta última, en concreto en el aprendizaje automático aplicado a imágenes médicas, se centrará el presente proyecto.

A lo largo de este capítulo, se ofrecerá una introducción a la motivación de este proyecto. En primer lugar, se contextualizará la motivación médica, que no es otra que la detección del cáncer de una forma más fiable. En segundo lugar, se introducirá la técnica utilizada para dicho propósito, la segmentación. Por último, se hablará brevemente del reto al que intenta responder este trabajo de fin de máster.

## 1.1 Motivación médica

El pasado año 2018, se diagnosticaron alrededor de 18 millones de casos nuevos de cáncer, según los datos del Centro Internacional de Investigación sobre el Cáncer (IARC) [6], de los cuales aproximadamente 300000 fueron casos de melanoma, lo que representa el 1,6% del total. Esto supone un 20% de todos los casos de cáncer de piel; sin embargo, este porcentaje sube hasta el 50% cuando se habla de tasas de mortalidad, lo que convierte al melanoma en el cáncer de piel más letal.

### 1.1.1 Detección del melanoma

A pesar de estos datos de incidencia y mortalidad, el melanoma es un cáncer con una alta tasa de supervivencia si se detecta temprano, pudiendo tratarse y evitar así su extensión de las capas superiores de la piel, a las capas más profundas y a otras partes del cuerpo, donde su tratamiento se hace más complicado [7].

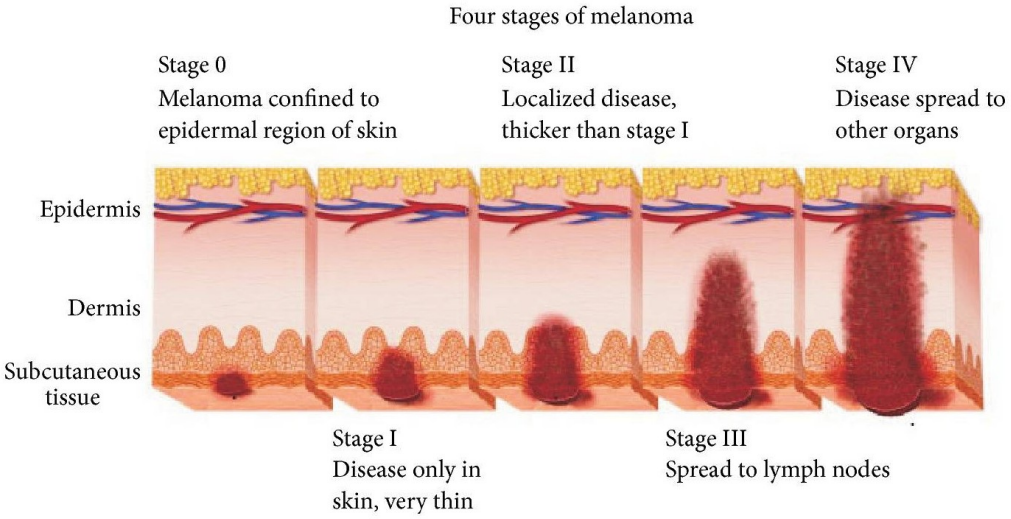



*Figura 1-1: Etapas del melanoma [8].*

En la *Figura 1-1* se muestran las etapas del melanoma. Cuanto más pequeño y más fino sea –primeras etapas–, mayor será la probabilidad de curación pues se podrá eliminar satisfactoriamente mediante una escisión. Por ello, es muy recomendable la revisión frecuente de las lesiones de la piel, para poder diagnosticarlo de manera precoz y evitar que llegue a las fases III y IV.

Sin embargo, los melanomas en las primeras etapas pueden ser difíciles de distinguir de otras lesiones benignas como los lunares, las manchas marrones o los crecimientos de la piel, como puede observarse en la Figura 1-2. Son lo que se conoce como **lesiones pigmentadas**: lesiones cutáneas que tienen una coloración anormal. De hecho, aproximadamente entre el 20% y el 30% de los melanomas se desarrollan a partir de lesiones ya existentes [7]. Por ello, y dado que son visibles en la superficie de la piel, se puede –y se debe– vigilar las propias lesiones pigmentadas y acudir a dermatólogos expertos a que las inspeccionen [9] [10].

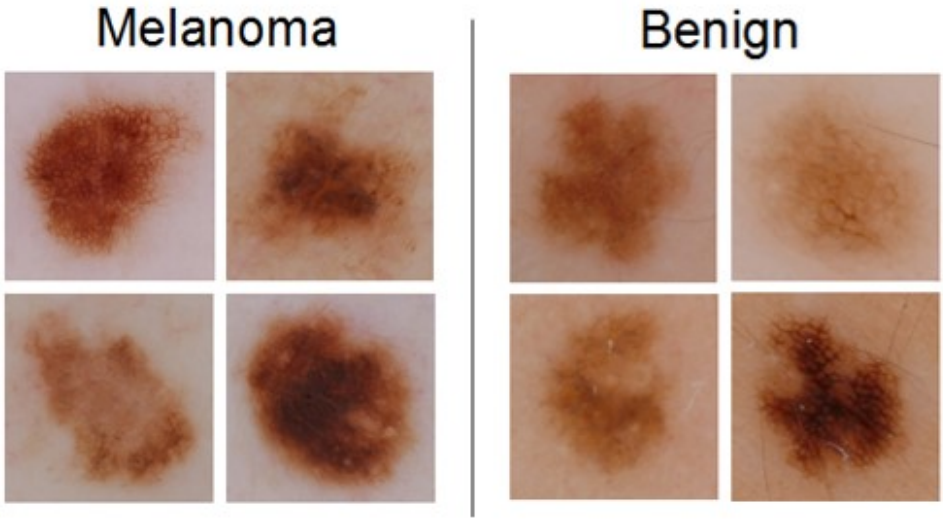


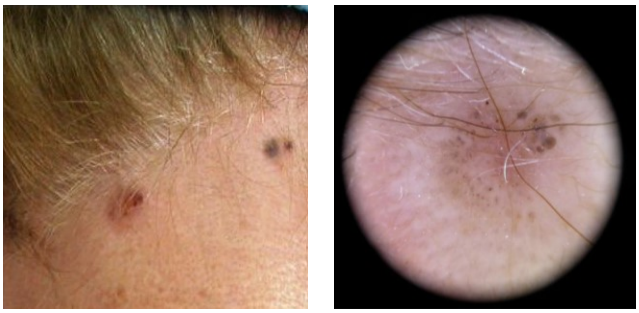

*Figura 1-2: Lesiones pigmentadas malignas (melanoma) y benignas [11].*

### 1.1.2 Dermatoscopia

La técnica utilizada en dermatología para la visualización de las lesiones de la piel es la **dermatoscopia**. Es un procedimiento no invasivo para el cual se emplea un dispositivo –dermatoscopio– sobre la piel, que amplifica 10 veces la imagen a la vez que elimina las reflexiones, permitiendo visualizar las características lesión con más detalle y apreciar capas más profundas de la misma [9] [12].

*Figura 1-3: Dermatoscopio [9].*

En comparación con un examen de la lesión a simple vista, el uso del dermatoscopio mejora la precisión en el diagnóstico del melanoma, principalmente por el mayor nivel de detalle que se obtiene, que permite la apreciación de características de la lesión no discernibles a simple vista [13]. Esta diferencia puede observarse en la siguiente figura, en la que se recogen dos imágenes de la misma lesión, una con dermatoscopio y otra con una cámara normal.



(a) (b)

*Figura 1-4: Lesión pigmentada [10].*
*(a) Imagen macroscópica (a simple vista), (b) imagen dermatoscópica.*

Adicionalmente, hay dermatoscopios que pueden conectarse a sistemas informáticos donde poder ver la imagen con más detalle, lo que se suele denominar **dermatoscopia digital**. Estos sistemas permiten la digitalización y el almacenamiento de las imágenes, facilitando el control y seguimiento de los lunares, especialmente si el paciente presenta un gran número de ellos. [10] [12]

Asimismo, la imagen digital obtenida se puede analizar computacionalmente, procesándola con algoritmos que permitan la detección automática del melanoma. Los esfuerzos de la comunidad científica en este sentido están centrados actualmente en la aplicación de métodos de aprendizaje automático, y más concretamente en técnicas de aprendizaje profundo (*deep learning*). Este enfoque está ganando interés debido a los prometedores datos de rendimiento que se están obteniendo recientemente [9].

## 1.2 Segmentación de lesiones pigmentadas

El objetivo de este proyecto, como indica su propio título, será la **segmentación** de imágenes dermatoscópicas de lesiones pigmentadas, con el propósito de posteriormente clasificarlas en diferentes enfermedades. En esta sección se estudiará brevemente en qué consiste la segmentación y cuál es su utilidad en este tipo de problemas, dejando para el capítulo 2 un análisis más detallado de las técnicas de segmentación y el estado del arte en cuanto a lesiones pigmentadas.

### 1.2.1 Introducción a la segmentación

La segmentación consiste en dividir una imagen en varias partes o zonas llamadas segmentos. Normalmente, su finalidad es localizar objetos en la imagen, o bien detectar límites o bordes dentro de la misma. Para ello, a cada píxel de la imagen se le asigna una etiqueta o **categoría**, separándolos en regiones que comparten alguna característica, como puede ser el color, la textura, el nivel de gris, el brillo, el contraste, etc.

En las siguientes figuras se muestran varios ejemplos de segmentación. En la *Figura 1-5* se puede observar una segmentación de una imagen en varias categorías. La división genera 4 clases que pueden posteriormente identificarse como *tigre*, *césped*, *arena* y *río*; es lo que se conoce como **segmentación semántica**. En la *Figura 1-6* se muestra cómo se puede segmentar la misma imagen, pero esta vez con el objetivo de hallar los **bordes** o discontinuidades presentes en ella. En este caso, se puede decir que hay 2 categorías: *borde* y *no borde.* Finalmente, en la *Figura 1-7* se clasifican los píxeles de la imagen también en 2 categorías: el **objeto o región de interés** –el *tigre,* en este ejemplo– y el fondo.

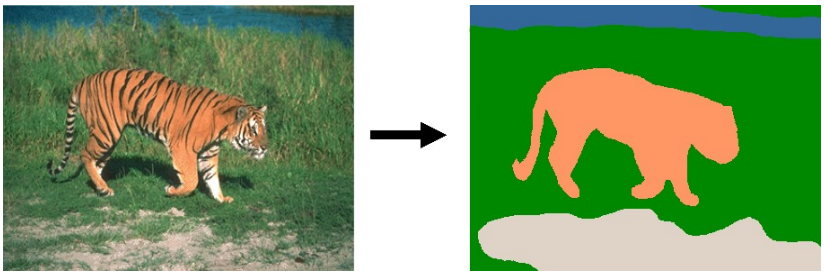


*Figura 1-5: Ejemplo de segmentación en 4 categorías [14].*

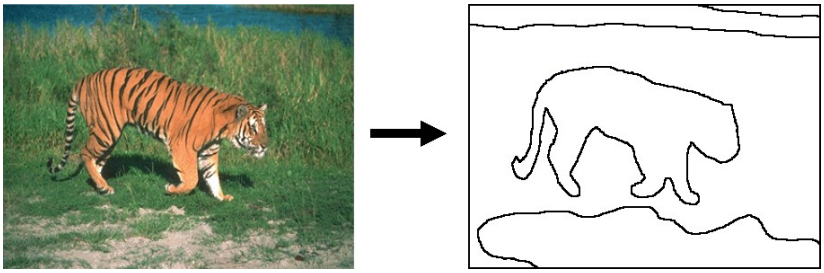


*Figura 1-6: Ejemplo de segmentación para detección de bordes.*

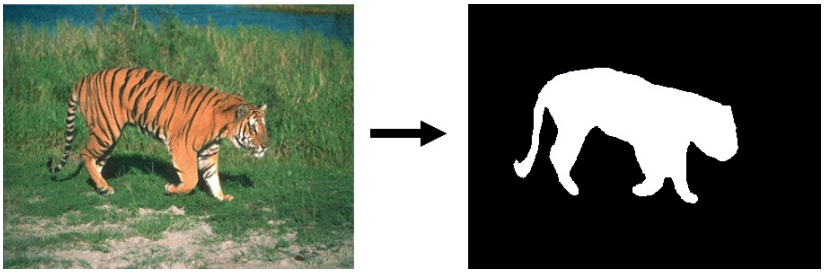


*Figura 1-7: Ejemplo de segmentación en 2 categorías (objeto de interés y fondo).*

Como se puede observar, el resultado de la segmentación de una imagen es otra imagen, llamada **máscara**, en la que cada píxel tiene asignado un color único según la categoría asignada. Suele ser común representar la imagen original con la máscara superpuesta, de forma que se pueda apreciar con más detalle la calidad de la segmentación. La superposición de los ejemplos anteriores con su máscara correspondiente se muestra en la *Figura 1-8.*

(a) (b) (c)

*Figura 1-8: Superposición de imagen original y máscara de segmentación.*
*(a) 4 categorías. (b) Detección de bordes. (c) 2 categorías (objeto de interés y fondo).*

En definitiva, el fin último de una segmentación va a ser simplificar o modificar de algún modo la representación de una imagen en algo que sea más significativo y/o más fácil de analizar [13].

### 1.2.2 Aplicación en imágenes médicas

Como se ha comentado, los métodos automáticos de análisis de las imágenes de posibles melanomas están ganando interés. Esto es cierto tanto para el caso de la detección del melanoma, como para el análisis de imágenes médicas de cualquier tipo, y es lo que se conoce como **diagnóstico asistido por ordenador** (DAO, o CAD por sus siglas en inglés). El uso de estos sistemas en imágenes médicas resulta muy conveniente para profesionales de la medicina ya que les pueden ayudar a interpretar toda la información disponible, a resaltar lo más relevante y en definitiva a alcanzar un diagnóstico más completo y óptimo [13] [15].

El análisis computacional de imágenes suele incluir 4 etapas [11] [15]:

i) **Preprocesamiento**, cuyo objetivo es la eliminación del ruido de la imagen y en ocasiones la corrección de niveles en la misma.

ii) **Segmentación**, para detectar las diferentes regiones de la imagen.

iii) **Extracción de características**, que analizará el resultado de la segmentación con el propósito de hallar atributos de interés en la imagen, que dependerán del problema a tratar. También se le llama reconocimiento de patrones.

iv) **Clasificación**, basándose en las características o no presentes en la imagen y en su importancia para el problema a tratar.

La segmentación es por tanto un paso importante en el procesamiento de imágenes médicas, pues va a detectar la estructura de la imagen, la cual se necesitará para posteriormente poder hallar las regiones de interés o las características relevantes, que tendrán más o menos importancia en el diagnóstico.

De hecho, en el caso particular de las imágenes médicas, se suele evitar el preprocesamiento para perder la menor información posible, pues todo puede ser relevante para el diagnóstico. Así, la segmentación se convierte en el primer paso para una clasificación de la imagen, el cual es el objetivo final, y por tanto de especial transcendencia para evitar errores posteriores.

En el caso de las lesiones pigmentadas, a las que concierne este proyecto, se tratará de segmentar las imágenes con el objetivo de delimitar la localización de la lesión, siendo ésta la **región de interés**, y obviar de esta forma la piel sana y otros elementos como el vello. En la siguiente figura se recogen varios ejemplos.

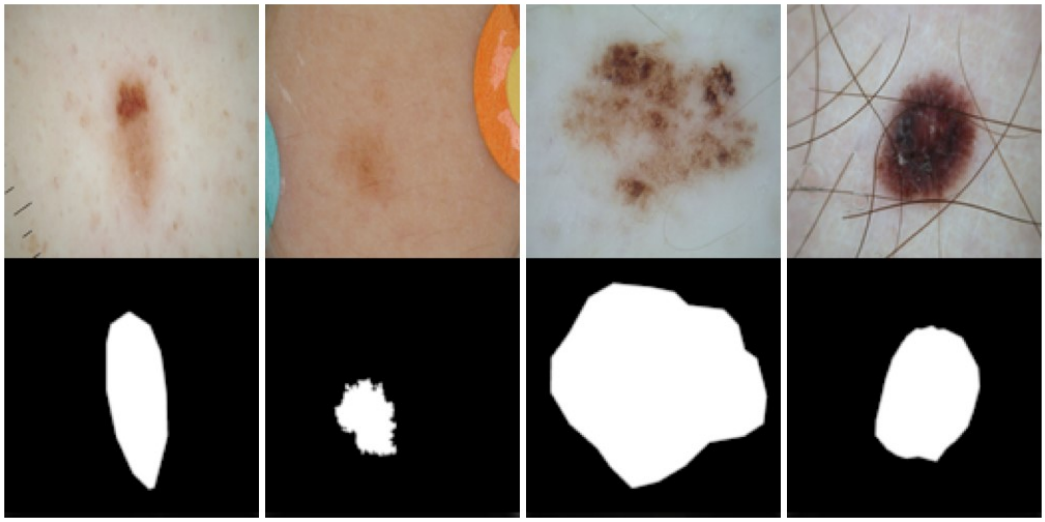


*Figura 1-9: Segmentación de lesiones pigmentadas [11].*

## 1.3 ISIC Challenge

Para intentar reducir la tasa de mortalidad del melanoma, se creó la *International Skin Imaging Collaboration* (ISIC) [11], una asociación entre la industria y la comunidad académica diseñada para facilitar el análisis computacional de imágenes de la piel. Está centrada en el desarrollo de estándares para las tecnologías, técnicas y terminología utilizadas en este ámbito, así como el mantenimiento de un archivo de acceso público de imágenes digitales de lesiones cutáneas con las que poder validar los estándares propuestos. Este archivo sirve además como un **recurso público de imágenes** para la enseñanza y para el desarrollo y prueba de sistemas de diagnóstico automático.

### 1.3.1 ISIC Challenge: Análisis de lesiones cutáneas para la detección del melanoma

Para impulsar los avances orientados a la mejora de la eficiencia y la precisión en el diagnóstico de melanoma, la ISIC organiza cada año, desde el 2016, un concurso o *challenge* cuyo objetivo es el desarrollo de herramientas de análisis de imágenes para el diagnóstico automático del melanoma. Este proyecto tratará de responder al *challenge* correspondiente al pasado año 2018.

Este reto, así como los de los 2 años anteriores, se divide en 3 tareas:

i) **Segmentación**, cuyo propósito es hallar los bordes de la lesión pigmentada en una imagen dermatoscópica.

ii) **Detección de características**, cuyo objetivo es la localización de atributos en la lesión considerados como patrones clínicamente significativos.

iii) **Diagnóstico**, que tratará de clasificar la lesión según las categorías de enfermedades establecidas, siendo una de ellas el melanoma.

Para cada tarea, se proporciona a los participantes amplias bases de datos consistentes en **imágenes dermatoscópicas** procedentes del archivo de ISIC, así como los resultados esperados para cada una de las tareas, de forma que se pueda comparar con el rendimiento del algoritmo desarrollado.

Es un proyecto que está teniendo un éxito notable, y que va mejorando cada año, tanto en cantidad de imágenes disponibles con sus resultados asociados, como en posibles características presentes en la lesión (tarea 2), y en número de categorías de clasificación de las lesiones (tarea 3). Para el *challenge* de 2018, más de 10 mil imágenes fueron proporcionadas para las 3 tareas, procedentes de las fuentes de datos arXiv:1902.03368 [16] y HAM10000 [17]. Es un crecimiento significativo si se compara con las disponibles en el primer desafío en 2016, que fueron 900.

Con el crecimiento de las bases de datos, ha sido posible además refinar la clasificación de las lesiones. Cuando en la convocatoria de 2016 se proponía la clasificación de las lesiones entre melanoma y lesión benigna, en el *challenge* de 2018 se distinguen 7 categorías, de las cuales 4 son malignas, incluido el melanoma. Esto implica también un aumento en los atributos dermatológicos característicos de cada una de las categorías.

### 1.3.2 ISIC Archive

El archivo de la ISIC contiene en la actualidad más de 13000 imágenes dermatoscópicas, adquiridas en una variedad de dispositivos y en diferentes centros clínicos líderes a nivel internacional. La contribución al archivo es por tanto muy extensa y heterogénea, de forma que se asegure una muestra representativa y clínicamente relevante.

Todas las imágenes que se aportan al archivo son analizadas para garantizar su calidad. La mayoría incluyen sus propios metadatos clínicos, los cuales son examinados por expertos en melanoma. Asimismo, parte de las imágenes que no tienen ya atributos asignados son caracterizadas también por expertos. De esta forma, las imágenes incluirán las características dermatológicas necesarias para poder discriminar entre los distintos tipos de lesiones cutáneas y para ser susceptibles de análisis por partes de algoritmos de diagnóstico automático.

# 2 METODOLOGÍA

*La inteligencia es la habilidad para adaptarse a los cambios.*

– Stephen Hawking –

La inteligencia artificial está de moda. Desde que se han desarrollado las capacidades de procesamiento necesarias, las tecnologías de inteligencia artificial han avanzado considerablemente, impulsado también por la cantidad de datos disponibles actualmente en Internet, necesarios para entrenar los algoritmos. Aunque son métodos que la comunidad científica y académica lleva estudiando desde mediados del siglo XX, no ha sido hasta este despegue y su consecuente aplicación en el mundo empresarial que se han empezado a oír frecuentemente términos como *machine learning* o inteligencia artificial. Casi cualquier negocio puede hacer uso de ello para analizar sus datos y buscar tendencias, personalizar sus interacciones con los usuarios o las campañas de marketing. Gigantes como Amazon, Google o Microsoft están apostando por estas tecnologías y haciendo disponibles servicios de *machine learning* al público general, con la potencia de procesamiento requerida, en sus respectivas plataformas: Google Cloud, AWS y Azure.

En este capítulo se verá la aplicación del *machine learning* al ámbito concreto que atañe a este proyecto: la segmentación de imágenes dermatoscópicas. Se comenzará por un repaso al estado del arte en segmentación de lesiones pigmentadas, para dar una visión general de las técnicas utilizadas. A continuación, se entrará de lleno en los conceptos y técnicas de *deep learning*, incluyendo su enmarcamiento dentro del mundo de la inteligencia artificial y el *machine learning*. Finalmente, se terminará con el propósito último de este capítulo y este proyecto: el método propuesto para segmentación de lesiones pigmentadas.

## 2.1 Estado del arte en segmentación de lesiones pigmentadas

Las primeras evidencias en la literatura de análisis computacional de lesiones pigmentadas se remontan a 1987 [13], y desde entonces se han ido aplicando distintos algoritmos con más o menos éxito, hasta llegar a la actualidad en la que el estándar se ha convertido en las redes neuronales.

Como se verá en los próximos apartados, la segmentación de lesiones se ha convertido en una de las áreas importantes de investigación y se encuentran disponibles numerosos algoritmos y técnicas de segmentación, debido a su importancia como primer paso crucial en el análisis de imágenes dermatoscópicas [13] [18].

### 2.1.1 Técnicas clásicas de segmentación

Existen numerosas técnicas de segmentación, y la complejidad del problema y los diferentes propósitos buscados hace que sea muy complicado encontrar un método que se ajuste a todos los tipos de aplicaciones. Para

segmentación de imágenes dermatoscópicas, los métodos basados en agrupamientos (*clustering*) y los basados en umbrales (*thresholding*) han sido históricamente los más estudiados [18].

### Thresholding

El *thresholding* o umbralización es la técnica de segmentación más sencilla, y consiste en establecer umbrales de intensidad del color, para después dividir a los píxeles en grupos dependiendo de a qué umbral pertenezcan. Para lesiones pigmentadas, se establece un umbral y se compara el color de cada píxel con el mismo, de forma que si es más oscuro se clasifica como activo (valor binario '1') –lo que indica presencia de lesión–, y en caso contrario se considerará que es fondo (no activo o '0') [19].

El umbral puede establecerse conforme a la intensidad del nivel de gris, una combinación de la intensidad de los niveles de cada componente de color, o bien considerar la intensidad de una sola componente [19] [20]. Experimentalmente, se demostró que la componente RGB que mejor funciona para este tipo de lesiones es la azul [21].

