ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Ingeniería de la Salud

Clasificación de imágenes de úlceras por presión mediante el uso de técnicas de aprendizaje profundo.

Classification of pressure ulcer images through the use of deep learning techniques.

Realizado por
Pablo Casado Gallardo
Tutorizado por
Rafael Marcos Luque Baena
Cotutorizado por
Francisco Javier Veredas Navarro
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, septiembre de 2019

Fecha defensa:

Fdo.: El Secretario del Tribunal

1. RESUMEN.

Resumen: Este Trabajo Fin de Grado ha consistido en un estudio del uso de redes convolucionales para la clasificación de imágenes de úlceras cutáneas producidas por presión.

Las úlceras cutáneas producidas por presión pueden parecer heridas sencillas pero en realidad son heridas de una elevada complejidad, las cuáles suponen uno de los principales costes para los sistemas sanitarios.

Para aplicar un tratamiento eficaz sobre este tipo de heridas, es necesario un diagnóstico fiable y preciso. Dicho diagnóstico, se realiza mediante la inspección visual por parte del personal sanitario, el cual está sujeto a una gran subjetividad por parte del operario.

El desarrollo de un sistema de clasificación automático, capaz de realizar una clasificación eficiente y fiable de este tipo de heridas, sería un gran avance, pues supondría una herramienta de gran utilidad para el personal médico, la cual reduciría los costes para los sistemas sanitarios y, lo que es más importante, se produciría una mejora en el tratamiento de los pacientes.

A lo largo del trabajo, se ha realizado el habitual flujo de trabajo completo para el desarrollo de un sistema inteligente.

Se ha comenzado con el diseño de varios sistemas convolucionales, los cuales se han entrenado y testeado, para asentar una base sólida en la primera fase del proceso. Tras realizar el test de los diferentes sistemas con imágenes desconocidas para ellos, se ha continuado con la aplicación de una técnica llamada *Data Augmentantion*, junto con diferentes cambios en las arquitecturas iniciales, teniendo como objetivo mejorar los resultados obtenidos por los primeros sistemas.

Acto seguido, se ha realizado un estudio de los resultados obtenidos por cada sistema. Se ha observado la evolución de ciertos parámetros que miden la eficiencia del sistema, como son la *Accuracy*, el *F1–SCORE* o el *Recall*. También se ha empleado la matriz de confusión para observar cómo clasifican los diferentes sistemas cada imágen dependiendo de la clase a la que pertenece.

En base a esta actuación, se ha seleccionado el sistema más prometedor

para la optimización de sus hiper parámetros mediante la aplicación de dos algoritmos de optimización, el *Grid Search* y el *Randomized Search*.

Para terminar, se ha realizado una reestructuración del problema de clasificación, diviéndolo en dos partes: primero, una clasificación entre *piel/ no piel* y después, la clasificación de los tejidos No Piel. Cada subproblema también ha sido optimizado, con el objetivo de encontrar el mejor resultado, posibilitar una comparación entre los dos enfoques y obtener la conclusión mas fiable.

Palabras clave: úlcera, aprendizaje profundo, red neuronal, convolución, red convolucional, clasificación, multiclase, binario, optimización, hiper parámetros.

Abstract: This Final Grade Project has consisted in a study about the use of convolutional networks to the classification of pressure ulcer images, following the usual work–flow in the development of a intelligent system.

The pressure ulcer can look like simple wounds however they are really high complexity wounds and they supose one of the main cost for the sanitary system.

To apply an affective treatment for this type of wounds, it is necessary an effective and accurate diagnosis. That diagnosis is made by the view of the care provider, which has a great subjectivity.

It has begun with the design of some convolutional systems. Furthermore, every initial system has been trained and tested to secure a solid base in the first steps of the development process. After testing the systems with some unknown images to them, a technique named Data Augmentation has been applied to the systems and some changes have been made over the initial architecture trying to improve the results obtained before.

Then, a studied has been made to every system. The evolution of some parameters such as the accuracy, the recall or F1–Score has been analyzed. In addition, the confusion matrix has been used to see how every image is classified.

Having in mind these results, the system more promising has been chosen to optimize its hyper parameters using two different optimization algorithms like the Grid Search and the Randomized Search.

To end, the initial classification problem has been restructured and it has been divided in two sub problems. The first part, it is a classification of skin or not skin images and the second part is a classification of the different tissues in the not skin class. Every sub problem has been optimized, trying to earn the best result as possible, to make possible a comparison between the two approaches and to have the most reliable conclusion.

Key words: Ulcer, deep learning, neuronal network, convolution, convolutional network, classification, multiclass, binary, optimization, hyper parameters.

Índice

Índice de figuras

Índice de cuadros

2. INTRODUCCIÓN.

Una úlcera cutánea producida por presión es una patología clínica que consiste en un daño localizado en la piel y en los tejidos subyacentes, la cual ha sido producida por los efectos de la presión, del cizallamiento y/o de la fricción. A pesar de la aparente sencillez de estas patologías, los procedimientos de diagnóstico, tratamiento y/o cuidado pueden suponer unos costes muy elevados para los sistemas sanitarios.

Para poder obtener un diagnóstico fiable, es crucial apoyarse en una evaluación precisa de la herida, y obtener así un tratamiento de éxito, ya que en algunos casos puede estar en juego la vida del paciente. Sin embargo, los procedimientos de diagnóstico que se siguen actualmente están basados en la evaluación visual de la herida por parte del personal médico, la cual puede carecer de la precisión necesaria para cumplir esta importante tarea. [1] [2]

En la actualidad, el problema ha sido abordado desde diferentes puntos de vista de la computación pero ningún enfoque ha conseguido dar una solución fiable al mismo. Así, aprovechando los avances en sistemas inteligentes y redes neuronales, se han aplicado nuevos algoritmos en el procesamiento de imágenes y técnicas de aprendizaje profundo para facilitar la detección y clasificación de los bordes que diferencian los diferentes tipos de tejidos que se pueden encontrar en este tipo de heridas, los cuales son muy importantes de delimitar para el correcto diagnóstico de dichas patologías.

Uno de los principales estudios en esta línea, publicados por los tutores de este TFG (Veredas et al., 2015), emplea estrategias de segmentación basadas en *mean-shift clustering* para la clasificación de tejidos en imágenes de úlceras producidas por presión, como son las redes neuronales, las máquinas de soporte vectorial y random forest. Estas técnicas presentan unos índices de eficacia de clasificación elevados aunque requieren un costoso proceso previo de extracción de características de color y pre–procesamiento de datos.

Por este motivo, el actual trabajo fin de grado propone el uso de estrategias de aprendizaje profundo, concretamente, el uso de redes convolucionales, para el diseño de clasificadores de tejidos en imágenes de úlceras producidas por presión.

Se ha elegido este tipo de redes neuronales debido a que son de tipo end-to-end, realizan una extracción automática de características de alto nivel de

abstracción y permiten clasificar imágenes "crudas", sin necesidad de realizar una fase previa de extracción e ingeniería de características. Además, los modelos de aprendizaje profundo constituyen hoy en día una de las técnicas más utilizadas en cuanto a clasificación automática de imágenes se refiere.

Todos los archivos de este Trabajo Fin de Grado se encuentran disponibles en el repositorio GitHub, en el url: `https://github.com/PabloCassado/Clasificacion-de-imagenes-ulcerosas`

3. PROBLEMA.

El problema que se afronta en el presente Trabajo Fin de Grado encaja dentro de un problema de clasificación multiclase, ya que cada imagen del conjunto de datos puede ser clasificada como 5 tipos diferentes de tejido.

Para acotar y simplificar el problema, cada imagen solamente podrá ser clasificada como un único tipo de tejido, es decir, solamente podrá pertenecer a una única clase. Se establece este criterio debido a que dentro de una misma imagen, podemos encontrar más de un tipo de tejido. Por lo tanto, estamos ante un problema ***single-label*** **de clasificación multiclase.**

Las clases que se van a tratar, se corresponden con los diferentes tejidos que podemos encontrar dentro de una úlcera cutánea producida por presión, los cuales son: piel, tejido de granulación, tejido de cicatrización, tejido necrótico y esfacelos, los cuales corresponden con las clases ***Skin, Granulation, Healing, Necrosis y Slough*** respectivamente.

El problema tiene una dificultad añadida y es el propio conjunto de datos con el que se cuenta para intentar resolver el problema. Las úlceras cutáneas son únicas, es decir, no hay dos úlceras iguales. Para aumentar la dificultad, las úlceras cutáneas no poseen una estructura definida como ocurre, por ejemplo, en una cara humana o en un símbolo, y por tanto, no se conoce de antemano que puede aprender a reconocer cada capa de la red debido a que no hay patrones o bordes dentro de la herida. Los únicos bordes que podemos encontrar, y de forma difusa, son las zonas en las que el tipo de úlcera cambia.

4. DATASET.

Un equipo médico realizó fotografías a color de las úlceras de pacientes que recibían asistencia sanitaria en sus domicilios. Fueron fotografiadas las úlceras localizadas en la cadera y el sacro de los pacientes, empleando la cámara digital Canon EOS 40D. Además, se empleó un objetivo Sigma EM–140 para obtener la mejor calidad posible y se realizaron las fotografías a una distancia aproximada de 30–40 cm del plano de la herida. Este objetivo minimiza los reflejos y sombras que pudieran aparecer en las imágenes [1].

Para minimizar el margen de error, la lente de la cámara estaba orientada en paralelo al plano de la herida. La lente Canon EF–S 60 mm f/2.8 macro USM fue utilizada para asegurar el buen enfoque de las imágenes a una distancia tan cercana de la herida.

Un grupo de expertos clínicos seleccionó 113 imágenes, las cuales consideraron adecuadas debido a la presencia de todos los tejidos significativos para la evaluación de una úlcera por presión (Ver Figura 1).

Figura 1: Imagen úlcera por presión tomada a un paciente.

Estas 113 imágenes seleccionadas constituyen el conjunto de datos con el que se realizará el presente Trabajo Fin de Grado.

Cada una de las 113 imágenes que componen el conjunto de datos, han sido segmentadas utilizando el algoritmo *k–nearest neighbor* calculado mediante una adaptación a aplicaciones de visión del *local–sensitive hashing algorithm*(LSH).

Para reducir los efectos de los reflejos de la luz, se aplicó un filtro mediana sobre las imágenes antes de que la segmentación hubiera comenzado.

Con objetivo de reducir el número de regiones resultantes del proceso de segmentación y limitar la complejidad de clasificación del problema, la resolución inicial de las imágenes de 1632 x 1224 *pixels* se redujo a 204 x 153 *pixels* por las limitaciones del proyecto que data de 2006. Esta reducción fue empíricamente reajustada, con la ayuda de expertos clínicos, para mantener el número de regiones con una cantidad de tejido significativo. Tras esto, se aplicó un algoritmo de *mean shift smoothing* para obtener una correcta conservación de los bordes de la imagen. (Ver Figura 2).

Tras este algoritmo, se utilizó tambien un algoritmo de crecimiento de regiones para conservar por completo los bordes que delimitaban las regiones que se obtuvieron de cada imagen. (Ver Figura 3)

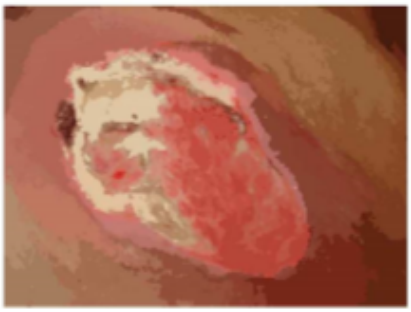

Figura 2: Imagen úlcera por presión tomada a un paciente

Figura 3: Resultado pre–procesado de la imagen

El número medio de regiones obtenidas por imagen fue aproximadamente de 150. Si se contaba con 113 imágenes, el resultado fue de 17007 regiones obtenidas tras la segmentación de todas las imágenes.

Con estas regiones obtenidas, el equipo de expertos realizó una segmentación a mano de las imágenes de úlceras por presión y se obtuvieron 15879 regiones, a las cuáles se les asignó una determinada etiqueta según el tipo

de tejido que se encontrase en su píxel central. A continuación, se procede a mostrar un ejemplo de las imágenes que forman el conjunto de datos del problema. (Ver Figura 4).

Figura 4: Ejemplo de imágen del conjunto de imágenes

Además, se aplicaron dos segmentaciones diferentes al conjunto de imágenes: una segmentación en regiones de tamaño 8x8 *pixels* y otra segmentación en regiones de tamaño 16x16 *pixels*.

Cada una de las imágenes que se han obtenido tras la segmentación, ha sido nombrada de la siguiente manera: *img_XX_reg_YY.jpg*. La etiqueta de cada imagen se encuentra contenida en el archivo *patterns_jpg_labels.csv*.

Dicho archivo está compuesto por dos columnas: en la primera columna se hace referencia a la imagen y en la segunda columna se hace referencia a la etiqueta de la imagen en cuestión.

Para asegurar que el tamaño de cada región generada es el adecuado y homogeneizar todas las regiones, de cada una de las regiones generadas tras la segmentación, se ha extraído una región de la misma a partir de su centroide. Esto se ha hecho para las regiones de tamaño 8x8 y 16x16. Además, cada región obtenida desde el centroide de la imagen, si había presencia de huecos en los bordes, estos se han rellenado con dos tipos de rellenado diferente.

El primer tipo de rellenado que se le ha aplicado a las imágenes es conocido como *random fill*, el cual consiste en rellenar los bordes de cada imagen utilizando valores aleatorios que estén comprendidos entre el valor mínimo y el valor máximo de la imagen en cuestión.

El segundo tipo de rellenado que se le ha aplicado a las imágenes es conocido como *zero fill*, el cual consiste en rellenar los bordes de la imagen en cuestión utilizando ceros.

Tras este proceso de segmentación y rellenado, se ha obtenido un total de 4 conjuntos de regiones. Cada uno de estos conjuntos ha sido almacenado en una carpeta, la cual ha sido nombrada con el tamaño de la segmentación que se ha realizado y el tipo de rellenado al que han sido sometidas las imágenes. Por tanto, tendremos los siguientes cuatro conjuntos de imágenes:

- 16x16 rand fill.
- 16x16 zero fill.
- 8x8 rand fill.
- 8x8 zero fill.

Ya de por sí, el tamaño del conjunto de datos es bastante pequeño, ya que solo se cuenta con un total de 15838 imágenes. El número de imágenes de cada tipo es el siguiente:

- Clase *Skin*: 8512 imágenes.
- Clase *Healing*: 2901 imágenes.
- Clase *Necrosis*: 371 imágenes.
- Clase *Granulation*: 2702 imágenes.
- Clase *Slough*: 1352 imágenes.

Como puede observarse, el conjunto de imágenes que se tiene está desbalanceado, es decir, todas las clases no tienen el mismo número de imágenes. Esto se debe a la diferencia de dificultad para encontrar cada tipo tejido. Por ejemplo, encontrar tejido necrótico en una úlcera cutánea es bastante extraño. Sin embargo, se encuentra tejido piel en cada úlcera cutánea que se trata.

5. PARTES DE UNA RED CONVOLUCIONAL.

Una red convolucional se divide en las siguientes partes [3]:

5.1. Capas.

Las capas son la unidad básica de la estructura de cualquier red neuronal. Una capa es un módulo de procesamiento de datos que toma como entrada un tensor con un determinado tamaño y devuelve a su salida uno o más tensores con el mismo o diferente tamaño que el tensor de entrada. La combinación de diferentes capas de forma lineal crean una red neuronal.

Hay diferentes tipos de capas. Dependiendo del tipo de problema que se intente resolver, se utilizan unas capas u otras. En este problema, un problema de clasificación multiclase de imágenes, los datos se encuentran almacenados en tensores de 3 dimensiones, por lo que se van a usar capas Conv2D para procesar dichos datos.

En nuestro caso, el tensor de entrada a la red tiene un tamaño de 16x16x3 y tras cada capa *MaxPooling* se puede observar como su tamaño se reduce a la mitad. (Ver Figura 8)

Cada capa está formada por unos parámetros denominados ***weights***, en los cuáles se alberga el "conocimiento"de esa capa, por tanto, todos los *weights* forman el conocimiento de la red.

Las principales capas que se van a usar para la creación de las redes convolucionales de este Trabajo Fin de Grado son las siguientes:

- Capa Convolucional 2D.
- Capa MaxPooling 2D.
- Capa Dropout.
- Capa Dense.

5.1.1. Operación Convolución.

Las redes convolucionales reciben este nombre debido a que utilizan la operación de convolución para realizar su aprendizaje. Esta operación se da en las capas *Convolucional2D)*. Tienen dos propiedades muy interesantes y útiles:

- Los patrones que aprenden son invariantes, es decir, el patrón aprendido por la capa es reconocido sin importar el lugar que ocupe en la imagen que se esté tratando.
- Pueden aprender jerarquías espaciales de patrones. Una primera capa convolucional aprende patrones locales como un borde, una segunda capa convolucional aprende patrones más extensos basándose en los patrones aprendidos por las capas convolucionales previas como se ve en Figura 5.

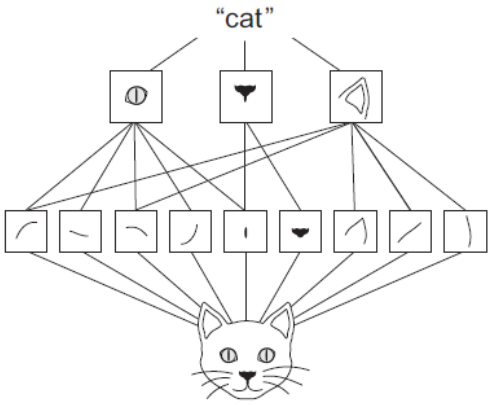

Figura 5: Funcionamiento de varias convoluciones.

La operación de **convolución** [4] consiste en ir tomando grupo de píxeles cercanos de la imagen de entrada y aplicar un filtro. Esta aplicación consiste en una operación matemática, el producto escalar. Por cada filtro que se aplica a la imagen de entrada, se genera una neurona oculta en la capa siguiente. Cada neurona detecta una característica significativa de la imagen.

5.1.2. Max Pooling

El *MaxPooling* consiste en extraer ventanas del tensor de entrada del tamaño indicado en la capa y devolver un tensor de salida con el máximo valor hallado en cada ventana generada durante la operación.

Su papel es reducir de forma agresiva el tamaño del tensor de entrada y así, reducir el número de neuronas de las capas ocultas de la red para evitar que la red tenga demasiadas neuronas que procesar.

El *MaxPooling* no es la única forma de reducir el tamaño del tensor. Existen otras formas de reducir el tamaño del tensor como el *AveragePooling*, el cual utiliza un valor promedio del canal. No obstante, funciona mucho peor debido a que es más informativo buscar un patrón en su máxima expresión, como hace el *MaxPooling*, que buscarlo en su expresión promedio, como ocurre con el *AveragePooling*.

5.1.3. Relu y Batch Normalization.

La capa *ReLu* es una capa de activación lineal y funciona como una función de activación rectificador lineal que no añade ningún *weight* al sistema. La función *ReLu* devuelve el tensor de entrada pero haciendo 0 los valores negativos del tensor.

La capa *Batch_Normalization* normaliza la activación de la capa anterior, aplicando una transformación que mantiene la activación media cercana a 0 y la activación de la desviación estándar cercana a 1.

5.1.4. Dropout.

Tras la serie de capas que componen la red neuronal, se añade una capa *Dropout*. Su función es eliminar un porcentaje de las neuronas de las capas internas de la red. Esto tiene como objetivo evitar el sobreentrenamiento de la red.

5.2. Clasificador.

Esta parte del sistema es el encargado, como bien dice su nombre, de clasificar las imágenes en las diferentes clases. En realidad, el clasificador es otra red densamente conectada, la cual está formada por una o varias capas *Dense*. Este es el modelo de clasificador que se va a usar en todas las redes convolucionales del presente Trabajo Fin de Grado.

Las capas *Densely connected* procesan vectores, los cuáles tienen dimensión 1. Los tensores generados en la salida de las redes convolucionales de este Trabajo Fin de Grado tienen dimensión 3. Por este motivo, antes de añadir el clasificador al modelo, se añade una capa *Flatten*, la cual convierte el tensor de dimensión 3 en un vector de dimensión 1.

5.3. Loss Function.

La *Loss Function* o función objetivo es la variable que se va a minimizar durante el entrenamiento. Representa la medida de éxito del problema que se está tratando. Por tanto, cuanto más cerca esté su valor de 0, más cerca estaremos de la resolución exitosa del problema. [5]

La elección de esta función objetivo es crucial. La red, por así decirlo, cogerá todos los atajos que encuentre hasta alcanzar la solución más óptima y/o adecuada para el problema, por lo que la función objetivo y el problema a tratar deben encajar a la perfección o la red acabará haciendo cosas que no queremos que haga.

Para este problema, la elección más correcta es bastante sencilla. Como se trata de un problema de clasificación multiclase, la *loss function* más adecuada es ***categorical_crossentropy***.

Dicha *loss function*, no es más que una activación ***softmax*** más una ***cross–entropy loss*** (Ver Figura 6). Si se usa esta *function loss*, en la salida de la red convolucional se obtendrá un vector en el que cada posición i, representa la probabilidad de que la imagen j pertenezca a la clase i. [6]

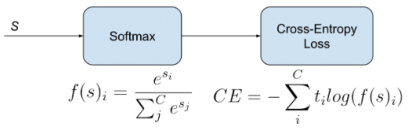

Figura 6: Ecuación *Categorical cross–entropy loss*

5.4. Optimizador.

El optimizador determina cómo ocurre el aprendizaje de la red, es decir, decide como los pesos de la red convolucional van a ser actualizados basándose en el resultado de la función objetivo.

Aquí se tiene un mundo de posibilidades debido a que en Keras hay implementados numerosos optimizadores. En este caso, se ha elegido el optimizador Adam debido a los siguientes motivos:

- Fácil de implementar.
- Alta eficiencia computacional.
- Apropiado para problemas con mucho ruido.
- Optimizador más rápido en obtener buenos resultados.
- Su tasa de aprendizaje varía según los weights de cada capa (Figura 7).

El algoritmo de optimización Adam es una extensión del gradiente descendiente estocástico que ha sido recientemente adoptado para el *deep learning* en aplicaciones de visión por computador y procesamiento natural del lenguaje. [7].

