Trabajo Fin de Grado
Ingeniería de la Telecomunicación

Implementación basada en aprendizaje profundo (*Deep Learning*) para la segmentación de lesiones pigmentadas de la piel

Autor: Javier Guillén Cano
Tutoras: Begoña Acha Piñero
Carmen Serrano Gotarredona

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

Sevilla,2020

Departamento de Teoría de la Señal y Comunicaciones

Portada

Implementación basada en *Deep Learning* para la segmentación de lesiones pigmentadas de la piel

Trabajo Fin de Grado
Ingeniería de Telecomunicación

Implementación basada en aprendizaje profundo (*Deep Learning*) para la segmentación de lesiones pigmentadas de la piel

Autor:
Javier Guillén Cano

Tutoras:
Begoña Acha Piñero

Carmen Serrano Gotarredona

Catedráticas

Dep. Teoría de la Señal y Comunicaciones

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Implementación basada en aprendizaje profundo (*Deep Learning*) para la segmentación de lesiones pigmentadas de la piel

Autor: Javier Guillén Cano
Tutoras: Begoña Acha Piñero y Carmen Serrano Gotarredona

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Tras unos largos años de mi vida, de esfuerzo, rutina, perseverancia y sacrificio, he conseguido plasmar en este trabajo todos aquellos conocimientos que he ido adquiriendo durante todo este tiempo.
Todo el apoyo y la ayuda recibida se la quiero agradecer a todos y cada uno de los docentes que me han formado en la escuela, de manera especial a mis tutoras del TFG, Begoña y Carmen, que me han estado aguantando durante toda la realización de este trabajo, y que agradezco mucho todo su tiempo dedicado. Todo esto no podría haber sido posible sin su ayuda. Agradecer también, el apoyo de mi familia, que nunca me ha faltado, sin olvidar a cada uno de mis amigos que se han preocupado por mí, apoyándome en las malas y en las buenas.
Por todo esto y mucho más, gracias a todos.

Javier Guillén Cano
Sevilla,2020

Resumen

Las estrategias de detección precoz del melanoma maligno, es un punto de gran importancia para prevenir el riesgo del mismo. Son muchas las técnicas empleadas para ello como el mapeo corporal total, la dermatoscpía o el seguimiento dermatoscópico digital, pero no es suficiente, ya el número de muertes por este cáncer aumenta cada año.

La motivación principal de este trabajo ha sido la búsqueda de una red neuronal convencional para la segmentación de imágenes que contienen lesiones pigmentadas en la piel, mediante algoritmos de *Deep Learning.* Esta exploración se ha llevado a cabo indagando en las redes neuronales usadas en segmentación, realizando pruebas sobre una red convolucional compleja empleando la técnica de *transfer learning*. Se ha desarrollado una red SegNet [1], basada en segmentación semántica, algoritmo de *Deep Learning*, que asocia una etiqueta o categoría a cada pixel en una imagen.

Abstract

Strategies for early detection of malignant melanoma are of great importance to prevent the risk of it. There are many techniques used for this purpose, such as total body mapping, dermatoscopy or digital dermatography monitoring, but they are not enough, since the number of deaths from this cancer increases every year.

The main reason for this project has been the search for a conventional neural network for the segmentation of images containing pigmented lesions in the skin, using Deep 4egroing algorithms. This exploration has been carried out by investigating the neural networks used in segmentation, testing a complex convolutional network using the technique of transfer learning.

A SegNet [1] has been developed, based on semantic segmentation, an algorithm created by Deep Legroing, which associates a label or category to each pixel in an image.

Índice

Tabla de contenido

Índice de figuras

Índice de Tablas

Acrónimos

ACS	American Cancer Society
ISIC	International Skin Imaging Collaboration
AI	Artificial Intelligence
CNN	Convolutional Neural Network
RNA	Redes Neuronales Artificiales
ROI	Region Of Interest
ISIC	International Skin Imaging Collaboration
GPU	Graphics Processing Unit

1. Introducción

El uso del conocimiento y la investigación, originó el desarrollo de la tecnología. Desde mediados del siglo XX [2] se han logrado grandes avances científico-médicos.

En 1956, John McCarthy, al cual se le denomina padre de la inteligencia artificial, definió a la inteligencia artificial como *"La ciencia e ingenio de hacer máquinas inteligentes, especialmente programas de cómputo inteligentes"* [2].

Este hito junto con la incidencia de la tecnología y la informática en la medicina, originaron nuevos avances en los campos de la biomedicina, la biotecnología y la ingeniería de la salud. Algunos de estos grandes avances han sido las técnicas de diagnóstico basadas en aprendizaje automático y la implantación de redes neuronales en algoritmos para la detección de enfermedades como el cáncer de piel.

En este apartado, se introduce la motivación médica del proyecto. La segmentación de lesiones y una breve introducción del melanoma. Por último, se presenta la organización precursora del ISIC (*International Skin Imaging Collaboration*) *Challenge,* en el cual está basado este proyecto.

1.1. Motivación Médica

El melanoma es un tumor que tiene origen en las células del cuerpo humano llamadas **melanocitos**. Estas son las productoras de la melanina. Este pigmento proporciona color a nuestra piel, al mismo tiempo la protege de la radiación ultravioleta. [3]

Cada año la Asociación Americana del Cáncer(ACS: *American Cancer Society*) trabaja en la investigación y recopilación de los casos sobre la incidencia, mortalidad y la supervivencia del cáncer. Se registran 1.762.450 de nuevos casos de cáncer y 606.880 muertes anuales en todo el territorio de los Estados Unidos a causa de esta enfermedad. Alrededor de un 1,91% de estas muertes son debidas al cáncer de piel o melanoma [4].

Por este motivo, Estados Unidos es uno de los países del mundo que mas invierte e investiga acerca de este cáncer.

1.1.1. Detección del melanoma

La detección temprana del melanoma es un punto clave a la hora de reducir el riesgo del mismo. Tras la detección, se realiza un proceso llamado estadificación, que consiste en averiguar si el cáncer se ha propagado. Es por esto que se puede clasificar el melanoma en 5 fases diferentes, dependiendo del tamaño, grosor, incidencia y expansión del mismo:

- **Etapa 0:** El cáncer está confinado en la epidermis, no se ha propagado a los ganglios linfáticos ni a las partes del cuerpo. Recibe el nombre de melanoma *in situ.*
- **Etapa 1:** El grosor del tumor mide menos de 1mm. Puede o no estar ulcerado. No se ha propagado aún a los ganglios linfáticos ni al resto del cuerpo.
- **Etapa 2:** La única diferencia respecto a la etapa 1, es que el grosor del tumor es mayor, alcanzando entre 1 y 2 mm.
- **Etapa 3:** El tumor alcanza un grosor entre los 2 y 4 mm. Puede estar o no ulcerado. A partir de esta etapa el tumor se extiende por algunos ganglios adyacente, pero sin llegar a propagarse por otras partes distintas del cuerpo.
- **Etapa 4:** En esta etapa el tumor puede llegar a medir más de 4mm de profundidad. Como en el resto de etapas puede o no estar ulcerado. Se puede producir metástasis, provocando que el tumor se propague por ganglios linfáticos distantes o a órganos como pulmones, el hígado o el cerebro. [5]

En la *figura 1.1*, se pueden observar las diferentes etapas del melanoma y el alcance de las mismas. Se observa que en las fases 0 y 1 que el melanoma no traspasa la epidermis (capa rosada). En cambio, en las fases 2, 3 y 4 el melanoma traspasa las capas subcutáneas (capa rosada), propagándose por los nódulos linfáticos (fases 3 y 4), para llegar a la dermis (capa amarillenta). En la fase 4 se puede ver que el melanoma está extendido a la epidermis (capa rojiza), lo que provoca la metástasis. [5]

Figura 1.1:Etapas del melanoma[6]

El sistema usado con mayor frecuencia para la determinación de la etapa en la que se encuentra el melanoma es el sistema TNM:

- **T:** Tumor. Se analiza la profundidad del melanoma y si esta ulcerado.
- **N:** *Lymph Nodes* (Ganglios Linfáticos). Se analiza si se ha diseminado el cáncer a ganglios linfáticos cercanos.
- **M:** Metástasis. Se analiza si se ha diseminado el cáncer a ganglios linfáticos u órganos distantes.

El conocimiento y el correcto uso de los métodos de detección del melanoma, son factores muy importantes para aumentar la probabilidad de supervivencia del paciente. En fases 0, 1 y 2 la tasa de supervivencia relativa a 5 años se encuentra en torno al 98%, en cambio, a medida que el tumor se extiende por los ganglios linfáticos (fase 3), disminuye hasta un 64%. Una vez alcanzada la fase 4, la probabilidad de supervivencia relativa del paciente se reduce a un 23%. [4]

1.1.2. Dermatoscopia

La dermatoscopia es una técnica de diagnóstico en vivo, no invasiva, desarrollada para el estudio de lesiones cutáneas. Se realiza directamente sobre la piel por medio de un instrumento llamado dermatoscopio, que proporciona al especialista un diagnóstico precoz junto con una mejor precisión en la detección de lesiones hiperpigmentadas, potencialmente malignas como lo es el melanoma.[3b]

En la *figura 1.2*, se observan dos ejemplos de los dermatoscopios disponibles actualmente en el mercado. Por un lado, se muestra el dermatoscopio de contacto o inmersión. Este tipo de dermatoscopios requieren de la aplicación de un bálsamo directamente sobre la lesión para su posterior observación y permiten un aumento de

x10 de la lesión pigmentada. Por otro lado, se observa el dermatoscopio de luz polarizada. En estos instrumentos, al contrario que en los anteriormente explicados, no se necesita la aplicación del bálsamo ni el contacto directo con la lesión pigmentada. Este tipo de dermatoscopios permiten identificar estructuras más profundas como vasos sanguíneos y el colágeno. Ambos instrumentos permiten la detección de lesiones y estructuras superficiales como queratosis seborreicas, los *nevos* o áreas de regresión. [7]

Figura 1.2: Dermatoscopio de inmersión (Izq.) y dermatoscopio de luz polarizada cruzada (Drcha.)[8]

En el ámbito de la dermatoscopia, el proceso más extendido es el denominado "*2-step process"* o proceso de los dos pasos. Primero, se busca la diferenciación entre lesiones cutáneas melanocíticas o no melanocíticas. En segundo lugar, se lleva a cabo la diferenciación de las neoplasias melanocíticas benignas del melanoma. Para la realización de este segundo paso, se pueden utilizar diferentes tipos de algoritmos. [9] En la *Figura 1.3*, se muestra el proceso de los dos pasos.

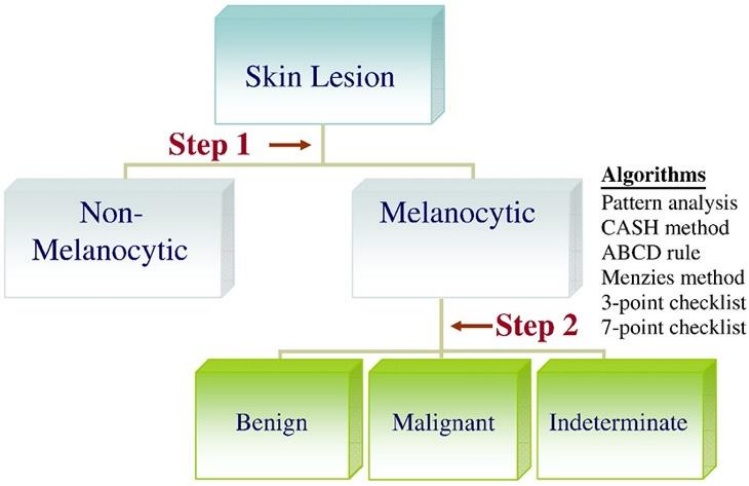

Figura 1.3: Representación gráfica del proceso "2-steps" [9]

El desarrollo de la tecnología dio origen a la dermatoscopia digital. Los dermatoscopios digitales permiten una mejor observación de las lesiones pigmentadas. Se pueden apreciar con diferentes aumentos y luces, tanto polarizadas como no polarizadas, y permiten ser almacenadas para su posterior estudio y evaluación.

1.2. Segmentación de lesiones

El objetivo de este trabajo es, la segmentación de imágenes dermatoscópicas de lesiones pigmentadas. En este apartado, se explica en qué consiste la segmentación de imágenes y sus aplicaciones. En la siguiente sección, se introducen las técnicas usadas en segmentación y el estado del arte sobre las lesiones pigmentadas.

1.2.1. Introducción a la segmentación

El análisis de imágenes es una extracción de información significativa en formato de dos o tres dimensiones, a través del análisis visual como digital (procesamiento digital de imágenes).

Hay 4 fases en el análisis de imágenes. Una de estas fases es la segmentación de imágenes. Se explican a continuación y aparecen representados en la *figura 1.4*[10]:

- **Clasificación**: Se categoriza la imagen con el objetivo de predecir las categorías o conceptos más relevantes. *Figura 1.4(a).*
- **Segmentación**: Divide la imagen en regiones bien definidas o grupos de píxeles, para su identificación y clasificación. *Figura 1.4(b).*
- **Detección de Objetos**: Detecta objetos dentro de una imagen. Dibuja un rectángulo a su alrededor y los clasifica. Por ejemplo, una persona, un perro o un gato. *Figura 1.4(c).*
- **Instanciación**: Eleva la segmentación a un nivel superior de detalle. Identifica diferentes individuos de una misma categoría, como personas o vehículos. *Figura 1.4(d).*[10]

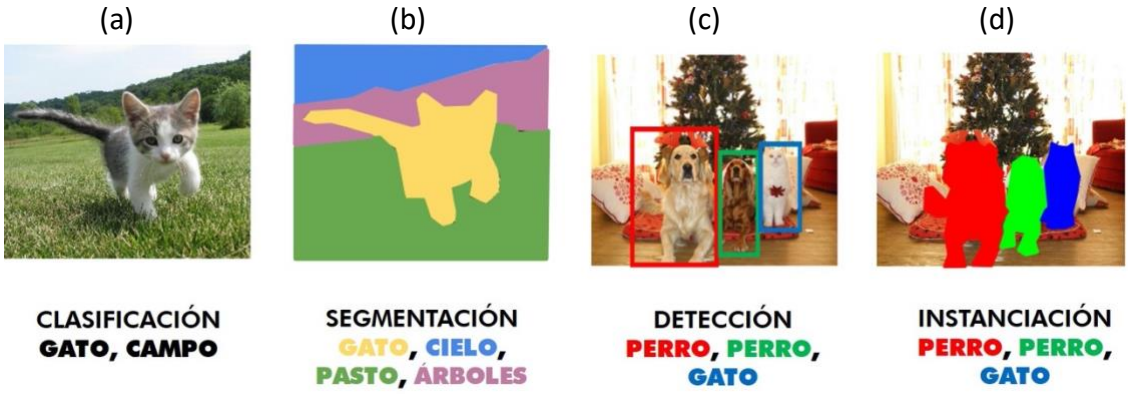

Figura 1.4: Niveles de análisis de las imágenes[10]

La segmentación de imágenes consiste en dividir y separar una imagen en regiones para simplificar su análisis. Las regiones representan objetos o partes de objetos, que conforman conjuntos de píxeles o "superpíxeles" que pertenecen a una misma región/clase.

Tras la segmentación, se logra conocer qué píxel pertenece a cada objeto y analizar sus propiedades como pueden ser el tamaño, ubicación, forma, composición, textura, amplitud de luminosidad, colores, bordes etc.

La segmentación, delimita regiones de interés (RDE), para localizar estructuras anatómicas o regiones patológicas; la extracción de rasgos; medición de datos; la visualización de imágenes o la compresión de las mismas. [11]

La segmentación de imágenes se puede llevar a cabo de 3 formas diferentes según el grado de interacción humana:

- **Segmentación manual**: En este tipo de segmentación, se hace uso de la información de la imagen y del conocimiento adicional.

- **Segmentación semiautomática**: La intervención humana es necesaria para inicializar el método, chequear o corregir los resultados.

- **Segmentación automática**: La segmentación no necesita ningún tipo de interacción humana. Todo el proceso lo realiza un ordenador.[12]

En este proyecto, la segmentación realizada es automática, ya que el proceso, una vez entrenada la red, no necesita la intervención humana.

En resumen, el objetivo de la segmentación es simplificar o modificar 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

El rápido desarrollo de las tecnologías de digitalización de imágenes médicas está revolucionado la medicina. La información que contienen las imágenes se procesa a través de computadoras muy potentes que permiten mejorar su calidad.

Esta mejora de la calidad, posibilita realzar y segmentar las zonas de la imagen en diferentes partes constituyentes, facilitando el diagnóstico por imágenes. Permite a los profesionales de la salud observar lesiones para buscar indicios de enfermedades o afecciones médicas. [14]

El diagnóstico de imágenes médicas se realiza en 4 pasos: [15]

1. **Pre-procesamiento**: Resalta determinadas características de la imagen y elimina el ruido presente en la imagen.
2. **Segmentación**: Detecta diferentes segmentos o regiones de la imagen.
3. **Búsqueda de características**: Detección de características, mediante la localización y clasificación de patrones que sean de interés para el diagnóstico.
4. **Clasificación**: En base a las características obtenidas en el paso anterior, se clasifican los objetos segmentos en los tipos o clases a diferenciar.

El uso de la segmentación en aplicaciones médicas es la combinación de dos tareas complementarias: el reconocimiento y la delineación. En aplicaciones médicas, el reconocimiento, es una tarea de alto nivel que consiste en la determinación a grandes rasgos del lugar o ubicación de la zona de interés estudiada. La delineación es una tarea de bajo nivel, que determina de manera precisa la extensión espacial de la zona de interés y su composición [15].

En nuestro trabajo emplearemos estas dos tareas complementarias para evitar elementos que no afecten a nuestra zona de interés, como vello, piel sana u objetos. De esta forma focalizaremos la segmentación únicamente en aquellas zonas donde exista lesión pigmentada. Podemos observarlas en la *figura 1.5*, un ejemplo de una imagen y su correspondiente segmentación, pertenecientes al *dataset* del ISIC *archive.*

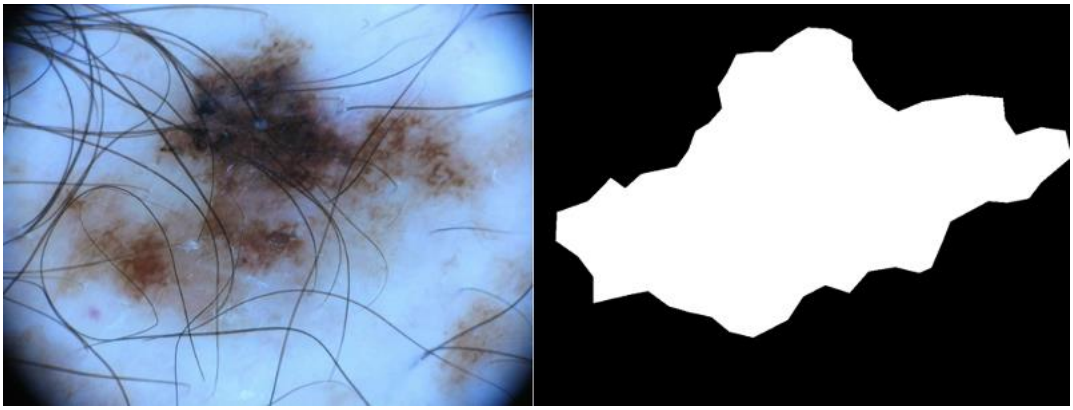

Figura 1.5: Ejemplo de segmentación de lesión pigmentada[16]

1.3 International Skin Imaging Collaboration (ISIC)

La *International Skin Imaging Collaboration* (ISIC) es una asociación entre el mundo académico y la industria destinada a facilitar la aplicación de las imágenes digitales de la piel para ayudar a reducir la mortalidad por melanoma. [16]

El objetivo que persigue consiste en el desarrollo y ampliación de un archivo de acceso público y código abierto llamado (*ISIC Archive*), que contiene imágenes de lesiones pigmentadas de la piel. Se pretende hacer accesible estas imágenes, para poder educar a profesionales y al público en general en el reconocimiento del melanoma; de esta forma, ayudar directamente en el diagnóstico del melanoma a través de la teledermatología, facilitando la toma de decisiones clínicas y el diagnóstico automatizado.