Por otro lado, la estrategia de elección del umbral puede ser **global** o **local**. La umbralización global se basa en el histograma para establecer un umbral único para toda la imagen. Si por el contrario el umbral depende de las propiedades locales de algunas regiones de imagen, como puede ser el nivel de gris medio de la zona, el umbral se llama local. [22]

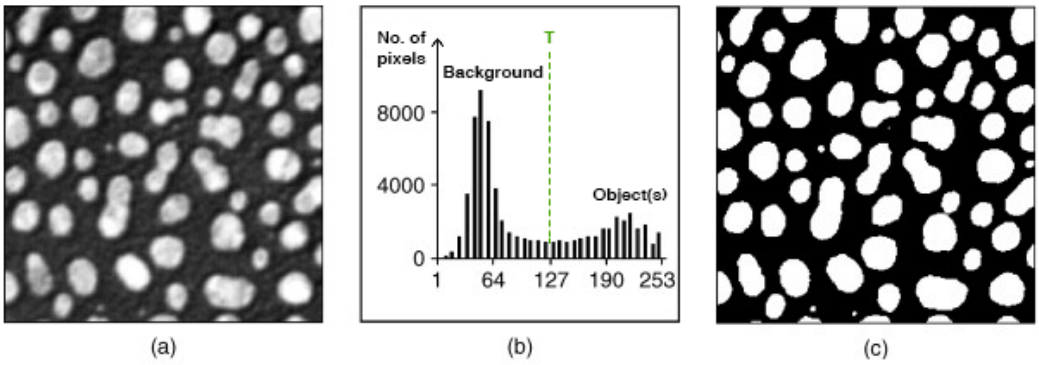



*Figura 2-1: Ejemplo de umbralización global [22].*
*(a) Imagen original, (b) histograma de la imagen,*
*(c) resultado de la umbralización con T=127.*

### Clustering

Las técnicas de *clustering* o agrupación consisten a grandes rasgos en agrupar píxeles que tienen características similares en un mismo grupo o *cluster.* Los píxeles serán más semejantes –con respecto de la característica que se tenga en cuenta– a los píxeles de su propio grupo que a los de otro [19].

Uno de los algoritmos de *clustering* más utilizados es el **k-medias** o *k-means,* el cual asume que la característica o características conforme a las que agrupar forman un espacio vectorial, de forma que cada píxel será representado por un punto en dicho espacio vectorial. Es por tanto típico utilizar el valor de intensidad del color, en el espacio de color que se desee [19].

De esta forma, se puede calcular la semejanza de cada píxel con los *clusters* utilizando la distancia euclídea. En concreto, el algoritmo buscará minimizar la distancia entre cada píxel y los centros de los grupos. Por lo tanto, para el k-medias será necesario el número de grupos, *k*, y los valores iniciales de los centros [20].

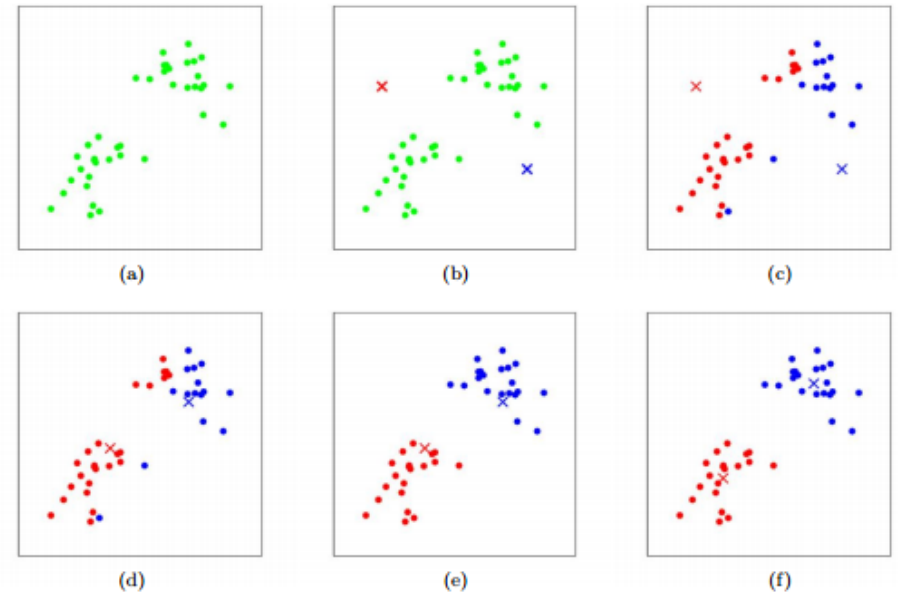



*Figura 2-2: Algoritmo k-medias [23].*
*Los datos de entrada se muestran como puntos, y los centros de los clusters como cruces. (a) Conjunto de datos original. (b) Centros iniciales aleatorios. (c-f) Ilustración de la ejecución de dos iteraciones del algoritmo. En cada iteración, se asigna cada punto al centro más cercano; luego se recalcula el valor de cada centro a la media de los puntos asignados.*

**Comparativa**

El método de umbralización tiene la ventaja de ser muy simple, y su velocidad es alta. Sin embargo, tiene el problema de que si el contraste entre la lesión y el fondo es bajo, se tornará complicado obtener una buena segmentación. En la Figura 2-3 se muestra un ejemplo de imagen en la que no es sencillo en el histograma distinguir entre el objeto y el fondo. Además, dado que no considera la información espacial de la imagen sino únicamente el nivel del color, será sensible al ruido.

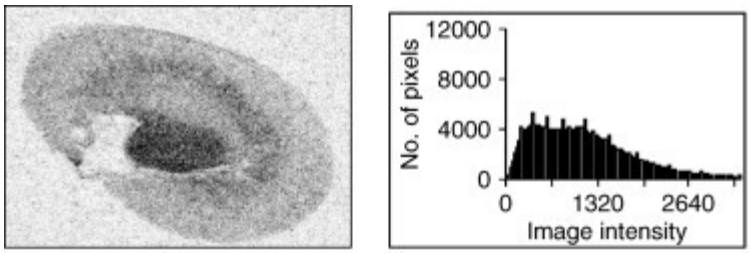



*Figura 2-3: Histograma de una imagen de bajo contraste [22].*

El k-medias es, al igual que el método de umbralización, un algoritmo sencillo y rápido. El inconveniente que tiene es que la selección del número de *clusters* k no tiene un criterio establecido y de hecho suele ser complicado de estimar. Además, es fácil quedarse atrapado en mínimos locales, en lo cual la elección de los centros iniciales puede tener una gran influencia. Aunque el basado en el color no tiene tampoco en cuenta la información espacial, puede tenerse en cuenta para ser más resistente al ruido.

En cualquier caso, ningún método es perfecto para todas las imágenes ni todos los métodos son adecuados para una sola, ya que los resultados de una segmentación dependen de numerosos factores como pueden ser la textura de la imagen, la intensidad y sus contenidos. Por supuesto, se pueden probar modificaciones de los algoritmos o incluso combinaciones con el objetivo de intentar mejorar los resultados [19] [24].

### 2.1.2 Técnicas actuales de segmentación

En la actualidad, las técnicas de segmentación usadas para imágenes dermatoscópicas están dominadas por las

redes neuronales convolucionales (CNN), enmarcadas dentro de lo que se conoce como aprendizaje profundo o *deep learning*. No obstante, la evolución del *deep learning* también ha conllevado su uso para clasificación de las imágenes sin previa segmentación, paso que hasta ahora se consideraba fundamental. De hecho, el *challenge* de ISIC de 2019 ya no propone las 3 tareas mencionadas anteriormente, sino que hay una única tarea de clasificación de las lesiones en distintas enfermedades. Sin embargo, esto no quiere decir que su estudio haya quedado totalmente descartado, ya que su necesidad o no en el análisis de lesiones cutáneas no ha sido concluyentemente demostrado [18] [24].

La segmentación en general es un campo de investigación muy activo debido a su gran importancia y emergencia en aplicaciones del mundo real, principalmente en aplicaciones de **visión artificial**, por lo que se espera seguir viendo muchos más trabajos en los próximos años. Otras iniciativas como el *challenge* de ISIC, relacionadas o no con el campo médico, impulsan del mismo modo la investigación en este área. Por lo tanto, aunque no esté aún demostrada la necesidad o no de la segmentación de lesiones pigmentadas previa a una clasificación, la segmentación es un campo que está muy vivo y cuyos avances en otras aplicaciones son susceptibles de extrapolarse al caso de imágenes dermatoscópicas.

## 2.2 *Deep learning*

Inteligencia artificial, *machine learning, deep learning, ...,* los términos se escuchan frecuentemente y a menudo indistintamente, por lo que es sencillo confundirse. Sin embargo, la inteligencia artificial abarca una gran cantidad de aspectos, y al mismo tiempo no todos los algoritmos se pueden considerar inteligencia artificial.

En esta sección se estudiará la diferencia entre estas expresiones y se profundizará más en la tecnología central de este proyecto: el *deep learning.*

### 2.2.1 Introducción al *machine learning*

Se considera **inteligencia artificial** (IA, o **AI** por sus siglas en inglés) todo aquel sistema informático que trate de resolver problemas imitando de alguna forma la inteligencia humana. La inteligencia humana tiene una serie de cualidades como son la habilidad para interactuar con el mundo real, el razonamiento, el aprendizaje, la adaptación o la toma de decisiones. Todo ello hace que las posibles aplicaciones de este tipo de tecnologías sean infinitas [25].

Asimismo, abarca un espectro muy amplio porque su definición no incluye nada sobre cómo se resuelven esos problemas. No depende únicamente de la aplicación, sino que un mismo problema se puede afrontar con varios enfoques diferentes. Uno de estos enfoques comenzó a utilizarse más ampliamente en la década de 1980: el aprendizaje automático.

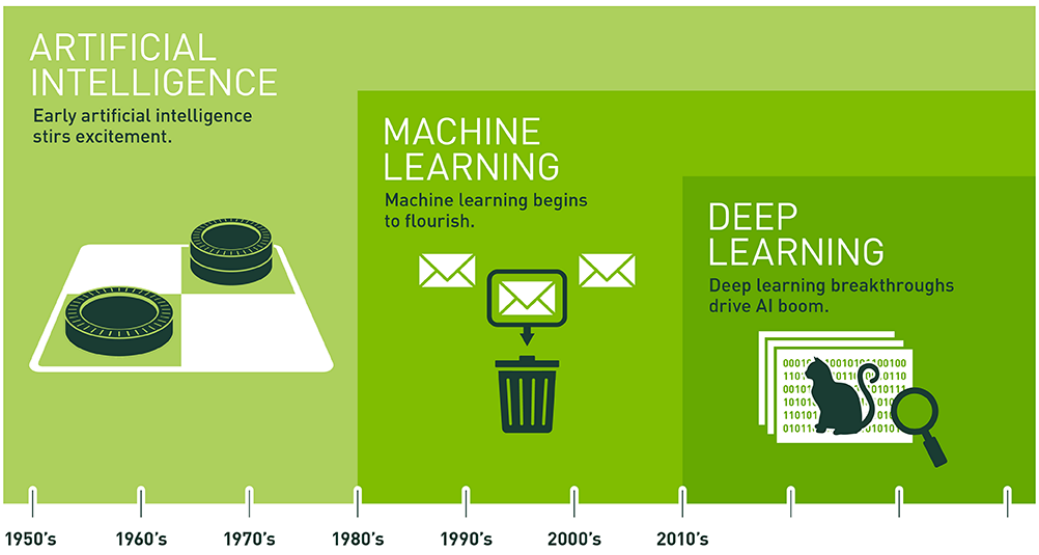



*Figura 2-4: Evolución de la inteligencia artificial [26].*

El **aprendizaje automático**, aprendizaje máquina o *machine learning* (**ML**), es por tanto un tipo de inteligencia artificial –un subconjunto dentro de ella– que se centra en la capacidad del cerebro humano para aprender. Para ello, se utilizan algoritmos capaces de analizar datos, aprender de ellos y luego hacer una predicción sobre datos nuevos. El proceso de aprendizaje de un modelo es lo que se conoce como **entrenamiento**, y si lo hacemos con la suficiente cantidad de datos, éste será capaz de predecir con exactitud la salida esperada para futuras entradas [26] [27] [28].

Éste va a ser el objetivo último del aprendizaje automático: predecir resultados dados unos datos de entrada. Con esto, la estructura general de cualquier problema de aprendizaje automático se puede representar como se muestra en la siguiente figura:

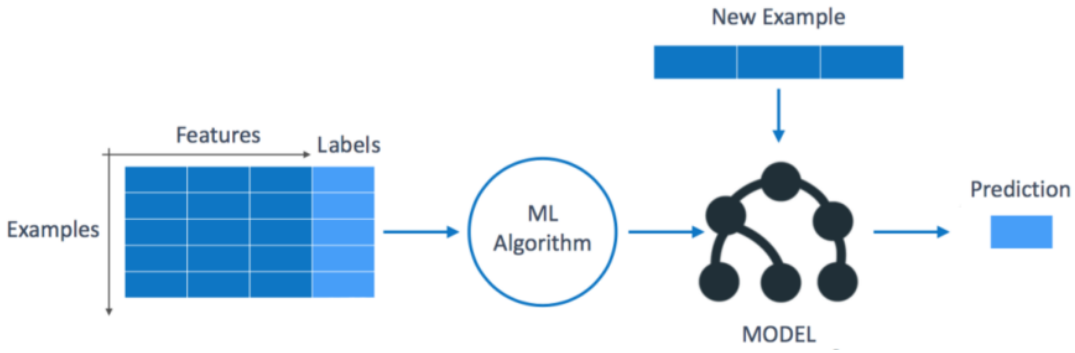



*Figura 2-5: Estructura de un problema de aprendizaje automático [28].*

En ella identificamos ciertos términos clave en el mundo del *machine learning* [25]:

- Ejemplos: son los datos con los que se "alimenta" al sistema, a los que se denomina **datos de entrenamiento.**
- **Etiquetas** (*labels*): es el resultado conocido de los datos de entrenamiento. También se le suele designar como *ground truth* o valor real.
- **Modelo**: es la función que predice las etiquetas dados unos datos de entrada. El modelo tendrá una serie de **parámetros** que se irán ajustando según se entrena el modelo mediante el algoritmo de aprendizaje, de forma que la diferencia entre las etiquetas predichas por el modelo (***predictions***), y las etiquetas reales (***ground truth***) sea mínima.
- Algoritmo de aprendizaje: se encarga de comparar la salida esperada (etiquetas) con la que predice el sistema, para realizar los ajustes necesarios a los parámetros del modelo. Para ello se utiliza una **función de coste** que penalizará los fallos de predicción o cualquier otra métrica relevante.
- Predicción: Una vez terminado el entrenamiento, el modelo será capaz de predecir la salida para nuevos datos del mismo tipo que los de entrenamiento, pero diferentes, y de los que se desconoce la etiqueta. Es lo que se denomina **generalización.**

### Entrenamiento, validación y evaluación

La elección del modelo es clave en cualquier problema de aprendizaje automático. Hay cantidad de modelos genéricos de diferente complejidad de los que se suele partir, y cada uno tiene sus propios parámetros y requerirán ser ajustados con distintos enfoques. Algunos ejemplos son los regresores, los árboles de decisión, los clasificadores gaussianos, las máquinas de soporte vectorial o las redes neuronales [28]. Además de sus parámetros característicos, los modelos tienen otra serie de parámetros que controlan la capacidad de aprendizaje y que no deben ser nunca ajustados en el entrenamiento: los **hiperparámetros**.

También es primordial, para que el modelo que generalice lo más correctamente posible y haga buenas predicciones, conseguir un buen conjunto de **datos de entrenamiento**. Para ello, no sólo es necesario que el conjunto de datos sea grande, sino que además deberá ser claramente representativo de la tarea a aprender, es decir, suficientemente general. Si un modelo no se ajusta correctamente a los datos de entrenamiento, se dice que está **subajustado** [25].

Para comprobar cómo de bien generaliza un modelo, se reserva parte de los datos de entrenamiento para realizar

una validación del mismo. Dado que se conoce la etiqueta esperada de los mismos, se puede medir la **precisión** o exactitud con la que el modelo está prediciendo. El modelo nunca deberá aprender de los **datos de validación**, de manera que la medida de la precisión sea "imparcial", pero sí se pueden utilizar para modificar los hiperparámetros y llegar a una solución óptima.

El objetivo de la validación es evitar que el modelo se ajuste demasiado a los datos de entrenamiento, ya que en este caso se estará modelando también el ruido y las anomalías presentes en estos datos, y consecuentemente generalizará peor. Esto es lo que se conoce como **sobreajuste** o *overfitting*.

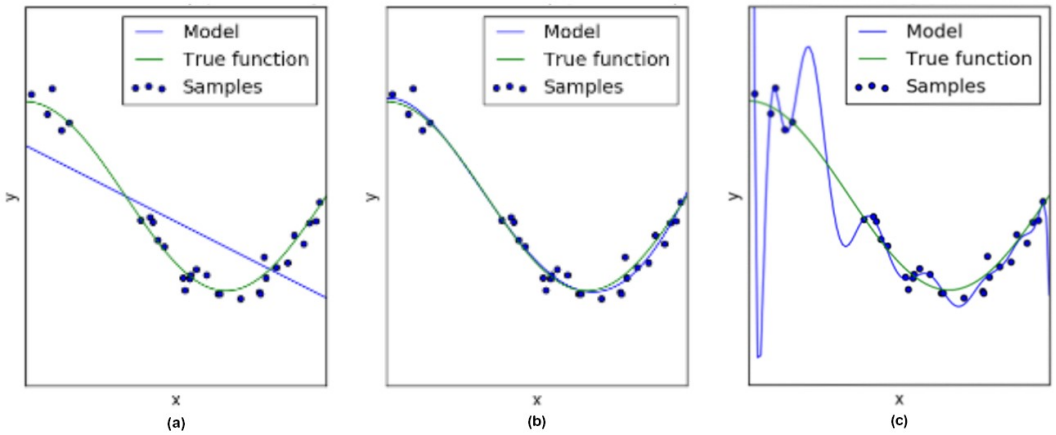



*Figura 2-6: Diferentes ajustes de un modelo de aprendizaje automático [25].*
*(a) Subajustado, (b) buen ajuste, (c) sobreajustado.*

Una vez finalizado el entrenamiento, y ajustados tanto los parámetros como los hiperparámetros, se puede obtener una medida final de la calidad del modelo evaluando sus capacidades de predicción con un conjunto de datos de evaluación o **datos de test**. El conjunto de test contiene datos cuidadosamente elegidos que abarcan las diversas clases a las que se enfrentaría el modelo cuando se utilizase en el mundo real [29].

### Tipos de algoritmos de aprendizaje automático

Este funcionamiento –basado en comparar valores predichos con valores reales– es el tipo más común de *machine learning*, sin embargo se pueden dividir los algoritmos de aprendizaje automático en 3 clases [30]:

- **Aprendizaje supervisado**: se tiene disponible un conjunto de datos con sus etiquetas reales. El algoritmo tratará de minimizar la diferencia entre la salida predicha y la real, realizando un entrenamiento.
- **Aprendizaje no supervisado**: no se conoce la salida real de los datos de entrada. El sistema tratará de buscar similitudes o patrones en los datos, para interpretarlos y buscar soluciones.
- ***Reinforcement learning***: no se tienen las salidas reales de los datos de entrada, pero se interacciona con un entorno dinámico que le va indicando lo bien o mal que lo está haciendo, con lo que el sistema aprende a actuar en ese entorno concreto.

Dentro del aprendizaje supervisado, se distinguen 2 tipos de problemas: los de **regresión** y los de **clasificación**. En un problema de regresión, se tratará de predecir un valor continuo (un número), y en los de clasificación la salida es discreta, indicando a qué categoría pertenece cierta entrada [29].

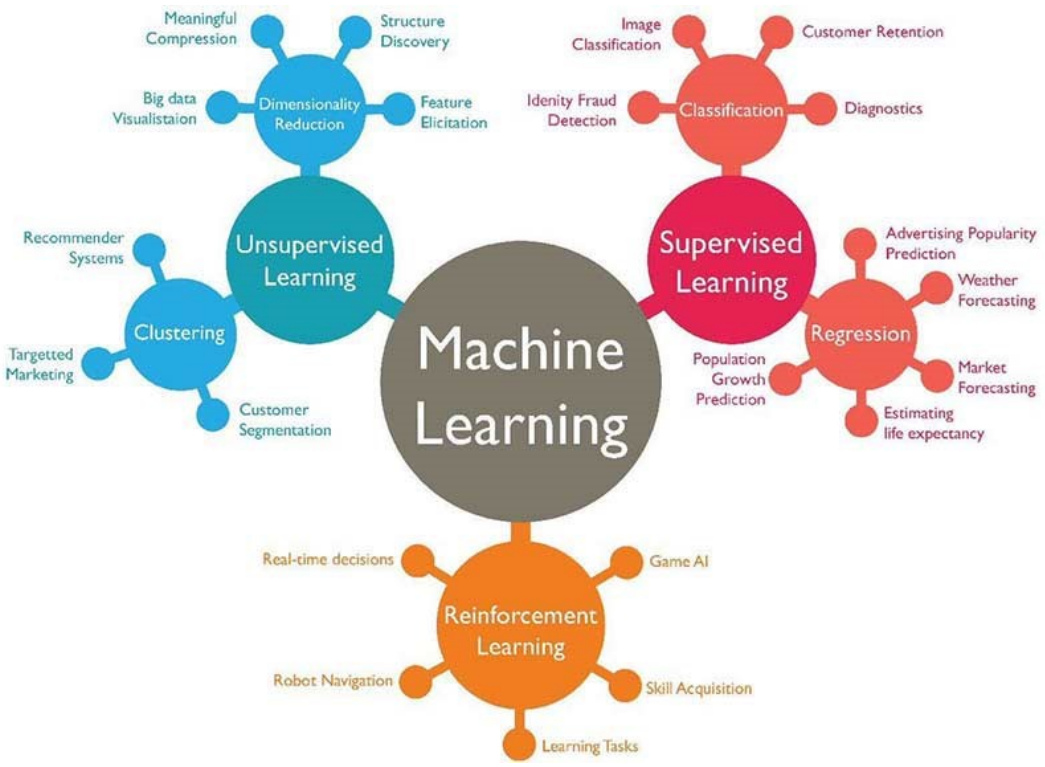



*Figura 2-7: Clasificación de los algoritmos de aprendizaje automático [30].*

Cuando se afronta un problema de **clasificación**, lo habitual es codificar la categoría de la salida en un vector con tantos elementos como clases, en el que se indique 1 en el elemento correspondiente a la clase a la que pertenece (se "activa" la clase correspondiente), y 0 en caso contrario. Esta estrategia se denomina ***one-hot vector***. Por ejemplo, si las clases son rojo, verde y azul, la codificación *one-hot* sería la siguiente:

| *Categoría* | *Código numérico* |
|---|---|
| ***Rojo*** | 1 |
| ***Verde*** | 2 |
| ***Azul*** | 3 |

| *Categoría* | *Codificación one-hot* | | |
|---|---|---|---|
| ***Rojo*** | 1 | 0 | 0 |
| ***Verde*** | 0 | 1 | 0 |
| ***Azul*** | 0 | 0 | 1 |

*Figura 2-8: Codificación one-hot para clasificación.*

### 2.2.2 Redes neuronales y aprendizaje profundo

Las redes neuronales son de este modo una subdivisión dentro del aprendizaje automático, un enfoque concreto para realizar aprendizaje supervisado. Son modelos que siguen centrándose en la capacidad del cerebro humano para aprender, pero que además intentan emular la manera en que funcionan las propias neuronas.

#### Modelo computacional neuronal

El elemento básico de una red neuronal es el **perceptrón**, el cual no es más que el modelo computacional de una neurona, llamado también **neurona artificial**. En la siguiente figura se muestra la estructura de ambas, y se pueden apreciar las similitudes entre ellas.