Adam es diferente al clásico gradiente descendiente estocástico. El gradiente descendiente estocástico mantiene un único *learning rate* para todas las actualizaciones de los *weights* y no cambia durante todo el proceso de entrenamiento del sistema. Sin embargo, en el algoritmo de optimización Adam, se mantiene un *learning rate*, a la vez que es adaptado durante el entrenamiento, para cada *weight* del sistema.

Los autores de este algoritmo de optimización lo describen como una combinación de las ventajas de otras dos extensiones del gradiente descendiente estocástico, las cuáles son:

- ***Adaptative Gradiente Algorithm (AdaGrad)*** que mantiene por cada parámetro del sistema un *learning rate* que mejora la actuación ante problemas con gradientes escasos.
- ***Root Mean Square Propagation (RMSProp)*** que también mantiene un *learning rate* para cada parámetro del sistema y además, es adaptado en base a la media de las magnitudes recientes de los gradientes del *weight* en cuestión.

Por tanto, Adam recibe los beneficios de los dos algoritmos de optimización nombrados anteriormente.

El algoritmo de optimización Adam adapta su *learning rate* basándose en la media del primer momento como ocurre en RMSProp pero Adam también hace uso de la media del segundo momento de los gradientes.

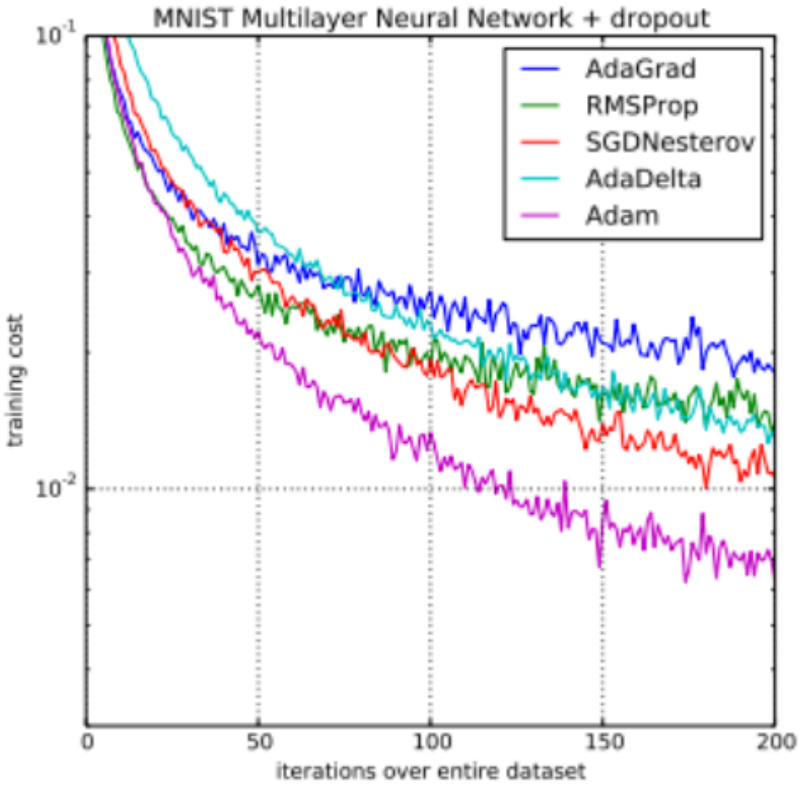

Figura 7: Comparación de Adam con otros optimizadores.

El algoritmo de optimización Adam en el campo del *deep learning* es muy

popular debido a que alcanza buenos resultados rapidamente. (Ver Figura 7)

En la Figura 7, se puede ver una demostración empírica de la convergencia de varios algoritmos de optimización. Se observa que el algoritmo de optimización Adam tiene el coste de entrenamiento, es decir los recursos necesarios para realizar el entrenamiento de la red, más pequeño de todos los algoritmos de optimazación usados para este experimento, y por tanto, es muy útil para problemas de *deep learning*.

Como se ha visto, el proceso de aprendizaje de la red convolucional ocurre mediante una retroalimentación entre el optimizador y los *weights*. Se procesa el conjunto de entrenamiento y se genera un valor de la función objetivo mediante el mapeo entre las predicciones de la red y los objetivos reales. Con ese valor, el optimizador actualiza los valores de los *weights* de cada capa de la red y se repite el proceso hasta encontrar el estado más óptimo de la red.

6. RED CONVOLUCIONAL BASE.

En esta sección se va a explicar todo el proceso de creación de la red neuronal inicial, además de justificar cada decisión que se ha tomado a lo largo de este proceso.

De esta red neuronal derivan el resto de redes neuronales que se han tratado a lo largo de este Trabajo Fin de Grado.

Para la implementación de todos los sistemas que se van a tratar en este Trabajo Fin de Grado, se va a utilizar Keras 2.2.4 debido a que se adapta perfectamente a las necesidades de desarrollo de este TFG.

Keras es una API red neuronal de alto nivel que se encuentra escrita en Python y es capaz de ejecutarse sobre otras librerías como TensorFlow, CNTK o Theano. Además, ha sido diseñada para ser utilizada en experimentación rápida. Keras también se adapta sin problema a un rápido prototipado y soporta redes convolucionales y redes recurrentes e incluso combinaciones de ambas. Y no menos importante, el tiempo de ejecución y cálculo de los resultados del sistema es muy parecido tanto en GPU como en CPU. [8]

La red que se ha elegido, es el ***Sistema 1 rand fill.*** Este sistema recibe

como entrada las imágenes del conjunto ***16x16 rand fill***.

La arquitectura del ***Sistema 1 rand fill*** depende directamente del tamaño de la imagen de entrada a la red. Como consecuencia, un sistema que tiene como entrada imágenes de tamaño 16x16, no tendrá la misma arquitectura que un sistema que recibe como entrada imágenes de tamaño 8x8. La arquitectura del Sistema 1 para una imagen de entrada de 16x6 u 8x8 se muestra en la Figura 8 y Figura 9 respectivamente.

Las Figuras 8 y 9 están formadas por 3 columnas, que indican lo siguiente:

- ***Layer (type)***: Es la sucesión de las distintas capas que forman la red convolucional. Entre paréntesis está indicado el tipo de capa en cuestión.
- ***Output Shape***: Indica las dimensiones del tensor de salida de la capa en cuestión. Como puede verse, las dimensiones del tensor no son las mismas en las Figuras 8 y 9 debido al tamaño de las imágenes que entran en cada red.
- ***Param***: Indica el número de parámetros que forman la capa. Las únicas capas que añaden parámetros a la red son las capas: *Conv2D, Batch Normalization 2, Dropout y Dense*.

Tras estas tres columnas, encontramos un breve resumen de la ariquitectura de la red en cuestión. En él se muestra el número total de parámetros totales que forman la red, los parámetros entrenables, es decir, los que pueden modificar su valor y los parámetros no entrenables, aquellos parámetros que no pueden modificar su valor.

6.1. Estructura base.

Como puede observarse en la Figura 9 y en la Figura 8, la red base, para ambos tamaños de entrada, está compuesta por un bloque básico que se repite hasta que se alcanza el tamaño objetivo del tensor que se trata. Dicho bloque es el que se muestra en la Figura 10. Este bloque básico está compuesto por una capa *Convolcional2D*, una capa *ReLu*, una capa *Normalization_Batch* y por último una capa *MaxPooling2D*.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 16, 16, 32)	416
re_lu_1 (ReLU)	(None, 16, 16, 32)	0
batch_normalization_1 (Batch	(None, 16, 16, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 32)	0
conv2d_2 (Conv2D)	(None, 8, 8, 64)	8256
re_lu_2 (ReLU)	(None, 8, 8, 64)	0
batch_normalization_2 (Batch	(None, 8, 8, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
conv2d_3 (Conv2D)	(None, 4, 4, 64)	16448
re_lu_3 (ReLU)	(None, 4, 4, 64)	0
batch_normalization_3 (Batch	(None, 4, 4, 64)	256
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 64)	0
conv2d_4 (Conv2D)	(None, 2, 2, 64)	16448
re_lu_4 (ReLU)	(None, 2, 2, 64)	0
batch_normalization_4 (Batch	(None, 2, 2, 64)	256
flatten_1 (Flatten)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 64)	16448
dense_2 (Dense)	(None, 5)	325

Total params: 59,237
Trainable params: 58,789
Non-trainable params: 448

Figura 8: Arquitectura Sistema 1 para imágenes 16x16.

Para formar la red convolucional, solamente hay que repetir el bloque básico nombrado anteriormente hasta que se alcanza el tamaño deseado de la imagen. Esto se ha hecho de la siguiente manera. (Ver Figura 11.)

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 8, 8, 32)	416
re_lu_1 (ReLU)	(None, 8, 8, 32)	0
batch_normalization_1 (Batch	(None, 8, 8, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 4, 4, 32)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	8256
re_lu_2 (ReLU)	(None, 4, 4, 64)	0
batch_normalization_2 (Batch	(None, 4, 4, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 2, 2, 64)	0
conv2d_3 (Conv2D)	(None, 2, 2, 64)	16448
re_lu_3 (ReLU)	(None, 2, 2, 64)	0
batch_normalization_3 (Batch	(None, 2, 2, 64)	256
flatten_1 (Flatten)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 64)	16448
dense_2 (Dense)	(None, 5)	325

Total params: 42,533
Trainable params: 42,213
Non-trainable params: 320

Figura 9: Arquitectura Sistema 1 para imágenes 8x8.

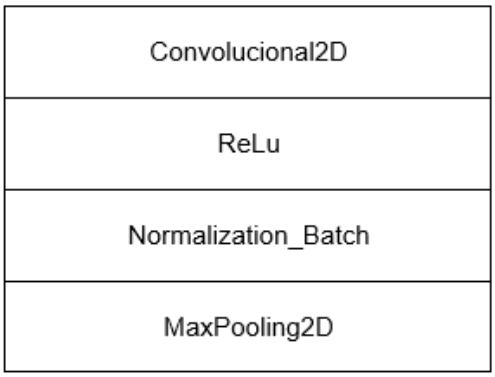

Figura 10: Bloque básico.

```
###CAPAS###
from keras import layers
from keras import models

model =models.Sequential()
model.add(layers.Conv2D(32,(2,2),padding='same', activation = 'relu', input_shape=(16,16,3)))
model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='zeros'
model.add(layers.MaxPooling2D(pool_size =(2,2)))
model.add(layers.Conv2D(64,(2,2),padding='same', activation ='relu'))
model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='zeros'
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(2,2), padding ='same', activation ='relu'))
model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='zeros'
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(2,2), padding ='same', activation ='relu'))
model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='zeros'

```

Figura 11: Código para la creación de la red neuronal con entrada 16x16.

6.2. Clasificador del Sistema 1.

El clasificador del Sistema 1 está formado por dos capas ***Dense.*** La primera de ellas consiste en 64 neuronas densamente conectadas. La característica de este tipo de capa es que cada neurona que compone la capa, está conectada a todas y cada una de las demás neuronas existentes en dicha capa.

La segunda y última capa del clasificador, está formada por 5 neuronas. Cada una de ellas, representa una de las clases del problema a tratar, es decir, cada neurona representa un tipo de tejido diferente. (Ver Figura 13)

Un aspecto a tener en cuenta es la activación de la última capa del clasificador. Su activación debe ser ***softmax***.

La activación ***softmax*** es una forma de regresión logística que normaliza un valor de entrada en un vector de valores que representa una distribución de probabilidad y cuyos valores suman 1. Los valores del vector de salida se encuentran en un rango [0,1], lo cual es de gran utilidad ya que es capaz de esquivar la clasificación binaria y es facilmente acomodable a cualquier cantidad de clases en nuestra red neuronal. [9]

La ecuación de la activación *softmax* se muestra en la Figura 12. [10]

Dicha activación devuelve un vector de tamaño 5 en el que cada poisición i representa la posibilidad de que la imagen j pertenezca a la clase i.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Figura 12: Ecuación *softmax*.

Otro aspecto a tener en cuenta es que el clasificador solamente es capaz de manejar vectores de una sola dimensión pero la salida de nuestra red es un tensor de 3 dimensiones. Esto se soluciona facilmente añadiendo una capa ***Flatten.*** Se dice que esta capa "aplana" los tensores y los convierte en vectores de una sola dimensión. (Ver Figura 13)

```
#Capa Flatten para convertir el tensor a vector de dimension 1
model.add(layers.Flatten())

#Capa Dropout para eliminar el 25% de las neuronas
model.add(layers.Dropout(0.25))

#Primera capa del clasificador. 64 neuronas densamente conectada con una activacion relu
model.add(layers.Dense(64, activation ='relu'))
#Segunda y última capa del clasificador.
#5 neuronas desament conectada porque tengo 5 clases para clasificar. skin, healing, necrotic, slough and granulation
model.add(layers.Dense(5, activation = 'softmax'))

```

Figura 13: Código del clasificador del sistema.

6.3. Optimizador del Sistema 1.

El optimizador es la útlima parte que forma el Sistema 1 pero no por ello es menos importante. Para el Sistema 1, se ha elegido el optimizador Adam y se ha utilizado su *learning rate* predeterminado de valor **0.001**. Este parámetro, como muchos otros se ajustará y optimizará más adelante. (Ver Figura 14)

Otro parámetro del clasificador que se debe ajustar correctamente es el ***loss.*** Se le debe indicar que estamos en una clasificación multiclase, y por lo tanto, debe ser igualado a ***categorical crossentropy.*** (Ver Figura 14)

```
1
2  from keras import optimizers
3
4  #Optimzador ADAM como una tasa de aprendizaje de 0.0001 que se va a centrar en la accuracy del sistema
5  model.compile(loss='categorical_crossentropy',
6                optimizer = optimizers.Adam(lr=1e-4),
7                metrics =['acc'])
8
9  #Estructura del sistema.
10 model.summary()
```

Figura 14: Código del optimizador del sistema.

7. ENTRENAMIENTO DEL SISTEMA.

El entrenamiento de la red convolucional es el procedimiento más importante en el proceso de creación de una red neuronal, ya que va a determinar la capacidad de clasificación que va a poseer el sistema. Para asegurar la calidad del entrenamiento y su correcta ejecuión, se van a realizar una serie de pasos previos.

En este apartado, se explica de forma detallada el proceso de entrenamiento que sigue cada sistema, así como algunos conceptos previos que son necesarios para entender algunas decisiones que se toman durante este proceso.

Este proceso solamente se va a describir para una única red convolucional debido a que para el resto, el proceso es idéntico. El sistema elegido para detallar todo este proceso es el sistema ***Sistema 1 rand fill.***, el cual recibe como entrada las imágenes del conjunto *16x16 rand fill.*

7.1. Separación del conjunto de datos en subconjuntos.

Para asegurar que la red convolucional creada ha conseguido aprender y reconocer cada tipo de tejido, se divide el conjunto de imágenes en tres subconjuntos: **Entrenamiento, Validación y Test.**

El **subconjunto Entrenamiento** es el más amplio y, en este caso, se ha decidido que abarque el 70 % de las imágenes totales de cada clase. Esto es así, debido a que cuanto mayor sea el tamaño de este subconjunto, mejor va a "aprender" la red a reconocer y clasificar los diferentes tipos de tejido. Como ya se puede suponer, este subconjunto va a ser utilizado para entrenar a la red.

El **subconjunto Validación** tiene un tamaño intermedio entre el subconjunto Entrenamiento y el subconjunto Test, ya que, en este caso, abarca el 20 % de las imágenes totales de cada clase. Este subconjunto es utilizado para comprobar el estado de la red durante su entrenamiento, es decir, para medir cuánto ha aprendido la red en cada época del proceso de entrenamiento.

El **subconjunto Test** es el más reducido de todos y está compuesto por el 10 % restante de imágenes del total de cada clase. Dichas imágenes no han sido vistas previamente por la red, por lo que son totalmente desconocidas para ella, y por tanto, este subconjunto es utilizado para comprobar cómo de buena es la red tras terminar el entrenamiento, es decir, este subconjunto es utilizado para comprobar si la red ha aprendido a clasificar cada imagen en su correspondiente clase. La separación en subconjuntos se encuentra en el fichero *genera_sets.ipynb*.

La separación de los subconjuntos se ha hecho mediante un bucle *for* para poder recorrer el contenido de toda la carpeta que contiene las imágenes. Para cada imagen encontrada, se busca el nombre de la correspondiente imagen dentro del fichero *patterns_jpg_labels.csv* y se guarda esa posición.

El archivo en cuestión contiene dos columnas: la primera columna contiene el nombre de la imagen y la segunda columna contiene la etiqueta que le ha sido asignada a dicha imagen.

A continuación, se comprueba en la segunda columna del fichero la etiqueta que se encuentra en la posición que se ha guardado y se copia la imagen en el destino que le corresponda a esa imagen.

Para saber a que subconjunto corresponde la imagen, se comprueba el valor de contador del tipo de imagen. Por lo tanto, si el contador no supera el 70 % de las imagenes totales de una determinada clase, la imagen se copia en el subconjunto Entrenamiento. Si el contador supera dicho porcentaje pero no supera el 90 % del total de dicha clase, la imagen se copia en el subconjunto Validación. Y por último, si el contador supera el 90 % de las imágenes totales de dicha clase, la imagen se copia en el subconjunto Test.

Además, se han realizado diferentes pruebas para detectar posibles fallos en la la división de los subconjunto como:

- Comprobación de si existen imágenes repetidas dentro de un mismo

subconjunto.

- Solapamiento entre subconjuntos.
- Ausencia de algunas imágenes.

Las diferentes pruebas realizadas a los subconjuntos se encuentran en el fichero *comprobación.ipynb*. Los errores han sido corregidos mediante la eliminación o inserción del fichero de forma manual.

Para proceder al entrenamiento de la red convolucional, se le debe indicar al sistema donde se encuentran los diferentes subconjuntos dentro de la máquina. No es necesario indicarle al sistema donde se encuentran las diferentes clases, ya que entiende que cada carpeta existente dentro de la carpeta de cada subconjunto es una clase diferente. Por tanto, el código para formar la ruta de cada subconjunto es el sisguiente (Ver Figura 15.)

2. DIRECTORIOS

```
#Carpeta Principal
base_dir = '/Users/Pablo/Desktop/IngBio/TFG/Imagenes/set_16x16_rand_fill'

#directorio SUBCONJUNTO ENTRENAMIENTO
train_dir = os.path.join(base_dir,'train')

#directorio SUBCONJUNTO VALIDACIÓN
validation_dir = os.path.join(base_dir,'validation')

#Directorio SUBCONJUNTO TEST
test_dir = os.path.join(base_dir,'test')

```

Figura 15: Formación de la ruta para cada subconjunto.

7.2. Underfitting y Overfitting

Como se ha dicho en el apartado anterior, cuanto aprende la red convolucional, depende directamente de la duración del entrenamiento. Se puede pensar que si el entrenamiento tiene una duración suficientemente larga, la red aprenderá sin problema a reconocer cada tipo de tejido. Esto es cierto pero solamente aprenderá a reconocer las úlceras del subconjunto de Entrenamiento. Si introducimos en la red una imagen desconocida para ella, la red no será capaz de clasificar correctamente la imagen. [11]

Este fenómeno es conocido como ***overfitting*** o sobreajuste. Ocurre cuando la red neuronal ha extraído demasiadas características propias del subconjunto de Entrenamiento y se ha especializado en reconocer las imágenes que

lo componen, en otras palabras, la red no ha conseguido aprender el concepto general de cada tipo de tejido que encontramos en este tipo de heridas.

Para combatir este fenómeno, únicamente hay que ajustar la duración del entrenamiento del sistema y además, en este Trabajo Fin de Grado, se aplicarán diversas técnicas que se explicarán con más detalle en el momento de su aplicación.

Por otro lado, también se puede dar el fenómeno contrario, el ***underfitting***. Como el *overfitting*, está originado por la duración del entrenamiento del sistema. Ocurre cuando el entrenamiento de la red no ha sido suficientemente largo y el sistema no ha tenido tiempo para extraer todas las características necesarias para clasificar correctamente cada tipo de tejido.

En este caso, la solución a este problema es bastante sencilla. Simplemente se le debe dar más tiempo al sistema para extraer dichas características del conjunto de Entrenamiento, es decir, debe aumentarse la duración del entrenamiento.

7.3. Pre Procesado de las imágenes.

Antes de iniciar el entrenamiento, se debe realizar un preprocesado de las imágenes. El principal preprocesado es realizar un reescalado de la imagen para que el canal se encuentre entre 0 y 1 en lugar de 0 y 255.

Este cambio se realiza mediante el uso de una clase ya existente en Keras, la clase ***ImageDataGenerator***. Esta clase nos permite aplicar numerosas técnicas de preprocesado de imágenes como rotaciones, desplazamientos o zooms en una zona determinada de la imagen. Esta clase será muy útil más adelante, cuando se vaya a aplicar la técnica de *Data Augmentation*.

Una vez construida la clase ImageDataGenerator con el preprocesado de las imágenes, se crea un iterador, conocido como **generador** para el subconjunto de imágenes que va a tratar. El iterador actuará sobre el *batch* de imágenes que se le indique.

Como se tienen las imágenes almacenadas en un directorio, se utiliza la función ***flow_from_directory()***. Además, se pueden especificar otros parámetros como el tamaño objetivo de las imágenes, el tamaño del batch o

el tipo de clasificación que se va a utilizar.

Un parámetro muy importante que se debe configurar de forma adecuada, es el parámetro ***Shuffle*** pero solamente en el generador del subconjunto Test. Este parámetro le indica al generador si se desea que se "barajen" las imágenes tras cada batch. Este parámetro tiene que tener asignado el valor ***False*** porque si no es así, se producirán problemas cuando se utilice el generador para probar el sistema.

Dicho problema consiste en que la imagen que se introduce en el sistema para su clasificación, puede haber sido ya clasificada, y por tanto, se obtendrán datos incorrectos en las predicciones realizadas por el sistema.

Surge otro inconveniente a la hora de medir la precisión del problema. Si se barajan las imágenes tras cada *batch*, no hay forma de saber cómo han sido barajadas, por lo que no podremos calcular la precisión del sistema, ya que las etiquetas que nos produce el generador, no corresponden con el orden en el que las imágenes han sido introducidas en el sistema.

