# Índice general

# Índice de figuras

# Índice de tablas

# Parte I

# Introducción

# Capítulo 1

# Introducción

El término *Artificial Inteligent*(Inteligencia Artificial) surgió en los años 50 cuando Alan Mathinson Turing, un matemático y científico de la computación nacido en Reino Unido, con su artículo *Computing Machinery and Intelligence* [1] introdujo una serie de juegos de los que, una máquina no lógica, podría llegar a aprender. Desde ese preciso momento, han salido nuevas tecnologías capaces de simular el comportamiento humano e incluso, algunas, su razonamiento.

Nuevos términos han surgido a partir de esta idea, tales como *Machine Learning* [2] o *Deep Learning* [3]. Estos términos hacen referencia a las técnicas utilizadas para que una máquina sea capaz de resolver o aprender un problema complejo, de tal manera que el ser humano no tenga que perder su tiempo a la hora de realizar esa tarea. Estas dos técnicas son muy parecidas, aunque su principal diferencia es que el término *Deep Learning* o *Aprendizaje profundo*, eleva el nivel de aprendizaje a uno más detallado que el usado en *Machine Learning*.

Con la aparición de este tipo de técnicas, el ser humano se ha dado cuenta de que él mismo tiene fallos, sobre todo a la hora de analizar una tarea con su propia visión. Es por esto que mucho científicos, matemáticos y estadísticos, se han unido a otros profesionales de distintos gremios para ayudarles con sus tareas, tales como poder realizar una clasificación de imágenes según su contenido, o con el pilotaje automático de automóviles.

Este tipo de tareas son las de menor relevancia, ya que, uno de los gremios más importantes que se han unido con estos expertos en inteligencia artificial son los médicos. El gremio de la medicina siempre ha sido uno de los más importantes en la sociedad, es por ello que han visto obligados utilizar sis-

temas informáticos para poder así realizar, con muchísima mayor precisión, muchas de las tareas que realizan.

Según *Amerian Cancer Society* [4], para el año 2019, sólo en los Estados Unidos, aproximadamente 96,480 nuevos casos de melanoma serán diagnosticados, de los cuales, aproximadamente 7,230 personas morirán por causa de melanoma [5]. Es por esto por lo que muchos médicos y científicos se han unido para poder detectar esta enfermedad antes de que ésta aumente, ya que esta enfermedad puede dividirse en cuatro etapas, desde la 0, donde todavía no se ha propagado el melanoma, hasta la IV(4), donde el melanoma se ha propagado por una mayor zona [6].

A la hora de poder detectar la aparición de melanomas en la piel, esta asociación nos indican los distintos factores a tener en cuenta. Esta asociación hace uso de las primeras letras del alfabeto, siendo A por asimetría, B por el borde de la lesión, C por el color, D por el diámetro y E por la evolución de la lesión. Por ejemplo, en el caso de tener una lesión en la piel en la que su forma y borde no son uniformes, el color de ésta es oscuro, su diámetro es mayor de 6mm y vemos que durante un tiempo su forma ha cambiado y que tenemos nuevos síntomas, entonces, según estas reglas, estaríamos ante un tipo de lesión maligna, es decir, ante un melanoma.

Muchos científicos, expertos en la inteligencia artificial, se han dado cuenta de que, utilizando esos factores, se podría construir una red neuronal en la que, introduciendo estos factores, la red neuronal nos devuelva un porcentaje representando la posibilidad de ser un melanoma o un tipo de lesión benigna (no cancerígena).

Los autores del artículo *Automated Melanoma Recognition in Dermoscopy Images via Very Deep Residual Networks* [7] se dieron cuenta de la posibilidad de crear una red neuronal que hiciera la tarea de clasificación por nosotros, o en un caso extremo, para certificar la clasificación predicha por el médico experto. En este artículo, los autores utilizan una red neuronal residual completa [8], en la que utilizan la forma y el color de la lesión para poder predecir la etiqueta de la lesión.

De igual forma, este proyecto ha sido creado de manera alternativa al expuesto en el *Artículo* [7]. La diferencia con éste es el framework utilizado para la creación de la red neuronal residual profunda. En el caso del *Artículo* [7], el framework utilizado por los autores es *Caffe* [9]. Este framework es utilizado para la creación de redes neuronales basadas en el *Deep Learning*.

En cambio, para este proyecto hemos utilizado la API de *Keras* [10], junto con la biblioteca *Tensorflow* [11], utilizando el lenguaje de programación *Python* [12]. La utilización de este lenguaje y bibliotecas es debida a su sencillez a la hora de realizar una red neuronal profunda. Al igual que en el *Artículo* [7], el modelo utilizado es una red residual profunda, ya que así evitaremos tener problemas a la hora del entrenamiento.

# Capítulo 2

# Definición del problema

En este capítulo se procederá a dar una explicación concisa del problema a solventar. Para ello, este capítulo estará dividido en dos partes. La primera parte, el problema real, explicará, desde el punto de vista del mundo real, cuál es el problema que vamos a abordar en este proyecto. La segunda consistirá en explicar el problema desde un punto de vista técnico, es decir, desde el punto de vista de un ingeniero. Para la comprensión de ambas definiciones, se recomienda la previa lectura del Capítulo 4.

## 2.1. Definición del problema real

En este *Trabajo de Fin de Grado* se abordará el problema real de la clasificación de tipos de lesiones en la piel. Además del problema comentado, existe otro que consiste en el ajuste de una red neuronal residual profunda para realizar lo mejor posible esta clasificación.

El problema de clasificación consiste en clasificar el tipo de lesión en dos tipos, de tipo benigno o de tipo maligno. Para ello, nuestro modelo se fijará en la forma y el color de la lesión. Este problema resulta bastante complicado cuando las imágenes utilizadas para el entrenamiento presentan alteraciones, como son el vello o sombras en la imagen.

Las imágenes que se utilizarán para la clasificación se pueden encontrar en la base de datos pública *ISIC* [13]. Esta base de datos es proporcionada por La Colaboración Internacional de Imágenes de Piel (*The International Skin Imaging Collaboration*, la cual es una asociación entre la academia y la industria diseñada para facilitar la aplicación de imágenes digitales de la piel con el objetivo de reducir la mortalidad por melanoma.

Con respecto al problema del ajuste de la red, tendremos en cuenta tanto los hiper-parámetros que presenta nuestra red, como el número de capas necesarias para reducir el error de clasificación lo máximo posible. Estos hiper-parámetros serán los siguientes:

- **Learning rate:** También llamado *ratio de aprendizaje*, consiste en el salto que realiza nuestra red entre épocas. Cuanto mayor sea el valor de éste, más probabilidad habrá de no alcanzar el óptimo de la función. En cambio, si el coste de éste es muy pequeño, el valor computacional es mayor al tener que realizar más épocas para encontrar ese óptimo.

- **Batch size:** Es el *tamaño del lote*, que quiere decir el número de imágenes que vamos a considerar para entrenar en cada una de las épocas. A mayor número de muestras, mayor nivel computacional.

- **Epochs:** Consiste en el número de iteraciones que son necesarias para que nuestra red neuronal profunda sea capaz de aprender correctamente la diferencia entre los dos tipos de lesiones. El problema de que nuestra red realice muchas iteraciones es que podamos caer en el problema de *sobre-aprendizaje*, que quiere decir que nuestra red no es capaz de generalizar de manera adecuada.

- **Patience:** Es la variable que indicará el número de épocas que nuestro modelo aguantará entrenando sin que el error de validación mejore (es decir, disminuya).

## 2.2. Definición del problema técnico

Tras definir el problema real de este trabajo, procederemos a explicar el problema técnico, es decir, como su nombre indica, explicaremos el problema a solventar en este trabajo desde un punto de vista técnico, para un que una persona experta pueda comprender mejor el problema abordado. Para ello, vamos a describir este problema basándonos en la metodología PDS (*Product Design Specification*).

### 2.2.1. Funcionamiento

El funcionamiento de este proyecto será el de de una herramienta software que nos permitirá crear una red neuronal residual profunda. Además, dentro de esta herramienta software, incluiremos una serie de funciones que

nos permitirán pre-procesar la imagen para poder así realizar un mejor entrenamiento en nuestro modelo.

El proyecto realizado nos permitirá realizar un entrenamiento de la red en el cual se guardarán los resultados de manera automática en cada época en un fichero con formato específico. Esto se hace para poder, después de la finalización del entrenamiento, crear una serie de gráficas que nos permitan obtener, de manera visual, el comportamiento de nuestra red durante el entrenamiento.

Además, el programa creado nos permitirá realizar una validación que se hará periódicamente, en nuestro caso, cada vez que finalice una época del entrenamiento. Junto con todo esto, este proyecto permitirá al usuario realizar una predicción de una imagen concreta para poder comprobar si el entrenamiento ha sido satisfactorio.