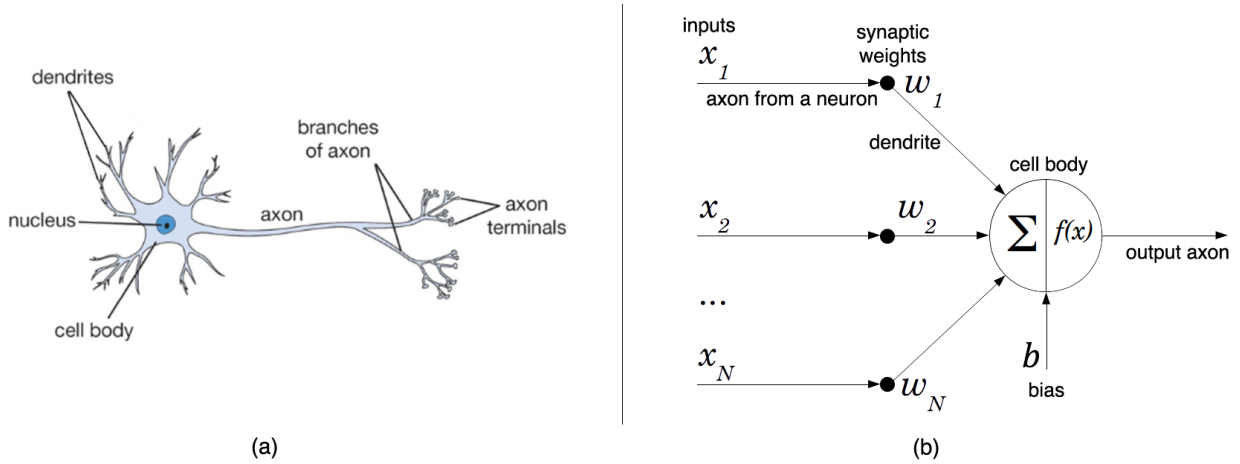



*Figura 2-9: Neuronas. (a) Estructura de una neurona biológica, (b) modelo computacional (perceptrón) [31].*

Cada neurona recibe impulsos por sus dendritas y genera otro impulso que sale a través de su axón, el cual se puede dividir múltiples veces y conectarse mediante sinapsis a las dendritas de otras neuronas. En el modelo computacional, las señales de entrada ($x_i$) interaccionan en forma de multiplicación por un peso (peso sináptico, $w_i$) con las dendritas, y la neurona se **activa** en función de la suma de sus entradas. De este modo, la salida de la neurona viene dada por la siguiente expresión [25] [31]:

$$y = f(\sum w_i x_i + b),$$

donde $f(x)$ es la llamada **función de activación**, la cual normalmente es la función escalón:

$$f(x) = \begin{cases} 1, & si \quad \sum_i w_i x_i + b > 0 \\ 0, & e.o.c. \end{cases}$$

El sesgo $b$ es, por tanto, el umbral que debe superar la suma de las entradas para que la neurona se active, y los pesos $w_i$ representan la fortaleza de la conexión sináptica entre neuronas. La idea fundamental del perceptrón es que los valores de los pesos y el sesgo se pueden **aprender**, y controlar de este modo la **influencia** que tienen unas neuronas sobre otras.

### Redes neuronales artificiales

El modelo del perceptrón es capaz de realizar clasificaciones binarias y lineales. Para realizar otros tipos de clasificación o regresión se generaliza el modelo en lo que se denominan redes neuronales artificiales (**ANNs**, por sus siglas en inglés).

A diferencia del modelo cerebral de interconexión, en el que las neuronas de conectan de manera aparentemente aleatoria y amorfa, las redes neuronales artificiales siguen una estructura más ordenada. En ellas, las neuronas se organizan en capas –de ahí que se denomine también **perceptrón multi-capa** (**MLP**)–, y suelen ser unidireccionales, es decir que sus enlaces no forman bucles [25].

Un perceptrón multicapa está formado por una capa de entrada, varias capas intermedias de perceptrones, y una capa de salida formada también por perceptrones. El tipo de capa más común es la **capa completamente conectada**, en la que cada neurona se conecta a todas las neuronas de la capa anterior, y no hay conexiones entre neuronas de la misma capa:

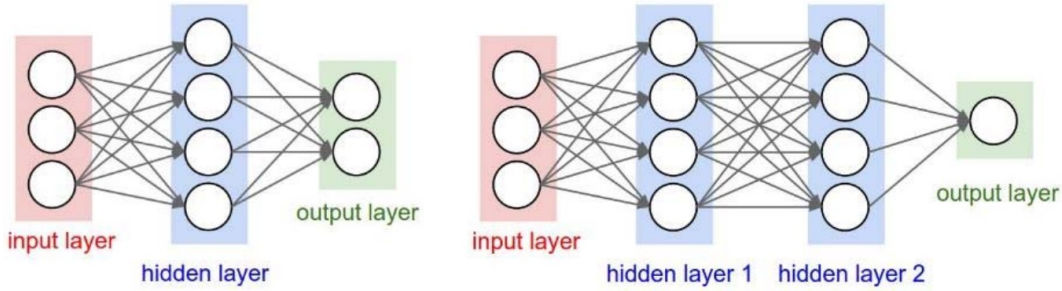
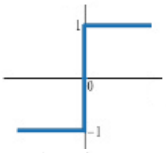
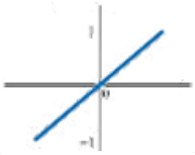
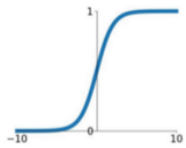
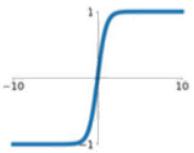
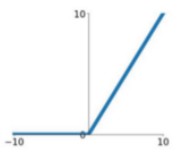
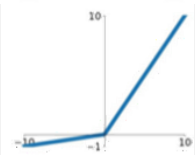



*Figura 2-10: Red neuronal artificial con capas completamente conectadas [31].*

En una ANN, siempre habrá una capa de entrada y una de salida, además de un número variable de capas intermedias –llamadas **capas ocultas**–, el cual dependerá del problema. Así, se puede considerar al perceptrón como un caso particular de red en la que se tiene una sola capa (la de salida) con una sola neurona. Por otro lado, cuando se habla de una red de *N* capas, dicho número se refiere al número de capas ocultas más la capa de salida, ya que la capa de entrada no está formada por neuronas.

Normalmente, todas las neuronas de una misma capa tendrán las mismas propiedades, es decir serán del mismo tipo y tendrán la misma función de activación. Además de las 2 funciones lineales ya comentadas para el perceptrón, se pueden utilizar otros tipos diferentes de funciones de activación. Las más comunes se recogen en la siguiente tabla:

*Tabla 2-1: Funciones de activación comunes en redes neuronales.*

| *Función* | *Ecuación* | *Gráfica* |
|---|---|---|
| ***Escalón*** | $f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$ |  |
| ***Signo*** | $f(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases}$ |  |
| ***Lineal*** | $f(x) = x$ |  |
| ***Logística (sigmoide)*** | $f(x) = \frac{1}{1 + e^{-x}}$ |  |
| ***Tangente hiperbólica*** | $f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$ |  |
| ***ReLU*** | $f(x) = \max(0, x)$ |  |
| ***Leaky ReLU*** | $f(x) = \max(0.01x, x)$ |  |

Las redes neuronales permiten implementar comportamientos altamente no lineales, y de hecho está demostrado que con una red de una sola capa oculta se puede aproximar cualquier función continua. Conforme se aumenta el número de capas o el número de neuronas por capa se incrementa la capacidad de representación de la red [31].

### Aprendizaje de redes neuronales

Al igual que para cualquier problema de aprendizaje automático, es necesario entrenar el modelo para ajustar los parámetros y que sea capaz de realizar las predicciones esperadas para el problema en cuestión. El algoritmo de aprendizaje más utilizado en redes neuronales es el de **retropropagación** (*backpropagation*) [25] [31].

A grandes rasgos, el método se basa en calcular la influencia que tiene cada uno de los parámetros de la red en el error final mediante el uso de derivadas parciales, y ajustar proporcionalmente los parámetros en la dirección de la derivada. Es decir, en cada iteración, se calcula el error total de la red, $\delta_T$, como la suma de los errores de cada salida:

$$\delta_T = \sum_k \delta_k = \sum_k (a_k - y_k), \qquad k = 1..núm.salidas$$

Donde $a_k$ es el valor real de la salida $k$, e $y_k$ es el valor predicho por la red. Con ello, se actualiza cada parámetro $w_i$ mediante la siguiente expresión:

$$w_i^+ = w_i - \eta \cdot \frac{\partial \varepsilon_T}{\partial w_i}$$

La derivada parcial del error respecto a cada parámetro se calcula haciendo uso de la regla de la cadena desde el error a la salida de la red hacia la entrada, de forma que es como si el error se "propagase hacia atrás". Es por ello que se suele decir que en una iteración se hace primero una pasada hacia delante (para realizar la predicción) y otra hacia atrás (para actualizar parámetros: retropropagación).

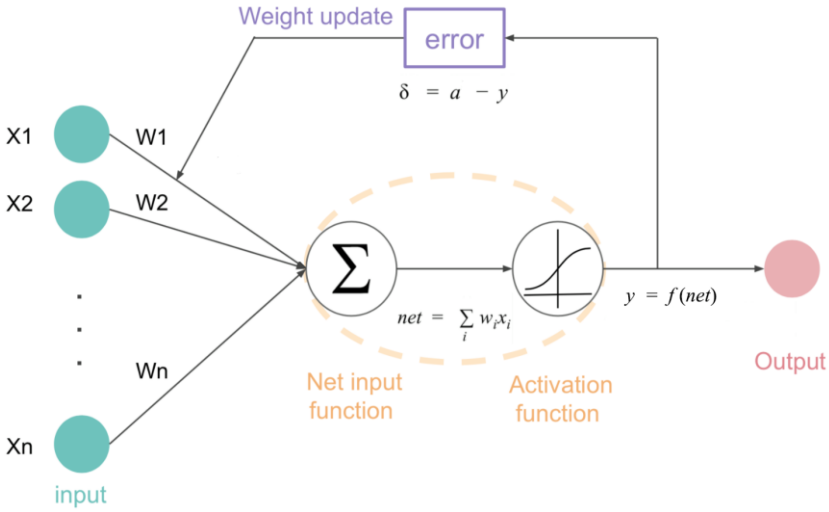



*Figura 2-11: Retropropagación en una red de 1 perceptrón [28].*

El parámetro $\eta$ se denomina **ritmo de aprendizaje** (*learning rate*), y especifica la velocidad a la que se lleva a cabo el aprendizaje. Es uno de los **hiperparámetros** más importantes en el aprendizaje de redes neuronales, y es en general difícil de ajustar, ya que si es muy alto los valores de los parámetros darán grandes saltos y será muy probable saltarse la solución óptima o no converger, y si es muy bajo puede tardar mucho en converger o quedarse atascado en mínimos locales.

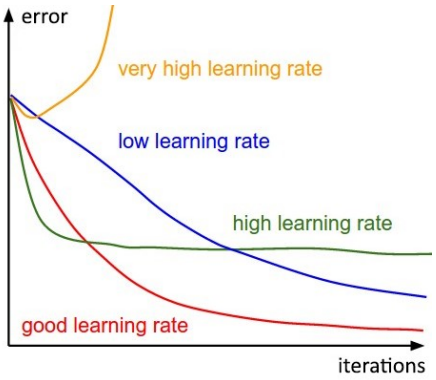



*Figura 2-12: Comportamiento de diferentes ritmos de aprendizaje respecto al tiempo [31].*

Otros hiperparámetros de las redes neuronales son el número de épocas (***epochs***) y el tamaño del lote (***batch size***) [31]. Una época es una pasada hacia delante y hacia atrás (predicción + retropropagación) de todo el conjunto de datos completo por la red neuronal. En general, cuantas más épocas se hagan mejor entrenada estará la red, sin embargo hay que tener cuidado de no sobreentrenarla:

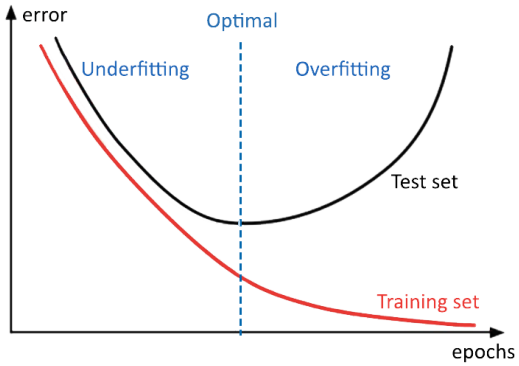



*Figura 2-13: Error frente al número de épocas [32].*

En cuanto al lote o *batch,* es el conjunto de muestras que se hacen pasar por la red en una iteración hacia adelante y hacia atrás, antes de actualizar los parámetros. Por ejemplo, si se tiene un set de datos de 2000 muestras, y se toma un tamaño de *batch* de 10, en cada época se iterará hacia delante y hacia atrás 200 veces.

### Redes neuronales convolucionales

Las redes neuronales artificiales son una solución de carácter general que se puede utilizar para casi cualquier tipo de problema, sin embargo hay aplicaciones en las que su estructura no es la más conveniente, como es el caso del procesamiento de imágenes. La gran dimensión de los datos de entrada en este caso hace que el número de parámetros crezca exponencialmente [25] [32].

Por ejemplo, para una imagen a color de 32x32, con 3 valores por píxel, se tendría un total de 32x32x3=3072 valores de entrada, que junto al sesgo hacen un total de 3073 parámetros por **una sola neurona** de la capa. Y eso siendo una imagen tremendamente pequeña.

Ante esta necesidad surgieron las redes neuronales convolucionales (**CNNs**), las cuales tienen en cuenta la disposición espacial de los datos de entrada. Así, en lugar de tener en cada capa neuronas completamente conectadas, las neuronas se disponen espacialmente del mismo modo que los datos de entrada.

Por ejemplo, si la entrada es la imagen de 32x32x3, la capa de entrada tendrá la siguiente forma:

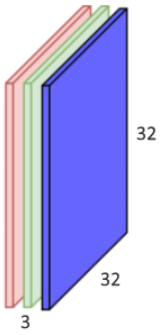



*Figura 2-14: Capa de entrada en una red convolucional [29].*

Cada neurona de una capa convolucional constituye un **filtro** que se aplica a una región concreta de los datos procedentes de la capa anterior, procesando todos sus canales y aplicando la **función de activación**, produciendo como resultado otro canal de salida:

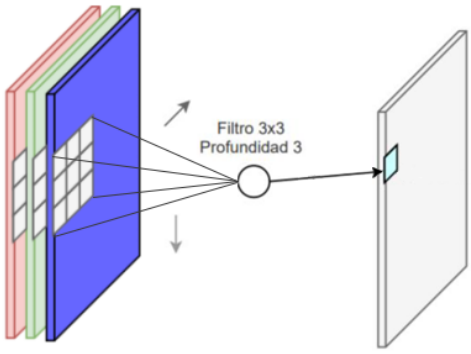



*Figura 2-15: Capa convolucional de un filtro [29].*

Un solo filtro o neurona recorre todos los píxeles de la imagen con los mismos pesos, como se indica con las flechas en la figura, produciendo una salida cada vez que se aplica, de forma que es como si el filtro se desplazase (convolucionase) sobre la imagen. El paso de desplazamiento (*stride*) va a condicionar el tamaño de la salida, la cual será de las veces que "quepa" el filtro en la imagen. Dado que suele ser de 1, si se quiere conservar a la salida el tamaño de la entrada, será necesario rellenar con ceros el borde de la imagen, lo que se denomina *zero-padding*.

En este ejemplo, el filtro es de tamaño 3x3x3, por lo que sólo habrá 27 pesos, a pesar de que la imagen es de 32x32x3. Como se puede observar se ha reducido considerablemente el número de parámetros.

Además, en una misma capa convolucional puede haber varios filtros que trabajen con los mismos datos de entrada, produciendo cada uno de ellos un canal de salida:

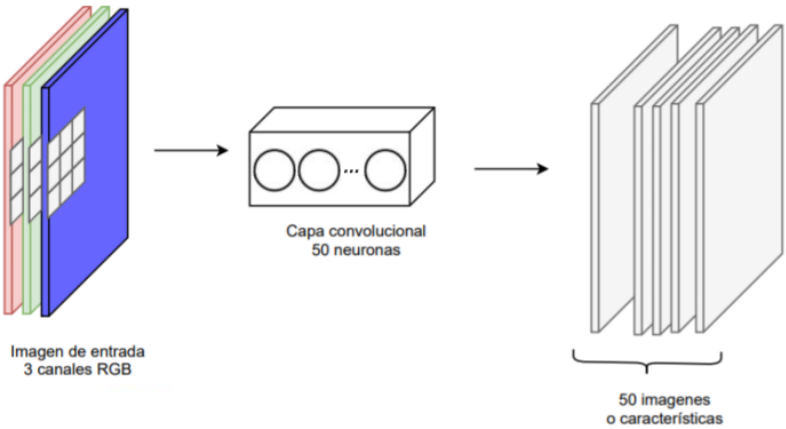



*Figura 2-16: Capa convolucional de 50 filtros [29].*

La salida de una capa convolucional se denomina **mapa de características**, y equivale a una imagen transformada. Además de capas convolucionales, las CNNs pueden contener también **capas de submuestreo**, para disminuir el tamaño de las salidas, y capas completamente conectadas, las cuales suelen situarse en el lado de la salida para tomar una decisión integrando la información extraída por las capas convolucionales y de submuestreo.

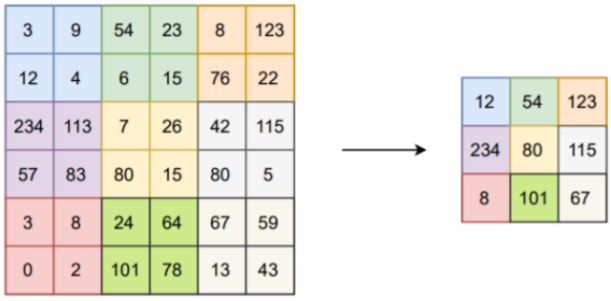



*Figura 2-17: Filtro de submuestreo "max-pooling" de 2x2 y paso 2 en un canal de entrada [29].*

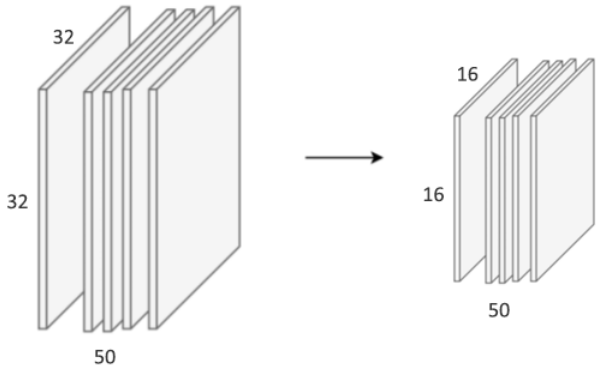



*Figura 2-18: Capa de submuestreo para reducción del tamaño a la mitad [29].*

La arquitectura de una red neuronal convolucional, incluyendo el número de capas y su orden, no está predefinido, aunque para problemas de clasificación suele ser habitual intercalar capas convolucionales con capas de submuestreo, y al final no más de 2 capas completamente conectadas.

Las redes neuronales convolucionales se utilizan para identificar objetos en fotos y vídeos, para reconocimiento facial, generación y mejora de imágenes, creación de efectos como cámara lenta y mejora de la calidad de las imágenes, etc. En definitiva, hoy en día las CNNs se utilizan en todos los casos que involucran imágenes y vídeos, enmarcado dentro del campo de la **visión artificial** (*computer vision*) [14] [31].

### Aprendizaje profundo

Las redes neuronales convolucionales ofrecen más prestaciones cuanto más número de capas se utilicen. Conforme se aumenta la profundidad de la red, se pueden resolver problemas más complejos que otros algoritmos no pueden afrontar. Las redes neuronales simples con cientos o incluso miles de neuronas, conectadas de una manera relativamente simple, no pueden duplicar lo que el cerebro humano puede hacer, lo cual resulta evidente si se tiene en cuenta que los cerebros humanos tienen alrededor de 86 billones de neuronas y una interconectividad muy compleja.

Es aquí donde surge el concepto de aprendizaje profundo o *deep learning*. A grandes rasgos, el aprendizaje profundo consiste en utilizar redes neuronales con más neuronas, capas e interconectividad. Como es lógico, todavía se está muy lejos de imitar el cerebro humano en toda su complejidad, pero se está avanzando en esa dirección [27].

En la siguiente imagen se muestra una arquitectura típica para un problema de clasificación de imágenes:

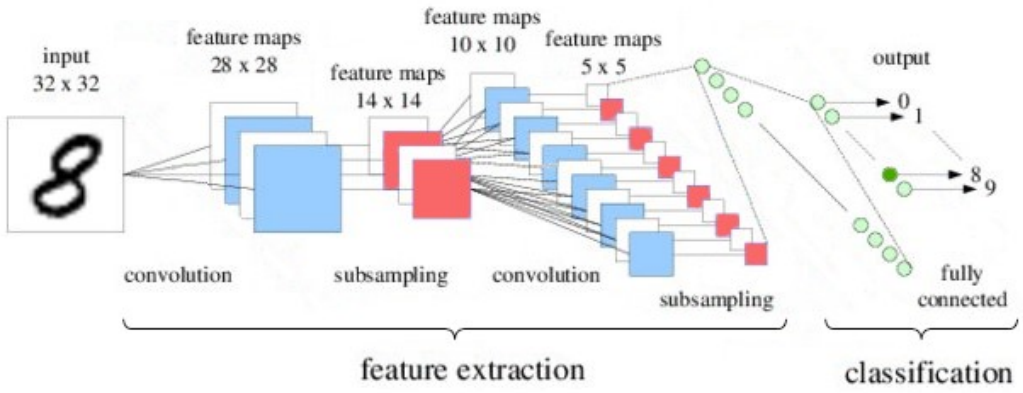



*Figura 2-19: Ejemplo de red neuronal profunda para clasificación [33].*

Los avances en computación, desde los vehículos autónomos, pasando por las supercomputadoras que juegan al *Go, y* hasta el reconocimiento de voz, utilizan por debajo de alguna forma el aprendizaje profundo. En definitiva, el aprendizaje profundo ha hecho posibles numerosas aplicaciones prácticas del aprendizaje automático y, por extensión, del campo general de la inteligencia artificial [26] [27].

### 2.2.3 Redes neuronales usadas en segmentación

Los problemas de segmentación tienen la particularidad de que no se trata de clasificar toda la imagen en una categoría, sino que se clasifica cada píxel de la misma. Es por ello que la salida de estas redes deberá tener el mismo tamaño que la entrada.

La mayoría de las arquitecturas propuestas para segmentación, están basadas en CNNs de clasificación ya conocidas, y que han supuesto contribuciones significativas al campo de la visión artificial. Estas redes tienen arquitecturas basadas en la CNN clásica representada en la Figura 2-19, la cual es la conocida y pionera **LetNet** de 1998 [34]. Concretamente destacan [35]:

- **AlexNet** [36]: Una CNN de la Universidad de Toronto pionera en la clasificación de imágenes, que ganó el *ImageNet Large Scale Visual Recognition Challenge* de 2012 con una precisión del 84,6%. Consta de 5 capas convolucionales, capas de submuestreo, activaciones con ReLUs, y 3 capas completamente conectadas. Supuso una revolución pues fue la primera solución basada en CNNs que ganó el *challenge* de ImageNet.
- **VGG-16** [37]: Este modelo de la Universidad de Oxford quedó segunda en el concurso ImageNet 2014 con una precisión del 92,7%. Utiliza una pila de capas convolucionales con pocos canales de salida en las primeras capas, en lugar de pocas capas con muchas salidas como propone la *AlexNet*.
- **GoogLeNet** [38]: Es la red que ganó el ImageNet de 2014 con una precisión del 93,3%, perteneciente a Google. Está compuesta por 22 capas y un nuevo bloque llamado módulo *inception,* por ello se la conoce también como *Inception* (a día de hoy hay 3 versiones). El módulo *inception* realiza diferentes transformaciones en los mismos datos de entrada (una convolución de 5×5, una de 3×3, una de 1x1 y un submuestreo de 3x3), vinculando los resultados en una sola salida. Con las convoluciones pequeñas se consiguió reducir drásticamente el número de parámetros, y de hecho a pesar de tener 22 capas de profundidad redujo el número de parámetros de 60 millones (*AlexNet*) a 4 millones.
- **ResNet** [39]: Un modelo de Microsoft que ganó el *challenge* de ImageNet en 2015 con una precisión del 96,4%. Es conocida por su profundidad (152 capas) y la introducción de bloques residuales. Los bloques residuales abordan el problema de la formación de una arquitectura realmente profunda sin incrementar demasiado la complejidad, permitiendo que las capas puedan copiar sus entradas a la siguiente capa. De esta forma, la red proporciona a las capas un punto de referencia desde el cual aprender, en lugar de comenzar desde cero.