Un aspecto muy importante de estos iteradores es que solamente actúan en tantas imágenes como indica su tamaño de batch y, solamente devuelven en su salida tantas imágenes procesadas, además de sus correspondientes etiquetas, como su tamaño de batch indica. Por ejemplo, si el iterador tiene un tamaño de batch de valor 20, el iterador devolverá las 20 primeras imágenes con la respectiva etiqueta de cada imagen.

Una vez que se han creado estos iteradores, ya se puede entrenar el sistema con el uso de la función ***fit_generator()***, ya que estamos usando generadores. Si no usaramos generadores, deberíamos usar la función *fit()*.

7.4. Entrenamiento del Sistema.

Una vez establecidas todas las partes del sistema, así como los iteradores y generadores para el pre–procesado de las imágenes, se procede al entrenamiento del sistema. [12]

Para ello, se utiliza la función ***fit_generator()***. En esta función, se le indica al sistema qué generador se usa para el entrenamiento del sistema y cual es el generador que se usa para la validación del sistema durante el entre-

namiento. En la Figura 16, se puede observar como se establece el generador ***train_generator*** para entrenar el sistema y el ***validation_generator*** para la validación del sistema.

Dentro de esta función encontramos otro parámetros. Los más importantes son los siguientes: (Ver Figura 16)

- ***Epochs***: También conocidas como etapas. Es el número de veces que el sistema va a recorrer el subconjunto de Entrenamiento. Establece la duración del entrenamiento.
- ***Steps per epoch***: Determina cuantos batchs van a ser visto por el sistema en cada época. Este parámetro debe ser establecido como el tamaño del iterador del subconjunto de Entrenamiento. Así nos aseguramos que el sistema recorra todo el subconjunto de Entrenamiento.

```
#Entrenamiento del sistema
history = model.fit_generator(
    #Generador utilizado para entrenar el sistema.
    train_generator,
    #Cuantos batches va a ver el sistema en cada época
    steps_per_epoch=len(train_generator),
    #Generador utilizao para la validación del sistema.
    validation_data=validation_generator,
    #Batches vistos por el sistema para cada validación.
    validation_steps=len(validation_generator),
    #Número de épocas.
    epochs=18)

#Guardo el estados del sistema, es decir, los weights de cada capa
model.save('16x16_rand_fill.h5')

#Muestro el estado del entrenamiento
import matplotlib.pyplot as plt

%matplotlib inline

#Parámetros a mostrar: accuracy, loss, validation accuracy y loss accuracy
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Figura 16: Entrenamiento del sistema.

El estado del sistema, es decir los *weights* de cada capa, se guarda con la función ***save()***. Se genera un archivo de formato h5 con el nombre especificado en la misma dirección que los archivos del sistema en cuestión. Cada archivo h5 generado, se nombra con el nombre del sistema.

Un archivo **.h5** es un fichero de datos guardado en *Hierarchical Data Format (HDF)*. Estos ficheros contienen arrays multidimensionales de datos científicos. Este tipo de ficheros son comumente usados en investigaciones científicas. [13]

8. TEST DEL SISTEMA.

Tras el entrenamiento del sistema, se debe realizar una prueba final para comprobar como se comporta el sistema ante imágenes que nunca antes ha visto. En este punto es cuando al fin se utiliza el subconjunto Test. [12]

Recordemos que el subconjunto Test es una pequeña parte de las imágenes del conjunto de datos que se ha reservado para evaluar el comportamiento del sistema tras su entrenamiento.

Para realizar las predicciones del sistema, se utiliza la función ***predict_generator()***. Esta función recibe como parámetros de entrada, el generador del subconjunto Test y los *steps*, los cuales tienen la misma función que el parámetro *steps_per_epoch* que hemos visto en la función *fit_generator()*.

La función **predict_generator** genera un array de 5 posiciones y en cada posición i se encuentra la probabilidad de que la imagen en cuestión pertenezca a la clase i. Este vector no es útil para hacer un análisis de resultados y no es admitido por las funciones que se van a usar.

Para solucionar este problema, se utiliza la función ***argmax()*** que devuelve la posición más alta de cada array de las predicciones generadas por el sistema. Además, una ventaja de esta función es que maneja arrays y devuelve un array en su salida. Por tanto, la salida de la función ***argmax()*** es un array que contiene en cada posición i, la predicción de nuestro sistema para la imagen i, es decir, en qué clase clasifica cada imagen del subconjunto Test. (Ver Figura 17).

```
import numpy
#Reseteamos el sgenerador del subconjunto TEST
test_generator.reset()

#Utilizamos la funcion predict_generator para pasarle las imégenes al sistema
#Steps tiene el valor del tamaño del generador del subconjunto Test
Y_pred = model.predict_generator(test_generator, steps = 1593/20)

#Etiquetas reales, es decir, lasetiquetas generadas por el generador
classes = test_generator.classes[test_generator.index_array]

#Las predicciones producidas por el sistema son un array de 5 posiciones y en cada posicino se alberga a posibilidad de
#que la imagen corresponda a esa clase.

#Utilizo la funcion argmax para encontrar en cada vector la posicino con la probabilidad más altay generar un vector con
#esas posiciones. Ese vector corresponde con las etiquetas generadas por el sistema para el subconjunto Test
y_pred = numpy.argmax(Y_pred, axis=-1)
```

Figura 17: Predicciones del Sistema.

9. ANÁLISIS DE RESULTADOS.

Tras realizar las predicciones del sistema, se procede a comparar las posiciones del array producido por el generador y el array que ha generado el sistema entrenado. Si en la posición i de ambos arrays el valor es el mismo, quiere decir que el sistema ha clasificado correctamente dicha imagen. Si el valor difiere, esto indica que la imagen no ha sido clasificada correctamente. Se cuenta el número total de coincidencias. Tras recorrer ambos arrays por completo, se divide el número de coincidencias por el tamaño del subconjunto Test. Así obtenemos la precisión general del sistema. (Ver Figura 18)

```
#ANALISIS DE RESULTADOS
import sklearn
from sklearn.metrics import confusion_matrix

#Matriz de confusion
print(confusion_matrix(test_generator.classes[test_generator.index_array],y_pred))
target_names = ['Granulation', 'Healing', 'Necrosis','Skin','Slough']

#Informe de clasificacion. Me devuelve parámetros como F1-score, recall o precision
print(sklearn.metrics.classification_report(test_generator.classes[test_generator.index_array], y_pred, target_names=target_

#Balance de precicion de las clases.
print(sklearn.metrics.balanced_accuracy_score(test_generator.classes[test_generator.index_array], y_pred, sample_weight=None
```

Figura 18: Código para el análisis de resultados.

Esta medida no es la más adecuada para determinar como de buena ha sido la actuación del sistema debido a que el *dataset* está desbalanceado. Por ello, se procede a realizar un estudio en profundidad de otros aspectos del sistema, los cuáles se van a explicar a continuación.

9.1. Parámetros de estudio.

En este apartado se procede a realizar una explicación de los parámetros que se van a estudiar para evaluar la eficiencia de cada sistema. El objetivo de este apartado es ofrecer una visión más adecuada y profunda sobre la actuación de los sistemas que se manejan.

9.1.1. Matriz de Confusión.

Todos los parámetros relevantes para el estudio de la eficiencia de una red convolucional derivan de esta matriz. [14]

La matriz de confusión es una herramienta de vital importancia para medir la eficiencia de clasificación de un algoritmo. En ella se muestra cómo está clasificando el sistema, es decir, es una matriz donde se representan las salidas del sistema y describe la actuación completa del sistema.

La forma genérica de una matriz de confusión para un problema binario se muestra en la Tabla 1.

	Predicción Negativa	Predicción Positiva
Etiqueta Negativa	TN	FP
Etiqueta Positiva	FN	TP

Tabla 1: Matriz de Confusión.

- **True Positives(TP):** Son los casos en los que la predicción del sistema fue SÍ y etiqueta era SÍ.
- **True Negatives (TN):** Son los casos en los que el sistema predijo NO y la etiqueta es NO.
- **False Positives (FP):** Son los casos en los que el sistema predijo SÍ y la etiqueta es NO.
- **False Negatives (FN):** Son los casos en los que el sistema predice No y la etiqueta es SÍ.

Como se observa en Tabla 1, los aciertos en la clasificación por parte del sistema se encuentran en la diagonal de la matriz mientras que los errores en la clasificación se representan fuera de la diagonal. Por tanto, cuanto más elevado sea el valor de la diagonal principal de la matriz y más cercano a 0 sean los valores fuera de la diagonal, mejor es la actuación del sistema.

9.1.2. Parámetros derivados de la Matriz de Confusión.

Los parámetros que se van a estudiar para evaluar la eficiencia de las redes convolucionales son los siguientes:

- **Accuracy**: Es la proporción entre el total de predicciones acertadas y el número de imágenes de dicha clase. Solamente funciona bien si el conjunto de datos está balanceado.

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

- **Recall**:Es el coeficiente entre el número de predicciones acertadas entre todos los casos relevantes, es decir FP y TP.

$$Recall = \frac{TP}{TP + FN}$$

- **F1-SCORE**: Combinación entre *Accuracy y Recall.*

$$F1 - SCORE = \frac{2TP}{2TP + FN + FP}$$

- **Balanced Accuracy:** Se define como la media del parámetro Recall obtenido en cada clase. Es útil cuando el se emplea un conjunto de datos desbalanceado. [15]

9.2. Resultados del Sistema 1 con el conjunto 16x16 rand fill.

En este apartado se van a analizar los resultados del Sistema 1 con el conjunto 16x16 rand fill, variando la duración del entrenamiento, es decir, variando el valor del parámetro *epochs*.

El procedimiento que se va a seguir es el mismo para todos los sistemas, y es el siguiente. Primero se entrena el sistema durante un elevado número de etapas, buscando su sobre entrenamiento. Tras ello, se observan las gráficas generadas, especialmente la gráfica del parámetro *loss*.

El *overfitting* se produce cuando el error del parámetro *loss* para el subconjunto de Entrenamiento sigue reduciéndose, mientras que el error del parámetro *loss* para el subconjunto Validación comienza a elevarse. Por este motivo, la gráfica que muestra la evolución del parámetro *loss* es tan importante.

9.2.1. Entrenamiento con 80 etapas.

Los resultados obtenidos tras un entrenamiento de 80 etapas con dicho conjunto de imágenes se muestra en la Tabla 2.

Parámetros del entrenamiento	
Loss	0.25
Accuracy	0.91
Validation Loss	0.92
Validation Accuracy	0.78
Accuracy	0.71
Balanced Total Accuracy	0.64

Tabla 2: Resultados del entrenamiento durante 80 etapas.

A continuación, se muestra la evolución del valor de los parámetros *Accuracy y Loss* durante el entrenamiento en la Figura 19 y la Figura 20 respectivamente.

Como puede observarse en la Figura 19, la precisión del Sistema 1 sobre el subconjunto de Entrenamiento es muy elevada, alcanzando un valor del 91 %. Sin embargo, la precisión sobre el subconjunto Validación asciende rápidamente en las primeras etapas de entrenamiento pero se estabiliza entorno al 78 %.

Si se mira la Figura 20, se observa que el valor de la función loss para el subconjunto de Entrenamiento disminuye constantemente hasta que finaliza el entrenamiento, alcanzando un valor de 0.25. Por otro lado, el valor de la

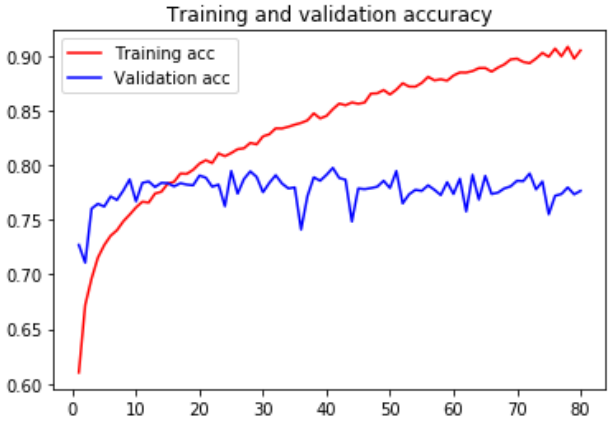

Figura 19: Evolución del parámetro Accuracy durante el entrenamiento.

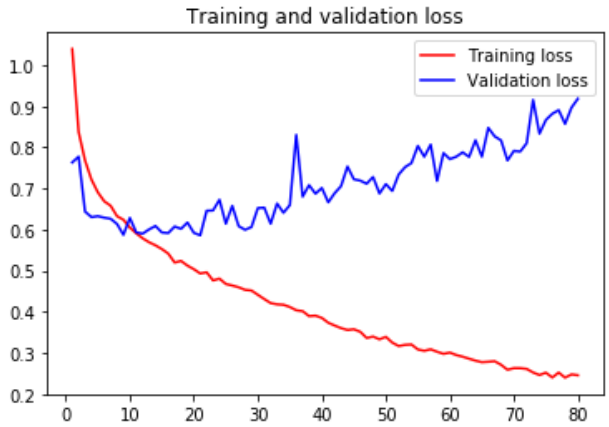

Figura 20: Evolución del parámetro Loss durante el entrenamiento.

función loss para el subconjunto Validación disminuye rápidamente en las primeras etapas pero a partir de la etapa 20 aproximadamente empieza a ascender.

Esto es una clara señal de ***overfitting*** en el sistema. Para combatirlo, se reduce el entrenamiento del sistema a una duración cercana al número de etapas en las que el valor de la función loss para el subconjunto Validación empieza a aumentar.

9.2.2. Entrenamiento durante 18 etapas.

Los resultados obtenidos tras un entrenamiento de 80 etapas con dicho conjunto de imágenes se muestra en la Tabla 3

Parámetros del entrenamiento durante 18 etapas	
Loss	0.51
Accuracy	0.80
Validation Loss	0.57
Validation Accuracy	0.80
Accuracy	0.70
Balanced Total Accuracy	0.65

Tabla 3: Resultados del entrenamiento durante 18 etapas.

A continuación, se muestra la evolución del valor de los parámetros *Accuracy y Loss* durante el entrenamiento en la Figura 21 y la Figura 22 respectivamente.

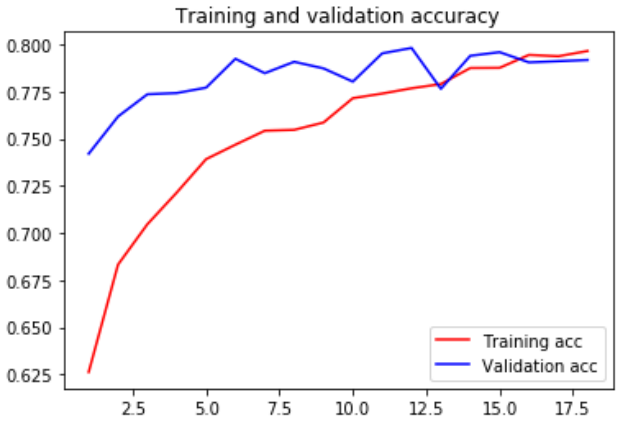

Figura 21: Evolución del parámetro Loss durante el entrenamiento.

Como se puede observar en la Figura 21, la gráfica correspondiente a la precisión del subconjunto de Entrenamiento tiene una clara tendencia ascendente y es siempre creciente. En cambio, si se mira el valor de la precisión sobre el subconjunto Validación en la Figura 21, se observa que su valor incrementa en las primeras etapas del entrenamiento y luego oscila en torno al 80 % de precisión.

Si se mira la Figura 22, el valor de la función *Loss* para el subconjunto Entrenamiento es siempre descendente. Por otro lado, el valor de la función

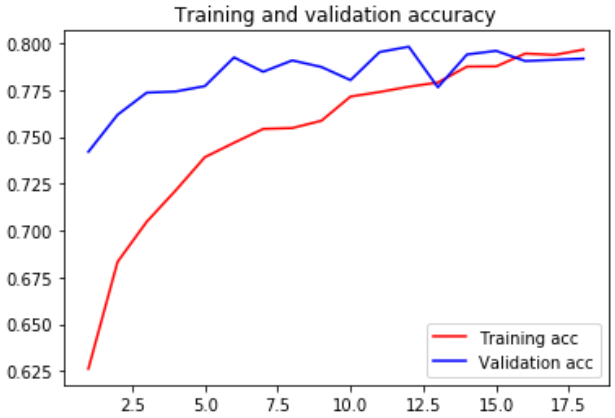

Figura 22: Evolución del parámetro Loss durante el entrenamiento.

Loss para el subconjunto Validación, se estabiliza a partir de la etapa 10 aproximadamente. Esto quiere decir que se debe cortar el entrenamiento, ya que el sistema no va a aprender ninguna característica general, si no que a partir de este punto solamente va a aprender características propias del subconjunto Entrenamiento, es decir, se va a empezar a producir el *overfitting*.

- Test del Sistema.

Para realizar el test de la red, le pasamos el subconjunto Test y vemos los resultados que se obtienen. Como se ha dicho anteriormente, el análisis del rendimiento comienza con la Matriz de Confusión del Sistema 1, la cual se muestra en la siguiente matriz.

$$\begin{pmatrix} 193 & 17 & 8 & 23 & 34 \\ 40 & 100 & 12 & 99 & 34 \\ 0 & 0 & 32 & 4 & 3 \\ 69 & 47 & 1 & 724 & 12 \\ 4 & 2 & 38 & 23 & 70 \end{pmatrix}$$

En la Figura 9.2.2 están representadas las clases *Granulation, Healing, Necrosis, Skin y Slough* empezando por la primera fila de la matriz. Ocurre igual para cada columna de la matriz de confusión. Esto será así para todas las matrices que se vean en este Trabajo Fin de Grado a no ser que se indicque lo contrario.

A continuación, se detalla un ejemplo de cual es el significado de cada fila de la matriz de confusión. Para ello, se procede a explicar la primera fila de la matriz representada en la Figura 9.2.2.

En la posición [1,1] de la matriz se representa cuantas imágenes etiquetadas como *Granulation*han sido clasificadas dentro de esa clase. En la posición [1,2] se muestra la cantidad de imágenes etiquetadas como *Granulation* han sido clasificadas dentro de la clase *Healing*. Así, hasta llegar al final de la fila.

Class	Accuracy	Recall	F1-SCORE	SUPPORT
Granulation	0.63	0.71	0.67	272
Healing	0.60	0.34	0.44	292
Necrosis	0.36	0.82	0.50	93
Skin	0.83	0.85	0.84	853
Slough	0.44	0.51	0.47	137
Micro Avg	0.70	0.70	0.70	1593
Macro avg	0.57	0.65	0.58	1593
Weigthed avg	0.71	0.70	0.70	1593

Tabla 4: Resultados del Sistema 1 con el conjunto 16x16 rand fill.

Antes de comenzar con el análisis de los resultados obtenidos por los diferentes sistemas, aclarar que el parámetro ***SUPPORT*** representar el total de las imágenes de una determinada clase dentro del subconjunto Test.

Como puede observarse en la Tabla 4, la cantidad de imágenes de cada clase está directamente relacionada con el valor de los parámetros.

La clase *Skin* es la clase con más imágenes, 853 imágenes para ser exactos, y presenta los mejores parámetros, los cuales tienen un valor superior al 80 %.

Un caso destacable es la clase *Necrosis* que presenta solamente 39 imágenes en total pero tiene un Recall del 84 %. Esto se explica debido a que el sistema clasifica correctamente la mayoría de las imágenes debido a que normalmente el tejido necrótico presenta un color cercano al negro, el cual es fácilmente distinguible de los colores rosáceos y/o blanquecinos del resto de tejidos. Sin embargo, presenta muy mala Accuracy y muy mal F1-SCORE debido a que una clasificación errónea reduce mucho el valor de estos parámetros. Los fallos en esta clase seguramente se deban a la presencia de otro tipo de tejidos dentro de la imagen.

Para el resto de clases, el clasificador no es capaz de distinguir completamente una clase de otra ya que las confunde, principalmente la clase *Healing* que la confunde gravemente con la clase *Skin*. Esto puede ser debido a que el tejido en sanación está cerca de ser piel y, si se añade la presencia de piel en la imagen, el sistema es más propenso a confundirlas.

9.3. Resultados del Sistema 1 con el cojunto 16x16 zero fill.

En este apartado se van a explicar los resultados del sistema 1 con el conjunto 16x16 zero fill variando la duración del entrenamiento.

9.3.1. Entrenamiento durante 30 etapas.

Con un entrenamiento de 30 etapas, los parámetros del sistema que se han obtenido se muestran en la Tabla 5.

Parámetros del entrenamiento durante 30 etapas	
Loss	0.44
Accuracy	0.82
Validation Loss	0.54
Validation Accuracy	0.80
Accuracy	0.71
Balanced Total Accuracy	0.68

Tabla 5: Resultados del entrenamiento durante 30 etapas.

Como puede observarse en la Figura 24, el sistema empieza a sobre entrenarse a partir de la etapa 15 del entrenamiento. Ocurre el mismo fenómeno que en el sistema 1 con el conjunto 16x16 rand fill.

9.3.2. Entrenamiento durante 15 etapas.

Con un entrenamiento de 15 etapas, los parámetros obtenidos se muestran en la Tabla 6

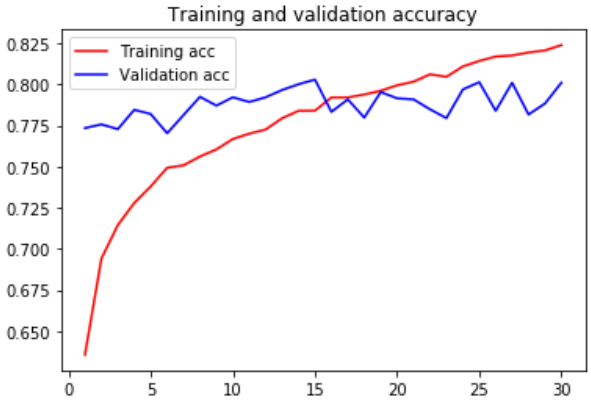

Figura 23: Evolución del parámetro Accuracy durante el entrenamiento.

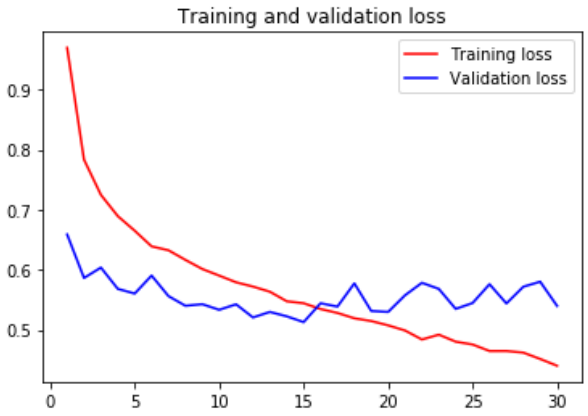

Figura 24: Evolución del parámetro Loss durante el entrenamiento.

A continuación, se muestra la evolución de la *Accuracy* y de *Loss* a lo largo del entrenamiento del Sistema 1 en la Figura 25y 26 respectivamente.

Como se observa en la Figura 26, el valor de loss para el subconjunto Training va reduciendo su valor conforme van avanzando las etapas del entrenamiento. Por otro lado el valor del loss para el subconjunto Validation empieza a ascender en la etapa 12, ahí, incluso unas etapas antes, se debe parar el entrenamiento del sistema.

- Test del Sistema 1 con el conjunto 16x16 zero fill.

Repetimos el proceso que hemos realizado en el Test con el conjunto

Parámetros del entrenamiento durante 15 etapas	
Loss	0.54
Accuracy	0.79
Validation Loss	0.61
Validation Accuracy	0.78
Accuracy	0.70
Balanced Total Accuracy	0.63

Tabla 6: Resultados del entrenamiento durante 15 etapas.

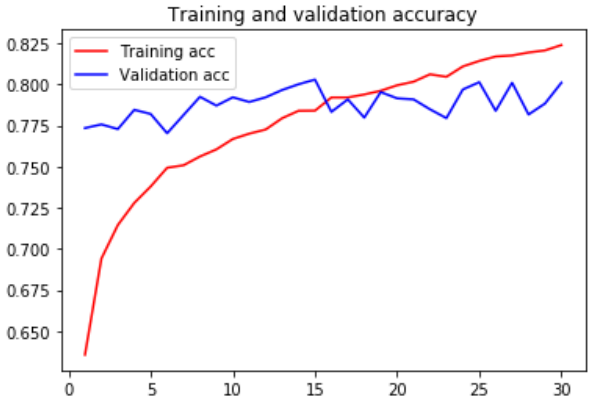

Figura 25: Evolución del parámetro Accuracy durante el entrenamiento de 15 etapas.

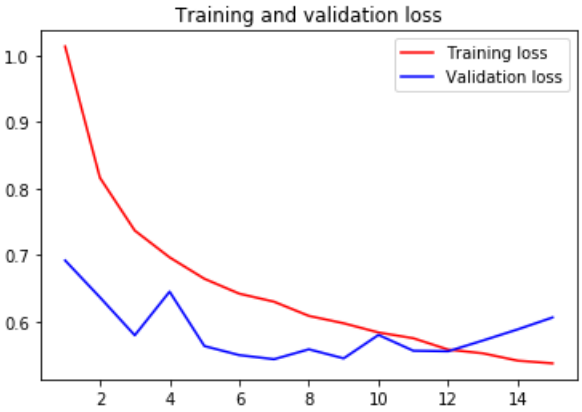

Figura 26: Evolución del parámetro Loss durante el entrenamiento de 15 etapas.

random fill. La matriz de confusión que obtenemos es la siguiente

$$\begin{pmatrix} 186 & 26 & 5 & 16 & 39 \\ 46 & 112 & 8 & 99 & 27 \\ 0 & 0 & 36 & 1 & 2 \\ 59 & 50 & 0 & 727 & 17 \\ 5 & 5 & 43 & 10 & 74 \end{pmatrix}$$

Los parámetros obtenidos tras la evaluación del Sistema 1 se muestran en la Tabla 7

Class	Accuracy	Recall	F1-SCORE	SUPPORT
Granulation	0.63	0.68	0.65	272
Healing	0.58	0.38	0.46	292
Necrosis	0.39	0.92	0.50	55
Skin	0.85	0.85	0.85	853
Slough	0.47	0.54	0.50	137
Micro Avg	0.71	0.71	0.71	1593
Macro avg	0.58	0.68	0.60	1593
Weigthed avg	0.72	0.71	0.71	1593

Tabla 7: Resultados del Sistema 1 con el conjunto 16x16 zero fill.

Como se puede observar en la Tabla 7, los resultados obtenidos con este conjunto apenas han variado respecto a los resultados obtenidos con el conjunto anterior.

La clase con mejores parámetros sigue siendo la clase *Skin*. Si miramos la columna SUPPORT, se ve que esta clase cuenta con más del 50 % del subconjunto Test. La única diferencia es que la clase *Necrotic* presenta un 91 % de Recall pero el resto de parámetros han empeorado. Además, los valores para el resto de clases apenas han variado, por tanto, se puede decir que no se ha obtenido una mejoría significativa cambiando de tipo de rellenado de bordes.

9.4. Resumen de resultados del Sistema 1.

Para facilitar la comparación de los resultados obtenidos con los diferentes conjuntos de imágees, además de hacer una valoración general de todo el sistema, se procede a presentar una tabla con todos los resultados obtenidos por el Sistema 1 con los diferentes conjuntos y correctamente entrenado. Dichos resultados se presentan en la Tabla 8.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Loss	**0.51**	0.54	0.56	0.60
Training Accuracy	0.77	**0.79**	0.78	0.77
Validation Loss	**0.57**	0.61	0.64	0.60
Validation Accuracy	**0.79**	0.78	0.77	**0.79**
Precisión del sistema	**0.70**	0.67	0.67	0.68
Precisión Granulation	0.63	**0.66**	0.56	0.60
Recall Granulation	0.71	0.71	**0.74**	0.69
F1 Granulation	0.67	**0.68**	0.64	0.64
Precision Healing	0.60	0.70	**0.73**	0.64
Recall Healing	**0.34**	0.21	0.21	0.30
F1 Healing	**0.44**	0.32	0.32	0.41
Precisión Necrosis	0.36	0.31	**0.38**	0.33
Recall Necrosis	0.82	**0.87**	**0.87**	0.85
F1 Necrosis	0.50	0.46	**0.53**	0.47
Precisión Skin	**0.83**	0.79	0.75	0.75
Recall Skin	0.54	**0.89**	0.83	0.85
F1 Skin	**0.84**	**0.84**	0.79	0.80
Precisión Slough	0.44	0.45	0.52	**0.53**
Recall Slough	**0.51**	0.47	0.46	0.30
F1 Slough	0.47	0.46	**0.49**	0.38
Balanced Accuracy Score	**0.64**	0.63	0.62	0.60

Tabla 8: Resultados del Sistema 1.

Como se observa en la Tabla 8, el Sistema 1 no consigue superar el 70 % de acierto con ninguno de los conjuntos de imágenes.

Además, la clase que presenta mejores resultados, y con diferencia, es la clase *Skin*. Esto es debido a que es la clase con mayor número de imágenes para entrenar y cuantas más imágenes en el conjunto de Entrenamiento, más

características propias aprende el sistema de esa clase.

La clase con peores resultados es la clase *Healing*. Esto se explica, como se ha dicho con anterioridad, el tejido en sanación es muy parecido a la piel y esto provoca que al sistema le cueste mucho diferenciarlo de la piel.

La clase *Necrosis* presenta unos resultados muy sorprendentes, porque aunque tenga una *Accuracy* bastante baja, la más baja en todos los conjuntos de imágenes, presenta el *Recall* más elevado en todos los conjuntos. Esto nos indica que el sistema reconoce dicha clase sin ningún problema pero que un fallo en la clasificación, reduce considerablemente la precisión de la clase debido a que solamente se cuentan con 39 imágenes de la clase *Necrosis* en el subconjunto Test.

Con la clase *Necrosis* podemos observar que en determinados casos, no es necesario una gran cantidad de datos en el subconjunto de Entrenamiento para que el sistema aprenda a clasificar de forma adecuada una imagen. Basta con que la clase difiera en gran medida con el resto de clases.

En este caso, la clase *Necrosis* presenta unos colores de tonos negros que los diferencia claramente del resto de clases, las cuales presentan tonos rosáceos o rojizos. Estos dos tonos se diferencian con gran facilidad por lo que puede ser un motivo que explique porqué el sistema clasifica tan bien la clase *Necrosis* aunque haya un número de imñagenes de Entrenamiento bastante pequeño.

Respecto a los conjuntos de imágenes de tamaño 8x8, los resultados obtenidos por el Sistema 1 no son destacables. No se aprecia ningun mejora respecto a los resultados obtenidos mediante el uso de las imágenes de tamaño 16x16. Si es cierto que hay cierta mejoría en algunas clases pero son mejorías insignificantes que no alcanzan en ningún caso un valor superior al 70 %.

Destacar que los mejores resultados los presenta la clase *Skin*, al igual que con las imágenes de tamaño 16x16.

10. DATA AUGMENTATION.

El Data Augmentation es una técnica que busca prevenir el *overfitting* del sistema durante el entrenamiento del mismo. Este procedimiento genera nuevas imágenes utilizando las imágenes existentes, aplicándoles deformaciones predefinidas por el usuario.

Como consecuencia de tener más imágenes en el subconjunto de entrenamiento, el sistema va a poder ser sometido a un proceso de entrenamiento más largo, sin que se produzca el *overfitting*, y por tanto, va a tener más tiempo para extraer características generales de las clases, es decir, va a "aprender" más.

Las deformaciones que se les aplican a las imágenes son muy variadas y son las siguientes: volteos, desplazamientos en cualquier eje, rotaciones y zooms.

Se debe tener en cuenta el papel que juega el píxel central de cada imagen a la hora de clasificar dicha imagen. Según que tejido se encuentre en el píxel central, la imagen pertenecerá a una clase u otra. Por ejemplo, si el píxel central es identificado como tejido piel, dicha imagen pertenece a la clase piel y se le asgina dicha etiqueta.

Por tanto, hay ciertas restricciones a la hora de aplicar las deformaciones del Data Augmentation al conjunto de imágenes. No se pueden aplicar aquellas transformaciones que desplacen o varíen el valor del píxel central de la imagen. Por tanto, las deformaciones que se le van a aplicar a las imágenes solamente serán volteos, rotaciones y zooms.

Para aplicar estas deformaciones se vuelve a recurrir a la clase ***ImageDataGenerator***. Se crea un objeto de esta clase con su constructor, en el cual se especifican todas las deformaciones que se le van a aplicar de forma aleatoria a la imagen que se trate en cada caso. (Ver Figura 27).

En la Figura 27 pueden observarse las transformaciones que se van a aplicar y son las siguientes:

- **Reescalado de la imagen:** La imagen se normaliza de [0,255] a [0,1] para que el entrenamiento sea más eficaz.