Finalmente, el programa nos generará una matriz de confusión y una gráfica con los resultados de los valores de error o pérdidas tanto en entrenamiento, como en test, para poder apreciar el comportamiento de nuestra red. Éstas se guardarán en imágenes con la extensión *.jpg* para que el usuario pueda abrirlas en cualquier dispositivo.

### 2.2.2. Entorno

El entorno de programación para este *Trabajo de Fin de Grado* es el entorno utilizado por el lenguaje de programación orientado a objetos *Python* [12]. Este lenguaje de programación nos permite ejercutar nuestro programa en cualquier sistema operativo, ya esté basado en *UNIX*(Linux, Mac OS) o en *MS-DOS*(Windows). Para este proyecto hemos utilizado sistemas basados en *UNIX*, los cuales son *Debian* por parte del servidor y *Mac OS* para la codificación del mismo.

Con respecto a la implementación del modelo neural, hemos utilizado una biblioteca adicional de *Python* llamada *Keras* [10], la cual utiliza como soporte la biblioteca *Tensorflow* [11]. Estas dos bibliotecas están dedicadas especialmente para el diseño y entrenamiento de redes de aprendizaje profundo. Con respecto a las gráficas realizadas para el seguimiento del comportamiento de la red, hemos utilizado una biblioteca de *Python* llamada *Tensorboard* [14], la cuál, al igual que la biblioteca *Keras*, está soportada gracias a *Tensorflow*.

Este proyecto está dirigido a un público experto, es decir, personal que sepa sobre aprendizaje profundo junto con unos conocimientos mínimos de cómo configurar los parámetros de una red para el mejor comportamiento de la misma. Además de estos conocimientos, el usuario de este proyecto debe de tener unos conocimientos sobre cómo ejecutar un programa a través de la línea de comandos, ya que este proyecto tiene la necesidad de tener que ser ejecutado de esa manera.

### 2.2.3. Vida esperada

Con respecto a la vida esperada de este *Trabajo de Fin de Grado* es muy difícil de predecir, ya que al tratarse de un trabajo de investigación, todo vendrá dado por el avance tecnológico de la materia. Como ya sabemos, el mundo de la informática, pero sobre todo el área de la inteligencia artificial, está en continuo cambio, con lo que hoy podría considerarse como óptimo, dentro de unos meses podría darse el caso de que se creara un nuevo software que optimizase el utilizado en este proyecto.

Además de ser un proyecto relacionado con la inteligencia artificial, es un proyecto que está relacionado con otro mundo que también está en continuo crecimiento, la medicina.

Con lo que, por lo dicho hasta ahora, podríamos decir que la vida esperada para este proyecto es más bien corta.

### 2.2.4. Ciclo de mantenimiento

Una vez hemos definido la vida esperada de este proyecto, se puede apreciar que el ciclo de mantenimiento para este proyecto también será corto. Esto quiere decir que lo más probable sea que en un futuro tengamos que hacer algunos cambios en el software al tener nuevas herramientas para poder crear redes profundas. Además, este trabajo puede servir a futuros alumnos para poder implementar este modelo de red neuronal, pero con otras herramientas, o para resolver otro tipo de problema similar.

En conclusión, al tratarse de un trabajo de investigación, conlleva que la planificación del mantenimiento del mismo sea muy complejo.

### 2.2.5. Competencia

Con respecto a la competencia que podemos encontrar para este proyecto, es muy amplia, ya que, como hemos comentado anteriormente, se trata de un proyecto el cual incluye tanto un papel informático, como un papel médico. Además, el hecho de que las enfermedades en la piel está en aumento, hace que este tipo de proyectos sean muy comunes. Por ejemplo, tenemos el trabajo realizado por Àdria Romero-López [15], en el cual se realiza el mismo estudio que en este proyecto pero utilizando un tipo de red *CNN* (*Convolutional Neural Network*) [16].

En el trabajo anteriormente comentado, además de realizar un modelo para la clasificación de las imágenes, el autor del mismo realiza otro modelo de red denominado *UNet* [17], el cual es utilizado para predecir la segmentación de la imagen de la lesión.

El trabajo comentado anteriormente es sólo un ejemplo del tipo de competencia que podemos encontrar en relación a este proyecto. Además, empresas de gran nivel están invirtiendo mucho tiempo y dinero en aplicaciones del estilo a la realizada en este proyecto.

### 2.2.6. Aspecto externo

Este proyecto será entregado en un CD-ROM debido a su amplia capacidad de almacenamiento, la rapidez que éste presenta y la posibilidad de poder ser utilizado en multitud de dispositivos.

Además, en el CD-ROM se podrá encontrar el manual técnico, manual de usuario y el manual de código en formato PDF. Hemos decidido utilizar este formato por su amplio uso en el intercambio de documentos, además de ser un formato que no ocupa mucho almacenamiento. El manual de usuario será un documento en el que se explicará cómo usar esta aplicación junto con unas gráficas que explicarán el proceso.

Con respecto al manual de código, va dirigido al usuario experto para que pueda apreciar cómo se ha creado el modelo junto con los módulos adicionales que son necesarios para el funcionamiento del clasificador presentado en este proyecto.

### 2.2.7. Estandarización

La estandarización usada en este proyecto será la estudiada durante los años de la carrera. Esta estandarización es necesaria para una mejor comprensión del código y para que, así, el usuario que vaya a realizar un estudio de este trabajo, no tenga ninguna complicación en entender cuál es la función de cada una de las variables que se presentan en el código.

Esta estandarización presenta las siguientes características:

- El nombre de las variables debe de ser lo más explicativo posible, evitando utilizar abreviaturas para facilitar así su compresión.
- Todas las variables han de estar en minúscula, conteniendo un subrayado _ en el caso de ser una palabra compuesta.
- El nombre de las clases creadas han de empezar con mayúscula, identificando al objeto que se va a crear.
- Las bibliotecas utilizadas en cada uno de los ficheros han de estar al principio de éste, aunque *Python* sea capaz de detectar una biblioteca en mitad del fichero.
- Cada clase estará definida por un comentario explicativo encima de la creación de ésta. Además, cada función tendrá un comentario con el nombre, funcionalidad y qué es devuelto por ésta.

Cada una de estas características sigue la estandarización *PEP 8* [18] del lenguaje de programación *Python*.

### 2.2.8. Calidad y fiabilidad

Para poder evaluar la calidad de este proyecto será necesaria la intervención del alumno y de los directores del mismo. Este proyecto seguirá los estándares definidos por la *Ingeniería del Software* para la mejor compresión del mismo. Al finalizar le proyecto, éste será evaluado por un tribunal adjudicado por el centro en el que se realizó.

Este proyecto contará con la fiabilidad de que señalizará los errores cometido en la ejecución del programa. El usuario recibirá un error en el caso de introducir parámetros incorrectos. Además, el sistema informará cuando el usuario no ha introducido un parámetro, avisando que se utilizará los valores por defecto.

### 2.2.9. Programa de tareas

Este proyecto se dividirá en las siguientes fases:

1. **Fase de preparación:** En esta fase se hará un estudio plenamente teórico, estudiando las diferentes herramientas que serán utilizadas por nuestro proyecto. Además, esta fase se realizará la preparación del entorno de trabajo que consistirá en la configuración de *Python*, *Tensorflow* y *Keras*.

2. **Fase de diseño:** Una vez hemos adquirido los conocimientos teóricos necesarios para este proyecto, pasaremos al diseño de nuestro proyecto, dejando claro cuáles son los datos necesarios, los flujos de controles que necesitará nuestro proyecto, etc.

3. **Fase de implementación:** Adquiridos los conocimientos teóricos y declarado el diseño de nuestro proyecto, se procederá a la creación del código correspondiente, usando el lenguaje orientado a objetos *Python* junto con las bibliotecas de *Keras* y *Tensorflow*.

4. **Fase de pruebas:** Una vez el código de nuestro proyecto ha sido completado, empezará la fase de pruebas. Esta fase será necesaria para conseguir una optimización de nuestro código, pero sobre todo, para la optimización de nuestro modelo de red neuronal.

5. **Fase de documentación:** Una vez optimizado el código y el modelo, se pasará a la documentación de los mismos. Esta fase será la más tediosa al tener que documentar tanto el código como el por qué hemos elegido este modelo para la clasificación de la base de datos utilizada.

### 2.2.10. Pruebas

Este proyecto será sometido a diversas pruebas, entre las que destacan:

- **Pruebas de Unidad o Caja blanca:** Estas serán las pruebas que se realizarán durante la codificación del proyecto, solucionando los errores que vayan apareciendo por el uso de este tipo de pruebas.

- **Pruebas Funcional o de Caja negra:** Esta pruebas serán las que se realicen en al finalización del código. Serán las que nos permitan saber si nuestro modelo está configurado correctamente o si hace falta algún que otro ajuste de los hiper-parámetros.

### 2.2.11. Seguridad

En este proyecto, la seguridad del mismo no es lo más importante, ya que la temática no da oportunidad a problemas de seguridad. El uso de la privacidad de la aplicación no es de interés ya que los datos no son nominativos.