1.3.1. El desafío del ISIC: Análisis de lesiones cutáneas para la detección del melanoma

Este proyecto es una implementación del ISIC c*hallenge* de 2018. El reto propuesto por la ISIC tiene como objetivo reducir las muertes relacionadas por el melanoma. Evitando biopsias innecesarias, gracias a la mejora de la eficiencia y la precisión en la detección temprana del melanoma.

El desafío de 2018 consistió en 3 tareas:

1. **Segmentación de lesiones:** El objetivo de esta primera tarea consistía en segmentar (calcular los bordes de la lesión) las imágenes dermatoscópicas de la base de datos.
2. **Clasificación de características:** Detección automática de características dermatoscópicas, mediante localización y clasificación de patrones.

3. **Clasificación de enfermedades:** Clasificación de enfermedades en diferentes categorías, entre las cuales se incluye el melanoma, mediante predicciones automatizadas.

Para cada tarea, los datos son imágenes y sus correspondientes características, divididos en 3 conjuntos de datos: entrenamiento, validación y test, y almacenados en la base de datos de ISIC. [16]

Con el paso de los años, el éxito y reputación de este *challenge* ha ido en aumento, tanto en número de participantes como en la calidad de los resultados asociados. Esta mejora, es notable en todas las tareas. En la tarea 2, han aumentado los patrones de lesión presentes en cada imagen, mientras que en la tarea 3 el número de categorías para la clasificación de lesiones también se ha incrementado.

1.3.2 La base de datos de ISIC

El archivo ISIC o ISIC *archive* es la colección más extensa disponible para el público, de imágenes dermatoscópicas de lesiones cutáneas, recopiladas y catalogadas por los principales centros clínicos a nivel internacional.

En los orígenes, en 2016, el archivo de la ISIC constaba de 900 imágenes, año tras año se han ido añadiendo imágenes a este archivo, hasta un total de más de 10.000 imágenes en 2018. En la actualidad contiene más de 30.000 imágenes de lesiones cutáneas, caracterizadas y etiquetadas. El incremento del ISIC *Archive*, ha conseguido proporcionar mejores prestaciones en el rendimiento de los algoritmos creados para cada tarea.

Para la clasificación de las imágenes que forman parte del ISIC *archive*, se utilizan técnicas de *crowdsourcing* supervisadas por expertos en lesiones pigmentadas, con el objetivo de perfeccionar las herramientas de anotación y marcado de imágenes. El ISIC

Archive es accesible de manera pública, pudiendo encontrarse en la página web del ISIC *Challenge* [16].

2. Estado del arte

2.1. Estado del arte

Hoy en día, la segmentación de lesiones es una parte muy importante en muchas tareas de análisis y procesamiento digital de imágenes. Se encuentra integrada en numerosas áreas de investigación médica.

En los próximos apartados, se presentan diferentes técnicas de segmentación, desde las técnicas clásicas pasando por empleo de máscaras, umbralización y técnicas de *clustering*, hasta las técnicas más innovadoras que implementan el uso de redes neuronales convolucionales para el análisis de imágenes dermatoscópicas.

2.1.1. La segmentación clásica

Las técnicas de segmentación de imágenes se basan en una de las dos propiedades básicas de los valores del nivel de gris: discontinuidad y similitud.

- **Discontinuidad**

El método consiste en dividir una imagen en función de los cambios bruscos del nivel de gris. Los algoritmos de discontinuidad detectan los píxeles que tienen distinto valor respecto a los que les rodean aplicando distintas máscaras. En la figura 2.1, se representa la máscara de una imagen. R es, el resultado del sumatorio del producto de los pesos w_i de la máscara multiplicado por cada pixel z_i. Los valores de los pesos w_i tendrán unos valores diferentes en función del tipo de técnica que estemos utilizando. [17]

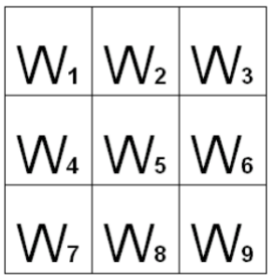

Figura 2.1: Máscara de una imagen [18]

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i$$

Las técnicas más usadas para la detección de discontinuidades del nivel de grises son:

a) **Detección de puntos aislados:** Se aplica la máscara a un pixel y sus vecinos. Supondremos que el nivel de gris es distinto a los que forman las imágenes.

-1	-1	-1
-1	8	-1
-1	-1	-1

Figura 2.2: Máscara para la detección de un punto aislado[18]

b) **Detección de líneas**: Aplicaremos determinadas máscaras a los píxeles. Encontraremos diferentes comportamientos en relación a la dirección de la línea.
Si en cierto punto de la imagen |Ri| > |Rj| para todo i ≠ j, se dice que este punto estará asociado con la máscara de la dirección i. [17]

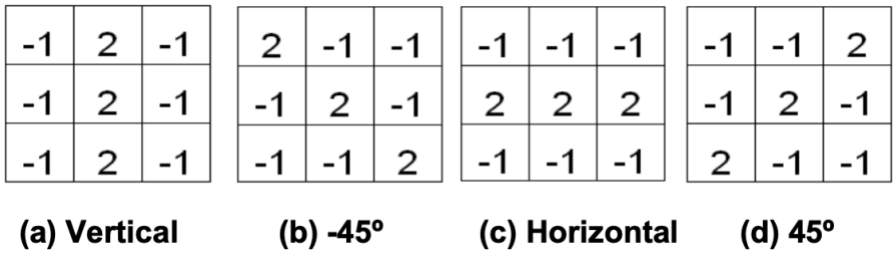

Figura 2.3: Máscaras para la detección de líneas en función de su dirección[18]

c) **Detección de bordes de una imagen**: Consiste en la diferencia de una característica en dos regiones adyacentes, que indica la existencia de un borde.
La detección de bordes se representa en la figura 2.4. La figura de la izquierda muestra la imagen original, en la figura de la derecha se representa la detección de bordes de la imagen original.

Figura 2.4: Ejemplo de detección de bordes[23]

- **Similitud**

a) Se divide la imagen conforme a la búsqueda de zonas que contengan valores similares, en relación a unos criterios prefijados: **Umbralización** (*thresholding*): Es uno de los métodos más simples a la hora de ser implementado. Utiliza la luminancia o intensidad de cada píxel para realizar la segmentación. Las imágenes médicas son bastante heterogéneas, a pesar de ello la diferencia entre la lesión y el fondo de la imagen suele ser clara. En el caso de las lesiones pigmentadas en la piel, estableceremos un patrón binario para la clasificación de los píxeles. Obtendremos un valor '1' en el caso de que la lesión se encuentre presente, por lo contrario, obtendremos un valor '0' cuando no haya lesión, o lo que es lo mismo, sea considerado fondo. Los valores de 0's o 1's asignados a los píxeles depende de si el valor de sus intensidades es mayor o menor que un umbral (*threshold*) fijado. [17][19]

   Los umbrales podrán ser locales o globales. Se puede obtener de diferentes formas. En la mayoría de los casos, en las imágenes de estudio, el umbral no puede obtenerse de manera sencilla. Como solución se acude a un histograma para poder predecir el umbral. [20 [21] En la *figura 2.5*, se puede observar el uso de esta técnica para un *threshold*. A la izquierda la imagen original, y a la derecha el resultado de aplicar un umbral=127 a la imagen original.

Figura 2.5: Segmentación mediante la técnica thresholding[21]

b) **Crecimiento de la región (***Region Grow)*: El objetivo de esta técnica es la partición de la imagen en diferentes regiones que tienen un cierto criterio predefinido que las une entre ellas. Este criterio puede estar basado en información relevante sobre las intensidades y/o bordes de la imagen. Este método requiere la selección de píxeles clave (*seed points*), los cuales son elegidos por el usuario. Estos a su vez extraen los píxeles que estén conectados a ellos, o lo que es lo mismo, vecinos que tengan intensidades similares. Introduciremos en una región los pixeles seleccionados para intentar obtener el menor coste computacional posible.

   El gran inconveniente de esta técnica es la interacción manual para la obtención de los píxeles clave. El ruido es otro problema al cual es sensible esta técnica, causante de que las regiones extraídas tengan agujeros e incluso que se desconecten. [22]

En la figura 2.6 podemos observar el proceso de segmentación de dos imágenes originales (a) y (b). En primer lugar, se realiza una segmentación mediante la técnica de umbralización a las imágenes originales obteniendo las imágenes (c) y (d). Una vez realizada la umbralización, mediante la técnica *Region Growing* obtenemos las imágenes segmentadas (e) y (f) mucho más detalladas que únicamente si aplicamos la técnica de *thresholding*. Por tanto el proceso de la imágenes será en el orden siguiente, usando las diferentes técnicas. (a)->(c)->(e) y (b)->(d)->(f)

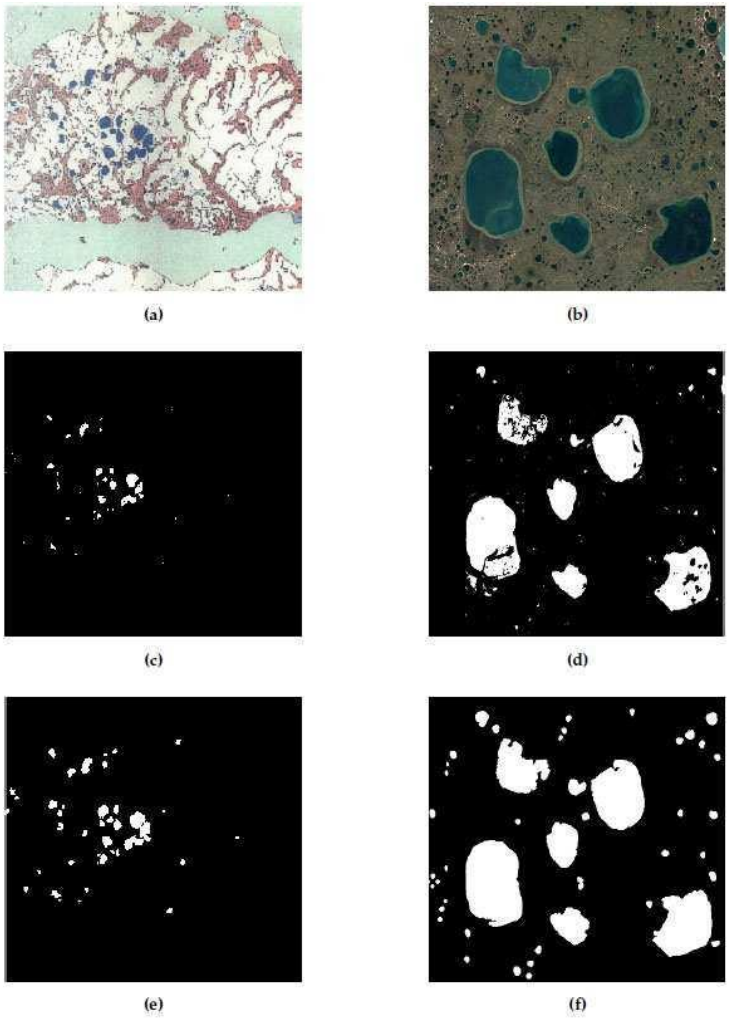

Figura 2.6:Segmentación de imagen mediante Region-Growing[22]

Hasta ahora hemos considerado la determinación de umbrales para una única variable: el nivel de gris de la imagen. En ciertas ocasiones, se dispone de varias variables que caracterizan cada pixel de la imagen. Por ejemplo, las imágenes a color, en las que se disponen de las componentes RGB. En el caso de las imágenes a color, cada píxel se caracteriza por 3 variables, por lo que se puede construir un histograma en 3 dimensiones. La segmentación mediante umbral da lugar al empleo de otra técnica llamada búsqueda de agrupaciones o *clustering.*

- **Clustering**

La técnica de agrupación o *clustering* consiste en agrupar los píxeles conforme a su similitud en las características, en grupos o *clusters*. Si en una imagen se encuentran K *clusters* significativos en el espacio de las variables, podremos segmentar la imagen asignando a cada píxel una de las K etiquetas a todos aquellos píxeles cercanos al *cluster* al que corresponda esa etiqueta.

En el caso de que se conozca el número de *clusters*, podremos emplear el algoritmo LGB o K-*means* (K-medias). Este algoritmo determina mediante iteraciones el centroide de cada uno de los K *clusters* y la partición del espacio de las K zonas, una para cada *cluster*. En la figura 2.7, se puede observar un ejemplo del algoritmo K-medias, agrupando los diferentes pixeles en grupos o *clusters*. [24][25]

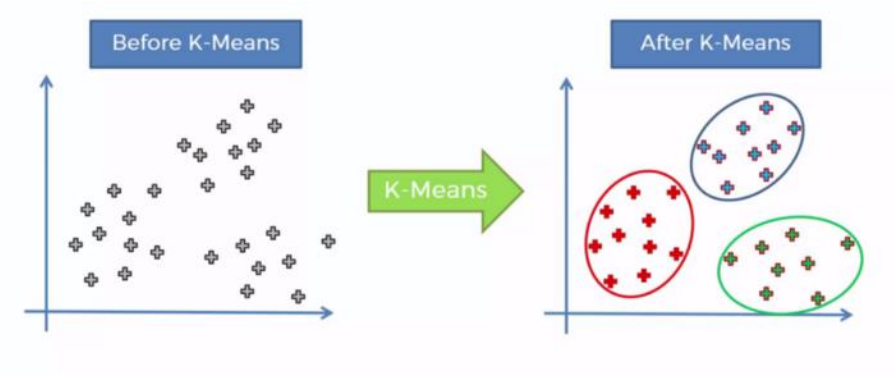

Figura 2.7: Ejemplo del algoritmo K-media[27]

2.1.2. Innovación en la segmentación de imágenes

Las técnicas actuales usadas en segmentación de imágenes dermatoscópicas implementan el uso de redes neuronales convolucionales, un tipo de redes neuronales pertenecientes al aprendizaje profundo o *Deep Learning.*

Se han desarrollado algoritmos y técnicas para segmentar imágenes. Estas mejoras han permitido su aplicación en diferentes campos, principalmente en relación con la inteligencia artificial. En el ámbito clínico se han desarrollado pruebas de diagnóstico, cirugía guiada por ordenador y hasta medida de volúmenes de tejido. A parte de estos usos, ha logrado avances como el sensor de huella digital o el reconocimiento de cara e iris. [25][26]

3. Aprendizaje profundo aplicado a la segmentación de imágenes

La **Inteligencia Artificial** (IA), es una rama de las ciencias computacionales. Estudia modelos de cómputo, que se encargan de realizar actividades humanas en base a dos características fundamentales: el razonamiento y la conducta.[28] La clave de la IA se encuentra en el **aprendizaje**.

Desde mitad del siglo XX la Inteligencia Artificial solo se desarrollaba en laboratorios de investigación o en películas de ciencia ficción. A finales del siglo XX, debido al incremento del poder computacional de las máquinas, origina un crecimiento en su uso y aplicación en la sociedad.

Al no poder pre-programar las máquinas para las infinitas combinaciones de *data sets* de entradas, nos encontramos ante la necesidad de que las máquinas pueden **auto-programarse**. Básicamente consiste en el aprendizaje de las máquinas a partir de su propia experiencia. Es aquí donde nace el **Aprendizaje Automático** o más conocido como ***Machine Learning***, gracias a este aprendizaje automatizado podemos crear aplicaciones que aprendan de forma autónoma gracias a los datos que ingieren.

Todos los grandes como Amazon, Microsoft o Google compiten actualmente por este nuevo mercado. Cada uno de ellos ofrecen sus servicios, dando lugar a plataformas como: Amazon Machine Learning (Amazon), Azure Machine Learning (Microsoft), Google Cloud (Google) y otras muchas como TensorFlow o BigML.

A continuación, vamos a indagar un poco más en el *Machine Learning* aplicado a las ideas en las que se basa el proyecto: segmentación de imágenes de lesiones pigmentadas en la piel. En este 2º capítulo comenzaremos con la explicación del estado del arte en segmentación de lesiones pigmentadas. Explicaremos tanto las técnicas tradicionales como las más innovadoras. Posteriormente, contextualizaremos el proyecto en el marco de la IA y el *Deep Learning*. Finalizaremos esta sección explicando la arquitectura y la implementación del código utilizado, para lograr el objetivo.

3.1 Aprendizaje Automático

La Inteligencia Artificial (IA) se define como la capacidad que posee una máquina o computadora de comportarse de forma inteligente como una mente humana con capacidad para aprender y resolver problemas. [29]

Esta definición no es nueva ya que fue en el año 1950 a raíz de la Prueba De Turing, donde se fijaban los parámetros y habilidades por los cuales a una máquina se le podía otorgar la cualidad de *inteligente.*

Se establecen diferentes tipos o categorías de sistemas de IA según su comportamiento[30]:

a) **Sistemas que piensan como humanos**: Intentan imitar procesos basados en el pensamiento humano. Por ejemplo, las redes neuronales artificiales (las cuales son la base de este proyecto).
b) **Sistemas que actúan como humanos**: Aquellos sistemas cuyo objetivo es imitar el comportamiento humano. Por ejemplo, la robótica.
c) **Sistemas que piensan racionalmente**: Son sistemas de aprendizaje altamente profundo, los cuales intentan imitar el pensamiento lógico y racional del ser humano.
d) **Sistemas que actúan racionalmente**: Va mucho más allá que la inteligencia artificial rudimentaria. Estos sistemas pretenden imitar el comportamiento del ser humano de forma racional, ya que toman decisiones en función de razonamientos.

En la *figura 3.1*, se representan los tres campos de desarrollo de la inteligencia artificial, ordenados en orden de ascendencia, siendo ésta la madre de todos ellos, originada en la década de los 50. A partir de los años 80 nació el aprendizaje automático, con el objetivo de desarrollar técnicas capaces de permitir que las máquinas aprendieran de forma autónoma. Por último, en pleno siglo XXI, surgió un método destinado a la creación, entrenamiento y uso de redes neuronales llamado *Deep Learning*, el cual forma parte del *Machine Learning* y este a su vez de la IA.[31]

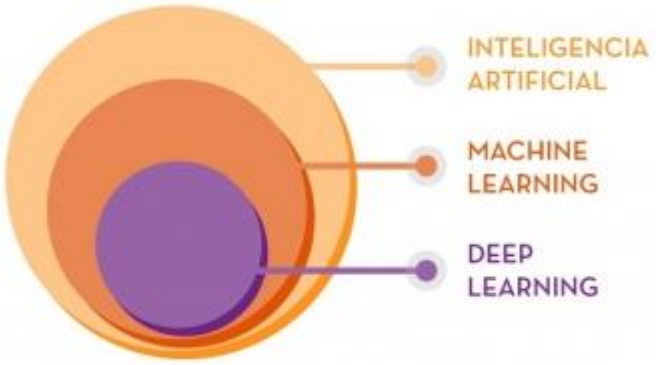

Figura 3.1: Campos de desarrollo de la inteligencia artificial [32]

Un sistema de aprendizaje automático se divide en dos partes. Un sistema de aprendizaje y un sistema de predicción. En la *figura 3.2*, se representan las dos partes diferenciadas y se explica brevemente los procesos que se desarrollan en cada una.

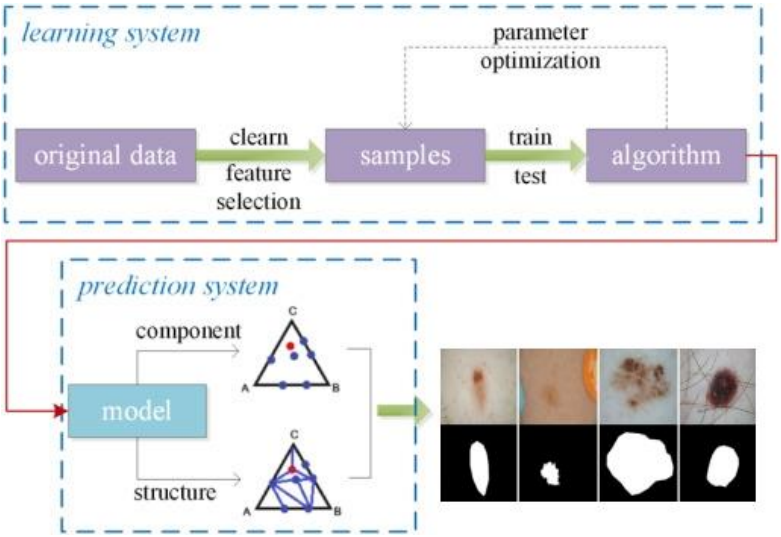

Figura 3.2: Estructura de un problema de aprendizaje profundo[31]

- **Sistema de aprendizaje**: Procesa un conjunto de datos de entrada. Realizando una limpieza de datos, identificando datos incompletos, incorrectos, partes inexactas o irrelevantes, para posteriormente realizar una selección de características y convertir los datos originales en muestras. Para cada muestra genera un resultado conocido denominado etiqueta, también conocido como valor real. Una vez generadas las muestras, se entrenan, validan y testean los datos para generar un algoritmo de aprendizaje que compara la salida esperada (etiquetada) con la que predice el sistema. Para este proceso utiliza una función de coste, encargada de

optimizar los parámetros de la red neuronal, penalizando los fallos de predicción[31].