```
#PRE-PROCESADO.
from keras.preprocessing.image import ImageDataGenerator

#Empleo la data-augmentation para provocar deformaciones en las imagenes
train_datagen = ImageDataGenerator(
    #Re-escalado de la imagen.
    rescale = 1./255,
    #Ángulo de rotación.
    rotation_range = 25,
    #Intesidad de corte.
    shear_range = 0.5,
    #Zoom que se aplica.
    zoom_range = 0.8,
    #Se permiten flips horizontales.
    horizontal_flip =True)

```

Figura 27: Definicion de las transformaciones de Data Augmentation.

- **Rotaciones:** Se producen rotaciones aleatorias en la imagen con un valor de 25.
- **Cortes:** Se aplicarán cortes de 0.5º en sentido contrario a las agujas del reloj.
- **Zooms:** Se van a plciar zooms de un valor del 0.8.

En la Figura 27 también puede obsevarse como el Data Augmentation solamente se aplica en el conjunto de Entrenamiento, ya que la única transformación que se establece para los subconjuntos Validación y Test es es reescalado de las imágenes a vaores entre 0 y 1.

10.1. Análisis de resultados con Data Augmentation.

A continuación, se muestra una tabla resumen con los resultados del Sistema 1 habiendo aplicado esta técnica a cada uno de los conjuntos de imágenes. Ver Tabla 9.

Si comparamos las Tabla 8 y Tabla 9, se aprecia claramente que se consigue una mejora general en el valor de *Validation Accuracy* pero este aumento no se traduce en una mejoría en la precisión general del Sistema 1 cuando se somete a una prueba con el subconjunto Test. Añadir que la única mejora significativa es el *Recall* de la clase *Necrosis* que ha aumentado aún más.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Loss	0.59	0.54	0.63	0.63
Training Accuracy	0.78	0.79	0.75	0.75
Validation Loss	0.56	0.61	0.62	0.67
Validation Accuracy	0.80	0.80	0.77	0.77
Precisión del sistema	0.70	0.70	0.67	0.67
Precisión Granulation	0.59	0.60	0.54	0.53
Recall Granulation	0.82	0.75	0.81	0.81
F1 Granulation	0.69	0.66	0.64	0.64
Precision Healing	0.64	0.59	0.69	0.70
Recall Healing	0.30	0.41	0.28	0.28
F1 Healing	0.41	0.48	0.40	0.40
Precisión Necrosis	0.31	0.30	0.33	0.29
Recall Necrosis	0.97	0.92	0.87	0.87
F1 Necrosis	0.47	0.45	0.48	0.48
Precisión Skin	0.85	0.87	0.82	0.84
Recall Skin	0.84	0.81	0.79	0.79
F1 Skin	0.84	0.84	0.81	0.81
Precisión Slough	0.52	0.51	0.40	0.40
Recall Slough	0.42	0.53	0.40	0.40
F1 Slough	0.47	0.52	0.40	0.40
Balanced Accuracy Score	0.67	0.68	0.64	0.63

Tabla 9: Resultados del Sistema 1 aplicando Data Augmentation.

También se puede observar con facilidad que la clase *Skin* sigue siendo la mejor clasificada y además, aumenta su precisión. La peor clase clasificada pasa a ser la clase *Slough*. Por último, las clases *Slough, Healing y Granulation* siguen siendo confundidas por el Sistema 1.

11. AUMENTO CAPAS DROPOUT.

Buscando reducir el número de neuronas y el exceso de información que trata la red convolucional, se ha procedido a añadir una capa *Dropout* tras cada capa de *MaxPooling* que existe en la red (Ver Figura 28). Además, se ha eliminado la capa *Dropout* del clasificador. (Ver Figura 29.)

```
###CAPAS###
from keras import layers
from keras import models

model =models.Sequential()
model.add(layers.Conv2D(32,(2,2),padding='same', activation = 'relu', input_shape=(16,16,3)))
model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='zeros'
model.add(layers.MaxPooling2D(pool_size =(2,2)))
model.add(layers.Dropout(0.25))
model.add(layers.Conv2D(64,(2,2),padding='same', activation ='relu'))
model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='zeros'
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.25))
model.add(layers.Conv2D(64,(2,2), padding ='same', activation ='relu'))
model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='zeros'
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.25))
model.add(layers.Conv2D(64,(2,2), padding ='same', activation ='relu'))
model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='zeros'

```

Figura 28: Sistema 1 con capas Dropout.

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation ='relu'))
model.add(layers.Dense(5, activation = 'softmax'))
```

Figura 29: Clasificador para Sistema 1 con capas Dropout.

Este cambio se aplica en el Sistema 1 para las imágenes de 16x16 y las imágenes de 8x8, quedando la arquitectura de la siguiente forma (Ver Figura 30 y Figura 31).

La nomenclatura que se ha utilizado para nombrar los sistemas que se explicarán más adelante es sencilla. Si aparece el sufijo **drop** en el nombre del sistema, se quiere indicar la existencia de una capa *Dropout* tras cada *MaxPooling.* A continuación, se muestra la arquitectura del Sistema 1 Drop para imágenes de tamaño 16x16 (Ver Imagen 30) y para imágenes de 8x8 (Ver Imagen 31) respectivamente.

11.1. Análisis de resultados.

En este apartado se procede a hacer un análisis de los resultados obtenidos tras hacer este cambio en la arquitectura. Para ello, se seguirá el mismo procedimiento que en el apartado anterior, es decir, a continuación se mostrará una tabla resumen con todos los datos obtenidos tras entrenar con los diferentes conjuntos de imágenes y haber sometido al sistema al test de clasificación con el subconjunto Test.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 16, 16, 32)	416
re_lu_1 (ReLU)	(None, 16, 16, 32)	0
batch_normalization_1 (Batch	(None, 16, 16, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 32)	0
dropout_1 (Dropout)	(None, 8, 8, 32)	0
conv2d_2 (Conv2D)	(None, 8, 8, 64)	8256
re_lu_2 (ReLU)	(None, 8, 8, 64)	0
batch_normalization_2 (Batch	(None, 8, 8, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
dropout_2 (Dropout)	(None, 4, 4, 64)	0
conv2d_3 (Conv2D)	(None, 4, 4, 64)	16448
re_lu_3 (ReLU)	(None, 4, 4, 64)	0
batch_normalization_3 (Batch	(None, 4, 4, 64)	256
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 64)	0
dropout_3 (Dropout)	(None, 2, 2, 64)	0
conv2d_4 (Conv2D)	(None, 2, 2, 64)	16448
re_lu_4 (ReLU)	(None, 2, 2, 64)	0
batch_normalization_4 (Batch	(None, 2, 2, 64)	256
flatten_1 (Flatten)	(None, 256)	0
dense_1 (Dense)	(None, 64)	16448
dense_2 (Dense)	(None, 5)	325

Total params: 59,237
Trainable params: 58,789
Non-trainable params: 448

Figura 30: Arquitectura del Sistema 1 Drop para imágenes de tamaño 16x16.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 8, 8, 32)	416
re_lu_1 (ReLU)	(None, 8, 8, 32)	0
batch_normalization_1 (Batch	(None, 8, 8, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 4, 4, 32)	0
dropout_1 (Dropout)	(None, 4, 4, 32)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	8256
re_lu_2 (ReLU)	(None, 4, 4, 64)	0
batch_normalization_2 (Batch	(None, 4, 4, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 2, 2, 64)	0
dropout_2 (Dropout)	(None, 2, 2, 64)	0
conv2d_3 (Conv2D)	(None, 2, 2, 64)	16448
re_lu_3 (ReLU)	(None, 2, 2, 64)	0
batch_normalization_3 (Batch	(None, 2, 2, 64)	256
flatten_1 (Flatten)	(None, 256)	0
dense_1 (Dense)	(None, 64)	16448
dense_2 (Dense)	(None, 5)	325

Total params: 42,533
Trainable params: 42,213
Non-trainable params: 320

Figura 31: Arquitectura del Sistema 1 Drop para imágenes de tamaño 8x8.

En la Tabla 10 se observa claramente como la poda de neuronas tras cada operación de *MaxPooling* afecta negativamente a los parámetros *Recall y F1* de todas las clases, incluso los parámetros que refieren a la clase *Skin*.

Como conclusión de este Sistema 1, si añadimos una capa *Dropout* tras cada operación *MaxPooling*, el Sistema pierde información y confunde aún más cada clase. Además, el Sistema 1 está aún más desbalanceado debido a que la clasificación de la clase *Skin* sigue siendo bastante eficiente pero la clasificación del resto de clases empeora considerablemente.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.71	0.74	0.75	0.76
Training Loss	0.72	0.66	0.63	0.62
Validation Accuracy	0.67	0.69	0.76	0.77
Validation Loss	0.66	0.81	0.64	0.60
Precisión del sistema	0.62	0.63	0.67	0.70
Precisión Granulation	0.80	0.90	0.68	0.68
Recall Granulation	0.31	0.24	0.56	0.60
F1 Granulation	0.44	0.38	0.36	0.64
Precision Healing	0.76	0.70	0.69	0.764
Recall Healing	0.19	0.15	0.32	0.35
F1 Healing	0.30	0.25	0.44	0.45
Precisión Necrosis	0.15	0.42	0.39	0.39
Recall Necrosis	0.05	0.41	0.69	0.69
F1 Necrosis	0.08	0.42	0.50	0.50
Precisión Skin	0.61	0.62	0.71	0.74
Recall Skin	0.99	0.99	0.90	0.90
F1 Skin	0.75	0.76	0.79	0.81
Precisión Slough	0.22	0.56	0.41	0.58
Recall Slough	0.23	0.22	0.24	0.34
F1 Slough	0.01	0.31	0.30	0.43
Balanced Accuracy Score	0.03	0.40	0.54	0.56

Tabla 10: Resultados del Sistema 1 Drop.

11.2. Análisis de resultados aplicando Data Augmentation.

En este subapartado, se procede a realizar un análisis de los resultados obtenidos tras aplicar Data Augmentation al Sistema 1 (Ver Tabla 11).

Tras aplicar la Data Augmentation se aprecia una mejoría con respecto a no aplicar dicha técnica. Los valores de *Training Loss y Validation Loss* han disminuido pero esto no se ha visto reflejado en un aumento de la precisión del Sistema. Se puede destacar que, en líneas generales, los parámetros F1 y Recall han mejorado respecto a no aplicar Data Augmentation, lo cual, no era muy complicado. Aún así, no se consigue mejorar los resultados del Sistema 1 inicial. De hecho, si comparamos los resultados de ambos procesos, se han

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.75	0.74	0.76	0.71
Training Loss	0.64	0.67	0.62	0.67
Validation Accuracy	0.75	0.65	0.46	0.78
Validation Loss	0.69	0.66	0.62	0.61
Precisión del sistema	0.69	0.70	0.69	0.68
Precisión Granulation	0.78	0.79	0.65	0.63
Recall Granulation	0.51	0.58	0.58	0.89
F1 Granulation	0.62	0.67	0.61	0.61
Precision Healing	0.81	0.77	0.66	0.69
Recall Healing	0.21	0.19	0.24	0.30
F1 Healing	0.34	0.30	0.35	0.42
Precisión Necrosis	0.36	0.37	0.37	0.31
Recall Necrosis	0.87	0.85	0.82	0.69
F1 Necrosis	0.51	0.45	0.51	0.43
Precisión Skin	0.71	0.71	0.73	0.74
Recall Skin	0.97	0.97	0.91	0.89
F1 Skin	0.82	0.82	0.81	0.81
Precisión Slough	0.51	0.56	0.58	0.45
Recall Slough	0.26	0.27	0.42	0.36
F1 Slough	0.35	0.36	0.19	0.40
Balanced Accuracy Score	0.57	0.57	0.66	0.56

Tabla 11: Resultados del Sistema 1 Drop Data Augmentation.

empeorado los resultados ya que se ha empeorado en todos los parámetros de evaluación, incluso los referidos a la clase *Skin* y se ha agravado el problema ya existente de confundir las clases *Slough, Healing y Granulation.*

La única conclusión favorable hacia dicho proceso es que, la clase mejor clasificada es la clase *Healing* con un 81 %, superando incluso a la clase *Skin.*

12. RESUMEN DE RESULTADOS.

A continuación, se procede a presentar los resultados obtenidos con los diferentes sistemas que se han derivado del Sistema 1. Se utilizará una tabla

resumen que contendrá los parámetros de evaluación del sistema correctamente entrenado, así como, los parámetros de evaluación de cada clase.