# Capítulo 3

# Objetivos

Una vez presentados el problema real y técnico de este proyecto, procederemos a explicar los objetivos del mismo. En este capítulo, se explicarán los objetivos que se han de cumplir al finalizar el mismo. *Grosso modo*, los objetivos generales de este proyecto son la compresión del uso de *Python*, como lenguaje de programación, y el uso de las bibliotecas *Tensorflow* y *Keras* para la codificación de una red neuronal residual profunda que resuelva el problema de la clasificación de lesiones benignas y melanomas.

Los objetivos de este proyecto se presentan a continuación:

- Estudio en profundidad del lenguaje orientado a objetos *Python*.
- Estudio en profundidad de la biblioteca para creación de modelos neuronales *Keras*.
- Estudio teórico de las diferentes ecuaciones relacionadas con el entrenamiento de una red residual profunda.
- Estudio de la arquitectura de una red residual profunda para la clasificación de imágenes.
- Estudiar las diferentes posibles configuraciones en este tipo de redes.
- Estudio del procesamiento de imágenes anterior a su uso en el entrenamiento de la red.
- Aplicación de redes neuronales para su uso como clasificador para la base de datos utilizada.
- Comprensión de las distintas lesiones que puede presentar un paciente en al piel.

- Comprensión de los distintos tipos de extensiones de ficheros para poder guardar la información relacionada con una imagen.
- Estudio de las diferentes formas de representación, en una gráfica, de los datos obtenidos.
- Estudio de la metodología para la clasificación de lesiones de piel presentada en el artículo [7]. Este

Junto a estos objetivos también se debe de mencionar el estudio de la creación de un entorno virtual en un sistema operativo además del estudio de los diferentes sistemas de composición de texto, que en nuestro caso ha sido *LaTeX* [19],

# Capítulo 4

# Antecedentes

## 4.1. Redes neuronales residuales

Dentro de los distintos modelos que se pueden utilizar para la clasificación de lesiones en la piel, hemos elegido el modelo *red neuronal residual* (ResNet) [8]. Este modelo se basa en las construcciones conocidas como las células piramidales en la corteza cerebral. Este modelo realiza esta simulación creando una serie de *conexiones de omisión* o *atajos* para saltarse algunas capas. Este tipo de *atajos* se pueden apreciar gráficamente en la Figura 4.1.

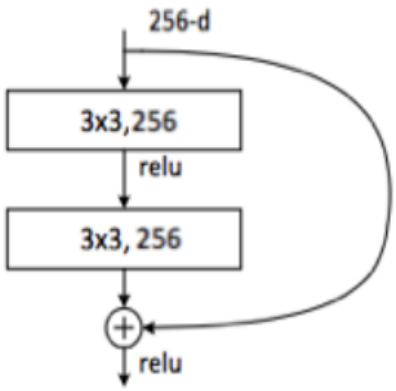


Figura 4.1: *Atajo* creado por la ResNet

El motivo por el que se crean estos atajos, es porque, gracias a los mismos, conseguimos reducir el error producido por la desaparición del gradiente, ya que reutiliza las activaciones de las capas anteriores hasta que la capa adyacente haya aprendido sus pesos.

Este modelo de red neuronal también se caracteriza porque a mayor número de capas utilizadas para construir el modelo, mejor resultado obtendremos, mejora que no ocurre con otros modelos profundos, el los que, cuando se añade un mayor número de capas, el error cometido en entrenamiento, en vez de descender, comienza a aumentar.

Además, *ResNet* presenta una característica que los demás modelos no tienen, y es que el número de épocas necesarias para conseguir el óptimo es mucho menor, aunque también hay que decir que el número de iteraciones por época es mucho más costoso que en los demás modelos.

## 4.2. Tensorflow

*Tensorflow* es una biblioteca para el lenguaje *Python*. Es de código abierto, lo que significa que cualquier usuario puede utilizarla sin necesidad de comprarla. Esta biblioteca se utiliza para el aprendizaje automático a través de un rango de tareas. Fue desarrollada por *Google* para satisfacer la necesidad de éste de construir redes neuronales profundas de manera mucho más sencilla.

Puede ser ejecutada tanto en las CPUs como en las GPUs, dependiendo de como se configure la instalación. Además de proporcionar una biblioteca para el lenguaje *Python*, también está disponible para los lenguajes *C++* [20], *Haskell* [21], *Java* [22], y otros.

## 4.3. Keras

Es una biblioteca pensada para crear modelos de inteligencia artificial. Esta biblioteca fue creada por *François Chollet* [23] el 27 de Marzo del 2015. Además de poder ser ejecutada sobre *Tensorflow*, también puede ser ejecutada sobre *Microsoft Cognitive* [24] o *Theano* [25].

Permite al usuario común poder crear modelos de redes neuronales artificiales profundas de manera muy sencilla y rápida. Cabe destacar que la comunidad que hay detrás de esta biblioteca es muy grande, con lo que el código que implementa tendrá más funciones y permitirá en un futuro que el usuario pueda crear cualquier tipo de red neuronal artificial. Además, esta biblioteca permite generar modelos de Deep Learning [3] en dispositivos con

iOS o Android.

## 4.4. NumpPy

*NumPy* [26] es una extensión del lenguaje *Python*, la cual permite la creación de vectores y matrices de una forma sencilla y rápida. El autor de esta extensión fue *Jim Hugunin*, el cual creó un programa el cual permitía el manejo de vectores y de matrices en el lenguaje orientado a objetos *Python*. Pero fue en 2005 cuando *Travis Oliphant* [27] mejoró el anterior programa introduciendo al mundo la extensión *NumPy*. Cabe destacar que esta sencilla manera de creación y modificación de vectores o matrices es posible ya que su autor creó un tipo de objeto denominado *numpy.array*, el cuál es el que permite realizar operaciones complejas de manera muy sencilla.

## 4.5. Tensorboard

*Tensorboard* es una herramienta soportada por *Tensorflow* que permite al usuario la creación de una serie de gráficas cuya información es la relacionada con el entrenamiento del modelo de red neuronal. Esta herramienta necesita, para su funcionamiento, la existencia de unos ficheros denominados *events*.

## 4.6. Cuadernos Jupyter

*Jupyter Notebook* [28] es un entorno computacional basado en la red para crear documentos interactivos utilizando el lenguaje *Python*. Estos cuadernos, como ya hemos dicho, son interactivos, permiten al usuario poder ejecutar el código generado en los mismos, guardando en memoria todas las variables utilizadas en el momento y no siendo necesaria la creación de funciones cuyos parámetros de entrada son pasados por referencia.

## 4.7. Visual Studio Code

*Visual Studio Code* [29] es un editor de código fuente desarrollado por *Microsoft* para los sistemas operativos *Windows, Linux y Mac OS*. Este editor incluye una serie de funciones que nos permiten la depuración del código junto con la posibilidad de instalar extensiones para visualizar y formatear nuestro código de manera cómoda y sencilla. Además, es un editor de texto de código abierto, no incluyendo ninguna función de pago.

## 4.8. Hierarchical Data Format

*Hierarchical Data Format* [30] es un tipo de formato para un fichero, en nuestro caso *HDF5*, que permite el almacenamiento jerárquico de grandes ficheros. Originalmente, fue desarrollado en el *National Center for Supercomputing Applications* [31], y en la actualidad está soportado por *The HDF Group*.

## 4.9. Conjunto de datos *ISIC*

El conjunto de datos ISIC [13] fue creado por una asociación académica para facilitar la obtención de imágenes digitales de la piel con el objetivo final de reducir la mortalidad por melanoma. Este conjunto de casos clínicos se elabora de manera colaborativa entre varias industrias internacionales y cuenta con un total de 23.906 imágenes de lesiones en la piel, las cuales son clasificadas como benignas, malignas, intermedias o desconocidas. Para este estudio sólo se han recogido imágenes de lesiones benignas y malignas.

Además de la existencia de una base de datos pública, esta asociación realiza una serie de competiciones anuales en las cuales se busca el mejor modelo de red neuronal que pueda realizar las siguientes tareas:

- **Segmentación**: Esta tarea consiste en conseguir como resultado una imagen que pueda usarse como máscara para la posterior segmentación de la imagen original.
- **Detección de atributos de la lesión**: Esta tarea, como su propio nombre indica, consiste en la creación de una red neuronal que sea capaz de detectar el mayor número de atributos dentro de una imagen de una lesión en la piel.
- **Clasificación de lesiones**: Esta última tarea consiste en la clasificación de las imágenes según el tipo de lesión que presenten.