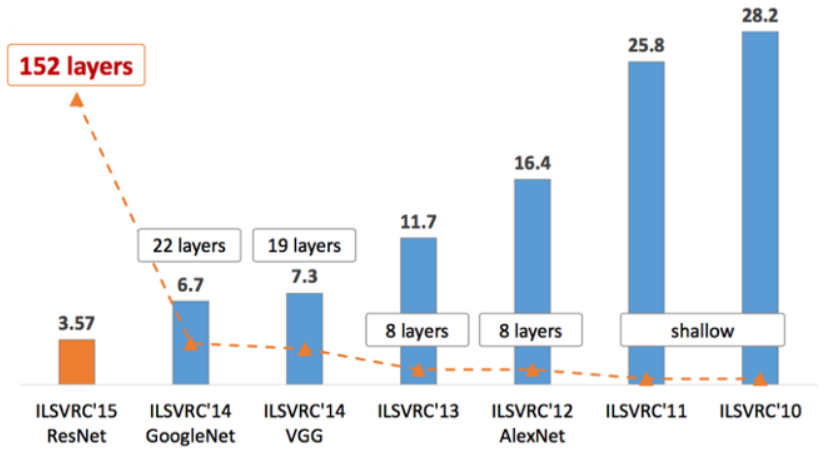



*Figura 2-20: Tasa de error (top-5) en el ImageNet Large Scale Visual Recognition Challenge [35].*

Para segmentación, la arquitectura de red neuronal que se utiliza es en general una red de **codificadores** seguida de una red de **decodificadores**. El codificador se encargará de extraer las características de la imagen, y la tarea del decodificador será la de recuperar los detalles de los objetos proyectando las características extraídas (baja resolución) en el espacio de píxeles, para obtener una clasificación de los mismos (alta resolución).

Para ello, se suele partir de una de las arquitecturas ya comentadas, y se reemplazan las capas completamente conectadas (las que realizan la clasificación) por capas que realicen una interpolación o *upsampling* con el objetivo de generar una salida con el mismo tamaño que la entrada. Este tipo de redes son por tanto **completamente convolucionales** (*Fully Convolutional Networks*, **FCN**).

Los primeros en implementar una arquitectura de este tipo fueron unos investigadores de la Universidad de Berkeley, en la que se tiene una única interpolación:

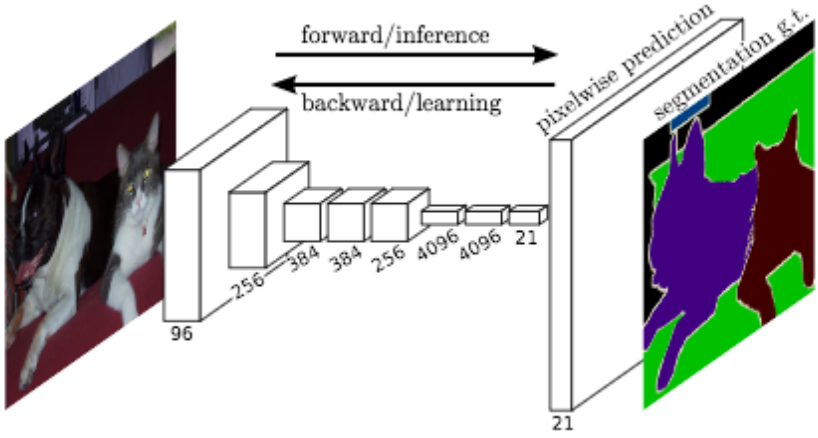



*Figura 2-21: Estructura de la FCN para segmentación [40].*

El problema de esta FCN básica es que las capas convolucionales y de submuestreo reducen la resolución de los mapas de características de salida, por lo que sus predicciones son típicamente de baja resolución, resultando en bordes de objetos relativamente borrosos. Como respuesta a ello, se han propuesto una variedad de enfoques basados en FCN pero más avanzados, que modifican de algún modo el camino de **decodificación** para mejorarlo y conseguir mayor precisión.

Entre ellas se incluyen ***SegNet*** [41], ***Dilated Convolutions*** [42]**,** ***DeepLabv1*** [43] y ***U-Net*** [1], todas ellas de 2015, y ***FPN*** [44]**,** ***PSPNet*** [45]**,** ***DeepLabv2*** [46] *y* ***RefineNet*** [47], de 2016.

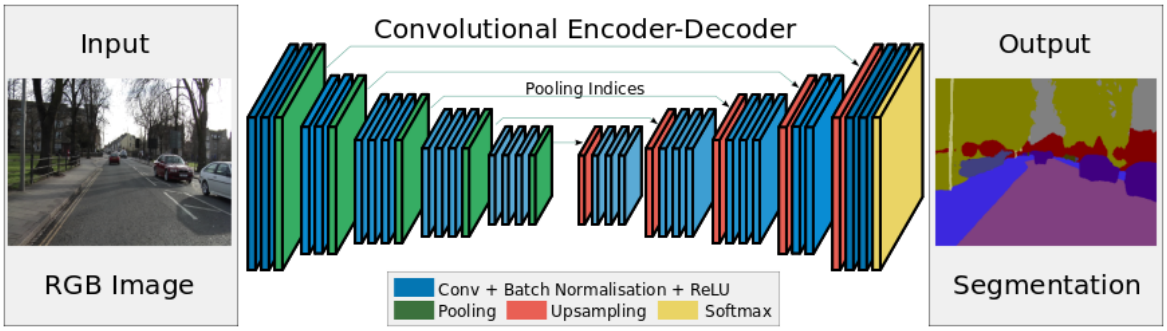



*Figura 2-22: Arquitectura de la SegNet [41].*

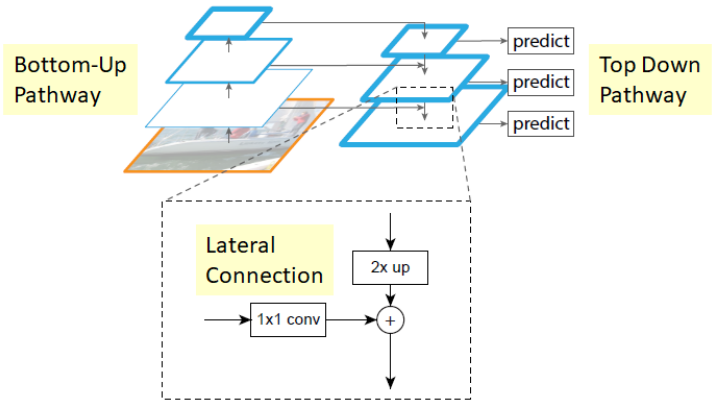



*Figura 2-23: Arquitectura de la FPN [44].*

### 2.2.4 Plataformas de aprendizaje automático

A día de hoy, *Python* lidera el ranking de lenguajes de programación utilizados en aprendizaje automático. Su aumento de popularidad en los últimos 3 años ha coincidido con el descenso de R, que aun así todavía le sigue de cerca. También se utiliza, aunque en mucha menor medida, Java y C/C++ [48].

De las 11 *frameworks* más utilizados, 10 de ellos funcionan con *Python.* Los más populares son *TensorFlow*, *Keras*, *PyTorch* y *Caffe*. Un estudio de uso, interés y popularidad [49] otorgó la siguiente puntuación a las plataformas más utilizados, basándose en las demandas de empleo, la encuesta de uso de *KDnuggets* [48], las publicaciones (*Medium, Amazon Books, ArXiv*) y la actividad en *GitHub*:

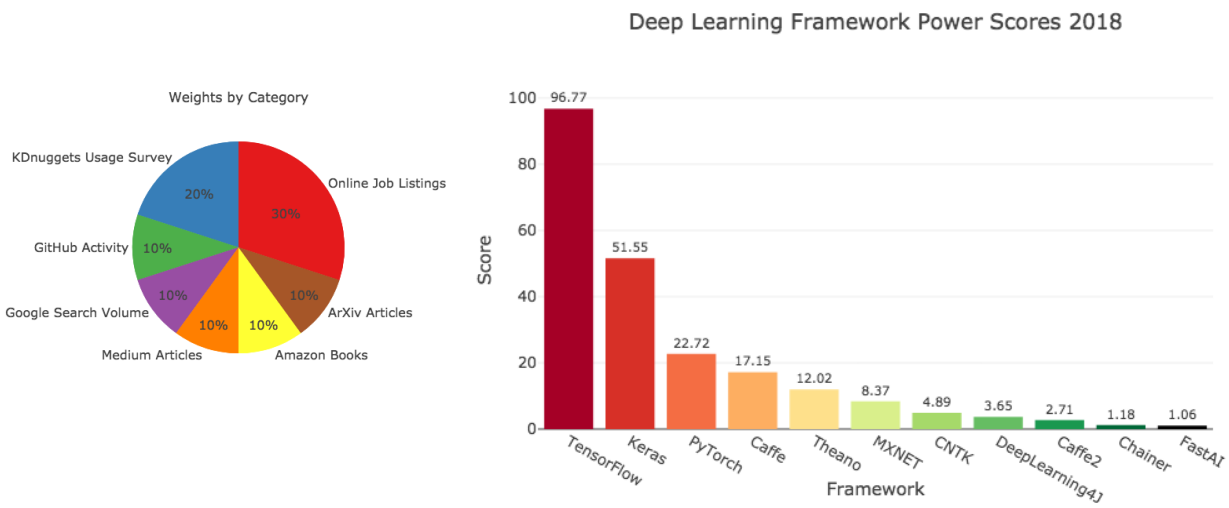



*Figura 2-24: Principales plataformas de deep learning [49].*

***Tensorflow***, respaldado por Google, y ***Pytorch***, con el respaldo de Facebook, son las dos plataformas independientes que mejor puntuación han obtenido, y ambas están disponibles para Python. Aunque *Tensorflow* es el ganador indiscutible, *Pytorch* está creciendo mucho en el último año.

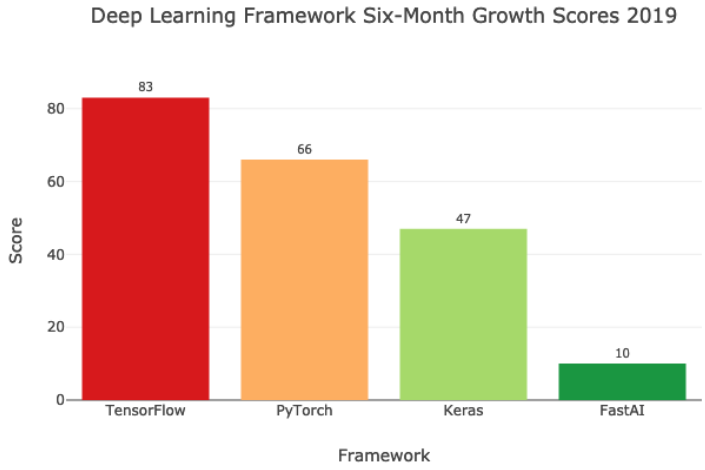



*Figura 2-25: Crecimiento de frameworks de deep learning en 2019 [50].*

Son dos plataformas con un alto nivel de personalización, lo cual permite implementar mejores modelos pero a su vez hace que la curva de aprendizaje sea más lenta. ***Keras*** y ***FastAI*** son APIs de alto nivel muy intuitivas, para *Tensorflow* y *Pytorch* respectivamente, y muy recomendadas si se está empezando en el mundo del *deep learning.*

En particular, *Keras* [51] queda en la segunda posición total, teniendo a día de hoy una gran base de usuarios, gran demanda en el mercado, y muchos artículos en *Medium.* De hecho, su popularidad está creciendo si se realiza un análisis con *Google Trends:*

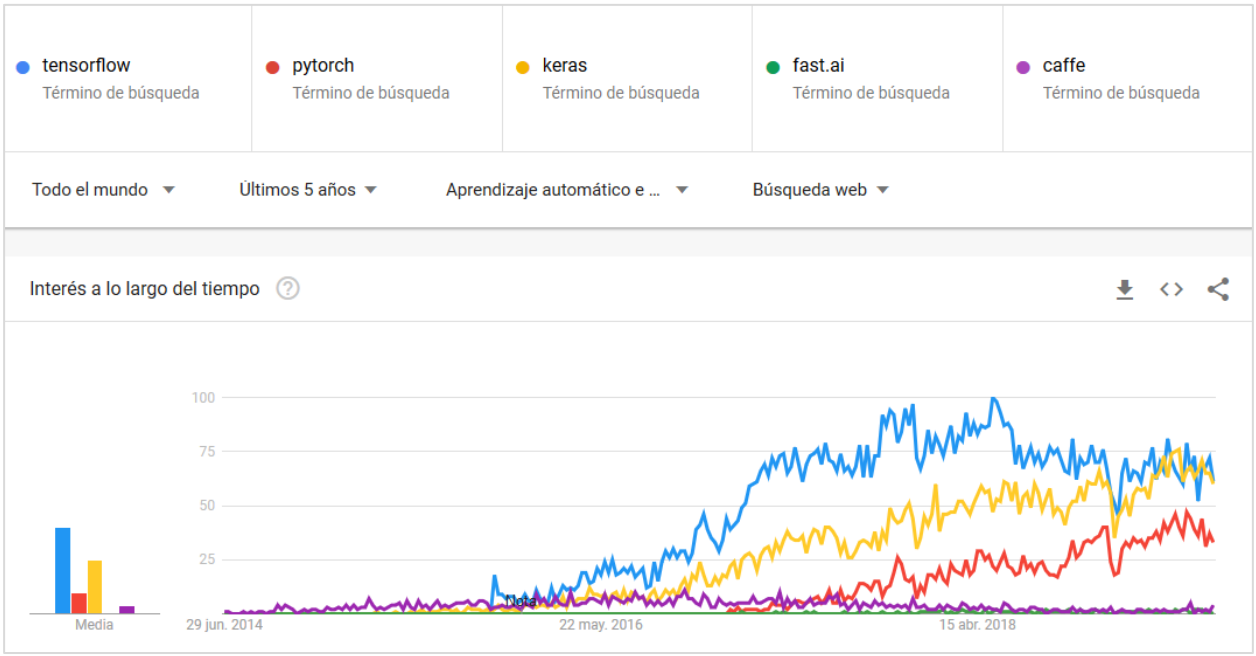



*Figura 2-26: Análisis de tendencias de las principales plataformas de aprendizaje profundo, a junio de 2019 [Fuente: Elaboración propia con Google Trends].*

Por ello, y por su API fácil de usar para principiantes, es el *framework* que se utilizará en este proyecto.

## 2.3 Método propuesto

Para este proyecto se utilizará la **U-Net** [1], una red de segmentación basada en la arquitectura codificador-decodificador de la FCN, diseñada para la segmentación de imágenes biomédicas. La idea fundamental que persigue es la de transferir la información espacial desde el codificador al decodificador, de forma parecida a lo que se proponía en la ResNet con los bloques residuales. Este tipo de conexiones, en las que se copian o se suman las salidas de unas capas a otras que no son inmediatamente posteriores, "saltándose" a capas intermedias, se denominan ***skip connections***. Con la introducción de las *skip connections* en las redes FCN lo que se busca es tratar de mejorar el detalle de la segmentación, y conseguir formas y bordes mucho más precisos.

### 2.3.1 U-Net

#### Arquitectura

Para implementar las *skip connections*, la arquitectura de la U-Net introduce simetría en la FCN mediante 2 estrategias:

i) Aumentando progresivamente el tamaño en el decodificador para que coincida con el del codificador. De esta forma, se puede transferir la información de las capas de bajada a las de subida realizando una **concatenación**, y mejorando así la resolución de la salida final.

ii) Manteniendo en el decodificador un gran número de canales de características. Esto permite a la red propagar la información de contexto a las capas de mayor resolución.

Esto queda reflejado en la Figura 2-27, en la que se muestra la arquitectura de la U-Net.

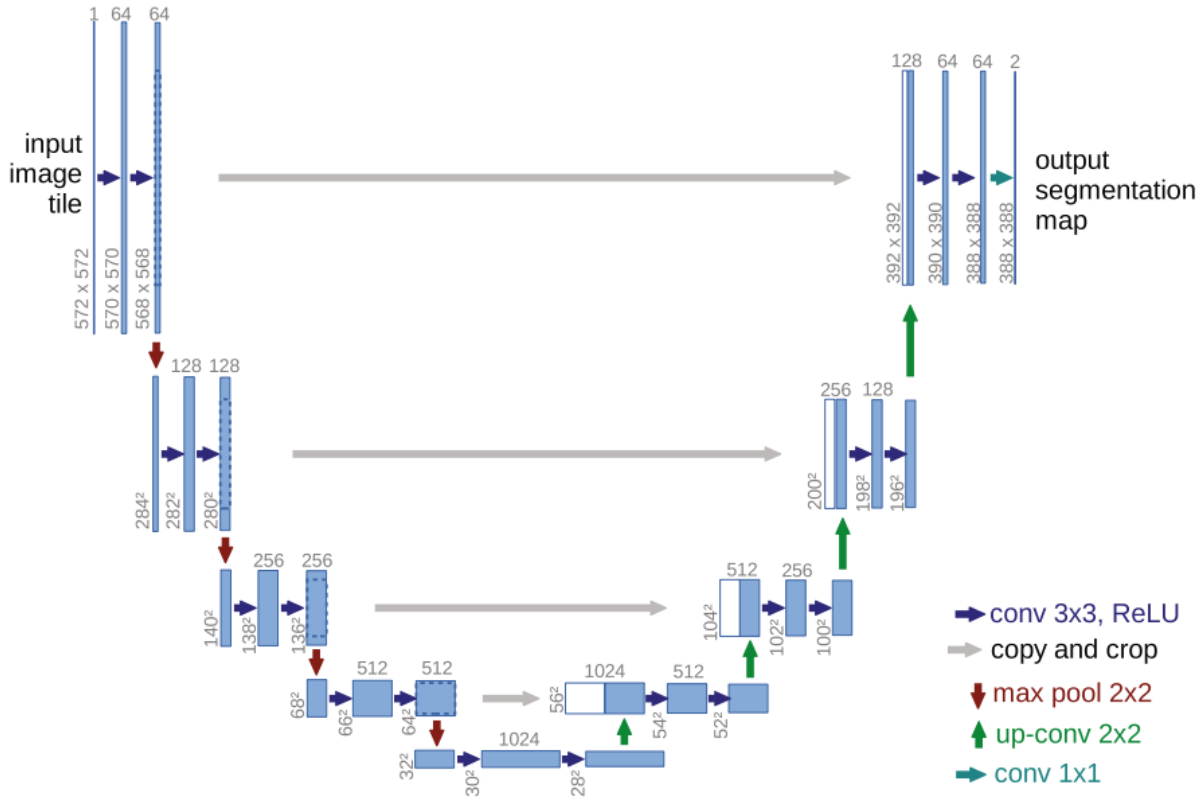



*Figura 2-27: Arquitectura de la U-Net [1].*

Se puede observar que la simetría de la arquitectura le da forma de U, razón por la cual se le da el nombre de U-Net. En ella se distinguen sus 2 partes: el codificador en el lado izquierdo y el decodificador a la derecha. Al codificador lo denominan **camino de contracción,** ya que se va reduciendo el tamaño debido a las capas de submuestreo, y al decodificador **camino de expansión**, en el que se realizan las interpolaciones que van aumentando progresivamente el tamaño de la imagen. Asimismo, se pueden distinguir en cada camino 5 **niveles**,

que vendrán diferenciados por el tamaño de sus salidas.

En total la red tiene 23 capas completamente convolucionales. La **capa de entrada** de la red es una capa dispuesta espacialmente como la imagen de entrada, la cual tiene un solo canal, es decir que será una imagen a escala de grises. A continuación, se encuentran en cada nivel del camino de contracción los siguientes elementos:

- **Convoluciones**: Se aplica, para cada nivel, 2 filtros convolucionales seguidos de tamaño 3x3 y con función de activación ReLU. Las convoluciones son sin *padding*, por lo que se perderán a la salida los píxeles de los bordes, es decir la imagen será 2 píxeles de altura y 2 píxeles de anchura más pequeña tras cada convolución. En el nivel superior las capas convolucionales son de 64 filtros, y cada vez que se desciende un nivel se dobla el número de filtros, es decir se dobla el número de canales del mapa de características.
- **Submuestreo**: Cada nivel termina con una capa de submuestreo de tamaño 2x2 con paso 2 cuyo efecto es la reducción del tamaño de los mapas de características a la mitad, descendiendo de esta forma al nivel inferior.

En cuanto al camino de expansión, está formado en cada nivel por:

- **Deconvolución** (*up-convolution* o *transposed convolution*): Para aumentar el tamaño de los datos, se propone el uso de la deconvolución o convolución traspuesta, la cual se implementa mediante una interpolación o *upsampling* seguida de una convolución 2x2. El *upsampling* dobla el tamaño del mapa de características, y la convolución reduce el número de canales a la mitad.
- **Concatenación**: Se implementan las *skip-connections* concatenando el resultado de la deconvolución con el mapa de características correspondiente de su nivel en el camino de contracción. De esta forma se alimenta a las sucesivas convoluciones con la información espacial necesaria. Ambas entradas de la concatenación deben tener el mismo tamaño, por lo que el mapa de características procedente del camino de contracción necesitará ser recortado acorde a las pérdidas de píxeles que se han sucedido debido a las convoluciones.
- **Convoluciones**: En cada nivel, 2 filtros convolucionales idénticos a los del camino de contracción. En este caso, cada vez que se sube un nivel se divide por la mitad el número de filtros de la capa convolucional.

Finalmente se tiene una **capa de salida** consistente en una convolución 1x1 que mapea los 64 canales del mapa de características resultantes al número de clases deseadas, utilizando para cada píxel una codificación ***one-hot***.

#### Entrenamiento

En el artículo en el que se propone la arquitectura de la *U-Net* [1], se establece como función de coste una combinación de la función ***softmax*** de cada píxel con respecto al número de clases, combinado con la función de **entropía cruzada** (*cross entropy*).

Siendo *x* cada píxel, e $y_k(x)$ la activación del píxel *x* para la clase *k*, la función *softmax* es la siguiente:

$$p_k(x) = \frac{e^{y_k(x)}}{\sum_j e^{y_j(x)}}$$

Con esto lo que se consigue es que $p_k \approx 1$ para la clase *k* que tenga la máxima activación (más cercana a 1), y $p_k \approx 0$ para las demás clases. Además, es una función de distribución de probabilidad para las clases, es decir, representa la probabilidad de que el píxel *x* pertenezca a la clase *k,* y la suma de los valores para todas las clases es 1.

$$\Pr[x = k] = p_k(x)$$

$$\sum_i p_i(x) = 1$$

A este valor se le aplica la función de entropía cruzada, la cual viene dada por:

$$J(x) = \sum_k a_k(x) \cdot \log(p_k(x))$$

Donde $a_k(x)$ es el valor real de activación del píxel *x* para la clase *k*. Dado que los valores reales, según la

codificación *one-hot,* sólo pueden ser 0 ó 1, el sumatorio para todas las clases únicamente contribuirá para la clase a la que pertenece el píxel. También se le denomina función de coste logarítmica (*log loss*), y su gráfica se muestra en la siguiente figura:

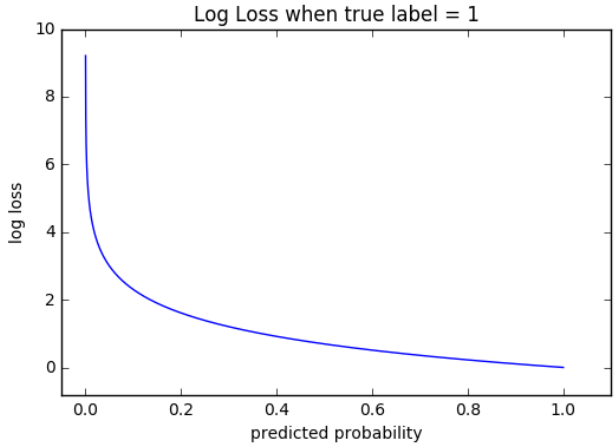



*Figura 2-28: Función de coste entropía cruzada cuando la etiqueta esperada es 1 [52].*

Por tanto, la función de coste resultante, teniendo en cuenta todos los píxeles, es la siguiente:

$$J = \sum_{x} \log(p_{true}(x))$$

Donde $p_{true}(x)$ es la predicción *softmax* correspondiente a la clase real a la que pertenece el píxel. Esta función lo que hace es penalizar en cada píxel la desviación del valor predicho para la clase real $p_{true}$ con 1, que es el valor de activación. En la práctica, esta arquitectura se implementa utilizando *softmax* como función de activación de la capa de salida, y la entropía cruzada como función de coste.

