UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Optimización Bayesiana de Hiperparámetros

Realizado por
Kevin Ortiz Falcón

Para la obtención del título de
Grado en Ingeniería Informática

Dirigido por

Tutora: María Dolores Afonso Suárez

Cotutor: Abián Hernández Guedes

Realizado en el departamento de
Ingeniería Informática

2022/2023.

Agradecimientos

Me gustaría agradecer antes que nada a mi tutora y cotutor que me han asesorado perfectamente durante el desarrollo de este proyecto. Además, aprecio de forma especial la implicación que ha tenido mi cotutor Abian Hernández Guedes. Ha sido una pieza fundamental en la resolución de dudas y conflictos durante el transcurso del mismo.

Quiero agradecer también a mi familia, entre ellos a mis padres y a mi hermano por el apoyo que he recibido y por crear un ambiente de trabajo idóneo desde casa.

Resumen

Las redes neuronales artificiales son el modelo computacional más utilizado en el área de inteligencia artificial en la actualidad. Estas redes intentan imitar las redes neuronales biológicas del cerebro humano. En estas redes artificiales existen una serie de configuraciones de parámetros ajustables que permiten controlar el proceso de entrenamiento, estos parámetros se llaman hiperparámetros.

Este proyecto consiste en el estudio de la optimización de estos hiperparámetros. Los hiperparámetros que utilizan las redes neuronales muchas veces tienen valores escogidos de manera aleatoria, o en algunos casos valores seleccionados de acuerdo a experiencias personales del desarrollador. Existen numerosos métodos para optimizar estos hiperparámetros, los criterios de elección entre estos diferentes métodos dependerá bastante de las prestaciones y el tiempo que predisponga el usuario. Por nuestra parte, se plantea que el proceso de optimización se realice mediante optimización Bayesiana, que consiste en crear un modelo probabilístico basado en el teorema de Bayes. Además, para poner a prueba y analizar bien los resultados de nuestra optimización, se entrenan diferentes modelos para solventar problemas de clasificación y de segmentación. Después del proceso de optimización, se realiza una tarea de análisis y comparación de los resultados con los de otros proyectos en los que se ve una mejoría significativa en algunos casos.

En conclusión, la tarea de optimización de los hiperparámetros es totalmente necesaria. Incluso cuando el desarrollador no se puede permitir realizarlo por optimización bayesiana se puede decantar por otros métodos tradicionales que aunque sean menos efectivos son más asequibles.

Palabras clave: Optimización Bayesiana, clasificación, segmentación, análisis, hiperparámetros, redes neuronales, inteligencia artificial.

Abstract

Nowadays, neural networks are the most widely used method in state-of-the-art artificial intelligence. These networks aim to imitate the biological neural networks of the human brain. There are a series of adjustable configurations parameters that allow us to control the training process. These parameters are known as hyperparameters.

This project focuses on optimizing these hyperparameters. The hyperparameters passed to the neural networks are chosen randomly or based on personal experience in some cases. There are several methods available to obtain the best configurations. The choice of these methods depends on factors such as available time, economic resources and material requirements. On our side, we propose the Bayesian optimization as the optimization process, which consists of creating a probabilistic model based on Bayes theorem. In addition, we have developed different models and experiments to solve classification and segmentation problems for testing and analyzing the results of our optimization. Significant improvements have been observed in some cases.

In conclusion, the task of optimizing the hyperparameters is absolutely necessary, even if the user does not have the time or economic and material requirements to perform it by Bayesian optimization, it can be optimized using others traditional methods which, although less effective, are much faster.

Keywords: Bayesian optimization, classification, segmentation, hyperparameters, neural networks, artificial intelligence.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Se puede entender la Inteligencia Artificial (IA) como la capacidad que tienen las máquinas de poder percibir, razonar o aprender, entre otras cosas, de manera automática. En el campo del aprendizaje automático, los modelos y algoritmos de *Machine Learning* (ML) se caracterizan por la presencia de parámetros que deben ser configurados de manera óptima para lograr un rendimiento óptimo en la tarea de interés. Estos parámetros, conocidos como hiperparámetros, no se pueden aprender directamente del conjunto de datos, sino que deben ser ajustados o estimados a través de un proceso específico. En el caso de los algoritmos basados en *Deep Learning* (DL), este es un área de ML que se centra en imitar las redes neuronales biológicas [1], véase la Figura 1.1. Es posible entender que DL es una capa de mayor abstracción que la que se encuentran los algoritmos de ML que se caracterizan por utilizar las redes neuronales artificiales. Estas redes neuronales mencionadas se caracterizan por su estructura compuesta por múltiples capas de nodos, las cuales incluyen una capa de entrada, una o varias capas ocultas y una capa de salida. Cada nodo, también conocido como neurona artificial, establece conexiones con otros nodos y posee un peso y un umbral asociados. Cuando la salida de un nodo individual supera el umbral especificado, dicho nodo se activa y envía datos a la siguiente capa de la red [2].

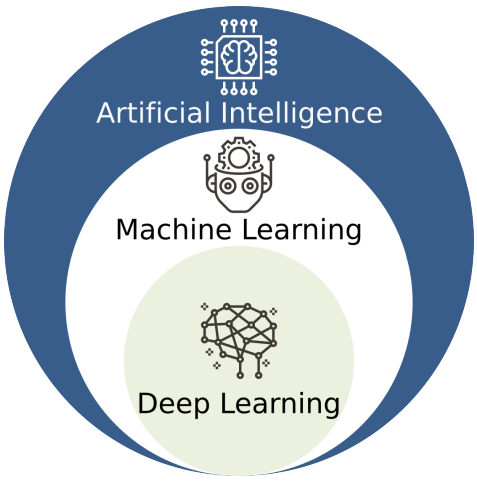


Figura 1.1: Diagrama de Venn que representa la relación entre la inteligencia artificial, machine learning y deep learning.

El primer modelo matemático de una red neuronal artificial surge en 1943, pero incluso antes, en 1842, Ada Lovelace ya hablaba de la posibilidad de que una máquina pudiese actuar sobre otras cosas además de los números [3]. En cambio, la primera red neuronal artificial vino de la mano de Warren McCulloch y Walter Harry Pitts, pero no es hasta 1956 que el primer programa de inteligencia artificial fue lanzado.

Warren McCulloch y Walter Harry Pitts pretendían simular el funcionamiento de una neurona en el cerebro humano. El modelo que ofrecieron era un modelo básico que no se llegó a implementar físicamente debido a las limitaciones técnicas de la época. Este primer intento de imitar el funcionamiento de nuestro cerebro tenía como modelo base el perceptrón de Rosenblatt [4]. En la actualidad, con el progreso de las tecnologías, especialmente el desarrollo de las Graphic Processing Units (GPUs), se ha logrado facilitar la creación de modelos de redes neuronales en un contexto actual. Esto se debe a sus amplias capacidades de procesamiento en paralelo. [5].

En los modelos de redes neuronales artificiales aparece un concepto sumamente importante, se llama hiperparámetro. Estos hiperparámetros son variables que nos ayudan a configurar y controlar el proceso de entrenamiento de nuestro modelo. Los hiperparámetros se configuran de manera manual antes de comenzar el proceso de entrenamiento del modelo. Debido al desconocimiento de las redes neuronales el elegir los valores adecuados para estos hiperparámetros puede ser un problema en algunos casos. No hay un valor fijo para las variables, dependiendo del modelo irá mejor con una configuración u otra.

Con el paso de los años se han usado diferentes técnicas para buscar la configuración más eficiente de los hiperparámetros. Tradicionalmente, se usaban búsquedas en cuadrículas, que consiste en ir probando todos los diferentes valores posibles de hiperparámetros que ha propuesto el usuario mediante fuerza bruta. Este tipo de búsquedas son conocidas por ser ineficientes y costosas tanto desde el punto de vista computacional como del tiempo de entrenamiento. Con esta búsqueda se obtiene la solución óptima de las diferentes configuraciones que ha propuesto el usuario, pero esto no quiere decir que se vaya a encontrar la configuración óptima para el modelo, ya que puede que sea una configuración no contemplada por el usuario. Posteriormente, se fueron desarrollando algoritmos más eficientes que generaban mejores resultados, entre ellos están los algoritmos genéticos y los algoritmos basados en la optimización bayesiana.

Como se ha destacado previamente, los hiperparámetros desempeñan un papel fundamental en el rendimiento y la eficiencia de los modelos de ML y DL. Con el propósito de superar las limitaciones de la búsqueda exhaustiva, este estudio tiene como objetivo lograr la configuración óptima de dichos hiperparámetros mediante el uso de la optimización bayesiana. Para evaluar la eficacia obtenida, se llevarán a cabo experimentos basados en el Trabajo Fin de Grado de referencia [6].

1.1. Estado actual

En la actualidad la IA se encuentra presente en distintas ramas de la ciencia como, por ejemplo, procesamiento de imágenes [7], redes sociales [8], ciencia, medicina [9], usos financieros [10] o incluso en los deportes [11]. La mayoría de los modelos que aparecen en estas ramas de la ciencia se basan en los anteriormente mencionados modelos de DL, como las redes neuronales. El objetivo de dichos modelos es simular el comportamiento del cerebro humano. La finalidad de este objetivo es que las máquinas aprendan de una manera similar a como lo hace nuestro cerebro. Además de esto, se le dota de otras habilidades como la de razonar o tomar de decisiones.

En este proyecto, se busca optimizar un modelo para resolver una tarea específica. Los problemas de optimización buscan minimizar o maximizar el valor de una variable, o métrica, es decir, calcular el valor máximo o mínimo de una función o de una variable objetivo. Los modelos se definen por un conjunto de parámetros que, durante el proceso de entrenamiento, se optimizan para resolver la tarea en cuestión. A lo largo de este trabajo, se realizará una la búsqueda de los valores adecuados de los hiperparámetros empleados en modelos de DL. La estrategia que generalmente se sigue para identificar dichos valores es probando diferentes combinaciones y evaluarlas mediante métodos de validación. En este trabajo se empleará la optimización bayesiana de hiperparámetros como metodología que permita, de forma automática, determinar el parámetro óptimo para cada configuración, evitando así trabajar con prueba y error o estudiando todas las configuraciones posibles.

La estimación de los valores óptimos de estos hiperparámetros es una tarea crucial para maximizar el rendimiento o la eficacia del modelo en una tarea específica. Existen distintos métodos como: *Grid Search*

(búsqueda exhaustiva)[12], *randomized search* [13], algoritmos genéticos[14] u optimización bayesiana[15], entre otros.

Con respecto a las tareas a resolver, en este proyecto se trabajará con dos tipos de problemas: clasificación y segmentación de imágenes. Los problemas de clasificación de imágenes, como su nombre indica, se plantean para clasificar las imágenes según su contenido. Para los problemas de clasificación ya existen técnicas tradicionales de resolución, como puede ser la regresión logística o los árboles de decisión. En este trabajo se hará uso de redes neuronales.

En cuanto a los problemas de segmentación, no tendremos que clasificar imágenes, sino que tenemos que clasificar los diferentes tipos de objetos 1.2 que aparecen en ellas. La distinción primordial entre los problemas de clasificación reside en el enfoque utilizado para la segmentación. En los problemas de segmentación, nos enfrentamos al desafío de clasificar a nivel de píxel, como se menciona en el Trabajo Fin de Grado de referencia [16]. Por otro lado, en los problemas de clasificación, la tarea consiste en clasificar a nivel de imagen en su totalidad.

Figura 1.2: Imagen segmentada

1.2. Objetivos

El objetivo principal de este trabajo es desarrollar un estudio que proponga el uso de métodos Bayesianos para el ajuste de hiperparámetros para descartar la evaluación de las múltiples opciones individualmente. Para ello se plantean los siguientes objetivos:

- **Se utilizarán procesos gaussianos como modelos subrogados del optimizador Bayesiano**. Estos procesos servirán como métrica de validación del modelo cuyos hiperparámetros queremos optimizar. De esta forma, la búsqueda se debe ir redirigiendo en cada iteración a aquellas regiones de mayor interés.

- **Se seleccionarán bases de datos de imágenes médicas, para ilustrar los resultados obtenidos**. Estas bases de datos no necesariamente deben estar diseñadas para uso clínico.

- **Se estudiarán dos casos concretos: segmentación y clasificación de imágenes médicas**. Para estos casos se utilizará el optimizador Bayesiano para obtener unos Hiperparámetros óptimos.

- **Evaluación de los resultados obtenidos**. Se realiza una tarea de análisis con los resultados obtenidos en los diferentes modelos de redes neuronales implementados utilizando las configuraciones de hiperparámetros obtenidos del optimizador bayesiano.

1.3. Competencias cubiertas

- G1. Poseer y comprender conocimientos en un área de estudio (Ingeniería Informática) que parte de la base de la educación secundaria general, y se suele encontrar a un nivel que, si bien se apoya en libros de texto avanzados, incluye también algunos aspectos que implican conocimientos procedentes de la vanguardia de su campo de estudio;

- G2. Aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio;

- G3. Reunir e interpretar datos relevantes (normalmente dentro de su área de estudio) para emitir juicios que incluyan una reflexión sobre temas relevantes de índole social, científica o ética;

- G4. Transmitir información, ideas, problemas y soluciones a un público tanto especializado como no especializado;

- G5. Desarrollar aquellas habilidades de aprendizaje necesarias para emprender estudios posteriores con un alto grado de autonomía.

1.3.1. Competencias ULPGC

- N1. Comunicarse de forma adecuada y respetuosa con diferentes audiencias (clientes, colaboradores, promotores, agentes sociales, etc.), utilizando los soportes y vías de comunicación más apropiados (especialmente las nuevas tecnologías de la información y la comunicación) de modo que pueda llegar a comprender los intereses, necesidades y preocupaciones de las personas y organizaciones, así como expresar claramente el sentido de la misión que tiene encomendada y la forma en que puede contribuir, con sus competencias y conocimientos profesionales, a la satisfacción de esos intereses, necesidades y preocupaciones.
- N2. Cooperar con otras personas y organizaciones en la realización eficaz de funciones y tareas propias de su perfil profesional, desarrollando una actitud reflexiva sobre sus propias competencias y conocimientos profesionales y una actitud comprensiva y empática hacia las competencias y conocimientos de otros profesionales.
- N3. Contribuir a la mejora continua de su profesión, así como de las organizaciones en las que desarrolla sus prácticas a través de la participación activa en procesos de investigación, desarrollo e innovación.

1.3.2. Competencias del título

- T3. Capacidad para diseñar, desarrollar, evaluar y asegurar la accesibilidad, ergonomía, usabilidad y seguridad de los sistemas, servicios y aplicaciones informáticas, así como de la información que gestionan. (G1, G2).
- T5. Capacidad para concebir, desarrollar y mantener sistemas, servicios y aplicaciones informáticas, empleando los métodos de la ingeniería del software como instrumento para el aseguramiento de su calidad, de acuerdo con los conocimientos adquiridos según lo establecido en apartado 5 de la resolución indicada. (G1, G2).
- T6. Capacidad para concebir y desarrollar sistemas o arquitecturas informáticas centralizadas o distribuidas integrando hardware, software y redes, de acuerdo con los conocimientos adquiridos

según lo establecido en apartado 5 de la resolución indicada. (G1, G2).

- T7. Capacidad para conocer, comprender y aplicar la legislación necesaria durante el desarrollo de la profesión de Ingeniero Técnico en Informática y manejar especificaciones, reglamentos y normas de obligado cumplimiento. (N4).
- T8. Conocimiento de las materias básicas y tecnologías, que capaciten para el aprendizaje y desarrollo de nuevos métodos y tecnologías, así como las que les doten de una gran versatilidad para adaptarse a nuevas situaciones. (G3, N3).
- T11. Capacidad para analizar y valorar el impacto social y medioambiental de las soluciones técnicas, comprendiendo la responsabilidad ética y profesional de la actividad del Ingeniero Técnico en Informática. (G5, N2, N4, N5).
- CII01 Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

2. Estado del arte

En este apartado se explican los algoritmos de búsquedas actuales comúnmente utilizados y el porqué utilizarlos, haciendo hincapié en el algoritmo que se va a emplear en nuestra tarea de optimización. A su vez, se describirán las diferentes etapas que se pueden diferenciar en la optimización de hiperparámetros.

2.1. Algoritmos de búsquedas

En la actualidad, es bien conocido que los modelos son cada vez de mayor tamaño, aumentando el número de parámetros que estos emplean. Un ejemplo de esto se encuentra en los modelos basados en GPT. Los modelos GPT se encargan de generar texto de manera autónoma convirtiéndose en una de las herramientas más utilizadas en la actualidad. La primera versión de GPT constaba de un total de 110 millones de parámetros [17], una cifra ya considerablemente alta. En versiones posteriores este número aumenta considerablemente, como puede ser con GPT 2 que tiene 1.5 billones de parámetros y GPT 3 que tiene un total de 175 billones de parámetros [17]. A su vez, el número de hiperparámetros también aumenta en los modelos modernos. Gracias a esto, los algoritmos de optimización cada vez son más necesarios. En general, las redes neuronales cada vez abarcan un mayor tipo de problemas mejorando, con ciertas diferencias, las metodologías tradicionales. A lo largo de este trabajo, se utilizarán modelos relativamente sencillos para evaluar el resultado de los optimizadores bayesianos para el ajuste de los mencionados hiperparámetros.

Los hiperparámetros no son todos los parámetros que puede tener una red neuronal. Los parámetros que nos podemos encontrar se diferencian en tres grandes tipo. Los parámetros de entrada, los parámetros de aprendizaje y los parámetros de configuración o hiperparámetros. Este último es el que vamos a intentar optimizar y son los que nos permiten configurar la estructura de la red.

Desde la perspectiva de un proceso de optimización de hiperparámetros hay que tener en cuenta varios conceptos. Tenemos el espacio de

estados que se corresponde con las diferentes configuraciones de los hiperparámetros [18]. Luego están las acciones, que se corresponden con operaciones utilizadas por el algoritmo de búsqueda para generar los estados. Y finalmente, la función objetivo que permite guiar el proceso de búsqueda para maximizar o minimizar los valores obtenidos para cada estado.

Entre los diferentes métodos de búsqueda, los siguientes son los más utilizados [19, 20]:

- **La búsqueda en cuadrícula:** También conocida como búsqueda exhaustiva. Este algoritmo fue desarrollado en 1974[12] y es uno de los algoritmos más básicos en la optimización de hiperparámetros. En este algoritmo se prueban todas las configuraciones que ha impuesto el usuario mediante fuerza bruta. Se anotan todas las configuraciones en una cuadrícula. Nos moveremos por esta cuadrícula probando todas estas configuraciones y se elegirá la que ofrezca mejores resultados en la función de pérdida. Este algoritmo asegura devolver la configuración óptima de hiperparámetros que ha impuesto el usuario, pero debido al coste computacional y a la responsabilidad recaída en el usuario de poner diferentes configuraciones se ha ido optando por otros algoritmos de optimización. Cabe resaltar que se consigue la configuración de hiperparámetros óptima de las configuraciones propuestas por el usuario, pero esto no quiere decir que esta configuración sea la mejor para nuestro modelo.

- **La búsqueda aleatoria:** Como su nombre indica es una búsqueda aleatoria de hiperparámetros. Se ha desarrollado este algoritmo a partir del algoritmo de la búsqueda en cuadrícula. Se ha demostrado en el trabajo de James Bergstra y Yoshua Bengio [13] que la búsqueda aleatoria es un algoritmo más eficiente que su predecesor, la búsqueda en cuadrícula. Cabe resaltar que es más rentable utilizar la búsqueda aleatoria cuanto mayor es nuestro espacio de búsqueda. Esta búsqueda se realiza en un espacio de búsqueda finito, aunque este algoritmo puede tener ciertas variaciones dependiendo de las personas que lo estén utilizando. Una de ellas consiste en modificar el espacio de búsqueda según resultados previos. En el estudio de la búsqueda aleatoria de hiperpárametros [13] nos demuestra que el algoritmo de búsqueda aleatoria es más eficiente, computacionalmente hablando. Además indica que este algoritmo puede dar mejores resultados que la búsqueda en cuadrícula, aun-

que no siempre sea así. En la Figura 2.1, se muestra una ilustración que contrasta este método con la búsqueda exhaustiva.

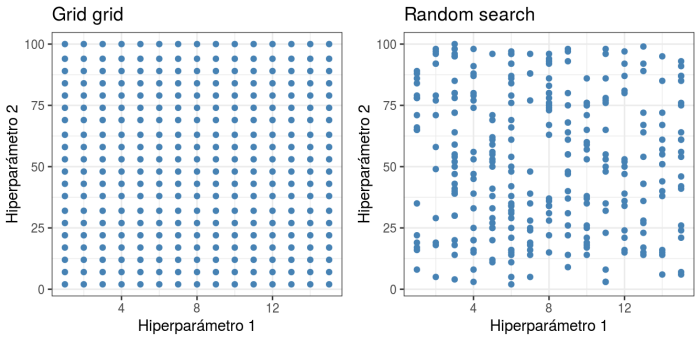


Figura 2.1: Búsqueda en cuadrícula y búsqueda aleatoria

- **La búsqueda mediante algoritmos genéticos**. Los algoritmos genéticos buscan imitar la teoría de la evolución biológica. Para el desarrollo de estos algoritmos se debe contar previamente con un buen conjunto de posibles configuraciones, que serán los individuos de cada generación. Se evalúan estos individuos del conjunto y los más aptos se utilizarán en la reproducción para producir una nueva generación. En cada nueva generación se podrán introducen mutaciones, una cierta probabilidad de alterar o cambiar algunos de estos individuos. Este proceso puede terminar después de una serie de iteraciones o detenerlo cuando no se produzcan más cambios en las últimas generaciones obtenidas, considerando así haber llegado a la solución. [21].

- **La búsqueda mediante optimización bayesiana.** Este algoritmo es el que utilizaremos en nuestro trabajo de optimización de hiperparámetros. El algoritmo deriva del teorema de Bayes [22], que principalmente es un proceso de búsqueda probabilístico donde a cada hiperparámetro se le asigna una puntuación de probabilidad en la función objetivo. Este algoritmo, aunque en algunos casos consuma mucho tiempo computacional, intenta optimizar la evaluación de la función objetivo [23]. Este tipo de búsqueda utiliza retroalimentación de resultados previos para construir la siguiente prueba. Se trata el problema como un problema de regresión y se asume que se tiene una distribución que se actualiza de manera iterativa con diferentes configuraciones de hiperparámetros. Viendo las diferentes configuraciones probadas, el algoritmo tiene que mantener un equilibrio entre seguir explorando las diferentes áreas posibles en las configuraciones o explotar las áreas que está dando

mejores resultados.

Estos algoritmos mencionados son los más famosos actualmente, pero no son los únicos existentes, hay otros que generan buenos resultados como puede ser el de la búsqueda mediante optimización basada en gradientes [24].

2.2. Etapas de la optimización

Independientemente del algoritmo de optimización que se utilice, para la optimización de hiperparámetros siempre encontraremos cuatro fases principales:[20]:

- **Definición de hiperparámetros.** Se define el espacio de valores de los hiperparámetros. Este espacio de búsqueda puede variar según el algoritmo que estemos utilizando y el usuario que esté definiendo este espacio. Si el programador ya tiene experiencia sobre algunos hiperparámetros, podría inducir el valor óptimo de estos. Aparte del criterio del usuario, existen algoritmos que se retroalimentan de iteraciones pasadas, como puede ser el optimizador bayesiano, que provoca cambios en el espacio de búsqueda.
- **Definición del algoritmo de búsqueda.** En esta etapa definimos la configuración de la búsqueda que vamos a realizar. Obviamente, se pueden realizar búsquedas en algunos casos de infinitas iteraciones, por lo que en esta fase aplicaremos un criterio de parada o de duración de la búsqueda de hiperparámetros.
- **Búsqueda.** En esta etapa se realiza la búsqueda de manera automática. Se seleccionan los hiperparámetros que vamos a utilizar en el entrenamiento y se evalúa el modelo generado.
- **Selección de la configuración de hiperparámetros.** En esta última etapa ya tenemos una serie de configuraciones de hiperparámetros candidatas a ser elegidas. Se seleccionará la configuración de manera automática o manual en algunos casos, dependiendo del criterio establecido por el desarrollador.

Estas etapas descritas pueden ser, en algunos casos, automatizadas. Esto se encuentra ilustrado en la Figura 2.2. La primera etapa, que es la que se encarga en definir el espacio de búsqueda, será una tarea manual

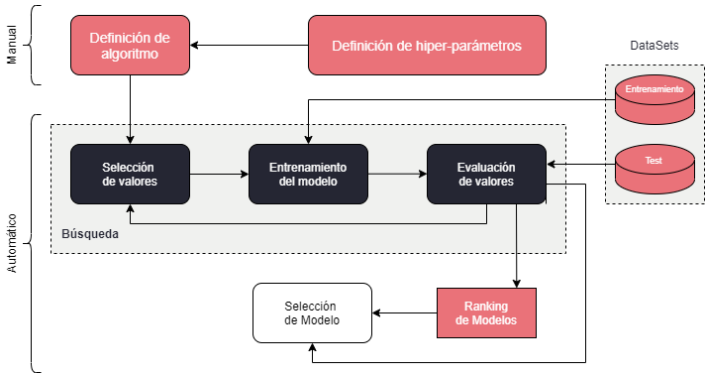


Figura 2.2: Algoritmo de optimización

a criterio del desarrollador, mientras que el resto de etapas pueden llegar a ser automatizadas.

Nosotros en este proyecto utilizaremos la técnica del optimizador bayesiano, con la que veremos reflejadas todas y cada una de las etapas ya mencionadas. Indiferentemente del algoritmo que vayamos a utilizar, es recomendable realizar una tarea de optimización de hiperparámetros para crear modelos más robustos. Dependiendo del algoritmo que vayamos a utilizar esta tarea de optimización puede llevar grandes costes computacionales y de tiempo. Por ejemplo, por parte del optimizador bayesiano lo más importante es el tiempo, al ser un algoritmo que se retroalimenta de resultados anteriores, pues mientras más rondas de optimizaciones se ponga en la configuración en la segunda etapa de definición del algoritmo de búsqueda, mejores resultados se obtendrán.