- **Sistema de predicción:** Emplea un modelo obtenido del sistema de aprendizaje. Predice la salida para nuevos datos, diferentes a los usados en el entrenamiento, de los cuales se desconoce la etiqueta. Ajusta una serie de parámetros**,** de forma que la diferencia entre las etiquetas predichas por el modelo, y las etiquetas reales sea mínima. En el sistema de aprendizaje se lleva a cabo una clasificación por estructuras, componentes, propiedades, etc. [33][31]

3.2. Metodología de modelado de una red neuronal

La creación de una red neuronal, consiste en la estructuración de una arquitectura consistente y coherente formada por neuronas unidas entre sí constituyendo capas neuronales. Estas conexiones entre neuronas, se caracterizan por unos valores llamados pesos, que ponderan los valores que van a intercambiarse entre unas y otras. Estos pesos se ajustan en la fase de entrenamiento. En el siguiente paso, se valida la efectividad del modelo, y finalmente se testean la red con el *dataset* específico para comprobar que su funcionalidad es correcta. En los siguientes apartados se explica en detalle cada uno de estos. [34]

3.2.1. Entrenamiento (*Training*)

En la fase de entrenamiento se proporciona un algoritmo para poder entrenar el modelo o la red que se haya creado. Los parámetros de un modelo de aprendizaje automático son estimados, aproximados y optimizados, usando un conjunto de datos previamente etiquetados con sus respectivas características de dominio y clase. Este conjunto de datos de entrenamiento tiene la propiedad de que posee etiquetas de clase para cada dato.

Existen unos parámetros elegidos y ajustados a la hora de entrenar el modelo que rigen el proceso de entrenamiento, llamados **hiperparámetros.** Por ejemplo, son hiperparámetros, el número de capas ocultas de una red o la cantidad de nodos de cada capa.[35]

3.2.2. Validación (*Validation*)

La fase de validación se encarga de proporcionar un algoritmo que valide la efectividad del modelo mediante la utilización de un conjunto de datos diferente al usado en la fase de entrenamiento. En esta fase los datos también se encuentran etiquetados.

Se toman medidas cuantitativas como la **entropía** o el **error cuadrático medio,** como parámetros de control, los cuales nos darán una estimación del rendimiento de nuestro modelo. En esta fase debemos volver a configurar nuestro modelo para que los resultados sean óptimos y no sea ni poco eficiente, ni **sobreajustado** [36].

3.2.2.1. Overfitting y Underfitting

En la fase de validación pueden ocurrir dos posibles errores en el ajuste del modelo que podemos ver representados en la figura 3.3. En el caso de proporcionar pocas características para entrenar el modelo, el modelo puede identificar de forma errónea datos desconocidos, de forma que tendremos que aumentar la selección de características para reducir este problema de **desajuste** o *under-fitting*.

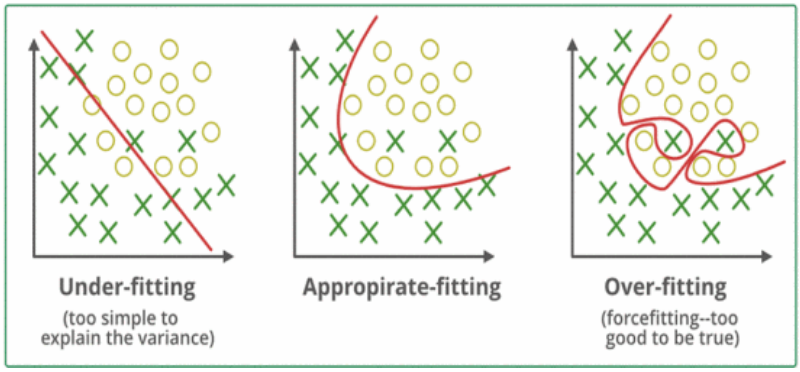

Figura 3.3: Aproximaciones en el ajuste del modelo[37]

En cambio, por otro lado, el **sobreajuste** o *overfitting* ocurre cuando se entrena demasiado el algoritmo de aprendizaje con un conjunto de datos de entrenamiento, provocando que el resultado quede fuera de los rangos establecidos en el entrenamiento, de forma que sea poco fiable y preciso.[38]

3.2.3. Test (*Testing*)

Es la fase final. Consiste en probar si el modelo que hemos entrenado previamente y al cual le hemos aplicado la validación, funciona con un conjunto de datos diferente al que se usó en el entrenamiento, o en la fase de validación.
Hay varias medidas para este propósito, llamadas medidas cualitativas, que son utilizadas para medir el rendimiento del modelo: precisión, sensibilidad, especificidad y exactitud que serán explicadas exhaustivamente más adelante. [36]

Las 3 fases poseen una parte del conjunto de datos o *dataset.* Suelen destinarse un 60% del conjunto total de datos para el entrenamiento, un 20% para la validación y el 20% restante para el test. En la figura 3.4, vemos representados la división del *dataset* para cada una de las 3 fases, y las funciones que cumplen cada una de ellas.

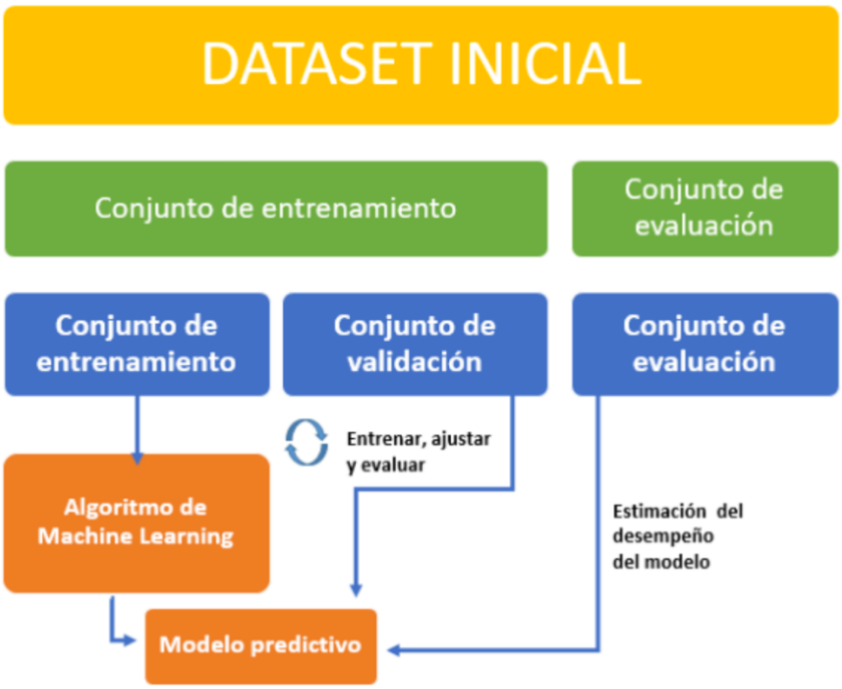

Figura 3.4: Relación entre el dataset y las fases de Entrenamiento, Validación y Test [39]

3.3. Tipos de aprendizaje automático

1. **Aprendizaje supervisado:** En el aprendizaje supervisado los algoritmos trabajan con datos etiquetados (*labeled data).* El objetivo es encontrar una función que, dadas unas variables de entrada (*input data),* les asigne las etiquetas adecuadas de salida. [40] [41]
   Los dos tipos de aprendizaje supervisado son:
   - **Regresión**: Tiene como resultado un número específico. Un ejemplo de regresión se observa en la izquierda de la figura 3.5.
   - **Clasificación**: En este caso, el algoritmo pretende encontrar diferentes patrones. Como objetivo clasifica los elementos en diferentes grupos. Se observa un ejemplo de clasificación en la gráfica de la derecha de la figura 3.5.

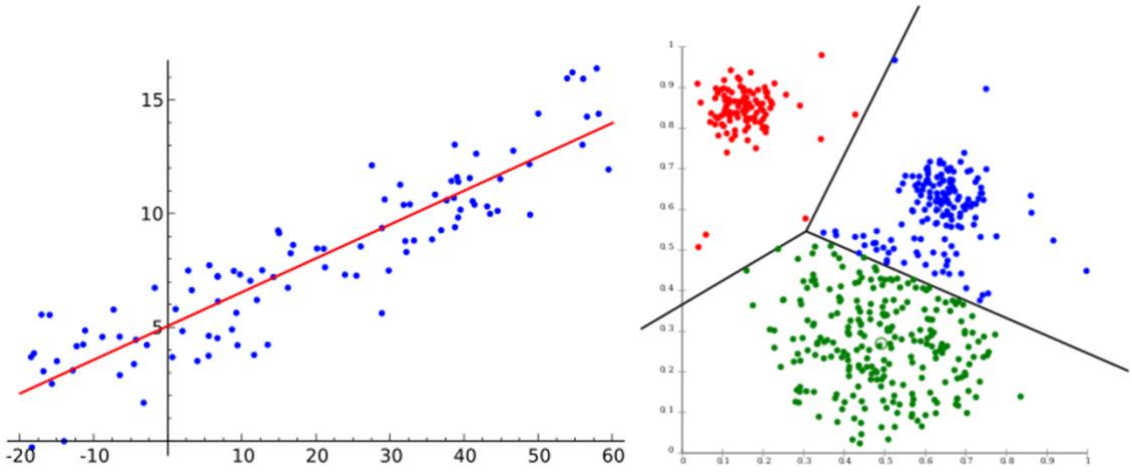

Figura 3.5: Los 2 tipos de aprendizajes supervisados[42]

2. **Aprendizaje no supervisado:** No se conoce las características de las entradas, por lo que será el propio algoritmo el encargado de catalogar el conjunto de datos. .[40][41]

3. **Aprendizaje semi-supervisado:** Combina un pequeño número de datos etiquetados, con un gran número de datos etiquetados. Mejorando así la exactitud del aprendizaje.[40][41]

4. **Aprendizaje por refuerzo** (*Reinforcement Learning*): El propio algoritmo aprende a actuar observando el entorno. Cada acción tiene un impacto en el entorno, que proporciona la información necesaria para aprender a actuar en base a ese comportamiento. .[40][41]

En la figura 3.6, observamos un esquema del funcionamiento del aprendizaje por refuerzo. El agente (*agent*), realiza acciones dentro del entorno (*environment*). El entorno devuelve una recompensa (*reward*), que indica si lo hemos hecho bien o lo hemos hecho mal. Gracias al resultado que devuelve el entorno, el agente, es capaz de interpretar qué acciones realizar en base a los resultados obtenidos. Este conjunto de interacciones acción-recompensa va a permitir al agente aprender y saber qué acciones tomar, según el resultado que desee.

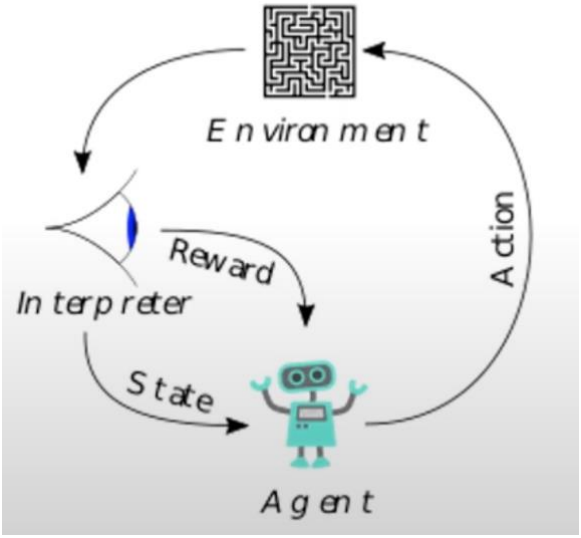

Figura 3.6 : Esquema del funcionamiento del aprendizaje por refuerzo.[43]

5. **Transducción:** Similar al aprendizaje supervisado, con la diferencia de que no construye una función clasificadora como tal, sino que en vez de eso, predice nuevas en base a entradas de entrenamiento, salidas de entrenamiento y nuevas entradas. .[40][41]

6. **Aprender a aprender** (*Learning to learn*): El algoritmo aprende su propio sesgo basado en experiencia previa. .[40][41]

3.4. Redes de neuronas

Las redes de neuronas o redes neuronales son modelos matemáticos basados en el comportamiento fisiológico de las neuronas y en la forma de organización estructural que conforma nuestro cerebro. Su característica principal esta fundamentada en la capacidad del cerebro humano para aprender.[44]

3.4.1. La unidad neuronal básica: El perceptrón

Las neuronas artificiales se diseñan de forma que se asimilen al comportamiento de una neurona cerebral. En la figura 3.7, podemos observar las similitudes entre las neuronas bilógicas y las neuronas artificiales. Tienen un núcleo o nodo que conforma la estructura central, llamado cuerpo, en el cual se realiza el sumatorio de la multiplicación de las **entradas (x_n)** y los **pesos (w_n)** de cada una de las dendritas. Las conexiones entre neuronas artificiales tienen el símil a la sinapsis de las neuronas del cerebro humano. [44]

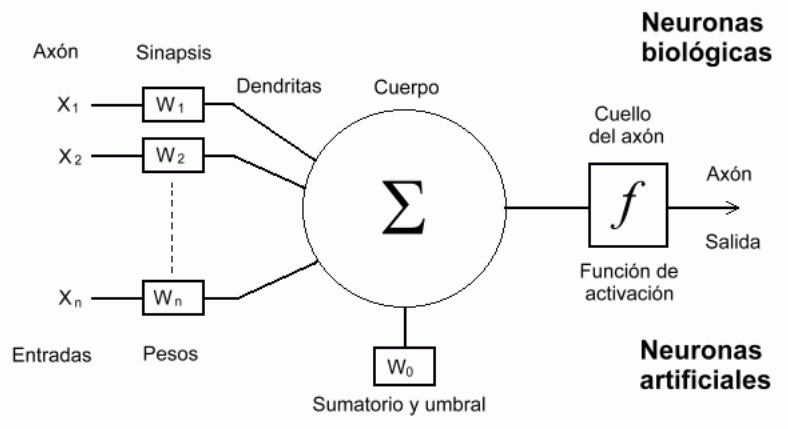

Figura 3.7: Relación entre las neuronas biológicas y las neuronas artificiales [44]

El científico Frank Rosenblatt, fue el encargado de desarrollar el **perceptrón**, unidad más simple de una red neuronal.

El funcionamiento de las neuronas sigue una regla simple. Las neuronas tienen entradas x1, x2,…, que se multiplican por los pesos w1, w2,…, números reales que expresan la importancia de las entradas respecto a la salida. La salida de la neurona será un 0 o un 1. Se determina en función de si la suma ponderada representada a continuación, es mayor o menor que un valor umbral. [45]

$$Y = f\left(\sum_{j=1}^{n} w_j\, x_j\right)$$

El umbral es un parámetro de la neurona, al igual que los pesos, es un número real. La *f*(x), recibe el nombre de **función de activación**. En términos generales la podemos expresar como:

$$f(x) = \begin{cases} 0 \; si \sum_{j=1}^{n} \boldsymbol{w_j\, x_j} \leq umbral \\ 1 \; si \sum_{j=1}^{n} \boldsymbol{w_j\, x_j} > umbral \end{cases}$$

3.4.2. El cerebro humano convertido en red

La búsqueda de modelos computacionales que se asimilan a la estructura del cerebro humano, originó la aparición de las Redes Neuronales Artificiales (RNA). Una RNA, se considera un modelo matemático basado en las actividades teóricas mentales y cerebrales.

Los elementos básicos de un sistema neuronal es la neurona o ***perceptrón***. A su vez estas unidades básicas, se unen formando **capas o niveles**. Deben cumplir una condición; las entradas de la capa provienen de la misma fuente (capa anterior o de los datos de entrada en el caso de ser la primera capa) y las salidas se dirigen al mismo destino (capa posterior).
Las capas a su vez se unen formando una **red**. Cuando configuramos una red junto con un algoritmo, a la que se le añade un conjunto de entradas, esperando un conjunto de salidas, se le denomina **sistema neuronal**. En la figura 3.8 se representa esta jerarquía neuronal.[47]

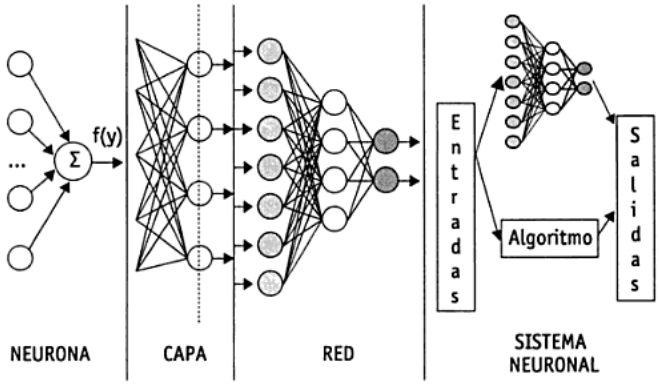

Figura 3.8: Estructura jerárquica de un sistema basado en RNA[46]

En el apartado anterior, se explica el funcionamiento de una neurona artificial. Los valores de entrada de cada neurona llegan al núcleo o nodo a través de las dendritas. Una vez analizadas, se les aplica una función de activación. Una función de activación, cumple la función de devolver una salida, la cual ha sido generada en el interior de una neurona tras haber recibido una o varias entradas. Estas neuronas, a su vez forman capas que conforman la estructura de la red. Cada una de ellas posee una función de activación, que les permite predecir o reconstruir.
En la figura 3.9, se representa un ejemplo de función de activación, en este caso particular se trata de la función de activación *sigmoide.* Es una función de activación peculiar, se suele ubicar en la última capa de la red, ya que clasifica los datos en dos categorías.

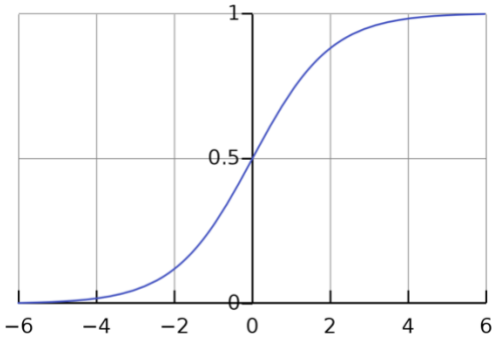

Figura 3.9: Función de activación('sigmoide')[elaboración propia]

El poder y la precisión de la red neuronal se puede amplificar mediante la adición de un número variable de capas ocultas (*hidden layers*) que se encuentran entre la capa de salida y de entrada. [48]

3.4.3. Funcionamiento de una red convolutiva

La convolución, es la operación matemática que consiste en "mezclar" dos funciones. De este resultado obtenemos una tercera función como resultado de la superposición de ambas funciones. A la hora de detectar objetos en una imagen, la convolución resulta muy útil, ya que permite detectar estructuras en una imagen. Cuando una red posee muchas capas, esto es muy útil, ya que estas operaciones se repiten sobre el resultado de la capa previa.

Esta operación matemática, permitió que pudiera resolverse el problema de tratar con un gran número de datos, y originaron la creación de las **redes neuronales convolucionales (CNN)**. Las redes neuronales convolucionales son diseñadas para poder trabajar con imágenes. Gracias a estas redes, se ha conseguido un gran desarrollo en el campo del tratamiento de imágenes. [49]

La enorme cantidad de datos de entrada, provoca que el número de parámetros crezca exponencialmente a medida que nos adentramos en la red. Por ejemplo, una imagen 30 x 30 = 900, junto con su sesgo, equivale a 901 parámetros por una sola neurona de la capa, en el caso de 1 color (escala de grises). Si la imagen fuese a color, necesitamos 3 canales RGB (*red*, *green* and *blue).* En este caso, el número de parámetros sería 30 x 30 x 3 = 2400 con su sesgo, hacen un total de 2401 parámetros para una sola neurona de la capa. [50]

El número de parámetros disminuye inmensamente cuando utilizamos redes CNN, ya que pasa a ser el número de coeficientes del filtro con el que se realiza la convolución. Un solo filtro recorre todos los píxeles de la imagen con los mismo pesos.Las redes toman como entradas los píxeles de una imagen, se les asigna importancia (pesos) a ciertos elementos, permitiendo diferenciarlos unos de otros.