Adicionalmente, el artículo propone multiplicar la función de coste por unos pesos que dan más importancia a ciertos píxeles en el entrenamiento, y forzar a la red a aprender las pequeñas separaciones entre objetos contiguos en la imagen pertenecientes a la misma clase. Esto da lugar a una función de coste de **entropía cruzada ponderada**.

Finalmente, en redes profundas como ésta, con muchas capas convolucionales y diferentes caminos a través de la red, partes de la red pueden dar activaciones excesivas, mientras que otras partes nunca contribuyen. Para evitarlo, lo ideal es adaptar los pesos de manera que cada mapa de características de la red tenga aproximadamente **varianza unidad**. Es por ello que recomiendan inicializar los pesos de la red mediante una distribución Gaussiana con varianza 2/N, donde N es el número de canales a la entrada de la capa.

### 2.3.2 Variaciones introducidas

Una vez que ya se ha visto a fondo la arquitectura y el funcionamiento de la red, se comentarán las variaciones que se han hecho sobre la misma para este trabajo fin de máster, que tienen en cuenta las particularidades del problema a tratar y algunos avances que ha habido desde la publicación del artículo.

**i) Capas convolucionales**

Las capas convolucionales del modelo original son sin *padding*, lo que supone una pérdida de los píxeles del borde tras cada convolución. Uno de los requisitos del *challenge* de ISIC es que la salida debe tener el mismo tamaño que la entrada; por ello, se utilizarán capas convolucionales con 1 pixel de *padding*, de forma que tras la aplicación de la convolución de tamaño 3x3 con paso 1, **se conserve el tamaño**.

Esto tiene la ventaja adicional de que se simplifican las concatenaciones, dado que ya no será necesario recortar los mapas de características del camino de contracción para que coincidan en tamaño con el mapa correspondiente del camino de expansión.

**ii) Tamaño de la entrada**

La única restricción respecto al tamaño de las imágenes de entrada, es que tiene que ser tal que la entrada

a todas las operaciones de submuestreo sea par, para que no se desplace la información espacial al deshacer los submuestreos con las deconvoluciones.

Para ello hay que tener en cuenta los píxeles que se pierden en cada convolución y el número de veces que se reduce el tamaño a la mitad tras las capas de submuestreo. Dado que las convoluciones se han cambiado para que conserven el tamaño, únicamente hay que asegurarse de que la imagen es divisible por 2 las 4 veces que va a sufrir un submuestreo, es decir que sea **divisible por 16**.

Una posible estrategia para ello es, a la entrada de la red, rellenar los bordes de la imagen que no cumpla este criterio hasta el múltiplo de 16 más cercano, y recortarle estos píxeles a la salida. Esto sería lo ideal teniendo en cuenta el criterio del concurso de ISIC de que se mantenga el tamaño. Sin embargo, por simplicidad del modelo, y dado que este trabajo es con fines académicos, lo que se propone es redimensionar las imágenes de entrada y sus máscaras asociadas a un tamaño fijo antes de introducirlas en la red. Esto evita tener que modificar las imágenes a la entrada de la red y calcular el *padding* necesario, y además permite diseñar las capas de la red con un tamaño fijo en lugar de indefinido.

En concreto, y debido a las limitaciones de procesamiento, se redimensionará las imágenes a 128x128.

**iii) Función de coste**

Sobre la función de coste se realizarán 3 modificaciones. En primer lugar, no se utilizará la versión ponderada de la entropía cruzada ya que para el presente problema no habrá objetos contiguos de la misma clase, sino que hay 1 sola lesión por imagen.

Como se comentó, la aplicación de *softmax* se hace en la práctica en la activación de la capa de salida, y después se le aplica como función de coste la entropía cruzada.

Dado que únicamente hay 2 clases en este caso (lesión y fondo), se sabe por la función *softmax* que $p_{k=1}(x) = 1 - p_{k=0}(x)$, lo que hace innecesario utilizar una salida de 2 dimensiones. En lugar de ello, se utilizará una salida unidimensional cuya activación indica la presencia de lesión. La particularización de la función *softmax* para este caso es la función sigmoidal, lo cual es sencillo de deducir observando las expresiones de ambas. Por lo tanto, el mapa de características de salida será de **1 canal de activación sigmoidal.**

Finalmente, también se puede considerar una particularización de la función de coste para 2 clases, que es la **función de entropía cruzada binaria**.

***iv) Batch Normalization***

La normalización por lotes, o *batch normalization*, es una técnica introducida por los desarrolladores de Google Ioffe y Szegedy en 2015 [53] para acelerar el entrenamiento de redes muy profundas con activaciones no lineales. El problema que intentan resolver es que la distribución de los datos de entrada a cada capa cambia durante el entrenamiento, debido a que cambian los **parámetros de las capas anteriores**, lo que obliga a la capa en cuestión a ajustarse a diferentes distribuciones constantemente y ralentiza el entrenamiento. Para evitarlo, proponen normalizar la entrada a la capa, para de este modo reducir la cantidad en la que desplazan los datos de una iteración a otra.

En el artículo de la U-Net se recomienda que la varianza de los mapas de características sea idealmente aproximadamente unitaria, para evitar que haya activaciones que alcancen niveles muy altos o muy bajos. El *batch normalization* realiza justo eso, razón por la cual se utilizará en el modelo, detrás de cada capa de activación ReLU.

En general, esta técnica va a permitir que cada capa de la red aprenda por sí misma un poco más independientemente de las otras capas, y además va a reducir la dependencia que se tiene de la inicialización de pesos, la cual tiene una gran influencia en el rendimiento del entrenamiento. [54] [55]

### Aprendizaje transferido (*Transfer learning*)

El *transfer learning* es un método de aprendizaje automático en el que un modelo desarrollado para una tarea se reutiliza como punto de partida para un modelo en una segunda tarea. Es muy utilizado en aprendizaje profundo debido a la gran cantidad de tiempo, capacidad de procesamiento etiquetados necesarios para entrenar completamente una red neuronal profunda, la cual puede tardar días e incluso semanas [56].

Por ello, es muy común partir de modelos preentrenados en una amplia base de datos, que ya han aprendido a realizar una tarea diferente pero relacionada con el problema a tratar.

Se pueden distinguir 3 estrategias [31]:

- CNN preentrenada como un extractor de características fijo. Para ello, se parte de una de las CNN de clasificación comentadas en la sección 2.2.3, se eliminan las últimas capas completamente conectadas que realizan la clasificación, y se trata el resto de la red como un extractor de características fijo para el nuevo conjunto de datos. Una vez extraídas las características, éstas pueden utilizarse para entrenar un clasificador o un decodificador con el nuevo conjunto de datos. Este proceso tenderá a funcionar si las características son generales, es decir, adecuadas tanto para las tareas de base como para las de destino, en lugar de específicas para la tarea original.
- Ajuste fino de una CNN preentrenada (*fine tuning*). Consiste no sólo en reemplazar el clasificador de la CNN y entrenarlo con el nuevo conjunto de datos, sino también en ajustar con precisión los pesos de la red completa, incluyendo la parte previamente entrenada, continuando la retropropagación. Se pueden afinar todas las capas de la CNN o mantener fijas algunas de las primeras capas (**niveles bajos**) y sólo afinar algunas de la parte final (**niveles superiores**). Esto está motivado por la observación de que los primeros mapas de características contendrán atributos más genéricos (por ejemplo, detectores de bordes o detectores de color) que pueden ser útiles para muchas tareas, y las capas posteriores se vuelven más específicas para los detalles del conjunto de datos original.
- Modelos preentrenados. En este caso se utiliza un modelo completo que ya está entrenado, y se ajusta más finamente con el nuevo conjunto de datos. Es muy común en la comunidad publicar modelos en un punto de control que funciona correctamente, para el beneficio de otros que pueden utilizar las redes para realizar ajustes.

La estrategia a elegir dependerá de la capacidad de procesamiento de que se disponga, de la similitud entre el conjunto de datos original y el del problema a tratar, y del tamaño del conjunto de datos del nuevo problema. Evidentemente también hay que tener en cuenta la disponibilidad o no de dicho modelo preentrenado en el lenguaje o paquete que se quiera utilizar [31] [56].

Para el caso de este proyecto, las capacidades de procesamiento que se tienen son limitadas, por lo que lo ideal sería obtener un modelo de la *U-Net* preentrenado y ajustarlo más finamente al nuevo set de datos. Sin embargo, no se ha encontrado disponible un modelo de la misma para *Keras*, por lo que lo que se hará es construir una arquitectura alternativa de la U-Net basada en la CNN de clasificación de imágenes ***VGG-16*** [37] como proponen Iglovikov y Shvets en [57]. La arquitectura de la VGG-16 es la siguiente:

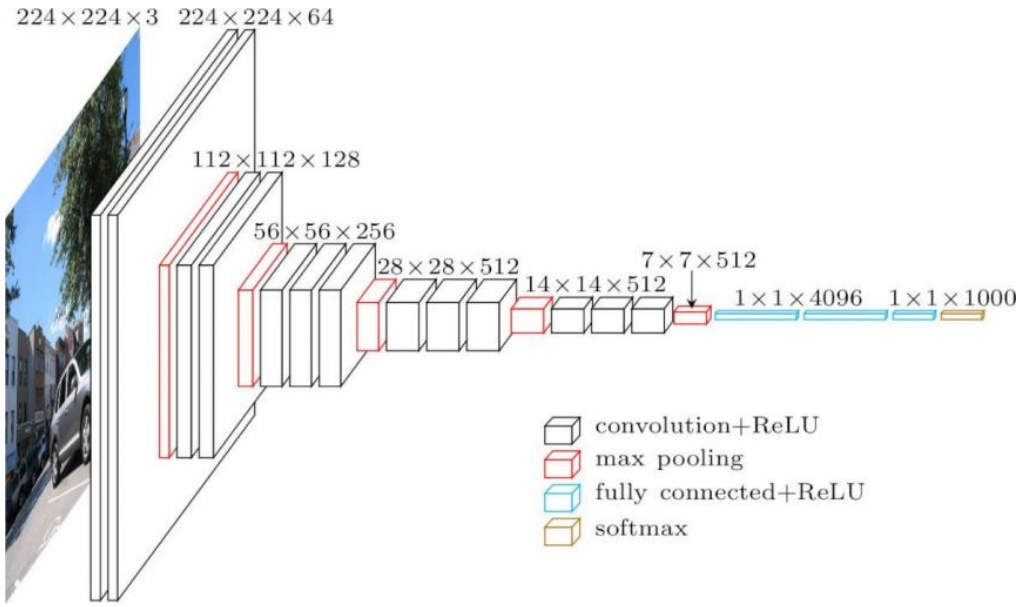



*Figura 2-29: Arquitectura de la VGG-16 [58].*

Como se puede observar, obviando las capas completamente conectadas, la estructura de las capas convolucionales es prácticamente igual al camino de contracción de la U-Net, salvo algunas diferencias:

i) La entrada es una imagen RGB, la cual tiene 3 canales.

ii) En algunos bloques hay 3 capas convolucionales en lugar de 2.

iii) Hay un submuestreo más que en la *U-Net*.

iv) En el último bloque los mapas de características son de 512 canales, frente a los 1024 que se utilizan en la *U-Net*.

No son diferencias muy notables, con lo que sin problema se puede construir una red que como codificador utilice el camino convolucional de la *VGG-16* –menos la última capa de submuestreo– y construir simétricamente un decodificador, siendo esencialmente una U-Net.

El motivo de elegir la *VGG-16*, además de por la evidente similitud en la arquitectura, es que esta red, junto con las otras CNNs mencionadas (*AlexNet*, *Inception*, *ResNet*, …) son redes muy comunes de clasificación de imágenes, y sus pesos preentrenados con la base de datos *ImageNet* están disponibles en múltiples plataformas.

#### Arquitectura final del modelo propuesto

Finalmente, tras todas las modificaciones comentadas, la arquitectura final de la red se muestra en la siguiente figura:

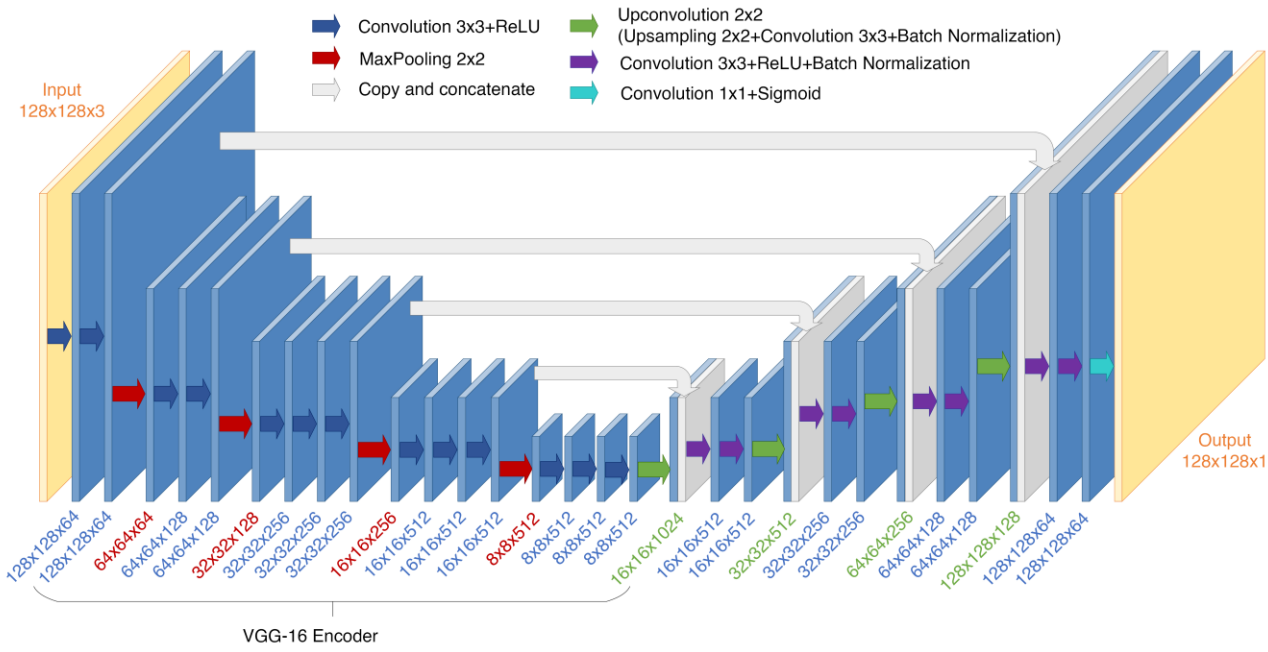



*Figura 2-30: Arquitectura del modelo propuesto [Fuente: elaboración propia].*

### 2.3.3 Implementación

#### Materiales

El lenguaje elegido para el desarrollo de este proyecto es *Python,* concretamente se utilizará *Keras* encima de *TensorFlow.* Para entrenar redes neuronales es necesaria una gran potencia de procesamiento, y en particular cuando son para procesamiento de imágenes se considera indispensable hoy en día el uso de unidades de procesamiento gráfico (GPUs), las cuales han aumentado 10 veces o más la velocidad de procesador que se tenía con las CPU.

En un esfuerzo por reducir la barrera de entrada al aprendizaje automático, Google ha puesto disponible de forma gratuita su plataforma ***Google Colaboratory*** [59]. Ésta permite programar modelos de *machine learning* en la nube, con *Python*, con la posibilidad de añadir el poder de procesamiento de una GPU NVIDIA Tesla T4 [60],

una GPU especialmente diseñada para plataformas de *deep learning*.

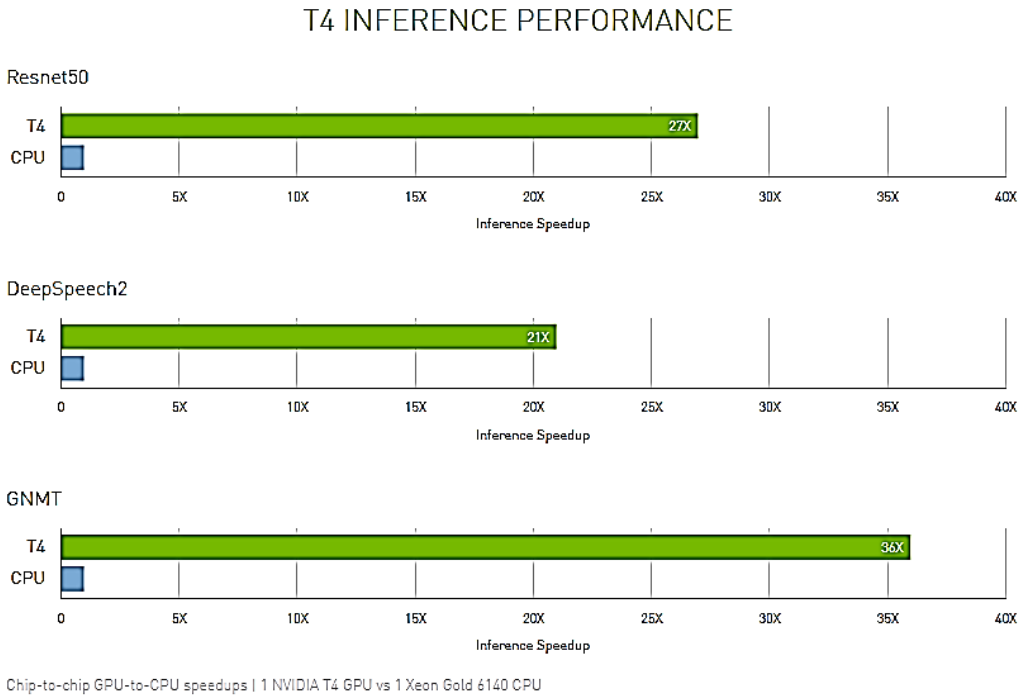



*Figura 2-31: Rendimiento de la GPU NVIDIA Tesla T4 [60].*

Para el uso de *Google Colab* se accede a través de cualquier navegador, y ofrece, entre otras cosas:

- Posibilidad de activar una GPU, y utilizarla durante 12 horas cada vez
- Utilizar *Python* 2 ó 3
- El uso de *Jupyter Notebooks*, un enfoque para programar en Python basado en web
- Tiene preinstaladas las librerías comunes usadas en aprendizaje automático, y ofrece la posibilidad de instalar otras que se necesiten
- Enlazar con la cuenta de *Google Drive* para poder leer desde ahí los datos de entrada y guardar las salidas deseadas.

Todo ello resulta muy conveniente para entornos académicos, o desarrolladores que están empezando y necesitan probar sus modelos de una forma eficiente pero no tienen acceso a una GPU, como es el caso de este Trabajo Fin de Máster.

En la siguiente tabla se recogen los detalles del hardware utilizado en *Google Colab* para este proyecto:

| *Parámetro* | *Valor* |
|---|---|
| ***GPU*** | NVIDIA Tesla T4 |
| ***GPU RAM*** | 15.8 GB |
| ***CPU*** | Intel® Xeon® CPU |
| ***CPU RAM*** | 12.72 GB |
| ***Velocidad CPU*** | 2.30 GHz |
| ***Núcleos CPU*** | 2 |

| | |
|---|---|
| ***Duración máxima de sesión*** | 12 h |
| ***Tiempo ocioso máximo*** | 1.5 h |

*Tabla 2-2: Especificaciones hardware (Google Colab).*

### Modelo

A continuación se incluye el código *Python* de implementación del modelo, el cual está ligeramente basado en el modelo *VGGUnet* publicado por Jibin Mathew en *GitHub* [61]:

```
from keras.models import *
from keras.layers import *

def Unet(VGG_Weights_path, freeze_vgg=True, input_height=128, input_width=128,
         image_ordering="channels_last"):

    # Input
    assert input_height%16 == 0
    assert input_width%16 == 0
    if image_ordering=="channels_first":
        input_shape = (3,input_height,input_width)
        channel_axis = 1
    else :
        input_shape = (input_height,input_width,3)
        channel_axis = 3
    img_input = Input(shape=input_shape)



    # ##################################### Encoder ########################################
    # Block 1
    x = Conv2D(64, (3,3), activation='relu', padding='same', name='block1_conv1',
               data_format=image_ordering)(img_input)
    x = Conv2D(64, (3,3), activation='relu', padding='same', name='block1_conv2',
               data_format=image_ordering)(x)
    f1 = x
    x = MaxPooling2D((2,2), strides=(2,2), name='block1_pool',data_format=image_ordering)(x)

    # Block 2
    x = Conv2D(128, (3,3), activation='relu', padding='same', name='block2_conv1',
               data_format=image_ordering)(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same', name='block2_conv2',
               data_format=image_ordering)(x)
    f2 = x
    x = MaxPooling2D((2,2), strides=(2,2), name='block2_pool',data_format=image_ordering)(x)

    # Block 3
    x = Conv2D(256, (3,3), activation='relu', padding='same', name='block3_conv1',
               data_format=image_ordering)(x)
    x = Conv2D(256, (3,3), activation='relu', padding='same', name='block3_conv2',
               data_format=image_ordering)(x)
    x = Conv2D(256, (3,3), activation='relu', padding='same', name='block3_conv3',
               data_format=image_ordering)(x)
    f3 = x
    x = MaxPooling2D((2,2), strides=(2,2), name='block3_pool',data_format=image_ordering)(x)
```

```
# Block 4
x = Conv2D(512, (3,3), activation='relu', padding='same', name='block4_conv1',
           data_format=image_ordering)(x)
x = Conv2D(512, (3,3), activation='relu', padding='same', name='block4_conv2',
           data_format=image_ordering)(x)
x = Conv2D(512, (3,3), activation='relu', padding='same', name='block4_conv3',
           data_format=image_ordering)(x)
f4 = x
x = MaxPooling2D((2,2), strides=(2,2), name='block4_pool',data_format=image_ordering)(x)

# Block 5
x = Conv2D(512, (3,3), activation='relu', padding='same', name='block5_conv1',
           data_format=image_ordering)(x)
x = Conv2D(512, (3,3), activation='relu', padding='same', name='block5_conv2',
           data_format=image_ordering)(x)
x = Conv2D(512, (3,3), activation='relu', padding='same', name='block5_conv3',
           data_format=image_ordering)(x)
f5 = x
x = MaxPooling2D((2,2), strides=(2,2), data_format=image_ordering)(x)


# Create VGG encoder model and freeze weights
vgg  = Model(img_input, x)
vgg.load_weights(VGG_Weights_path)
if freeze_vgg:
    for layer in vgg.layers:
        layer.trainable = False


# ######################################## Decoder ########################################
# Block 6
o = f5
o = UpSampling2D(name='block6_upsamp', size=(2,2))(o)
o = Conv2D(512, (2,2), activation='relu', padding='same', name='block6_upconv',
           data_format=image_ordering, kernel_initializer='he_normal')(o)
o = BatchNormalization(axis=channel_axis, name='block6_norm1')(o)
o = concatenate([o, f4], axis=channel_axis)
o = Conv2D(512, (3,3), activation='relu', padding='same', name='block6_conv1',
           data_format=image_ordering, kernel_initializer='he_normal')(o)
o = BatchNormalization(axis=channel_axis, name='block6_norm2')(o)
o = Conv2D(512, (3,3), activation='relu', padding='same', name='block6_conv2',
           data_format=image_ordering, kernel_initializer='he_normal')(o)
o = BatchNormalization(axis=channel_axis, name='block6_norm3')(o)

# Block 7
o = UpSampling2D(name='block7_upsamp', size=(2,2))(o)
o = Conv2D(256, (2,2), activation='relu', padding='same', name='block7_upconv',
           data_format=image_ordering, kernel_initializer='he_normal')(o)
o = BatchNormalization(axis=channel_axis, name='block7_norm1')(o)
o = concatenate([o, f3], axis=channel_axis)
o = Conv2D(256, (3,3), activation='relu', padding='same', name='block7_conv1',
           data_format=image_ordering, kernel_initializer='he_normal')(o)
o = BatchNormalization(axis=channel_axis, name='block7_norm2')(o)
o = Conv2D(256, (3,3), activation='relu', padding='same', name='block7_conv2',
           data_format=image_ordering, kernel_initializer='he_normal')(o)
o = BatchNormalization(axis=channel_axis, name='block7_norm3')(o)
```

```
    # Block 8
    o = UpSampling2D(name='block8_upsamp', size=(2,2))(o)
    o = Conv2D(128, (2,2), activation='relu', padding='same', name='block8_upconv',
               data_format=image_ordering, kernel_initializer='he_normal')(o)
    o = BatchNormalization(axis=channel_axis, name='block8_norm1')(o)
    o = concatenate([o, f2], axis=channel_axis)
    o = Conv2D(128, (3,3), activation='relu', padding='same', name='block8_conv1',
               data_format=image_ordering , kernel_initializer='he_normal')(o)
    o = BatchNormalization(axis=channel_axis, name='block8_norm2')(o)
    o = Conv2D(128, (3,3), activation='relu', padding='same', name='block8_conv2',
               data_format=image_ordering , kernel_initializer='he_normal')(o)
    o = BatchNormalization(axis=channel_axis, name='block8_norm3')(o)

    # Block 9
    o = UpSampling2D(name='block9_upsamp', size=(2,2))(o)
    o = Conv2D(64, (2,2), activation='relu', padding='same', name='block9_upconv',
               data_format=image_ordering, kernel_initializer='he_normal')(o)
    o = BatchNormalization(axis=channel_axis, name='block9_norm1')(o)
    o = concatenate([o, f1], axis=channel_axis)
    o = Conv2D(64, (3,3), activation='relu', padding='same', name='block9_conv1',
               data_format=image_ordering, kernel_initializer='he_normal')(o)
    o = BatchNormalization(axis=channel_axis, name='block9_norm2')(o)
    o = Conv2D(64, (3,3), activation='relu', padding='same', name='block9_conv2',
               data_format=image_ordering, kernel_initializer='he_normal')(o)
    o = BatchNormalization(axis=channel_axis, name='block9_norm3')(o)

    o = Conv2D(1, (1,1), activation='sigmoid', padding='same', name='output',
               data_format=image_ordering)(o)


    model = Model(inputs=img_input, outputs=o)

    return model
```

Como se puede observar, la definición del modelo es muy inmediata; partiendo de la arquitectura se pueden ir definiendo las capas una detrás de otra con sus tipos y parámetros correspondientes.