2.3. Optimización Bayesiana

Con el fin de alcanzar el objetivo establecido en este estudio, se emplea la técnica de optimización bayesiana. En esta sección, se presentará una breve descripción que facilite la comprensión del funcionamiento de dicho enfoque optimizador.

2.3.1. Teorema de Bayes

El teorema de Bayes apareció por primera vez en un artículo en 1763 firmado por Thomas Bayes (tres años después de su muerte) [25]. El teorema de Bayes es uno de los teoremas más utilizados en el área de la

probabilidad en la actualidad. El resultado de este teorema es nombrado como fórmula de Bayes y se describe de la siguiente manera:

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} \tag{2.1}$$

Para entender la fórmula del teorema de Bayes hay que explicar los diferentes factores que aparecen en ella. Son los siguientes:

- **Distribución a priori, $\mathbf{p}(\theta)$.** Esta distribución nos indica qué sabemos de θ antes de mirar la información que nos ofrece y. Dependiendo de lo que sabemos de θ se elige una distribución a priori de forma específica.
- **Verosimilitud, $\mathbf{p}(\mathbf{y}|\theta)$.** Esta función de probabilidad es considerada la más importante en el teorema de Bayes. Representa la probabilidad condicional de los datos observados, dada una configuración específica de los parámetros del modelo.
- **Distribución a posteriori $\mathbf{p}(\theta|\mathbf{y})$.** La distribución a posteriori es el resultado final del proceso de análisis bayesiano de nuestro modelo. Este resultado no es un único valor, sino una distribución de probabilidad para θ.
- **Probabilidad marginal, $\mathbf{p}(\mathbf{y})$.** Es la probabilidad de observar el promedio de los datos conocidos sobre los posibles valores que los parámetros pueden tomar. Este desempeña un papel fundamental en el teorema de Bayes al proporcionar un factor de normalización que permite obtener una probabilidad condicional adecuada y coherente.

2.3.2. Modelos bayesianos

Los modelos bayesianos son modelos estadísticos que se basan en la teoría de la probabilidad bayesiana [26]. De esta forma, estos modelos utilizan el teorema de Bayes para actualizar y estimar la probabilidad de un evento o una hipótesis en función de la evidencia observada. Los modelos bayesianos siguen principalmente tres pasos:

1. Teniendo la información de algunos datos previos y sabiendo cómo se han generado se diseña un modelo basado en distribuciones de probabilidad.

2. Se utiliza el teorema de Bayes para añadir nuevos datos o evidencias para actualizar el modelo.
3. Finalmente, se evalúan los resultados del modelo con base en los resultados obtenidos.

2.3.3. Inferencia bayesiana

La inferencia estadística es el conjunto de técnicas que permiten deducir, a partir de información ya proporcionada, cuál es el comportamiento de una determinada población [27]. En este conjunto, la inferencia bayesiana es una estrategia de inferencia estadística que se basa en el teorema de Bayes. Esta técnica se retroalimenta de las experiencias u observaciones pasadas para actualizar las probabilidades de nuestro modelo. Esta retroalimentación es lo que diferencia el modelo estadístico bayesiano de modelos clásicos de estadística, aparte de otros sucesos externos que pueden tener relación con los datos que se están tratando [28].

Si extrapolamos la definición de inferencia bayesiana a nuestra vida cotidiana, se pueden observar numerosos ejemplos. Por ejemplo, supongamos que a una persona se le ha caído la cartera al suelo. Una cartera que usualmente, por el estereotipo infundado en la sociedad, llevaría un hombre. La persona a la que se le ha caído es una persona con el pelo totalmente rapado. Por experiencias pasadas, una persona con el pelo rapado y con una cartera con un estilo y diseño masculino, lo más probable es que sea de un hombre. Pero, finalmente, nos encontramos que esta persona entra en el aseo de mujeres. Esta última información actualiza nuestra idea de esta persona y nos decantamos definitivamente porque lo más probable es que sea una mujer. Este proceso de conocimiento previo y actualización a medida que obtenemos más información es lo que hace la inferencia bayesiana. No solo podemos encontrar ejemplos en nuestro día a día, numerosas aplicaciones utilizan inferencia bayesiana en sus proyectos.

Un ejemplo del uso de la inferencia bayesiana se encuentra en la plataforma *Google optimize* [29]. Proporciona herramientas que utilizan técnicas de inferencia bayesiana, para llevar a cabo experimentos y optimizar la experiencia del usuario en un sitio web, con el objetivo de mejorar el rendimiento, aumentar la tasa de conversión y lograr los objetivos comerciales.

3. Metodología

Para el desarrollo de este proyecto se han empleado diferentes técnicas y herramientas. En proyectos a largo plazo, a menudo se produce la falta de entrega de un producto acordado, incluso cuando el proyecto es grande. Esto se atribuye generalmente a la confianza excesiva generada al tener suficiente tiempo antes de la fecha límite. Por lo tanto, se requiere una metodología de trabajo que permita medir el rendimiento y evitar la acumulación de trabajo, facilitando la organización y estimación de las tareas durante el período de desarrollo.

En este apartado se detalla la planificación de este proyecto y las herramientas utilizadas para llevarlo a cabo. La organización entre alumno y tutores, y las tecnologías utilizadas.

3.1. Planificación del trabajo

Se ha visto en los últimos años una incorporación inmediata de metodologías ágiles en las empresas de desarrollo. Para mantener desarrollos largos es imprescindible utilizar este tipo de metodologías mejorando así la calidad del producto y la calidad del trabajo que se ha hecho. Gracias a ellas, no solo mejora la calidad de los productos, sino que también mejora la estimación del tiempo y otros recursos que se utilizarán así como los criterios de elección. Aunque el tiempo y el dinero utilizado pueda variar, de esta forma se puede dar una cierta seguridad al cliente sobre estimaciones y costes. Además, algo muy importante a tener en cuenta es que durante el desarrollo de diferentes productos la petición del cliente puede cambiar, una metodología ágil facilitará la adaptación a los cambios que puedan surgir.

Las herramientas más utilizadas en la actualidad para gestionar el uso de una metodología de desarrollo ágil son *Scrum* y *Kanban*. Nosotros utilizaremos el sistema *Kanban* para este proyecto, lo que nos facilitará medir nuestro rendimiento, evitar la acumulación de trabajo y distribuir mejor las tareas planteadas a lo largo del tiempo.

3.1.1. Kanban

Antes de comentar cómo hemos utilizado *Kanban* introduciremos brevemente los principios Lean. La metodología Lean nace en Japón por la empresa Toyota. Con esta filosofía se busca eliminar todos los gastos innecesarios y centrarse en lo que realmente nos han pedido. Para que todo esto fuese posible se postularon una serie de principios. Estos principios varían sus nombres dependiendo de la fuente de información utilizada. Aunque en sus inicios eran 14 principios, se suelen recomendar 5 principios fundamentales para la correcta implementación de la metodología Lean. Son los siguientes [30]:

1. **Identificar el valor desde la perspectiva del cliente.** La empresa debe aprender el valor que le da el cliente a sus servicios y productos. La empresa debe esforzarse en eliminar todo lo que no sea necesario y limitarse a lo que está dispuesto a pagar el cliente.

2. **Mapear el flujo de valores.** Con este principio se entiende registrar los materiales necesarios que se necesitan para elaborar un producto. Todo esto tiene como fin identificar todo lo que no sea necesario. Este proceso abarca cualquier etapa de la vida del producto.

3. **Crear el flujo.** Se debe intentar optimizar el tiempo de entrega de los productos y asegurar que estos procesos desde que se empieza a crear un producto hasta que se entrega sean lo más fluidos posible. Esto significa que se debe identificar futuras posibles interrupciones en el proceso de elaboración del producto.

4. **Establecer un sistema pull.** Esto significa que solo se elabora un producto si este producto tiene demanda en ese mismo instante.

5. **Perseguir la perfección con la mejora continua del proceso.** Se debe mejorar y perfeccionar el proceso de elaboración del producto. Esto hace hincapié en eliminar todo lo que no sea necesario.

Kanban es una metodología que se centra en los principios Lean. De manera que intenta eliminar todo lo que es innecesario en un proyecto. Considerando que las personas implicadas en este proyecto trabajan de forma paralela en otros, se hace necesaria la flexibilidad. Por lo que sería ineficiente plantear una metodología ceñida a fijar *sprints*de duraciones determinadas, ya que es posible que siempre no se pueda llevar a cabo las

diferentes tareas planteadas. Por ello se ha optado por el uso de *Kanban*.

La palabra *Kanban* viene del idioma japones que significa tarjeta, también es entendido como un otorgador de permisos. *Kanban* plantea definir un número máximo de tarjetas para cada estado del proceso. Este número máximo de tarjetas nos ayuda a manejar la carga de trabajo de los trabajadores. Esto nos otorga la oportunidad de anticiparnos a posibles atascos o problemas que nos puedan ir surgiendo durante el trabajo. Se plantea una tabla con diferentes estados, donde se detallan las diferentes tareas a realizar en el proyecto, las tareas que están en proceso y las que están terminadas.

Además del sistema *Kanban* para este proyecto, se han utilizado buenas prácticas de otras metodologías ágiles, como las definidas por el marco de trabajo *Scrum*. Una de ellas es mantener reuniones entre el alumnado y los tutores del proyecto con determinada frecuencia, que podía variar dependiendo de la carga de trabajo de cada uno de ellos. Las de aspectos técnicos se han llevado a cabo cada semana de manera presencial. En las reuniones se debatía sobre lo que se ha hecho durante la semana de trabajo y los problemas que han ido surgiendo, cómo se han solventado estos problemas o, en el caso de que no se haya conseguido solucionar, buscar una solución para los mismos. Además, en estas reuniones se debatía sobre las tareas a desarrollar la siguiente semana, organizándolas en el tablero *Kanban* y moviendo las tareas terminadas a su correspondiente columna[31].

3.2. Herramientas utilizadas

En este apartado detallan las diferentes herramientas que se han utilizado para la elaboración de este trabajo. Tanto las herramientas utilizadas para el desarrollo y control de los informes como de las tecnologías utilizadas para la elaboración del estudio.

3.2.1. Overleaf

Se ha debatido que es igual de importante tener un control en los informes desarrollados y en el informe final como en el código elaborado en el proyecto. Overleaf [32] es una herramienta de edición en línea que nos facilita el proceso de redacción, edición y documentación.

Overleaf, al ser una herramienta online, no necesita actualizaciones. Por lo tanto, no habrá problemas de compatibilidad de versiones. Una de las principales ventajas de esta herramienta es el control de versiones incorporado que tiene, dándonos la posibilidad de restaurar versiones anteriores del proyecto. A su vez, nos facilita diferentes plantillas para trabajos de fin de grado, trabajos de fin de máster o, incluso, tesis doctorales que han sido elaborados por la biblioteca de la Univesidad Carlos III de madrid (UC3M).

3.2.2. Git y Github

En la elaboración de este proyecto hemos necesitado una herramienta de control de versiones. Hemos elegido *GitHub* utilizada durante la carrera, en la actualidad, el controlador de versiones más utilizado.

Hemos utilizado *Github* para alojar nuestro proyecto. Gracias a esta herramienta de control de versiones, los tutores pueden tener conocimiento del estado actual del proyecto y failitará la toma de decisiones sobre el desarrollo de código. Esto también dependerá del manejo que le dé el usuario. Se recomienda hacer *Commit* por cada pequeña funcionalidad que se desarrolle, ya que luego será más fácil retroceder o navegar por las diferentes versiones que se hayan subido.

3.2.3. Google colab

Para la creación de código, que debía estar escrito en Python, se valoraron diferentes herramientas. Nos decantamos en un principio por la utilización de *Google Colab* [33]. *Google Colab* es un producto de *Google research* que nos permite ejecutar código Python desde el navegador. Es un perfecto recurso para el desarrollo de este proyecto, pero debido a la latencia producida en los entrenamientos de las redes neuronales se decidió, durante el transcurso del proyecto, hacerlo en local. Se decidió hacerlo así por dos razones principales, la primera es que el proceso de lectura de las imágenes desde *Google Drive* para el entrenamiento era demasiado largo, y la segunda es que disponíamos de dispositivos más potentes que lo que nos puede proporcionar *Google Colab*.

3.2.4. Anaconda

Este proyecto fue realizado y probado principalmente en local. Se ha utilizado el sistema de paquetería Anaconda [34] para el desarrollo de los experimentos. Anaconda es una distribución de varios lenguajes de programación y utilidades, ampliamente utilizada en el ámbito de la ciencia de datos, que nos ayuda en nuestro trabajo. Cabe resaltar que este programa fue desarrollado por científicos de datos para científicos de datos, por lo que soporta tanto Python como R.

Con Anaconda es posible tener diferentes entornos de trabajo con diferentes versiones del lenguaje de programación con el que estemos trabajando. La principal ventaja que nos ofrece Anaconda es la fácil instalación, actualización y control de paquetes.

3.2.5. AX Framework

AX [35] es un *framework* que sirve para optimizar cualquier experimento. Existen diferentes problemas o experimentos donde se dispone de una serie de configuraciones de variables. Para obtener la configuración óptima es posible que se tenga que invertir mucho tiempo. Este *framework* nos facilita la obtención de los valores de la mejor configuración.

Este *framework* nos ayuda a optimizar cualquier experimento, entre ellos nos puede ayudar a optimizar los hiperparámetros de una red neuronal. Entre sus métodos de optimización está la optimización bayesiana.

AX nos proporciona diferentes métodos para ayudarnos a realizar el optimizador. Sus métodos de entrenamientos y evaluación están principalmente hechos para modelos de clasificación. En nuestro caso de estudio también se realizan experimentos sobre modelos de segmentación, por lo tanto, se han desarrollado por nuestra cuenta los métodos de evaluación y entrenamiento.

Se utiliza el método *optimize* de *AX* que es el que se encarga de ejecutar el optimizador bayesiano. A este método le pasamos los parámetros que queremos optimizar y nuestro método de evaluación y entrenamiento ya creado. También se le pasan cuántas rondas queremos que se ejecuten en el optimizador. Hay que tener en cuenta que al utilizarse el teorema de Bayes es conveniente poner un número elevado ya que necesita re-

troalimentarse de los valores anteriores de la ejecución. Este método nos devuelve los mejores parámetros que se han encontrado para nuestro modelo, además nos devolverá otras variables del modelo de optimización utilizado.

También disponemos de diferentes métodos para generar gráficas suministrados por el mismo *framework*. A este tipo de métodos se le pasa el modelo optimizado que nos devuelve el optimizador para que genere las gráficas con todos los valores que se han utilizado en las diferentes rondas. Se utiliza estas gráficas más adelante para realizar el análisis.

Interfaces de AX

Como se ha documentado, *AX* [35] es un *framework* que sirve para optimizar cualquier experimento y ha sido utilizado para el desarrollo de este trabajo. Se hace uso de este framework para realizar la optimización bayesiana de hiperparámetros. Ax nos proporciona tres modos de uso, a partir de diferentes *Application Programming Interface* (API), que son los siguientes [36]:

- **Loop API.** Este modo de uso es el que usaremos en la optimización de nuestros experimentos. Loop API está pensado para experimentos síncronos. Después de realizar la optimización se dispone de la posibilidad de realizar diferentes gráficas para analizar el proceso de optimización llevado a cabo. Se suele recomendar para los casos más básicos.

- **Service API.** En este modo se suele utilizar para experimentos, pueden evaluarse en paralelo, se recomienda para la optimización de hiperparámetros. Lo bueno de estos dos primeros modos es que no se necesitan muchos conocimientos de la arquitectura que tiene AX internamente.

- **Developer API.** Este modo es el más complejo y el más recomendado para los ingenieros de ciencias de datos. Esta API permite un mayor grado de personalización. Es totalmente recomendable, pero se necesita cierto conocimiento de la arquitectura de AX.

Para el desarrollo de este trabajo se ha empleado la *Loop API*. Se elige este modo debido a la facilidad de uso del mismo, simplemente con la llamada al método *optimize* podemos ejecutar la optimización

bayesiana. En el caso de que se requiera una configuración más específica, se puede elegir otro modo de uso.

4. Clasificación

Los problemas de clasificación de imágenes se centran en clasificar imágenes enteras, al contrario de lo que ocurre en los problemas de segmentación, que clasifican los diferentes elementos de las imágenes. En el problema de clasificación que vamos a abarcar, cada imagen tendrá su propia etiqueta.

En este apartado hablaremos de los problemas de clasificación de imágenes. Comentaremos el *dataset* de imágenes elegido para realizar los experimentos y los diferentes modelos que utilizamos para entrenar nuestra red neuronal, los problemas encontrados y los experimentos que vamos a realizar.

4.1. Dataset

Se ha decidido elegir el dataset DermaMNIST que nos la proporciona MedMNIST [37]. Como el mismo nombre indica, es un dataset centrado en enfermedades de la piel. MedMNIST nos proporciona diferentes datasets de imágenes médicas, no solo de problemas de la piel. En total tiene 12 datasets para imágenes 2D y 6 datasets para imágenes 3D. El dataset seleccionado, DermaMNIST, se compone de imágenes 2D.

Estas imágenes están especialmente seleccionadas para problemas de clasificación. En los datasets de MedMNIST nos podemos encontrar con datasets multi-class, multi-label, binary-class y ordinal-regression. En la figura 4.1 se encuentra un ejemplo de las distintas imágenes médicas que se pueden encontrar. El que hemos elegido, DermaMNIST, es un dataset multi-class, en concreto tiene 7 clases diferentes. Conteniendo un total de 10015 imágenes, de las cuales, 7007 han sido destinadas para entrenar nuestra red neuronal, 1003 muestras se utilizan para la validación y 2005 para test.

Tabla 4.1: *Partición del dataset*

Total	Entrenamiento	Validación	Test
10015	7007	1003	2005

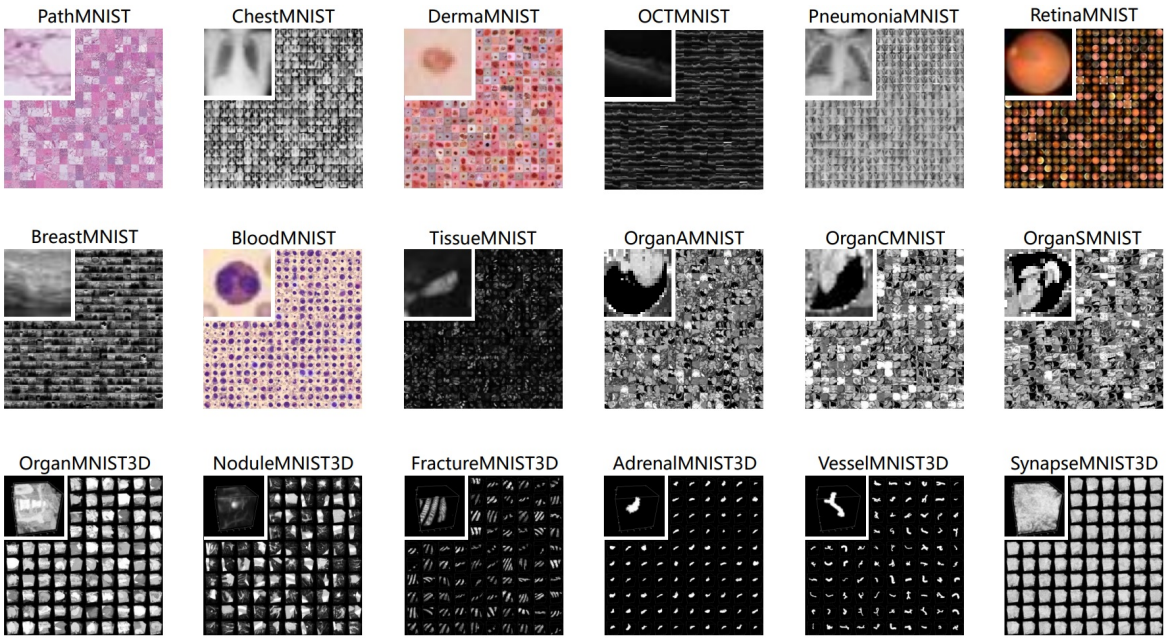


Figura 4.1: Datasets de MedMNIST

En este dataset nos encontramos con 7 clases muy poco balanceadas. Se dice que un dataset de imágenes se encuentran desbalanceado cuando las diferentes clases del dataset no tienen el mismo número de imágenes. En el dataset DermaMNIST la clase con mayor proporción de imágenes es la "melanocytic nevi", que se corresponde con las típicas lesiones cutáneas reconocidas como lunar común. Esta clase tiene un total de 6705 de imágenes. Mientras que la clase con menos imágenes es la "vascular lesions", que se corresponde con un problema relacionado con los vasos sanguíneos. Esta categoría tiene un total de 99 imágenes. Como se puede observar hay un claro desbalance en las clases. Estamos trabajando con un dataset con un total de 10015 imágenes, por lo que, que una sola clase tenga la mayoría de las imágenes del dataset será un problema que hará que nuestra red neuronal no sea fiable. Dicho desbalance en el conjunto de datos provoca que se decante por señalar la clase que tiene la mayoría de las imágenes, "melanocytic nevi". Esto da un buen resultado en la red, pero porque si señala la mayoría de las veces que la imagen es de la clase "melanocytic nevi"hace que acierte la mayoría de las veces. En el siguiente apartado se plantea una serie de diferentes técnicas para arreglar este problema de balanceo.

Tabla 4.2: *Número de imágenes en las clases*

Nombre de las clases	Número de imágenes
actinic keratoses and intraepithelial carcinoma	327
basal cell carcinoma	514
benign keratosis-like lesions	1099
dermatofibroma	115
melanoma	1113
melanocytic nevi	6705
vascular lesions	142

4.1.1. Solución al desbalance de imágenes

Como se describió en la sección anterior, el dataset sufre un claro desbalanceo orientado a la clase "melanocytic nevi". En esta sección se describen tareas específicas para lidiar con ese problema. En los siguientes apartados se describen las técnicas utilizadas.

Data Augmentation

En los principios de las redes convolucionales el principal problema para realizar los entrenamientos era la ausencia de imágenes. Indiferentemente del tipo de problema que abarcaba el ejercicio que se estuviese haciendo, había pocas imágenes. Debido a este problema se desarrolló una serie de soluciones para crear nuevas imágenes a partir de las ya existentes. Estas soluciones planteadas también se pueden extrapolar para solucionar el problema del balance que se tiene en muchos casos con los datasets.

Con *Data Augmentation* lo que intentamos hacer es balancear nuestro dataset alterando las mismas imágenes del mismo dataset y así aumentar el tamaño de las clases perjudicadas. Las diferentes técnicas que se usa con Data Augmentation son fáciles de implementar porque las diferentes librerías que se usan en inteligencia artificial nos proporcionan diferentes métodos que nos ayudan a llevarla a cabo. En este caso, se utilizan distintas transformaciones afines aleatorias como el reescalado de las imágenes, rotación y traslación.

Muestreo ponderado

Los modelos basados en el muestreo son una técnica comúnmente utilizada en estadísticas y aprendizaje automático para obtener una muestra representativa de un conjunto de datos más grande. La idea detrás del muestreo es que, al seleccionar una muestra aleatoria de los datos disponibles, se puede inferir información sobre el conjunto de datos completo sin tener que examinar todos los elementos. Esta es otra de las técnicas más utilizadas para solucionar los problemas de balance de imágenes. Con esta técnica lo que se busca es que en los batches, el subconjunto del muestreo, de imágenes haya un número proporcional de imágenes de cada clase.

Para el uso de esta técnica utilizamos un método que nos proporciona Pytorch, llamado WeightedRandomSampler [38]. Se debe calcular previamente el peso de las diferentes clases que tenemos. Esto se hace dividiendo el número total de imágenes que por cada número de imágenes que tiene cada clase. Luego se le pasa una lista con los pesos de las imágenes al método WeightedRandomSampler. El resultado de este método será el muestreador que utilizaremos en el cargador de imágenes de entrenamiento.

Focal Loss

En la actualidad, existen funciones objetivo que permiten lidiar con los problemas de desbalanceo de los datos. En este caso, se propone el uso del *Focal Loss* [39] el cual es ampliamente utilizado para lidiar con los problemas inherentes del desbalance del conjunto de datos.

El método actual que está predominando en la detección de objetos es el conocido Two-stage Detectors [40]. En esta primera etapa normalmente se utiliza el llamado Region Proposal Network (RPN) [41], pero existen otras técnicas como puede ser selective search [42]. Se utilizan estos métodos para saber qué áreas son de interés y así reducir el coste computacional. La segunda etapa consiste en clasificar las principales áreas propuestas. Antes del uso de este método, se usaba el método One-stage detectors. En este caso no se hace una búsqueda previa para conocer las principales áreas de interés. Se hace directamente la clasificación de objetos.

La principal ventaja que tiene utilizar los One-stage detectors es

la velocidad con la que se realiza, pero logra menos accuracy que el método Two-stage detectors producido por el incorrecto balance que hay entre las clases. De esto nace la función de pérdida Focal loss [39], de querer obtener una exactitud similar o superior con el método One-stage detectors.

La función de entropía cruzada, del inglés *Cross-Entropy* (CE), se utiliza ampliamente en problemas de clasificación. Esta función penaliza las salidas que no están correctamente clasificadas. Una variante de esta función consiste en agregar una constante para enfatizar una clase en particular, lo que ayuda a abordar problemas de desequilibrio de clases. La fórmula para el cálculo de la CE, de un conjunto de probabilidades estimadas, P, se puede expresar como:

$$CE(P) = -\frac{1}{N}\sum_{\hat{p}\in P}\alpha_c \mathbb{1}(\hat{p} = c)\log \hat{p} \tag{4.1}$$

donde α_c es un factor de ponderación para la clase c, $\mathbb{1}(.)$ es una función identificadora que extrae el subconjunto correspondiente a la clase c [43], $\hat{p}$ representa la estimación de probabilidad de las distintas clases contenidas en P, y N es el número total de muestras.