Para alimentar la capa de entrada de nuestra red, es necesario normalizar los valores de los píxeles antes de introducirlos. Se convierten los valores de los píxeles a color de los 3 canales RGB, con valores entre 0 y 255, en valores entre 0 y 1.

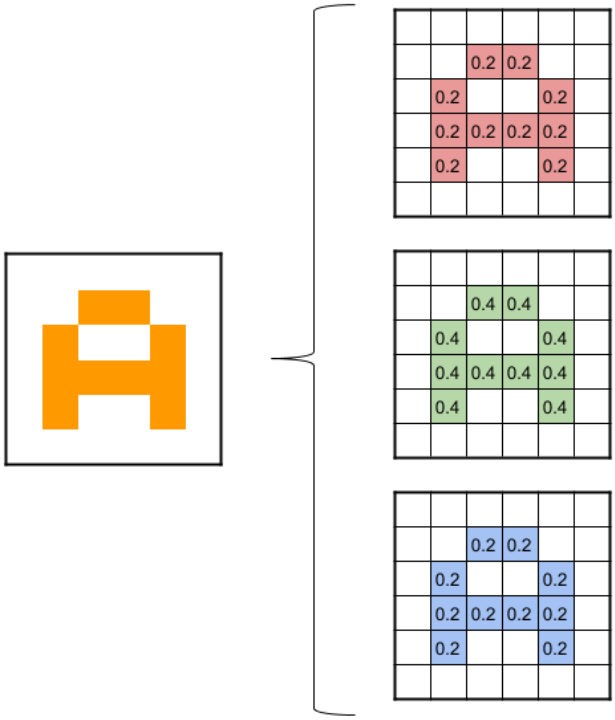

Figura 3.10 : Canales RGB con valores de los pixeles normalizados[51]

3.4.3.1. Capas convolutivas

Lo más distintivo de las CNN, son las capas convolucionales y las capas de *pooling* (agrupación). La función de las capas convolucionales es tomar un grupo de píxeles cercanos de la imagen de entrada, y mediante la utilización de un ***kernel***, realizar un producto escalar. El *kernel*, es un filtro que se le aplica a la imagen para obtener un patrón o características importantes de esta.[52]

El *kernel* recorre todas las neuronas de entrada obteniendo una nueva matriz, que será una de las *hidden layers* o capas ocultas. Si la imagen es de color, tendrá 3 *kernel*s del mismo tamaño que se sumarán a la hora de obtener la imagen de salida.
El resultado de realizar una convolución con 3 *kernels* correspondientes a los 3 canales RGB, se representa en la figura 3.11.[52]

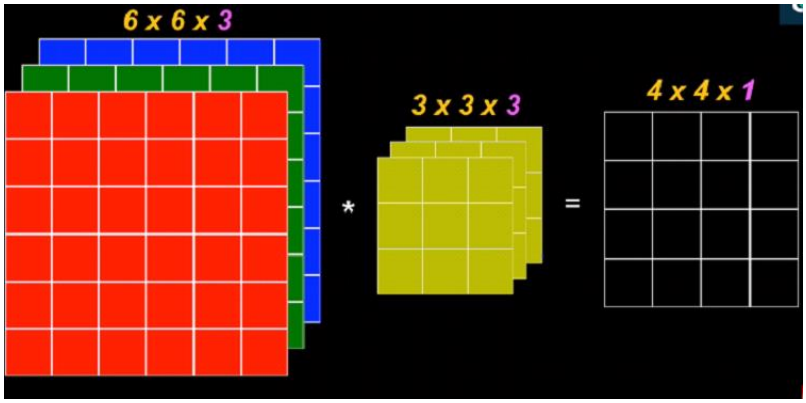

Figura 3.11 : Resultado de la convolución de los 3 canales RGB y sus respectivos kernels[53]

Después de realizar el proceso de convolución, se pueden originar dos tipos de problemas. Estos pueden resolverse con la operación ***zero-padding***. Esta técnica consiste en agregar píxeles de valor 0 alrededor de la imagen original. Los problemas pueden ser los siguientes:

a) La imagen resultante debe tener el mismo tamaño que la original. Se rellena la imagen resultante con ceros hasta que alcance las mismas dimensiones que la imagen original.
b) En ocasiones, la información relevante queda localizada en los bordes, concretamente en las esquinas. Si realizamos una convolución, el filtro pasa más por el centro que por las esquinas, las cuales son las zonas de interés. Por ello añadimos pixeles en los alrededores de la imagen, de forma que la información relevante esté más cerca del centro, consiguiendo que al realizar la convolución, el kernel pase más por las zonas de interés.

Tras la convolución y el uso del *zero-padding*, usaremos una función de activación. En nuestro ejemplo usaremos una de las más conocidas, **ReLu** (*Rectifier Linear Unit).* Obteniendo un **mapa de detección de características**. Este proceso podemos observarlo en la figura 3.12.[54]

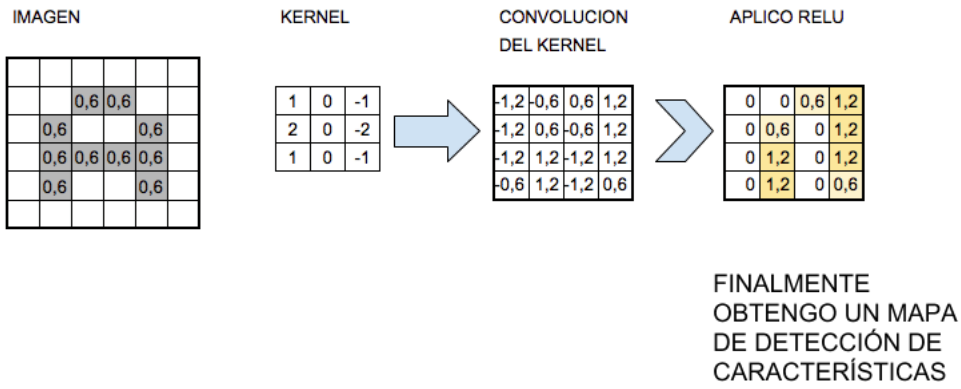

Figura 3.12: Proceso de convolución mediante un kernel y aplicación de la función de activación.[51]

3.4.3.2. Capas de muestreo o *subsampling*

A medida que avanzamos dentro de la red neuronal, no podemos aplicar convoluciones de forma indefinida, ya que el poder computacional que necesitaríamos sería muy grande. Para reducir el tamaño de la próxima capa de neuronas, se realiza un **submuestreo** o ***subsampling,*** de forma que se preserven las características más importantes que detectó el filtro de la anterior capa convolucional. Se realiza la técnica de muestreo más usada, llamada "*Max-pooling*". El *Max-Pooling* localiza el valor máximo entre una ventana o matriz de muestra, y pasa este valor como resumen de características sobre ese área.[55][56]

En nuestro caso como se puede observar en la figura 3.13, realizaremos el "*Max-pooling"* con un tamaño de 2 x 2, reduciendo el tamaño de la salida a la mitad.[33]

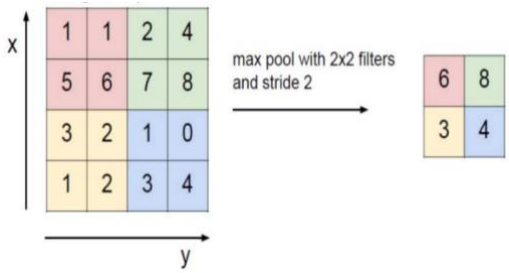

Figura 3.13: Proceso de muestreo mediante la aplicación de Max-Pooling 2 x 2 [57]

3.4.3.3 *Backpropagation* y *Learning rate*

El último paso, consiste en aplanar la última capa a la que le hayamos realizado el proceso de muestreo, convirtiendo una capa tridimensional en una capa plana completamente conectada o *fully connected.*

Una de las grandes ventajas del proceso de aprendizaje en las CNN es, el ajuste del valor de los pesos de los distintos *kernels*. Este ajuste se realiza mediante el algoritmo de ***backpropagation***. Es un método utilizado para calcular el ajuste de los pesos entre conexiones iterando en primer lugar hacia delante (se realiza la predicción) y en segundo lugar hacia atrás (en la que se actualizan los parámetros: retropropagación o *backpropagation*).[58][59]

En cada iteración, se calcula el error total de la red, como el sumatorio de las diferencias del valor real de la salida o *target* ($\hat{y}$) y el valor predicho por la red o *output* (y). Este método se representa en la figura 3.14.

$$\varepsilon = \sum_k (y - \hat{y})$$

En nuestro caso al solo tener una salida, este sumatorio solo lo hacemos de un valor real de salida y un valor predicho por la red. Con ello, se actualiza cada parámetro *w* mediante las siguientes expresiones:

$$w_{kj}^{+} = w_{kj} - \eta \cdot \frac{\partial \varepsilon}{\partial w_{kj}}$$

$$w_{ji}^{+} = w_{ji} - \eta \cdot \frac{\partial \varepsilon}{\partial w_{ji}}$$

La derivada parcial del error respecto a cada peso, se calcula realizando la regla de la cadena desde la salida de la red hasta la entrada, de forma que el error es como si se "propagase hacia atrás"[59].

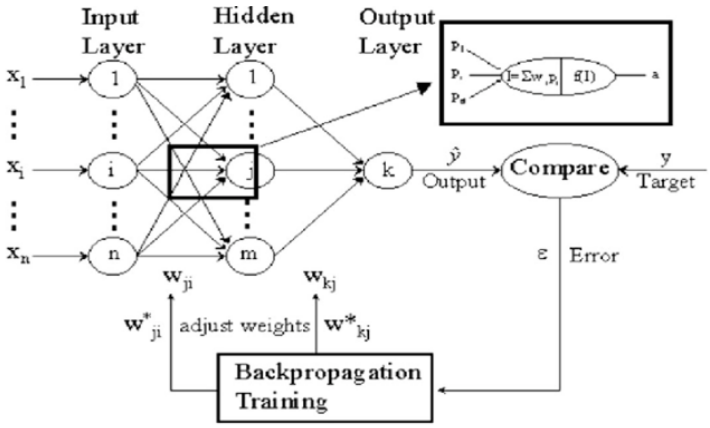

Figura 3.14: Algoritmo de backpropagation en una red de 3 capas[59]

El parámetro η de las ecuaciones descritas anteriormente para la actualización de los pesos, se denomina **ritmo de aprendizaje** o ***learning rate***. Este hiperparámetro juega un papel decisivo en el proceso de entrenamiento de una red neuronal. Controla cuánto cambia el modelo en respuesta al error estimado, cada vez que se actualizan los pesos del modelo. [60]

Es difícil encontrar un valor óptimo para el *learning rate*. Un valor demasiado pequeño puede provocar un proceso de entrenamiento largo disminuyendo la velocidad de convergencia, con posibilidad de que el modelo acabe atrapado en un mínimo local. Por el contrario, un ritmo de aprendizaje demasiado grande puede provocar inestabilidades en la función de error, impidiendo que se alcance la convergencia debido a que se darán saltos alrededor del mínimo sin alcanzarlo.

En la figura 3.15, se representan diferentes ritmos de aprendizaje respecto al tiempo de entrenamiento. En la gráfica de la parte inferior, el eje X representa otro hiperparámetro, la **época**. Una época es un periodo de tiempo conformado por una ejecución hacia adelante y hacia atrás (predicción + retropropagación) de todo el conjunto de datos completo por la red neuronal. El eje Y, representa el **error**. En las figuras del nivel superior, se muestran 4 representaciones de posibles comportamientos del modelo de entrenamiento para diferentes valores de tasas de aprendizaje. La de más a la izquierda, está atrapada ya que tiene una tasa muy baja. Por el contrario, la que está más a la derecha, al tener una tasa de aprendizaje muy alta, impide que se alcance la convergencia. La tasa de aprendizaje suele estar comprendida entre 0.001 y 0.5.[60]

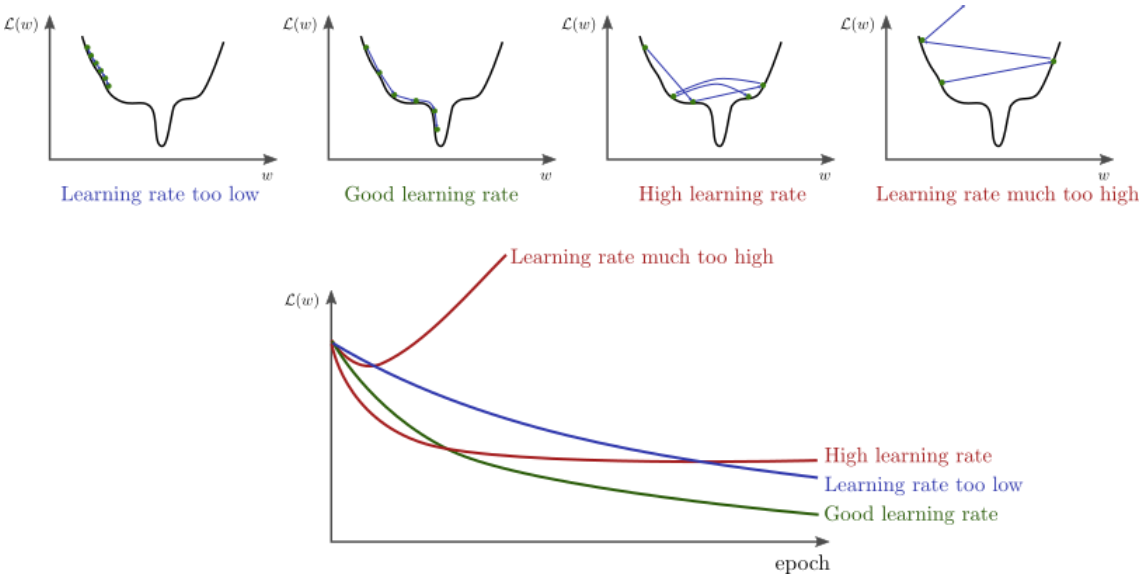

Figura 3.15: Comportamiento de una red ante diferentes ritmos de aprendizaje[61]

Existe otro hiperparámetro en relación con la época, llamado **lote o *batch*.** El b*atch* es el número de muestras de cada conjunto de datos que se hace pasar por la red en cada iteración de una época. Si tenemos un conjunto de datos de 10.000 muestras, y tomamos un *batch* de 500, en cada época, iteraremos hacia adelante y hacia atrás un total de 20 veces.[62]

Obtenemos como resultado final, una red neuronal convolucional, que ofrecerá mayores prestaciones a medida que aumente su número de capas. A medida que aumentamos la profundidad de la red, se pueden afrontar problemas más complejos.

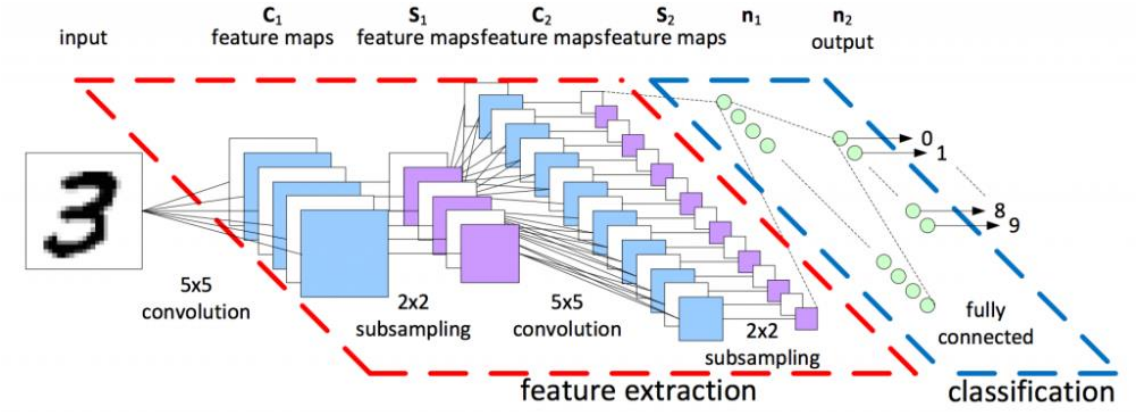

Figura 3.16: Red neuronal convolucional clasificadora[62]

Las redes neuronales están cada vez más implementadas en todos los campos de la ciencia. La mayoría de sus aplicaciones consisten en realizar reconocimiento y clasificación de patrones de forma completamente automatizada.
Aunque sea una tecnología muy avanzada que permite crear redes de un gran número de neuronas, todavía está muy lejos de obtener resultados y velocidades parecidas a las del cerebro humano, el cual posee alrededor de 86 billones de neuronas y una interconectividad muy compleja.

3.5. Dropout

En ocasiones, el entrenamiento de una red puede provocar *overfitting* o sobreajuste. Esto provoca que, a la hora en testear la red, esta no ofrezca buenos resultados, ya que durante la fase de aprendizaje, se ha sobreajustado la red, provocando que no obtenga buenas predicciones cuando el *dataset* difiere en demasía respecto al *dataset* de entrenamiento. Este problema puede solucionarse gracias al método del ***dropout*** o deserción.

En este método, neuronas seleccionadas al azar se eliminan durante la fase de entrenamiento, son "abandonadas" de forma completamente arbitraria, provocando que neuronas que se encuentran debajo de la neurona "abandonada", no reciban contribución de esta, y con ello que no puedan actualizarse sus pesos en el paso hacia atrás. Gracias a esta técnica, se consigue que las neuronas aprendan varias representaciones internas independientes entre ellas, de forma que optimicen mucho más su rendimiento.[63]
En la figura 3.17, se representa una red neuronal de dos formas; a la izquierda, la red con todas sus neuronas activadas y las relaciones entre ellas; y a la derecha se aplica el

método del *dropout*, en el cual algunas neuronas han sido "abandonadas", con ello, las conexiones hacia arriba y abajo, dejan de estar presentes.

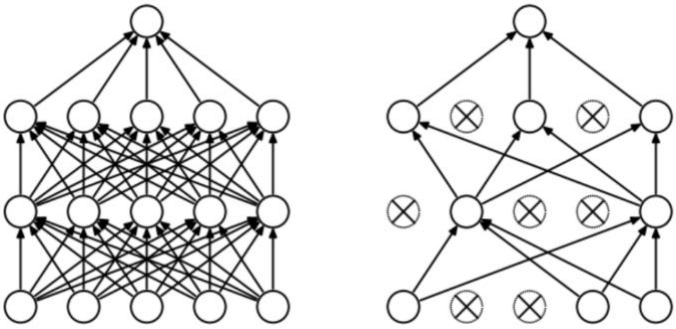

Figura 3.17: Representación del método del dropout [63]

Al realizar predicciones sobre nuevos datos de entrada, se necesita compensar aquellas neuronas que no han sido entrenadas durante la fase de entrenamiento, en la cual el método *dropout* estaba activo. Para ello se modifica la probabilidad de descarte de todas ellas por el valor nulo, para que no vuelvan a descartarse ninguna neurona.[63]
ste método consigue que ninguna de las neuronas de la red memorice el *dataset* de entrada a la red, evitando el *overfitting*.

3.6. Aprendizaje transferido (*Transfer Learning*)

Crear una red neuronal desde 0 para aplicaciones de *Machine Learning o Deep Learning,* es un trabajo muy costoso, debido a la dificultad que supone establecer las conexiones entre capas, y la disposición de estas en una arquitectura nueva.