## 4.10. Optimizadores

La optimización es un paso fundamental para el funcionamiento de una red neuronal. Esto conlleva a su uso casi obligatorio a la hora de crear una red neuronal. Para nuestro modelo, hemos seleccionado tres tipos de optimizadores que nos permitirán conseguir, o no, mejores resultados a la hora del entrenamiento. Los optimizadores usados son los siguientes:

- **Optimizador *Adam***: Al utilizar este optimizador, estamos reduciendo el problema del gradiente descendente. Este optimizador presenta características que otros optimizadores no tienen, tales como:
  - Necesita muy pocos requisitos de memoria.
  - Es invariante a la escala diagonal de los gradientes.
  - Los hiperparámetros requieren poca sintonización.
- **Optimizador *RMSprop***: Este optimizador restringe las oscilaciones del descenso del gradiente en la dirección vertical, lo que conlleva que podamos aumentar nuestro ratio de aprendizaje para que nuestro algoritmo pudiera dar pasos más grandes y, así, converger más rápido.
- **Optimizador *SGD***: Este optimizador es el más usado en redes neuronales, ya que nos permite introducir el *momentum*, el ratio de aprendizaje y la variante del *momentum* de *Nesterov*.

# Capítulo 5

# Restricciones

En este capítulo se expondrán las distintas restricciones para este proyecto. Estas restricciones se documentarán desde un punto de vista del diseño, explicando el por qué hemos decidido utilizar un lenguaje o herramienta para la creación del mismo. Este capítulo se dividirá, a su vez, en dos secciones. La primera, los **factores dato** y la segunda los **factores estratégicos**.

## 5.1. Factores dato

Estos factores son los inherentes al problema, y no pueden ser modificados bajo ningún concepto. Estos factores, normalmente, son indicados por el cliente al comienzo del proyecto, y entre ellos podemos destacar los siguientes:

- **Restricciones humanas:** Al tratarse de un Trabajo de fin de grado, este factor viene dado por los directores del proyecto junto con el alumno que lo ha realizado. Este factor viene estipulado por las directrices de la *Universidad de Córdoba*.

- **Restricciones económicas:** Como para la anterior restricción, al tratarse de un *Trabajo de Fin de Grado*, este factor viene dado por el uso de software libre y en el caso de no serlo, que dicho software permita ser usado sin realizar ningún coste económico.

- **Restricciones temporales:** Como viene estipulado en las directrices de un *Trabajo de Fin de Grado* en la *Escuela Politécnica de Córdoba*, este proyecto ha de ser empleando un total de 300 horas. Además, para que este proyecto no se alargue indefinidamente, se da como fecha estimada de finalización la convocatoria extraordinaria de Septiembre del 2019.

- **Restricciones de hardware:** Con respecto a las restricciones de hardware, el alumno ha de utilizar su propio equipo informático para la realización de este proyecto, tanto para la codificación como para la documentación del mismo. Además, las pruebas necesarias para la comprobación del proyecto se realizarán en las máquinas del grupo AYRNA [32].

## 5.2. Factores estratégicos

Con respecto a los factores estratégicos, son aquellos que son impuestos por el alumno que desarrolla el proyecto, junto con los directores del mismo. En esta sección se dará una explicación del por qué de la elección del lenguaje utilizado para la creación de la red, y del lenguaje utilizado para la documentación del proyecto.

Para este proyecto, el lenguaje elegido es el leguage orientado a objetos *Python*, ya que el alumno ya contaba con el suficiente conocimiento como para poder crear el proyecto. Además, este lenguaje permite la inclusión de varias bibliotecas, como lo son *Tensorflow* y *Keras* que permiten la creación de modelos de red neuronal de manera muy sencilla y rápida. Junto con lo comentado, hemos elegido *Python* como lenguaje para la creación de la red ya que contamos con varias librerías que permiten el manejo, tanto de imágenes, como de vectores y matrices, con sólo un par de líneas.

Para la creación del modelo, el alumno ha hecho uso de cuadernos de *Jupyter* ya que permiten realizar ejecutar código sin necesidad de preparar un entorno con antelación. Además, hemos decidido utilizar *Tensorboard* ya que nos permite realizar un seguimiento en tiempo real y de manera visual del comportamiento de la red neuronal.

Para el desarrollo del código de la red neuronal, hemos decidido utilizar *Visual Studio Code*, ya que nos permite visualizar el código con una paleta de colores que depende del tipo de variable. Esto hace que el entendimiento del código sea mucho más rápido y llevadero. Junto con esto, *Visual Studio Code* nos permite tener integrada una terminal en la que podremos ejecutar nuestro código sin tener que cambiar de pantalla una y otra vez.

Para la documentación de este proyecto hemos decidido utilizar la plataforma web *Overleaf*, además de porque el alumno ya tenía conocimientos de la misma, porque nos permite tener nuestros documentos *LaTeX* en cual-

quier dispositivo que tenga conexión a internet, ya que todos los archivos son subidos a los servidores web remotos.

# Capítulo 6

# Recursos

En este capítulo se expondrán los recursos, tanto humanos, como materiales que han sido necesarios para la creación de este proyecto.

## 6.1. Recursos humanos

- **Autor**: Juan José Méndez Torrero.

  Alumno de cuarto curso del Grado en Ingeniería Informática en la mención de computación.

- **Directores**:

  - D. Pedro Antonio Gutiérrez Peña.

    Profesor Titular de Universidad del Departamento de Informática y Análisis Numérico de la Universidad de Córdoba y miembro investigador del grupo AYRNA.

  - D. Javier Sánchez Monedero.

    Investigador de la Universidad de Cardiff y miembro investigador del grupo AYRNA.

## 6.2. Recursos materiales

### 6.2.1. Recursos software

- Mac OS Mojave Versión 10.14.4 y Ubuntu 18.04 como Sistemas Operativos.

- *Visual Studio Code* como entorno de desarrollo.

- *Python* como lenguaje de programación.
- *Keras* como herramienta para la creación de los modelos neuronales haciendo uso como *backend* de *TensorFlow*.
- *LaTeX* para la creación de la documentación.

### 6.2.2. Recursos hardware

Para la realización de este proyecto se hará uso de un equipo personal con las siguiente características:

- MacBook Pro (Retina 13 pulgadas, año 2017)
- Procesador: 2,3 GHz Intel Core i5.
- Memoria: 8GB 2133 MHz LPDDR3.
- Gráficos: Intel Iris Plus Graphics 640 1536 MB.
- 128 GB almacenamiento flash PCIe.

Para la realización de las pruebas se hará uso de los recursos hardware del grupo de investigación AYRNA[32], los cuales consisten en un clúster con las siguientes características:

- Procesador: 8 procesadores Intel Xeo 3.2 GHz (32 bits y 2 núcleos de procesos) y 8 procesadores Intel Xeon 2 GHz E5405 CPUs (64 bits y 4 núcleos de procesos).
- Memoria: 1GB de RAM por núcleo de proceso.
- Capacidad: Disco duro de 2 TB.

# Parte II

# Especificación de requisitos

# Capítulo 7

# Especificación de requisitos

En este capítulo se expondrán, desde un punto de vista técnico, las especificaciones que han sido necesarias para la creación de este proyecto.

## 7.1. Introducción

Como ya hemos comentado, en este capítulo se expondrá "qué debe de hacer" nuestro proyecto. La especificación de requisitos *(ERS)* es una de las partes más importantes de este proyecto ya que, en gran medida, determinará la calidad del proyecto.

Este proyecto ha sido creado por la necesidad de crear un sistema informático que soporte las conclusiones de un profesional médico a la hora de determinar el tipo de lesión en la piel que sufre un paciente. Para ello, en este capítulo, podremos encontrar las siguientes secciones:

- **Participantes del proyecto:** Se podrá ver quién es el autor del proyecto junto con sus directores.
- **Descripción del sistema:** En esta sección se podrá entender, aún mejor, el sistema a crear por el autor del proyecto.
- **Objetivos del sistema:** Aquí, se expondrán diferentes tablas en las que se podrá ver los objetivos a cumplir por el sistema.
- **Requisitos del sistema:** Esta sección estará dividida a su vez por las siguientes secciones:
    - **Requisitos de la información:** Estos requisitos definen los componentes que el sistema debe tener para realizar determinadas tareas.

- **Requisitos funcionales:** Son las características requeridas del sistema que expresa una capacidad de acción del mismo.
- **Requisitos no funcionales:** Son las características requeridas del sistema, del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que señala una restricción del mismo.