Para cerciorarse de cómo es el modelo resultante, se puede ejecutar la función *summary()* sobre el mismo, en cuyo resultado se pueden ver las capas definidas y las conexiones entre ellas, así como el número de parámetros del modelo:

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input (InputLayer)           (None, 128, 128, 3)       0
_________________________________________________________________________________________________
block1_conv1 (Conv2D)        (None, 128, 128, 64)      1792        input_2[0][0]
_________________________________________________________________________________________________
block1_conv2 (Conv2D)        (None, 128, 128, 64)      36928       block1_conv1[0][0]
_________________________________________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0           block1_conv2[0][0]
_________________________________________________________________________________________________
block2_conv1 (Conv2D)        (None, 64, 64, 128)       73856       block1_pool[0][0]
_________________________________________________________________________________________________
block2_conv2 (Conv2D)        (None, 64, 64, 128)       147584      block2_conv1[0][0]
_________________________________________________________________________________________________
block2_pool (MaxPooling2D)   (None, 32, 32, 128)       0           block2_conv2[0][0]
_________________________________________________________________________________________________
block3_conv1 (Conv2D)        (None, 32, 32, 256)       295168      block2_pool[0][0]
_________________________________________________________________________________________________
block3_conv2 (Conv2D)        (None, 32, 32, 256)       590080      block3_conv1[0][0]
_________________________________________________________________________________________________
block3_conv3 (Conv2D)        (None, 32, 32, 256)       590080      block3_conv2[0][0]
```

```
block3_pool (MaxPooling2D)        (None, 16, 16, 256)    0          block3_conv3[0][0]
block4_conv1 (Conv2D)             (None, 16, 16, 512)    1180160    block3_pool[0][0]
block4_conv2 (Conv2D)             (None, 16, 16, 512)    2359808    block4_conv1[0][0]
block4_conv3 (Conv2D)             (None, 16, 16, 512)    2359808    block4_conv2[0][0]
block4_pool (MaxPooling2D)        (None, 8, 8, 512)      0          block4_conv3[0][0]
block5_conv1 (Conv2D)             (None, 8, 8, 512)      2359808    block4_pool[0][0]
block5_conv2 (Conv2D)             (None, 8, 8, 512)      2359808    block5_conv1[0][0]
block5_conv3 (Conv2D)             (None, 8, 8, 512)      2359808    block5_conv2[0][0]
block6_upsamp (UpSampling2D)      (None, 16, 16, 512)    0          block5_conv3[0][0]
block6_upconv (Conv2D)            (None, 16, 16, 512)    1049088    block6_upsamp[0][0]
block6_norm1 (BatchNormalization) (None, 16, 16, 512)    2048       block6_upconv[0][0]
concatenate_5 (Concatenate)       (None, 16, 16, 1024)   0          block6_norm1[0][0]
                                                                    block4_conv3[0][0]
block6_conv1 (Conv2D)             (None, 16, 16, 512)    4719104    concatenate_5[0][0]
block6_norm2 (BatchNormalization) (None, 16, 16, 512)    2048       block6_conv1[0][0]
block6_conv2 (Conv2D)             (None, 16, 16, 512)    2359808    block6_norm2[0][0]
block6_norm3 (BatchNormalization) (None, 16, 16, 512)    2048       block6_conv2[0][0]
block7_upsamp (UpSampling2D)      (None, 32, 32, 512)    0          block6_norm3[0][0]
block7_upconv (Conv2D)            (None, 32, 32, 256)    524544     block7_upsamp[0][0]
block7_norm1 (BatchNormalization) (None, 32, 32, 256)    1024       block7_upconv[0][0]
concatenate_6 (Concatenate)       (None, 32, 32, 512)    0          block7_norm1[0][0]
                                                                    block3_conv3[0][0]
block7_conv1 (Conv2D)             (None, 32, 32, 256)    1179904    concatenate_6[0][0]
block7_norm2 (BatchNormalization) (None, 32, 32, 256)    1024       block7_conv1[0][0]
block7_conv2 (Conv2D)             (None, 32, 32, 256)    590080     block7_norm2[0][0]
block7_norm3 (BatchNormalization) (None, 32, 32, 256)    1024       block7_conv2[0][0]
block8_upsamp (UpSampling2D)      (None, 64, 64, 256)    0          block7_norm3[0][0]
block8_upconv (Conv2D)            (None, 64, 64, 128)    131200     block8_upsamp[0][0]
block8_norm1 (BatchNormalization) (None, 64, 64, 128)    512        block8_upconv[0][0]
concatenate_7 (Concatenate)       (None, 64, 64, 256)    0          block8_norm1[0][0]
                                                                    block2_conv2[0][0]
block8_conv1 (Conv2D)             (None, 64, 64, 128)    295040     concatenate_7[0][0]
block8_norm2 (BatchNormalization) (None, 64, 64, 128)    512        block8_conv1[0][0]
block8_conv2 (Conv2D)             (None, 64, 64, 128)    147584     block8_norm2[0][0]
```