Existen una gran cantidad de arquitecturas ya diseñadas e implementadas, como las citadas en la sección anterior (VGG Net, U-Net, SegNet o ResNet), previamente entrenadas por un conjunto de imágenes como puede ser **Imagenet**; conjunto de datos de imágenes, diseñada para evaluación de redes y algoritmos de clasificación y reconocimiento de imágenes.[64]
Se emplea mucha carga computacional y tiempo en el entrenamiento de redes con este conjunto de datos, por lo que trabajar con redes pre-entrenadas, permite aprovechar el conocimiento adquirido por estas, y aplicárselo a los filtros de las primeras capas de la red que estemos diseñando, capaces de detectar bordes, texturas, colores, sombras o elementos primarios. Por ello se desarrollo un proceso llamado *Transfer Learning*, por el cual se aprovechan los conocimientos aprendidos de un modelo para aplicarlos en otro del mismo ámbito.[65]
Para usar una red pre-entrenada, hay que adaptarla a las necesidades del problema, este método se llama ajuste fino o *fine tunning.*

3.6.1 Ajuste fino de los pesos (*Fine Tunning*)

Tras realizar *Transfer Learning*, existe un método de ajuste de los pesos mediante la "descongelación" del modelo obtenido o una parte de este, volviendo a realizar el entrenamiento de la red con el dataset de entrenamiento elegido para ello, llamado **Ajuste Fino** o ***Fine Tunning***. Este método de ajuste, consigue mejoras significativas, adaptando de forma gradual aquellas capas entrenadas previamente. El requisito indispensable para ajustar los pesos de manera adecuada, consiste en optimizar la tasa de aprendizaje o *learning rate* (~0.001), permitiendo que toda la red ajuste sus pesos correctamente.[66]

4. Método Propuesto

Tras presentar el método utilizado en este trabajo. Es el momento de hablar del método que hemos utilizado para resolver el problema a tratar. En los siguiente apartados se describen tanto el componente *hardware* como el componente *software* que han permitido segmentar las imágenes del dataset. De forma detallada, se explica como hemos llegado a la solución final, aplicando modificaciones al modelo de red neuronal elegido, que en este caso a sido una red SegNet [1].

4.1 Componentes

Los dos principales componentes implementados son, *hardware,* y *software.* La unión de ambos, han proporcionado los resultados finales tras una ardua tarea en la cual se han realizado multitud de pruebas, hasta que se ha conseguido alcanzar el resultado óptimo.

4.1.1 Componente *'Hardware'*

Las redes neuronales requieren de extensos *datasets* para su entrenamiento y su posterior uso. Estas redes requieren la implementación de librerías de *Deep Learning.* las cuales llevan mucho tiempo de procesamiento y por ello se usa la Unidad de Procesamiento Gráfico o (GPU), del inglés *Graphics Processing Unit.* Esta unidad se localiza en una tarjeta gráfica, permitiendo realizar cálculos complejos en un espacio corto de tiempo.
El entrenamiento de redes neuronales complejas requiere un coste computacional alto. En ocasiones el uso de una sola tarjeta no es suficiente y es necesario varias tarjetas trabajando a la vez, lo cual requiere un coste económico elevado.
A la hora de elaborar este trabajo, la necesidad de una GPU era necesario, por lo tanto podía disponer de ella a través de dos alternativas diferentes:

1. La primera opción sería usar una computadora con una GPU integrada capaz de poder soportar el entrenamiento de mi red neuronal. La gama de la GPU necesaria para la aplicación en cuestión, debía de ser media, ya que en caso contrario los tiempos de entrenamiento serían demasiado altos. La ventaja de usar una GPU, es poder ejecutar el entrenamiento en local, lo que permite la detección de errores de manera más rápida y eficiente.

2. Hacer uso de un servicio *Cloud* como *Google Cloud* basado en los Notebooks de *Jupyter*, que permite el uso de manera gratuita de GPUs y TPUs, y permite el uso de librerías como *PyTorch*, *TensorFlow*, *Keras* y *OpenCV*, y en lenguaje Python.

En este caso, al no disponer de una computadora con GPU integrada, hubo que optar por la segunda opción, la cual ha proporcionado resultados realmente buenos y eficientes.
El servicio de *Google Colab*, tiene la opción de potenciar las prestaciones del servicio, ofreciendo a los usuarios *Google Colab Pro*.
Es un servicio de pago que permite el uso de memorias RAM más rápidas, y la disponibilidad plena de GPUs y TPUs. Este servicio solo está disponible en Estados Unidos y Canada. Es sencillo su uso en España sin restricción, haciendo uso de proveedores de VPNs gratuitos como el utilizado para este caso, *ProtonVPN.* El servidor de *VPN* permite establecer una relación cliente/servidor entre el dispositivo que actúa como cliente y un servidor localizado en el lugar que el cliente desee.

En la tabla 4.1, se representa la comparación de las características de ambas versiones de Google Colab. La versión gratuita proporciona peores prestaciones que la versión PRO, especialmente en la velocidad de la memoria RAM, y en los modelos de GPU.

Tabla 4.1: Diferencias entre Google Colab y Google Colab Pro

	Google Colab	*Google Colab PRO*
Duración de las sesiones	12 horas	24 horas
Tiempo de inactividad	Cortos	Largos (variables)
Disponibilidad de GPUs y TPUs	Según demanda	Siempre
Lenguaje	Python 2 o 3	Python 2 o 3
GPUs disponibles	NVIDIA K80 o NVIDIA Tesla T4	NVIDIA Tesla T4, NVIDIA Tesla P100 o NVIDIA Tesla V100
Memoria RAM	RAM-estándar (hasta 12,72 GHz)	High-RAM (Hasta 25,51 GHz)

Las ventajas que ofrece la versión PRO, mejoraron notablemente las velocidades de entrenamiento, a pesar del gran número de datos de los que disponía el *dataset.* Se ha dispuesto de la GPU NVIDIA Tesla V100, que es la más potente ofrecida por el servidor *cloud* y la memoria High-RAM con hasta 25,51 GHz.
Las especificaciones finalmente escogidas del servicio *cloud* proporcionadas por *Google Colab PRO* para el entrenamiento, validación y testeo de la red neuronal, son las siguientes:

Tabla 4.2: Especificaciones hardware (Google Colab Pro)

Parámetro	*Valor*
CPU	Intel®Xeon® CPU
CPU High-RAM	25.51 GHz
Núcleos CPU	2
Velocidad CPU	2.30 GHz
GPU	NVIDIA Tesla P100
GPU RAM	16 Gb (hbm2)
Duración máxima de una sesión	24 horas
Tiempo ocioso máximo o tiempo sin inactividad máximo	Tiempo variable

4.1.2 Componente '*Software*'

La parte con mayor carga en este proyecto ha sido la componente *software*. Ha requerido de mucho tiempo para estructurar la arquitectura de la red neuronal elegida, y un conjunto de funciones de apoyo a las cuales se recurre tanto en el proceso de entrenamiento y validación como en el de test.
A continuación haremos un estudio previo de las diferentes alternativas de redes neuronales que se pueden implementar para segmentar imágenes, definiendo y desarrollando la arquitectura escogida, y finalizaremos la sección con la explicación de los *frameworks* escogidos para nuestra aplicación de *Deep Learning*.

4.1.2.1 Modelos de red para solucionar problemas de segmentación de imágenes

A la hora de crear una red neuronal, hay que elegir de manera adecuada el modelo que se quiere diseñar, para que mediante el método de *Transfer Learning,* los resultados obtenidos sean los óptimos.

El objetivo de este trabajo es la segmentación de imágenes, y para ello las redes neuronales convolucionales usadas en segmentación tienen una particularidad, la entrada y la salida deben de ser del mismo tamaño. Esto se debe a que no se clasifica toda la imagen en una categoría, sino que se categoriza píxel a píxel.

Los elementos que constituyen la red neuronal convolucional son fáciles de entender. La dificultad reside en el diseño de arquitecturas de modelos que utilicen estos elementos y mejoren sus prestaciones.

En el año 1998, Yann LeCun, desarrollo con éxito la primera aplicación de las redes neuronales convencionales. Fue bautizada con el nombre de LeNet-5, logrando una precisión de clasificación del 99,2% [67].

En el año 2012, el interés en las redes neuronales y el dominio del aprendizaje profundo, dieron origen al reto de reconocimiento visual a gran escala de ImageNet o **ImageNet Large Scale Visual Recognition Challenge ILSVRC**). A raíz de este reto aparecieron nuevas arquitecturas de redes como AlexNet, ZFNet, VGG-16, GoogleNet (también conocida como *Inception)* o ResNet que fue creada por Microsoft.
Todas ellas fueron ganadoras, menos la VGGNet16, que quedó en segunda posición, del reto de ImageNet, y que aparecen representadas en la tabla 4.2.

Tabla 4.3: Redes neuronales más destacadas

Año	*CNN*	*Desarrollada por*	*Posición en el reto ImageNet*	*Top-5 error rate*	*Nº de parámetros*
1998	**LeNet (8)**	**Yann LeCun**			**60.000**
2012	**AlexNet (7)**	**Alex Krizhevsky,Geoffrey Hinton, Ilya Sutskever**	**1º**	**15.3%**	**60 millones**
2013	**ZFNet ()**	**Matthew Zeiler and Rob Fergus**	**1º**	**14.8%**	

2014	**VGG Net (16)**	**Simonyan, Zisserman**	**2º**	**7.3%**	**138 millones**
2014	**Inception (19)**	**Google**	**1º**	**6.67%**	**4 millones**
2015	**ResNet (152)**	**Kaiming He**	**1º**	**3.6%**	

La arquitectura más implementada en segmentación para una red neuronal consiste en tres bloques. Una red de **codificadores** conectada a una red de **decodificadores**, mediante una **capa convolutiva** 1 x 1. Esta estructura de red se ve representada en la figura 4.1. La primera parte corresponde al codificador, el cual btiene un mapa de características, dada una imagen de entrada. El segundo bloque es la capa convolutiva de 1×1. Este bloque consiste en sustituir la capa de neuronas completamente conectadas o *Fully Connected* de la CNN por una capa convolutiva de kernel 1×1. Permite generar una clasificación de la imagen, a la vez que se preserva la información espacial. La red de decodificadores se encarga de recuperar los detalles de los objetos, proyectando las características obtenidas (baja resolución) en el espacio de píxeles. Finalmente, mediante la proyección de las características, obtenemos una clasificación de los píxeles (alta resolución) [68][69].

Para realizar este tipo de redes, se parte de una de las arquitecturas ya creadas. Se reemplazan las capas completamente conectadas, por capas que realizan una **interpolación o *upsampling***. Su objetivo es generar una salida con las mismas dimensiones que la entrada (requisito indispensable en segmentación). A este tipo de redes se les denomina **redes completamente convolucionales** o **FCN** (*Fully Convolutional Networks)* [69].

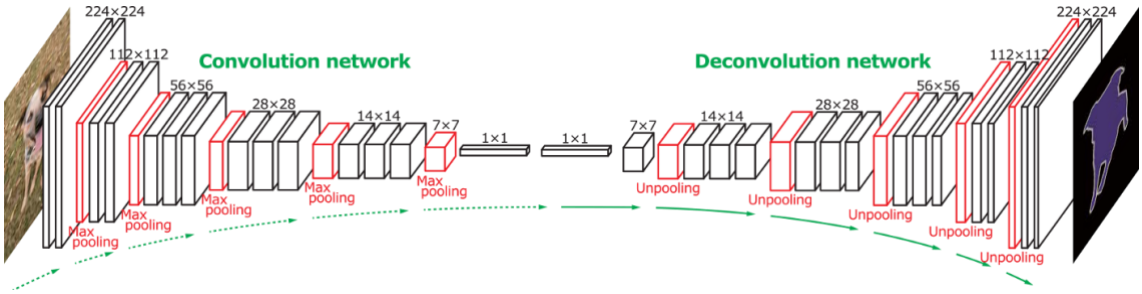

Figura 4.1 : Estructura de una FCN (Fully Convolutional Network)[69]

En el proceso que hemos explicado anteriormente encontramos un problema de pérdida de resolución de los mapas de características de salida, debido a las capas convolucionales y de submuestreo, provocando bordes borrosos en las imágenes de salida. Como solución se han propuesto diferentes modificaciones avanzadas de las FCN

para proporcionar una mejora en la precisión en el camino de decodificación. Algunos ejemplos son la **U-Net**, **SegNe**t y **DeelLabv1**, o algunas más actuales como la **PSPNet**, **DeepLabv2** o la **FPN**, de 2016.

4.1.2.2 Arquitectura escogida

La red utilizada en este trabajo es una red **SegNet**. Como hemos dicho en el apartado anterior, es necesario una red de segmentación basada en una arquitectura de codificador-decodificador, con mejoras en su arquitectura de decodificación, para conseguir mejorar la precisión en la fase de decodificación.[1]

La arquitectura de la red SegNet se representa en la figura 4.2. Esta red se caracteriza por ser una red convolucional codificador-decodificador. Como hemos mencionado antes, son redes *fully convolutional,* es decir, cada capa está conectada a la siguiente capa, de manera que la información circula de izquierda a derecha mediante "*feed forward*".

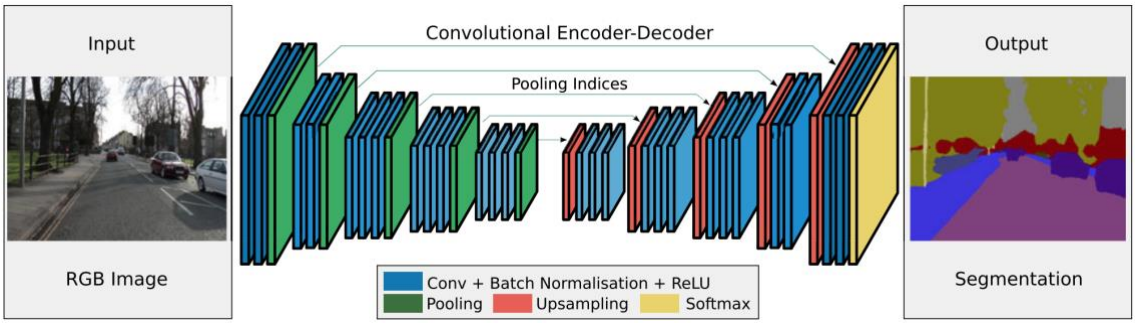

Figura 4.2 : Arquitectura de una red SegNet[70]

4.1.2.3 Desarrollo del modelo escogido

La arquitectura de la red SegNet mantiene una relación codificador-decodificador con una capa final de clasificación de píxeles mediante una función de activación '*softmax*'. En la fase del codificador se produce una contracción de la imagen, y por el contrario en la fase del decodificador se produce una expansión los datos de entrada:

1. **Codificador o camino de contracción**
   En esta etapa, mediante varios niveles sucesivos y conectados entre ellos, se somete a los datos de entrada a un proceso de reducción de sus tamaños mediante capas de submuestreo.

2. **Codificador o camino de expansión**
   El proceso es el contrario al ocurrido en el camino de compresión. En lugar de submuestrear la imagen, el proceso a seguir es la interpolación progresiva de esta, hasta alcanzar las mismas dimensiones al camino de compresión.
3. **Capa de activación**
   La capa final es la más importante de toda la red. Tras haber recorrido cada capa oculta de la red, los pxeles se clasifican mediante una función de activación 'softmax'.

La capa de entrada de la red,se define como la imagen de entrada espaciada, la cual posee un solo canal, o lo que es lo mismo, una imagen a escala de grises, ya que una imagen a color debería de tener 3 canales. La red posee 26 capas convolucionales que se disponen entre codificador y decodificador, o lo que es lo mismo, entre el camino de compresión y el camino de expansión. Se explican detalladamentelas capas que contienen cada uno de ellos a continuación.

Etapa del codificador o camino de compresión:

- **Convoluciones:** La capa convolucional se aplica mediante filtros en cada nivel. En los dos primeros niveles del codificador se aplican 2 filtros convolucionales con dimensiones 3x3, y los 3 bloques siguientes dispondrán de 4 filtros de dimensiones 3x3, es en estos niveles donde la red posee 1 filtro más por nivel, diferenciando la red (VGG19), de una red VGG16. Estas capas no disponen de *padding* (método por el cual se introduce información irrelevante con un objetivo determinado), con lo cual producirá una leve pérdida de información en los bordes de la imagen. El número de filtros es 64 en el primer nivel, de forma que irá aumentando según una potencia de 2, hasta un total de 512 filtros en los niveles 4 y 5 del decodificador o lo que es lo mismo, duplica el número de canales del mapa de características. La función *'relu',* es la función de activación elegida.

- **Max-Pooling:** Al final de cada nivel, se incluye una capa de Max-Pooling o submuestreo. Esta capa se construye mediante filtros 2x2, que cumplen la función de preservar las características más importantes, reduciendo de esta forma el mapa de características a un 50%.

Etapa del decodificador o camino de expansión:

- **ZeroPadding:** Esta capa realiza una función de agregación de filas y columnas de ceros en la parte derecha, izquierda, superior e inferior de un tensor de imagen. De esta forma damos importancia a aquellas imágenes con información localizada en los bordes. Mediante *padding* 1x1.

- **BatchNormalization:** En cada nivel aplicamos una capa de normalización del lote de datos (*batch size*). La función de esta capa es evitar que la distancia en que cada dato sea muy alta ya que, por ejemplo, en una imagen de color la distancia entre un punto claro y otro oscuro puede ir desde 0 hasta 255. Normalizando estos valores reducimos esta distancia en un margen de valores de entre 0 y 1, reduciendo los problemas que se pueden producir en la fase de entrenamiento al tener muchas capas.
- **Upsampling o Submuestreo:** En la etapa del decodificador, también llamada etapa de expansión, aumentamos el tamaño de los datos mediante la inversa de la convolución o deconvolución. Se realiza una interpolación o *upsampling*, mediante un filtro 2x2, que culmina con una capa convolutiva con el mismo tamaño de filtros (3x3), que en la etapa de compresión. A diferencia de la capa convolutiva, una capa *upsampling*, duplica el tamaño del mapa de características.
- **Convolución:** En la etapa de expansión, las capas convolucionales son idénticas a las de la etapa de compresión, con filtros 2x2, con la única diferencia, que en estas capas, el número de filtros se reduce desde 512, hasta 64 filtros, después del paso por todos los niveles, o lo que es lo mismo reduciendo a la mitad el número de canales del mapa de características cada vez que pasa de un nivel a otro.

4.1.2.4. *Frameworks* de entrenamiento para *Machine Learning* (*Keras* y *TensorFlow*)

Python lidera los lenguajes de *Machine Learning* debido a dos motivos: simplicidad y facilidad de aprendizaje. Es el lenguaje de éxito entre los programadores principiantes en *Machine Learning*. Este crecimiento rápido de Python en los últimos años ha provocado que descienda el número de usuarios de R, siguiéndole muy de cerca en segundaposición.

Hay 4 características que hacen un lenguaje más atractivo para programar que otros. Se muestran a continuación: velocidad de computación, curva de aprendizaje, costo (software de código abierto o de pago) y el apoyo de la comunidad. Estas características han provocado que este lenguaje se encuentre presente en muchos de los proyectos de *software* con mayor relevancia a nivel mundial.

Tecnologías basadas en IA como Machine Learning o Deep Learning han apostado por este lenguaje como su base principal, al igual que Tensor-Flow. Todo esto ha provocado que sea el lenguaje idóneo para desarrollar este proyecto, ya que al igual que su facilidad

de aprendizaje, el apoyo de la comunidad y las herramientas de uso, lo han hecho aún más atractivo para ser el candidato para este proyecto.

La elección del entorno sobre el que desarrollar la red neuronal, se basó en la búsqueda de la herramientas más usadas por los usuarios. Para ello una referencia con mucho peso ha sido analizar cuál es la herramienta de *software* más utilizada por los usuarios en *Kaggle*.
Kaggle es una plataforma gratuita que ofrece a los usuarios problemas a resolver con diferentes temáticas como ciencias de datos, análisis predictivo y *machine learning*.

En la figura 4.3, se representa las herramientas más usadas en estas competiciones. La clara ganadora que se coloca en primera posición es **Keras.** Las herramientas en naranja, han sido las utilizadas para aplicaciones de *Deep Learning*, y las que están e azul, han sido implementadas en aplicaciones clásicas.