## 7.2. Participantes del proyecto

En esta sección se expondrán el autor y los directores del proyecto. La información relativa a éstos se podrá observar en las tablas 7.1, 7.2 y 7.3.

| **Participante** | Juan José Méndez Torrero |
|---|---|
| **Organización** | Escuela Politécnica Superior de Córdoba |
| **Desarrollador** | Sí |
| **Cliente** | No |
| **Usuario** | Sí |
| **Comentarios** | Alumno de cuarto curso del grado de ingeniería informática en la mención de computación. |

Tabla 7.1: Participante Autor

| **Participante** | Pedro Antonio Gutiérrez Peña |
|---|---|
| **Organización** | AYRNA |
| **Desarrollador** | No |
| **Cliente** | Sí |
| **Usuario** | Sí |
| **Comentarios** | Profesor Titular de la Universidad de Córdoba en el Departamento de Informática y Análisis Numérico de la Politécnica Superior de Córdoba. |

Tabla 7.2: Participante director 1 del proyecto

| **Participante** | Javier Sánchez Monedero |
|---|---|
| **Organización** | AYRNA |
| **Desarrollador** | No |
| **Cliente** | Sí |
| **Usuario** | Sí |
| **Comentarios** | Investigador de la Universidad de Cardiff y miembro investigador del grupo AYRNA. |

Tabla 7.3: Participante director 2 del proyecto

## 7.3. Descripción del sistema

Este proyecto consiste en la creación de una red neuronal artificial del tipo residual, la cual sea capaz de clasificar tipos de lesiones en la piel con el mayor porcentaje de acierto y, al mismo tiempo, el menor porcentaje de error posible. La base de datos utilizada es una serie de imágenes de lesiones en la piel descargadas de la galería de imágenes de *ISIC*. Para ello, se utilizará el filtro que nos brinda la web para sólo descargarnos imágenes de tipo maligno y de tipo benigno. Además, este programa será capaz de devolver una serie de ficheros que nos servirán para tener un seguimiento del comportamiento de nuestra red a través de la herramienta *Tensorboard*. Gracias a esto podremos comparar varios resultados, obtenidos de diferente manera al cambiar los hiper-parámetros de nuestro modelo. Como características destacables de este proyecto tenemos:

- Implementará una red neuronal residual.
- Creará una interfaz de comandos que permitirá al usuario configurar los parámetros necesarios para la construcción y entrenamiento de nuestro modelo.
- Nos devolverá un fichero a utilizar junto con la plataforma *Tensorboard* en el que podremos ver la precisión y el error de nuestro modelo.
- Permitirá reducir el tamaño de la imagen de entrada y configurar el procesamiento de la misma.
- Aplicará una clasificación binaria de una imagen de entrada en *tipo maligno* y *tipo benigno*.
- Permitirá ver en tiempo real los resultados de cada iteración por consola. Al final de cada época, se mostrará una media de los valores

conseguidos y se realizará una validación que devolverá la precisión y el error.

Como resumen, este proyecto se basa en la creación de un programa que permita la clasificación binaria de imágenes según el tipo de lesión que presente, entre las que consideraremos *maligno* y *benigno*. Ya existen estudios relacionados con este tema, siendo éste el primero en utilizar las herramientas *Keras* y *Tensorflow* para la creación de un modelo de red neuronal residual.

## 7.4. Objetivos del sistema

En esta sección se dará una explicación de los distintos objetivos que presenta este proyecto.

| | |
|---|---|
| **OBJ-0001** | Clasificación de lesiones en la piel a través de una red neuronal residual profunda |
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Desarrollo de una red neuronal residual profunda para la correcta clasificación de imágenes de lesiones en la piel. |
| **Subobjetivos** | OBJ-0002 Implementación de una interfaz<br>OBJ-0003 Lectura de datos<br>OBJ-0004 Diseño de un modelo de red<br>OBJ-0006 Predicción de etiqueta en imágenes nuevas<br>OBJ-0007 Seguimiento de resultados |
| **Comentarios** | |

Tabla 7.4: Objetivo OBJ-0001: Clasificación de lesiones en la piel a través de una red neuronal residual profunda

| **OBJ-0002** | Implementación de una interfaz |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Permitir al usuario la posibilidad de cambiar los hiper-parámetros del modelo de manera sencilla. |
| **Subobjetivos** | |
| **Comentarios** | |

Tabla 7.5: Objetivo OBJ-0002: Implementación de una interfaz

| **OBJ-0003** | Lectura de datos |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Este objetivo consiste en la creación de una serie de funciones que permitan agrupar las imágenes introducidas en dos etiquetas. Las imágenes correspondientes a lesiones benignas y las lesiones malignas. |
| **Subobjetivos** | |
| **Comentarios** | Esta lectura no guardará información adicional, es decir, sólo se guardará el tipo de lesión. |

Tabla 7.6: Objetivo OBJ-0003: Lectura de datos

| **OBJ-0004** | Diseño de un modelo de red |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Este objetivo consiste en la creación de ciertas funciones que nos permitan, compilar, entrenar y evaluar un modelo de red neuronal residual profunda. |
| **Subobjetivos** | OBJ-0005 Diseño de métodos para almacenar el modelo entrenado |
| **Comentarios** | Dentro de la aplicación, estas dos funciones irán a la par, ya que no sirve de nada crear un modelo si no es para ser entrenado. |

Tabla 7.7: Objetivo OBJ-0004: Diseño de un modelo de red

| | |
|---|---|
| **OBJ-0005** | Diseño de métodos para almacenar el modelo entrenado |
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Creación de unas funciones que guardarán el modelo junto con sus correspondientes pesos para poder así utilizarlos en la predicción de nuevas imágenes. |
| **Subobjetivos** | |
| **Comentarios** | |

Tabla 7.8: Objetivo OBJ-0005: Diseño de métodos para almacenar el modelo entrenado

| | |
|---|---|
| **OBJ-0006** | Predicción de etiqueta en imágenes nuevas |
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Este objetivo consiste en la creación de funciones que nos permitan cargar los datos recolectados en el objetivo OBJ-0005. |
| **Subobjetivos** | |
| **Comentarios** | Esta función se encontrará fuera del código principal, ya que queremos una predicción visual. Es por ello que esta función se encontrará dentro de un fichero *Jupyter*. |

Tabla 7.9: Objetivo OBJ-0006: Predicción de etiqueta en imágenes nuevas

| **OBJ-0007** | Seguimiento de resultados |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Descripción** | Consiste en la creación de una serie de funciones que nos permitan visualizar, ya sea en tiempo de ejecución o una vez finalizado el entrenamiento, unas gráficas para ver el comportamiento del modelo con los parámetros definidos en el momento de ejecución |
| **Subobjetivos** | |
| **Comentarios** | |

Tabla 7.10: Objetivo OBJ-0007: Seguimiento de resultados

## 7.5. Requisitos del sistema

En esta sección se expondrán los requisitos del sistema. En concreto, los requisitos de información, los requisitos funcionales y por último los requisitos no funcionales.

### 7.5.1. Requisitos de información

Al tratarse de un proyecto de investigación y no de la creación de un sistema software, los requisitos de la información son escasos, siendo éstos los expuestos en las tablas 7.11 y 7.12.

| **IRQ-0001** | Datos de entrada |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-0003 Desarrollo de las funciones de lectura de datos |
| **Descripción** | La aplicación creada en el proyecto deberá leer la imágenes proporcionadas junto con su fichero *Json* dependiendo de la ruta que se especifique en la ejecución del mismo. |

Tabla 7.11: Requisito de información IRQ-0001

| **IRQ-0002** | Datos de salida |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-0005 Desarrollo de funciones para guardar el modelo entrenado<br>OBJ-0007 Desarrollo de funciones de seguimiento |
| **Descripción** | La aplicación creará un fichero de tipo *h5py* en el cual se guardarán los pesos de las conexiones entre neuronas del modelo creado. Además, el programa creará una gráfica en la que se podrá ver tanto el *accuracy* como *loss* del entrenamiento realizado. |

Tabla 7.12: Requisito de información IRQ-0002

### 7.5.2. Requisitos funcionales

En esta sección se explicarán cuáles han sido los requisitos funcionales identificados en el proyecto.

| **FRQ-0001** | Lectura de datos |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos** | OBJ-0002 Desarrollo de la interfaz de comandos<br>OBJ-0003 Desarrollo de funciones de lectura de datos |
| **Requisitos** | IQR-0001 Datos de entrada |
| **Descripción** | El sistema leerá tanto las imágenes como los fichero *Json* dependiendo de la ruta como parámetro de entrada. |
| **Comentarios** | Esta lectura realizará un cambio de tamaño en la imagen y asignará una etiqueta a la misma. |

Tabla 7.13: Requisito funcional FRQ-0001

| **FRQ-0002** | Modo de ejecución |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos** | OBJ-0002 Desarrollo de la interfaz de comandos<br>OBJ-0003 Desarrollo de funciones de lectura de datos |
| **Requisitos** | IRQ-0001 Datos de entrada |
| **Descripción** | El sistema permitirá la configuración de los hiper-parámetros correspondientes al modelo creado |

Tabla 7.14: Requisito funcional FRQ-0002

| **FRQ-0003** | Predicción de imágenes |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos** | OBJ-005 Desarrollo de funciones para guardar el modelo entrenado<br>OBJ-0006 Creación de funciones de predicción |
| **Requisitos** | Ninguno |
| **Descripción** | El modelo de red entrenado podrá ser probado con nuevas imágenes de entrada al poder guardar el modelo resultante al momento de su finalización |