12.1. SISTEMA 2.

La arquitectura que presenta Sistema 2 es la siguiente: (ver Figura 32). Esta arquitectura está constituida por un aumento exponencial de las capas *Conv2D* tras cada capa de *MaxPooling2D.*. Cada vez que aparece una capa *MaxPooling2D*, se dobla el número de capas *Conv2D* y la última de este conjunto es una capa *DepthwiseConv2D*, también conocida como **convolución profunda**.

La principal ventaja de estas capas es que el número de parámetros entrenables del sistema aumenta considerablemte y el sistema tiene más capacidad de aprendizaje pero se requiere más tiempo para procesar cada época del entrenamiento, ya que el sistema necesita ajustar más parámetros.

Los resultados del Sistema 2 son los siguientes (ver Tabla 12):

Como se puede ver en la Tabla 12, el Sistema 2 no ha conseguido mejorar ni igualar la precisión alcanzada por el Sistema 1 en ninguno de los conjuntos. Esto puede ser debido a que al aumentar considerablemente el número de parámetros entrenables, la información se diluya en el sistema.

Además, ninguna de las clases obtiene una mejoría considerable. El sistema 2 confunde aún más las clases ya que los parámetros de *precision, recall y F1* para las clases *Healing, Granulation y Slough* han disminuido.

- Resultados Sistema 2 con Data Augmentation.

Si se observa la Tabla 13, aplicar *Data Augmentation* se refleja en un aumento de la *precision, recall y F1* de todos los parámetros, lo cual era de esperar. Se mejoran los resultados si se compara con el Sistema 1. De hecho, el Sistema 2 aplicando *Data Augmentation* a las imágenes del conjunto 16x16 rand fill alcanza una precisión del 71 %, lo cual mejora la precisión alcanzada por el Sistema 1. Sin embargo, el Sistema 2 está más desbalanceado que el Sistema 1 aunque muy levemente.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 16, 16, 32)	416
re_lu_1 (ReLU)	(None, 16, 16, 32)	0
batch_normalization_1 (Batch	(None, 16, 16, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 32)	0
conv2d_2 (Conv2D)	(None, 8, 8, 64)	8256
depthwise_conv2d_1 (Depthwis	(None, 8, 8, 64)	65600
re_lu_2 (ReLU)	(None, 8, 8, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
batch_normalization_2 (Batch	(None, 4, 4, 64)	256
conv2d_3 (Conv2D)	(None, 4, 4, 64)	16448
conv2d_4 (Conv2D)	(None, 4, 4, 64)	16448
conv2d_5 (Conv2D)	(None, 4, 4, 64)	36928
depthwise_conv2d_2 (Depthwis	(None, 4, 4, 64)	65600
re_lu_3 (ReLU)	(None, 4, 4, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 64)	0
batch_normalization_3 (Batch	(None, 2, 2, 64)	256
conv2d_6 (Conv2D)	(None, 2, 2, 64)	16448
conv2d_7 (Conv2D)	(None, 2, 2, 64)	16448
conv2d_8 (Conv2D)	(None, 2, 2, 64)	36928
conv2d_9 (Conv2D)	(None, 2, 2, 64)	36928
conv2d_10 (Conv2D)	(None, 2, 2, 128)	73856
conv2d_11 (Conv2D)	(None, 2, 2, 128)	147584
conv2d_12 (Conv2D)	(None, 2, 2, 128)	147584
depthwise_conv2d_3 (Depthwis	(None, 2, 2, 128)	131200
re_lu_4 (ReLU)	(None, 2, 2, 128)	0
flatten_1 (Flatten)	(None, 512)	0
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65664
dense_2 (Dense)	(None, 5)	645

Total params: 883,621
Trainable params: 883,301
Non-trainable params: 320

Figura 32: Arquitectura del Sistema 2.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.83	0.88	0.76	0.80
Training Loss	0.46	0.30	0.61	0.50
Validation Accuracy	0.77	0.77	0.78	0.77
Validation Loss	0.62	0.68	0.66	0.74
Precisión del sistema	0.69	0.68	0.66	0.67
Precisión Granulation	0.65	0.62	0.52	0.58
Recall Granulation	0.63	0.67	0.73	0.68
F1 Granulation	0.64	0.65	0.61	0.62
Precision Healing	0.63	0.54	0.52	0.65
Recall Healing	0.20	0.41	0.43	0.41
F1 Healing	0.31	0.47	0.47	0.50
Precisión Necrosis	0.40	0.32	0.31	0.37
Recall Necrosis	0.56	0.87	0.64	0.64
F1 Necrosis	0.47	0.47	0.42	0.47
Precisión Skin	0.74	0.83	0.81	0.76
Recall Skin	0.89	0.83	0.74	0.82
F1 Skin	0.82	0.83	0.77	0.79
Precisión Slough	0.43	0.37	0.46	0.39
Recall Slough	0.46	0.32	0.36	0.28
F1 Slough	0.45	0.34	0.41	0.33
Balanced Accuracy Score	0.55	0.62	0.58	0.57

Tabla 12: Resultados Sistema 2.

Algo que no ha cambiado ha sido que la mejor clase clasificada es la clase *Skin*, la cual alcanza una precisión del 83 %. También encontramos otro valor bastante elevado como la precisión de la clase *Healing* que alcanza un 71 %. Esto puede indicar que el Sistema 2, al tener mayor capacidad de aprendizaje, puede diferenciar algo mejor estas clases.

12.1.1. Resultados Sistema 2 Drop.

La arquitectura del Sistema 2 Drop se puede observar en la Figura 33. En esta aquitectura se puede apreciar como existe una capa *Dropout* tras cada capa de *MaxPooling2D*. Además también se observa como se ha eliminado la capa *Dropout* del clasificador del sistema.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.79	0.77	0.74	0.75
Training Loss	0.53	0.60	0.66	0.64
Validation Accuracy	0.79	0.78	0.75	0.77
Validation Loss	0.55	0.56	0.72	0.64
Precisión del sistema	0.71	0.67	0.64	0.65
Precisión Granulation	0.66	0.58	0.49	0.51
Recall Granulation	0.64	0.71	0.76	0.75
F1 Granulation	0.65	0.64	0.60	0.60
Precision Healing	0.71	0.47	0.59	0.58
Recall Healing	0.30	0.58	0.41	0.31
F1 Healing	0.42	0.52	0.48	0.40
Precisión Necrosis	0.29	0.31	0.26	0.27
Recall Necrosis	0.85	0.92	0.90	0.90
F1 Necrosis	0.43	0.46	0.40	0.41
Precisión Skin	0.83	0.89	0.84	0.79
Recall Skin	0.88	0.73	0.71	0.76
F1 Skin	0.86	0.80	0.77	0.78
Precisión Slough	0.42	0.59	0.47	0.50
Recall Slough	0.60	0.36	0.36	0.50
F1 Slough	0.50	0.45	0.41	0.50
Balanced Accuracy Score	0.65	0.66	0.63	0.61

Tabla 13: Resultados del Sistema 2 Data Augmentation.

Si observamos la Tabla 14 se observa como el sistema alcanza una precisión del 70 % ante ambos conjuntos de 16x16 y el conjunto 8x8 rand fill. También se puede observar como la precisión de todas las clases excepto la clase *Skin* han aumentado considerablemente.

Para la clase *Necrosis*, los parámetros precisión, recall y F1 han disminuido. Esto puede deberse a la eliminación de neuronas tras cada operación de MaxPooling. Eliminar neuronas supone pérdida de información. Esto, junto a que es la clase Necrosis cuenta con menos imágenes de entrenmiento, cualquier pérdida de información se ve rápidamente reflejada en la clasificación de la clase.

- Resultados Sistema 2 drop aplicando Data Augmentation.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.80	0.80	0.76	0.76
Training Loss	0.52	0.52	0.61	0.61
Validation Accuracy	0.78	0.76	0.78	0.78
Validation Loss	0.60	0.67	0.61	0.62
Precisión del sistema	0.71	0.70	0.70	0.69
Precisión Granulation	0.70	0.70	0.64	0.64
Recall Granulation	0.65	0.66	0.60	0.60
F1 Granulation	0.68	0.65	0.62	0.62
Precision Healing	0.73	0.73	0.62	0.75
Recall Healing	0.30	0.21	0.31	0.36
F1 Healing	0.43	0.33	0.42	0.48
Precisión Necrosis	0.30	0.40	0.37	0.38
Recall Necrosis	0.72	0.74	0.69	0.62
F1 Necrosis	0.43	0.52	0.48	0.47
Precisión Skin	0.75	0.74	0.75	0.74
Recall Skin	0.93	0.94	0.89	0.89
F1 Skin	0.83	0.83	0.81	0.81
Precisión Slough	0.54	0.40	0.56	0.51
Recall Slough	0.28	0.46	0.47	0.42
F1 Slough	0.37	0.50	0.51	0.46
Balanced Accuracy Score	0.56	0.59	0.59	0.57

Tabla 14: Resultados del Sistema 2 Drop.

Si se observa la Tabla 15, se observa que para el conjunto 16x16 zero fill, se obtiene la precisión más alta de todos los sitemas que se han testeado, alcanzando un valor en la precisión del 72 %. Como era de esperar, los resultados son algo mejores al aplicar Data Augmentation.

Sin embargo, aun que la precisión general del sistema sea la más elevada, los parámetros de clasificación del sistema no son muy buenos. La clase *Healing y Slough* son fuertemente confundidas con la clase *Skin*.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 16, 16, 32)	416
re_lu_1 (ReLU)	(None, 16, 16, 32)	0
batch_normalization_1 (Batch	(None, 16, 16, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 32)	0
dropout_1 (Dropout)	(None, 8, 8, 32)	0
conv2d_2 (Conv2D)	(None, 8, 8, 64)	8256
depthwise_conv2d_1 (Depthwis	(None, 8, 8, 64)	65600
re_lu_2 (ReLU)	(None, 8, 8, 64)	0
batch_normalization_2 (Batch	(None, 8, 8, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
dropout_2 (Dropout)	(None, 4, 4, 64)	0
conv2d_3 (Conv2D)	(None, 4, 4, 64)	16448
conv2d_4 (Conv2D)	(None, 4, 4, 64)	36928
depthwise_conv2d_2 (Depthwis	(None, 4, 4, 64)	65600
re_lu_3 (ReLU)	(None, 4, 4, 64)	0
batch_normalization_3 (Batch	(None, 4, 4, 64)	256
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 64)	0
dropout_3 (Dropout)	(None, 2, 2, 64)	0
conv2d_5 (Conv2D)	(None, 2, 2, 64)	16448
conv2d_6 (Conv2D)	(None, 2, 2, 64)	16448
conv2d_7 (Conv2D)	(None, 2, 2, 64)	36928
conv2d_8 (Conv2D)	(None, 2, 2, 64)	36928
conv2d_9 (Conv2D)	(None, 2, 2, 128)	73856
conv2d_10 (Conv2D)	(None, 2, 2, 128)	147584
conv2d_11 (Conv2D)	(None, 2, 2, 128)	147584
depthwise_conv2d_3 (Depthwis	(None, 2, 2, 128)	131200
re_lu_4 (ReLU)	(None, 2, 2, 128)	0
batch_normalization_4 (Batch	(None, 2, 2, 128)	512
flatten_1 (Flatten)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65664
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 5)	645

Total params: 884,197
Trainable params: 883,621
Non-trainable params: 576

Figura 33: Arquitectura Sistema 2 Drop.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.71	0.76		0.76
Training Loss	0.73	0.60		0.60
Validation Accuracy	0.73	0.78		0.78
Validation Loss	0.74	0.55		0.63
Precisión del sistema	0.67	0.72		0.68
Precisión Granulation	0.77	0.74		0.67
Recall Granulation	0.43	0.57		0.61
F1 Granulation	0.56	0.64		0.64
Precision Healing	0.82	0.56		0.65
Recall Healing	0.23	0.34		0.34
F1 Healing	0.35	0.42		0.44
Precisión Necrosis	0.40	0.36		0.35
Recall Necrosis	0.6	0.95		0.67
F1 Necrosis	0.48	0.52		0.46
Precisión Skin	0.66	0.79		0.73
Recall Skin	0.98	0.92		0.88
F1 Skin	0.79	0.85		0.80
Precisión Slough	0.49	0.61		0.51
Recall Slough	0.12	0.52		0.31
F1 Slough	0.20	0.56		0.39
Balanced Accuracy Score	0.48	0.66		0.56

Tabla 15: Resultados Sistema 2 Drop Data Augmentation.

12.2. Resultados del Sistema 3.

La arquitectura del Sistema 3 se encuentra reflejada en la Figura 34. La arquitectura del sistema está compuesta por un aumento progresivo de las capas *Conv2D* conforme el tamaño de la imagen se va reduciendo en la red. Esto es así, debido a que cuantas más características extraiga el sistema de la imagen, mayor será su aprendizaje.

También puede verse en la Figura 34, que el clasificador está compuesto dos capas *Dense* de 128 neuronas cada una y para finalizar, una capa *Dense* de 5 neuronas que representa cada una de las clases.

Estos dos cambios suponen un aumento significativo de los parámetros

del sistema, lo cual le da más capacidad de aprendizaje en el proceso de entrenamiento. También puede darse el fenómeno contrario, es decir, el espacio de aprendizaje es demasiado grande y la información puede diluirse en él. Esto se debería a que nuestro conjunto de datos es muy pequeño y desbalanceado, como consecuencia puede que no haya suficiente información para que el sistema aprenda.

Los resultados del Sistema 3 se encuentran en la Tabla 16.

Puede observarse que las precisiones obtenidas por el sistema no son muy elevadas. Solamente el conjunto 8x8 rand fill alcanza el 70 %. Como hemos dicho antes, la información se encuentra muy diluida en el espacio de aprendizaje del sistema 3 debido a las pocas imágenes que forman el conjunto de datos.

Este fenómeno también puede verseen los parámetros de cada clase. La única clase bien clasificada es la clase *Skin*. El resto de clases tienen valores bastante bajos si los comparamos con los valores obtenidos por los otros sistemas. Además, el sistema 3 es el más desbalanceado de todos los sistemas probados.

- Resultados sistema 3 con Data Augmentation.

12.2.1. Resultados del Sistema 3 Drop.

La arquitectura del Sistema Drop puede verse en la Imagen35. Pueden verse las capas *Dropout* tras cada operación de MaxPooling. Además se ha mantenido el mismo clasificador que en el Sistema 3.

Observando la Tabla 18 se ve que la precisión del Sistema 3 Drop para cualquier de los conjuntos no alcanza el 70 %, muy lejos de ese 72 % alcanzado con el Sistema 2. Lo único que se puede destacar de este sistema es que por primera vez, los parámetros de la clase *Skin* no son los mejores cuando se habla de los conjunto de imágenes de tamaño 16x16.

Para dichos conjuntos, la clase *Skin* se clasifica con una precisión del 69 % y 66 %. Sin embargo, en estos conjuntos, la clase mejor clasificada es la clase *Granulation* que alcanza un 80 % en el conjunto de imágenes 16x16 rand fill

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 16, 16, 32)	416
re_lu_1 (ReLU)	(None, 16, 16, 32)	0
batch_normalization_1 (Batch	(None, 16, 16, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 32)	0
conv2d_2 (Conv2D)	(None, 8, 8, 64)	8256
conv2d_3 (Conv2D)	(None, 8, 8, 64)	36928
re_lu_2 (ReLU)	(None, 8, 8, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
batch_normalization_2 (Batch	(None, 4, 4, 64)	256
conv2d_4 (Conv2D)	(None, 4, 4, 64)	16448
conv2d_5 (Conv2D)	(None, 4, 4, 64)	16448
conv2d_6 (Conv2D)	(None, 4, 4, 64)	36928
conv2d_7 (Conv2D)	(None, 4, 4, 64)	36928
re_lu_3 (ReLU)	(None, 4, 4, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 64)	0
batch_normalization_3 (Batch	(None, 2, 2, 64)	256
conv2d_8 (Conv2D)	(None, 2, 2, 64)	16448
conv2d_9 (Conv2D)	(None, 2, 2, 64)	16448
conv2d_10 (Conv2D)	(None, 2, 2, 64)	36928
conv2d_11 (Conv2D)	(None, 2, 2, 128)	73856
conv2d_12 (Conv2D)	(None, 2, 2, 128)	147584
conv2d_13 (Conv2D)	(None, 2, 2, 128)	147584
re_lu_4 (ReLU)	(None, 2, 2, 128)	0
flatten_1 (Flatten)	(None, 512)	0
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65664
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 5)	645

Total params: 674,661
Trainable params: 674,341
Non-trainable params: 320

Figura 34: Arquitectura Sistema 3.

y un 87 % con el conjunto de imágenes 16x16 zero fill. El resto de clases no presentan ninguna novedad respecto a los sistemas anteriormente nombrados.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.76	0.76	0.76	
Training Loss	0.61	0.61	0.62	
Validation Accuracy	0.77	0.76	0.75	
Validation Loss	0.62	0.64	0.70	
Precisión del sistema	0.67	0.54	0.64	
Precisión Granulation	0.59	0.77	0.54	
Recall Granulation	0.72	0.63	0.69	
F1 Granulation	0.65	0.52	0.60	
Precision Healing	0.50	0.45	0.51	
Recall Healing	0.47	0.48	0.42	
F1 Healing	0.49	0.33	0.46	
Precisión Necrosis	0.33	0.85	0.40	
Recall Necrosis	0.67	0.47	0.67	
F1 Necrosis	0.44	0.52	0.50	
Precisión Skin	0.82	0.84	0.78	
Recall Skin	0.77	0.71	0.74	
F1 Skin	0.80	0.77	0.76	
Precisión Slough	0.44	0.32	0.37	
Recall Slough	0.31	0.31	0.35	
F1 Slough	0.37	0.31	0.36	
Balanced Accuracy Score	0.59	0.62	0.57	

Tabla 16: Resultados Sistema 3.

Si nos fijamos en los datos obtenidos por los conjuntos de imágenes de tamño 8x8, todo vuelve a la "normalidad". La clase mejor clasificada es la clase *Skin* mientras que el resto de clases siguen sin estar bien clasificadas.

Destacar que este sistema 3 ha obtenido los valores del parámetro *Balanced Accuracy Score* más bajos de todos los los sistemas estudiados. Esto indica que se clasifica muy bien algunas clases pero que el resto de clases se clasifican muy mal.

- Resultados Sistema 3 Drop Data Augmentation

En la Tabla 19 se observan los resultados obtenidos por el Sistema 3 Drop aplicando Data Augmentation al subconjunto de Entrenamiento. Se ve que

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.77	0.78	0.73	0.73
Training Loss	0.59	0.57	0.69	0.68
Validation Accuracy	0.79	0.76	0.76	0.74
Validation Loss	0.56	0.64	0.67	0.69
Precisión del sistema	0.69	0.69	0.51	0.65
Precisión Granulation	0.56	0.54	0.77	0.54
Recall Granulation	0.80	0.81	0.62	0.72
F1 Granulation	0.66	0.65	0.61	0.62
Precision Healing	0.67	0.63	0.38	0.50
Recall Healing	0.37	0.39	0.47	0.52
F1 Healing	0.48	0.48	0.38	0.51
Precisión Necrosis	0.34	0.41	0.38	0.27
Recall Necrosis	0.97	0.85	0.69	0.87
F1 Necrosis	0.51	0.55	0.49	0.42
Precisión Skin	0.85	0.88	0.78	0.86
Recall Skin	0.79	0.76	0.73	0.70
F1 Skin	0.82	0.81	0.75	0.77
Precisión Slough	0.50	0.46	0.40	0.50
Recall Slough	0.49	0.64	0.40	0.42
F1 Slough	0.49	0.53	0.40	0.45
Balanced Accuracy Score	0.48	0.69	0.59	0.64

Tabla 17: Resultados Sistema 3 Data Augmentation.

las precisiones obtenidas por el sistema son algo más alevadas que al no aplica rData Augmentation. De hecho con los conjuntos de imagenes 16x16 rand fill y 16x16 zero fill se obtienen unas precisiones del 70 % y 71 % respectivamente.

13. OPTIMIZACIÓN DEL SISTEMA.

Antes de proseguir con este Trabajo Fin de Grado, aclarar que, ante los resultados obtenidos por los diferentes sistemas que se han probado, se va a elegir un sistema para realizar la optimización de sus parámetros, así como diferentes pruebas que se van a ir explicando a lo largo que se avanza en este Trabajo Fin de Grado.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 16, 16, 32)	416
re_lu_1 (ReLU)	(None, 16, 16, 32)	0
batch_normalization_1 (Batch	(None, 16, 16, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 32)	0
dropout_1 (Dropout)	(None, 8, 8, 32)	0
conv2d_2 (Conv2D)	(None, 8, 8, 64)	8256
conv2d_3 (Conv2D)	(None, 8, 8, 64)	36928
re_lu_2 (ReLU)	(None, 8, 8, 64)	0
batch_normalization_2 (Batch	(None, 8, 8, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
dropout_2 (Dropout)	(None, 4, 4, 64)	0
conv2d_4 (Conv2D)	(None, 4, 4, 64)	16448
conv2d_5 (Conv2D)	(None, 4, 4, 64)	36928
conv2d_6 (Conv2D)	(None, 4, 4, 64)	36928
conv2d_7 (Conv2D)	(None, 4, 4, 64)	36928
re_lu_3 (ReLU)	(None, 4, 4, 64)	0
batch_normalization_3 (Batch	(None, 4, 4, 64)	256
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 64)	0
dropout_3 (Dropout)	(None, 2, 2, 64)	0
conv2d_8 (Conv2D)	(None, 2, 2, 64)	16448
conv2d_9 (Conv2D)	(None, 2, 2, 64)	16448
conv2d_10 (Conv2D)	(None, 2, 2, 64)	36928
conv2d_11 (Conv2D)	(None, 2, 2, 128)	73856
conv2d_12 (Conv2D)	(None, 2, 2, 128)	147584
conv2d_13 (Conv2D)	(None, 2, 2, 128)	147584
re_lu_4 (ReLU)	(None, 2, 2, 128)	0
batch_normalization_4 (Batch	(None, 2, 2, 128)	512
flatten_1 (Flatten)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65664
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 5)	645

Total params: 695,653
Trainable params: 695,077
Non-trainable params: 576

Figura 35: Arquitectura Sistema 3 Dropout.