```
block8_norm3 (BatchNormalization) (None, 64, 64, 128)    512         block8_conv2[0][0]
_____________________________________________________________________________________________
block9_upsamp (UpSampling2D)      (None, 128, 128, 128   0           block8_norm3[0][0]
_____________________________________________________________________________________________
block9_upconv (Conv2D)            (None, 128, 128, 64)   32832       block9_upsamp[0][0]
_____________________________________________________________________________________________
block9_norm1 (BatchNormalization) (None, 128, 128, 64)   256         block9_upconv[0][0]
_____________________________________________________________________________________________
concatenate_8 (Concatenate)       (None, 128, 128, 128   0           block9_norm1[0][0]
                                                                     block1_conv2[0][0]
_____________________________________________________________________________________________
block9_conv1 (Conv2D)             (None, 128, 128, 64)   73792       concatenate_8[0][0]
_____________________________________________________________________________________________
block9_norm2 (BatchNormalizatio   (None, 128, 128, 64)   256         block9_conv1[0][0]
_____________________________________________________________________________________________
block9_conv2 (Conv2D)             (None, 128, 128, 64)   36928       block9_norm2[0][0]
_____________________________________________________________________________________________
block9_norm3 (BatchNormalization) (None, 128, 128, 64)   256         block9_conv2[0][0]
_____________________________________________________________________________________________
output (Conv2D)                   (None, 128, 128, 1)    65          block9_norm3[0][0]
=============================================================================================
Total params: 25,866,177
Trainable params: 11,145,729
Non-trainable params: 14,720,448
_____________________________________________________________________________________________
```

El modelo está configurado por defecto para que deje “congelados” los pesos preentrenados del codificador *VGG-16*, es decir que no se modifiquen durante el entrenamiento. Por ello, algo más la mitad de los parámetros aparecen como no entrenables. Esto va a reducir el tiempo de entrenamiento significativamente.

# 3 RESULTADOS

*La calidad nunca es un accidente; siempre es el resultado de un esfuerzo de la inteligencia.*

– John Ruskin –

A lo largo de este capítulo se presentarán los resultados del entrenamiento de la red, así como sus capacidades de predicción. Para ello se utilizará el conjunto de datos de entrenamiento del *challenge* de segmentación del ISIC 2018, el cual tiene un total de 2594 imágenes en color en formato JPEG, con sus máscaras asociadas en blanco y negro en formato PNG [16] [17]. En dicho concurso se incluyen también sets de datos de validación y test final, sin embargo no están disponibles las máscaras asociadas ya que se comprueba la validez a través de su propio sistema, el cual está ya desactivado. Por lo tanto, de dichas imágenes se tomará aproximadamente el 15% (400 imágenes) para realizar la validación, y las restantes 2194 para el entrenamiento de la red.

## 3.1 Mediciones de rendimiento

Para medir la calidad de un sistema de aprendizaje automático hay muchas posibles métricas, dependiendo de lo que se quiera valorar con ello, o dar más importancia a ciertos tipos de errores o aciertos.

Se puede decir que, píxel a píxel, hay 4 tipos de predicciones, considerando el positivo como presencia de lesión (píxel activo) y el negativo como no lesión o fondo [29]:

- **Verdadero Positivo** (TP): ejemplo positivo que se ha predicho como positivo.
- **Falso Positivo** (FP): ejemplo negativo que se ha predicho positivo. También se le denomina error tipo I.
- **Verdadero Negativo** (TN): ejemplo negativo predicho correctamente como negativo.
- **Falso Negativo** (FN): ejemplo positivo que se ha predicho como negativo. Denominado error tipo II.

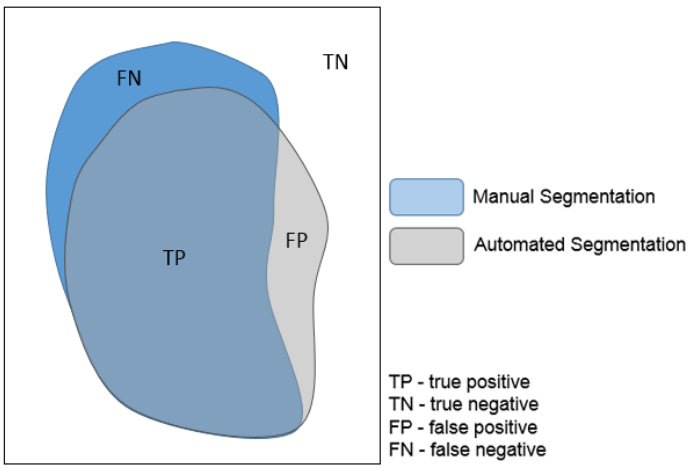



*Figura 3-1: Tipos de predicciones en una segmentación [62].*

En función de estos valores, se pueden calcular varios tipos de métricas, entre las que se incluyen las que se recogen a continuación. Para el reto del ISIC 2018 la métrica utilizada es el índice Jaccard, aunque por proporcionar una información completa del comportamiento del algoritmo también se calculan las demás.

#### Tasa de acierto (*accuracy*)

Mide, de entre todos los datos, qué proporción se han clasificado correctamente, ya sea por verdadero positivo o verdadero negativo:

$$acc = \frac{TP + TN}{TP + FP + TN + FN}$$

Aunque es la métrica que se suele utilizar por defecto en los sistemas de aprendizaje automático, no es adecuada cuando hay una mayoría de ejemplos de uno de los tipos (positivos o negativos). Esto es así debido a que no tiene en cuenta dicho desequilibrio, y tendrán por tanto más peso los datos del tipo mayoritario.

#### Precisión (*precision*)

Mide la tasa de los clasificados como positivos que realmente lo son:

$$P = \frac{TP}{TP + FP}$$

#### Sensibilidad (*sensitivity* o *recall*)

Mide la proporción de los que realmente son positivos que se han clasificado como tal. También se le denomina tasa de verdaderos positivos o TPR.

$$R = TPR = \frac{TP}{TP + FN}$$

#### Especificidad (specificity)

Mide la proporción de los que realmente son negativos que se han clasificado como tal. Se le llama también tasa de falsos negativos.

$$TNR = \frac{TN}{TN + FP}$$

#### Índice Jaccard (*Jaccard* index)

Es una medida de la similitud entre dos conjuntos de muestras, A y B. En particular, se puede considerar A como el valor predicho y B como el valor real, siendo ambos imágenes y las muestras sus píxeles correspondientes. Se define como:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

También se le llama *Intersection over Union* (IoU). Es una métrica muy utilizada en visión artificial, pues es una medida del solapamiento entre los píxeles de dos imágenes:

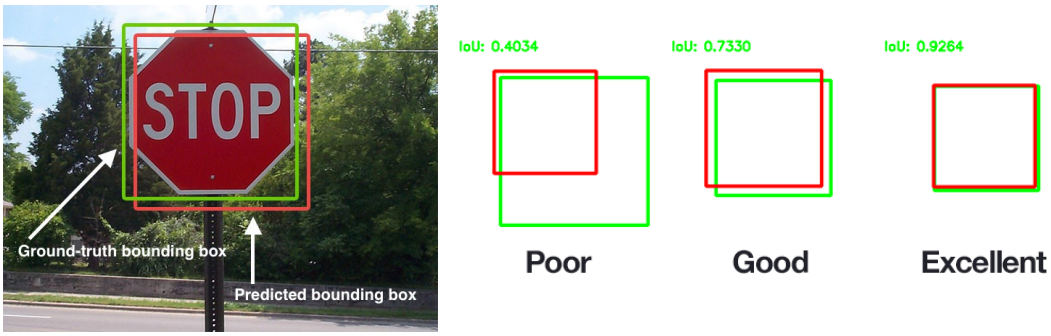



*Figura 3-2: Índice Jaccard para detección de objetos en imágenes [62]*

En función de los tipos de predicción, se puede obtener como:

$$J = \frac{TP}{TP + FN + FP}$$

**Coeficiente de Sorensen-Dice (*Sorensen-Dice coefficient*)**

El coeficiente de Sorensen-Dice, o coeficiente de Dice, también mide la similitud entre dos conjuntos, definiéndose como:

$$DSC(A,B) = \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

Igualmente, en función de los tipos de predicción binaria se puede expresar como:

$$DSC = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

Este coeficiente es equivalente a lo que se conoce como medida $F_1$ o *score* $F_1$, el cual es un compromiso entre la precisión y la sensibilidad, definido como:

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP} = DSC$$

## 3.2 Entrenamiento y validación

### 3.2.1 Parámetros iniciales

En la siguiente tabla se recogen los parámetros utilizados para el entrenamiento definitivo de la red:

*Tabla 3-1: Parámetros de entrenamiento.*

| *Parámetro* | *Valor* |
|---|---|
| ***Dimensiones de entrada*** | 128x128x3 |
| ***Tamaño del conjunto de datos*** | 2194 |
| ***Tamaño del batch*** | 2 |
| ***Épocas*** | 50 |
| ***Tipo de optimizador*** | Adam |
| ***Tasa de aprendizaje*** | $1 \times 10^{-3}$ (valor por defecto) |

El modelo del que se ha partido es el indicado en la sección 2.3.3, en el que se tiene el codificador *VGG-16* con

los pesos preentrenados y congelados, y las demás capas de la *U-Net* inicializadas mediante una distribución normal.

### 3.2.2 Resultados

A continuación se presentarán los resultados del entrenamiento y validación de la red. En primer lugar se muestra la evolución de la tasa de acierto (*accuracy*) y la función de coste (*loss*) a lo largo del tiempo, tanto para el conjunto de datos como para el conjunto de validación:

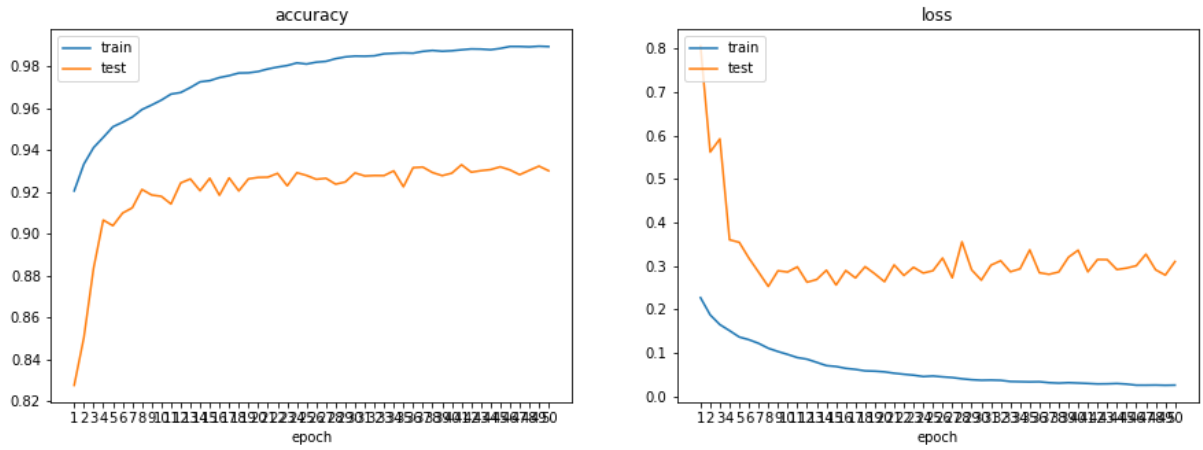



*Figura 3-3: Evolución de la tasa de acierto y la función de pérdidas de entrenamiento y validación.*

De esta figura se pueden hacer las siguientes observaciones:

i) La tasa de acierto tras 1 sola época, con los pesos *VGG-16* preentrenados y el resto de capas inicializadas a una distribución normal, es de un 92% para los datos de entrenamiento, lo cual es un valor bastante bueno teniendo en cuenta que apenas ha sido entrenada con el tipo de datos requeridos.

ii) Para la validación, tras apenas 8 épocas se consigue una tasa de acierto del 92%, estabilizándose en torno al 93% para las siguientes épocas. Por consiguiente, se ve que a pesar de que la *accuracy* sigue aumentando para el set de entrenamiento, la de validación apenas cambia en las últimas 40 épocas.

iii) La función de coste tiene el mismo comportamiento pero opuesto, es decir decrece progresivamente para el conjunto de entrenamiento, pero a partir de la décima época apenas disminuye para el set de validación, incluso crece levemente, lo que puede ser un síntoma de que está comenzando a sobreentrenarse.

Los valores obtenidos para las métricas de calidad finales se recogen en la siguiente tabla:

*Tabla 3-2: Métricas finales de rendimiento de la red neuronal implementada.*

| *Parámetro* | *Valor* |
|---|---|
| ***Tasa de acierto (accuracy)*** | 0.930142365 |
| ***Precisión (precision)*** | 0.911361552 |
| ***Sensibilidad (sensitivity)*** | 0.827341539 |
| ***Especificidad (specificity)*** | 0.974947274 |
| ***Coeficiente Dice*** | 0.844504763 |
| ***Índice Jaccard*** | 0.760913533 |

En el concurso del ISIC, la métrica utilizada para valorar los modelos no es en realidad el índice Jaccard, sino

una versión modificada de éste que sólo tiene en cuenta aquellas imágenes que superan un valor **umbral de 0.65**. Para las predicciones que queden por debajo, se considera un coeficiente Jaccard igual a 0, lo que indica que la **segmentación es fallida.** Con este criterio, se calcula un valor medio que se puede denominar **índice Jaccard con umbral.** Teniendo esto en cuenta, el valor obtenido es el siguiente:

*Tabla 3-3: Métrica final de valoración del modelo según el ISIC challenge 2018.*

| *Parámetro* | *Valor* |
|---|---|
| ***Índice Jaccard puro*** | 0.760913533 |
| ***Índice Jaccard con umbral*** | 0.681393492 |

Para tener una referencia de la calidad de este sistema, el ganador del *challenge* logró un índice Jaccard de 0.838, y un índice Jaccard con umbral de 0.802. Este método, propuesto por Qian, Jian y Liu [63], utiliza también una arquitectura codificador-decodificador, inspirada en la *DeepLab* [46] y la *PSPNet* [45], las cuales utilizan como extractor de características la red de Microsoft ***ResNet*** [39]**.**

### 3.2.3 Coste computacional

Se incluye también en la siguiente tabla el coste computacional de la ejecución del entrenamiento y validación:

*Tabla 3-4: Coste computacional del entrenamiento.*

| *Parámetro* | *Valor* |
|---|---|
| ***Épocas*** | 50 |
| ***Tiempo*** | 5h 39 m |
| ***RAM usada*** | 3.41 GB |
| ***GPU usada*** | 5.51 GB |

## 3.3 Predicción

En esta sección se muestran algunos ejemplos de predicciones de máscaras del set de datos de validación, incluyendo las 5 que mejor coeficiente Jaccard han obtenido y las 5 peores.

En primer lugar, se recoge en la siguiente tabla el número de imágenes con valores de índice Jaccard por encima de la media y por encima del umbral de 0.65.

*Tabla 3-5: Relación de predicciones por encima de la media y del umbral de índice Jaccard*

| *Parámetro* | *Valor* |
|---|---|
| ***Número total de imágenes*** | 400 |
| ***Índice Jaccard con umbral (valor medio)*** | 0.681 |
| ***Número de imágenes por encima de la media*** | 309 (77.2%) |
| ***Número de imágenes por encima del umbral Jaccard*** | 324 (81%) |

A continuación se muestran las predicciones de las segmentaciones con las que se han obtenido los 5 mejores valores de índice Jaccard.

*Tabla 3-6: Índice Jaccard de las 5 mejores predicciones.*

| *ID de la imagen* | *Índice Jaccard* | *Accuracy* |
|---|---|---|
| ***ISIC_0015113*** | 0.980652613 | 0.995910645 |
| ***ISIC_0014923*** | 0.969565217 | 0.997436523 |
| ***ISIC_0014846*** | 0.96665204 | 0.981445313 |
| ***ISIC_0014897*** | 0.964712269 | 0.99206543 |
| ***ISIC_0015139*** | 0.963675214 | 0.995849609 |

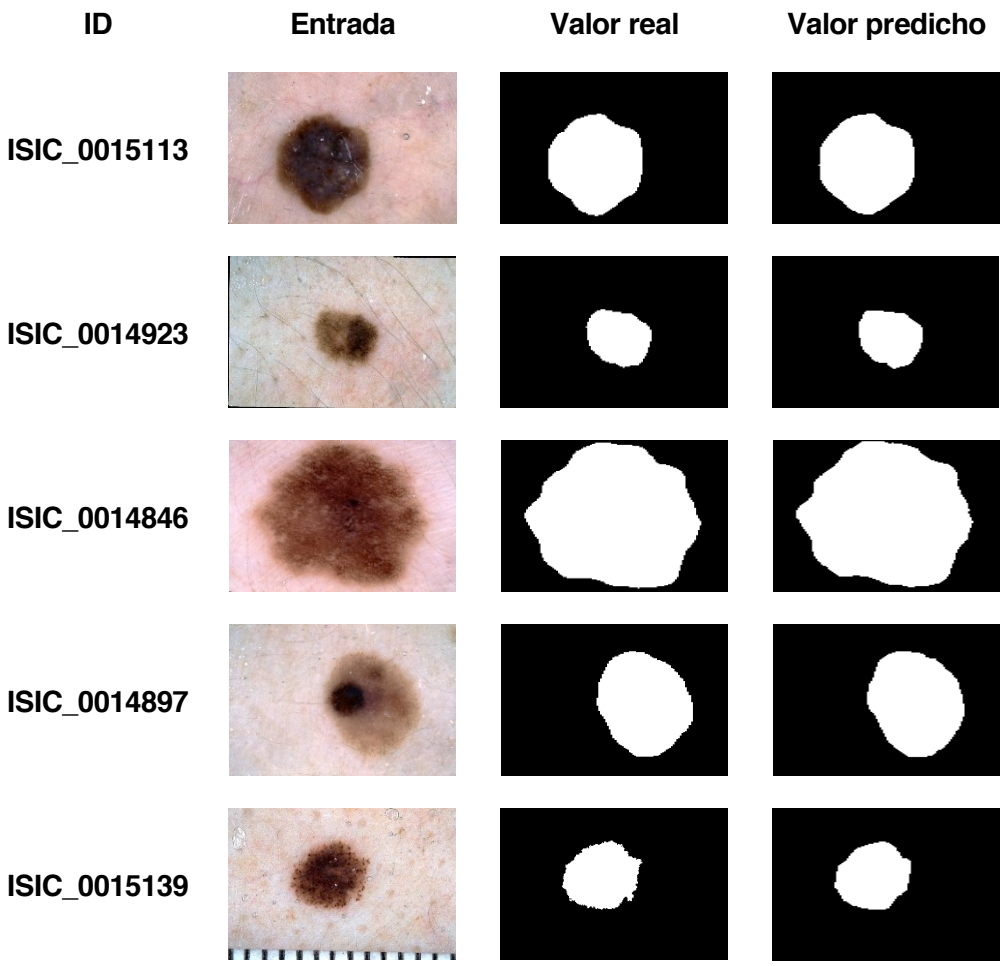



*Figura 3-4: Resultado de la segmentación de las 5 mejores predicciones.*

En las siguientes figuras se recogen las predicciones de las imágenes con los 5 peores valores de índice Jaccard.

*Tabla 3-7: Índice Jaccard de las 5 peores predicciones.*

| *ID de la imagen* | *Índice Jaccard* | *Accuracy* |
|---|---|---|
| ***ISIC_0016027*** | 0 | 0.99151611 |
| ***ISIC_0015978*** | 0 | 0.96765137 |
| ***ISIC_0016034*** | 0.004338395 | 0.91595459 |
| ***ISIC_0015251*** | 0.023715415 | 0.68341064 |
| ***ISIC_0014836*** | 0.077507201 | 0.78497314 |

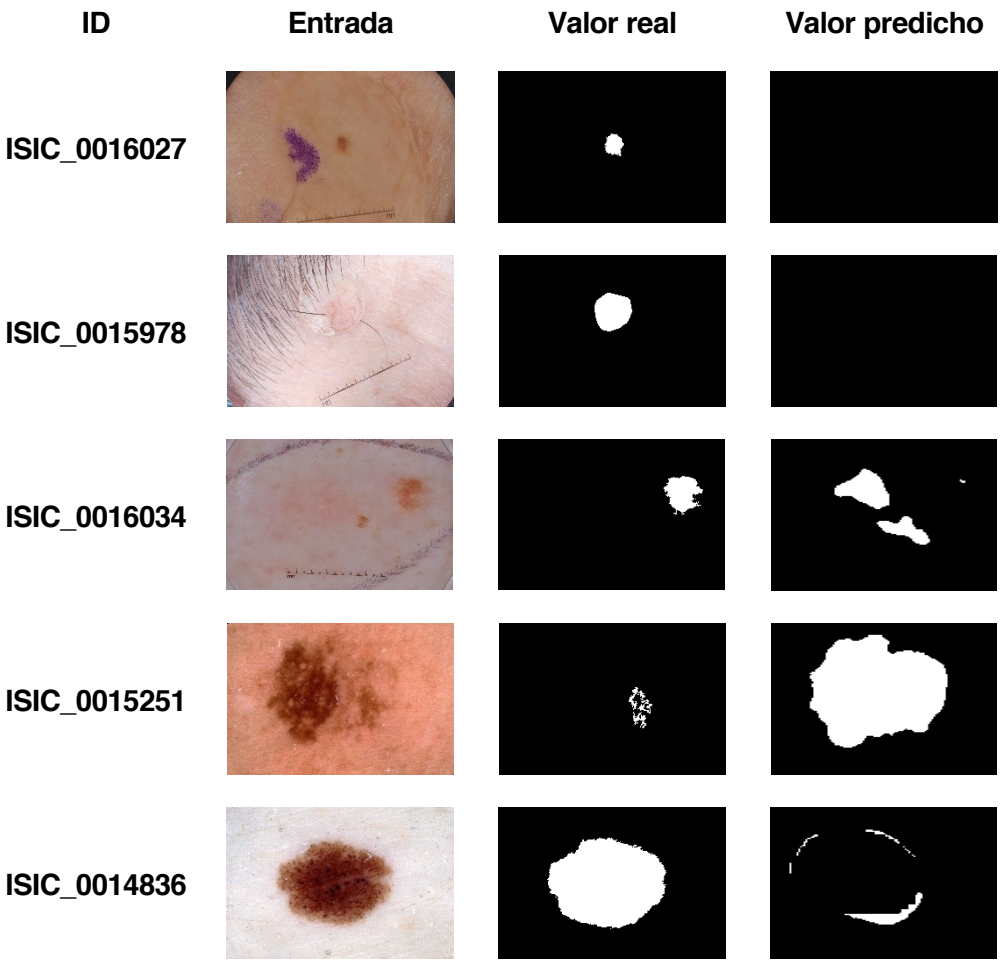



*Figura 3-5: Resultado de la segmentación de las 5 peores predicciones.*

En general, se puede deducir que los resultados son bastante buenos, y que las peores predicciones (a excepción de la última) corresponden como era de esperar a imágenes de lesiones muy particulares, pequeñas o con poca pigmentación, las cuales son más difíciles de aprender.

También resulta evidente la inadecuación de la tasa de acierto para medir la calidad de este tipo de sistemas, ya que por ejemplo para la imagen ISIC_0016027 el *accuracy* es del 99%, pero el índice Jaccard es de 0, dado que la predicción es una imagen totalmente negra. Esto es debido a que la lesión es muy pequeña, habiendo por tanto

una gran mayoría de píxeles no activos, los cuales evidentemente están bien clasificados pero el resultado final no es bueno.

Adicionalmente, se muestran a continuación también los resultados de predicción de datos más cercanos al punto medio; en concreto, se han considerado las 5 primeras imágenes con índice Jaccard inmediatamente superior a la media, la cual es de 0.681:

*Tabla 3-8: Índice Jaccard de las 5 primeras predicciones que superan el valor medio.*

| *ID de la imagen* | *Índice Jaccard* | *Accuracy* |
|---|---|---|
| ***ISIC_0014931*** | 0.695965418 | 0.884094238 |
| ***ISIC_0015466*** | 0.691771794 | 0.800170898 |
| ***ISIC_0015200*** | 0.691698595 | 0.778991699 |
| ***ISIC_0015975*** | 0.69140625 | 0.985534668 |
| ***ISIC_0015956*** | 0.682828283 | 0.961669922 |

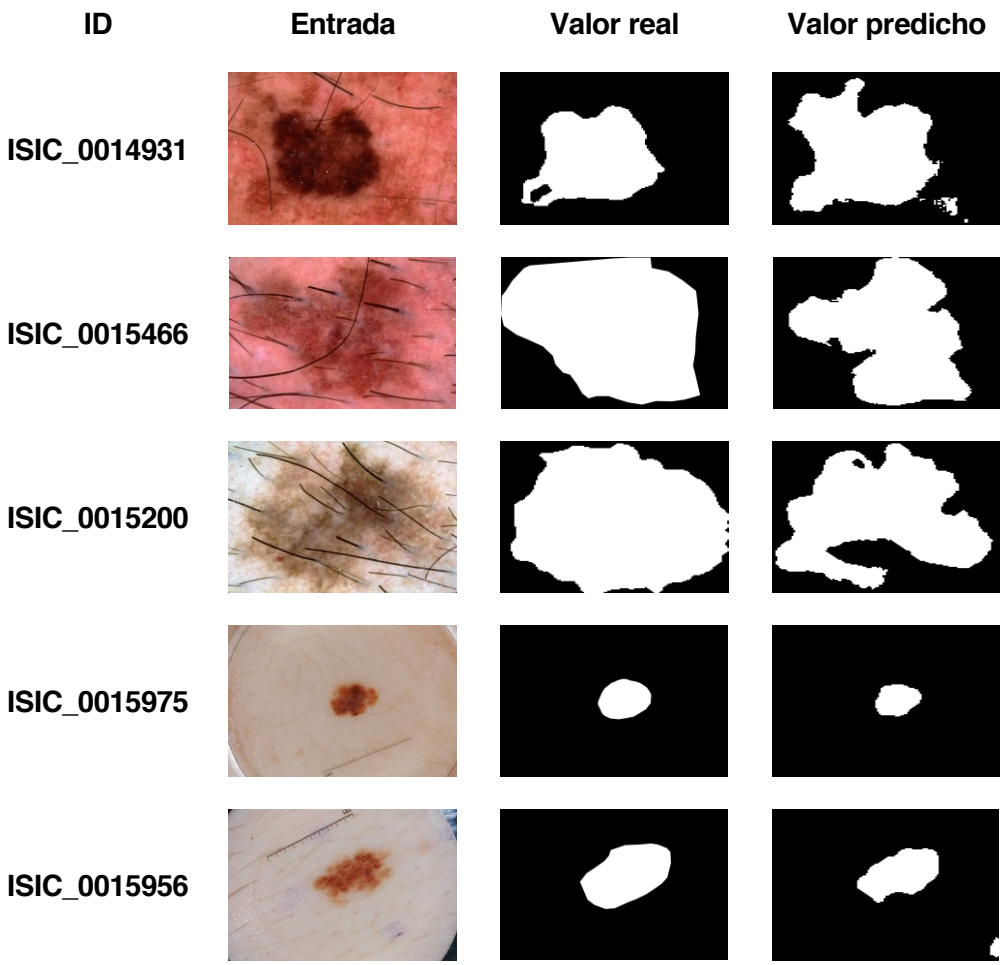



*Figura 3-6: Resultado de la segmentación de las 5 primeras predicciones que superan el valor medio del índice Jaccard.*

Para concluir, se ha querido también mostrar los resultados para las imágenes cuyo índice Jaccard se encuentra justo por encima y por debajo del umbral de 0.65.

*Tabla 3-9: Índice Jaccard de las 6 predicciones en torno al umbral de 0.65.*

| *ID de la imagen* | *Índice Jaccard* | *Accuracy* |
|---|---|---|
| ***ISIC_0014908*** | 0.655565294 | 0.928039551 |
| ***ISIC_0015995*** | 0.651982379 | 0.971069336 |
| ***ISIC_0016062*** | 0.650793651 | 0.985229492 |
| ***ISIC_0015982*** | 0.647776809 | 0.975341797 |
| ***ISIC_0016012*** | 0.646802326 | 0.985168457 |
| ***ISIC_0015185*** | 0.64005788 | 0.817810059 |

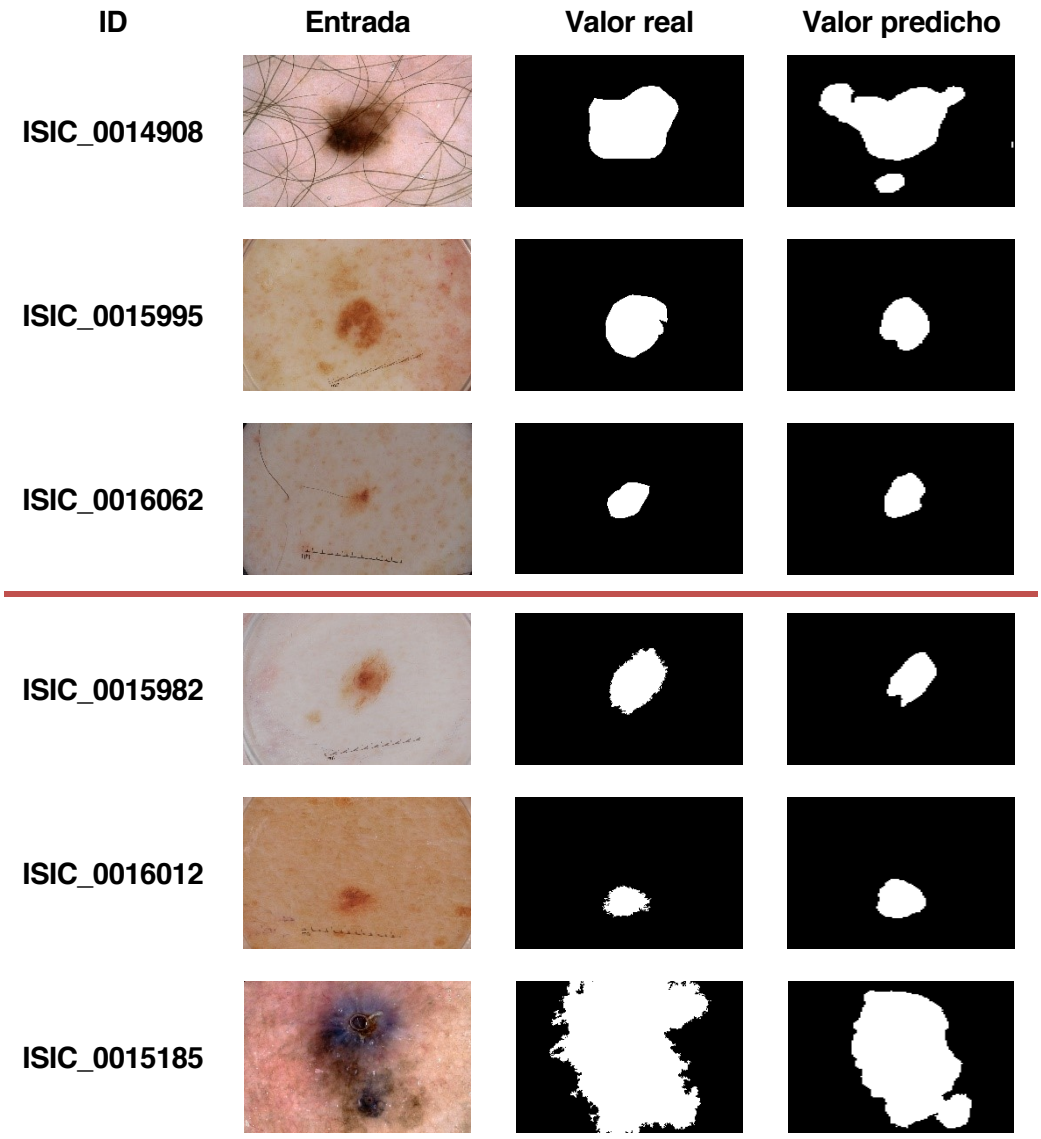



*Figura 3-7: Resultado de la segmentación de las 6 predicciones con índice Jaccard en torno al umbral de 0.65.*

La elección de uso del índice Jaccard con umbral en lugar del puro se basa en un análisis [64] que demostró que el uso directo de Jaccard como medida de rendimiento no reflejaba con precisión el número de imágenes en las que falla la segmentación automatizada. En la Tabla 2-1 se pueden consultar dichos números, que también se pueden observar en el histograma:

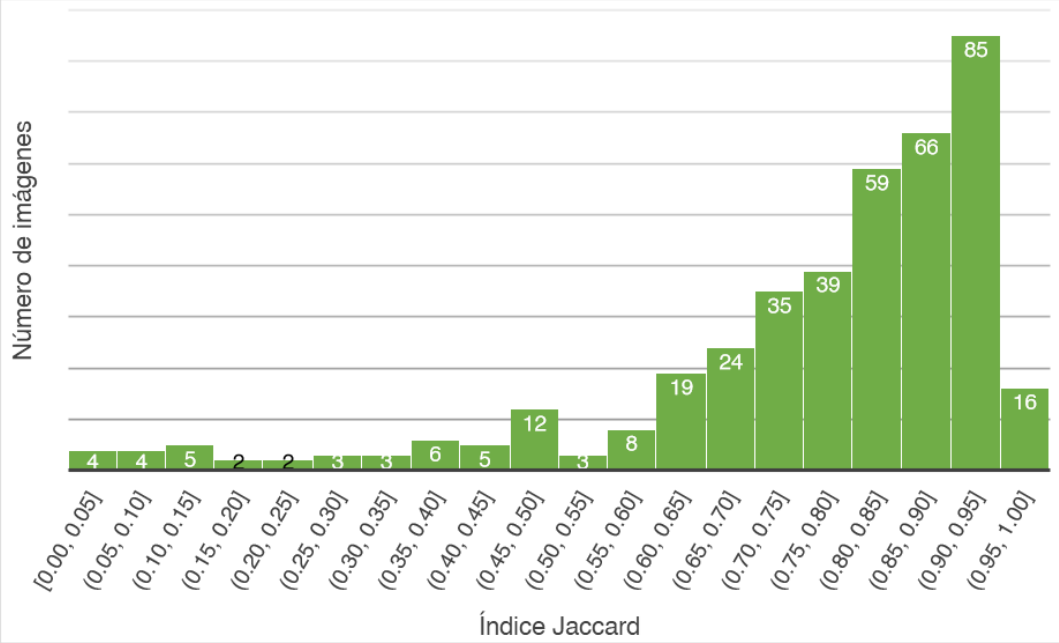



*Figura 3-8: Histograma del índice Jaccard de las predicciones*

Visualmente, cuando el índice Jaccard cae por debajo del umbral, la corrección o validez de la segmentación puede ser objeto de debate. En este caso, un total de 76 imágenes han quedado por debajo del umbral, lo que supone un 19% de las imágenes.

# 4 CONCLUSIONES

*La idea o la información es sólo una semilla en la mente. Lo que realmente hará que las cosas cambien es la acción.*

– Miguel Ángel Ruiz Macías –

El aprendizaje profundo ha revolucionado el mundo de la inteligencia artificial. Tras la realización de este Trabajo Fin de Máster, resulta evidente que se requiere de unas altas capacidades de procesamiento para poder obtener buenos resultados, y es por tanto comprensible el estancamiento que sufrió el *deep learning* hasta que dicha potencia de procesamiento fue desarrollada y aparecieron las GPUs.

Estando éstas disponibles –tras previo pago– las posibilidades son infinitas, no únicamente en el campo de la medicina. En los últimos años se está avanzando a pasos agigantados, y se puede pensar que las tecnologías de inteligencia artificial van a jugar un papel muy importante en la sociedad del futuro. De esta forma, en campos como el médico, se puede esperar que la tecnología dé el salto desde la investigación –muy activa en numerosas áreas– a la aplicación en entornos reales, ayudando a la detección y prevención de enfermedades.

## 4.1 Conclusiones

Tras la conclusión de este proyecto, se puede afirmar que se han cumplido los objetivos que en un principio se establecieron, esto es, desarrollar un sistema capaz de segmentar automáticamente imágenes de lesiones pigmentadas. Tras el entrenamiento del sistema, se ha logrado una tasa de acierto del 98.9% sobre el conjunto de datos utilizado, y consiguiendo un buen 93% de *accuracy* e **índice Jaccard de 0.761** sobre los datos utilizados para test.

Si bien no son resultados perfectos, se afirma que son buenos porque se puede comparar –aunque sin llegar a su nivel– con el rendimiento obtenido por los ganadores del reto de ISIC 2018, quienes obtuvieron 94.2% de tasa de acierto y un índice Jaccard de 0.838.

Esta diferencia se debe, además de la evidente diferencia entre los modelos, a varios factores, como pueden ser:

i) Los datos de test no son los mismos en ambos casos, no siendo posible por tanto una comparación objetiva. Sin embargo estas medidas pueden servir de indicativo de la calidad del sistema, y se pueden considerar aceptables en general teniendo en cuenta los resultados obtenidos por investigadores en este campo.

ii) El tamaño de los conjuntos de entrenamiento no es el mismo en ambos casos, debido a la necesidad en este proyecto de extraer los datos de test de dicho conjunto. Por consiguiente, no han tomado parte del entrenamiento estas 400 imágenes. Esto es solucionable utilizando la técnica del *data augmentation*, la cual se comenta en las líneas futuras (sección 4.2).

iii) No se han entrenado las capas del codificador debido a la técnica de *transfer learning* utilizada y a las limitaciones de computación. Éstas están únicamente entrenadas únicamente para la base de datos de *ImageNet,* por lo que no extraerán las posibles características específicas de las lesiones pigmentadas.

También es importante destacar el gran potencial que tiene la técnica del *transfer learning*. Con bases de datos como la de *ImageNet*, que contiene más de 1 millón de imágenes clasificadas en 1000 categorías, se impulsa enormemente el desarrollo de redes de clasificación de imágenes, que hacen disponibles los pesos obtenidos al que quiera beneficiarse de ellos. La capacidad de utilizar modelos existentes facilita su aplicación a nuevos retos y cuestiones, acelerando su desarrollo y permitiendo a los desarrolladores centrar sus esfuerzos en los desafíos específicos del problema a tratar.

A pesar de haber entrenado la red durante 50 épocas, los resultados han evidenciado que con 20 épocas es suficiente, además de recomendado para evitar que la red se sobreentrene. El **valor óptimo** en cuanto a coste de validación es 9 épocas, punto en el que se obtiene un valor de la función de coste de 0.252 y una tasa de acierto del 92.1%, siendo la tasa de acierto máxima de 93.3% en 42 épocas. Es posible parar el entrenamiento cuando la función de coste deja de mejorar y guardar el modelo con los pesos de ese momento, técnica que se denomina *early stopping*. No obstante, es necesario aclarar que los resultados de predicción recogidos en esta memoria no son para el modelo óptimo sino para el que se ha ejecutado durante 50 épocas.

Finalmente, conviene resaltar el valor del entorno ***Keras*** para introducirse en el mundo de aprendizaje profundo. Aunque es un *framework* menos utilizado que otros en entornos comerciales, es sin duda un paquete extremadamente intuitivo, que permite la definición y entrenamiento de redes neuronales de forma muy sencilla, rápida y bastante personalizable, además de estar respaldado por la potencia de *TensorFlow* si se desea. Asimismo, cuenta con una comunidad muy activa, que hace disponibles numerosos artículos, tutoriales, foros y modelos desarrollados por los propios usuarios, lo cual contribuye enormemente a aprenderlo de forma autodidacta, como ha sido el caso de este proyecto.

## 4.2 Líneas futuras

Habiendo cumplido los objetivos establecidos para este Trabajo Fin de Máster, se considera que se le podría dar continuidad al estudio con las siguientes líneas de trabajo con el objetivo de perfeccionar el modelo y evitar el sobreajuste:

i) Una forma de evitar el sobreajuste es proporcionando más datos de entrenamiento, para ayudar a la red a que generalice mejor. Dado que no se dispone de dichos datos, una técnica muy utilizada es el **aumento de datos** (*data augmentation*), por la cual se realizan transformaciones sobre las propias imágenes que se tienen disponibles, como son rotaciones, aumentos o transformaciones elásticas. De hecho, en el artículo de la *U-Net* se recomienda este tipo de práctica, y en general en el mundo de la imagen biomédica, ya que la deformación es la variación más común en el tejido real, pudiendo ser simulada eficientemente mediante transformaciones.

ii) Otra forma de solucionar el sobreajuste es implementar **regularización** en la red. La regularización es una técnica que trata de simplificar el modelo de la red, ya que cuanto más complejo sea más probabilidades hay de que se ajuste demasiado a los datos de entrenamiento. Para ello se establecen algunos parámetros a 0 de forma que no contribuyan a la solución, y consecuentemente simplificando el modelo. En redes neuronales, el método utilizado para ello es el ***dropout.*** Para la red *U-Net*, el lugar recomendado para implementar el *dropout* es al final del camino de contracción.

iii) Para mejorar el rendimiento de la red, es también importante controlar la tasa de aprendizaje. Implementar una **tasa de aprendizaje adaptativa**, que comience alta y vaya disminuyendo conforme avancen las épocas, asegura que al comienzo de la ejecución se avance lo suficientemente rápido, y conforme se acerque a la solución se realice un ajuste más fino, permitiendo llegar a la solución óptima.

iv) Implementar programáticamente el *early stopping* y guardar el modelo óptimo, de forma que se puedan realizar con él las predicciones definitivas de la red.

Todas estas mejoras propuestas son susceptibles de ser implementadas en *Keras*. La primera de ellas es muy sencilla de realizar, mediante el paquete `ImageDataGenerator`, aunque al tener más imágenes será más lento el

entrenamiento.

Para la regularización existe la capa `Dropout`, que no hay más que añadir al modelo detrás de la capa que se desee, y anulará los parámetros que considere óptimos mediante el entrenamiento. Ésta no se ha implementado debido a que el uso de *BatchNormalization* también tiene un efecto regularizador, y su uso en conjunto con *Dropout* no está probado que sea necesario, aunque se deja también como línea de investigación abierta el ver si se complementan entre ellas o cuál tiene más ventajas.

En cuanto a la variación dinámica de la tasa de aprendizaje, algunos optimizadores de *Keras* admiten parámetros que permiten controlar la caída (***decay***) de la tasa de aprendizaje, como es el caso de Adam. De hecho, además del *decay* personalizado, Adam ya implementa por defecto una caída de la tasa de aprendizaje.

Otra forma de adaptar la tasa de aprendizaje es el uso de ***callbacks*** durante el entrenamiento. En concreto, *Keras* ofrece la clase `LearningRateScheduler` que facilita su implementación. Asimismo, existe el *callback* `EarlyStopping` que permite parar el entrenamiento cuando una métrica dada deja de mejorar.

# ANEXO A: REFERENCIAS


[1] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," University of Freiburg, 2015.

[2] ISBI, "Segmentation of neuronal structures in EM stacks Challenge," 2012. [Online]. Available: http://brainiac2.mit.edu/isbi_challenge/. [Accessed 10 Jul 2019].

[3] D. Ciresan, L. Gambardella, A. Giusti y J. Schmidhuber, «Deep neural networks segment neuronal membranes in electron microscopy images,» *NIPS,* p. 2852–2860, 2012.

[4] ISBI, "Cell Tracking Challenge," [Online]. Available: http://celltrackingchallenge.net/. [Accessed 10 Jul 2019].

[5] The Gale Group Inc., "Technology: I. History of Medical Technology," in *Encyclopedia of Bioethics*, 2004.

[6] IARC, "Global Cancer Observatory," 2019. [Online]. Available: https://gco.iarc.fr/. [Accessed 9 Jun 2019].

[7] The Skin Cancer Foundation, "Skin Cancer Information: Melanoma," 2019. [Online]. Available: https://www.skincancer.org/skin-cancer-information/melanoma. [Accessed 9 Jun 2019].

[8] J. Jaworek-Korjakowska, "Computer-Aided Diagnosis of Micro-Malignant Melanoma Lesions Applying Support Vector Machines," *BioMed Research International,* vol. 2016 (Article ID 4381972), 2016.

[9] International Dermoscopic Society; International Skin Imaging Collaboration, "Dermoscopedia," 2019. [Online]. Available: https://dermoscopedia.org/. [Accessed 9 Jun 2019].

[10] J. J. C. García, "Lesiones Pigmentadas: Valoración Dermatoscópica," *Revista Clínica de Medicina en Familia,* vol. 4, no. 2, pp. 177-179, 2011.

[11] International Skin Imaging Collaboration, "International Skin Imaging Collaboration," 2019. [Online]. Available: https://www.isic-archive.com/. [Accessed 30 Jun 2019].

[12] M. Zortea, S. O. Skrøvseth, T. R. Schopf, H. M. K. and F. Godtliebsen, "Automatic Segmentation of Dermoscopic Images by Iterative Classification," *International Journal of Biomedical Imaging,* vol. 2011 (Article ID 972648), 2011.

[13] A. Masood and A. A. Al-Jumaily, "Computer Aided Diagnostic Support System for Skin Cancer: A Review of Techniques and Algorithms," *International Journal of Biomedical Imaging,* vol. 2013 (Article ID 323268), 2013.

[14] F.-F. Li and J. C. Niebles, "Computer Vision: Foundations and Applications, course notes (s131)," Stanford University, 2016. [Online]. Available:

http://vision.stanford.edu/teaching/cs131_fall1617/index.html. [Accessed 19 Jun 2019].

[15] N. Sharma, A. K. Ray, K. S. Sharma, K. Shukla, S. Pradhan and L. M. Aggarwal, "Segmentation and classification of medical images using texture-primitive features: Application of BAM-type artificial neural network," *Journal of Medical Physics,* vol. 33, no. 3, pp. 119-26, 2008.

[16] N. Codella, V. Rotemberg, P. Tschandl, M. E. Celebi, S. Dusza, D. Gutman, B. Helba, A. Kalloo, K. Liopyris, M. Marchetti, H. Kittler and A. Halpern, "Skin Lesion Analysis Toward Melanoma Detection 2018: A Challenge Hosted by the International Skin Imaging Collaboration (ISIC)," 2018.

[17] P. Tschandl, C. Rosendahl and H. Kittler, "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," *Scientific Data,* vol. 5, no. 180161, 2018.

[18] M. E. Celebi, N. Codella and A. Halpern, "Dermoscopy Image Analysis: Overview and Future Directions," *IEEE Journal of Biomedical and Health Informatics,* vol. 23, no. 2, pp. 474-478, March 2019.

[19] P. Bhati and M. Singha, "Segmentation Techniques for Skin Lesion: A Comparative Study," *International Journal of Tren in Research and Development,* vol. 5, no. 3, May 2018.

[20] V. Revathi and A. Chithra, "A Review on Segmentation Techniques in Skin Lesion Images," *International Research Journal of Engineering and Technology,* vol. 2, no. 9, Dec 2015.

[21] T. Mendoza et al., "Comparison of Segmentation Methods for Automatic Diagnosis of Dermoscopy Images," *29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society,* pp. 6572-6575, 2007.

[22] J. Rogowska, "Handbook of Medical Image Processing and Analysis," Academic Press, 2009.

[23] C. Piech, "Artificial Intelligence: Principles and Techniques, course notes (cs221)," Stanford University, 2013. [Online]. Available: https://stanford.edu/~cpiech/cs221/index.html. [Accessed 19 Jun 2019].

[24] S. Yuheng and Y. Hao, "Image Segmentation Algorithms Overview," SiChuan University, ChengDu, 2017.

[25] J. Capitán Fernández and J. Ferruz Melero, "Sistemas Inteligentes, course notes (51420030)," University of Seville, 2018.

[26] Nvidia Corporation, "Nvidia blog: Deep Learning," 2019. [Online]. Available: https://blogs.nvidia.com/blog/category/deep-learning/. [Accessed 19 Jun 2019].

[27] Oracle Corporation, "Oracle Big Data Blog," [Online]. Available: https://blogs.oracle.com/bigdata/. [Accessed 19 Jun 2019].

[28] BigML, Inc., "Machine Learning 101," 2019. [Online]. Available: https://bigml.com/ml101. [Accessed 19 Jun 2019].

[29] J. J. Murillo Fuentes and J. C. Aradillas Jaramillo, "Análisis de Datos y Procesado de la Información, course notes (51420022)," University of Seville, 2019.

[30] V. Jha, "Machine Learning Algorithm - Backbone of emerging technologies," 2017. [Online]. Available: https://www.techleer.com/articles/203-machine-learning-algorithm-backbone-of-emerging-

technologies/. [Accessed 26 Jun 2019].

[31] A. Karpathy, "Convolutional Neural Networks for Visual Recognition, course notes (cs231n)," Stanford University, 2019. [Online]. Available: http://cs231n.stanford.edu/. [Accessed 19 Jun 2019].

[32] S. Jain, "An Overview of Regularization Techniques in Deep Learning," 19 Apr 2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/. [Accessed 26 Jun 2019].

[33] M. Peemen, B. Mesman and H. Corporaal, "Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform," in *Advances Concepts for Intelligent Vision Systems - 13th International Conference*, Ghent, Belgium, 2011.

[34] Y. Lecun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE,* vol. 86, nº 11, pp. 2278-2324, November 1998.

[35] K. Nguyen, C. Fookes, A. Ross y S. Sridharan, «Iris Recognition With Off-the-Shelf CNN Features: A Deep Learning Perspective,» *IEEE Access,* vol. 6, pp. 18848-18855, 2018.

[36] A. Xrizhevsky, I. Sutskever y G. E. Hinton, «ImageNet Classification with Deep ConvolutionalNeural Networks,» University of Toronto, 2012.

[37] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," University of Oxford, 2014.

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," Google Inc., University of North Carlina, University of Michigan, 2014.

[39] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," Microsoft Research, 2015.

[40] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," UC Berkeley, 2014.

[41] V. Badrinarayanan, A. Kendall and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," 2015.

[42] F. Yu and V. Koltun, "Multi-Scale Context Aggregation by Dilated Convolutions," Princeton University, Intel Labs, 2015.

[43] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs," Univ. of California, Google Inc., ́Ecole Centrale Paris, 2014.

[44] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature Pyramid Networks for Object Detection," Facebook AI Research, Cornell University, Cornell Tech, 2016.

[45] H. Zhao, J. Shi, X. Qi, X. Wang and J. Jia, "Pyramid Scene Parsing Network," The Chinese University of Hong Kong, SenseTime Group Limited, 2016.

[46] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "DeepLab: Semantic Image

Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," 2016.

[47] G. Lin, A. Milan, C. Shen and I. Reid, "RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation," The University of Adelaide, Australian Centre for Robotic Vision, 2016.

[48] G. Piatetsky, "Machine Learning platforms: Trends and Analysis," 2019. [Online]. Available: https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.html. [Accessed 26 Jun 2019].

[49] J. Hale, "Deep Learning Framework Power Scores 2018," 20 Sep 2018. [Online]. Available: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a. [Accessed 26 Jun 2019].

[50] J. Hale, "Which Deep Learning Framework is Growing Fastest?," 1 Apr 2019. [Online]. Available: https://towardsdatascience.com/which-deep-learning-framework-is-growing-fastest-3f77f14aa318. [Accessed 26 Jun 2019].

[51] "Keras Docs," [Online]. Available: https://keras.io/. [Accessed 28 Jun 2019].

[52] "Machine Learning Cheatsheet," 2017. [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/. [Accessed 28 Jun 2019].

[53] C. S. Sergey Ioffe, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Google, Inc, 2015.

[54] F. Doukkali, "Batch normalization in Neural Networks," 20 Oct 2017. [Online]. Available: https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c. [Accessed 28 Jun 2019].

[55] F. Peccia, "Batch normalization: theory and how to use it with Tensorflow," 16 Sep 2018. [Online]. Available: https://towardsdatascience.com/batch-normalization-theory-and-how-to-use-it-with-tensorflow-1892ca0173ad. [Accessed 28 Jun 2019].

[56] D. Sarkar, "A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning," 15 Nov 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a. [Accessed 28 Jun 2019].

[57] V. Iglovikov and A. Shvets, "TernausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation," Lyft Inc., Massachusetts Institute of Technology, 2018.

[58] M. u. Hassan, «VGG16 – Convolutional Network for Classification and Detection,» 20 Nov 2018. [En línea]. Available: https://neurohive.io/en/popular-networks/vgg16/. [Último acceso: 30 Jun 2019].

[59] Google, Inc., "Google Colaboratory," [Online]. Available: https://colab.research.google.com/. [Accessed 30 Jun 2019].

[60] Nvidia Corporation, "Nvidia Tesla T4 GPU," 2019. [Online]. Available: https://www.nvidia.com/en-gb/data-center/tesla-t4/. [Accessed 30 Jun 2019].

[61] J. Mathew, "Image Segmentation Keras: Implementation of Segnet, FCN, UNet and other models in Keras.," [Online]. Available: https://github.com/jibinmathew69/image-segmentation-keras. [Accessed 19

Jun 2019].

[62] A. Rosebrock, "Intersection over Union (IoU) for object detection," PyImageSearch, 7 Nov 2016. [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. [Accessed 30 Jun 2019].

[63] C. Qian, H. Jiang y T. Liu, «ISIC 2018 – Skin Lesion Analysis,» MTLab, Meitu, Inc, 2018.

[64] N. C. F. Codella, D. Gutman, M. E. Celebi, B. Helba, M. A. Marchetti, S. W. Dusza, A. Kalloo, K. Liopyris, N. Mishra, H. Kittler and A. Halpern, "Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging Collaboration (ISIC)," 2017.

[65] Invicro, LLC, "Rhesus Teeth Segmentation by Use of Multi-Atlas Library," [Online]. Available: https://www.invicro.com/rhesus-teeth-segmentation-by-use-of-multi-atlas-library/. [Accessed 30 Jun 2019].

# ANEXO B: ÍNDICES DE FIGURAS

### Tablas



### Figuras

# ANEXO C: DETALLES DE IMPLEMENTACIÓN

En este anexo se recoge el código de las funciones adicionales utilizadas en el sistema de segmentación desarrollado en este proyecto, además del código del propio modelo que se puede encontrar en la sección 2.3.3.

### Lectura de las imágenes y máscaras