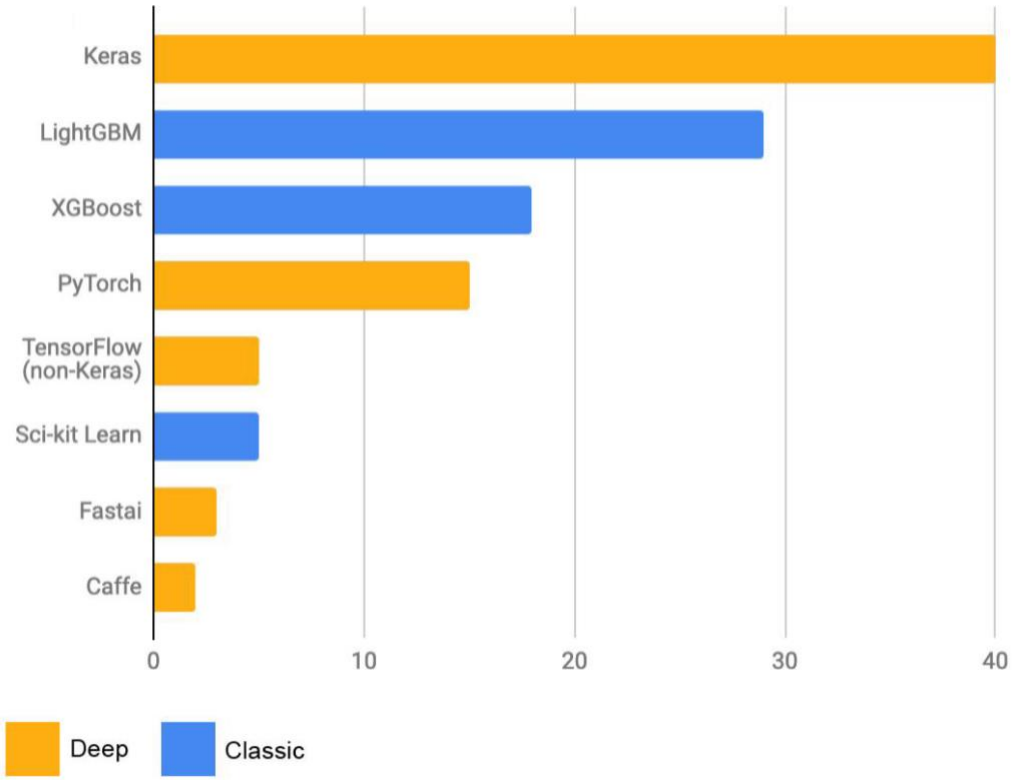

Figura 4.3: Herramientas más usadas en las competiciones de Kaggle.[71]

No sólo es relevante el uso por parte de los usuarios lo que hace de Keras una de las herramientas más usadas en aplicaciones de *Deep Learning,* también su curva de aprendizaje que facilita mucho la implementación en cualquier proyecto por programadores *principiantes.*

Keras es una API de alto nivel, facilita su uso sin conocer su estructura interna. Esta API permite al usuario tratar con interfaces de datos categóricos y numéricos o módulos de alto nivel para permitir implementar redes neuronales, permitiendo agregar estas herramientas con la simple inclusión de capas densas y determinando parámetros de entrada y salida, funciones de activación, optimizadores entre muchos otros parámetros.

Figura 4.4 : Logotipo de Keras[72]

Keras se hace aún más potente si se utiliza encima de **TensorFlow**. Esta aplicación que fue creada por Google en 2015, es muy popular entre los usuarios. Es tanto una herramienta de alto nivel como de bajo nivel, permitiendo implementar al usuario derivación automática. Su flujo de trabajo implementa grafos computacionales, relaciones entre operaciones de datos y tensores.

Figura 4.5: Logotipo de TensorFlow [72]

Por todo ello finalmente, se decidió por utilizar Keras sobre TensorFlow, desarrollando el código *software* en lenguaje Python.

4.2. Método final

En esta sección, se expone el modelo de arquitectura final de red, y las modificaciones que se han llevado a cabo, para mejorar los resultados, de forma que las pérdidas (*loss*) sean mínimas, y la tasa de acierto (*accuracy*) máxima. Primero se expondrán los resultados obtenidos en la fase de entrenamiento, y posteriormente las modificaciones

llevadas a cabo tras las múltiples pruebas que se han llevado a cabo con el *dataset* de entrenamiento.

4.2.1 Entrenamiento

4.2.2. Mejoras introducidas en el entrenamiento

En la fase de entrenamiento los objetivos principales son el aumento de la precisión (*accuracy*) y la reducción de las pérdidas. Para ello es necesario modificar diferentes parámetros hasta encontrar el modelo ideal.

4.2.2.1 Modificaciones de la red SegNet

La creación de una nueva arquitectura de la red desde cero, provoca que sea muy difícil llegar a un modelo consolidado y con buena precisión para el problema planteado. Por ello, tomando la red SegNet como punto de partida, se han introducido una serie de modificaciones que han hecho que el modelo sea más robusto y más fácil de entrenar aplicando el método de *Transfer Learning*. Las modificaciones del modelo son las siguientes:

a) Utilización de un modelo pre-entrenado *(Transfer Learning)*

La capacidad de procesamiento proporcionada por *Google Colab* es limitada, por ello la obtención de un modelo SegNet preentrenado proporcionado por Keras, sería la mejor altenativa. Dado que no existe un modelo preentrenado de Keras para esta red, ha sido necesario buscar otra alternativa.

La arquitectura que finalmente se ha construido es una arquitectura alternativa a la red SegNet, basada en la CNN de clasificación **VGG-19.** En la figura 4.6, se representa la arquitectura de la VGG-19. Se ha utilizado el lenguaje Python incluyendo librerías como numpy, Keras o TensorFlow entre otras.

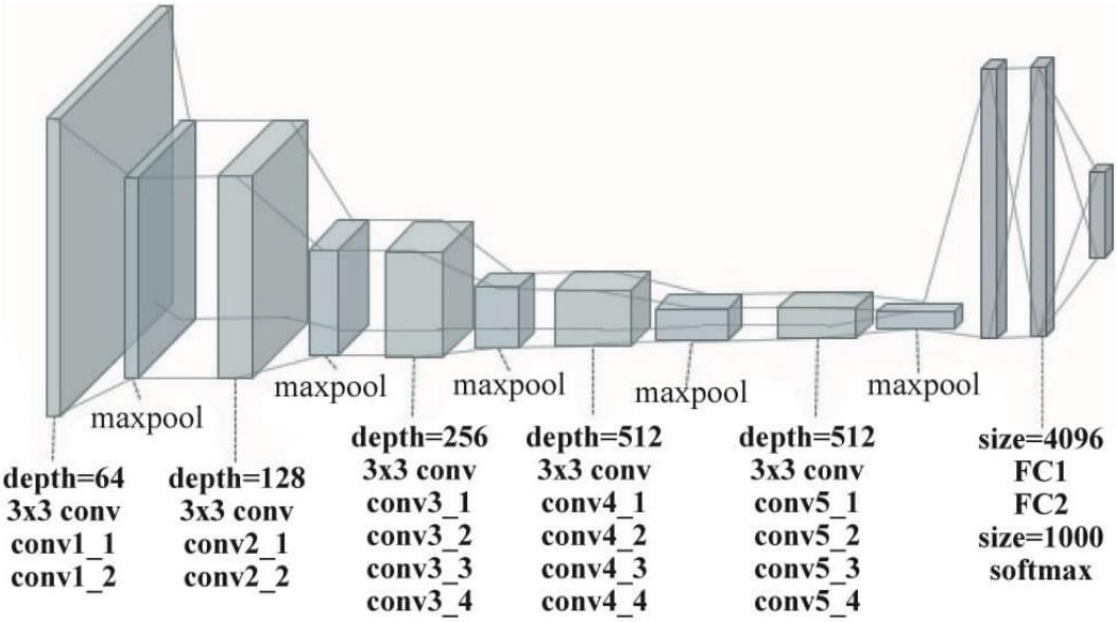

Figura 4.6: Arquitectura de la VGG-19

La red posee una arquitectura de las capas convolucionales similar al camino de contracción de la red SegNet, lo que facilita su implementación, salvando algunas diferencias notables entre una y otra:

1. La imagen de entrada tiene 3 canales (RGB), correspondientes a la composición del color en términos de la intensidad de los colores primarios de la luz.

2. El número de bloques convolucionales varían respecto a la red SegNet, ya que algunos bloques contienen hasta 4 capas convolucionales.

La transferencia de aprendizaje se ha aplicado, haciendo uso del conjunto de datos ***Imagenet.*** El proceso empleado, realiza *forward propagation,* para obtener el resultado del error en la predicción, e incluye *backward propagation,* para corregir los pesos de la red.

b) Modificaciones en las últimas capas

La arquitectura SegNet y su funcionamiento, ha quedado bien definida. Además del modelo pre-entrenado introducido, el cual se ha explicado en el apéndice anterior, para mejorar su funcionamiento y reducir los tiempos de entrenamiento, se han introducido variaciones en la red que se explican a continuación:

1. El primer cambio, es la introducción de la capa **reshape** tras acabar el camino de expansión. Esta capa transforma las entradas en la forma dada, en este caso el producto del ancho por el alto (128x128). Representamos la matriz como un vector de 128x128 = 16.384 números, de forma que las filas se encuentras concatenadas unas con otras. De esta forma es más sencillo procesar la imagen en la fase de entrenamiento. Sería equivalente el de aplanar la salida de la red, para luego aplicarle una función de activación.

2. El segundo punto a realizar es el de aplicar una función de activación 'sigmoid'. Al haber modificado la forma con la función **reshape,** a la salida solo tenemos un vector que contiene todos los píxeles de la imagen, por tanto no sería lógico usar la función de activación de la red SegNet, la función 'softmax', ya que aumentaría mucho las pérdidas.

c) Proceso de 'congelado' de pesos. *(Fine-Tunning)*

En la sección anterior, se ha explicado en lo que consistía el proceso de *fine-tunning,* cuyo objetivo consiste en adaptar los pesos de las capas de la red para optimizar el problema. Con ello incrementamos la capacidad del algoritmo de extraer características de la red sin necesidad entrenar el modelo durante un largo periodo de tiempo.
Para este proyecto, no se han congelado los pesos, ya que los resultados que se obtuvieron al congelar los pesos, fueron deficientes, por lo que se optó por entrenar la red sin congelar los pesos, aumentando con ello el tiempo de entrenamiento y validación.

d) *Learning Rate*

Con el objetivo de reducir las pérdidas de la red, se debe encontrar el valor más apropiado. Para resolver este problema, la solución es encontrar el valor óptimo de *Learning rate* o ritmo de aprendizaje. Éste es un parámetro que reduce las pérdidas a medida que aumenta su valor, como se muestra en la figura 4.7.

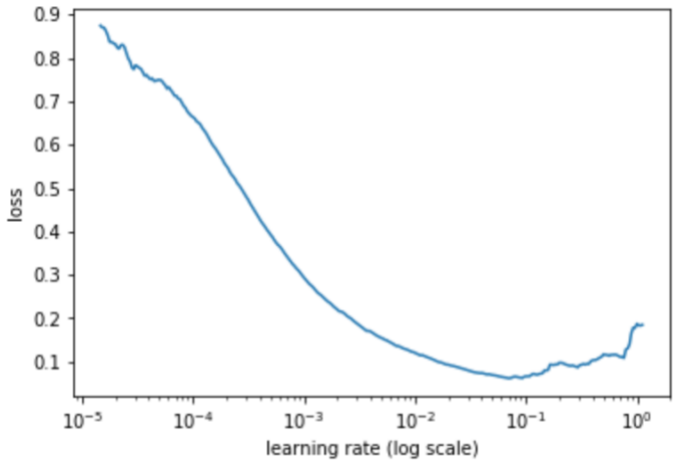

Figura 4.7: Evolución de las pérdidas en función de la tasa de aprendizaje [Creación propia]

Para este proyecto el *learning rate,* se ha ajustado hasta un valor de 0.0001, con lo que hemos introducido pérdidas a cambio de mejor el entrenamiento de la red. En la figura 4.8, se representa la línea de código en la cual se crea un optimizador '*Adam*', con el *learning rate* modificado, respecto al que tiene por defecto (0.001).

```
opt = keras.optimizers.Adam(learning_rate=0.0001)
```

Figura 4.8: Optimizador Adam [Creación propia]

4.3. Modelo final

Tras definir la arquitectura, a continuación se representa el código del modelo final, en el que aparece reflejado todo lo explicado en los puntos anteriores: desde la red SegNet, la CNN de clasificación **VGG-19,** hasta los parámetros que hemos modificado para conseguir los resultados obtenidos. A continuación se muestra el código del modelo:

```
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from keras.applications import vgg19
import h5py
import os
file_path = os.path.dirname( os.path.abspath(__file__) )
VGG_Weights_path =
file_path+"/weights/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5"
def VGGSegnet(n_classes , input_height=128, input_width=128,
                vgg_level=3):
    img_input = Input(shape=(3,input_height,input_width))

        #####Codificador o camino de compresión#####
    #bloque1
```

```
    x=Conv2D(64, (3, 3), activation='relu', padding='same',
        name='block1_conv1', data_format='channels_first')(img_input)
    x=Conv2D(64, (3, 3), activation='relu', padding='same',
        name='block1_conv2', data_format='channels_first')(x)
    x=MaxPooling2D((2,2),strides=(2,2),name='block1_pool',
        data_format='channels_first')(x)
    f1=x
    #bloque2
    x=Conv2D(128,(3,3), activation='relu', padding='same',
        name='block2_conv1', data_format='channels_first')(x)
    x=Conv2D(128,(3,3), activation='relu', padding='same',
        name='block2_conv2', data_format='channels_first')(x)
    x=MaxPooling2D((2,2),strides=(2,2),name='block2_pool',
        data_format='channels_first')(x)
    f2=x
    #bloque3
    x=Conv2D(256,(3,3), activation='relu', padding='same',
        name='block3_conv1', data_format='channels_first')(x)
    x=Conv2D(256,(3,3), activation='relu', padding='same',
        name='block3_conv2', data_format='channels_first')(x)
    x=Conv2D(256,(3,3), activation='relu', padding='same',
        name='block3_conv3', data_format='channels_first')(x)
    x=Conv2D(256,(3,3), activation='relu', padding='same',
        name='block3_conv4', data_format='channels_first')(x)
    x=MaxPooling2D((2,2),strides=(2,2),name='block3_pool',
        data_format='channels_first')(x)
    f3=x
    #bloque4
    x=Conv2D(512,(3,3), activation='relu', padding='same',
        name='block4_conv1', data_format='channels_first')(x)
    x=Conv2D(512,(3,3), activation='relu', padding='same',
        name='block4_conv2', data_format='channels_first')(x)
    x=Conv2D(512,(3,3), activation='relu', padding='same',
        name='block4_conv3', data_format='channels_first')(x)
    x=Conv2D(512,(3,3), activation='relu', padding='same',
        name='block4_conv4', data_format='channels_first')(x)
    x=MaxPooling2D((2,2),strides=(2,2),name='block4_pool',
        data_format='channels_first')(x)
    f4=x
    #bloque5
    x=Conv2D(512,(3,3), activation='relu', padding='same',
        name='block5_conv1', data_format='channels_first')(x)
    x=Conv2D(512,(3,3), activation='relu', padding='same',
        name='block5_conv2', data_format='channels_first')(x)
    x=Conv2D(512,(3,3), activation='relu', padding='same',
        name='block5_conv3', data_format='channels_first')(x)
    x=Conv2D(512,(3,3), activation='relu', padding='same',
        name='block5_conv4', data_format='channels_first')(x)
    x=MaxPooling2D((2,2),strides=(2,2),name='block5_pool',
```

```
    data_format='channels_first')(x)
f5=x

vgg= Model(img_input, x)
vgg.load_weights(VGG_Weights_path)

levels = [f1, f2 , f3 , f4 , f5]

o=levels[vgg_level]

#####Decodificador o camino de expansión######
o = (ZeroPadding2D((1,1) , data_format='channels_first'))(o)
o = (Conv2D(512,(3,3),padding='valid',
    data_format='channels_first'))(o)

o = (BatchNormalization())(o)

o = (UpSampling2D((2,2) , data_format='channels_first'))(o)


o = (ZeroPadding2D((1,1) , data_format='channels_first'))(o)
o = (Conv2D(256,(3,3),padding='valid',
    data_format='channels_first'))(o)
o = (BatchNormalization())(o)

o = (UpSampling2D((2,2) , data_format='channels_first'))(o)
o = (ZeroPadding2D((1,1) , data_format='channels_first'))(o)
o = (Conv2D(128,(3,3),padding='valid',
    data_format='channels_first'))(o)
o = (BatchNormalization())(o)

o = (UpSampling2D((2,2) , data_format='channels_first'))(o)
o = (ZeroPadding2D((1,1) , data_format='channels_first'))(o)
o = (Conv2D(64,(3,3),padding='valid',
    data_format='channels_first'))(o)
o = (BatchNormalization())(o)

#que hace aqui con n_classes
o = (UpSampling2D((2,2) , data_format='channels_first'))(o)
o = (ZeroPadding2D((1,1) , data_format='channels_first'))(o)
o = (Conv2D(64,(3,3),padding='valid',
    data_format='channels_first'))(o)
o = (BatchNormalization())(o)

o = Conv2D(1, (3,3) , padding='same',
    data_format='channels_first')(o)
o_shape = Model(img_input, o).output_shape
outputHeight = o_shape[2]
outputWidth = o_shape[3]
```

```
    o = (Reshape((-1, outputHeight*outputWidth)))(o)
    o = (Permute((2,1)))(o)
    o = (Activation('sigmoid'))(o)
    model = Model(img_input,o)

    return model
```

Keras pone a disposición del usuario multitud de herramientas para verificar los resultados obtenidos por parte del usuario. En este caso, mediante la función *summary(),* podremos visualizar los siguientes parámetros:

- Las capas y su orden en el modelo
- La forma de salida de cada capa
- El número de parámetros (pesos) en cada capa
- El número total de parámetros (pesos) en el modelo

A continuación se representa la salida de la línea de códido: *print(model.summary()*