Focal Loss [39] es una función de pérdida que añade un nuevo factor a la función de pérdida CE. Con este nuevo factor que se añade se intenta que nuestra función de pérdida se centre en el aprendizaje de los ejemplos difíciles y, también, en reducir la ponderación de los numerosos negativos.

$$FL(P) = -\frac{1}{N}\sum_{\hat{p}\in P}\alpha_c \mathbb{1}(\hat{p} = c)(1-\hat{p})^{\gamma}\log \hat{p} \tag{4.2}$$

Aunque la implementación inicial de Focal Loss fuese sin el factor α_c, se ha decidido añadirlo, ya que se ha comprobado que mejora los resultados obtenidos por nuestro modelo.

4.2. Modelo CNN

Las redes convolucionales han tenido éxito en el reconocimiento de imágenes y vídeo, entre otras cosas. Las *Convolutional Neural Network* (CNN), a diferencia de las tradicionales *Fully-Connected Networks*

(FCN), permiten capturar la información espacial en las imágenes. Las redes CNN se usan principalmente para la clasificación y categorización de imágenes u objetos, aparte de generar mejores resultados en diferentes tareas computacionales [44].

La estructura de las CNN se basa en que la entrada estará compuesta por imágenes. La arquitectura CNN constará de tres tipos de capas. Las capas convolucionales, las capas de *pooling*, que permiten reducir la resolución espacial, y las capas fully-connected que están presentes en las últimas capas del modelo [45].

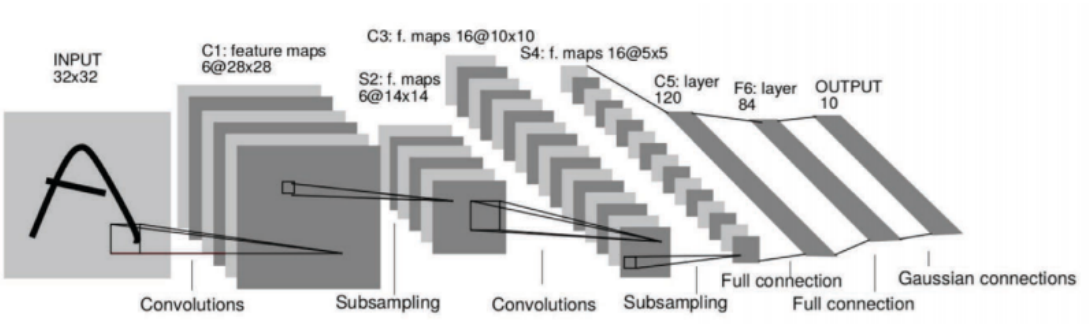


Figura 4.2: Ejemplo de modelo convolucional, CNN

Hay muchos tipos de redes convolucionales, entre ellas están las CNN y arquitecturas más específicas como AlexNet [46], ResNet [47], etc. Para nuestro caso de estudio hemos desarrollado una pequeña CNN. Una vez obtenidos los resultados, se comparará con un modelo del estado del arte.

4.2.1. Aritmética de convoluciones

Para el entendimiento de la arquitectura de los modelos de redes convolucionales (CNN) que se han desarrollado, se procede a definir una serie de conceptos y funciones que se utilizan en de desarrollo de los mismos [48].

Pooling es de las operaciones más importantes en las Convolutional Neural Network(CNN). Esta función consiste en reducir el tamaño de las características que se le pasa como entrada usando alguna función específica. Esto se logra pasando una "ventana"por la entrada, obteniendo y pasando el contenido leído a la función de Pooling. La función de

Pooling que vamos a utilizar es una de las más conocidas, la denominada *maxpooling*. Esta consiste en dejar el máximo valor que se nos ha pasado desde la ventana, como se puede ver en la Figura 4.3 la cual ha sido obtenido del trabajo propuesto por Vincent Dumoulin y Francesco Visin [48].

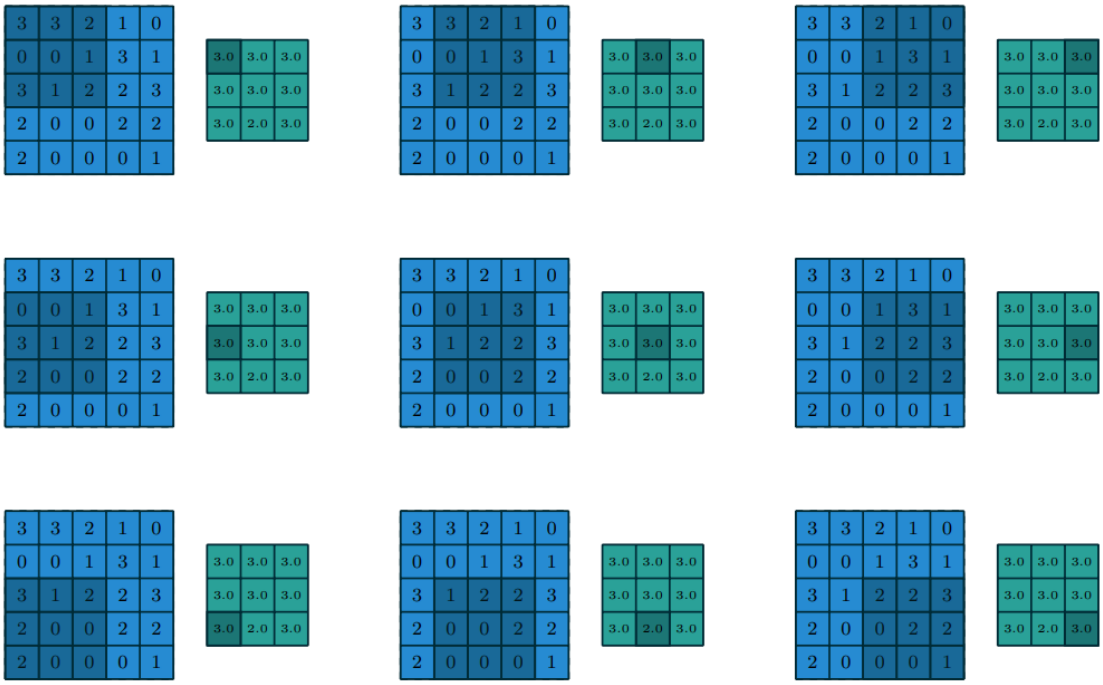


Figura 4.3: Ejemplo del uso de la función maxpooling.

Padding consiste en añadir una dimensión imaginaria a la entrada. Normalmente con la función pooling se reducen las dimensiones de la entrada, pero en algunos casos no queremos que esto ocurra, esto se logra haciendo uso del Padding. Se añade una o más dimensiones con valores 0 y así se logra no reducir las dimensiones con la función pooling utilizada. En la figura 4.4 se puede apreciar el funcionamiento de padding. Añadiendo una dimensión de 0 a la matriz de entrada y teniendo un stride a 1 se consigue que la salida tenga la misma dimensión que la entrada.

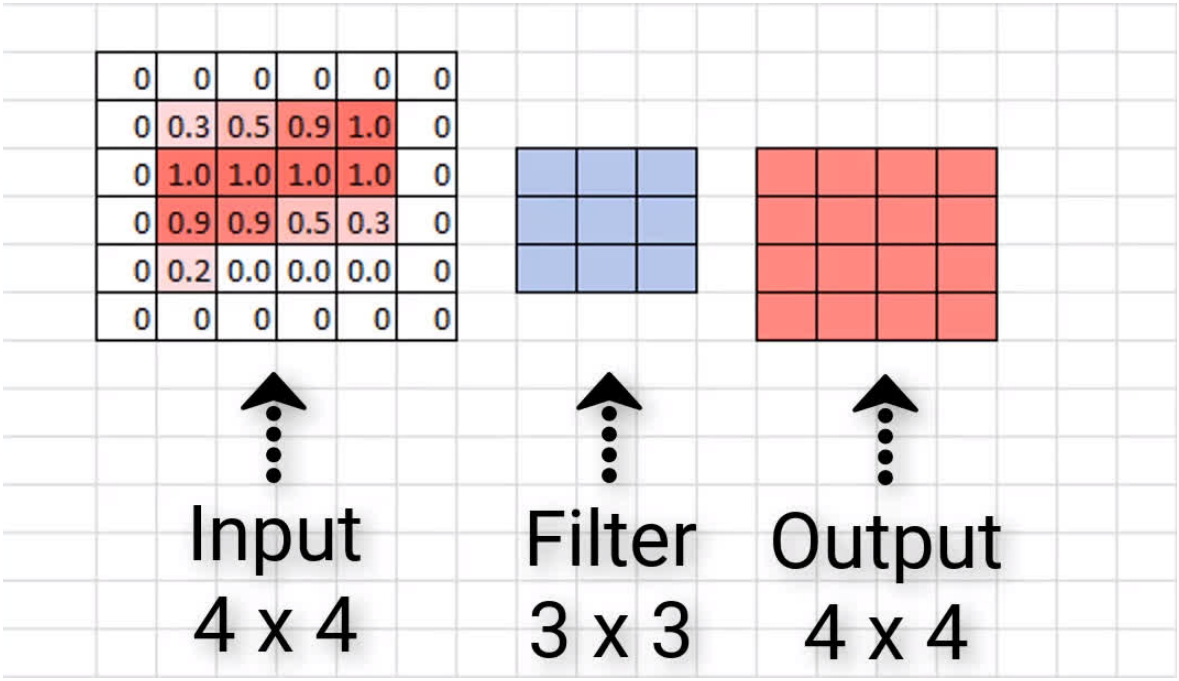


Figura 4.4: Ejemplo del uso de padding

Mientras que aumentando el valor de **Strides** se consigue totalmente lo contrario. Strides define cuántos píxeles avanza la ventana por cada etapa cuando se le pasa a la función de entrada4.5.

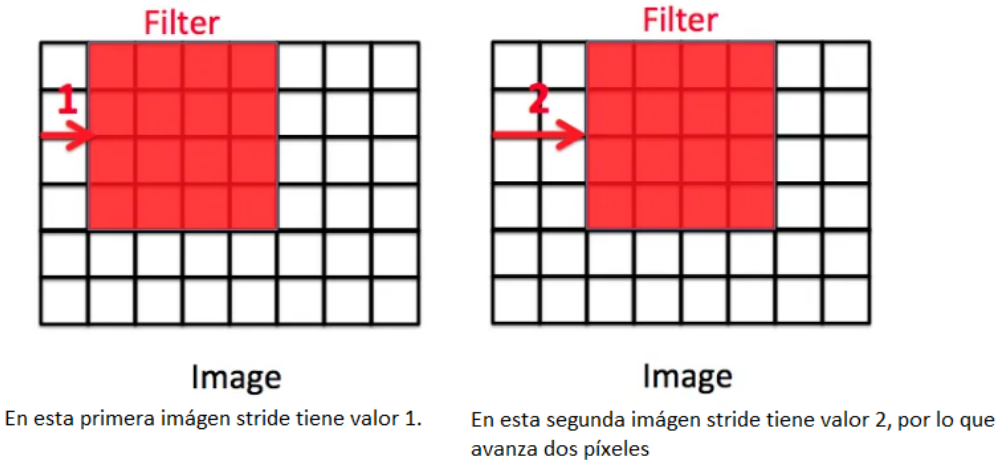


Figura 4.5: Ejemplo del uso de stride

4.3. Modelo VGG

Como referencia de modelo del estado del arte se utilizará VGG, Visual Geometry Group de la universidad de Oxford [49]. El modelo

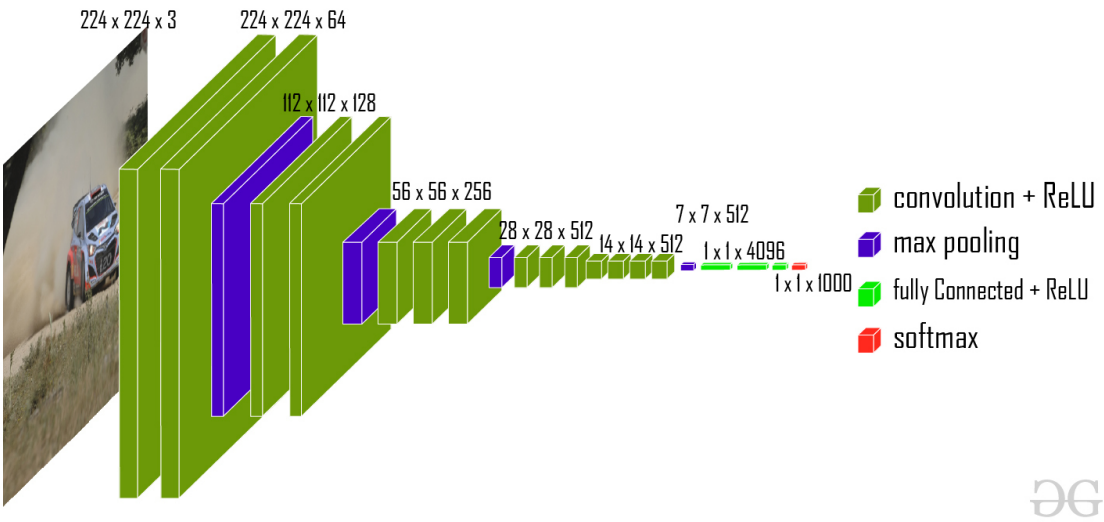


Figura 4.6: Arquitectura del modelo VGG16

VGG es un tipo de redes convolucionales que se centra principalmente en la profundidad de la red. Sus creadores demostraron que aumentar la profundidad de la red puede provocar una mejoría en el rendimiento de la misma.

En este modelo las capas convolucionales no son las encargadas de modificar el tamaño de las imágenes. Por lo tanto, sus núcleos de convolución serán de 3x3 con padding 1 y stride 1. El que se encarga de realizar esta disminución será el maxpooling con una dimensión de 2x2 y con un stride de 2. Además, tendrá finalmente tres capas concatenadas fully-connected.

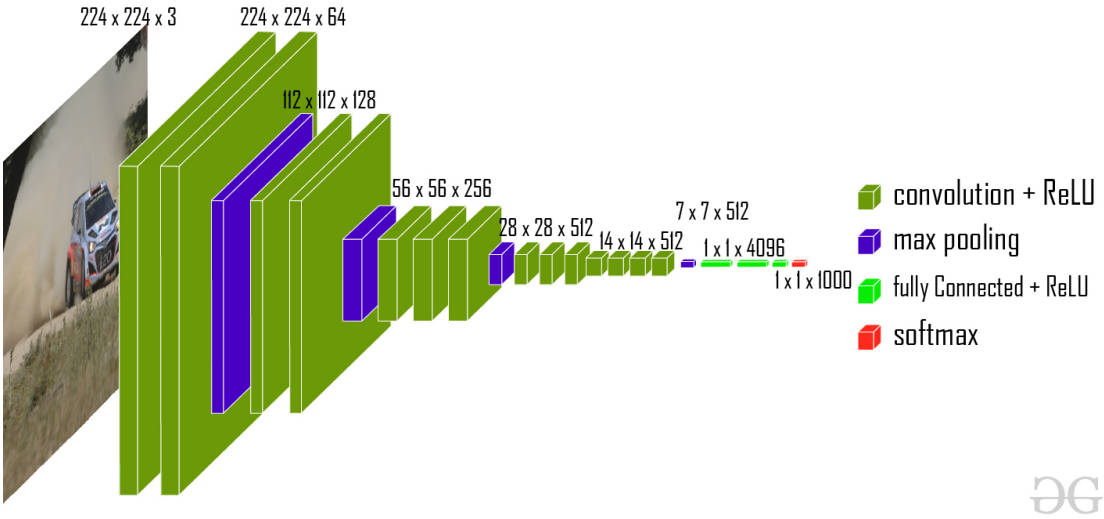


Figura 4.7: Arquitectura del modelo VGG16

4.4. Experimentos a realizar

Hemos decidido realizar 4 experimentos de optimización en nuestros modelos de clasificación. Con estos experimentos buscamos evaluar las diferentes técnicas de control de balance de imágenes que se están utilizando en la actualidad. Los casos a tratar son los siguientes:

- **Modelo CNN con CE** como función de pérdida y las diferentes soluciones planteadas con las técnicas de Data Augmentation.
- **Modelo CNN con Focal Loss** como función de pérdida.
- **Modelo VGG con CE** como función de pérdida y las diferentes soluciones planteadas con las técnicas de Data Augmentation.
- **Modelo VGG con Focal Loss** como función de pérdida.

Por parte de los experimentos con la función de pérdida Focal Loss hemos decidido no solo optimizar los hiperparámetros *learning rate* y *momentum*, sino también optimizaremos los hiperparámetros *alpha* y *gamma* de la función de pérdida *Focal Loss*.

4.5. Métricas de evaluación

La preparación de nuestros modelos, funciones de pérdida o de los datasets tienen gran importancia, pero también es muy importante medir el rendimiento de nuestro modelo y experimentos[50]. Para esto se hace uso de diferentes métricas de precisión. Después del proceso de entrenamiento de nuestro modelo tenemos que realizar una tarea de evaluación de los resultados.

En nuestro trabajo se hace una tarea de análisis comparando nuestros resultados con el de otros modelos, para esto nos apoyamos en diferentes métricas de precisión. La métrica principal con la que se realiza la tarea de análisis es la matriz de confusión. Esta matriz nos sirve para realizar una evaluación de nuestro modelo. Una matriz de confusión de tamaño $n \times n$ muestra la clasificación predicha por nuestro modelo y la clase real a la que pertenece cada muestra clasificada [51]. Los valores que aparecen en la matriz de confusión, véase Figura 4.8, son los siguientes:

- **True positive**. Se clasifican como true positive las predicciones positivas correctas.
- **True negative**. Se clasifican como true negative las predicciones negativas correctas.
- **False positive**. Se clasifican como false positive las predicciones positivas incorrectas.
- **False negative**. Se clasifican como false negative las predicciones negativas incorrectas.

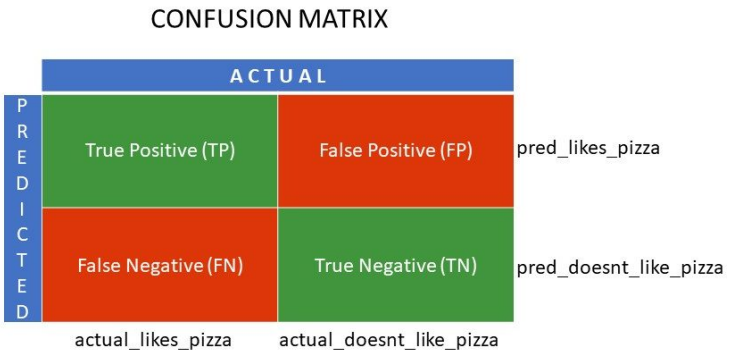


Figura 4.8: Matrix confusions

A partir de la realización de la matriz de confusión se pueden obtener diferentes métricas de precisión que nos pueden ayudar a analizar nuestro modelo. Para la correcta evaluación del modelo hay que tener en cuenta las diferentes métricas que se van a proponer, ya que si se realiza un análisis del modelo con una única métrica de precisión nos puede llevar a conclusiones erróneas.

4.5.1. Sensibilidad, precisión y F-Score

Sensibilidad y precisión son métricas de precisión bastante utilizadas en los problemas de clasificación de imágenes, sobretodo en los problemas de detección de anomalías [52]. La métrica sensibilidad nos devuelve un número decimal que se refiere a las anomalías reales detectadas con éxito. Mientras que la métrica precisión es otro número decimal que se refiere a todas las anomalías detectadas que son anomalías reales. Esta métrica de precisión tiene una fuerte relación con la métrica precisión. Analizando

estos dos valores se puede analizar la evaluación de nuestro modelo y entender para qué sirven estas métricas, se dice que mientras mayor sean estos dos valores mejor se está comportando nuestro modelo.

Para el cálculo de estas dos métricas se usa la fórmula 4.3 para la métrica sensibilidad y la formula 4.4 para el cálculo de la métrica precisión. Como se puede observar, se rescatan los términos true positive (TP), false negative (FN) y false positive (FP) que se han visto en el anterior (matriz de confusión).

$$sensibilidad = \frac{TP}{TP + FP} \tag{4.3}$$

$$precision = \frac{TP}{TP + FN} \tag{4.4}$$

A partir de estas dos métricas ya calculadas se ha decidido calcular la métrica F-Score. Esta métrica se calcula con la fórmula 4.5. La puntuación de esta fórmula oscila entre el 0 y el 1, y al igual que en las métricas que intervienen en ella mientras mayor sea el número es que el modelo está presentando un mejor rendimiento.

$$F - Score = \frac{2 * (precision * sensibilidad)}{precision + sensibilidad} \tag{4.5}$$

4.5.2. Specificity

La especificidad[53] es el número de muestras correctamente identificadas como negativos fuera del total de negativos. Realmente esta métrica es el antónimo de la métrica sensibilidad, como se puede ver reflejado en la fórmula 4.6.

$$Specificity = \frac{TN}{TN + FP} \tag{4.6}$$

4.5.3. Accuracy

La métrica accuracy, o exactitud, es de las métricas de precisión más utilizadas en la actualidad. Esta métrica nos indica la cantidad de

muestras clasificadas correctamente en comparación con el número total de muestras. Mientras mayor sea el valor obtenido de esta métrica, mejor rendimiento está dando nuestro modelo de clasificación. En la tarea de análisis se debe tener cuidado con esta métrica, ya que es una de las métricas más afectadas por datasets incorrectamente balanceados. La métrica accuracy se muestra en la fórmula 4.7.

$$Accuracy = \frac{TN + TP}{TN + TP + FP + FN} \tag{4.7}$$

4.6. Evaluación

En esta sección se realiza una comparación de los resultados de todos los experimentos realizados con los resultados de otros proyectos y publicaciones. Cabe destacar que los resultados que hemos obtenido provienen de realizar una tarea de optimización mediante optimización bayesiana. Se realiza el entrenamiento y evaluación para comparar los resultados después de haber obtenido los parámetros óptimos.

Los experimentos se dividen en dos grandes categorías, una para los experimentos realizados para la resolución del problema de clasificación y otro para los experimentos realizados para la resolución del problema de segmentación. En estos experimentos, se ha prestado atención a la configuración del optimizador estocástico utilizado para entrenar las redes neuronales. concretamente, el Stochastic Gradient Descent (SGD). Los hiperparámetros de SGD que proponemos optimizar, dada su importancia, son los hiperparámetros *learning rate* (lr) y *momentum* [54]. El algoritmo que se usa para descender por el gradiente no se caracteriza por su velocidad, se han ido implementando mejoras con los años que han mejorado este aspecto, pero solo ocurre en las primeras iteraciones. Por esto es tan importante optimizar el *momentum* del SGD, ya que con este hiperparámetro le damos una pequeña memoria a nuestro algoritmo y no perdemos del todo la velocidad con el paso de las iteraciones [55]. Por parte del lr, es un hiperparámetro que controla cuánto debería cambiar nuestro modelo en relación con lo que se obtiene en la función de pérdida, poner valores muy bajos o muy altos puede condicionar para mal, los resultados de nuestro modelo.

Como ya se ha comentado anteriormente, el optimizador bayesiano se retroalimenta de optimizaciones pasadas. Se ha decidido realizar 100

rondas de optimización por experimento, de las cuales en cada ronda se optimiza el proceso de entrenamiento y evaluación de nuestro modelo, buscando así los hiperparámetros óptimos. Aparte, por cada ronda de optimización se ha propuesto que el entrenamiento, donde se optimiza la red neuronal, dure 25 épocas. Cabe resaltar que mientras más parámetros intentemos optimizar, el espacio muestra se ampliaría y, por lo tanto, más rondas de optimización serían necesarias para obtener los valores óptimos. Hay algunos experimentos en los que se intenta optimizar 4 parámetros, en estos casos lo ideal sería aumentar el número de rondas de optimización. Se ha decidido no hacerlo porque hay experimentos donde puede llegar a tardar días la optimización, debido al tiempo que se maneja en este proyecto se decide dejarlo en 100 rondas para todos los experimentos por igual, independientemente de sus necesidades.

En el desarrollo de la resolución del problema de clasificación se han planteado una serie de soluciones para resolver el problema del balanceo en el *dataset* de imágenes. Se realizan diferentes optimizaciones para las diferentes soluciones propuestas para este problema. Se utiliza para estos experimentos el uso de dos modelos, el modelo VGG y un modelo CNN básico hecho por nosotros.

Para el desarrollo de los tres primeros experimentos se ha desarrollado una red convolucional básica menos compleja y más pequeña que la *VGG*, reduciendo el número de parámetros en comparación. La arquitectura de este modelo consta de 5 capas convolucionales. En la segunda y quinta capa se aplica una función *maxpooling* con *stride* 2. Y una última capa *fully-connected*, la cual es la capa de salida que genera la estimación en la clasificación.

4.6.1. Modelo CNN con Cross-Entropy

En este primer experimento se utiliza la función de pérdida CE. Se implementan algunas técnicas de *Data Augmentation* . Cabe resaltar que la técnica no es lo suficientemente efectiva como para solucionar el problema del desbalanceo del *dataset*, pero nos sirve para ver reflejado en la matriz de confusión lo que ocurre con un *dataset* no balanceado.

Para este experimento, debido a la simplicidad del modelo y de la función de pérdida, se decide optimizar únicamente los hiperparámetros *lr* y *momentum* del SGD. La figura 4.9 ilustra la distribución de ambos parámetros consideradas variables aleatorias para el optimizador baye-

siano. Observando ambas gráficas generadas, la mayoría de los puntos se concentran principalmente en dos grandes zonas. No se encuentran muchos resultados en otras zonas, ya que el optimizador bayesiano no obtuvo buenos resultados en esas franjas de valores. Mirando las gráficas se puede decir que el hiperparámetro *momentum* tiene un excelente comportamiento cuando su valor es 0 y cuando está alrededor del 0,5. Mientras que el *lr* está entre 10^{-1} y 10^{-2}.

Basándonos en lo anterior ya es posible deducir los rangos de valores óptimos para los hiperparámetros. En este caso, el framework AX propone como solución óptima los valores $lr = 0{,}0088$ y $momentum = 0{,}0$ para el SGD, es decir, el momentum se descarta en este caso.