Tabla 7.15: Requisito funcional FRQ-0003

| **FRQ-0004** | Almacenamiento de resultados |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos** | OBJ-005 Desarrollo de funciones para guardar el modelo entrenado |
| **Requisitos** | IQR-0002 Datos de salida |
| **Descripción** | El modelo creará un fichero *h5py* el cual contendrá el valor de los pesos de la red conseguidos gracias al entrenamiento de la misma. |

Tabla 7.16: Requisito funcional FRQ-0004

| **FRQ-0005** | Representación de resultados |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos** | OBJ-0007 Desarrollo de funciones de seguimiento<br>OBJ-0005 Desarrollo de funciones para guardar el modelo entrenado |
| **Requisitos** | IQR-0002 Datos de salida |
| **Descripción** | El modelo creará un fichero que se ejecutará dentro de la herramienta *Tensorboard* en el cual se podrá ver, visualmente, cómo ha sido el comportamiento de la red durante el entrenamiento. |

Tabla 7.17: Requisito funcional FRQ-0005

| **FRQ-0006** | Configuración de los hiper-parámetros |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Objetivos** | OBJ-0002 Desarrollo de la interfaz de comandos |
| **Requisitos** | IQR-0001 Datos de entrada |
| **Descripción** | El usuario tendrá la posibilidad de seleccionar los hiper-parámetros de la red, siendo estos *batch size*, *epochs*, *learning rate*, *optimizer type* y *patience*. |

Tabla 7.18: Requisito funcional FRQ-006

### 7.5.3. Requisitos no funcionales

En esta sección se expondrán los requisitos no funcionales del sistema, una vez ya definidos los requisitos funcionales y de información.

| **NFRQ-0001** | Lenguaje de programación Python |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-0001 Desarrollo de la aplicación<br>OBJ-0002 Desarrollo de la interfaz de comandos<br>OBJ-0003 Desarrollo de las funciones de lectura de datos<br>OBJ-0004 Desarrollo del modelo de red<br>OBJ-0005 Desarrollo de funciones para guardar el modelo entrenado<br>OBJ-0006 Creación de funciones de predicción<br>OBJ-0007 Desarrollo de funciones de seguimiento |
| **Descripción** | El leguaje de programación orientado a objetos *Python*, ha sido elegido como lenguaje de programación por su sencillez y por su facilidad de incorporar nuevas bibliotecas. |

Tabla 7.19: Requisito no funcional NFRQ-0001

| **NFRQ-0002** | Fiabilidad y optimización |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-0001 Desarrollo de la aplicación<br>OBJ-0002 Desarrollo de la interfaz de comandos<br>OBJ-0003 Desarrollo de las funciones de lectura de datos<br>OBJ-0004 Desarrollo del modelo de red<br>OBJ-0005 Desarrollo de funciones para guardar el modelo entrenado<br>OBJ-0006 Creación de funciones de predicción<br>OBJ-0007 Desarrollo de funciones de seguimiento |
| **Descripción** | El programa debe de estar codificado de tal manera que, cuando un error ocurre, el usuario pueda saber cuál ha sido el error y dónde se ha producido. Además, el código está optimizado, tanto en almacenamiento como en tiempo de ejecución. |

Tabla 7.20: Requisito no funcional NFRQ-0002

| **NFRQ-0003** | Usabilidad |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-0001 Desarrollo de la aplicación<br>OBJ-0002 Desarrollo de la interfaz de comandos<br>OBJ-0003 Desarrollo de las funciones de lectura de datos<br>OBJ-0004 Desarrollo del modelo de red<br>OBJ-0005 Desarrollo de funciones para guardar el modelo entrenado<br>OBJ-0006 Creación de funciones de predicción<br>OBJ-0007 Desarrollo de funciones de seguimiento |
| **Descripción** | El programa ha de ser fácil de utilizar, permitiendo al usuario, en poco pasos, una ejecución sin errores y sin complicaciones. Esta usabilidad, sobre todo, tiene que tenerse en cuenta para la creación de la interfaz de comandos. |

Tabla 7.21: Requisito no funcional NFRQ-0003

| **NFRQ-0004** | Formatos |
|---|---|
| **Versión** | 1.0 |
| **Autor** | Juan José Méndez Torrero |
| **Fuentes** | Pedro Antonio Gutiérrez Peña<br>Javier Sánchez Monedero |
| **Dependencias** | OBJ-0003 Desarrollo de las funciones de lectura de datos<br>OBJ-0006 Creación de funciones de predicción<br>OBJ-0007 Desarrollo de funciones de seguimiento |
| **Descripción** | El programa soportará, como entrada, imágenes en formato *JPEG* y ficheros que guarden su información en formato *Json*. Con respecto al entrenamiento, el formato de los ficheros de entrada ha de ser *HDF5*, ya que nos permitirá un almacenamiento menos pesado de los datos. Finalmente, los formatos de salida serán *HDF5* para los pesos de la red. |

Tabla 7.22: Requisito no funcional NFRQ-0004

# Parte III

# Análisis del sistema

# Capítulo 8

# Análisis funcional

Este capítulo explicará el análisis funcional de nuestro proyecto. Para ello, nos basaremos en los *Casos de Uso*, ya que éstos nos permitirán dar una explicación visual sobre cómo interactúan el usuario y el sistema, o el sistema con otro diferente. Para ello, nos basaremos en el lenguaje *UML* [33], el cuál es un lenguaje de modelado unificado.

Cada caso de uso que se expone en este capítulo explicará uno o varios escenarios de esa interacción Usuario-Sistema. Para ello, nos basaremos en una la plantilla representada en la Tabla 8.1, la cuál será capaz de ponernos en la situación de qué es necesario y cómo debe de hacerse esa interacción.

| **Caso de Uso** | Nombre del caso de uso |
|---|---|
| **Identificador** | Identificador del caso de uso |
| **Descripción** | Descripción del caso de uso |
| **Actores** | Actores que intervienen en el caso de uso |
| **Precondiciones** | Precondiciones necesarias para el caso de uso |
| **Flujo principal** | Comportamiento principal del caso de uso |
| **Flujo alternativo** | Comportamiento alternativo del caso de uso |
| **Post-condiciones** | Post-condiciones necesarias para el caso de uso |

Tabla 8.1: Plantilla para los casos de uso de contexto

## 8.1. Identificación de los usuarios del sistema

El programa realizado en este proyecto sólo tiene un tipo de usuario. Este usuario será el que cree el modelo, lo entrene, realice las predicciones pertinentes y finalmente compruebe que el modelo está funcionando correctamente gracias a la herramienta *Tensorboard*.

## 8.2. Casos de uso de contexto

En esta sección se expondrán los casos de uso de contexto de nuestro sistema. Para ello, haremos uso de la plantilla creada y mostraremos un diagrama de este caso de uso, ver Tabla 8.2 y Figura 8.1.

| | |
|---|---|
| **Caso de Uso** | Construcción de un modelo de red neuronal residual profundo |
| **Identificador** | CU-0 |
| **Descripción** | Se basa en la creación de un modelo de red neuronal residual profundo que permita la clasificación de imágenes de tipos de lesiones en la piel. |
| **Actores** | Usuario |
| **Casos de uso** | CU-1 Entrenamiento<br>CU-2 Evaluación<br>CU-3 Predicción |

Tabla 8.2: Caso de uso de contexto CU-0: Construcción de un modelo de red neuronal residual profundo

### 8.2.1. Diagrama de caso de uso de contexto

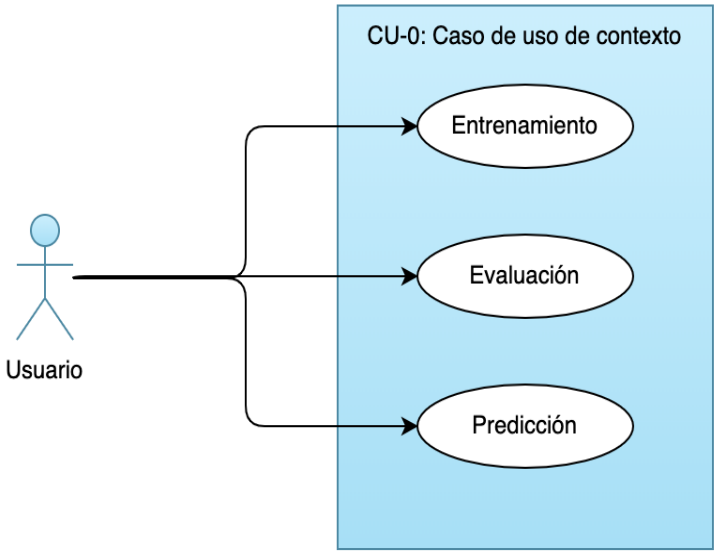


Figura 8.1: Diagrama de caso de uso de contexto

### 8.2.2. Caso de uso: Entrenamiento

El caso de uso correspondiente al entrenamiento de la red se puede observar en la Tabla 8.3.