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 3, 128, 128)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 128, 128)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 128, 128)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 128, 64, 64)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 128, 64, 64)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 128, 32, 32)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 256, 32, 32)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 256, 32, 32)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 256, 32, 32)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 256, 32, 32)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 256, 16, 16)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 512, 16, 16)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 512, 16, 16)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 512, 16, 16)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 512, 16, 16)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 512, 8, 8)         0
_________________________________________________________________
```

zero_padding2d (ZeroPadding2	(None, 512, 10, 10)	0
conv2d (Conv2D)	(None, 512, 8, 8)	2359808
batch_normalization (BatchNo	(None, 512, 8, 8)	32
up_sampling2d (UpSampling2D)	(None, 512, 16, 16)	0
zero_padding2d_1 (ZeroPaddin	(None, 512, 18, 18)	0
conv2d_1 (Conv2D)	(None, 256, 16, 16)	1179904
batch_normalization_1 (Batch	(None, 256, 16, 16)	64
up_sampling2d_1 (UpSampling2	(None, 256, 32, 32)	0
zero_padding2d_2 (ZeroPaddin	(None, 256, 34, 34)	0
conv2d_2 (Conv2D)	(None, 128, 32, 32)	295040
batch_normalization_2 (Batch	(None, 128, 32, 32)	128
up_sampling2d_2 (UpSampling2	(None, 128, 64, 64)	0
zero_padding2d_3 (ZeroPaddin	(None, 128, 66, 66)	0
conv2d_3 (Conv2D)	(None, 64, 64, 64)	73792
batch_normalization_3 (Batch	(None, 64, 64, 64)	256
up_sampling2d_3 (UpSampling2	(None, 64, 128, 128)	0
zero_padding2d_4 (ZeroPaddin	(None, 64, 130, 130)	0
conv2d_4 (Conv2D)	(None, 64, 128, 128)	36928
batch_normalization_4 (Batch	(None, 64, 128, 128)	512
conv2d_5 (Conv2D)	(None, 1, 128, 128)	577
reshape (Reshape)	(None, 1, 16384)	0
permute (Permute)	(None, 16384, 1)	0
activation (Activation)	(None, 16384, 1)	0

4.4. *Dataset utilizado*

Para este trabajo, el conjunto de datos de entrenamiento con el que se entrena a la red es el que se utilizó en el *ISIC challenge* de 2018, para el cual se disponían de un total de 2594 imágenes en color en formato JPEG. Cada imagen JPEG tiene una máscara asociada con formato PNG en blanco y negro.

Los conjuntos de *dataset* para las fases de entrenamiento y validación, son los siguientes.[73] :

1. ***Dataset* de entrenamiento:** 2194 imágenes a color formato (.jpg), son la imágenes de entrada (*input*), con sus correspondientes 2194 imágenes con formato (.png), correspondientes a las máscaras o imágenes de segmentación (*groundtruth*).

2. ***Dataset* de validación:** 400 imágenes a color formato (.jpg), son la imágenes de entrada (*input*), con sus correspondientes 400 imágenes con formato (.png), correspondientes a las máscaras o imágenes de segmentación (*groundtruth*).

De las 2194 imágenes repartidas entre ambos *datasets,* ninguna de ellas se repite. Todas las imágenes con formato (.jpg) son a color con dimensiones diferentes entre ellas, con sus correspondientes máscaras en blanco y negro con formato (.png).
El total de imágenes es suficiente para obtener buenos resultados en las fases de entrenamiento, validación y test, por lo que no ha sido necesario modificarlo, para incrementar el número de datos con los cuales entrenar nuestra red. Los resultados de ambas fases se representan en la siguiente sección del trabajo.

5. Resultados

En este capítulo se analizan los resultados obtenidos tras haber realizado el entrenamiento de la red neuronal explicada en el apartado anterior para el *dataset* proporcionado para su entrenamiento, validación y test. Los parámetros seleccionados para los cuales se han obtenido estos resultados, han proporcionado los mejores valores de medidas de rendimiento, como por ejemplo exactitud, especificidad, precisión o índice *Jaccard*.

5.1 Análisis de eficiencia del modelo

Los sistemas basados en *Machine Learning,* son evaluados mediante métricas de precisión de clasificación binaria {0,1} o {*false*, *true*}, {*negative*, *positive*}. Las métricas de clasificación binaria, cuantifican los dos tipos de predicciones correctas {**verdadero positivo o verdadero negativo** } y dos tipos de errores {**falso positivo o falso negativo**}.

Para la predicción píxel a píxel existen 4 tipos de predicciones. Considerando la presencia de lesión (píxel activo) como positivo y la no presencia de lesión (píxel no activo) como negativo:

- **Falso positivo (FP)**: Resultado negativo que se ha predicho como positivo. **Error tipo I.**
- **Verdadero positivo (TP)**: Resultado positivo que se ha predicho como positivo.
- **Verdadero negativo (TN)**: Resultado negativo que se ha predicho como negativo pero esta vez de forma correcta.
- **Falso negativo (FN)**: Resultado positivo que se ha predicho como negativo. **Error tipo II**. [20]

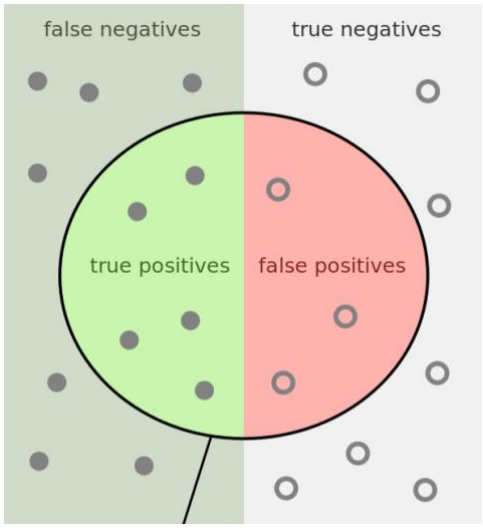

Figura 5.1 : Tipos de clasificaciones usadas en segmentación[74]

Las métricas de clasificación se calculan en función de estos valores. En este aparatado se calculan la tasa de acierto, la especificidad, la precisión, la sensibilidad y el coeficiente de Sorensen-Dice.
Para cada imagen, se realiza una comparación entre los píxeles de cada segmentación predicha y la segmentación real utilizando una métrica llamada índice *Jaccard*. Es una métrica utilizada en el *challenge* de la ISIC 2018.

- **Exhaustividad o Sensibilidad** (*Recall*): Es un medidor que indica cuál es la proporción de aquellos casos que son positivos y se han clasificado como tal. Esta tasa también se le denomina tasa de verdaderos positivos o TPR.

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

- **Especificidad** (*Specificity*): Calcula la proporción de aquellos negativos que se han clasificado como tales. Recibe también el nombre de tasa de falsos negativos.

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

- **Exactitud o tasa de acierto** (*Accuracy*): La exactitud o tasa de acierto, mide el porcentaje de casos en los que el modelo ha acertado. Tanto los verdaderos positivos como los verdaderos negativos.

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

Su uso no es muy adecuado, debido a que puede conducir a un problema de exactitud. Un modelo malo puede parecer mucho mejor de lo que realmente es, debido a que tiende a indicarnos el conjunto mayoritario. Por lo que un alto número de *Verdaderos Positivos*, produce una alta tasa de acierto.

- **Precisión (***Precision*): Mide el número de verdaderos positivos que realmente son positivos. Nos proporciona un valor de la **calidad** del modelo.

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

- **Índice Jaccard** (*Jaccard Index*): El índice o coeficiente *Jaccard* es un coeficiente que mide el nivel de semejanza entre dos conjuntos a comparar, independientemente del tipo de elemento. Su rango de valores discurre entre 0 y 1, tomando el 0 como semejanza nula, por el contrario, el 1 caracterizaría los dos conjuntos como idénticos.

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

El umbral *Jaccard* o *Jaccard Threshold,* con un valor (~ 0,65), con una tolerancia de error adicional es el valor que determina la calidad del trabajo realizado, considerando una segmentación fallida si el coeficiente *Jaccard* para esa segmentación queda por debajo del umbral establecido. Es un parámetro de calidad indispensable para el *ISIC challege* de 2018.

- **Coeficiente de Sørensen-Dice** (Sørensen-Dice *coefficient*): También llamado índice de Sørensen o coeficiente de Dice, compara la similitud entre dos conjuntos de muestras A y B. Se calcula:

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

Si aplicamos la expresión a datos binarios o booleanos, encontramos la siguiente expresión:

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

DSC es el **coeficiente de similitud de datos** y varía entre 0 y 1.

5.1.1 Matriz de confusión

Una vez calculados los tipos de decisiones y las métricas, se define una herramienta de rendimiento llamada matriz de confusión.
Es una herramienta que permite describir el rendimiento de un modelo supervisado de *Machine Learning* de los datos de prueba, donde se desconocen los verdaderos valores. Se llama "matriz de confusión" debido a que es fácil detectar dónde el sistema confunde dos clases. En la figura 3.2 se representa esta matriz.

Figura 5.2: Matriz de confusión con métricas de evaluación [75]

5.2 Resultados del entrenamiento y validación

En este apartado, se exponen las tablas con los resultados obtenidos en las fases de entrenamiento y validación. Se han llevado a cabo numerosos entrenamientos. Tras cada entrenamiento, se han ido mejorando los parámetros, para obtener los resultados de esta sección.

5.2.1 Parámetros de entrenamiento

Los parámetros de la tabla 5.1, son los utilizados en el entrenamiento final de la red propuesta en este trabajo, la modificación del *learning rate*, y las modificaciones en la arquitectura se han explicado en la sección anterior 4.2.
El tiempo de entrenamiento ha sido un problema a resolver, de forma que el *batch* elegido ha sido 128, para poder disminuirlo.

Tabla 5.1: Parámetros de entrenamiento

Parámetro	*Valor*
Tamaño del conjunto de datos	2194
Dimensiones de entrada	128x128x3
Épocas	20

Tamaño del *batch*	128
Tipo de optimizador	*Adam*
Tasa de aprendizaje o *learning rate*	0.0001

5.2.2 Coste computacional

En la tabla 5.2 se incluye el coste de computación tanto del entrenamiento como de la validación.

Tabla 5.2: Coste computacional del entrenamiento(Tabla 5.2)

Parámetro	*Valor*
Épocas	20
Tiempo	19h 46m
RAM usada	3.32 GB
GPU usada	5.43 GB

5.2.3 Resultados

Este quizás sea uno de los apartados más importantes, en el cual se representarán los resultados obtenidos del entrenamiento y validación de la red. La figura 5.3 ofrece la evolución de la exactitud de la fase de entrenamiento, y la de validación. Como es de esperar, la tasa de acierto del entrenamiento es mayor que la de validación, ya que de lo contrario el modelo no tendría sentido. Otro punto a tener en cuenta es el comportamiento de cada una por separado.

- **accuracy_train:** Ya que la red se entrena con los pesos de la red CNN VGG19, obtenemos una exactitud inicial entorno al 85%, incrementando su valor en tan solo 4 épocas hasta un valor aproximado de 93%.
- **accuracy_val:** Como ya hemos mencionado su valor es menor al de la tasa de acierto del entrenamiento. El segundo detalle a tener en cuenta, es la estabilización de ésta en torno al 85%, no presentando síntomas de *overfitting* al no variar bruscamente entre una época y la siguiente.

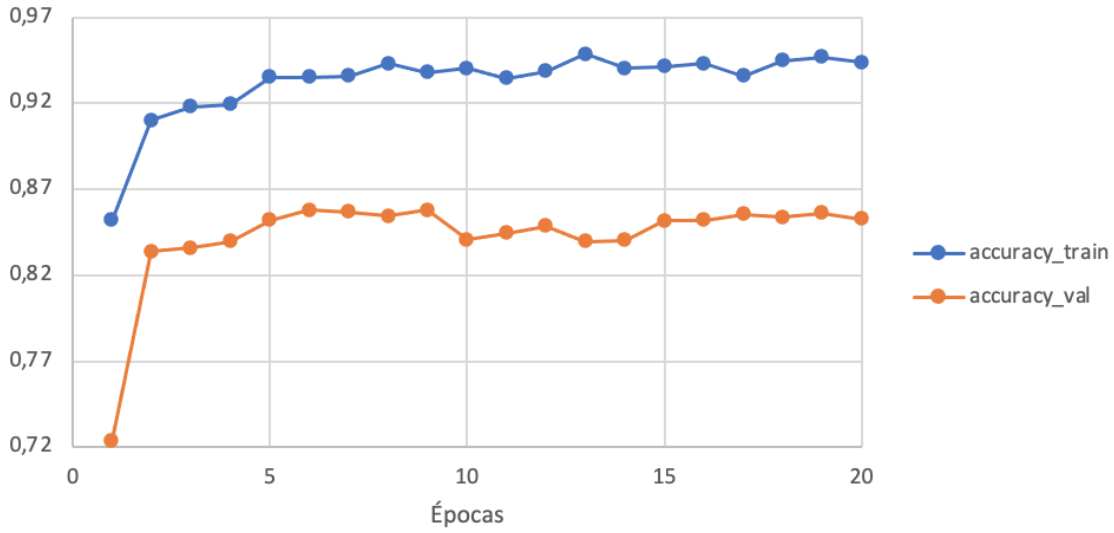

Figura 5.3: Evolución de la tasa de acierto de entrenamiento y validación

En la figura 5.4, se muestra la evolución de las pérdidas tanto de entrenamiento como de validación. Los valores que más relevancia tienen, son los de la curva de entrenamiento, estabilizada desde la época 5.

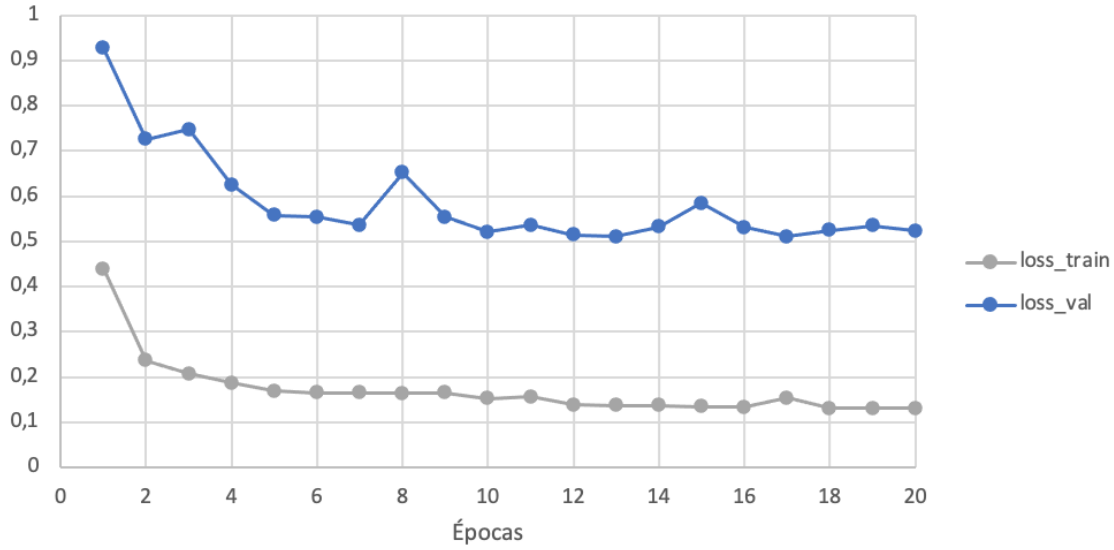

Figura 5.4: Evolución de la función de pérdidas de entrenamiento y validación

En la siguiente tabla, se muestran los valores medios de las métricas finales obtenidas:

Tabla 5.3: Valores medios de las métricas finales de rendimiento de la red neuronal implementada (5.2)

Parámetro	*Valor*
Exactitud (*Accuracy*)	0.911115
Especificidad (*Specificity*)	0.883347
Precisión (*Precision*)	0.841342
Exhaustividad o Sensibilidad (*Recall*)	0.959704
Coeficiente Dice	0.805721
Índice Jaccard	0.729692

Para la competición propuesta por el ISIC, se evalúa la calidad del modelo mediante una métrica especial llamada índice Jaccard. Para este caso, no es justamente el índice Jaccard, sino una modificación del mismo. El cálculo del índice Jaccard se formula con aquellas imágenes segmentadas que superen un valor de 0,65; en caso contrario la segmentación será nula o fallida.

Tabla 5.4: Índices de valoración según el ISIC challenge 2018(5.4)

Parámetro	*Valor*
Índice Jaccard	0.729692
Índice Jaccard modificado	0.667504

5.3 Resultados de las predicciones

En este último apartado, se representan tanto las mejores, como las peores predicciones de las máscaras de las 400 imágenes de test predichas por nuestro modelo. También se incluyen otros ejemplos de resultados, como los valores en torno al umbral Jaccard.

En la tabla 5.5, se muestran el número de imágenes testeadas, y el número de imágenes que están por encima del umbral de 0.65 y de la media.

Tabla 5.5: Predicciones que cumplen superan el índice Jaccard

Parámetro	*Valor*
Número de imágenes	400
Índice Jaccard con umbral(valor medio)	0.667504
Número de imágenes por encima de la media	292(73%)
Número de imágenes por encima del umbral *Jaccard*	306 (76,5%)

En la tabla 5.6, y en la 5.5, se muestra los resultados de las 5 mejores predicciones de nuestro modelo, con una tasa de acierto rondando el 99%, lo cual ratifica la calidad de modelado de nuestra red.

Tabla 5.6: Índice Jaccard de las 5 mejores predicciones

ID de la imagen	*Tasa de acierto (accuracy)*	*Índice Jaccard*
ISIC_0015218.jpg	0.983581543	0.9438061416
ISIC_0015118.jpg	0.9722290039	0.9588309808
ISIC_0015255.jpg	0.9725341797	0.9686934743
ISIC_0015274.jpg	0.9654541016	0.9610192837
ISIC_0016055.jpg	0.9877929688	0.9561307304

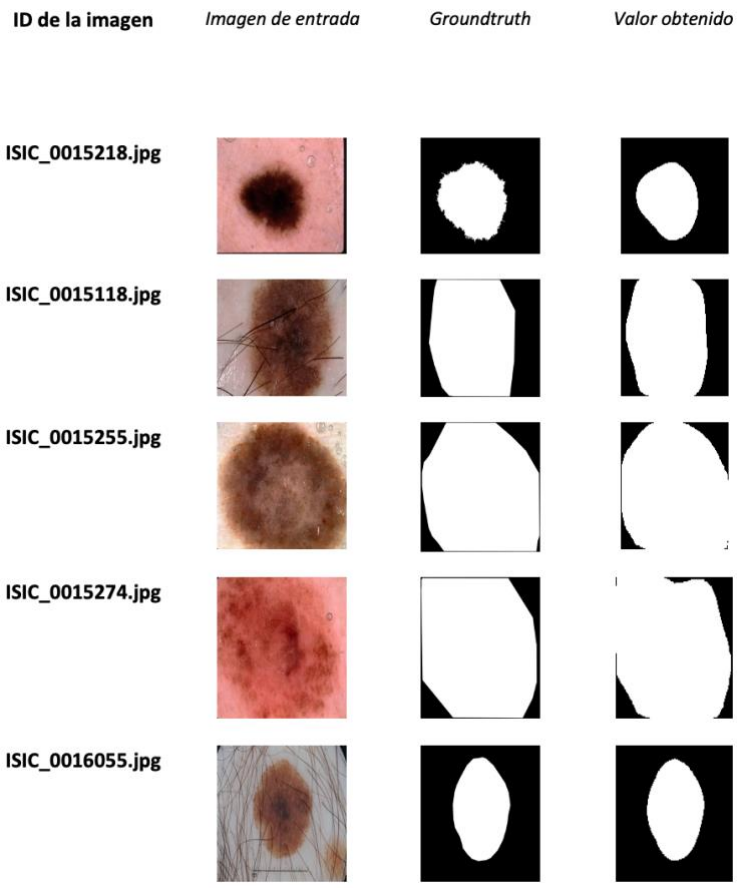

Figura 5.5: Representación de las 5 mejores predicciones

En la tabla 5.7, y en la figura 5.6, se representan los resultados obtenidos de las 5 peores predicciones según el índice Jaccard:

Tabla 5.7: Índice Jaccard de las 5 peores predicciones

ID de la imagen	*Tasa de acierto (accuracy)*	*Índice Jaccard*
ISIC_0015951.jpg	0.8188476563	0.01754385965
ISIC_0016000.jpg	0.3498535156	0.04816370298
ISIC_0016027.jpg	0.7724609375	0.03594517714
ISIC_0016034.jpg	0.6469116211	0.07157759589
ISIC_0016058.jpg	0.3939208984	0.03225806452

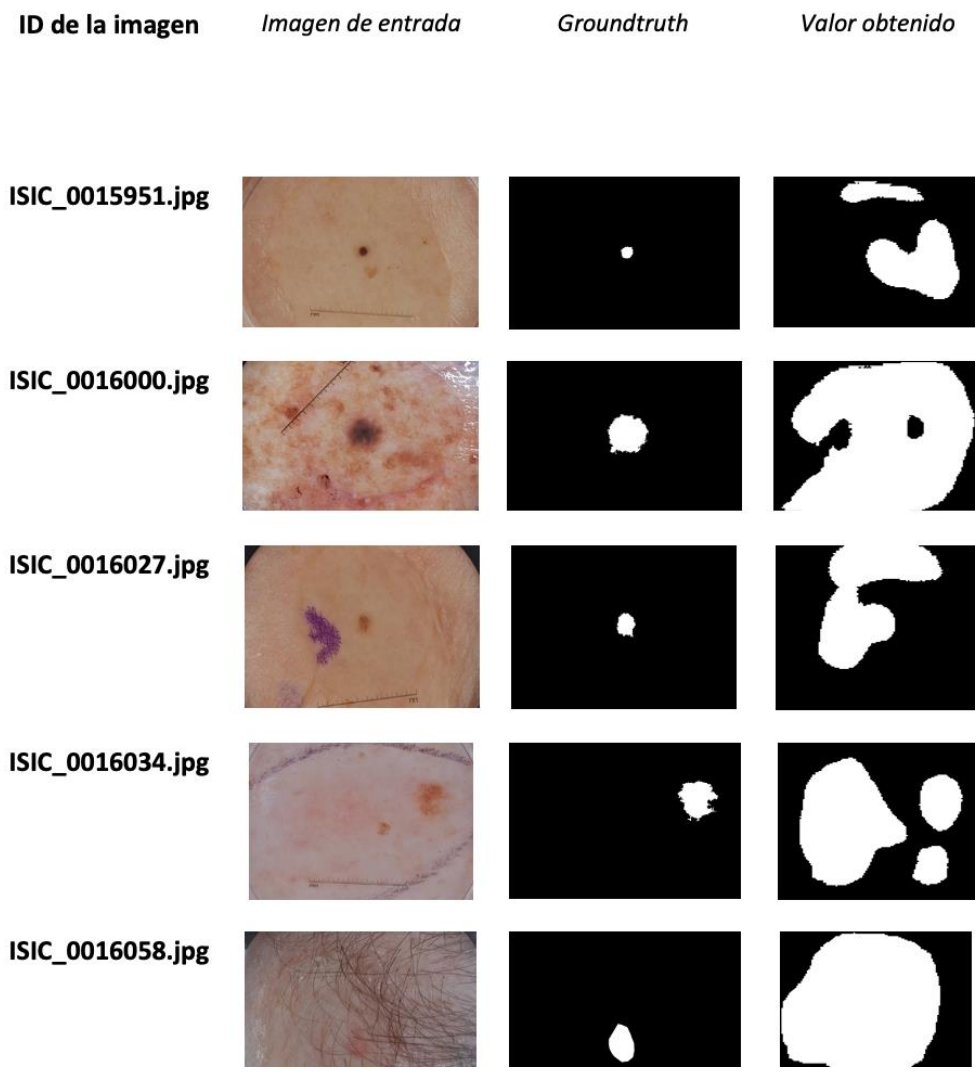