```
import cv2
import numpy as np

def getImage(path, width, height, ordering):
    try:
        img = cv2.imread(path, 1)  #rgb
        img = cv2.resize(img, ( width , height ))
        img = img/255.0
    except Exception:
        print(path)
        print("Unexpected error:", sys.exc_info()[0])
        img = np.zeros((  height , width  , 3 ))

    if ordering == 'channels_first':
        img = np.rollaxis(img, 2, 0)

    return img

def getSegmentation(path, width, height, ordering):
    seg_labels = np.zeros((height, width, 1))
    try:
        img = cv2.imread(path, 0)  #grayscale
        img = cv2.resize(img, ( width , height ))
        img = img/255.0
        seg_labels[: , : , 0 ] = (img >0.5 ).astype(int)
    except Exception:
        print("Unexpected error:", sys.exc_info()[0])

    if ordering == 'channels_first':
        seg_labels = np.rollaxis(img, 2, 0)

    return seg_labels
```

### Generador de imágenes para entrenamiento

```
import glob
import itertools
import numpy as np

def imageGenerator( images_path, segs_path,  batch_size, height, width, ordering):
    assert images_path[-1] == '/'
    assert segs_path[-1] == '/'

    images = glob.glob(images_path + "ISIC_*.jpg")
    images.sort()
```

```
    segmentations  = glob.glob(segs_path + "ISIC_*.png")
    segmentations.sort()

    assert len(images) == len(segmentations)

    zipped = itertools.cycle( zip(images,segmentations) )
    while True:
        X = []
        Y = []
        for _ in range(batch_size):
            im , seg = next(zipped)
            X.append( getImage(im, width, height, ordering) )
            Y.append( getSegmentation(seg, width, height) )

        yield np.array(X) , np.array(Y)
```

### Entrenamiento

```
import os
import matplotlib.pyplot as plt
import pandas as pd
import datetime
from unet_models import Unet
from utils import imageGenerator

# ################################## PARAMETERS ###################################
# ----------------- General options
validate = True
save_model = True
save_log = True
save_history = True
DIM_ORDER = "channels_last"

# ----------------- Train parameters
train_images_path = "data/train/input/"
train_segs_path = "data/train/groundtruth/"
train_size = 2194
batch_size = 2
epochs = 50
steps = train_size/batch_size
train_verbose = 1;

# ----------------- Validation parameters
val_images_path = "data/validation/input/"
val_segs_path = "data/validation/groundtruth/"
val_size = 400
val_batch_size = 50
val_steps = val_size/val_batch_size

# ----------------- Model parameters
height = 128
width = 128
vgg_th_weights_path = "weights/vgg16_weights_th_dim_ordering_th_kernels_notop.h5"
vgg_tf_weights_path = "weights/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5"
save_model_path = "checkpoint/vgg_model"
```

```
# ############################## MODEL CREATION ###################################
if DIM_ORDER=="channels_first":
  vgg_weights_path = vgg_th_weights_path
else :
  vgg_weights_path = vgg_tf_weights_path

m = Unet(vgg_weights_path, freeze_vgg=True, input_height=height, input_width=width,
         image_ordering=DIM_ORDER)
m.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])


# ############################## TRAINING #########################################
# -------- Training and validation
start = datetime.datetime.now()
Gtrain  = imageGenerator(train_images_path, train_segs_path, val_batch_size,
                         height, width, DIM_ORDER)
if not validate:
  history = m.fit_generator(Gtrain, steps, epochs=epochs, verbose=train_verbose)
else:
  Gval  = imageGenerator(val_images_path, val_segs_path, val_batch_size,
                         height, width, DIM_ORDER)
  history = m.fit_generator(Gtrain, steps, validation_data=Gval, validation_steps=val_steps,
                            epochs=epochs, verbose=train_verbose)
end = datetime.datetime.now()

# ------- Save weights, model, performance...
dte = end.strftime("%Y-%m-%d_%H%M")
if save_model:
  m.save( save_model_path + ".%s.acc%d.h5" %(dte, history.history.get('acc')[-1]*100))

if save_log:
  dur = (end-start).total_seconds()
  # Simulation logs: ["date","epochs","train_size","val_size","acc","val_acc","time"]
  log = [ [end.strftime("%d/%m/%Y %H:%M"), epochs, train_size, val_size,
           history.history.get('acc')[-1], history.history.get('val_acc')[-1],
           "%dh %dm" %(dur/3600, dur%3600/60)] ]
  pd.DataFrame(log).to_csv("records/records.csv", mode="a", index=False, header=False)

if save_history:
  # History file
  history_df = pd.DataFrame.from_dict(history.history)
  history_df.to_csv("records/history.%s.csv"%dte, mode="w+", index=False, header=True)

  # Accuracy plot
  x = np.arange(1, epochs+1, 1)
  plt.figure(figsize= (15,5))
  plt.subplot(121)
  plt.plot(x, history.history['acc'])
  plt.plot(x, history.history['val_acc'])
  plt.title('accuracy')
  plt.xlabel('epoch')
  plt.xticks(x)
  plt.legend(['train', 'test'], loc='upper left')

  # Loss plot
  plt.subplot(122)
  plt.plot(x, history.history['loss'])
  plt.plot(x, history.history['val_loss'])
  plt.title('loss')
  plt.xlabel('epoch')
  plt.xticks(x)
  plt.legend(['train', 'test'], loc='upper left')

  plt.savefig("records/history.%s.png" %dte)
```

### Cálculo de métricas

```
import math
import numpy as np

def model_metrics(y_true, y_pred):
    targ = y_true[:,:,0]
    predict = y_pred[:,:,0]
    npixels =  targ.size

    # tp: true=1 & pred=1
    tp = np.sum(targ*predict)

    # fp: true=0 & pred=1
    fp = np.sum((predict-targ)==1)

    # fn: true=1 & pred=0
    fn = np.sum((targ-predict)==1)

    # tn: true=0 & p=0
    tn = npixels-tp-fp-fn

    accuracy = (tp+tn)/(tp+fp+tn+fn)
    if math.isnan(accuracy):
        accuracy=0.0

    precision = tp/(tp+fp)
    if math.isnan(precision):
        precision=0.0

    recall = tp/(tp+fn)
    if math.isnan(recall):
        recall=0.0

    specificity = tn/(tn+fp)
    if math.isnan(specificity):
        specificity=0.0

    dice = 2*tp/(2*tp+fn+fp)
    if math.isnan(dice):
        dice=0.0

    jaccard = tp/(tp+fn+fp)
    if math.isnan(dice):
        jaccard=0.0

    jaccard_threshold = jaccard
    if jaccard_threshold<0.65:
        jaccard_threshold = 0.0

    metrics_array = [accuracy, precision, recall, specificity, dice,
                     jaccard, jaccard_threshold]
    return metrics_array
```

### Predicción y obtención de métricas

```
from keras.models import load_model
import matplotlib.pyplot as plt
import pandas as pd
import datetime
import numpy as np
import sys
from utils import getImage, getSegmentation, model_metrics


# ################################## PARAMETERS ##################################
DIM_ORDER='channels_last'
save_predictions = True
save_metrics = True

# ----------------- Test parameters
test_images_path = "data/validation/input/"
test_segs_path = "data/validation/groundtruth/"
test_size = 400
test_results_path = "data/results/"

# ----------------- Model parameters
load_model_path = "checkpoint/vgg_model.2019-06-29_1114.acc98.h5"
width = 128
height = 128


# ############################## MODEL LOADING ###################################
m = load_model(load_model_path)


# ############################## PREDICTION ######################################
images = glob.glob( test_images_path + "*.jpg"  )
images.sort()

dte = datetime.datetime.now().strftime("%Y-%m-%d_%H%M")
if save_predictions:
  output_path = test_results_path + "%s/" %dte
  try:
    os.mkdir(output_path)
  except OSError:
    print ("Creation of the directory %s failed" %output_path)
  else:
    print ("Successfully created the directory %s" %output_path)

headers = ['accuracy', 'precision', 'sensitivity', 'specificity', 'dice coeff',
           'jaccard', 'jaccard threshold']
index = []
metrics_list = []
i=0
start = datetime.datetime.now()
for imgName in images:
  # Predict segmentation mask
  X = getImage(imgName, width, height, DIM_ORDER)
  pr = m.predict( np.array([X]), verbose=0)[0]
  pr = (pr>0.5).astype(int)

  # Get ground truth and calculate metrics
  maskName = imgName.replace(test_images_path,test_segs_path).
                     replace(".jpg", "_segmentation.png")
  Y = getSegmentation(maskName, width, height, DIM_ORDER)
  metrics = model_metrics(Y, pr)
  metrics_list.append(metrics)
```

```
  # Save prediction image
  if save_predictions:
    index.append(imgName.replace(test_images_path, ""))
    pr_img = pr*255
    outName = maskName.replace(test_segs_path, output_path)
    cv2.imwrite(outName, pr_img)

  # Progress
  i=i+1
  print("\r", "Iteration: %d/%d" %(i,test_size), end="")
  if i%50==0:
    end = datetime.datetime.now()
    dur = (end-start).total_seconds()
    print('\t %d s' %dur)
    print('\t', headers)
    print('\t', np.mean(metrics_list, axis=0))
    start = datetime.datetime.now()


# Calculate mean
metrics_df = pd.DataFrame(metrics_list, index=index, columns=headers)
metrics_total = metrics_df.mean(axis=0, numeric_only=True)
print('\nMétricas finales:')
print(metrics_total)

# Save to csv file
if save_metrics:
  metrics_df.loc['MEAN'] = metrics_total
  metrics_df.to_csv("records/test_metrics.%s.csv"%dte, mode="w+", index=True, header=True)
```