El sistema elegido es el Sistema 1 debido a que es el sistema más balanceado y además presenta una precisión bastante elevada para los cinco conjuntos de imágenes que se están tratando.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.76	0.78	0.76	0.75
Training Loss	0.62	0.57	0.61	0.65
Validation Accuracy	0.73	0.72	0.77	0.79
Validation Loss	0.78	0.79	0.67	0.58
Precisión del sistema	0.68	0.67	0.67	0.67
Precisión Granulation	0.80	0.87	0.56	0.61
Recall Granulation	0.44	0.33	0.72	0.59
F1 Granulation	0.57	0.48	0.63	0.60
Precision Healing	0.72	0.68	0.62	0.65
Recall Healing	0.25	0.23	0.45	0.34
F1 Healing	0.37	0.34	0.52	0.45
Precisión Necrosis	0.35	0.47	0.39	0.34
Recall Necrosis	0.82	0.67	0.67	0.67
F1 Necrosis	0.49	0.55	0.50	0.45
Precisión Skin	0.69	0.66	0.75	0.74
Recall Skin	0.98	0.68	0.80	0.85
F1 Skin	0.81	0.79	0.78	0.79
Precisión Slough	0.50	0.67	0.54	0.45
Recall Slough	0.15	0.30	0.28	0.39
F1 Slough	0.23	0.41	0.37	0.42
Balanced Accuracy Score	0.53	0.50	0.58	0.57

Tabla 18: Resultados Sistema 3 Drop.

Los hiperparámetros son variables del sistema que necesitan ser definidas y ajustadas antes de entrenar la red neuronal con el subconjunto de Entrenamiento. Este proceso de optimización se realiza para obtener la mejor actuación del sistema cuando se pruebe con el subconjunto de Test. Los hiperparámetros pueden ser divididos en 2 grupos [16]:

- ***Optimizer Hyperparameters:*** Estos hiperparámetros están relacionados con la optimización y el proceso de entrenamiento del sistema.
- ***Model Specific Hyperparameters:*** También conocidos como los hiperparámetros específicos del sistema.

En el presente Trabajo Fin de Grado solamente se van a optimizar los *Optimizer Hyperparameters*.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.76	0.75	0.73	0.74
Training Loss	0.62	0.66	0.66	0.65
Validation Accuracy	0.76	0.77	0.76	0.77
Validation Loss	0.70	0.64	0.65	0.63
Precisión del sistema	0.71	0.70	0.66	0.67
Precisión Granulation	0.76	0.68	0.55	0.55
Recall Granulation	0.64	0.62	0.70	0.66
F1 Granulation	0.69	0.65	0.62	0.60
Precision Healing	0.82	0.65	0.72	0.71
Recall Healing	0.26	0.25	0.25	0.26
F1 Healing	0.39	0.36	0.37	0.38
Precisión Necrosis	0.44	0.45	0.29	0.33
Recall Necrosis	0.54	0.77	0.85	0.69
F1 Necrosis	0.48	0.57	0.43	0.45
Precisión Skin	0.71	0.73	0.79	0.76
Recall Skin	0.97	0.98	0.83	0.85
F1 Skin	0.82	0.83	0.81	0.80
Precisión Slough	0.52	0.63	0.41	0.44
Recall Slough	0.24	0.29	0.40	0.41
F1 Slough	0.33	0.40	0.41	0.42
Balanced Accuracy Score	0.53	0.58	0.61	0.57

Tabla 19: Resultados Sistema 3 Drop Data Augmentation.

13.1. Optimizer Hyperparameters.

Los *Optimizer Hyperparameters* que se van a optimizar son los siguientes:

- ***Learning rate*:** Es la tasa de aprendizaje del optimizador que se está utilizando. Es la medida en la que se van a modificar los *weigths* de las capas en cada época del entrenamiento. Es uno de los hiperparámetros más importantes a optmizar debido a que si su valor es muy pequeño, a la red le va a llevar cientos, o incluso miles de épocas, alcanzar su estado ideal de aprendizaje. Si este valor es demasiado grande, seguramente el sistema se salte su estado ideal de aprendizaje.
- ***Batch Size*:** Es el número de imágenes que van a entrar en el sistema

desde el generador. No se le pasan al sistema todas las imágenes juntas porque si no el proceso de entrenamiento sería muy costoso. Por tanto, el tamaño del batch tiene un efecto directo en la cantidad de recursos necesarios para el proceso de entrenamiento, su velocidad y el número de iteraciones por época.

Un valor elevado supone un extra para el sistema, el cual utiliza el producto de matrices para el cálculo del entrenamiento pero esto supone también un mayor coste de memoria para la realización de los cálculos.

Un valor más pequeño del *batch size* induce mayor ruido en el cálculo de los errores del sistema pero es muy útil para evitar que el sistema se estanque en un mínimo local durante el proceso de entrenamiento.

- **Número de épocas:** El número de épocas es el número de veces que el sistema va a recorrer el subconjunto de Entrenamiento.

Para la optimización de los hiperparámetros del sistema, se va a usar la librería ***scikit-learn*** [15] . Dicha librería es una poderosa herramienta, ya que hace posible la optimización de cualquier aspecto del sistema. Por ello, también se procede a realizar la optimización de otros parámetros del sistema como son los siguientes:

- ***Dropout rate***: Indica el porcertaje de neuronas que se eliminan en cada capa *Dropout*.
- **Activación de capa**: Indica la activación que tiene cada capa convolucional del sistema.
- **Optimizador**: Optimizador que va a utilizar el sistema para corregir la función *loss* durante el proceso de entrenamiento.

Los algoritmos de optimización de hiperparámetros que se van a utilizar son: ***Grid Search.*** y ***Random Search***.

Se ha decidido usar estos dos algoritmos porque son facilmente implementables con la librería *scikit learn*, además de que es la práctica común en todo proceso de optimización de una red neuronal y por último, pero no menos importante, para poder hacer una comparación entre los resultados obtenidos por un algoritmo de optimización y otro.

13.2. Espacio paramétrico.

El espacio paramétrico está constituido por todas las variables que se van a optimizar, así como todos los posibles valores que pueden tomar. El estado óptimo del sistema se encuentra en todas las posibles combinaciones que se pueden realizar entre todos los valores del espacio paramétrico [15].

Todos los parámetros nombrados en el apartado anterior, van a formar el espacio paramétrico que se va a utilziar para optimizar el sistema.

Los valores de los parámetros que se van a optimizar son los siguientes:

- ***Bathc size:*** 32, 64, 128 y 256.
- **Épocas:** 25, 50, 75 y 100.
- **Activaciones de capa:** Relu, Elu, Selu y Sigmoid.
- ***Dropout rate:*** 0.1, 0.2, 0.3 y 0.4.
- **Optimizadores:** SGD, RMSProp y Adam.
- ***Learning rate:*** 0.001, 0.01 y 0.1.

Un problema que se ha encontrado cuando se ha realizado la optimización de los hiper parámetros del sistema, ha sido que no se pueden optimizar la función de optimización y su *learning rate* al mismo tiempo. Por tanto, se ha decidido realizar una primera optimización en la cual se va a buscar el optimizador más adecuado usando su *learning rate* predeterminado, junto a las épocas, la activación de las capas y el *batch size*. A continuación, se crea una función que devuelve un sistema con los parámetros obtenidos en la primera optimización. Sobre este sistema se realiza una segunda optimización del *learning rate*, las épocas del entrenamiento, el *dropout rate* y el*batch size*.

13.3. Algoritmo optimizador.

El estado óptimo del sistema es una combinación que se encuentra dentro del espacio paramétrico. Sería muy tedioso comprobar a mano todas las

posibles combinaciones de las variables. Para facilitar este proceso, se utiliza lo que se conoce como **algoritmo optimizador.**

El algoritmo optimizador genera sistemas con diferentes combinaciones del espacio paramétrico siguiendo un criterio, hasta encontrar la mejor combinación para el sistema. Los algoritmo que se van a usar son: **Grid Search** y **Randomized Search.**

13.3.1. Grid Search.

El algoritmo de optimización ***Grid Search*** [17] [15] es un algoritmo que prueba todas la combinaciones posibles en el espacio paramétrico que se ha definido y obtiene la mejor combinación de estos para un determinado parámetro de puntuación.

El parámetro de puntuación es más conocido como *score*. En este caso, el *score* sobre el que se va a realizar la optimización es la ***accuracy*** del sistema.

13.3.2. Random Search.

El algoritmo ***Random Search*** es un algoritmo de optimización, que dentro del espacio paramétrico que se ha definido, utiliza combinaciones aleatorias de parámetros para encontrar el estado más óptimo del sistema. [15]

Al igual que con el algoritmo de optimización ***Grid Search***, se va a utilizar como parámetro de puntuación la ***accuracy*** del sistema.

13.4. K FOLD Validación Cruzada.

Como se ha ido comentando a lo largo de este Trabajo Fin de Grado, el conjunto de imágenes ha sido dividido en tres subconjuntos diferentes: Entrenamiento, Validación y Test.

Esta separación se hizo buscando evitar el ***overfitting***. Sin embargo debido a estas tres particiones, ocurre lo siguiente:

- Se reduce drásticamente el número de imágenes disponibles para entrenar al sistema.
- Puede haber una dependencia de los resultados con la forma en la que se ha hecho la separación de los subconjuntos.

Una solución para todas estas consecuencias es realizar una validación cruzada sobre el conjunto de datos de entrada. El procedimiendo que se va a usar en este Trabajo Fin de Grado es conocido como ***K FOLD Cross Validation.***

El *K FOLD Cross Validation* consiste en dividir el subconjunto de Entrenamiento en K partes. Para cada parte ocurre lo siguiente [15] (Ver Imagen 36):

1. El modelo se entrena utilizando K-1 partes como subconjunto de entrenamiento.
2. El modelo resultante es validado con la parte que no se ha utilizado en el entrenamiento.

La medida que evalua la actuación del sistema es la media de los valores computados en el bucle. Esta medida es muy costosa computacionalmente pero no se necesitan demasiados datos para realizarla. Además, no se necesita subconjunto de Validación, por lo que aumentan los datos de entrenamiento del sistema.

El algoritmo *K FOLD Cross Validation Score*, solamente se utiliza para conjuntos de datos que están balanceados, es decir, que tienen el mismo número de imágenes o datos en cada clase. Por tanto, no se puede usar este procedimiento en nuestro problema [15].

Para un conjunto de datos desbalanceado, como es el nuestro, el *K FOLD* que se emplea es conocido como ***Stratified K FOLD.*** Su funcionamiento es idéntico que el del *K FOLD* estándar pero tiene una peculiaridad. En cada partición que realiza, el *Stratified K FOLD* conserva el porcentaje de imágenes que hay en cada partición.

Este aspecto es de vital importancia debido a que, al tener un conjunto de datos desbalanceado, pueden crearse divisiones que se van usar para entrenar

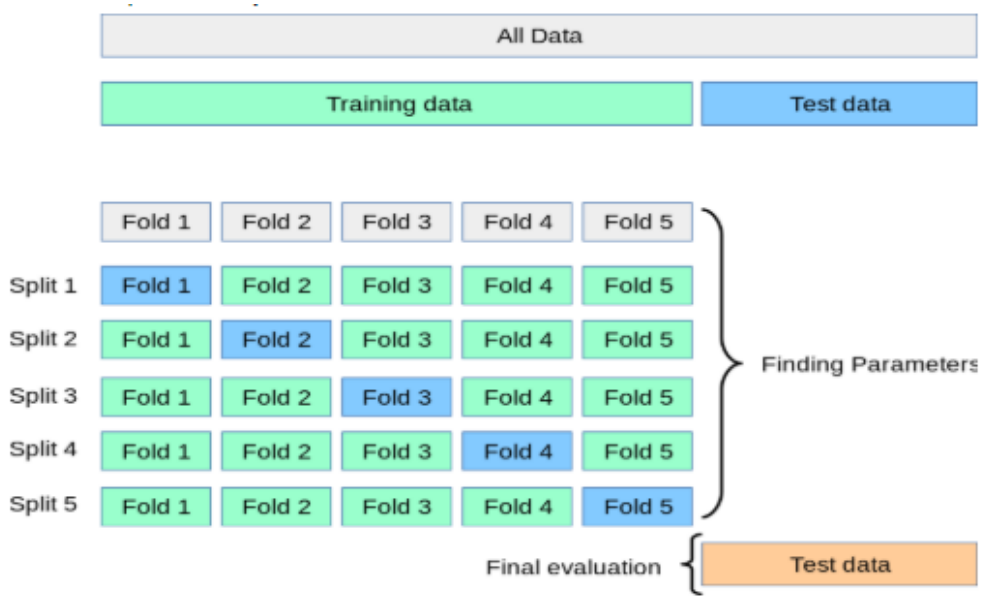

Figura 36: Proceso K Fold Cross Validation.

el sistema, que no contengan imágenes de alguna clase y, en consecuencia, el sistema no aprenda esa clase y se empeoren los resultados en el test del sitema con el subconjunto Test.

13.5. Proceso de Optimización.

La optimización del sistema es un proceso de elevada complejidad y, por tanto, se ha dividido en diferentes pasos, los cuales se van a ir explicando, comentando y justificando a continuación.

13.5.1. Obtención de las imágenes.

Para la obtención de imágenes, se va a proceder al igual que en los apartados anteriores. Además, para aumentar aún más el número de imágenes, se a aplicar la técnica de *Data Augmentation*. (Ver Figura 37)

Se va a usar un generador pero con *batch size* algo mayor que el número de imágenes del subconjunto en cuestión. Esto es asi debido a que, al aplicar

el *Data Augmentation*, el generador utiliza algunas imágenes predefinidas en sus datos, por tanto, para asegurarnos de que el generador utiliza todas las imágenes del subconjunto, usamos un *batch size* más grande.

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255)

train_datagen_da= ImageDataGenerator(
    rescale=1./255,
    rotation_range = 25,
    shear_range = 0.5,
    zoom_range = 0.8,
    horizontal_flip =True)

test_datagen = ImageDataGenerator(rescale=1./255)


#Creo un generador para cada subconjunto con el tamaño del generador anterior.
#Esto hace que obtenga todas las imágenes en un solo batch sin tener que saber cunatas imágenes hay en cada subconjunto.

train_generator_full = train_datagen_da.flow_from_directory(
    train_dir,
    target_size=(16, 16),
    classes = ['granulation','healing','necrosis','skin','slough'],
    batch_size=14300,
    class_mode='categorical')

x_train, y_train = train_generator_full.next()

test_generator_full= test_datagen.flow_from_directory(
    test_dir,
    target_size=(16, 16),
    classes = ['granulation','healing','necrosis','skin','slough'],
    batch_size=1588,
    class_mode='categorical',
    shuffle = False)

x_test, y_test = test_generator_full.next()
```

```
Found 14255 images belonging to 5 classes.
Found 1588 images belonging to 5 classes.
Found 14255 images belonging to 5 classes.
Found 1588 images belonging to 5 classes.
```

Figura 37: Obtención de las imágenes para KFold.

13.5.2. Creación del modelo genérico.

Para la optimización del sistema, se necesita tener el sistema con las variables que se van a optimizar escritas de forma genérica. Esto es así debido a que el algoritmo de optimización genera un sistema diferente para cada posible combinación de las variables, por lo que así es posible que vaya asignando diferentes valores a las variables.

Para ello, se ha creado una función llamada ***create_generic_model()***, la cual recibe como parámetros de entrada los valores de las variables del sistema que se van a optimizar. (Ver Figura 38) y devuelve a su salida el modelo escrito de forma genérica.

```python
from keras import layers
from keras import models
from keras import optimizers


def create_model_generic(activation = 'relu', dropout_rate = 0.1, optimizer = 'Adam'):

    model =models.Sequential()
    model.add(layers.Conv2D(32,(2,2),padding='same', activation = activation, input_shape=(16,16,3)))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D(pool_size =(2,2)))

    #La activacion de la capa se recibe como parámetro en la cabecera de la función
    model.add(layers.Conv2D(64,(2,2),padding='same', activation = activation))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(2,2), padding ='same', activation = activation))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(2,2), padding ='same', activation =activation))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.Flatten())

    #El porcertaje de Dropout se recibe como parámetro en la cabecera de la función
    model.add(layers.Dropout(rate = dropout_rate))
    model.add(layers.Dense(64, activation = activation))
    model.add(layers.Dense(5, activation = 'softmax'))

    #La función de optimización se recibe como parámetro en la cabecera de la función.
    model.compile(loss='categorical_crossentropy', optimizer = optimizer, metrics =['accuracy'])

    return model
```

Figura 38: Función sistema genérico.

13.5.3. Primer Grid Search.

En este paso se realiza la primera optimización del sistema. En ella, se optimiza la activación de cada capa, el optimizador utilizado, el *dropout rate*, así como las épocas del entrenamiento y el tamaño del batch. Todas estas variables crean el espacio paramétrico de esta optimización (Ver Figura 39)

A continuación se aplica el *Grid Search*, utilizando la función ***GridSearchCV***. Recibe como parámetros de entrada el modelo genérico, el espacio parámetrico y en cuantas partes se va a dividir el subconjunto de entrenamiento, es decir, K, utilizando el *Stratified K FOLD*. (Ver Figura 39)

Como paso final, se muestran los resultados. En la primera línea obtenemos la precisión más alta alcanzada por el sistema y cual ha sido la combinación que ha provocado dicho resultado (Ver Figura 40).

```python
#Importo las funciones y librerías necesarias.
import numpy
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

seed = 7
numpy.random.seed(seed)

#Se carga el modelo con la funcinon KerasClassifier. Esta funcion convierte el modelo en un stimator que es lo que necesita
#la funcnion GridSearch.
estimator = KerasClassifier(build_fn=create_model_generic, batch_size=10, epochs = 100, verbose = 0)

#Definimos los parámetros que se van a optimizar aunque solamente optimizo epochs y batch_size
batch_size = [32,64,128,256]
epochs = [25,50,75,100]

activation = ['relu','elu','selu','sigmoid']
optimizer = ['SGD', 'RMSprop','Adam']
param_grid = dict(batch_size = batch_size,epochs=epochs, activation = activation, optimizer = optimizer)

#GridSearch con los parámetros que hemos definido.
#Establezco un Stratified K FOLD en el cross valdiation Score
clf = GridSearchCV(estimator = estimator, param_grid = param_grid, n_jobs=-1, cv = StratifiedKFold(n_splits=3, shuffle=False
y_train_noh = numpy.argmax(y_train, axis = -1)
#Realizo el grid de los parámetros
grid_result = clf.fit(x_train,y_train_noh)

```

```
Fitting 3 folds for each of 192 candidates, totalling 576 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
C:\Users\pablo\Anaconda3\lib\site-packages\joblib\externals\loky\process_executor.py:706: UserWarning: A worker stopped while s
ome jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed: 72.9min
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed: 403.8min
[Parallel(n_jobs=-1)]: Done 426 tasks      | elapsed: 1222.8min
[Parallel(n_jobs=-1)]: Done 576 out of 576 | elapsed: 1655.4min finished
```

Figura 39: Primer Grid Search.

13.5.4. Segundo Grid Search.

Para realizar el segundo *Grid Search*, se construye un sistema con los parámetros obtenidos en la primera optimización. (Ver Figura 41)

En la primera optimización no se pudo realizar una optimización del *learning rate* del optimizador. Por tanto, se hace esta segunda optimización para optmizar el *learning rate*. Además, se aprovecha para volver a optimizar el *batch size* y las épocas del entrenamiento (Ver Figura 42).

Los resultados del segundo Grid Search se encuentran reflejados en la Figura 43.

13.5.5. Test del sistema Grid Search.

En este paso, se debería crear el sistema en cuestión con los parámetros optimizados, entrenarlo y testearlo pero no es necesario. En el algoritmo de

```
print("Mejor: %f usando %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean,stdev, param in zip(means,stds,params):
    print('%f (%f) with: %r' % (mean,stdev,param))
```

```
Mejor: 0.693694 usando {'activation': 'sigmoid', 'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 50, 'optimizer': 'SGD'}
0.627027 (0.056348) with: {'activation': 'relu', 'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 25, 'optimizer': 'SGD'}
0.614414 (0.028716) with: {'activation': 'relu', 'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 25, 'optimizer': 'RMSprop'}
0.621622 (0.033321) with: {'activation': 'relu', 'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 25, 'optimizer': 'Adam'}
0.652252 (0.013483) with: {'activation': 'relu', 'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 50, 'optimizer': 'SGD'}
0.585586 (0.048415) with: {'activation': 'relu', 'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 50, 'optimizer': 'RMSprop'}
0.632432 (0.042102) with: {'activation': 'relu', 'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 50, 'optimizer': 'Adam'}
0.650450 (0.002548) with: {'activation': 'relu', 'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 75, 'optimizer': 'SGD'}
0.569369 (0.153481) with: {'activation': 'relu', 'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 75, 'optimizer': 'RMSprop'}
0.596396 (0.024308) with: {'activation': 'relu', 'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 75, 'optimizer': 'Adam'}
```

Figura 40: Resultados primer Grid Search.