Figura 5.6: Representación de las 5 peores predicciones

Los resultados obtenidos son buenos, ya que como muestran las métricas correspondientes, y sus respectivas máscaras obtenidas, concuerdan con una gran tasa de acierto y exactitud en el resultado.
Si apreciamos los resultados obtenidos de las 5 peores predicciones, observamos cómo en las imágenes de entrada, es dificultoso apreciar las lesiones, ya que o estas son muy pequeñas, o el fondo es de la misma tonalidad que la lesión, lo que hasta para el ojo humano resultaría casi inapreciable. Por ello los malos resultados son coherentes con estas imágenes, que llegan hasta a coincidir en algún punto de la máscara predicha con la máscara original, pero sin buenos resultados.

Los siguientes resultados adicionales, muestran las 5 predicciones que más se ajustan al valor medio (0,6675), inmediatamente por encima. Los resultados obtenidos se representan en la tabla 5.8, y en la figura 5.7.

Tabla 5.8: Índice Jaccard de las 5 primeras predicciones que superan el valor medio

ID de la imagen	*Tasa de acierto (accuracy)*	*Índice Jaccard*
ISIC_0014944.jpg	0.9427490234	0.6845998655
ISIC_0014989.jpg	0.8900756836	0.6922419686
ISIC_0015013.jpg	0.7490844727	0.6759419833
ISIC_0015447.jpg	0.7166748047	0.6806549257
ISIC_0015993.jpg	0.9856567383	0.6793997271

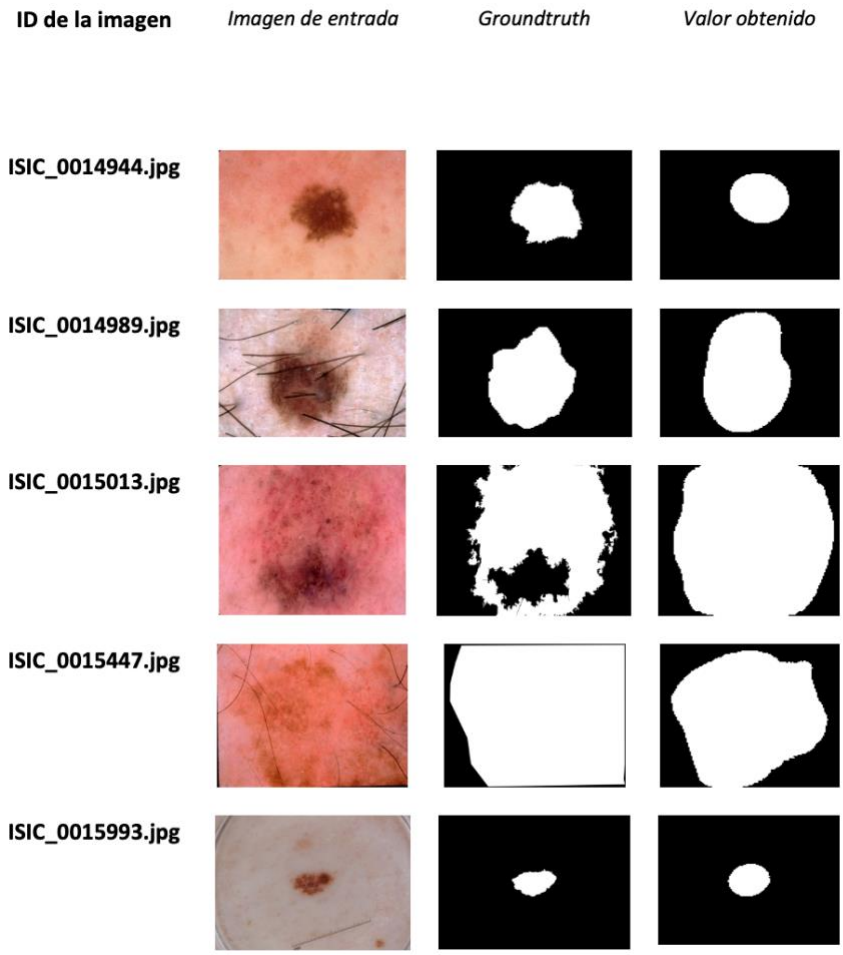

Figura 5.7: Representación de las 5 primeras predicciones que superan el valor medio

Para terminar de representar las predicciones, a continuación representamos las 3 predicciones inmediatamente superiores e inmediatamente inferiores al valor umbral establecido por el ISIC challenge.

Tabla 5.9: Índice Jaccard de las 6 predicciones en torno al umbral de 0.65.

ID de la imagen	*Tasa de acierto (accuracy)*	*Índice Jaccard*
ISIC_0014851.jpg	0.9105834961	0.6583488806
ISIC_0014987.jpg	0.9075927734	0.6544167998
ISIC_0014855.jpg	0.8814697266	0.6526560544
ISIC_0015481.jpg	0.9526367188	0.6493447808
ISIC_0015015.jpg	0.8331298828	0.647453256
ISIC_0015057.jpg	0.8288574219	0.6416613419

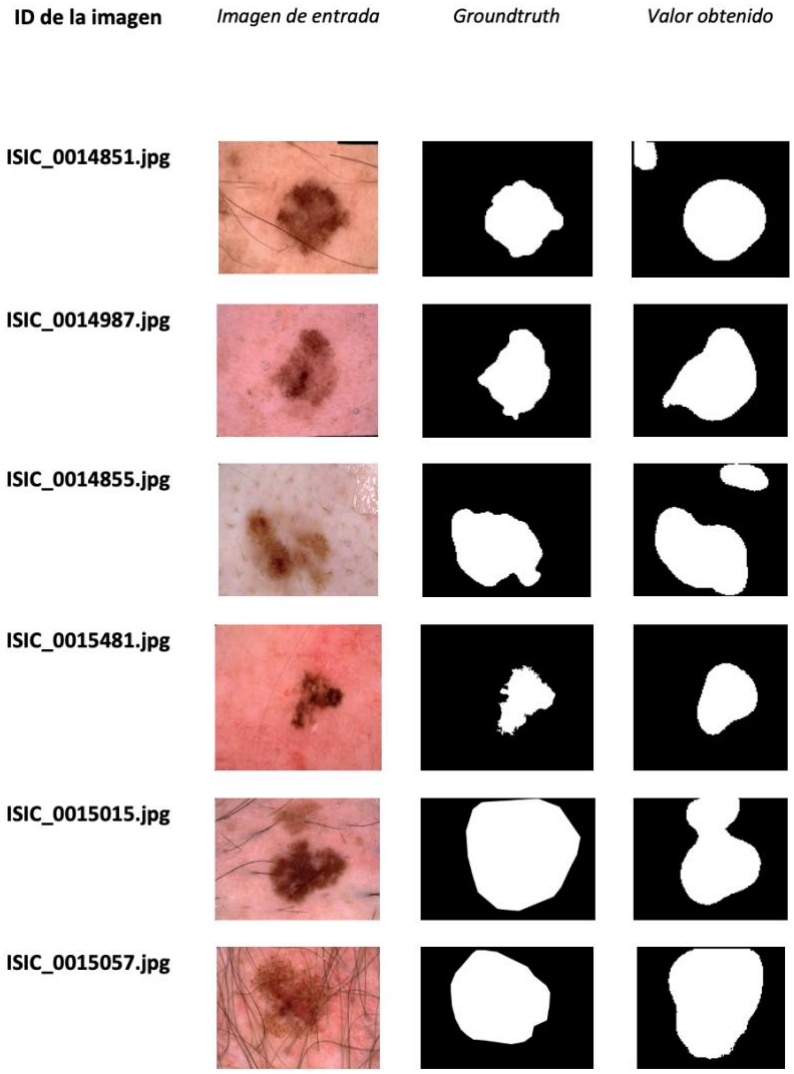

Figura 5.8: Representación de las 6 predicciones en torno al umbral de 0.65

6. Conclusiones del trabajo

El uso de una tecnología como el *Deep Learning,* está generando una revolución en cuanto a la creación de nuevas herramientas muy versátiles que cubren multitud de campos. En el ámbito de la medicina, su potencial es aún mayor, ya que los resultados obtenidos tienen en muchos casos una calidad y precisión mayor a la que el ser humano puede llegar a proporcionar. Estas herramientas no tratan de sustituir en ningún caso la presencia humana en áreas como la medicina, sino ofrecer herramientas de ayuda a los especialistas.

En un futuro, el desarrollo de la tecnología permitirá que los entrenamientos de las redes sean mucho más rápidos mediante GPUs y TPUs más potentes, capaz de tratar con *datasets* más grandes permitiendo ajustar aún más los parámetros de la red, obteniendo de esta forma resultados muy precisos.

6.1 Conclusiones

El modelo de red neuronal implementado para este proyecto junto con sus modificaciones desarrolladas durante la fase de entrenamiento, clarifica unos resultados que se podrían catalogar de éxito, ya que como se ha representado en el apartado 5.3, se ha conseguido una tasa de acierto media en las predicciones de hasta un 84%

Por tanto se concluye que la red neuronal SegNet, preentrenada con una CNN VGG19, ha sido una decisión acertada a la hora de desarrollar una implementación basada en *Deep Learning* para la segmentación de lesiones pigmentadas en la piel.

Un punto importante a señalar es la línea de trabajo desarrollada para este trabajo. De la misma manera que se ha aplicado el *Deep Learning* para segmentación, el flujo de trabajo que se ha seguido puede implementarse en otras muchas áreas con similar o distinta aplicación, realizando siempre un estudio previo de cuál podría ser la mejor solución para el problema a tratar, y conseguir con ello optimizar los resultados.

6.2 Líneas futuras

En el campo de la medicina, el *Deep Learning,* según los expertos, avanza hacia un futuro próximo de aprendizaje no supervisado, o lo que es lo mismo, aprendizaje sin necesidad de presencia humana que ratifique lo que es correcto o lo que no, de forma que obtengan esas conclusiones a partir del conjunto de datos ingerido por la red.

Una posible mejora para la segmentación de lesiones de la piel sería utilizar ensambladores de redes neuronales profundas que permitan obtener las ventajas de éstas a la hora de clasificar un píxel como perteneciente o no a la lesión.

Por último, la inclusión de información adicional a la imagen de entrada a la red, como podría ser el tipo de lesión o características del paciente, podría ayudar a obtener mejores resultados.

Referencias

[1] Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, *39*(12), 2481-2495.

[2] Takeyas, Bruno López. "Introducción a la inteligencia artificial." *Instituto Tecnológico de Nuevo Laredo. Web del autor: http://www. itnuevolaredo. edu. mx/takeyas* (2007).

[3] Braeuer RR, Watson IR, Wu CJ, Mobley AK, Kamiya T, Shoshan E, Bar-Eli M. Why is melanoma so metastatic? Pigment Cell Melanoma Res. 2014 Jan;27(1):19-36. doi: 10.1111/pcmr.12172. Epub 2013 Oct 17. PMID: 24106873.

[4] Siegel RL, Miller KD, Jemal A. Cancer statistics, 2019. *CA Cancer J Clin*. 2019;69(1):7-34.

[5] The American Cancer Society, “Melanoma Skin Cancer Early Detection, Diagnosis and Staging, 2020

[6] https://elhexagono.wordpress.com/2018/06/16/melanoma-el-asesino-silencioso/[imagen]

[7] Lee JB, Hirokawa D. Dermatoscopy: Facts and controversies. *Clin Dermatol*. 2010;28(3):303-310.

[8] https://www.dhmaterialmedico.com/dermatoscopio-dermlite-dl200-hybrid [imagen]

[9] Chen LL, Dusza SW, Jaimes N, Marghoob AA. Performance of the First Step of the 2-Step Dermoscopy Algorithm. *JAMA Dermatol*. 2015;151(7):715-721.

[10] Palomino, N. L. S., & Concha, U. N. R. (2009). Técnicas de segmentación en procesamiento digital de imágenes. *Revista de investigación de Sistemas e Informática*, *6*(2), 9-16.

[11]] IBÁÑEZ, L., SCHROEDER, W., NG, L. & CATES, J. 2005. The ITK Software Guide. Insight Software Consortium

[12] YAO, J. 2006. Image Processing in Tumor Imaging. New Techniques in Oncologic Imaging. 79-102.

[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] Herold-García, S., & Escobedo-Nicot, M. (2007). Segmentación de imágenes médicas con la aplicación de snakes. *Ciencia en su PC*, (4), 12-22.

[15] H.D. Cheng, X.H. Jiang, Y. Sun, Jingli Wang,Color image segmentation:advances and prospects,Pattern Recognition,Volume 34, Issue 12,2001,Pages 2259-2281,ISSN 0031-3203.

[16] International Skin Imaging Collaboration, "International Skin Imaging Collaboration," 2020. [Online]. Disponible : https://www.isic-archive.com/. [Accessed 7 Nov 2020].

[17] Palomino, N. L. S., & Concha, U. N. R. (2009). Técnicas de segmentación en procesamiento digital de imágenes. *Revista de investigación de Sistemas e Informática*, *6*(2), 9-16.

[18] Marcos Martín, Tecnicas Clásicas de Segmentación, 21 Mayo 2002 [imagen]

[19] Ruifrok, A. C., & Johnston, D. A. (2001). Quantification of histochemical staining by color deconvolution. *Analytical and quantitative cytology and histology*, *23*(4), 291-299.

[20] Kittler, J., & Illingworth, J. (1986). Minimum error thresholding. *Pattern recognition*, *19*(1), 41-47.

[21] Sahoo, P. K., Soltani, S. A. K. C., & Wong, A. K. (1988). A survey of thresholding techniques. *Computer vision, graphics, and image processing*, *41*(2), 233-260.

[22] Jiménez Hernández, S. (2016). *Aplicación de técnicas de procesado de imagen para las segmentación de núcleos en muestras histológicas humanas* (Doctoral dissertation).

[23] José Porras, Anotnio Morán, Clasification System Based On Computer Vision.

[24] Lorca, G., Arzola, J., & Pereira, O. (2010). Segmentación de Imágenes Médicas Digitales mediante Técnicas de Clustering. *Rev. Aporte Santiaguino*, 108-116.

[25] Drozdowicz, B., Bernasconi, G., Reyes, M., Saba, F., & Simón, G. (2005). Segmentación semiautomática de imágenes de resonancia magnética, basada en redes neuronales artificiales. *Ciencia, Docencia y Tecnología*, *16*(30), 117-155.

[26] KAYALIBAY, Baris; JENSEN, Grady; VAN DER SMAGT, Patrick. CNN-based segmentation of medical imaging data. *arXiv preprint arXiv:1701.03056*, 2017.

[27] http://exponentis.es/ejemplo-de-clustering-con-k-means-en-python [imagen]

[28] Takeyas, Bruno López. "Introducción a la inteligencia artificial." *Instituto Tecnológico de Nuevo Laredo. Web del autor: http://www. itnuevolaredo. edu. mx/takeyas* (2007).

[29] Kaplan, J. (2016). *Artificial intelligence: What everyone needs to know*. Oxford University Press.

[30] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, *40*.

[31] Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

[32] https://datos.gob.es/es/documentacion/tecnologias-emergentes-y-datos-abiertos-inteligencia-artificial [imagen]

[33] Liu, Y., Zhao, T., Ju, W., & Shi, S. (2017). Materials discovery and design using machine learning. *Journal of Materiomics*, *3*(3), 159-177.

[34] Salas, R. (2004). Redes neuronales artificiales. *Universidad de Valparaıso. Departamento de Computación*, *1*.

[35] Wang, B., & Gong, N. Z. (2018, May). Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)* (pp. 36-52). IEEE.

[36] Suthaharan S. Decision tree learning. In: *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning*. Boston, MA: Springer (2016). p. 237–269.

[37] https://rubialesalberto.medium.com/qu%C3%A9-es-underfitting-y-overfitting-c73d51ffd3f9 [imagen]

[38] J. Kolluri, V. K. Kotte, M. S. B. Phridviraj and S. Razia, "Reducing Overfitting Problem in Machine Learning Using Novel L1/4 Regularization Method," *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, Tirunelveli, India, 2020.

[39] Lucía Moreno González-Páramo, Universidad de Cantabria, 2017, Herramientas avanzadas de análisis de datos de aplicación en Ingeniería Civil.

[40] Portugal, I., Alencar, P., & Cowan, D. (2018). The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications*, *97*, 205-227.

[41] Taiwo Oladipupo Ayodele (February 1st 2010). Types of Machine Learning Algorithms, New Advances in Machine Learning, Yagang Zhang, IntechOpen.

[42]https://medium.com/@juanzambrano/aprendizaje-supervisado-o-no-supervisado-39ccf1fd6e7b [imagen]

[43] https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo[imagen]

[44] Hertz, J. A. (2018). *Introduction to the theory of neural computation*. CRC Press.

[45] Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*. Academic Press.

[46] Rico, C., Paredes, M., & Fernandez, N. (2009). Modelación de la estructura jerárquica de macroinvertebrados bentónicos a través de redes neuronales artificiales.Acta Biológica Colombiana,14(3), 71-96.

[47] Flórez-López, Raquel and J. F. Fernández. "Las redes neuronales artificiales: fundamentos teóricos y aplicaciones prácticas." (2008).

[48] Sarle, W. S. (1994). Neural networks and statistical models.

[49] Araujo, A., Pérez, J., & Rodriguez, W. (2018). Aplicacion de una Red Neuronal Convolucional para el Reconocimiento de Personas a Través de la Voz. In *Sexta Conferencia Nacional de Computación, Informática y Sistemas* (Vol. 11).

[50] Süsstrunk, S., Buckley, R., & Swen, S. (1999, January). Standard RGB color spaces. In *Color and Imaging Conference* (Vol. 1999, No. 1, pp. 127-134). Society for Imaging Science and Technology.

[51] https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/ [imagen]

[52] Edwards, M., & Xie, X. (2016). Graph based convolutional neural network. *arXiv preprint arXiv:1609.08965*.

[53] https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8 [imagen]

[54] Zhang, H., Weng, T. W., Chen, P. Y., Hsieh, C. J., & Daniel, L. (2018). Efficient neural network robustness certification with general activation functions. In *Advances in neural information processing systems* (pp. 4939-4948).

[55] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017, August). Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)* (pp. 1-6). IEEE.

[56] Y-Lan Boureau, Jean Ponce, and Yann Lecun. A theoretical analysis of feature pooling in visual recognition. In Proc. Int. Conf. Mach. Learning, 2010.

[57] https://medium.com/@cleverpysolutions/red-convolucional-en-pytorch-c499ae8af6ca [imagen]

[58] Robert Hecht-Nielsen ,Theory of the Backpropagation Neural Network, which appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989,Pages 65-93,9780127412528

[59] Chauvin, Y., & Rumelhart, D. E. (Eds.). (1995). *Backpropagation: theory, architectures, and applications*. Psychology press.

[60] JACOBS, Robert A. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1988, vol. 1, no 4, p. 295-307.

[61] http://www.bdhammel.com/learning-rates/ [imagen]

[62] HUTTER, Frank; HOOS, Holger; LEYTON-BROWN, Kevin. An efficient approach for assessing hyperparameter importance. En *International conference on machine learning*. PMLR, 2014. p. 754-762.

[63] Baldi, P., & Sadowski, P. J. (2013). Understanding dropout. *Advances in neural information processing systems*, *26*, 2814-2822.

[64] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Berg, A. C. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), 211-252.

[65] Torrey, L., & Shavlik, J. (2010). Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques* (pp. 242-264). IGI global.

[66] Anderson, G. W., & Castano, D. J. (1995). Measures of fine tuning. *Physics Letters B*, *347*(3-4), 300-308.

[67] Juan, R. Q., & Mario, C. M. (2011). Redes neuronales artificiales para el procesamiento de imágenes, una revisión de la última década. *RIEE&C, Revista de Ingeniería Eléctrica, Electrónica y Computación*.

[68] Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003, August). Best practices for convolutional neural networks applied to visual document analysis. (Vol. 3, No. 2003).

[69] Shan, Q., Li, Z., Jia, J., & Tang, C. K. (2008). Fast image/video upsampling. *ACM Transactions on Graphics (TOG)*.

[70] https://mi.eng.cam.ac.uk/projects/segnet/ [imagen]

[71] https://www.kaggle.com/getting-started/151903 [imagen]

[72] https://www.tensorflow.org/ [imagen]