### 8.2.3. Caso de uso: Evaluación

El caso de uso correspondiente a la evaluación de la red se puede observar en la Tabla 8.4.

### 8.2.4. Caso de uso: Predicción

El caso de uso correspondiente a la predicción de la red se puede observar en la Tabla 8.5.

| **Caso de Uso** | Entrenamiento |
|---|---|
| **Identificador** | CU-1 |
| **Descripción** | Este caso de uso expone el entrenamiento que hace la red que puede realizar el usuario. |
| **Actores** | Usuario |
| **Precondiciones** | Los datos datos de entrada deben de estar en formato *HDF5* |
| **Flujo principal** | El modelo entrena, con los datos de entrada, la red neuronal residual. |
| **Flujo alternativo** | En caso de error, el sistema devolverá al usuario un mensaje comentando cuál ha sido el error. |
| **Post-condiciones** | El entrenamiento es mostrado por pantalla época por época. |

Tabla 8.3: Caso de uso CU-1: Entrenamiento

| **Caso de Uso** | Evaluación |
|---|---|
| **Identificador** | CU-2 |
| **Descripción** | En este caso de uso se expone cómo es la evaluación de la red una vez finalizado el entrenamiento. |
| **Actores** | Usuario. |
| **Precondiciones** | Los datos datos de entrada deben de estar en formato *HDF5*. |
| **Flujo principal** | El modelo evalúa la red ya entrenada y devuelve el valor correspondiente al *accuracy* y el *loss*. |
| **Flujo alternativo** | En caso de error, el sistema devolverá al usuario un mensaje comentando cuál ha sido el error. |
| **Post-condiciones** | El sistema devolverá dos valores correspondientes al *accuracy* y el *loss* devueltos por la evaluación realizada por el sistema. |

Tabla 8.4: Caso de uso CU-2: Evaluación

| **Caso de Uso** | Predicción |
|---|---|
| **Identificador** | CU-3 |
| **Descripción** | El sistema permitirá al usuario realizar una predicción de una nueva imagen no vista hasta el momento. |
| **Actores** | Usuario |
| **Precondiciones** | Los datos datos de entrada deben de estar en formato *JPEG*. Además, el modelo y sus pesos han de estar guardados para poder así usarlos en la predicción |
| **Flujo principal** | El modelo predecirá la etiqueta de la nueva imagen. |
| **Flujo alternativo** | En caso de error, el sistema devolverá al usuario un mensaje comentando cuál ha sido el error. |
| **Post-condiciones** | El valor devuelto por la red será un vector con el valor de la etiqueta. |

Tabla 8.5: Caso de uso CU-3: Predicción

## 8.3. Casos de uso de Entrenamiento

En esta sección se explicarán los casos de uso relacionados con el diagrama de casos de uso mostrado en la Figura 8.2.

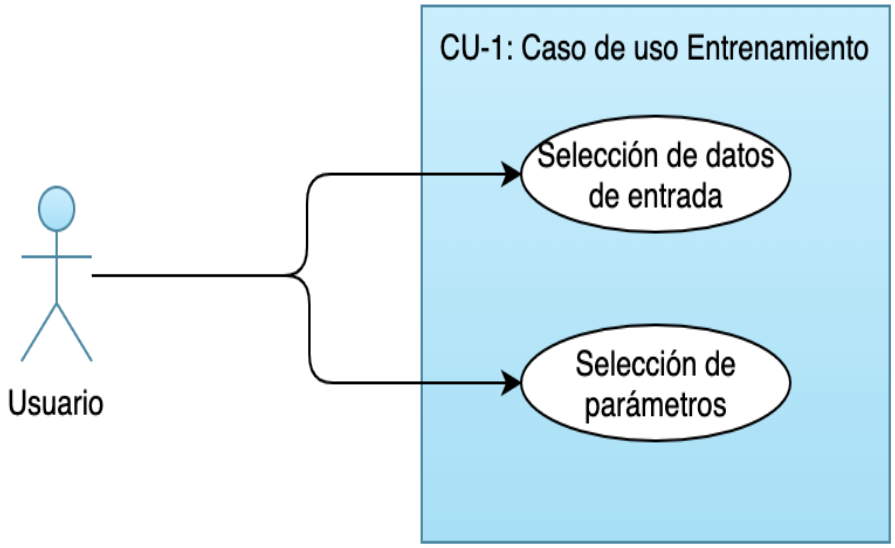


Figura 8.2: Diagrama de caso de uso CU-1: Entrenamiento

### 8.3.1. Caso de uso: Selección datos de entrada para entrenamiento

El caso de uso correspondiente a la selección de datos de entrada se puede observar en la Tabla 8.6.

### 8.3.2. Caso de uso: Selección de parámetros para entrenamiento

El caso de uso correspondiente a la selección de parámetros se puede observar en la Tabla 8.7.

| **Caso de Uso** | Selección de datos de entrada para entrenamiento |
|---|---|
| **Identificador** | CU-1.1 |
| **Descripción** | El sistema permitirá al usuario seleccionar el directorio donde se encuentran las imágenes de entrada. |
| **Actores** | Usuario |
| **Precondiciones** | Los datos de entrada han de estar en formato *HDF5*, es decir, ya deben de estar procesados con anterioridad. |
| **Flujo principal** | El usuario selecciona el directorio sin ningún error. |
| **Flujo alternativo** | En caso de error, el sistema devolverá al usuario un mensaje comentando cuál ha sido el error. |
| **Post-condiciones** | Estos datos quedan seleccionados, listos para ser entrenados. |

Tabla 8.6: Caso de uso CU-1.1: Selección de datos de entrada para entrenamiento

| **Caso de Uso** | Selección de parámetros para entrenamiento |
|---|---|
| **Identificador** | CU-1.2 |
| **Descripción** | El sistema permite al usuario seleccionar los parámetros necesarios para el entrenamiento. |
| **Actores** | Usuario |
| **Precondiciones** | Los datos introducidos han de ser iguales al tipo de dato necesario para poder entrenar el modelo. |
| **Flujo principal** | El modelo será creado con normalidad, siendo configurado como el usuario ha introducido. |
| **Flujo alternativo** | En caso de error, el sistema devolverá al usuario un mensaje comentando cuál ha sido el error. |
| **Post-condiciones** | El modelo estará compilado con los parámetros introducidos, listo para ser entrenado. |

Tabla 8.7: Caso de uso CU-1.2: Selección de parámetros para entrenamiento

## 8.4. Casos de uso de Evaluación

En esta sección se explicarán los casos de uso relacionados con el diagrama de casos de uso mostrado en la Figura 8.3.

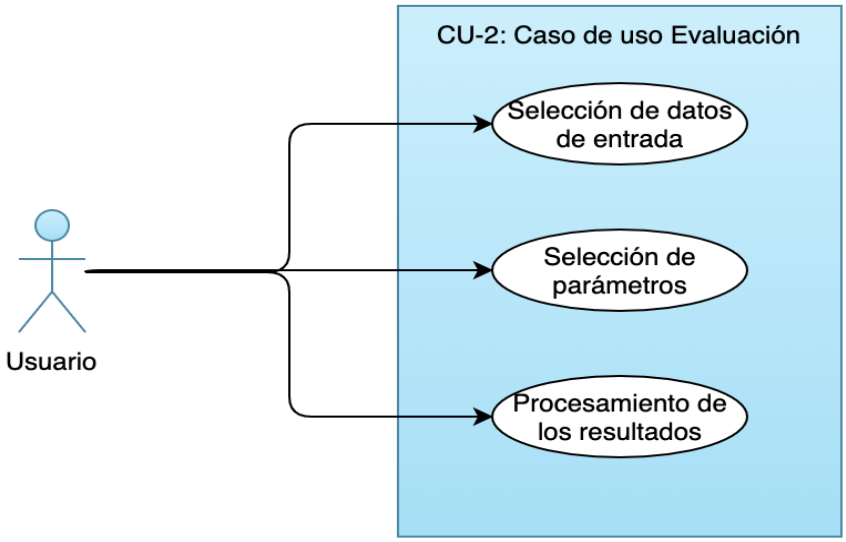


Figura 8.3: Diagrama de caso de uso CU-2: Evaluación

### 8.4.1. Caso de uso: Selección de datos de entrada para evaluación

El caso de uso correspondiente a la selección de datos de entrada se puede observar en la Tabla 8.8.