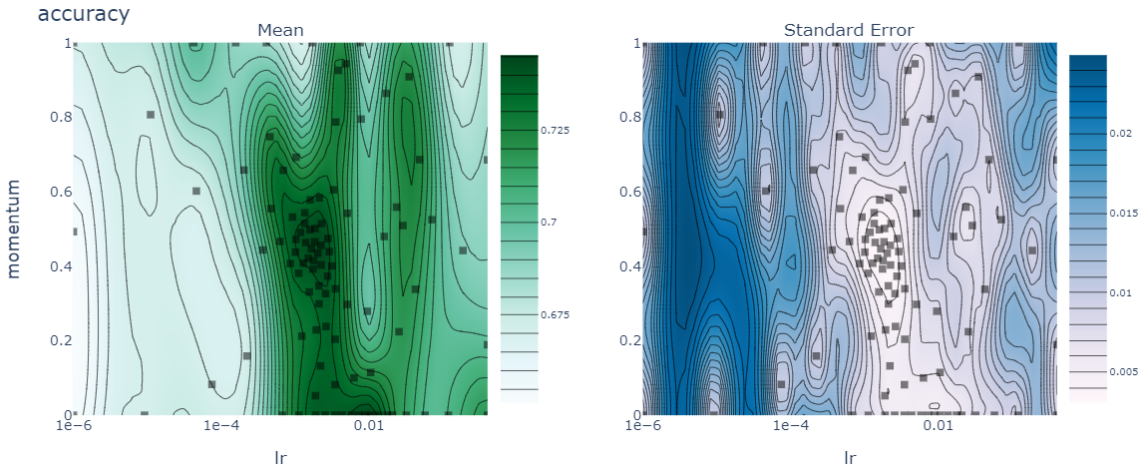


Figura 4.9: Distribución de los hiperparámetros *learning rate* (lr) y *momentum* para el optimizador SGD.

Una vez obtenidos los parámetros óptimos, se procede a entrenar el modelo y evaluarlo con un subconjunto del dataset, que no ha formado parte del entrenamiento. El propósito es testear. Este subconjunto del dataset contiene un total de 2005 imágenes como se menciona en la Tabla 4.1, estas muestras no están balanceadas. Este conjunto se conoce como conjunto de test. Los resultados obtenidos con el subconjunto de test muestra un *accuracy* de '0.74', véase Tabla 4.3. A simple vista parece un valor bastante aceptable dentro de lo que cabe con la capacidad del modelo de la red neuronal que hemos escogido, pero esto no es correcto. Analizando la matriz de confusión, Figura 4.10, se ve reflejado con claridad el problema ya comentado del *dataset* no balanceado. Como se puede apreciar, el haber realizado el entrenamiento de la red neuronal con un *dataset* no balanceado y no habiendo aplicado ninguna solución al respecto, provoca que las clases con menor porcentaje de imágenes en

el *dataset* sean más difíciles de clasificar. La clasificación de estas imágenes se ha realizado, en su mayoría, como si fuesen de la clase con mayor número de imágenes. Esto provoca que las imágenes de la clase que tiene la mayoría de las imágenes se clasifiquen correctamente y generando un *accuracy* bastante elevado. Además, analizando las otras métricas de precisión planteadas en la tabla 4.3, se observa que las métricas *precision*, *Recall* y *F-score* están gravemente perjudicadas en la mayoría de los casos. Indicando esto que la clasificación entre esas clases no se está haciendo correctamente, obteniendo un número preocupante de falsos negativos y falsos positivos.

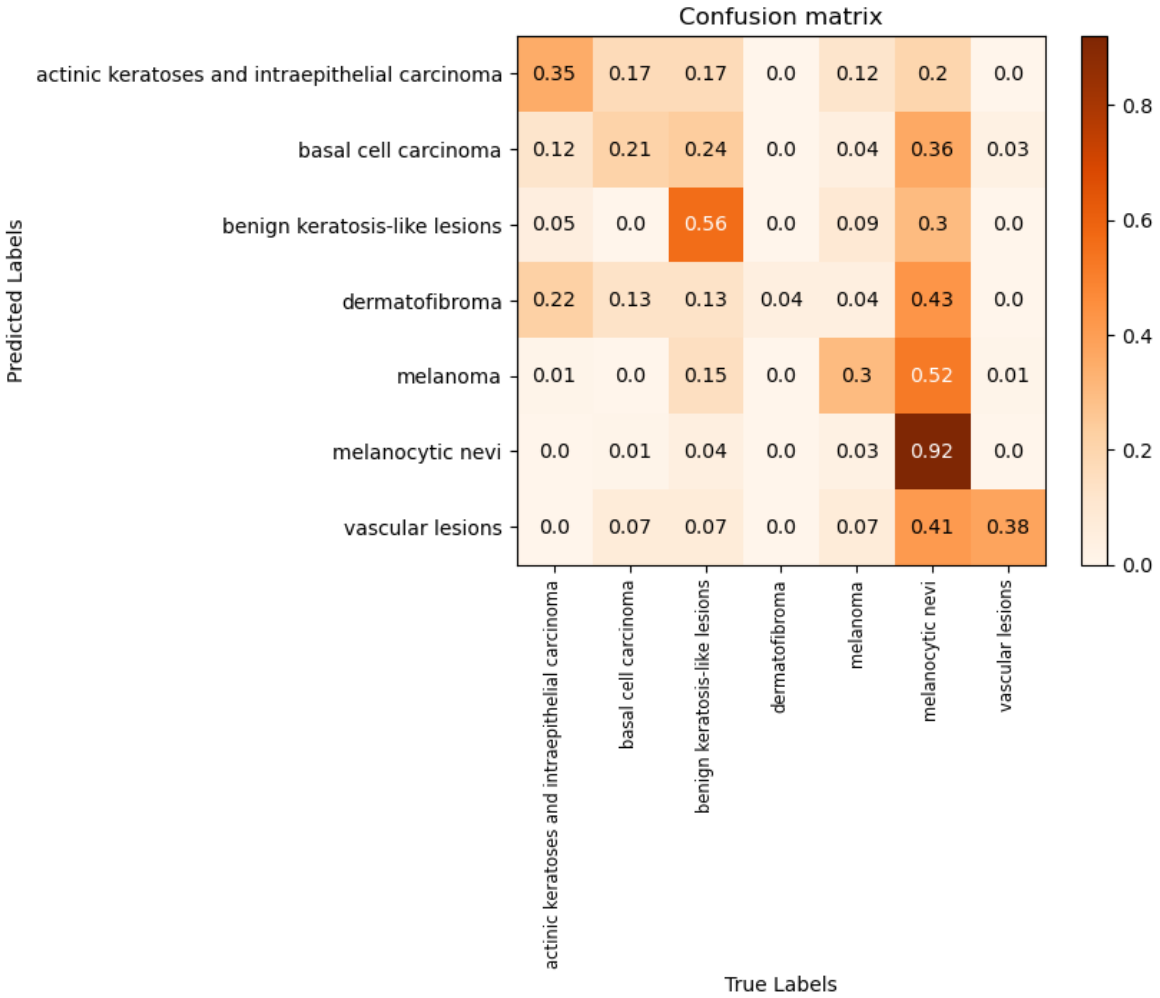


Figura 4.10: Imagen matriz de confusión sin muestreo ponderado

Comparando los resultados con el trabajo 'Deep Learning aplicado al Dataset MedMNIST' [6], véase Figura 4.11, se puede deducir que gracias a nuestro optimizador hemos conseguido obtener una configuración de hiperparámetros más eficiente. En cuanto al *accuracy*, su trabajo reporta una exactitud '0.528' mientras que nosotros un '0.740'. Entrando

Tabla 4.3: *Metricas de precisión para el modelo CNN con funcion de pérdida cross entropy con accuracy general de 0.74*

Category name	Presicion	Recall	Specificity	F-score
actinic keratoses...	0.348	0.548	0.978	0.426
basal cell cercinoma	0.35	0.456	0.965	0.396
benign keratosis-like lesions	0.427	0.55	0.931	0.481
dermatofibroma	0.043	1.0	0.989	0.082
melanoma	0.251	0.496	0.912	0.333
Melanocytic nevi	0.939	0.802	0.812	0.865
Vascular lesions	0.517	0.5	0.993	0.508

en detalles, en su matriz de confusión (Figura 4.11), vemos que con sus parámetros y con la configuración que él realizó en su entrenamiento se está empezando a notar el incorrecto balance del *dataset*. Es decir, se empieza a decantar por la clase predominante, al igual que nos está pasando a nosotros con nuestros parámetros. Aunque hayamos obtenido un mejor *accuracy* que el trabajo previo, esto no significa que nuestro modelo con nuestra configuración sea mejor que él suyo. Analizando ambas matrices de confusión podemos deducir que indiferentemente del *accuracy* no estamos generando resultados fiables.

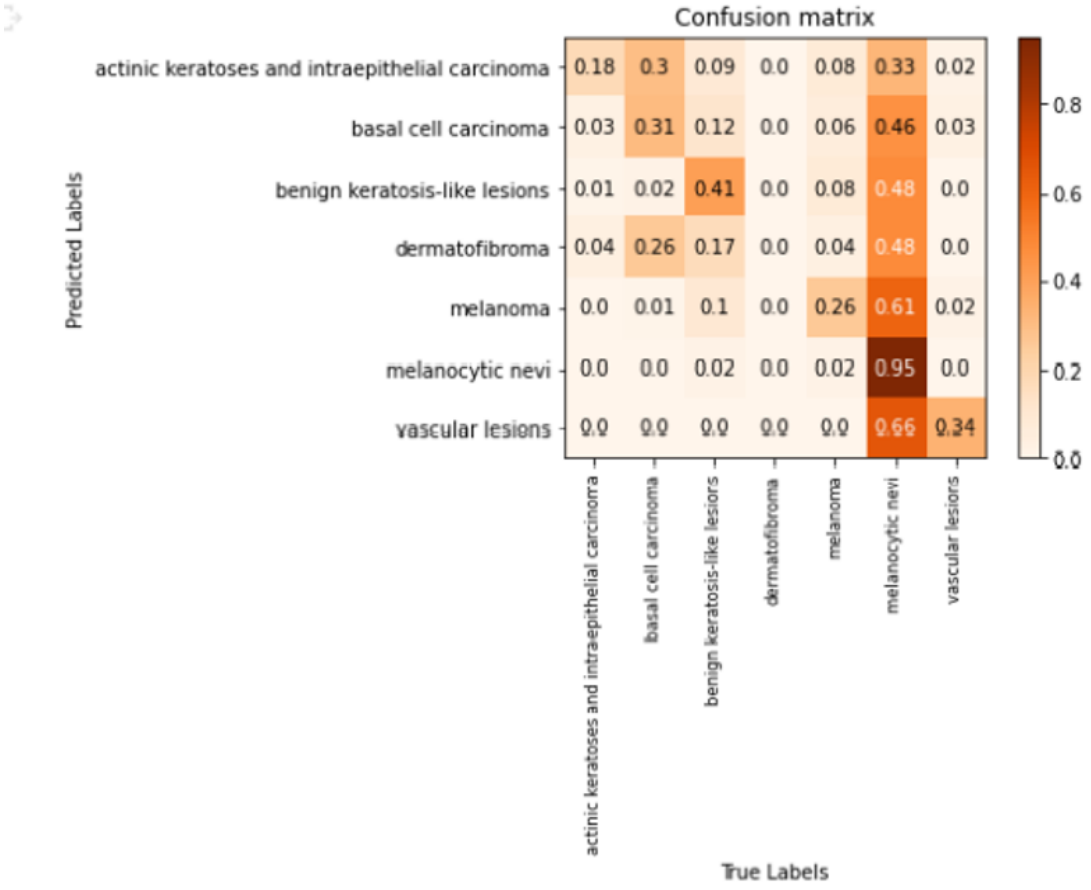


Figura 4.11: Imagen matriz de confusión sin muestreo ponderado de nuestro compañero

4.6.2. Modelo CNN con Cross-Entropy y muestreo ponderado

En este segundo experimento se continúa utilizando la función de pérdida CE. La mayor diferencia con el anterior experimento es que en este se utiliza la técnica de muestreo ponderado, véase sección 4.1.1, para intentar corregir el problema que se tiene con el balance del *dataset*. Además, se siguen utilizando las técnicas empleadas en el anterior apartado de *data augmentation* en las imágenes.

En este experimento hemos vuelto a optimizar solamente el *lr* y el *momentum* al igual que en el anterior experimento. Las diferentes áreas que ha decidido explorar el optimizador principalmente son diferenciadas por el valor que tiene *momentum*. Se sigue obteniendo, al igual que con el primer experimento, los mejores *accuracy* con *momentum* a 0 y cuando está alrededor de 0.5. A diferencia del anterior, la mayoría de los experimentos que ha decidido realizar el optimizador bayesiano se concentran cuando el *momentum* oscila el valor 0.5. Además, en el primer experimento surgieron experimentos aceptables en distintas zonas aisladas, generando pequeñas áreas de interés. Mientras que en este se concentran todos en dos áreas principales. Por parte del *lr*, analizando los dos experimentos se puede ver que ha tenido un ligero crecimiento en este segundo. En el primer experimento sus valores oscilaban entre 10^{-3} y 10^{-2}, mientras que ahora oscilan entre 10^{-2} y 10^{-1}.

Finalmente, los valores que se han obtenido con el optimizador corresponden con $lr = 0{,}0108$ y $momentum = 0{,}4959$. De esta forma, tenemos la configuración óptima para entrenar el VGG mediante el optimizador SGD.

Los resultados obtenidos en este segundo experimento generan un menor *accuracy*. El *accuracy* obtenido es 0,698, véase Tabla 4.4. Esta pequeña bajada en los aciertos en comparación con el primer experimento es lo esperado, también se prevé que pase algo similar en el siguiente experimento. En este segundo experimento se ha usado una técnica bastante eficaz para tratar con los *datasets* no balanceados, por ende ya no se comete el mismo error que se cometía en el primer experimento que nos daba un *accuracy* alto porque la mayoría de las imágenes solo pertenecía a una única clase.

Analizando la matriz de confusión (Figure 4.13), se puede decir que este *accuracy* aunque sea levemente más bajo, es en definitiva mejor y

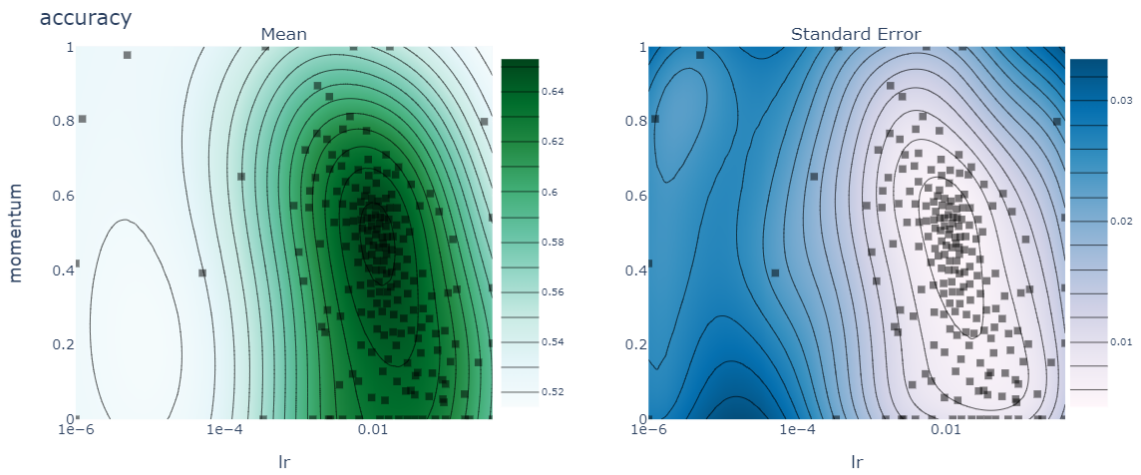


Figura 4.12: Distribución de los hiperparámetros *learning rate* (lr) y *momentum* para el optimizador SGD.

más fiable que el obtenido en el primer experimento. Se sigue teniendo problemas detectando varias clases del dataset como puede ser para la clase "dermatofibroma", y "melanoma". Esto posiblemente se deba a que las imágenes de estos dataset son por naturaleza más difíciles de clasificar. Examinando el resto de valores de las métricas de precisión de la Tabla 4.4 se ve una mejoría significativa en las métricas *Presicion*, *Recall* y *F-score*. Estas métricas tienen en cuenta para su cálculo los falsos positivos y los falsos negativos, al obtener mejores resultados en este experimento se concluye que este experimento se comporta mejor en comparación con el primero frente a los falsos positivos y a los falsos negativos. En este caso, como con esta técnica acabamos con el balance incorrecto de imágenes y, aun así, los mejores resultados se centran en la clase *Melanocytic Nevi*, se puede decir que las imágenes de esta categoría son más fáciles de clasificar que las del resto de categorías.

Haciendo una comparación entre la matriz de confusión obtenida en la Figura 4.13 en este experimento con la anterior Figura 4.10 se puede ver claramente la diferencia entre cuando se utiliza un *dataset* no balanceado y un *dataset* que sí lo está. En la matriz de confusión lograda se puede observar que las clases con menor número de imágenes ya no son clasificadas, en su mayoría, como si fuesen de la clase *Melanocytic Nevi*, que es la que tiene el mayor número de imágenes de este *dataset*. Viendo ambas matrices generadas se puede concluir que la solución propuesta con el muestreo ponderado soluciona perfectamente el problema que se obtiene con el incorrecto balance de las imágenes.

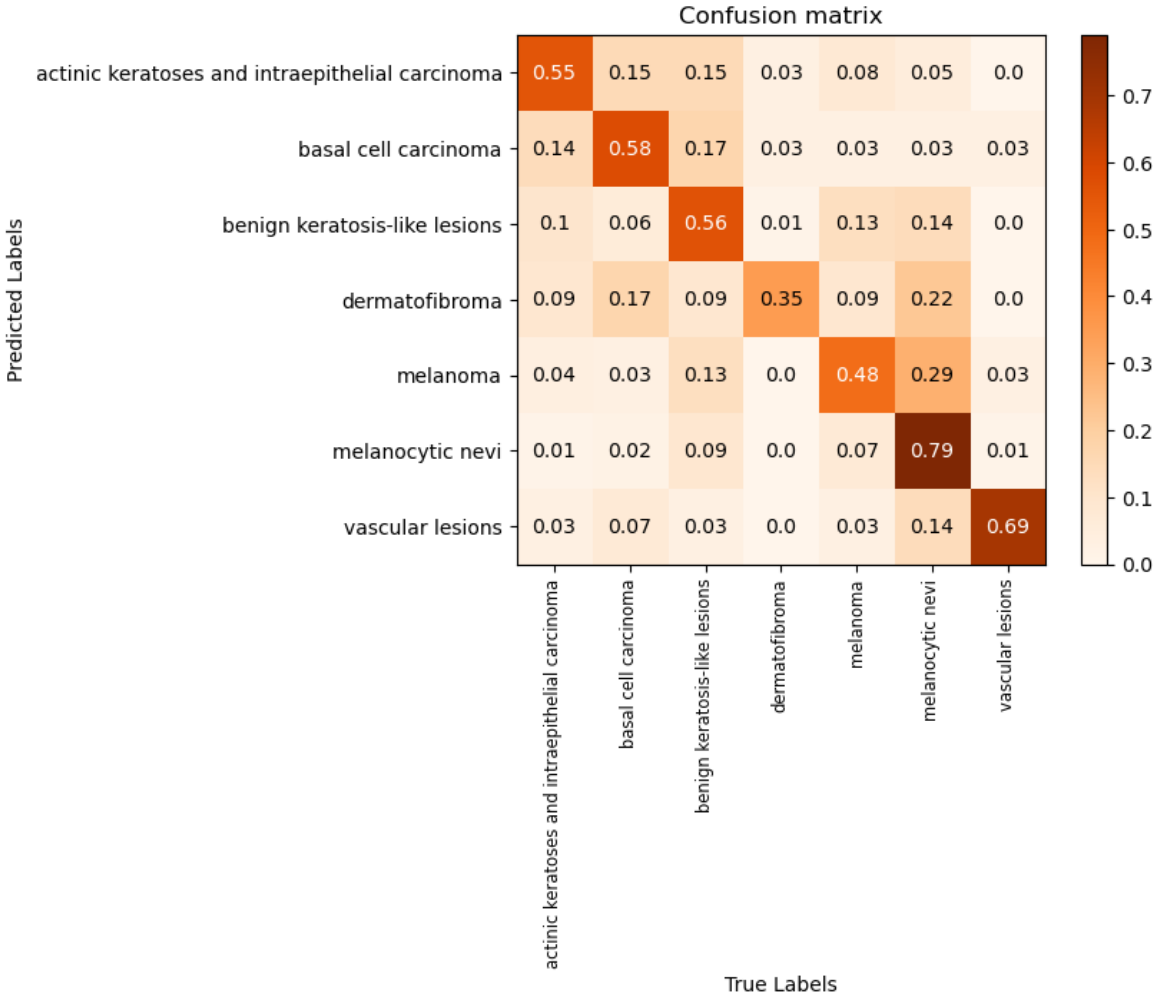


Figura 4.13: Imagen matriz de confusión con muestreo ponderado

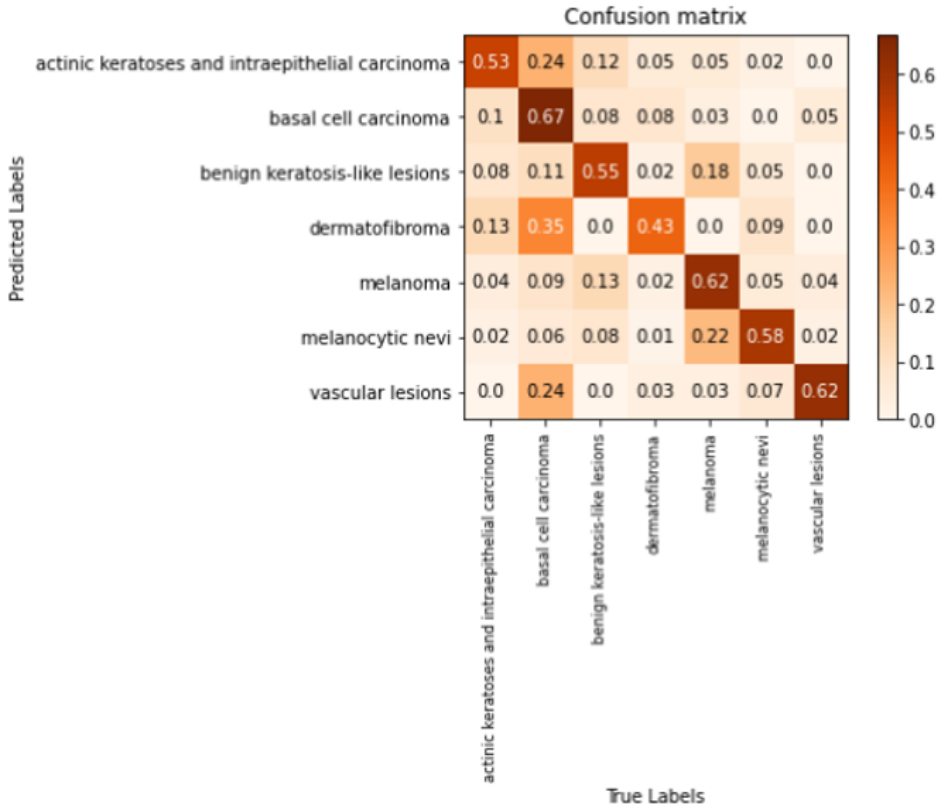


Figura 4.14: Imagen matriz de confusión con muestreo ponderado de nuestro compañero

Tabla 4.4: *Metricas de precisión para el modelo CNN con funcion de pérdida cross entropy utilizando muestreo ponderado con accuracy gneral de 0.698.*

Category name	Presicion	Recall	Specificity	F-score
actinic keratoses...	0.394	0.392	0.979	0.388
basal cell cercinoma	0.631	0.492	0.98	0.553
benign keratosis-like lesions	0.573	0.439	0.945	0.497
dermatofibroma	0.348	0.381	0.992	0.364
melanoma	0.565	0.371	0.942	0.448
Melanocytic nevi	0.774	0.907	0.648	0.835
Vascular lesions	0.414	1.0	0.991	0.586

Podemos concluir que nuestra tarea de optimización ha sido satisfactoria. El *accuracy* que se obtuvo en el trabajo previo de Yuncai Chen [6] fue de 0,664, ligeramente menor que el nuestro. Comparando su matriz de confusión, Figura 4.14, con la nuestra vemos que él también consiguió corregir el problema del balance de las imágenes pero con un resultado final ligeramente peor que el nuestro. Se puede decir que gracias a nuestra tarea de optimización con el optimizador bayesiano se ha generado una mejor configuración que la que usó él en sus experimentos.

4.6.3. Modelo CNN con Focal Loss

En este tercer experimento se hace una ligera modificación en la función de pérdida. Usaremos la *Focal Loss*, que realmente es la función de pérdida ya utilizada en los anteriores experimentos, pero añadiendo dos factores importantes para intentar manejar el incorrecto balance de nuestro *dataset*. Se ha decidido no utilizar técnicas de *data augmentation* para así ver cómo se comporta y realizar comparaciones con los experimentos anteriores.

En la tarea de optimización actual se decide optimizar la función de pérdida, aparte del *lr* y *momentum*, hiperparámetros para el optimizador SGD, que ya se ha optimizado en los experimentos anteriores. Esto es debido a que hay dos parámetros bastante interesantes que se pueden optimizar de la función, en la función objetivo, *Focal Loss*. Estos dos parámetros son α y γ, véase ecuación 4.2. Al fin y al cabo, estos factores tienen un papel crucial para los resultados que obtenemos a partir del incorrecto balance del *dataset*.