```
from keras import layers
from keras import models
from keras import optimizers
def model_gridsearch(learn_rate=0.001, dropout_rate = 0.1):

    model =models.Sequential()
    model.add(layers.Conv2D(32,(2,2),padding='same', activation = 'sigmoid', input_shape=(16,16,3)))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D(pool_size =(2,2)))
    model.add(layers.Conv2D(64,(2,2),padding='same', activation = 'sigmoid'))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(2,2), padding ='same', activation = 'sigmoid'))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(2,2), padding ='same', activation ='sigmoid'))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.Flatten())
    model.add(layers.Dropout(rate = dropout_rate))
    model.add(layers.Dense(64, activation = 'sigmoid'))
    model.add(layers.Dense(5, activation = 'softmax'))

    optimizer = optimizers.Adam(lr = learn_rate)
    model.compile(loss='categorical_crossentropy', optimizer = optimizer, metrics =['accuracy'])

    return model
```

Figura 41: Sistema tras primer Grid Search.

optimización, se ha establecido el parámetro de entrada **refit = True.** Esto nos permite usar la función ***predict()***. (Ver Figura 44)

Esta función recibe como parámetros de entrada las imágenes del subconjunto Test y ejecuta el sistema con los mejores parámetros, sin tener que desarrollar todo el proceso de entrenamiento, lo cual es largo y tedioso. Además, esta función devuelve un array unidimensional, donde en cada posición i, se alberga la clase que le ha sido asignada a la imagen i.

Tras la optimización del sistema mediante *Grid Search*, se observa en la Figura 43 que el sistema alcanza una precisión del 73 % para el subconjunto de entrenamiento. Mucho menor que el alcanzado por el sistema 1 sin optimizar,

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

estimator = KerasClassifier(build_fn=model_gridsearch, batch_size=10, epochs = 100, verbose = 0)
learn_rate = [0.001, 0.01, 0.1]
batch_size = [32,64,128,256]
epochs = [25,50,75,100]

param_grid = dict(batch_size = batch_size, epochs=epochs, learn_rate = learn_rate)
#GridSearch con los parámetros que hemos definido.
#Establezco un Stratified K FOLD en el cross valdiation Score
clf = GridSearchCV(estimator = estimator, param_grid = param_grid, n_jobs=-1, cv = StratifiedKFold(n_splits=3, shuffle=False
y_train_noh = numpy.argmax(y_train, axis = -1)
#Realizo el grid de los parámetros
grid_result = clf.fit(x_train,y_train_noh)
```

Figura 42: Segundo Grid Search.

```
print("Mejor: %f usando %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean,stdev, param in zip(means,stds,params):
    print('%f (%f) with: %r' % (mean,stdev,param))
```

```
Mejor: 0.735672 usando {'batch_size': 256, 'dropout_rate': 0.2, 'epochs': 50, 'learn_rate': 0.001}
0.717222 (0.019890) with: {'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 25, 'learn_rate': 0.001}
0.560435 (0.102540) with: {'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 25, 'learn_rate': 0.01}
0.537425 (0.000147) with: {'batch_size': 32, 'dropout_rate': 0.1, 'epochs': 25, 'learn_rate': 0.1}
```

Figura 43: Resultados segundo Grid Search.

cuya precisión para el conjunto de entrenamiento fue del 78 % aplicando la técninca de Data Augmentation.

Al hacer una predicción el Sistema 1 optimizado mediante *Grid Search*, los resultados son nefastos. Se obtiene una **accuracy del 67 %** y una **balanced accuracy del 61 %** mientras que en el sistema 1 sin optimizar se habían obtenido una accuracy del 70 % y una balanced accuracy del 68 %.

¿Cómo es posible que los resultados sean peores si el sistema es más óptimo? Esto se debe, a que cuando se han generado los subconjuntos, se ha producido un sesgo en la separacion que favorece la actuación del sistema. Al aplicar el ***Stratified K Fold*** este sesgo se habría eliminado y por tanto, ya no existiría ese plus y lo resultados empeorarían.

En cuanto a la matriz de confusión del sistema 1 optimizado mediante *Grid Search* (Ver Figura 13.5.5), se refleja perfectamente como se siguen clasificando mal ciertas clases. Las clases *Granulation, Healing, Skin y Slough* se confunden gravemente entre ellas.

Esto se refleja en las clases *Granulation*, donde un 28 % de las imágenes se clasifican como *Skin*, un 12 % de las imagenes de la clase *Granulation* se clasifican como imágenes de la clase *Slough*, un 15 % de las imágenes de la

```
y_pred = clf.predict(x_test)


classes = test_generator_full.classes[test_generator_full.index_array]

#Veo la precisión del sistema
print(sum(y_pred==classes)/1588)

#ANALISIS DE RESULTADOS
import sklearn
from sklearn.metrics import confusion_matrix

#Matriz de confusion
print(confusion_matrix(test_generator_full.classes[test_generator_full.index_array],y_pred))
target_names = ['Granulation', 'Healing', 'Necrosis','Skin','Slough']

#Informe de clasificacion. Me devuelve parámetros como F1-score, recall o precision
print(sklearn.metrics.classification_report(test_generator_full.classes[test_generator_full.index_array], y_pred, target_nam

#Balance de precicion de las clases.
print(sklearn.metrics.balanced_accuracy_score(test_generator_full.classes[test_generator_full.index_array], y_pred, sample_w
```

Figura 44: Test Grid Search.

clase *Necrosis* se clasifican como imágenes de la clase *Slough* y un 24 % y un 25 % de las imágenes de la clase *Slough* se clasifican como *Necrosis* y *Skin* respectivamente.

$$\begin{pmatrix} 183 & 27 & 4 & 23 & 34 \\ 39 & 126 & 9 & 84 & 33 \\ 1 & 0 & 27 & 4 & 6 \\ 104 & 72 & 0 & 649 & 27 \\ 1 & 0 & 33 & 35 & 67 \end{pmatrix}$$

Sus parámetros de clasificación por clase son : (Ver Figura 20)

Class	Accuracy	Recall	F1-SCORE	SUPPORT
Granulation	0.56	0.68	0.61	271
Healing	0.56	0.43	0.49	291
Necrosis	0.37	0.71	0.49	38
Skin	0.82	0.76	0.79	852
Slough	0.40	0.49	0.44	136
Micro Avg	0.66	0.66	0.66	1588
Macro avg	0.54	0.61	0.56	1588
Weigthed avg	0.0.68	0.66	0.67	1588

Tabla 20: Resultados del Sistema 1 Grid Search con el conjunto 16x16 rand fill.

13.5.6. Optimización mediante Random Search.

El proceso de optimización del sistema empleando el algoritmo ***Random Search*** es idéntico al proceso seguido para la optimziación mediante *Grid Search.*

Primero se crea un modelo genérico. En este caso se ha utilizado la función *create_model_generic* para crear el modelo escrito en forma general (Ver Figura 38). En segundo lugar, se ha creado el espacio paramétrico, se ha establecido el *Stratified K Fold* y se ha ejecutado el algoritmo de optimización por primera vez. (Ver Figura 45).

```
#Importo las funciones y librerías necesarias.
import numpy
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV
seed = 7
numpy.random.seed(seed)

#Se carga el modelo con la funcinon KerasClassifier. Esta funcion convierte el modelo en un stimator que es lo que necesita
#la funcnion GridSearch.
estimator = KerasClassifier(build_fn=create_model_generic, batch_size=10, epochs = 100, verbose = 0)

#Definimos los parámetros que se van a optimizar aunque solamente optimizo epochs y batch_size
batch_size = [32,64,128,256]
epochs = [25,50,75,100]

activation = ['relu','elu','selu','sigmoid']
optimizer = ['SGD', 'RMSprop','Adam']
param_grid = dict(batch_size = batch_size,epochs=epochs, activation = activation, optimizer = optimizer)

#GridSearch con los parámetros que hemos definido.
#Establezco un Stratified K FOLD en el cross valdiation Score
clf = RandomizedSearchCV(estimator = estimator, param_distributions = param_grid, n_jobs=-1, refit = True, cv = StratifiedKF
y_train_noh = numpy.argmax(y_train, axis = -1)
#Realizo el grid de los parámetros
grid_result = clf.fit(x_train,y_train_noh)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
C:\Users\pablo\Anaconda3\lib\site-packages\joblib\externals\loky\process_executor.py:706: UserWarning: A worker stopped while s
ome jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

Figura 45: Primer Random Search.

Como puede observarse en la Figura 45, el algoritmo RandomizedSearchCV no comprueba todas las posibles combinaciones del espacio hiperparamétrico, si no que solamente comprueba 10 combinaciones aleatorias. Esto se refleja en el recuadro rosa de la Figura 45.

Al igual que en la optimización meidante Grid Search, una vez terminada la primera optimización, se construye un modelo utilizando el valor hallado de los parámetros y con los parámetros que se van a optimizar en la segunda optimización escrito de forma general. Para ello, se utilza la función **randomizedCV_model()**. (Ver Figura 46). Esta función es la función equivalente

a *gridsearch_model()*.

```
from keras import layers
from keras import models
from keras import optimizers
def randomizedCV_model(learn_rate=0.001, dropout_rate = 0.1):

    model =models.Sequential()
    model.add(layers.Conv2D(32,(2,2),padding='same', activation = 'elu', input_shape=(16,16,3)))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D(pool_size =(2,2)))
    model.add(layers.Conv2D(64,(2,2),padding='same', activation = 'elu'))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(2,2), padding ='same', activation = 'elu'))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(2,2), padding ='same', activation ='elu'))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.Flatten())
    model.add(layers.Dropout(rate = dropout_rate))
    model.add(layers.Dense(64, activation = 'elu'))
    model.add(layers.Dense(5, activation = 'softmax'))

    optimizer = optimizers.Adam(lr = learn_rate)
    model.compile(loss='categorical_crossentropy', optimizer = optimizer, metrics =['accuracy'])

    return model
```

Figura 46: Modelo tras primer Random Search.

A continuación, se ha procedido con la segunda optmización mediante el uso del *Random Search*. Para ello, se ha definido el nuevo espacio paramétrico donde se van a buscar los valores óptimos de los nuevos parámetros a optimizar. (Ver Figura 47.)

```
#Creo el modelo con los valores hallados en la primera optimización.
estimator = KerasClassifier(build_fn=randomizedCV_model, verbose = 0)

#Creo el espacio paramétrico
learn_rate = [0.001, 0.01, 0.1]
batch_size = [32,64,128,256]
epochs = [25,50,75,100]
dropout_rate = [0.1,0.2,0.3,0.4]
param_grid = dict(batch_size = batch_size,epochs=epochs, learn_rate = learn_rate, dropout_rate =dropout_rate)


#Ejecuto el randomized Search con Stratified K Fold.
clf = RandomizedSearchCV(estimator = estimator, param_distributions = param_grid, n_jobs=-1, refit = True, cv = StratifiedKF
grid_result = clf.fit(x_train,y_train_noh)

```

Figura 47: Segundo Random Search.

13.5.7. Test del Sistema Random Search.

Para terminar este flujo de trabajo, se utiliza la función **predict()** para realizar las predicciones con el sistema optimizado. Se sigue el mismo procedimiento que en la Figura 44.

```
print("Mejor: %f usando %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean,stdev, param in zip(means,stds,params):
    print('%f (%f) with: %r' % (mean,stdev,param))
```

```
Mejor: 0.728376 usando {'learn_rate': 0.001, 'epochs': 25, 'dropout_rate': 0.4, 'batch_size': 64}
0.537425 (0.000147) with: {'learn_rate': 0.1, 'epochs': 50, 'dropout_rate': 0.1, 'batch_size': 128}
0.537425 (0.000147) with: {'learn_rate': 0.1, 'epochs': 75, 'dropout_rate': 0.4, 'batch_size': 64}
0.538548 (0.189537) with: {'learn_rate': 0.01, 'epochs': 25, 'dropout_rate': 0.1, 'batch_size': 32}
0.709085 (0.016949) with: {'learn_rate': 0.01, 'epochs': 100, 'dropout_rate': 0.2, 'batch_size': 256}
0.537425 (0.000147) with: {'learn_rate': 0.1, 'epochs': 75, 'dropout_rate': 0.4, 'batch_size': 256}
0.567310 (0.099331) with: {'learn_rate': 0.01, 'epochs': 100, 'dropout_rate': 0.2, 'batch_size': 64}
0.573623 (0.112871) with: {'learn_rate': 0.01, 'epochs': 25, 'dropout_rate': 0.4, 'batch_size': 32}
0.537425 (0.000147) with: {'learn_rate': 0.1, 'epochs': 100, 'dropout_rate': 0.3, 'batch_size': 128}
0.728376 (0.020239) with: {'learn_rate': 0.001, 'epochs': 25, 'dropout_rate': 0.4, 'batch_size': 64}
0.537496 (0.000232) with: {'learn_rate': 0.1, 'epochs': 100, 'dropout_rate': 0.3, 'batch_size': 64}
```

Figura 48: Resultado Random Search.

En la Figura 48, se observan los resultados de la optimizacion del sistema mediante *Random Search*. El sistema alcanza una precisión sobre el subconjunto de entrenamiento del 72 %. Como era de esperar, este resultado es más bajo que el del sistema optimizado mediante *Grid Search*. Esto es así debido a que el algoritmo de optimización *Random Search*, no prueba todas las combinaciones posibles, si no que prueba 10 combinaciones aleatorias de parámetros.

Por tanto, a no ser que el algoritmo escoja la combinación encontrada por el *Grid Search*, no hay forma de superar la optmización realizada por el *Grid Search*, como mucho, y es muy poco probable, igualarla.

Atendiendo a los resultados del sistema optimizado mediante Random Search, su matriz de confusión (Ver Figura 13.5.7), se observa claramente que para todas las clases, excepto la clase *Necrosis*, el sistema ha acertado menos clasificaciones que para el sistema optimizado mediante el algoritmo *Grid Search*.

Esto es de esperar ya que este sistema ha presentado una ***accuracy = 63 %*** y una ***balanced accuracy = 60 %*** sobre el conjunto Test, las cuales son menores que el sistema *Grid Search*.

$$\begin{pmatrix} 178 & 35 & 4 & 25 & 29 \\ 35 & 123 & 7 & 81 & 45 \\ 0 & 0 & 30 & 3 & 5 \\ 87 & 87 & 2 & 618 & 58 \\ 4 & 2 & 40 & 29 & 61 \end{pmatrix}$$

21)

Class	Accuracy	Recall	F1-SCORE	SUPPORT
Granulation	0.59	0.66	0.62	271
Healing	0.50	0.42	0.46	291
Necrosis	0.36	0.79	0.50	38
Skin	0.82	0.73	0.77	852
Slough	0.31	0.45	0.37	136
Micro Avg	0.64	0.64	0.64	1588
Macro avg	0.51	0.61	0.54	1588
Weigthed avg	0.66	0.64	0.64	1588

Tabla 21: Resultados del Sistema 1 Grid Search con el conjunto 16x16 rand fill.

En la Tabla 21, se observa claramente como las clases han sido peor clasificadas, incluso la clase *Skin* obtiene peores resultados. Además, la clase *Slough*, con una cantidad de imágenes muy superior a la clase *Necrosis*, tiene una *accuracy* del 31 %.

14. SEPARACIÓN PIEL/NO PIEL.

Durante el desarrollo de este Trabajo Fin de Grado, hay diferentes problemas que persisten en los diferentes sistemas que se han dieñado y evaluado, así como en sus respectivas optimizaciones, los cuales son:

- Gran desbalanceo de las clases.
- Confusión de varias clases como tejido Piel.

Para intentar solucionar estos problema, se va a dividir el problema de la clasificación de imágenes en dos partes, que son las siguientes [2]:

1. Clasificación entre clase Piel y clase No Piel.

2. Clasificación entre clases Healing, Granulation, Slough y Necrosis.

Para la formación de la clase No Piel, se ha procedido de forma muy sencilla. Se ha reestructurado el conjunto de datos, creándose dos carpetas: **binary y multiclass.**

En la carpeta binary, se han creado dos carpetas diferentes. Una carpeta para la clase Piel y otra carpeta para la clase No Piel. Dentro de cada carpeta, se ha divido la clase en tres subconjuntos, los cuales son los habituales, subconjunto de Entrenamiento, Validación y Test.

Para hacer esto, se ha realizado un scrip en Python llamdos ***generate_binary_sets***, el cual crea las carpetas y va copiando las imágenes de una carpeta a otra. El 70 % de las imágenes de cada clase ha ido a la carpeta de Entrenamiento, el 20 % a la carpeta de Validación y el 10 % restante a la carpeta de Test.

14.1. Clasificación entre Piel/No Piel.

Esta es la primera primera parte del problema. Consiste en una **clasificación binaria**, ya que se van a clasificar las imágenes como clase Piel o clase No Piel. Además, es un problema **single labeled** debido a que cada imagen solamente puede ser clasificada como Piel o No Piel, nunca como las dos clases a la vez. La clase No Piel está formada por el conjunto formado por las clases *Slough, Necrosis, Granulation y Healing* mientras que la clase Piel, evidentemente, solamente está fromada por la clase *Skin*.

El objetivo de esta primera clasificación es tener dos clases compuesta por un número parecido de imágenes. Esto va a formar un conjunto de datos más balanceado, lo que va a hacer que el sistema aprenda a clasificar de forma más eficiente las imágenes de la clase Piel del resto de clases.

Para este planteamiento, el sistema 1 ha sido entrenado hasta ser sobre entrenado y posteriormente, se ha reducido ese *overfitting* reduciendo el número de épocas. El sistema 1 ha sido entrenado con cada subconjunto de imágenes disponible.

Al realizar el Test del sistema con el subconjunto Test, se va a realizar

un análisis de los resultados obtenidos y viendo como cambia la clasificación de cada clase respecto a no realizar una separación entre Piel y No Piel. Tras esto, se va a elegir el problema con mejores estadíticas de clasificación y se va a realizar su optimización.

14.1.1. Sistema Binario Piel No Piel.

El sistema que se ha utilizado en este caso, ha sido el **Sistema 1** pero se le han tenido que hacer unas modificaciones bastante simples. Añadir que el sistema es devuelto por una función y está escrito de forma general. Esto es así para facilitar la optimización de algunos de sus hiperparámetros más adelante.

La primera modificación ha sido la activación de la última capa del clasificador que debe ser ***sigmoid***. Esta activación de capa hace que el clasificador devuelva la probabilidad de que la imagen i sea de la clase Piel. (Ver Imagen 49)

El siguiente cambio también se ha realizado en la última capa del clasificador del sistema. El número de neuronas que componen esta capa pasa de 5 a 1. Esto es así debido a que el número de neuronas de la última capa del clasificador representan las clases del problema. En este caso solamente tenemos 1 clase a clasificar y por tanto, el número de neuronas debe ser 1. (Ver Imagen 49)

El último cambio que se ha realizado en el sistema ha sido el parámetro loss, el cual ha sido igualado al valor ***binary crossentropy***, que indica que el tipo de problema al que se enfreta el clasificador es un problema de clasificación binaria. (Ver Imagen 49)

14.1.2. Generadores binarios.

Para procesar las imágenes, al igual que en todos los sistemas anteriores, se usan generadores pero en este caso, son generadores binarios [3].

Para crear generadores binarios simplemente se tiene que indicar el tipo de categoría a la que pertenece el problema. Esto se hace igualando el parámetro