Los valores de los hiperparámetros que se han obtenido del optimizador son $lr = 0{,}0055$ y $momentum = 0{,}8319$. Por parte del lr es un valor bastante cercano al obtenido en los otros experimentos, pero por parte del *momentum* se ha obtenido un valor bastante diferente. En este experimento se vuelve a diferenciar en las gráficas de la Figura 4.15 dos grandes áreas, pero diferentes con respecto a los experimentos anteriores. En este experimento no se está detectando que poner el *momentum* a 0 sea una buena opción, los valores del *momentum* en las dos áreas detectadas oscilan el 0.2 y el 0.8.

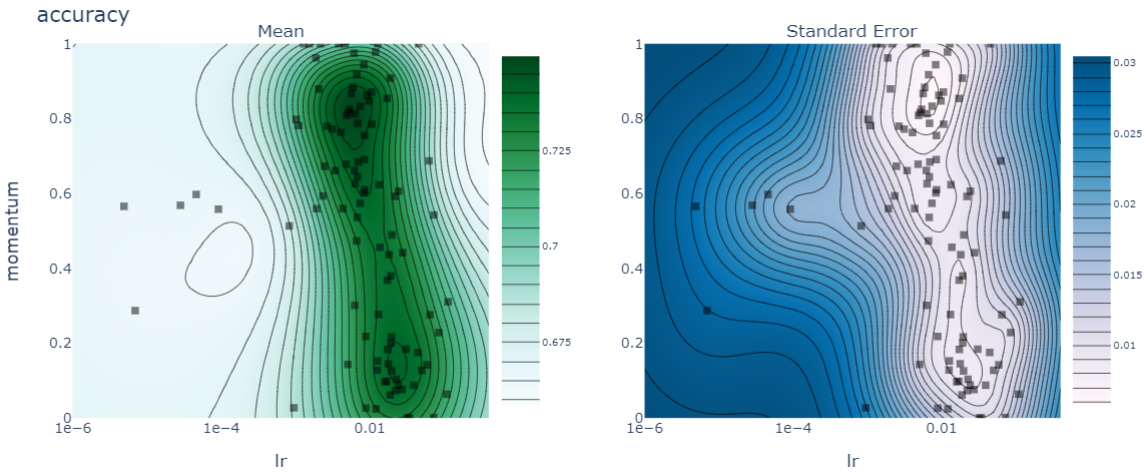


Figura 4.15: Distribución de los hiperparámetros *learning rate* (lr) y *momentum* para el optimizador SGD.

Mientras que los valores de los parámetros de la función de pérdida que se han obtenido son para γ '3.4208' y para α '0.4031'. Comparando estos valores con los del trabajo 'Focal Loss for Dense Object Detection' [56] de la función de pérdida utilizada se puede observar que difieren ligeramente. La configuración óptima con la que hicieron pruebas en este trabajo ha sido α a '0.25' y γ a '2.0'. Si analizamos ambas gráficas, Figura 4.16, geenradas en el proceso de optimización, se puede comprobar que estos valores están ligeramente lejos de la zona de calor, que es donde se ubican las mejores configuraciones para nuestro modelo. Cabe resaltar que en este trabajo no hicieron pruebas con valores γ entre 2 y 5 mientras que, según nuestro optimizador, los mejores valores oscilan sobre esa franja.

Analizando la matriz de confusión obtenida en nuestro experimento, Figura 4.17, se puede ver que se ha llegado a una mejora del problema que existía con el balance de las imágenes en el *dataset* de la matriz de confusión del primer experimento. Por lo general, las imágenes clasifi-

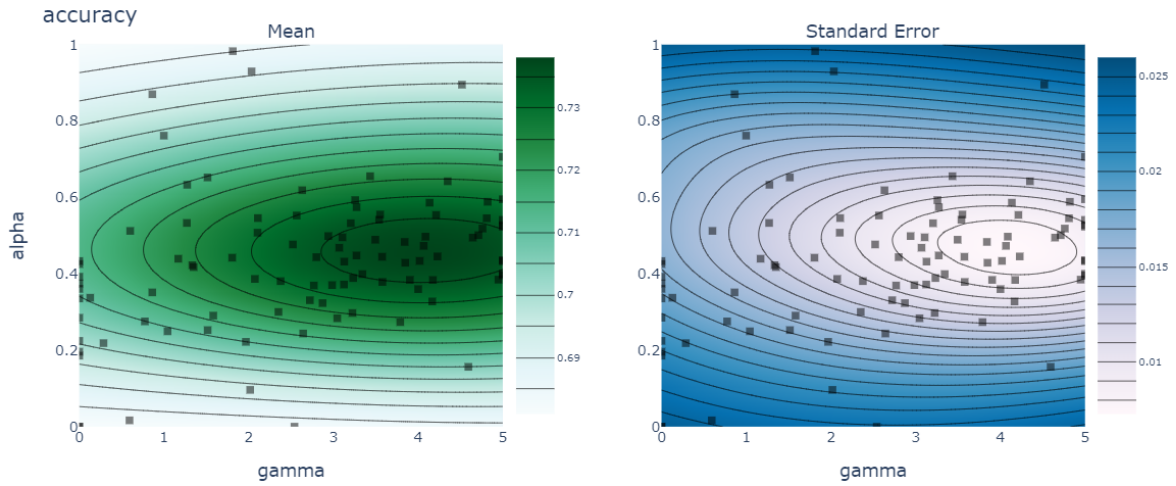


Figura 4.16: Distribución de los hiperparámetros 'gamma' (γ) y 'alpha' (α) para la función de pérdida Focal Loss

cadas erróneamente ya no se decantan tanto por ser clasificadas por la clase que tiene mayor número de imágenes. Pero, aun así, si se analiza en profundidad comparándola con la matriz de confusión del segundo experimento, se obtienen las siguientes conclusiones:

- Teniendo en cuenta las imágenes acertadas, en el segundo experimento por lo general la mayoría tienen un mayor porcentaje de aciertos.
- Las imágenes de las clases que se han clasificado incorrectamente en este último experimento tienden a clasificarse ligeramente más a la clase que tiene mayor número de imágenes.
- En este tercer experimento, la clase que contiene la mayoría de las imágenes, tiene un mayor porcentaje de acierto comparado con el resto. Mientras que en el segundo experimento la tasa de acierto está más igualada

Teniendo en cuenta estas conclusiones, se puede deducir que el uso de la función de pérdida Focal Loss para controlar los datasets que no están balanceados no es tan fiable como el uso del muestreo ponderado, a pesar de tener un mejor *accuracy*. Examinando el resto de métricas de precisión de la tabla 4.6, se ve una pequeña mejoría para algunas clases en comparación con el segundo experimento, pero, aun así, también se obtienen peores resultados para el resto de clases. En este experimento usando la función de pérdida Focal Loss no se ha conseguido solventar el problema del incorrecto balanceo del dataset de imágenes, pero se consigue clasificar mejor las imágenes más complejas del dataset.

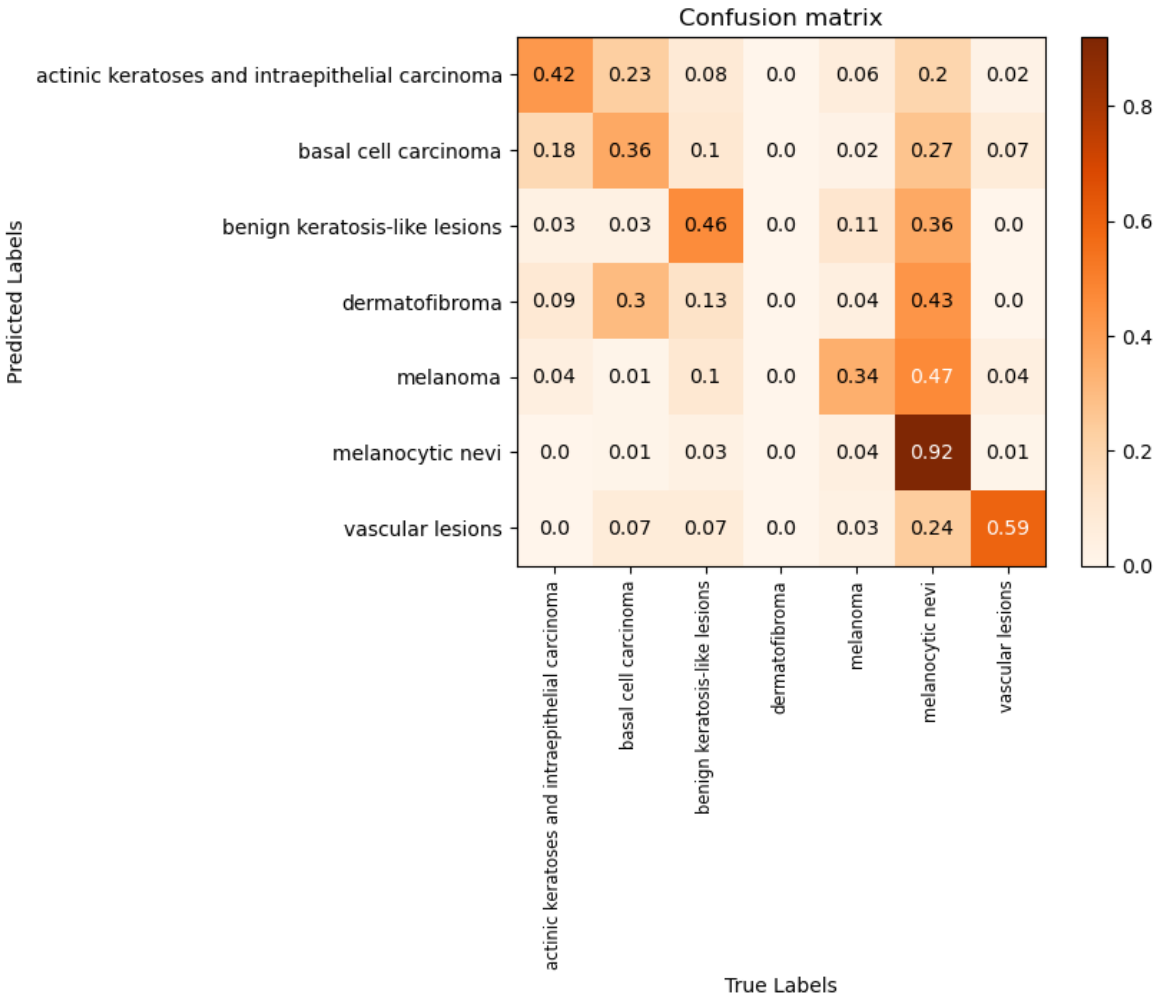


Figura 4.17: Imagen matriz de confusión con Focal Loss. γ y α

Tabla 4.5: *Metricas de precisión para el modelo CNN con función de pérdida focal loss con accuracy general de 0.737.*

Category name	Presicion	Recall	Specificity	F-score
actinic keratoses...	0.303	0.465	0.977	0.367
basal cell cercinoma	0.447	0.479	0.97	0.462
benign keratosis-like lesions	0.445	0.527	0.933	0.483
dermatofibroma	0	NaN	0.989	NaN
melanoma	0.229	0.49	0.91	0.312
Melanocytic nevi	0.946	0.814	0.837	0.875
Vascular lesions	0.483	0.737	0.992	0.584

Tabla 4.6: *Metricas de precisión para el modelo CNN con función de pérdida Focal Loss*

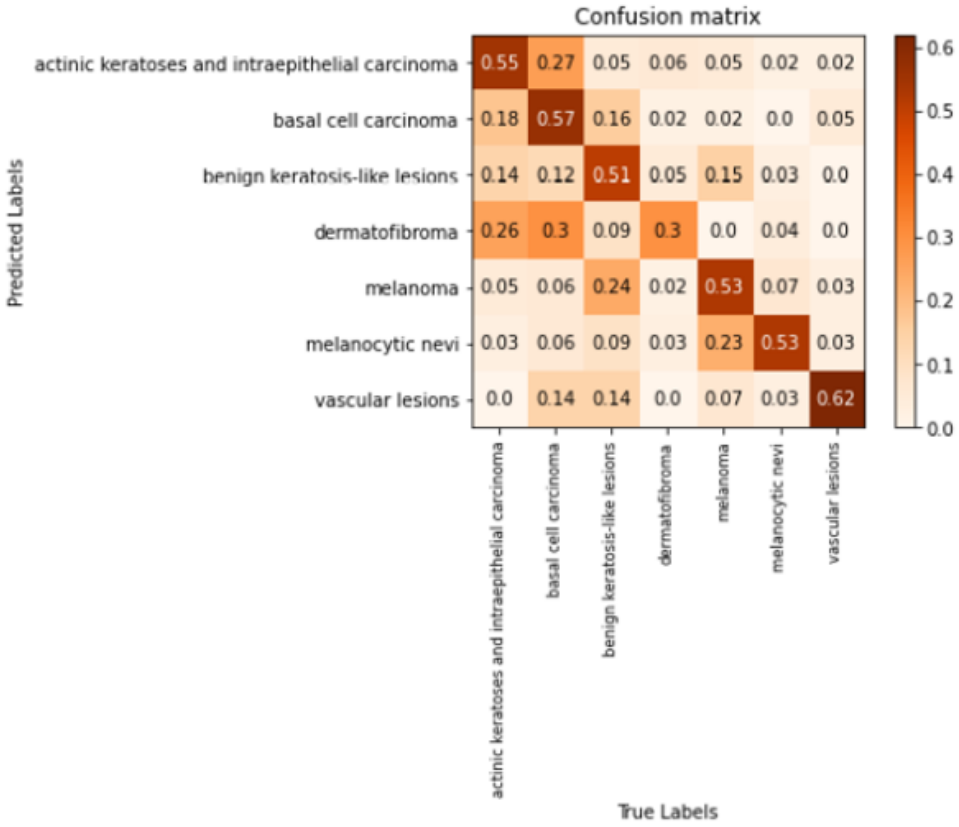


Figura 4.18: Imagen matriz de confusión con Focal Loss de nuestro compañero. γ y α

Analizando los resultados del proyecto de Yuncai Chen [6], su *accuracy* fue de '0.737', véase Tabla 4.6, que es levemente más bajo que el obtenido por el optimizador. Por lo tanto, nuestro proceso de optimización ha mejorado sus resultados. Cabe resaltar que el mejor resultado que obtuvo nuestro compañero fue utilizando γ con valor 5, un valor que no generó buenos resultados en el trabajo de Yuncai Chen [6]. Analizando su matriz de confusión de la Figura 4.18 vemos que tiene el mismo problema que nosotros con el balance de las imágenes, con la diferencia de que con nuestra configuración se obtiene mejores resultados.

4.6.4. Modelo VGG con Cross-Entropy

Este experimento tiene las mismas configuraciones que las realizadas en el primero. Con la diferencia que en este caso se hace uso del modelo VGG, que es un tipo de red convolucional. Se espera que con este modelo se obtengan mejores resultados con respecto al primer experimento al usar un modelo más complejo para realizarlo. Entre las configuraciones que se mantienen del primer experimento están la función de pérdida CE y algunas técnicas de data augmentation.

El modelo VGG utilizado es el modelo VGG16. Aunque este modelo sea ligeramente más complejo que el utilizado en el primer experimento, se seguirán optimizando únicamente los hiperparámetros *lr* y *momentum*. Los valores óptimos obtenidos por el optimizador son '0.0012' para el hiperparámetro *lr* y '0.0' para el *momentum*. Analizando la gráfica 4.19 generada por el optimizador, se puede apreciar cómo el optimizador bayesiano ha decidido explotar una misma franja de valores del *lr*, mientras que con el *momentum* ha variado bastante. Las mejores configuraciones de estos hiperparámetros, según el optimizador, se da cuando el *lr* está en el intervalo 10^{-4} y 10^{-2} y para el *momentum* se han encontrado configuraciones válidas en dos intervalos de valores. Para el intervalo 0.5-0.8 y para el intervalo 0-0.1. Pero las configuraciones que han tenido mayor *accuracy*, que es, por lo tanto, por donde el optimizador ha decidido elaborar más configuraciones, han sido aquellas en las que el *momentum* rondaba el valor 0.

Los resultados de las métricas de precisión obtenidos en nuestra etapa de entrenamiento y evaluación mejora levemente a los obtenidos en el primer experimento. Esto es lo esperado, ya que se está utilizando un modelo más complejo que en el primer experimento. El accuracy obteni-

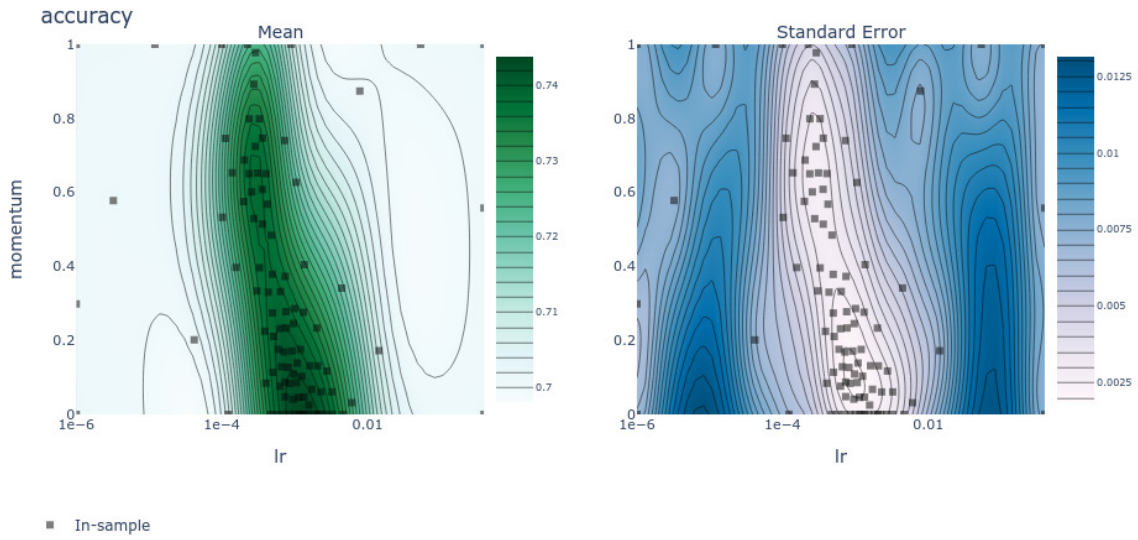


Figura 4.19: Distribución de los hiperparámetros *learning rate* (lr) y *momentum* para el optimizador SGD.

do de la Tabla 4.7, y las métricas de precisión en general mejoran las del primer experimento Tabla 4.3. Pero, aunque hayan sido mejoradas, nos encontramos con el mismo problema que nos encontrábamos en el primer experimento. El principal problema es el balanceo incorrecto del dataset de imágenes, provocando que las imágenes tiendan a clasificarse a las clases con mayor cantidad de imágenes. Esto produce que el accuracy tenga un valor bastante aceptable. Mientras que las métricas de precisión Recall, presicion y F-score siguen estando gravemente perjudicadas debido a este problema.

Tabla 4.7: *Metricas de precisión para el modelo VGG con funcion de pérdida cross entropy con accuracy general de 0.741.*

Category name	Presicion	Recall	Specificity	F-score
actinic keratoses...	0.364	0.436	0.978	0.397
basal cell cercinoma	0.447	0.455	0.97	0.451
benign keratosis-like lesions	0.455	0.541	0.934	0.494
dermatofibroma	0	NaN	0.989	NaN
melanoma	0.435	0.435	0.929	0.435
Melanocytic nevi	0.9	0.853	0.773	0.876
Vascular lesions	0.448	0.5	0.992	0.473

Al igual que en el primer experimento, este problema se ve reflejado perfectamente en la matriz de confusión Figura4.20, donde se ve claramente cómo las imágenes son clasificados en su mayoría como 'Me-

lanocytic nevi'.

Examinando los resultados obtenidos en el proyecto de Yuncai Chen [6], su accuracy tiene valor '0.73'. Por ende, aunque el accuracy obtenido por el optimizador sea levemente mejor, se ha obtenido una configuración de hiperparámetros más eficiente que logra mejorar sus resultados. Analizando su matriz de confusión 4.21 y la nuestra, se puede concluir que hemos mejorado sus resultados, pero debido a que todavía no se ha efectuado ningún tratamiento contra el incorrecto balanceo del dataset no se considera que este accuracy se deba tener en cuenta. Ya que mirando en conjunto las métricas de precisión obtenidas se ve claramente que nuestro modelo no logra clasificar bien la mayoría de las categorías de imágenes.

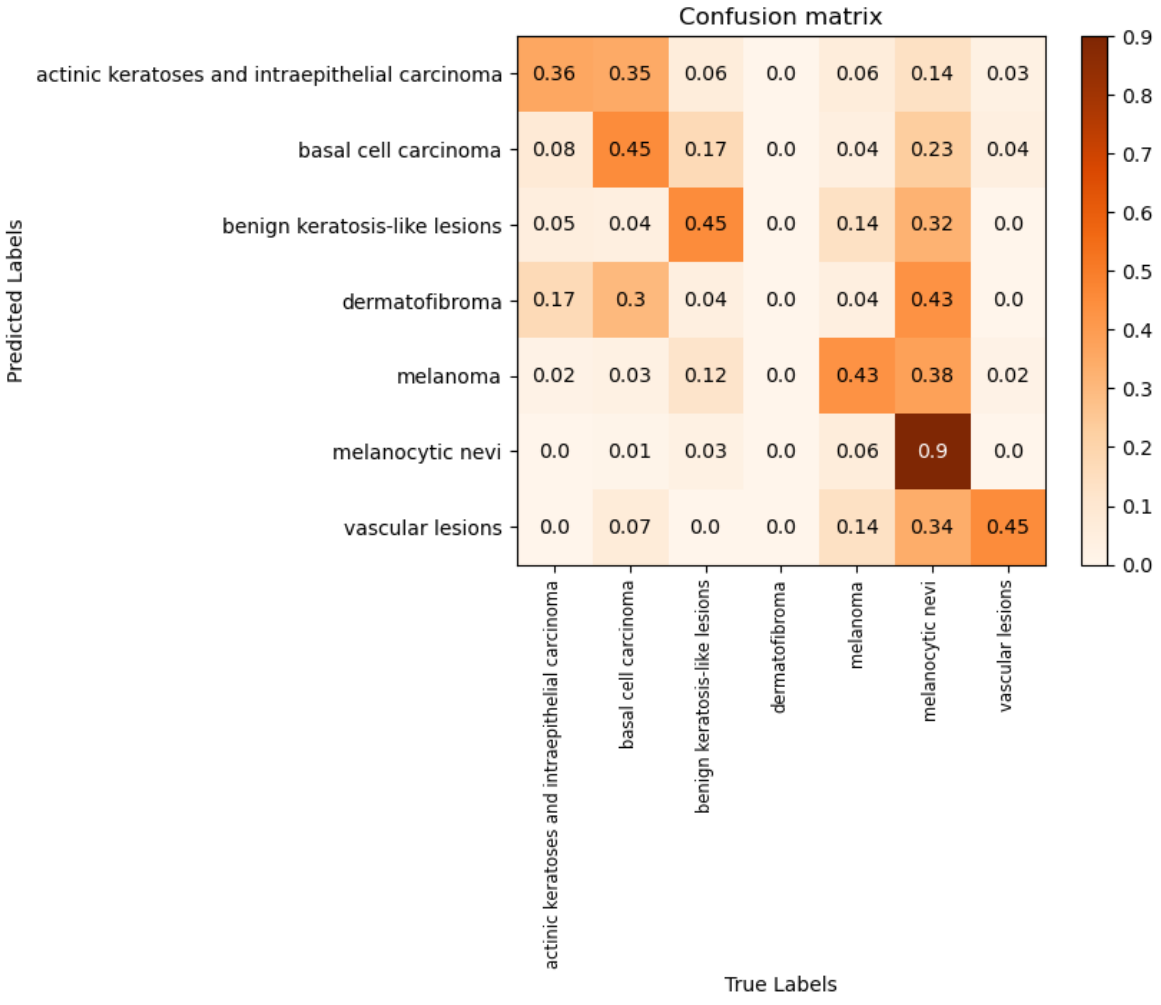


Figura 4.20: Imagen matriz de confusión para el modelo VGG sin muestreo ponderado

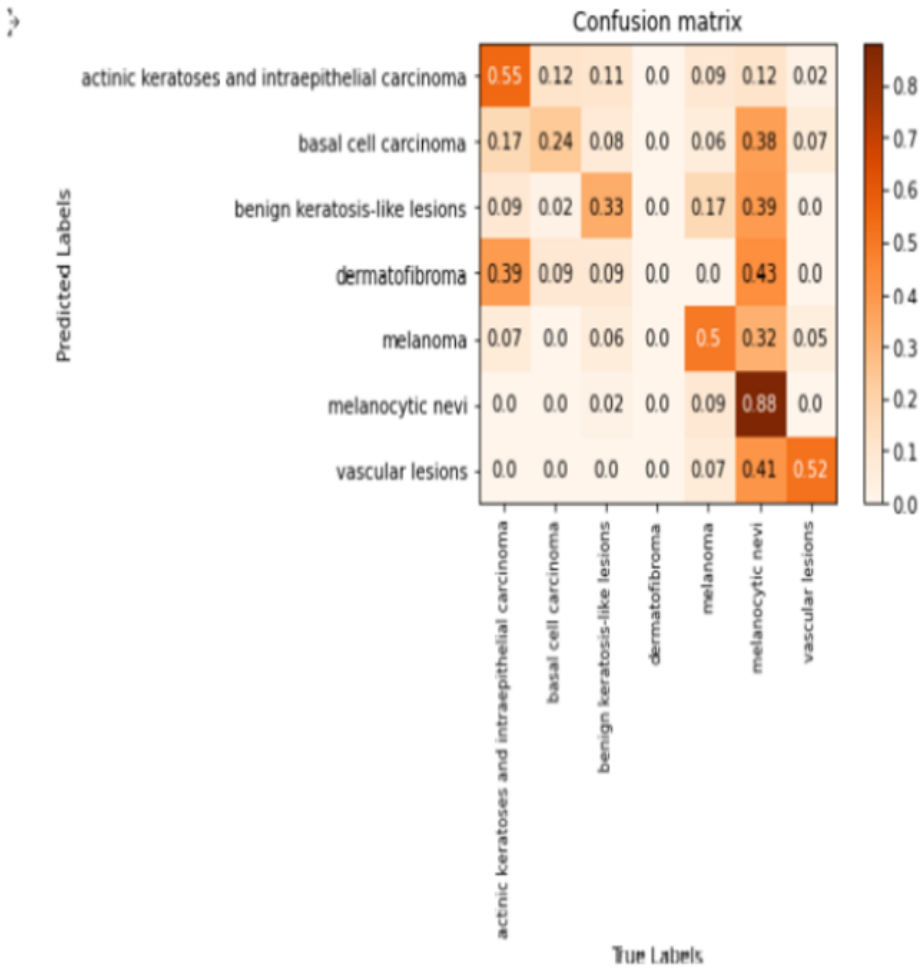


Figura 4.21: Imagen matriz de confusión para el modelo VGG sin muestreo ponderado de nuestro compañero

4.6.5. Modelo VGG con Cross-Entropy y muestreo ponderado

En este experimento se usan las configuraciones realizadas en el segundo experimento. Se utiliza la función de pérdida CE y se hace uso de la técnica muestreo ponderado que, como se ha comprobado ya, corrige perfectamente el problema del incorrecto balanceo del dataset de imágenes. Pero, con la diferencia de que se está realizando el experimento utilizando el modelo VGG16.

Se decide optimizar de nuevo los hiperparámetros *lr* y *momentum*, al igual que hicimos en el experimento anterior, el optimizador bayesiano nos ha devuelto como valores óptimo para los hiperparámetros del optimizador SGD los valores '0.0041' para el *lr* y '0.392' 'para el *momentum*. Examinando la Figura 4.22 generada por nuestro optimizador bayesiano, se ve claramente cómo el optimizador ha decidido explotar una única área de configuraciones. Sí es cierto que ha probado diferentes configuraciones para cuando el *momentum* tiene el valor 0, pero en su mayoría se centran en el intervalo de 0.2-0.6. Los resultados obtenidos por el optimizador son bastante diferentes a los obtenidos en el anterior experimento y a los obtenidos en los anteriores experimentos realizados con el modelo CNN básico.

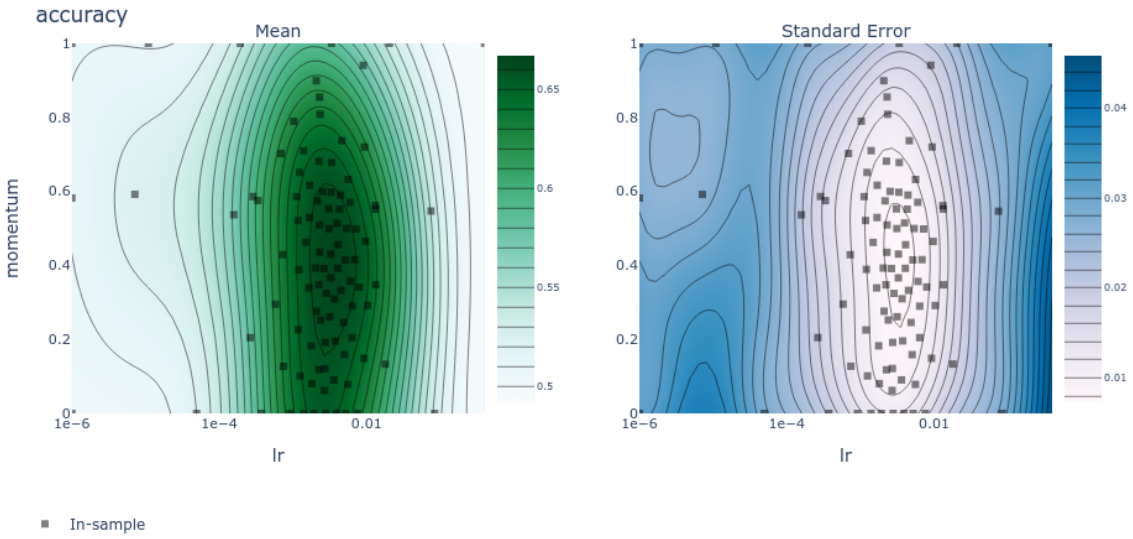


Figura 4.22: Distribución de los hiperparámetros *learning rate* (lr) y *momentum* para el optimizador SGD.

Comparando las métricas de precisión obtenidas en este experimento 4.8 con las del anterior 4.7 se ve una cierta mejoría en algunas de

ellas. Nuestro accuracy ha bajado considerablemente, pero las métricas como presicion, Recall o F-score han sufrido mejorías. Lo sorprendente en este caso es que se han obtenido peores resultados en las métricas de precisión, como puede ser el accuracy, con el modelo VGG que con el modelo CNN4.4 utilizado en el segundo experimento. Esto puede ser debido a que para este modelo fuese necesario más rondas con el optimizador bayesiano.

Tabla 4.8: *Metricas de precisión para el modelo VGG con funcion de pérdida cross entropy utilizando muestreo ponderado con accuracy general de 0.62*

Category name	Presicion	Recall	Specificity	F-score
actinic keratoses...	0.53	0.372	0.984	0.437
basal cell cercinoma	0.437	0.437	0.97	0.437
benign keratosis-like lesions	0.586	0.436	0.947	0.5
dermatofibroma	0.348	0.5	0.992	0.41
melanoma	0.776	0.293	0.965	0.425
Melanocytic nevi	0.624	0.962	0.556	0.757
Vascular lesions	0.724	0.6	0.996	0.656

Realizando una comparación sobre el accuracy obtenido con el del proyecto de Yuncai Chen [6], se ve que hemos obtenido incluso un peor accuracy que el suyo. No solo por el accuracy y las métricas de precisión, examinando las matrices de confusión 4.23,4.24 se ve claramente cómo la obtenida en este proyecto está dando peores resultados en la mayoría de las clases.

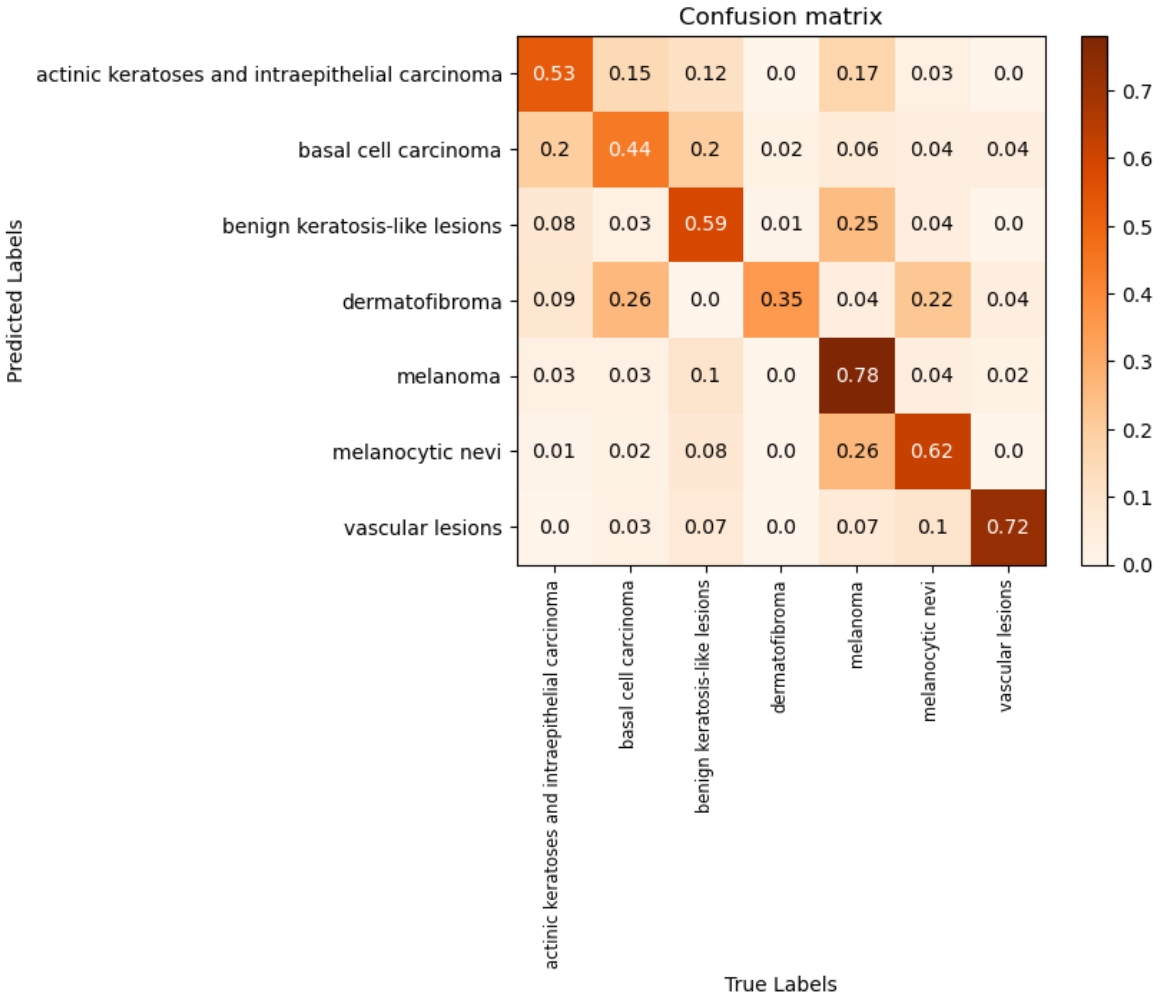


Figura 4.23: Imagen matriz de confusión para el modelo VGG con muestreo ponderado

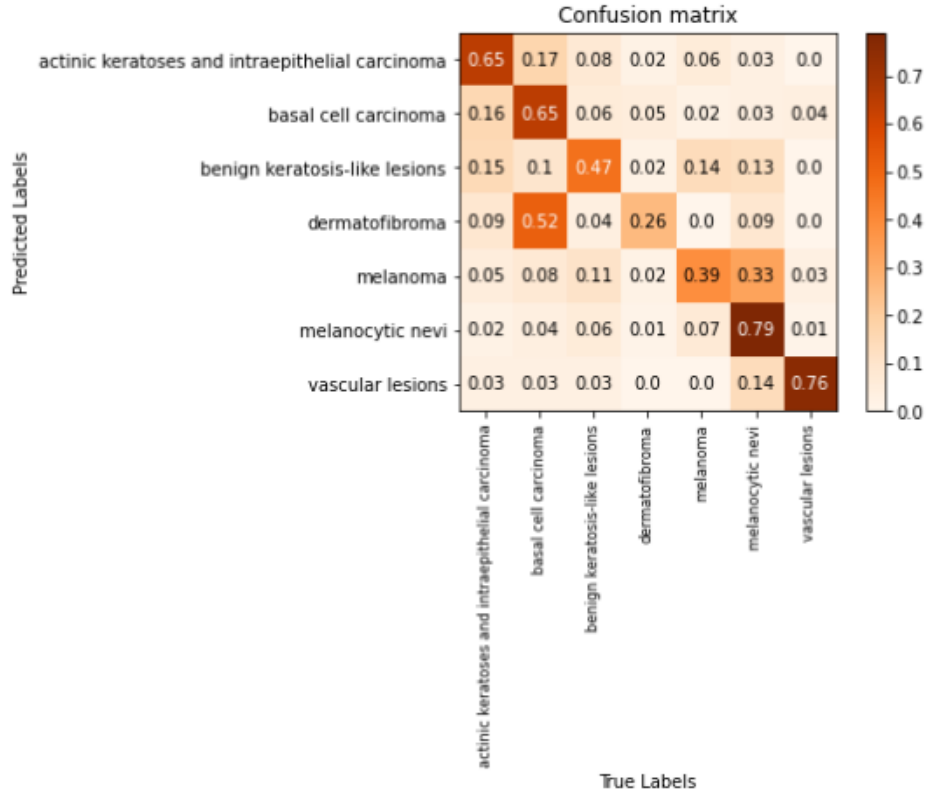


Figura 4.24: Imagen matriz de confusión para el modelo VGG con muestreo ponderado de nuestro compañero

4.6.6. Modelo VGG con Focal Loss

En este último experimento realizado para los problemas de clasificación, se toma la misma configuración utilizada en el tercer experimento realizado. Se hace uso de la función de pérdida Focal Loss y del modelo VGG16.

Al igual que en tercer experimento, se optimizan los hiperparámetros *learning rate* y *momentum* del optimizador SGD. También se optimizan las variables γ y α de la función de pérdida *focal loss*4.2 debido a su gran importancia en los resultados.

Los valores óptimos obtenidos por el optimizador bayesiano para este experimento son para el *lr* '0.0019', para el *momentum* '0.8835', para γ '2.0219' y para α '0.1728'. Analizando la gráfica 4.25, se pueden diferenciar dos áreas claras de exploración que ha tomado el optimizador. En estas dos áreas, el hiperparámetro *lr* no se ve gravemente alterado, mientras tanto el *momentum* está generando buenos resultados cuando su valor está en torno al 0 y cuando está en el intervalo 0.7-0.85. Aunque las mejores configuraciones estén en esta segunda área mencionada, cabe resaltar que en la mayoría de los experimentos están generando buenas configuraciones con el *momentum* a 0.

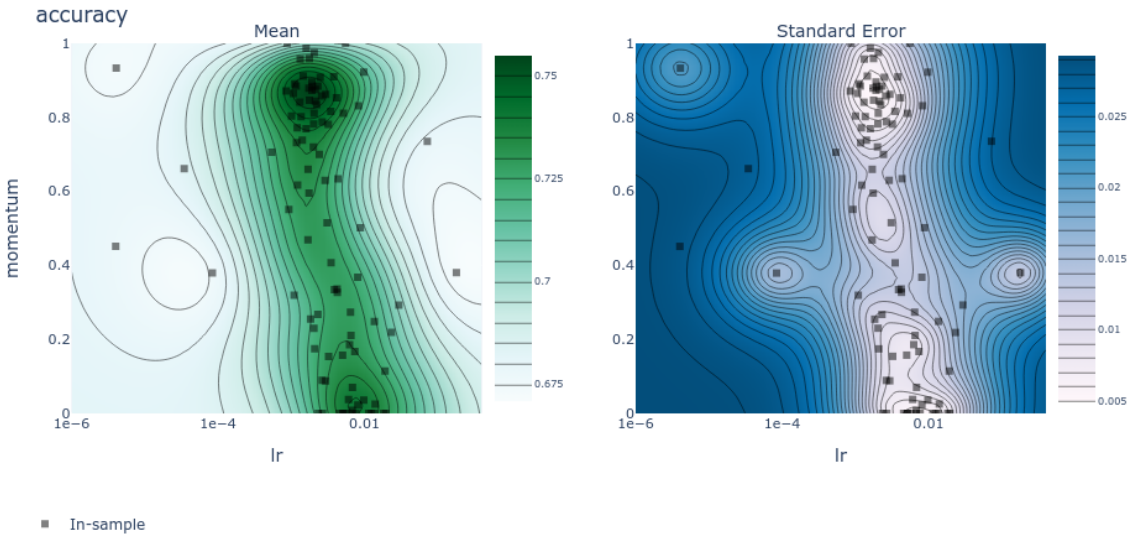


Figura 4.25: Distribución obtenida mediante el optimizador bayesiano. Parámetros *lr* y *momentum*

Se decide obtener una segunda gráfica, véase Figura 4.27, ésta tiene como referencia los parámetros alpha (α) y gamma (γ), que son las variables que se decidió optimizar de la función de pérdida. En este caso,

examinando la gráfica generada por el optimizador, solo se puede ver una única área de interés que ha decidido explorar nuestro optimizador bayesiano. En este caso, el optimizador bayesiano está obteniendo buenas configuraciones para valores relativamente bajos de α y γ. En comparación con el tercer experimento, para ambos parámetros se ha obtenido un valor menor. Si comparamos los valores óptimos obtenidos por el optimizador con los valores óptimos utilizado en el trabajo 'Focal Loss for Dense Object Detection' [56] vemos que son bastante parecidos. Los valores que han utilizado son son '0.25' para α y '2.0' para γ.

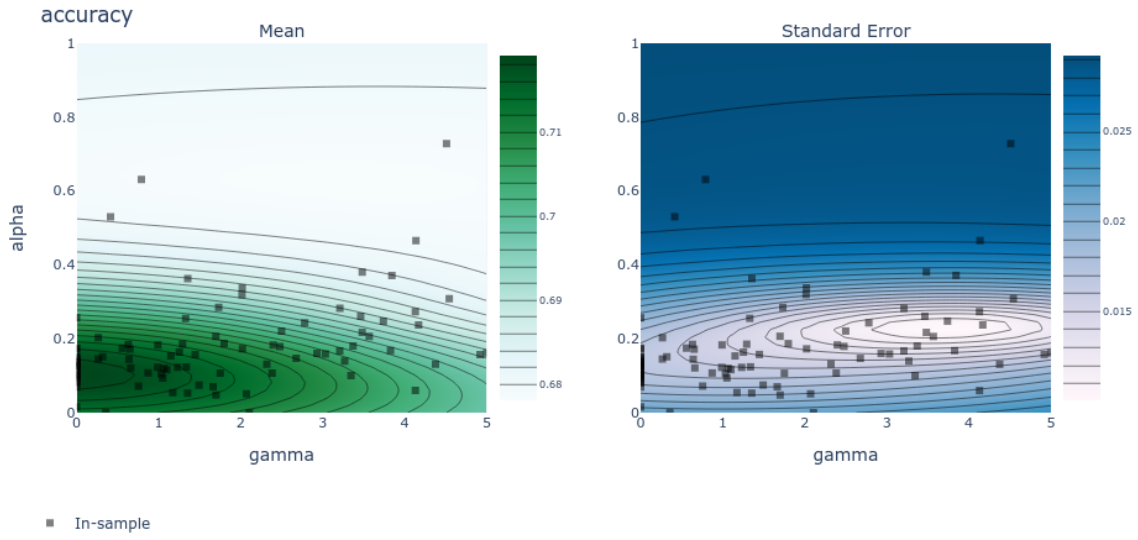


Figura 4.26: Distribución de los hiperparámetros 'gamma' (γ) y 'alpha' (α) para la función de pérdida Focal Loss

Analizando el accuracy obtenido, véase Tabla 4.9, ha mejorado ligeramente el accuracy del proyecto de Yuncai Chen [6]. A su vez, ha mejorado el resto de métricas de precisión del mismo proyecto. Cabe resaltar que aunque el accuracy esté dando valores bastante decentes, al igual que pasaba con el tercer experimento, esta función de pérdida no soluciona del todo el problema del incorrecto balance del dataset. Para analizar este experimento hay que analizar todas las métricas de precisión de la tabla 4.9. Si se compara con los resultados de las tablas 4.8 y 4.4 se ve claramente que los resultados de nuestras métricas son bastante bajas, dejando en evidencia que el accuracy no es una buena métrica a tener en cuenta para evaluar este experimento.

Aunque la evaluación de la métrica de precisión del accuracy no es de fiar para saber la eficiencia de este modelo, sí que se puede utilizar para analizar si hemos mejorado los resultados obtenidos en comparación con

Tabla 4.9: *Métricas de precisión para el modelo VGG con función de pérdida focal loss con accuracy general de 0.741.*

Category name	Presicion	Recall	Specificity	F-score
actinic keratoses	0.53	0.455	0.984	0.49
basal cell cercinoma	0.359	0.493	0.966	0.415
benign keratosis-like lesions	0.295	0.596	0.918	0.395
dermatofibroma	0.13	0.6	0.99	0.214
melanoma	0.291	0.485	0.916	0.364
Melanocytic nevi	0.943	0.805	0.823	0.869
Vascular lesions	0.621	0.514	0.994	0.562

otros proyectos. Comparando nuestro accuracy '0.741' con el del proyecto de Yuncai Chen [6] que es solo '0.6' se puede concluir que gracias a nuestro optimizador bayesiano hemos conseguido generar un modelo más eficiente que el de nuestro compañero. Todo esto también se ve reflejado en las matrices de confusión 4.27 y 4.28. Examinando los resultados de estas matrices de confusión se puede observar que aunque nuestro accuracy sea mejor, los resultados del proyecto de Yuncai Chen[6] son bastante buenos, clasificando algunas clases mejor que nuestro modelo.

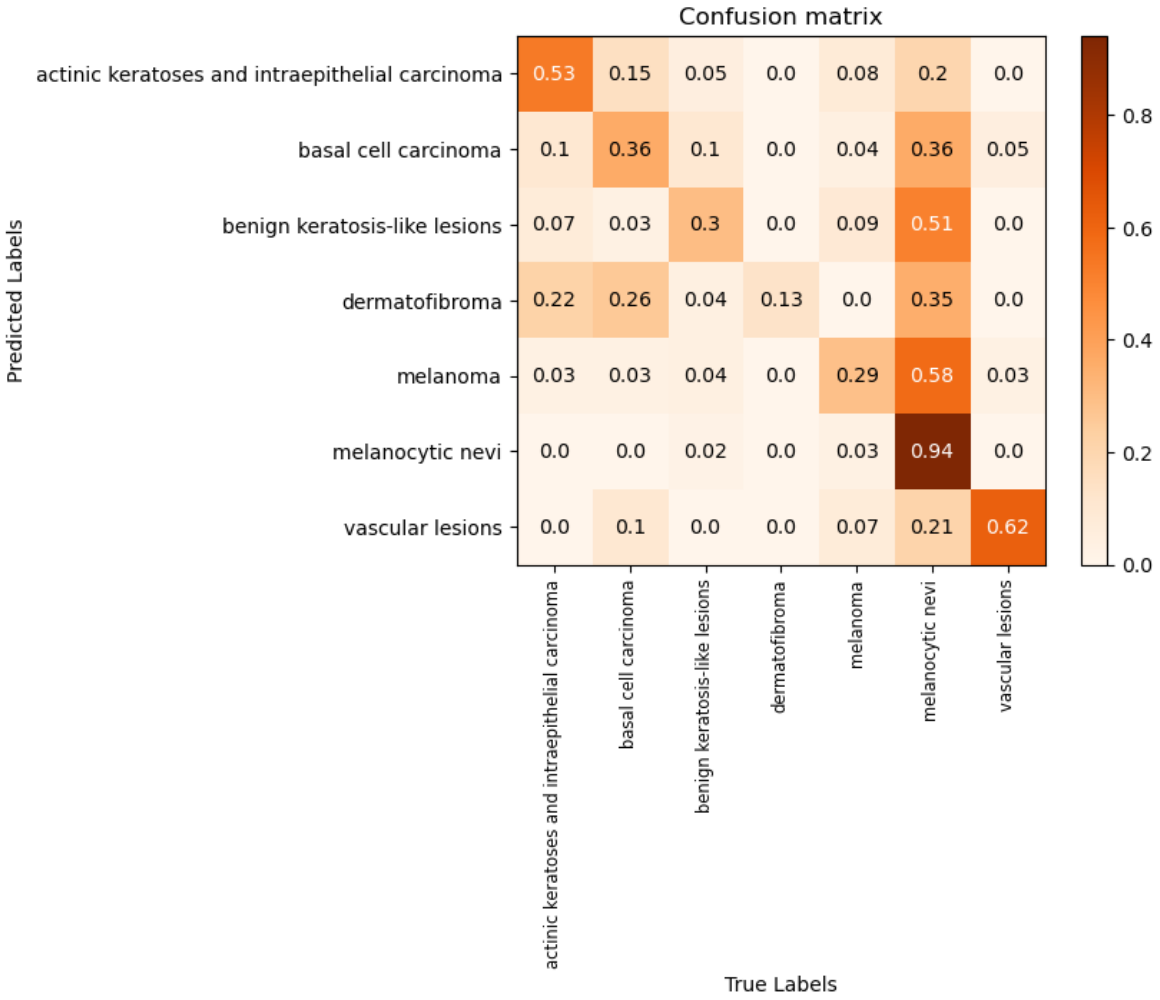
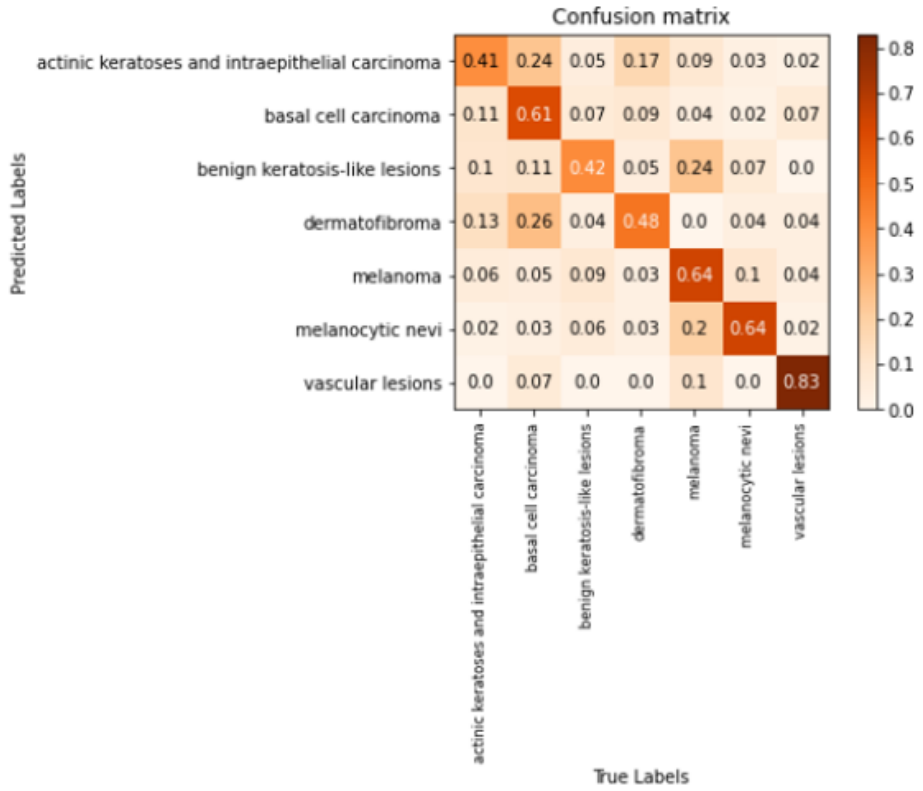


Figura 4.27: Imagen matriz de confusión para el modelo VGG con función de pérdida Focal Loss




```
The model accuracy is 0.6084788029925187
```

Figura 4.28: Imagen matriz de confusión para el modelo VGG con función de pérdida Focal Loss de nuestro compañero

5. Segmentación

Como se observó en la sección anterior, los problemas de clasificación tienen como objetivo obtener una etiqueta que clasifique a la imagen. En este caso, en los problemas de segmentación semántica, se clasifican los datos a nivel de píxel, es decir, obtener una etiqueta por píxel. De esta manera se clasifican los diferentes objetos que se pueden llegar a identificar en las imágenes.