```
from keras import layers
from keras import models
from keras import optimizers

def create_binary_model(activation = 'relu', learning_rate = 0.01, dropout_rate = 0.1):

    model =models.Sequential()
    model.add(layers.Conv2D(32,(2,2),padding='same', activation = activation , input_shape=(16,16,3)))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D(pool_size =(2,2)))
    model.add(layers.Conv2D(64,(2,2),padding='same', activation = activation))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(2,2), padding ='same', activation =activation))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(2,2), padding ='same', activation =activation))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.Flatten())
    model.add(layers.Dropout(rate = dropout_rate))
    model.add(layers.Dense(64, activation = activation))
    model.add(layers.Dense(1, activation = 'sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer = optimizers.Adam(lr = learning_rate), metrics =['acc'])

    return model
```

Figura 49: Sistema para la clasificación binaria.

class mode a binary. También hay que indicar el nombre de las clases que se van a clasificar, es decir, el nombre de las carpetas (*skin, not_skin*) (Ver Figura 50).

Para obtener el mejor resultado posible y aumentar el númerod e imágenes del que se dispone de cada clase, se va a aplicar la técnica de Data Augmentation en el subconjunto de Entrenamiento. (Ver Figura 50).

Como puede observarse en la Figura 50, el generador solamente detecta 2 clases en cada subconjunto pero se mantiene el número de imágenes que forma cada subconjunto.

14.1.3. Análisis de resultados.

A continuación, en la Tabla 22, se muestran los resultados obtenidos por el Sistema 1 tras ser entrenada con cada subconjunto de imágenes.

Como se puede observar en la Tabla 22, los conjuntos con imásgenesde tamaño 8x8 son mucho peor clasificados que los conjuntos con imágenes de tamaño 16x16. Por tanto, estos conjuntos son descartados para la optimización.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.88	0.88	0.83	0.83
Training Loss	0.28	0.28	0.37	0.37
Validation Accuracy	0.87	0.86	0.85	0.84
Validation Loss	0.30	0.31	0.36	0.35
Precisión del sistema	0.84	0.84	0.79	0.77
Precisión Not Skin	0.88	0.81	0.72	0.72
Recall Not Skin	0.75	0.86	0.89	0.84
F1 Not Skin	0.81	0.83	0.80	0.77
Precision Skin	0.81	0.87	0.88	0.83
Recall Skin	0.91	0.82	0.70	0.72
F1 Skin	0.86	0.85	0.78	0.77
Balanced Accuracy Score	0.83	0.84	0.80	0.78

Tabla 22: Resultados Sistema 1 Binario.

Si nos fijamos en los conjuntos formados por imágenes de tamaño 16x16, las estadísiticas de clasificación de la clase Skin mejoran considerablemente respecto a la clasificación de la clase *Skin* sin ser separada del resto de clases. En los conjuntos 16x16 rand fill y 16x16 zero fill se alcanza un 81 % y un 84 % de acierto respectivamente.

Por quedarnos con un conjunto de imágenes para la optimización del sistema, el conjunto **16x16 zero fill** obtiene las ***accuracy*** y ***balanced accuracy*** más elevadas, ambas son nos indican un 85 % de acierto en el sistema.

Se puede afirmar que esta separación entre Piel y No Piel mejora considerablemente los resultados de clasificación de la clase *Skin* del resto de clases.

14.1.4. Optimización Sistema Piel/ No Piel.

La optimizacón del sistema de clasificación binaria consiste en repetir el proceso de optimzación que se realizó anteriormente. Además, se va a emplear el mismo espacio parmetrico.

En este caso, solamente se va a emplear el método de optimización **Grid Search** porque muestra el mejor resultado posible dentro del espacio parmétri-

co definido, ya que prueba todas las combinaciones posibles del espacio paramétrico y por tanto, nos asegura el mejor rendimiento del sistema.

No se va a mostrar todo el proceso de optimización porque es exactamente igual que el que ha seguido en los apartados anteriores. Además, dicho porcesosolamente se va a realizar al Sistema 1 con el conjunto de imágenes que ha mostrado mejores resultados, es decir, el conjunto **16x16 zero fill**.

14.1.5. Análisis de la optimización.

Tras el procesode optimización, el algoritmo *Grid Search* encontró la mejor combinación, la cual está reflejada en la Figura 51

Los resultados de realizar el Test del sistema fueron los siguientes: (Ver Tabla 23)

Class	Accuracy	Recall	F1-SCORE	SUPPORT
Not Skin	0.78	0.84	0.81	740
Skin	0.85	0.80	0.83	853
Accuracy	0.82	0.82	0.82	1593
Macro avg	0.82	0.82	0.82	1593
Weigthed avg	0.82	0.82	0.82	1593

Tabla 23: Resultados Sistema Piel/No Piel optimizado.

Además el sitema ha alcanzado una ***acuraccy y una balanced acuraccy*** del 82 %.

Si se oberva la Tabla 23, se puede ver que los resultados del sistema son algo peores a los obtenidos antes de la optimización. Esto es debido a que hay un sesgo en la separación de las imágenes antes de la optimización del sistema.

En la optimización del sistema, al utilizarse el *Stratified K Fold*, el sesgo de las imágenes se elimina. Aún siendo eliminado el sesgo, se siguen manteniendo unos buenísimos resultados en cuanto a diferenciación del tejido Piel del resto de tejidos.

14.2. Clasificación resto de clases.

La segunda parte del problema consiste en la clasificación de las clases que componen la clase No Piel, es decir, las clases *Slough, Necrosis, Granulation y Healing*. Es el mismo tipo de problema que el problema inicial pero con una clase menos ya que no se va a clasificar la clase Piel.

Al eliminar la clase *Skin*, se elimina una fuente de problemas. Al ser la clase con más imágenes, el sistema extraía una gran cantidad de características de esta clase en comparación con el resto de clases por lo que puede generar que el sistema confunda el resto de clases con la clase *Skin*.

14.2.1. Sistema clasificación No Piel.

Para la clasificación del conjunto No Piel, se ha utilizado el **Sistema 1**, al cual se le han realizado una modificación. Simplemente se ha cambiado el número de neuronas en la última capa del clasificador, el cual ha pasado de 5 neuronas a 4, ya que ahora se van a clasificar 4 clases. (Ver Figura 52)

Este sistema también se ha escrito dentro de una función que recibe algunos parámetros como variables de entrada para facilitar la optimización del sistema más adelante. (Ver Figura 52)

14.2.2. Generadores Multiclase.

Las imagenes han sido procesadas y pasadas al sistema mediante el uso de generadores [3]. Solamente se ha eliminado la clase *Skin* del parámetros *classes*. Así se le indica al generador que esta vez va a manejar 4 clases en vez de 5. En este caso, también se aplica la técnica de *Data Augmentation*. (Ver Figura 53)

Como se puede apreciar en la Figura 53, cada generador detecta 4 clases en su respectivo subconjunto. También puede apreciarse, que al haberse elimnado la clase Skin, se detecta un número menor de imágenes que en la clasificación binaria o en los sistemas anteriores.

14.2.3. Análisis de resultados.

A continuación, en la Tabla 24 se muestran los resultados obtenidos por el Sistema 1 entrenado con los diferentes conjuntos de imágenes, habiéndose eliminado de cada uno de ellos la clase Piel.

	16x16 rand fill	16x16 zero fill	8x8 rand fill	8x8 zero fill
Training Accuracy	0.78	0.86	0.74	0.76
Training Loss	0.57	0.36	0.64	0.61
Validation Accuracy	0.75	0.92	0.80	0.71
Validation Loss	0.63	0.24	0.54	0.72
Precisión del sistema	0.68	0.70	0.71	0.63
Precisión Granulation	0.47	0.50	0.53	0.38
Recall Granulation	0.68	0.57	0.69	0.54
F1 Not Granulation	0.56	0.53	0.60	0.44
Precision Healing	0.91	0.91	0.87	0.96
Recall Healing	0.59	0.66	0.66	0.51
F1 Healing	0.72	0.77	0.75	0.67
Precision Necrosis	0.77	0.79	0.79	0.77
Recall Necrosis	0.73	0.77	0.76	0.77
F1 Necrosis	0.75	0.78	0.77	0.77
Precision Slough	0.38	0.32	0.40	0.30
Recall Slough	0.90	0.87	0.77	0.90
F1 Slough	0.53	0.47	0.53	0.45
Balanced Accuracy Score	0.73	0.72	0.72	0.68

Tabla 24: Resultados Sistema 1 Multiclass.

Atendiendo a los resultados de la Tabla 24, se ve que cuando se realiza una separación de la clase *Skin* del resto de clases, los parámetros obtenidos para la clasificación de estas clases mejoran considerablemente.

El mejor conjunto es el conunto 16x16 zero fill, el cual ha obtenido unos valores de ***accuracy y balanced accuracy*** del 70 % y 72 % respecticamente. Además, para este conjunto, exceto la *accuracy* de la clase *Slough*, no se ha obtenido ningún parámetro con un valor inferior del 50 %.

La clase que más ha mejorado, han sido las clases *Healing y Necrosis*. En la clase *Healing* se han alcanzado valores muy elevados. Para ser exactos unos

valores de *accuracy, recall y F1–Score* del 91 %, 64 % y 77 % respectivamente. Una gran mejoría, comparados con los valores obtenidos sin realizar la separación, los cuales son un 56 %, 40 % y 47 % para los mismo parámetros.

En el caso de la *Accuracy* se ha producido una mejora del 62 %. En el valor de *Recall* se ha producido un incremento del 60 % y en el valor de F1–Score se ha producido un incremento del 48 %.

Para la clase *Necrosis*, se han producido unas mejoras del 139 % y del 50 % en los parámetros *Accuracy y F1–Score*.

Sin embargo, la clase que no ha obtenido una peor clasificación ha sido la clase Granulation.. Pra dicha clase, el Sistema 1 ha obtenido unos valores de 47 %, 68 % y 56 % para los parámetros Accuracy, Recall y F1–Score respectivaente mientras que sin realizar la separación había obtenido unos valores de 61 %, 72 % y 66 % para los mismos parámetros. Esto supone un pequeño empobrecimiento en la clasificación de esta clase.

A pesar de la peor clasificación de la clase Slough, el sistema ha mejorado la clasificación del resto de clases. Esto se debe a que existe una gran diferencia entre el número de imágenes de la clase Skin y el resto de clase, lo cual provoca que el sistema extraiga una gran cantidad de características de esta clase y se empobrezca la clasificación del resto de clases. Esto explicaría la gran mejora que ha sufrido la clasificación del resto de clases

14.3. Optimización Sistema No Piel.

Para la optimización del **Sistema No Piel**, se va a aplicar el mismo procedimiento que en la optimización del Sistema Piel.

Además, se va a reutilizar el espacio paramétrico por lo que se van a optimizar los mismos parámetros que en el Sistema Piel.

14.3.1. Análisis de resultados.

Tras finalizar el proceso de optimización, la mejor combinación de parámetros hallada se encuentra reflejada en la Figura 54.

Los resultados obtenidos por el sistema fueron los siguientes (Ver Tabla 25.

Class	Accuracy	Recall	F1-SCORE	SUPPORT
Granulation	0.54	0.61	0.57	137
Healing	0.78	0.67	0.72	292
Necrosis	0.80	0.60	0.74	272
Slough	0.32	0.82	0.46	39
Accuracy	0.82	0.82	0.68	740
Macro avg	0.61	0.70	0.62	740
Weigthed avg	0.72	0.68	0.69	740

Tabla 25: Resultados Sistema No Piel optimizado.

Además, el sistema optimizado ha obtenido una *acuraccy y balanced acuraccy* del 68 % y 70 %respectivamente.

Estos parámetros son algo peores que el sistema sin optimizar. Este resultado vuelve a ser inferior debido a que se ha eliminado el posible sesgo del sistema sin optimizar.

Aún así, el sistema consigue distinguir perfectamente la clases *Necrosis y Slough*, las cuales clasifica correctamente el 70 % y 68 % de las imágenes. Sigue siendo una mejoría si se compara con las clasificación junto a la la clase *Skin*, ya que, cualquier sistema de los probados no ha conseguido superar el 70 % en ninguna de las dos clases.

15. CONCLUSIONES Y TRABAJO FUTURO.

Tras finalizar todos los experimentos desarrollados a lo largo de este Trabajo Fin de Grado, así como el análisis de los resultados obtenidos en cada uno de ellos, se puede concluir que:

Tras el diseño de los diversos sistemas probados en las secciones de las 6 a la 12, se puede afirmar que la mejor estrategia para la clasificación de las imágenes es utilizar un bloque básico como el que se observa en Figura 10.

Además, solamente es necesaria una capa de *Dropout* al final del sistema ya que las imágenes no albergan demasiada información para el sistema y cada característica extraída es muy valiosa.

Estas secciones también nos permiten afirmar que la técnica de *Data Augmentation*, es de gran ayuda para prolongar el entrenamiento del sistema hasta que se produce el *overfitting* pero que, en este caso, este aumento de tiempo en el entrenamiento no refleja un aumento en la precisión del sistema, pues solamente se ha conseguido mejorar la mejor precisión mostrada en un 1 %.

Tras el análisis de los resultados de estos sistema se observa que, para todos los sistemas existe un problema común. No son capaces de distinguir las clases *Healing, Granulation y Slough*, ya que muchas imágenes de estas clases son clasificadas o como *Skin* o como una de las anteriormente nombradas.

A continuación, se produjo la optimización del sistema utilizando dos métodos de optimización: el *Grid Search* y el *Randomized Search*.

Tras someter el subconjunto Test al sistema optimizado, se observó que la actuación del sistema había empeorado respecto al conjunto sin optimizar. ¿Cómo es posible? Esto es debido a que en la separación de los diferentes subconjuntos, se habría producido un sesgo, el cual favorecería la clasificación del subconjunto Test en el modelo sin optimizar.

Este sesgo se habría eliminado en el proceso de optimización del sistema mediante el uso del algoritmo Stratified K Fold, lo cual explicaría que el sistema optimizado tenga una peor actuación que el sistema sin optimizar.

El siguiente experimento que se ha realizado ha sido la división del problema de clasificación en dos problemas diferentes:

- Clasificación entre Piel y No Piel.
- Clasificación tejidos que forman la clase No Piel: *Slough, Necrosis, Granulation y Healing*.

En el primer subproblema, se ha observado como el sistema distingue perfectamente la clase *Skin* del resto de clases. Los mejores resultados se han obtenido con las imágenes de tamaño 16x16, alcanzándose, en cada subcon-

junto, precisiones cercanas al 90 %. Para los conjuntos de tamaño 8x8 los parámetros de clasificación disminuyen un porcentaje considerable.

Esto nos indica que las imágenes de tamaño 8x8 no contienen suficiente información, ya que, si no se consigue distinguir bien las imágenes de la clase *Skin*, que es la clase con mayor número de imágenes en el subconjunto Entrenamiento, no va a ser posible clasificar adecuadamente el resto de clases que cuenta con un número de imágenes de entrenamiento significativamente menor.

En el segundo subproblema, la clasificación de las clase que conforman la clase No Piel, se ha observado que los parámetros de clasificación obtenidos por el sistema mejoran considerablemente. El conjunto que obtiene los mejores resultados es el conjunto de imágenes 16x16 zero fill.

Tras el análisis de los resultados obtenidos por el sistema tras ser testeado con los conjuntos de imágenes de tamaño 16x16, se ha observado que, el sistema distingue perfectamente las clases *Healing* y *Necrosis* pero que las clases *Granulation* y *Slough* pero que aún no consigue distinguir completamente las imágenes de las clases *Healing* y *Necrosis* ya que un número considerable de ellas es clasificado como una de las clases restantes.

Esta mejora en la clasificación es debido a la eliminación de la clase *Skin*, ya que esto crea un conjunto de datos mucho más balanceado, lo cual hace que el sistema extraiga una proporción más equilibrada de características de cada clase de imágenes.

Para completar el flujo de trabajo, se realizó la optimización de los dos subproblemas en los que se había dividido la clasificación de las imágenes.

En ambos subproblemas, el sistema optimizado ha obtenido peores resultados que en el sistema sin optimizar. Eso confirma el sesgo que se ha producido en la separación de los subconjuntos. Aún así, los sistemas optimizados no han mostrado resultados.

Para el primer subproblema, la clasificación binaria entre imágenes piel y no piel, se ha alcanzado una *balanced accuracy* del 82 %. En el segundo subproblema, la clasificación de imagenes que no corresponden con el conjunto piel, el sistema optmizado ha alcanzado un 70 %, 3 puntos menos que el sistema sin optimizar. Aunque haya disminuido el porcentaje de acierto, esta prueba ha confirmado que la clase *Skin*, al haber una diferencia de imágenes

tan drástica con el resto de clases, condiciona en gran medida el aprendizaje del sistema.

Tras todo este proceso, se puede afirmar que el conjunto de datos con el que se ha trabajado sufre un gran desbalanceo a favor de la clase *Skin*, lo cual provoca que el sistema extraiga una gran cantidad de características de la clase *Skin*. Esto genera una gran confusión en la clasificación del resto de clases, ya que el sistema ha aprendido muy bien que es una imagen *Skin* pero no tiene tan claro el resto de imágenes.

Además, se ha visto que el tamaño de las imágenes que son introducidas a la red convolucional, tiene una gran influencia en los resultados obtenidos por dicha red, ya que los mejores resultados obtenidos, siempre han sido cuando el sistema ha procesado imágenes de tamaño 16x16.

Como conclusión a este Trabajo Fin de Grado, decir que la clasificación de imágenes de úlceras por presión mediante el uso de técnicas de *Deep Learning* es viable, sobre todo si se hace una diferenciación del problema, primero clasificando imágenes de tejido piel y no piel, y seguidamente, realizando una clasificación de las imágenes que componen el conjunto no piel.

No se han alcanzado unos resultados mejores debido al gran desbalanceo que sufre el conjunto de imágenes y, además de las pocas imágenes con las que se cuenta en algunas clases y que resultan insuficientes para que la red convolucional aprenda a reconocer y clasificar correctamente dicho tipo de imagen. El hecho de que nos quedemos con un región pequeña (8x8 ó 16x16) alrededor del pixel central de cada región segmentada ha podido también influir en unos resultados de clasificación no tan elevados.

Como posible mejora a este Trabajo Fin de Grado, se propone utilizar redes convolucionales ya existentes y que hayan sido entrenadas previamente con otro conjunto de imágenes totalmente diferente, como puede ser el conjunto **MNIST–10**.

Referencias

[1] Francisco Javier Veredas Navarro, Héctor Mesa y Laura Morente. "Binary Tissue Classification on Wound Images With Neural Network and Bayesian Classifiers". En: *IEEE Transactions on Medical Imaging* 29 (2010), págs. 41-426.

[2] Francisco Javier Veredas Navarro, Rafael Marcos Luque Baena y col. "Wound image evaluation with machine learning". En: *Elsevier* 164 (2015), págs. 112-122.

[3] François Chollet. *Deep Learning with Python*. Manning Publications Co, 2018. ISBN: 9781617294433.

[4] title = `https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/`, 10 de Agosto de 2019 ¿Cómo funcionan las Convolutional Neural Networks? year = 29 de Noviembre de 2018.

[5] title = `https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23`, 5 de Julio de 2019 Ravindra Parmar year = 2 de Septiembre de 2018.

[6] Softmax Loss Logistic Loss Focal Loss Understanding Categorical Cross-Entropy Loss Binary Cross-Entropy Loss y all those confusing names. *https://gombru.github.io/2018/05/23/cross_entropy_loss/, 15 de Agosto de 2019*. 23 de Mayo de 2018.

[7] Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. *https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, 15 de Agosto de 2019*. 6 de Agosto de 2019.

[8] François Chollet y col. *Keras*. `https://keras.io`. 2015.

[9] Simplified The Softmax Function. *https://towardsdatascience.com/softmax-function-simplified-714068bf8156, 24 de Agosto de 2019*. 26 de Noviembre de 2018.

[10] Neural Net Outputs as Probabilities The Softmax Function y Ensemble Classifiers. *https://towardsdatascience.com/the-softmax-function-neural-net-outputs-as-probabilities-and-ensemble-classifiers-9bd94d75932, 24 de Agosto de 2019*. 13 de Noviembre de 2017.

[11] What are Overfitting and Underfittin in Machine Learning. *https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690, 1 de Agosto de 2019*. 22 de Junio de 2018.

[12] Crear una Red Neuronal en Python desde 0. *https://www.aprendemachinelearning.com/crear-una-red-neuronal-en-python-desde-cero, 15 de Julio de 2018*. 26 de Julio de 2018.

[13] H5 File Extension. *https://fileinfo.com/extension/h5, 24 de Agosto de 2019*. 14 de Junio de 2018.

[14] Metrics to Evaluate your Machine Learning Algorithm. *https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234, 20 de Julio de 2018*. 24 de Febrero de 2018.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot y E. Duchesnay. "Scikit-learn: Machine Learning in Python". En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830.

[16] Towards Data Science. *www.towardsdatascience.com/understanding-hyperparameters-and-its-optimisation-techniques-f0debba07568, 5 de Julio de 2019*. 2018.

[17] Machine Learning Mastery. *https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/, 7 de Julio de 2019*. 2018.

```python
from keras.preprocessing.image import ImageDataGenerator

#Data Augmentation
train_datagen_da= ImageDataGenerator(
    rescale=1./255,
    rotation_range = 25,
    shear_range = 0.5,
    zoom_range = 0.8,
    horizontal_flip =True)

test_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)

#Generador Conjunto Entrenamiento
train_binary_generator = train_datagen_da.flow_from_directory(
    train_binary_dir,
    target_size=(16, 16),
    classes = ['not_skin','skin'],
    batch_size = 64,
    class_mode='binary')

#Generador Conjunto Validación
validation_binary_generator = validation_datagen.flow_from_directory(
    validation_binary_dir,
    target_size=(16, 16),
    classes = ['not_skin','skin'],
    batch_size=64,
    class_mode='binary')

#Generador Conjunto Test
test_binary_generator = test_datagen.flow_from_directory(
    test_binary_dir,
    target_size = (16,16),
    classes = ['not_skin','skin'],
    batch_size = 64,
    class_mode ='binary',
    shuffle = False)

```

```
Found 11089 images belonging to 2 classes.
Found 3156 images belonging to 2 classes.
Found 1593 images belonging to 2 classes.
```

Figura 50: Generadores clasificación binaria.

```
Mejor: 0.844366 usando {'activation': 'selu', 'batch_size': 128, 'dropout_rat
e': 0.2, 'epochs': 100, 'learn_rate': 0.01}
```

Figura 51: Resultado optimización Piel/No Piel.

```python
from keras import layers
from keras import models
from keras import optimizers
def create_multiclass_model(activation = 'relu', dropout_rate = 0.1, learning_rate = 0.01):


    model =models.Sequential()
    model.add(layers.Conv2D(32,(2,2),padding='same', activation = activation, input_shape=(16,16,3)))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D(pool_size =(2,2)))
    model.add(layers.Conv2D(64,(2,2),padding='same', activation = activation))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(2,2), padding ='same', activation = activation))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.MaxPooling2D((2,2)))
    model.add(layers.Conv2D(64,(2,2), padding ='same', activation =activation))
    model.add(layers.ReLU(max_value = None, negative_slope = 0.0, threshold = 0.0))
    model.add(layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='ze
    model.add(layers.Flatten())
    model.add(layers.Dropout(rate = dropout_rate))
    model.add(layers.Dense(64, activation = activation))
    model.add(layers.Dense(4, activation = 'softmax'))

    model.compile(loss='categorical_crossentropy', optimizer = optimizers.Adam(lr= learning_rate), metrics =['acc'])

    return model
```

Figura 52: Sistema clasificación clase No Piel.

```python
from keras.preprocessing.image import ImageDataGenerator

#Data Augmentation
train_datagen_da= ImageDataGenerator(
    rescale=1./255,
    rotation_range = 25,
    shear_range = 0.5,
    zoom_range = 0.8,
    horizontal_flip =True)
test_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)

#Declaro 3 generadores que devuelven las imágenes de cada conjunto en un único batch
train_multiclass_generator = train_datagen_da.flow_from_directory(
    multiclass_train_dir ,
    target_size=(16, 16),
    classes = ['granulation','healing','necrosis','slough'],
    batch_size = 64,
    class_mode='categorical')

validation_multiclass_generator = validation_datagen.flow_from_directory(
    multiclass_validation_dir,
    target_size=(16, 16),
    classes = ['granulation','healing','necrosis','slough'],
    batch_size=64,
    class_mode='categorical')

test_multiclass_generator = test_datagen.flow_from_directory(
    multiclass_test_dir,
    target_size = (16,16),
    classes = ['granulation','healing','necrosis','slough'],
    batch_size = 64,
    class_mode ='categorical',
    shuffle = False)

```

```
Found 5130 images belonging to 4 classes.
Found 1456 images belonging to 4 classes.
Found 740 images belonging to 4 classes.
```

Figura 53: Generadores sistema No Piel.

```
Mejor: 0.732918 usando {'activation': 'selu', 'batch_size': 128, 'dropout_rat
e': 0.2, 'epochs': 150, 'learn_rate': 0.01}
```

Figura 54: Resultado optimización sistema No Piel.