Principalmente, la segmentación semántica consiste en darle una propiedad a cada píxel de una imagen. Se puede encontrar este tipo de problemas en numerosos ámbitos hoy en día. Desde un vehículo de conducción autónoma, que tiene que identificar diversos objetos hasta imágenes médicas para la detección temprana de cáncer, tumores, anomalías, etc.

Aunque abordamos la segmentación semántica para este tipo de problemas, los enfoques tradicionales se basaban en la detección de líneas, bordes o puntos. Sus resultados no eran tan satisfactorios como sí lo pueden ser utilizando la resolución a nivel de píxel.

5.1. Dataset

Para evaluar el método propuesto en un caso de segmentación, se ha elegido el dataset de imágenes LiTS (The Liver Tumor Segmentation Benchmark). Dicho dataset está orientado para la detección de cáncer de hígado en imágenes volumétricas (3D). Este dataset contiene una gran variedad de imágenes de diferentes tumores primarios y secundarios de diferentes tamaños. Estadísticamente hablando, el cáncer en el hígado está reconocido como el segundo cáncer más mortal. Normalmente, se mira el estado del hígado para ver si hay principio de tumores, ya que es un sitio bastante común para el desarrollo de diferentes tipos de tumores. Además, los diferentes cánceres existentes por la zona del abdomen, como puede ser el de páncreas o el de colon, tienden a extenderse con facilidad hacia el hígado. [57].

Este dataset de imágenes tiene un total de 131 imágenes en 3D.

Debido al coste computacional, al coste en tiempo y a la complejidad que supone utilizar arquitecturas convolucionales para imágenes en 3D, se decidió fraccionar las imágenes en 3D y así trabajar sobre las imágenes de 2D. Posteriormente, se llevó a cabo un análisis del dataset ya segmentado en imágenes 2D y se pudieron apreciar imágenes sin etiquetas de hígado que, realmente, no son relevantes para nuestro entrenamiento. Por lo tanto, se ha decidido utilizar solo aquellas imágenes que tuviesen hígado o tumor para el entrenamiento de nuestro modelo.

En la Figura 5.1 se puede ver una imagen *CT (Computed Tomography)* del dataset LiTS. Una tomografía computarizada normalmente se usa en el área de estudio de imágenes médicas[58] para la detección previa de cáncer, tumores o, en el caso de que el paciente ya esté diagnosticado, también se usa para ver el tamaño y forma del tumor. Estas imágenes muestran los huesos, órganos y tejidos blandos de manera más clara y precisa que otras tecnologías. En la figura 5.1 se puede apreciar en la parte izquierda de la imagen el hígado, mientras que en la parte derecha están otros órganos del cuerpo como el riñón, páncreas o estómago.

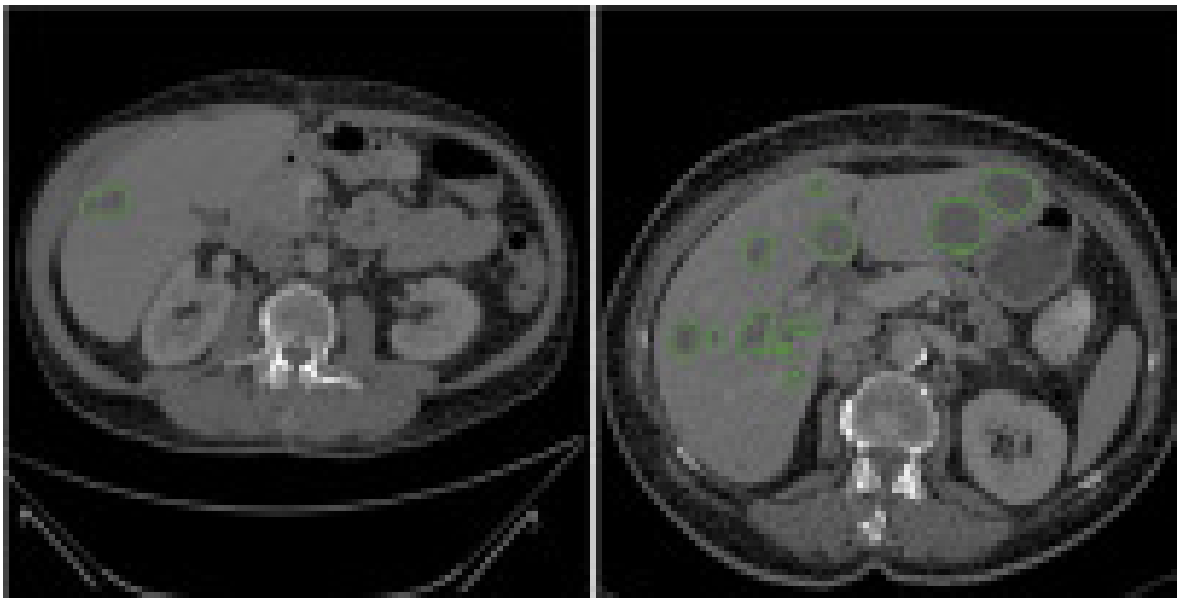


Figura 5.1: Higado

5.2. UNet

Para resolver los problemas de segmentación se han ido planteando diferentes modelos con el paso de los años, pero, el modelo más famoso es

el modelo de la UNet [59]. Este modelo fue presentado en una competencia de imágenes médicas donde solo disponían de 30 imágenes. El modelo fue creado para el mismo ámbito en el que nos movemos en este trabajo. Cabe resaltar que la obtención de imágenes médicas tiene un alto valor económico, por lo que no se suele entrenar las redes neuronales con una numerosa cantidad de imágenes. El desarrollo de esta red neuronal dio buenos resultados. Cabe resaltar que se desarrolló teniendo en cuenta que no iban a disponer de una gran cantidad de imágenes. Aunque este modelo se presentase para imágenes médicas, se ha probado que se puede utilizar para cualquier tipo de problema de segmentación.

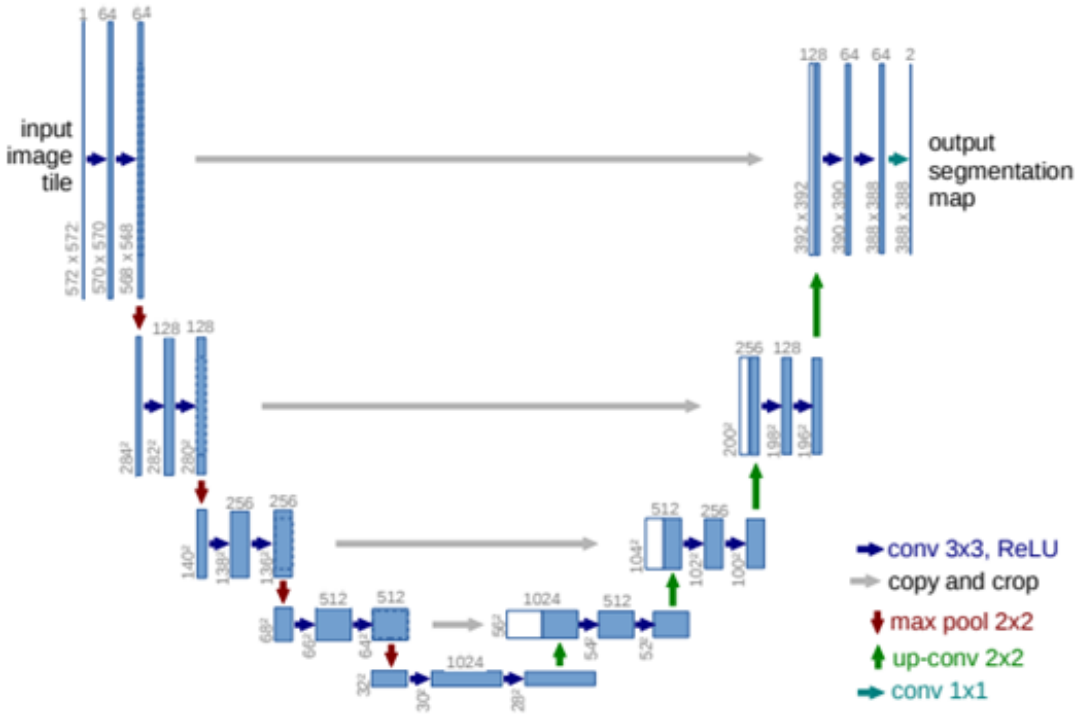


Figura 5.2: Arquitectura de una UNet.

La arquitectura del modelo presentado consta principalmente de dos partes importantes, véase Figura 5.2. La primera parte se llama codificador, la segunda parte se llama decodificador. La parte del codificador consta del sucesivo uso de las funciones de convolución 3×3 y sin padding. Esto va seguido del uso de la función de activación ReLU (Rectified Lineal Unit). Posteriormente con un maxpooling operation de 2×2. Cada vez que se utiliza esta operación se duplica el número de características. Las distintas operaciones de convoluciones han sido documentadas previamente en la Sección 4.2.1.

En cambio, en la segunda parte del problema se hace uso del up-

convolution que ha sido concatenado con su correspondiente mapa que se hizo en la parte del codificador para añadirle más características 5.2. Aparte, es necesario, ya que se están perdiendo píxeles en las funciones de convolución. Al contrario de lo que se hace anteriormente con el max-pooling, que duplicaba el número de canales, ahora se obtiene la mitad del número de canales. Es decir, hace lo contrario al maxpooling. Se sustituyen las operaciones de pooling con las de up-sampling para tener más resolución en las salidas, teniendo la localización más precisa de los píxeles. Seguido de funciones de convolución 3x3 con la función de activación ReLU. Esto se repite las veces que sean necesarias. El resultado final que se obtiene es la probabilidad que tiene cada píxel de pertenecer a una clase o a otra.

5.3. Experimentos a realizar

Para la evaluación de nuestro modelo lo haremos con tres diferentes funciones de pérdida. Por lo tanto, se utilizará la optimización bayesiana en los tres experimentos que se han realizado. El factor diferenciador entre los tres experimentos es la función de pérdida, o función objetivo de la etapa del entrenamiento y de la etapa de evaluación, que será modificada en función de dichos experimentos.

5.3.1. Primer experimento

Por parte del primer modelo utilizaremos la función de pérdida, Dice loss, que consiste en aplicar el coeficiente de Dice [60]. Esta función de pérdida es famosa por ser usada en problemas de segmentación, en concreto con problemas médicos. Esta función de pérdida intenta arreglar el desequilibrio que existen entre los datos, pero hay algunos casos en los que no lo consigue lograr. El coeficiente de Dice viene definido tal que:

$$Dice = \frac{2|X \cap Y|}{|X| + |Y|} = \frac{2TP}{2TP + FN + FP} \tag{5.1}$$

donde $|X|$ e $|Y|$ representan la cardinalidad de ambos conjuntos, es decir, el número de elementos en el conjunto.

5.3.2. Segundo experimento

Para este segundo modelo utilizaremos el índice de Jaccard [61]. El índice de Jaccard, también conocido como *Intersection Over Union* (IOU) es una de las funciones de pérdida más utilizadas. Es una función simple de calcular y comúnmente utilizada en los problemas de segmentación.

El cálculo del índice de Jaccard consiste en la cardinalidad de la intersección de dos regiones dividida por la cardinalidad de su unión, véase la ecuación 5.2. Si las dos regiones comparten exactamente los mismos datos, el valor obtenido será 1. Dado ese caso, en términos generales, para nuestro problema de segmentación significará que es correcto. En el caso contrario de que no compartan ningún dato, el valor obtenido será 0. Para nuestro problema práctico de segmentación, si el valor que nos devuelve es mayor que 0,5 lo daremos por válido, si es inferior a 0,5 significa que tendremos que hacer modificaciones en nuestra red.

$$IOU = \frac{|X \cap Y|}{|X \cup Y|} \tag{5.2}$$

5.3.3. Tercer experimento

En este tercer modelo utilizaremos las funciones de pérdida utilizadas anteriormente y el índice de Tversky [62]. Dicho coeficiente viene definido tal que:

$$TI = \frac{|X \cap Y|}{|X \cap Y| + \alpha|X|Y| + \beta|Y|X|} = \frac{TP}{TP + \alpha FN + \beta FP} \tag{5.3}$$

donde $|Y|X|$ representa el complemento relativo [63] de Y con respecto a X.

Con el índice de Tversky, al igual que con los dos modelos anteriores, se busca una solución al problema existente con el desequilibrio en las imágenes. El índice de Tversky es una generalización de la función de pérdida Dice y el índice de Jaccard. Se añaden dos parámetros nuevos al índice de Tversky. α y β. En el caso que $\alpha = \beta = 0{,}5$, la función de pérdida Tversky coincidirá con la ecuación del coeficiente Dice 5.1. Como restricción de ambos parámetros, la suma de estos dos elementos debe ser 1.

5.4. Evaluación

En el desarrollo de la resolución del problema de segmentación se han hecho tres experimentos diferentes. Lo que difiere entre ellos son las funciones de pérdida. Se han escogido tres funciones de pérdidas que son frecuentemente utilizadas en los problemas de segmentación de imágenes médicas. Se realizan los experimentos sobre los datos de LiTS (The Liver Tumor Segmentation Benchmark). Cabe resaltar que los valores obtenidos en el trabajo [6] son valores bastante bajos y optimizables. Lo esperado en estos experimentos es mejorar estos valores, teniendo en cuenta que mientras mayor sea los valores que obtenemos de las funciones de pérdida, mejor será el resultado.

En el tercer experimento de este problema de segmentación, en el entrenamiento de la red neuronal utilizaremos la función de Tversky, mientras que en la evaluación utilizaremos las dos funciones de pérdida ya utilizadas en los otros dos experimentos realizando la media entre ellas.

5.4.1. Experimento con Dice Loss

En este primer experimento se utiliza la función de pérdida Dice. Para la optimización de este experimento se ha decidido solo optimizar el Learning Rate (lr) y el *Dropout rate* (p) en las distintas capas dropout de la arquitectura.

Los valores de los hiperparámetros que se han obtenido del optimizador son $lr = 3{,}83 \cdot 10^{-4}$ y $p = 0{,}15$. Como se puede analizar en la gráfica de la Figura 5.3 se puede ver una única área de interés. El optimizador bayesiano se ha decantado, principalmente, por explotar los resultados con valores $lr \approx 10^{-4}$.

Se realiza un entrenamiento y evaluación con estos valores obtenidos del optimizador. Durante la etapa de entrenamiento con esta función de pérdida se localizan diferentes épocas donde el valor obtenido de la función de pérdida ronda, en su mayoría, en el intervalo del 0,6 al 0,95. Mientras que, finalmente, en la etapa de evaluación se acaba obteniendo '0.91'. Realizando una comparación de resultados con los obtenidos en el proyecto [6] se nota una ligera mejoría. En este proyecto se documenta que su valor obtenido para detectar tumores en el hígado es de '0.74'.

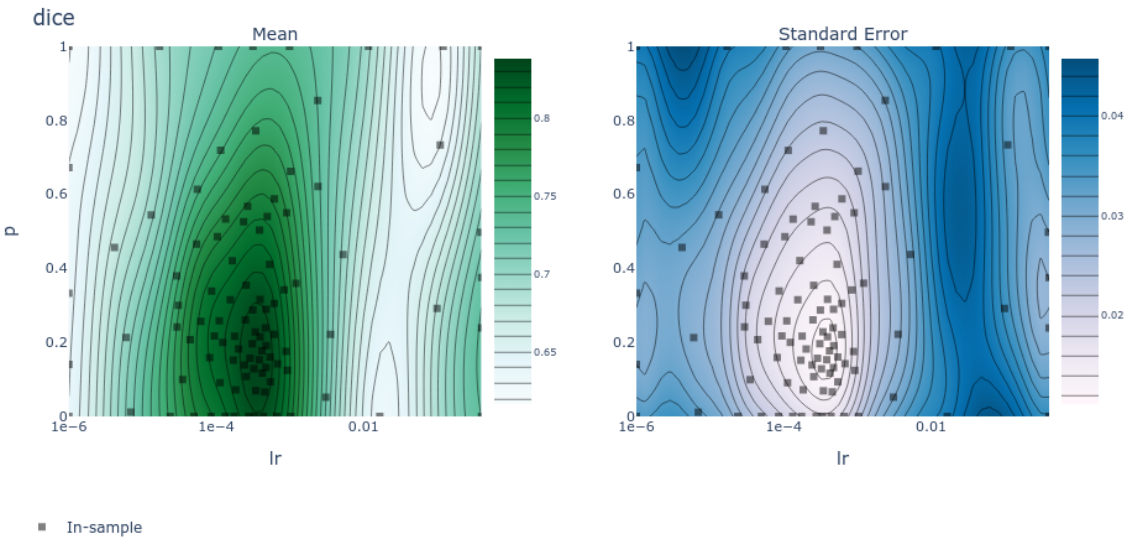


Figura 5.3: Distribución de los hiperparámetros *learning rate* (lr) y *Dropout*.

Por lo tanto, se concluye que el optimizador bayesiano ha mejorado los resultados para el problema de segmentación con esta función de pérdida.

Examinando el trabajo 'Training on polar image transformations improves biomedical image segmentation' [64], se puede utilizar sus resultados para realizar una comparación con los resultados obtenidos en nuestra etapa de optimización. Esto es debido a que se utiliza el mismo dataset de imágenes, LiTS, y arquitecturas de redes neuronales similares a la nuestra. Sus resultados usando la función de pérdida Dice son levemente mejores que los que hemos obtenido nosotros. Esto puede ser debido a que nuestro optimizador bayesiano necesitaba más rondas para lograr mejores resultados. Además, en el trabajo 'Training on polar image transformations improves biomedical image segmentation' [64] se utiliza arquitecturas para la segmentación de imágenes en 3D.

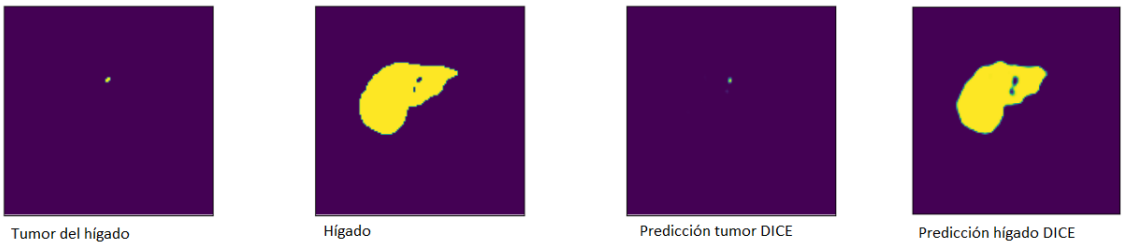


Figura 5.4: Evaluación de la primera imagen escogida con los resultados con la función de pérdida Dice

Como se puede observar en la Figura 5.4 se logra dibujar el hígado

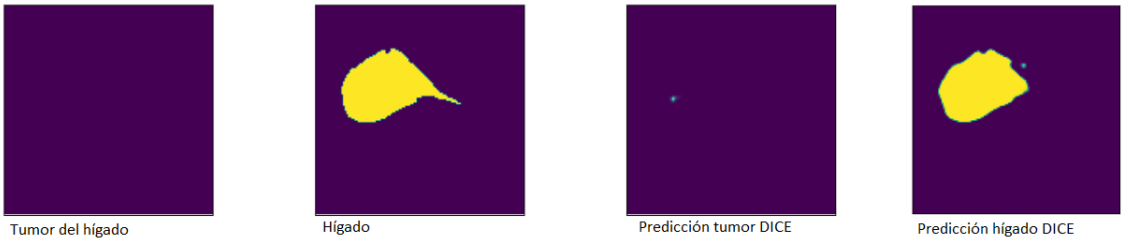


Figura 5.5: Evaluación de la segunda imagen escogida con los resultados con la función de pérdida Dice

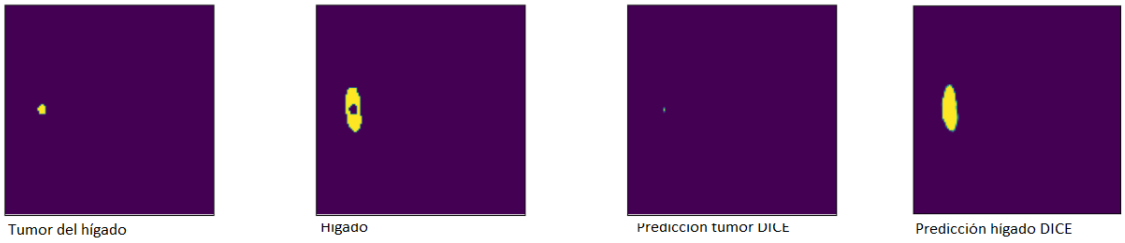


Figura 5.6: Evaluación de la tercera imagen escogida con los resultados con la función de pérdida Dice

sin dificultades. Además, también se consigue localizar el tumor del hígado para este caso en concreto. No en todas las muestras se ha conseguido segmentar correctamente el tumor del hígado, pero aún así se ha logrado obtener buenos resultados y localizar perfectamente en todos los casos el hígado. Se puede apreciar en las Figuras 5.5 y 5.6 que indiferentemente de la forma del hígado se consigue dibujarlo con cierta exactitud.

5.4.2. Experimento con Índice de Jaccard

En este segundo experimento haremos uso de la función de pérdida Índice de Jaccard (IOU). Esta función de pérdida es también frecuentemente utilizada para los problemas de segmentación de imágenes médicas. Al igual que en el caso anterior, procedemos a optimizar el **lr** y el p.

En este caso, los valores obtenidos para la configuración de hiperparámetros son ligeramente distintos a los del primer experimento con la función de pérdida dice. Como resultado, el optimizador de hiperparámetros propone como soluciones óptimas $lr = 1{,}75 \cdot 10^{-4}$ y $p = 0{,}37$. Examinando la Figura 5.7 que se ha generado con el optimizador, se pueden observar diferentes áreas de interés. Aunque sí es verdad que el optimizador bayesiano ha decidido concentrarse únicamente en una área.

Se puede ver en la gráfica cómo se han probado diferentes configuraciones de hiperparámetros en otras áreas de la misma, generando también buenos resultados.

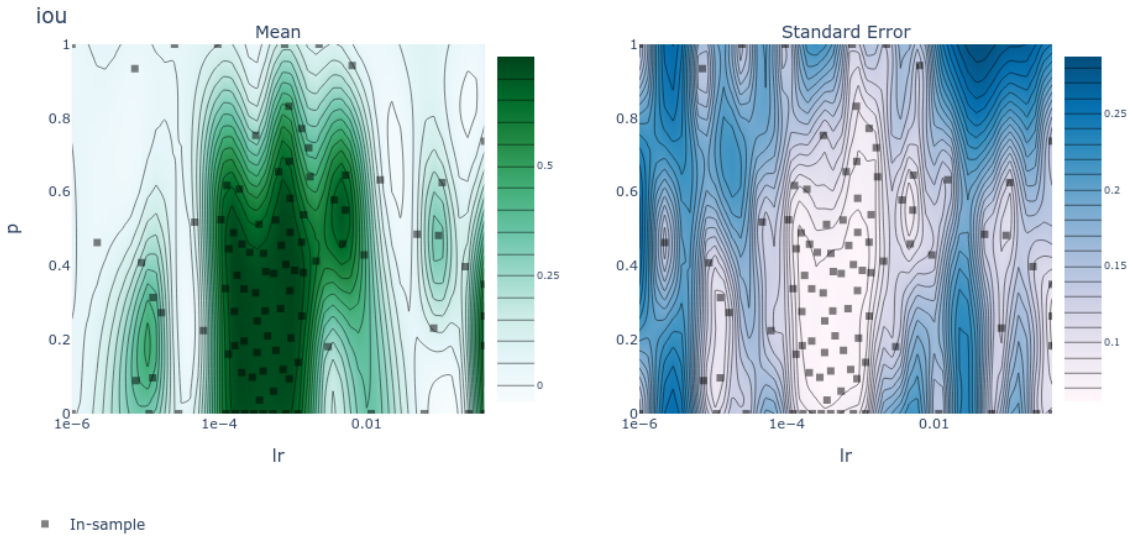


Figura 5.7: Distribución de los hiperparámetros *learning rate* (lr) y *Dropout*.

Al realizar la etapa de entrenamiento y evaluación se acaba obteniendo un valor de la función de pérdida levemente peor que el obtenido con la función de pérdida Dice. Durante la etapa de entrenamiento con la función de pérdida del Índice de Jaccard, se pueden ver muy buenos resultados que, en su mayoría, están en el intervalo de 0.6-0.85. Se obtuvo únicamente '0.8' como resultado de la función de pérdida en la etapa de evaluación; aunque no mejorase el resultado del primer experimento sigue siendo un buen valor. Examinando el proyecto de Yuncai Chen[6], se observa que los valores que los valores obtenidos con la función de pérdida Índice de Jaccard fue de '0.78'. Por lo tanto, se concluye que nuestro optimizador bayesiano no ha logrado obtener una gran diferencia en los resultados para el problema de segmentación con la función de pérdida del Índice de Jaccard.

En el paper 'Training on polar image transformations improves biomedical image segmentation' [64] en el que nos basamos para analizar los resultados del experimento anterior, también se hacen pruebas con esta misma función de pérdida. Sus resultados son levemente mejores que los nuestros, pero en este caso la diferencia es bastante insignificante como para decir que nuestro optimizador no ha obtenido buenos resultados. Aunque, al igual que con el anterior experimento, se espera que si au-

mentamos el número de rondas de nuestro optimizador se pueda llegar a mejorar los resultados.

Como se puede apreciar en las Figuras 5.8 y 5.10 se logra dibujar el hígado al igual que pasaba con el experimento anterior. Viendo las Figuras podemos observar que no se consigue localizar correctamente el hígado, además, no se han localizado muestras en las que con la función de pérdida del Índice de Jaccard consiga segmentar correctamente el tumor del hígado. Mientras que con el hígado se ha logrado localizar en diferentes muestras una buena precisión en el dibujo, incluso en muestras mas complejas como pueden ser las de las figuras 5.9 y 5.10.

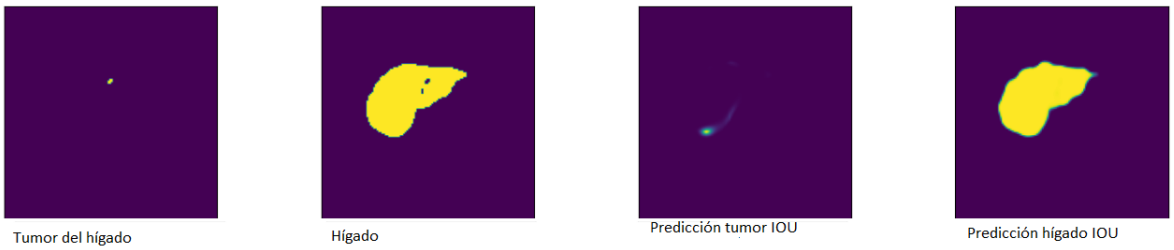


Figura 5.8: Evaluación de la primera imagen escogida con los resultados con la función de pérdida IOU

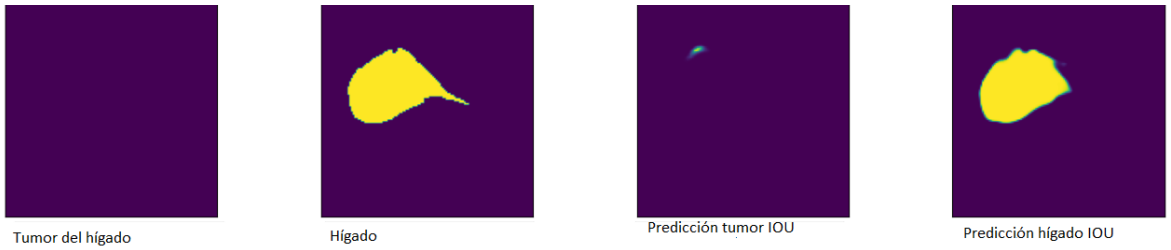


Figura 5.9: Evaluación de la segunda imagen escogida con los resultados con la función de pérdida IOU

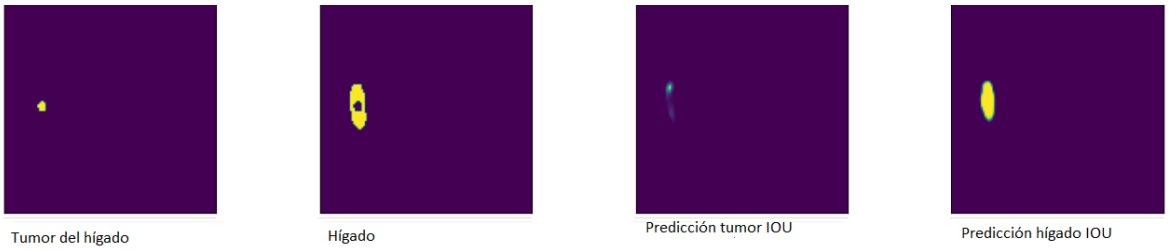


Figura 5.10: Evaluación de la tercera imagen escogida con los resultados con la función de pérdida IOU

5.4.3. Experimento con Tversky

En este tercer y último experimento que se realiza para el problema de segmentación, se hace uso de la función de pérdida Tversky para la etapa de entrenamiento de nuestra red neuronal. Mientras que para la etapa de evaluación se realiza la media ponderada de las funciones de pérdida Dice e Índice de Jaccard (IOU). En este caso, al igual que en los otros experimentos, se optimiza el lr y el p. A su vez, también se realiza una optimización del valor α de la función de pérdida Tversky. No se realiza optimización del valor β debido a que estos dos valores son complementarios, la suma de ambos debe dar 1, véase sección 5.3.3. Por lo tanto, se ha decidido optimizar el valor α y relizar el cálculo del valor β en el experimento a partir del valor α obtenido del optimizador, es decir, $\beta = 1 - \alpha$.

Para este caso, los valores obtenidos del optimizador son ligeramente diferentes al de los anteriores experimentos. Como resultado se obtiene que $lr = 7{,}7 \cdot 10^{-5}$, $p = 0{,}11$ y $\alpha = 0{,}32$ por lo que $\beta = 0{,}68$ en la función objetivo Tversky.

La Figura 5.11 nos muestra las diferentes configuraciones de hiperparámetros que se han ido escogiendo en las diferentes iteraciones realizadas, teniendo en cuenta el lr y el p para la elaboración de la misma gráfica. Examinando esta gráfica se observa que las diferentes configuraciones de estos hiperparámetros se centran en una única área de interés. El optimizador bayesiano ha decidido, en este caso, explotar una única área concreta.

Al realizar la etapa de entrenamiento y evaluación se acaba obteniendo para la función de pérdida el valor '0.85'. Este valor es levemente peor que el valor obtenido con la función de pérdida Dice y levemente mejor que el obtenido con el Índice de Jaccard. En la etapa de entrenamiento se observa un intervalo de valores entre 0.5-0.9. Son mejores que los obtenidos en la etapa de entrenamiento de la función de pérdida del Índice de Jaccard, pero no mejores que los obtenidos con Dice. El resultado obtenido con esta función de pérdida era lo esperado, teniendo en cuenta los resultados obtenidos en los anteriores experimentos. Analizando el proyecto *Deep learning aplicado al dataset medmnist* [6] se observa que el valor obtenido para la función de pérdida Tversky fue '0.77'. Por lo tanto, se concluye que hemos mejorado bastante sus resultados gracias a nuestro optimizador bayesiano.

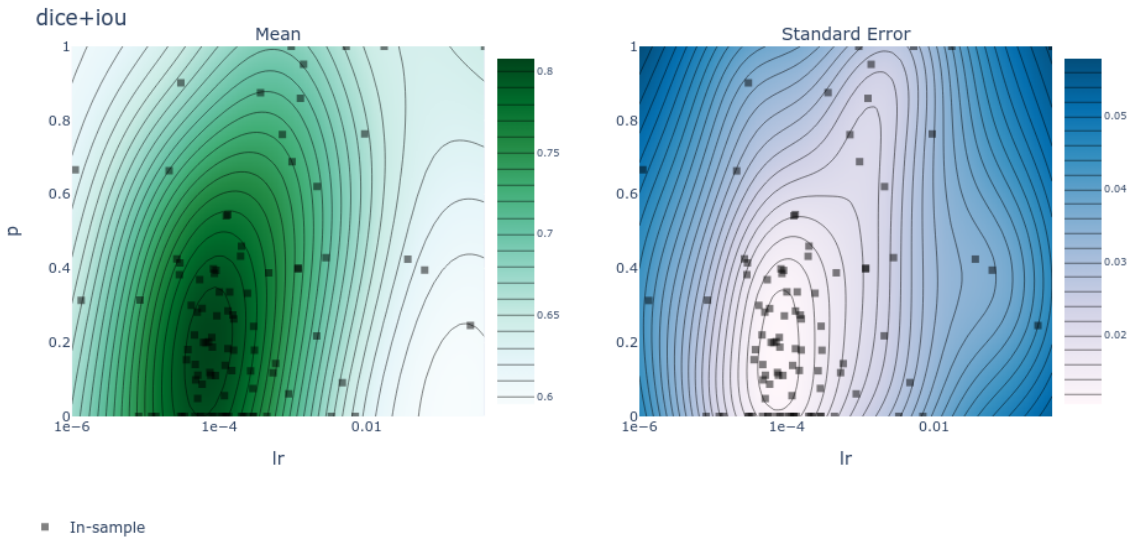


Figura 5.11: Distribución de los hiperparámetros *learning rate* (lr) y *Dropout*.

Además, analizando la Figura 5.12 se logra dibujar con facilidad el hígado, mientras que el tumor del hígado no lo consigue localizar del todo, al igual que pasaba con la función de pérdida del Índice de Jaccard del anterior experimento. Analizando las figuras 5.13 y 5.14 se puede apreciar que se logra dibujar el hígado sin complicaciones indiferentemente de la forma que tenga la muestra original.

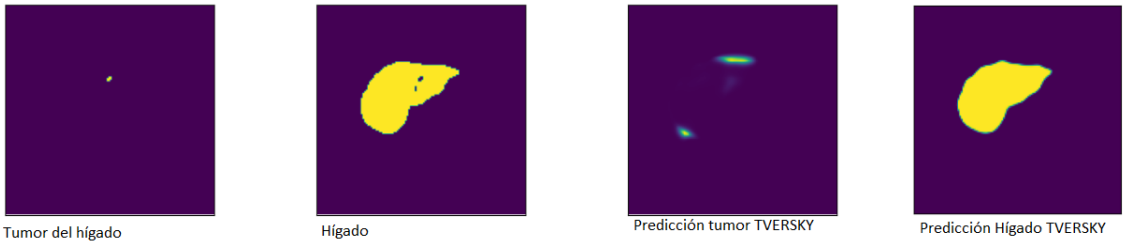


Figura 5.12: Evaluación de la primera imagen escogida con los resultados con la función de pérdida Tversky

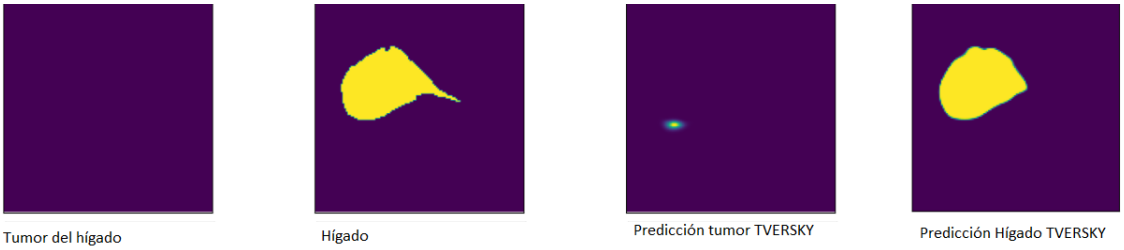


Figura 5.13: Evaluación de la segunda imagen escogida con los resultados con la función de pérdida Tversky

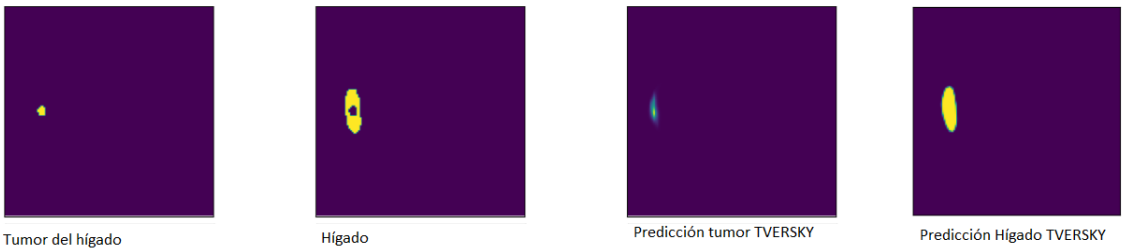


Figura 5.14: Evaluación de la tercera imagen escogida con los resultados con la función de pérdida Tversky

6. Conclusión

Este apartado recoge una breve conclusión teniendo en cuenta los resultados obtenidos, además se incluyen posibles extensiones del proyecto y la experiencia personal que he adquirido a lo largo del desarrollo.

6.1. Experiencia personal

Los conocimientos obtenidos en el desarrollo de este trabajo los considero bastante relevantes, ya que en un futuro tengo pensado ampliar mis estudios en los campos de ciencia de datos y de inteligencia artificial. Actualmente estoy trabajando como desarrollador de aplicaciones, pero en un futuro cercano tengo pensado ampliar mis estudios y al desarrollar este proyecto he podido confirmar mi interés en estos campos. A esto se une la adquisición de conocimientos que facilitará resolver problemas futuros.

Cabe resaltar que en este proyecto hemos estado utilizando metodologías ágiles en el desarrollo del mismo. Es cierto que estas metodologías se han ido utilizando en diferentes asignaturas con diferentes tipos de proyectos, pero aplicarlas de manera estricta en un caso real es muy diferente e importante. Desde un punto de vista laboral, el hecho de utilizar estas metodologías previamente ha sido realmente enriquecedor. En el mundo laboral dan por hecho que ya sabes aplicar los conceptos de las metodologías ágiles. Gracias a esta experiencia previa se me ha hecho más ameno el uso de las mismas en el mundo laboral.

6.2. Conclusiones del proyecto

Como se ha visto en el capítulo anterior, con el uso del optimizador bayesiano se ha logrado mejorar los resultados de otros proyectos que realizaban experimentos similares al nuestro. El optimizador bayesiano es considerado por muchos una de las mejores maneras de optimizar, no solo para los hiperparámetros, sino cualquier tipo de experimento.

Sí es cierto que optimizar por este método puede traer consecuencias transformadas en costos computacionales y económicos, pero, aun así, es totalmente recomendable debido a la gran diferencia que existe con los algoritmos tradicionales, como puede ser la búsqueda en cuadrícula o aleatoria.

El propósito principal de este proyecto se ha centrado en optimizar por el método de optimización bayesiana los hiperparámetros de la red neuronal, ya que tradicionalmente se han escogido los valores de manera aleatoria o con algún criterio específico hasta que se consiguiesen los resultados esperados. Además, se ha realizado una tarea de comparación de resultados con los de otros proyectos que nos ha facilitado tener referencias para saber si nuestro optimizador está dando buenos resultados.

Teniendo en cuenta los resultados obtenidos y la evaluación de los mismos se concluye que se han obtenido resultados satisfactorios. Es cierto que en el problema de clasificación no se ha obtenido mucha mejoría con los valores de los hiperparámetros obtenidos, sin embargo, en el problema de segmentación se han obtenido resultados que mejoran con bastante diferencia los de otros proyectos. Para esto solo hay que fijarse en los valores de las funciones de pérdida que se han obtenido. En el trabajo *Deep learning aplicado al dataset medmnist* [6], con la configuración que él utilizó, obtenía valores alrededor de '0.7' en sus funciones de pérdida, haciendo que no consiguiese detectar tumores en el hígado.

En el desarrollo de este proyecto, se ha tratado de optimizar modelos de inteligencia de artificial para problemas de segmentación y clasificación. Para estos casos, el principal problema se nos presentó con las herramientas de las que se disponía en un principio. En las primeras etapas del trabajo se utilizó Google colab. Esta plataforma no da malos resultados, incluso en su versión gratuita, pero para problemas de segmentación donde el requerimiento computacional era mayor hacía que aumentasen drásticamente los tiempos de los experimentos. Además, en esta plataforma para leer las imágenes en los experimentos de segmentación era necesario traerlas desde el Google Drive. Los tiempos de carga entre estas dos plataformas era demasiado altos, por lo que se decidió cambiar de dispositivo. En etapas posteriores se decidió utilizar un ordenador personal debido a que ofrecía mejores prestaciones computacionales como una mejor GPU y CPU. Aun así, surgieron problemas de tiempo para los experimentos de segmentación finales, donde el optimizador se configuraba para realizar un total de 100 rondas por cada uno de ellos. La ejecución final se realizó finalmente en otro dispositivo

aún más potente.

Las aportaciones que logramos en este proyecto se centran en un plano científico y técnico, empleando una de las técnicas más famosas de optimización de la actualidad y mostrando los resultados obtenidos.

6.3. Extensiones futuras

Los campos de la inteligencia artificial, *Machine Learning* y *Deep Learning* están avanzando a pasos agigantados en esta última década. Los casos de estudio de inteligencia artificial donde se ha aplicado el optimizador bayesiano eran modelos básicos con pocos parámetros a optimizar, por lo tanto, no se puede ver del todo el verdadero potencial del optimizador.

Lo ideal sería probar este optimizador con modelos más actuales, pero sí es cierto que si se trata de optimizar otros modelos con un mayor número de parámetros, el tiempo que necesita el optimizador para obtener unos resultados óptimos es mayor.

7. Bibliografía

[1] Mustafa Ergen et al. What is artificial intelligence? technical considerations and future perception. *Anatolian J. Cardiol*, 22(2):5–7, 2019.

[2] IBM. ¿qué son las redes neuronales? https://www.ibm.com/es-es/topics/neural-networks, 2023. Accedido: 31 de mayo de 2023.

[3] Andrés Abeliuk and Claudio Gutiérrez. Historia y evolución de la inteligencia artificial. *Bits de Ciencia*, (21):14–21, 2021.

[4] W Zamora-Cárdenas, M Zumbado, and Ignacio Trejos-Zelaya. Mcculloch-pitts artificial neuron and rosenblatt's perceptron: An abstract specification in z. *Technology Inside by CPIC*, 5:16–29, 2020.

[5] José María Cecilia Canales. *La gpu como procesador para computación novel: análisis y contribuciones*. PhD thesis, Universidad de Murcia, 2011.

[6] Yuncai Chen. Deep learning aplicado al dataset medmnist. *Universidad de Las Palmas de Gran Canaria*, 2023.

[7] Norio Nakata. Recent technical development of artificial intelligence for diagnostic medical imaging. *Japanese journal of radiology*, 37:103–108, 2019.

[8] Lasse Rouhiainen. Inteligencia artificial. *Madrid: Alienta Editorial*, 2018.

[9] Víctor Arias, Juan Salazar, Carlos Garicano, Julio Contreras, Gerardo Chacón, Maricarmen Chacín-González, Roberto Añez, Joselyn Rojas, and Valmore Bermúdez-Pirela. Una introducción a las aplicaciones de la inteligencia artificial en medicina: Aspectos históricos. *Revista Latinoamericana de Hipertensión*, 14(5):590–600, 2019.

[10] Pedro García. Inteligencia artificial. 2019.

[11] Francisco Baldinelli. Aplicaciones y límites actuales de la inteligencia artificial en el deporte. *Lecturas: Educación Física y Deportes*, 28(300), 2023.

[12] VV Markellos, W Black, and PE Moran. A grid search for families of periodic orbits in the restricted problem of three bodies. *Celestial mechanics*, 9:507–512, 1974.

[13] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[14] Daniel Alberto Montecino Montanares. Optimización de arquitecturas de redes neuronales convolucionales para el reconocimiento de patrones en imágenes mediante algoritmo genético de dos niveles. 2021.

[15] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[16] Universitat Oberta de Catalunya. Segmentación semánticas. https://blogs.uoc.edu/informatica/la-segmentacion-semantica-y-sus-benchmarks//, 2016. Accedido: 17 de mayo de 2023.

[17] Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694, 2020.

[18] Norbey Danilo Muñoz-Cañón and Jairo Andrés Romero-Triana. Optimización de los hiperparámetros de una máquina de regresión de soporte vectorial utilizando enjambre de partículas para el pronóstico de casos de covid-19. *Revista UIS ingenierías*, 20(2):181–196, 2021.

[19] Ajuste de hiperparámetros. https://learn.microsoft.com/es-es/azure/machine-learning/how-to-tune-hyperparameters, 2023. Accedido: 25 de mayo de 2023.

[20] Moisés Martínez. Optimizando tus hiper-parámetros: una visión teórica. https://www.paradigmadigital.com/dev/optimizando-hiper-parametros-una-perspectiva-teorica/, 2021. Accedido: 4 de Julio de 2023.

[21] Jorge Arranz de la Peñaa and Antonio Parra Truyol. Algoritmos genéticos. *Universidad Carlos III*, 2007.

[22] James Joyce. Bayes' theorem. 2003.

[23] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[24] Fernán Villa, Juan Velásquez, and Patricia Jaramillo. Conrprop: un algoritmo para la optimización de funciones no lineales con restricciones. *Revista Facultad de Ingeniería Universidad de Antioquia*, (50):188–194, 2009.

[25] Wikipedia. Teorema de bayes. https://es.wikipedia.org/wiki/Teorema_de_Bayes. Accedido: 17 de mayo de 2023.

[26] Lesley Ofelia Mesa Páez, Miller Rivera Lozano, and Jesús Andrés Romero Davila. Descripción general de la inferencia bayesiana y sus aplicaciones en los procesos de gestión. *La simulación al Servicio de la Academia*, 2:1–28, 2011.

[27] Universidad de Barcelona. Inferencia estadistica. http://www.ub.edu/aplica_infor/spss/cap4-1.htm. Accedido: 17 de mayo de 2023.

[28] Manuel Molina. ¿rioja o ribera? estadística frecuentista vs bayesiana. *Revista electrónica AnestesiaR*, 12(10):4, 2020.

[29] Google. Google optimize bayesian analysis. https://support.google.com/optimize/answer/9988285?hl=es. Accedido: 17 de mayo de 2023.

[30] J. Gothelf. *Lean UX : cómo aplicar los principios Lean a la mejora de la experiencia de usuario*. Unir emprende. Universidad Internacional de La Rioja S.A., 2014.

[31] Alonso Álvarez García y Rafael de las Heras del Dedo Carmen Lasa Gómez. *Métodos ágiles Scrum, Kanban, Lean*, volume 1. ANAYA, 2017.

[32] Overleaf. Overleaf, online latex editor. https://www.overleaf.com. Accedido: 4 de Julio de 2023.

[33] Google. Preguntas frecuentes. https://research.google.com/colaboratory/faq.html. Accedido: 4 de Julio de 2023.

[34] Anaconda. Anaconda. https://www.anaconda.com/. Accedido: 4 de Julio de 2023.

[35] META. Ax: Adaptive experimentation platform. https://github.com/facebook/Ax, 2019. Accedido: 17 de mayo de 2023.

[36] Ax api tutorial, https://ax.dev/docs/api.html. https://ax.dev/docs/api.html. Accedido: 17 de mayo de 2023.

[37] MedMNIST. Medmnist dataset. https://medmnist.com/. Accedido: 17 de mayo de 2023.

[38] Api de pytorch para iterar datasets , https://pytorch.org/docs/stable/data.html. https://pytorch.org/docs/stable/data.html. Accedido: 17 de mayo de 2023.

[39] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[40] Aditya Lohia, Kalyani Dhananjay Kadam, Rahul Raghvendra Joshi, and Anupkumar M Bongale. Bibliometric analysis of one-stage and two-stage object detection. *Libr. Philos. Pract*, 4910:34, 2021.

[41] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[42] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104:154–171, 2013.

[43] Wikipedia. Indicator function — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Indicator_function, 2023. Accedido: 22 de mayo de 2023.

[44] Eran Malach and Shai Shalev-Shwartz. Computational separation between convolutional and fully-connected networks. *arXiv preprint arXiv:2010.01369*, 2020.

[45] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[46] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[48] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[50] N Singh. Métricas de evaluación de modelos en el aprendizaje automático. https://www.datasource.ai/es/data-science-articles/metricas-de-evaluacion-de-modelos-en-el-aprendizaje-automatico, 2020. Accedido: 31 de mayo de 2023.

[51] Sofia Visa, Brian Ramsay, Anca L Ralescu, and Esther Van Der Knaap. Confusion matrix-based feature selection. *Maics*, 710(1):120–127, 2011.

[52] Nesime Tatbul, Tae Jun Lee, Stan Zdonik, Mejbah Alam, and Justin Gottschlich. Precision and recall for time series. *Advances in neural information processing systems*, 31, 2018.

[53] SB Data. Machine learning: Seleccion metricas de clasificacion. sitiobigdata. com, 2019.

[54] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[55] Gabriel Goh. Why momentum really works. *Distill*, 2017. Accedido: 17 de mayo de 2023.

[56] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[57] Patrick Bilic, Patrick Christ, Hongwei Bran Li, Eugene Vorontsov, Avi Ben-Cohen, Georgios Kaissis, Adi Szeskin, Colin Jacobs, Gabriel Efrain Humpire Mamani, Gabriel Chartrand, et al. The liver tumor segmentation benchmark (lits). *Medical Image Analysis*, 84:102680, 2023.

[58] Sociedad Americana Contra El Cáncer. Tomografía por computadora y el cáncer. https://www.cancer.org/es/cancer/diagnostico-y-etapa-del-cancer/pruebas/estudios-por-imagenes/tomografia-por-computadora-y-el-cancer.html. Accedido: 17 de mayo de 2023.

[59] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[60] Reuben R Shamir, Yuval Duchin, Jinyoung Kim, Guillermo Sapiro, and Noam Harel. Continuous dice coefficient: a method for evaluating probabilistic segmentations. *arXiv preprint arXiv:1906.11031*, 2019.

[61] Jeroen Bertels, Tom Eelbode, Maxim Berman, Dirk Vandermeulen, Frederik Maes, Raf Bisschops, and Matthew B Blaschko. Optimizing the dice score and jaccard index for medical image segmentation: Theory and practice. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part II 22*, pages 92–100. Springer, 2019.

[62] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. Tversky loss function for image segmentation using 3d fully convolutional deep networks. In *Machine Learning in Medical Imaging: 8th International Workshop, MLMI 2017, Held in Conjunction with MICCAI 2017, Quebec City, QC, Canada, September 10, 2017, Proceedings 8*, pages 379–387. Springer, 2017.

[63] Wikipedia. Complement (set theory) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Complement_(set_theory)#Relative_complement, 2023. Accedido: 11 de junio de 2023.

[64] Marin Benčević, Irena Galić, Marija Habijan, and Danilo Babin. Training on polar image transformations improves biomedical image segmentation. *IEEE Access*, 9:133365–133375, 2021.