### 8.4.2. Caso de uso: Selección de parámetros para evaluación

El caso de uso correspondiente a la selección de parámetros se puede observar en la Tabla 8.9.

| **Caso de Uso** | Selección de datos de entrada para evaluación |
|---|---|
| **Identificador** | CU-2.1 |
| **Descripción** | Permite al usuario seleccionar los datos de entrada para la evaluación del modelo. |
| **Actores** | Usuario |
| **Precondiciones** | Los datos introducidos han de ser iguales al tipo de dato necesario para poder entrenar el modelo. |
| **Flujo principal** | El modelo será evaluado con normalidad, siendo configurado como el usuario ha introducido. |
| **Flujo alternativo** | En caso de error, el sistema devolverá al usuario un mensaje comentando cuál ha sido el error. |
| **Post-condiciones** | Los datos quedan seleccionados. |

Tabla 8.8: Caso de uso CU-2.1: Selección de datos de entrada para evaluación

### 8.4.3. Caso de uso: Procesamiento de los resultados para evaluación

El caso de uso correspondiente al procesamiento de los resultados se puede observar en la Tabla 8.10.

| **Caso de Uso** | Selección de parámetros para evaluación |
|---|---|
| **Identificador** | CU-2.2 |
| **Descripción** | Permite al usuario seleccionar los parámetros para la evaluación del modelo. |
| **Actores** | Usuario |
| **Precondiciones** | Los datos introducidos han de ser iguales al tipo de dato necesario para poder entrenar el modelo. |
| **Flujo principal** | El modelo será evaluado con normalidad, siendo configurado como el usuario ha introducido. |
| **Flujo alternativo** | En caso de error, el sistema devolverá al usuario un mensaje comentando cuál ha sido el error. |
| **Post-condiciones** | Los parámetros quedan seleccionados. |

Tabla 8.9: Caso de uso CU-2.2: Selección de parámetros para evaluación

| **Caso de Uso** | Procesamiento de los resultados para evaluación |
|---|---|
| **Identificador** | CU-2.3 |
| **Descripción** | Permite al usuario procesar los datos de salida de la evaluación para su posterior visualización. |
| **Actores** | Usuario |
| **Precondiciones** | Ninguna. |
| **Flujo principal** | El modelo guarda los resultados de la evaluación época por época. |
| **Flujo alternativo** | En caso de error, el sistema devolverá al usuario un mensaje comentando cuál ha sido el error. |
| **Post-condiciones** | Los datos de salida han de estar en el formato *JPEG*. |

Tabla 8.10: Caso de uso CU-2.3: Procesamiento de los resultados para evaluación

## 8.5. Casos de uso de Predicción

En esta sección se explicarán los casos de uso relacionados con el diagrama de casos de uso mostrado en la Figura 8.4.

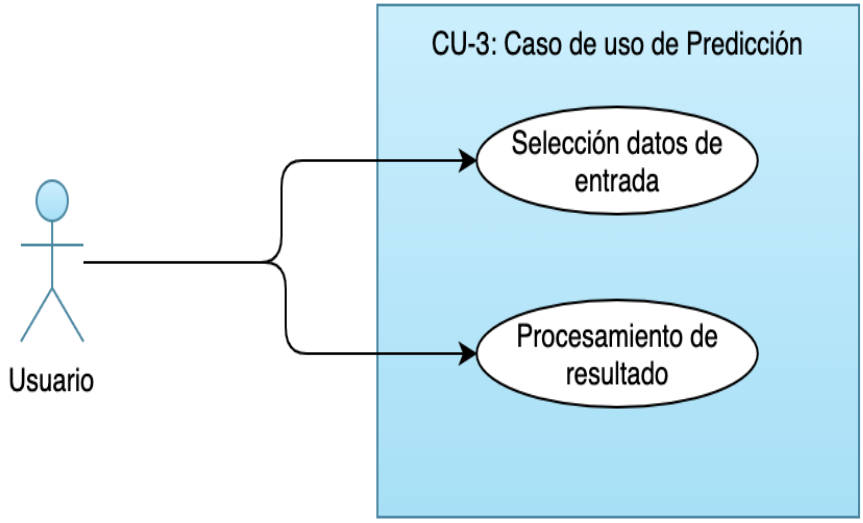


Figura 8.4: Diagrama de caso de uso CU-3: Predicción

### 8.5.1. Caso de uso: Selección de datos de entrada para predicción

El caso de uso correspondiente a la selección de datos de entrada se puede observar en la Tabla 8.11.

### 8.5.2. Caso de uso: Procesamiento de los resultados para predicción

El caso de uso correspondiente al procesamiento de los resultados se puede observar en la Tabla 8.12.

| **Caso de Uso** | Selección de datos de entrada para predicción |
|---|---|
| **Identificador** | CU-3.1 |
| **Descripción** | Permite al usuario seleccionar los datos de entrada para su posterior predicción. |
| **Actores** | Usuario |
| **Precondiciones** | La imagen de entrada ha de estar en formato *JPEG*. |
| **Flujo principal** | El modelo predice la etiqueta de la imagen de entrada. |
| **Flujo alternativo** | En caso de error, el sistema devolverá al usuario un mensaje comentando cuál ha sido el error. |
| **Post-condiciones** | Los resultados de salida se expondrán con la imagen de entrada con su clase original como etiqueta, y con su clase predicha. |

Tabla 8.11: Caso de uso CU-3.1: Selección de datos de entrada para predicción

| **Caso de Uso** | Procesamiento de resultado para predicción |
|---|---|
| **Identificador** | CU-3.1 |
| **Descripción** | El usuario podrá comprobar qué etiqueta predice a la imagen de entrada. |
| **Actores** | Usuario |
| **Precondiciones** | La imagen de entrada ha de estar en formato *JPEG*. |
| **Flujo principal** | El modelo predice la etiqueta de la imagen de entrada. |
| **Flujo alternativo** | En caso de error, el sistema devolverá al usuario un mensaje comentando cuál ha sido el error. |
| **Post-condiciones** | El resultado será un entero con la clase predicha para la imagen de entrada. |

Tabla 8.12: Caso de uso CU-3.2: Procesamiento de resultados para predicción

# Capítulo 9

# Modelo dinámico

En este capítulo, una vez detallados los casos de uso de nuestro sistema, pasaremos a describir los diagramas de secuencia. Para ello, nos basaremos en el lenguaje *UML*, el cuál nos permitirá explicar el flujo normal que sigue nuestro sistema. Al referirnos a *flujo normal*, nos referimos al comportamiento de nuestro sistema cuando no presenta ningún error.

## 9.1. Diagramas de secuencia

En esta sección se explicarán, a través de diagramas de secuencia, los diferentes casos de uso especificados en el Capítulo 8. Para ello, al igual que antes, expondremos los diagramas uno por uno mostrando los casos de uso *CU-1 Entrenamiento*, *CU-2 Evaluación* y *CU-3 Predicción*.

## 9.2. Diagrama de secuencia CU-1 Entrenamiento

### 9.2.1. Diagrama de secuencia

El diagrama de secuencia correspondiente al caso de uso de entrenamiento puede ser observada en la Figura 9.1.

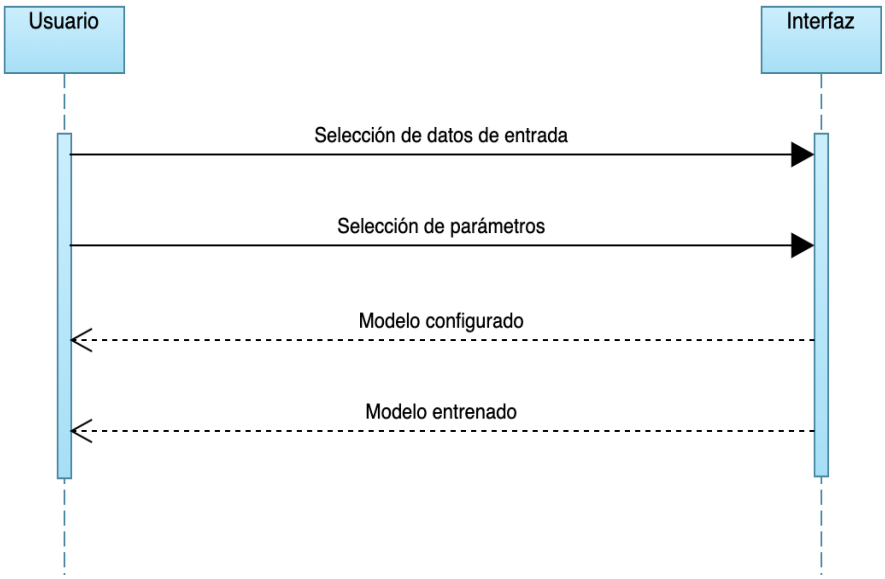


Figura 9.1: Diagrama de secuencia del caso de uso CU-1: Entrenamiento

### 9.2.2. Comportamiento principal

El comportamiento principal de este diagrama de secuencia es el siguiente:

1. El usuario introduce los datos de entrada.
2. El usuario seleccionas los parámetros de configuración para el modelo.
3. La interfaz devuelve el modelo configurado.
4. La interfaz los resultados del entrenamiento.

## 9.3. Diagrama de secuencia CU-2 Evaluación

### 9.3.1. Diagrama de secuencia

El diagrama de secuencia correspondiente al caso de uso de evaluación puede observarse en la Figura 9.2.

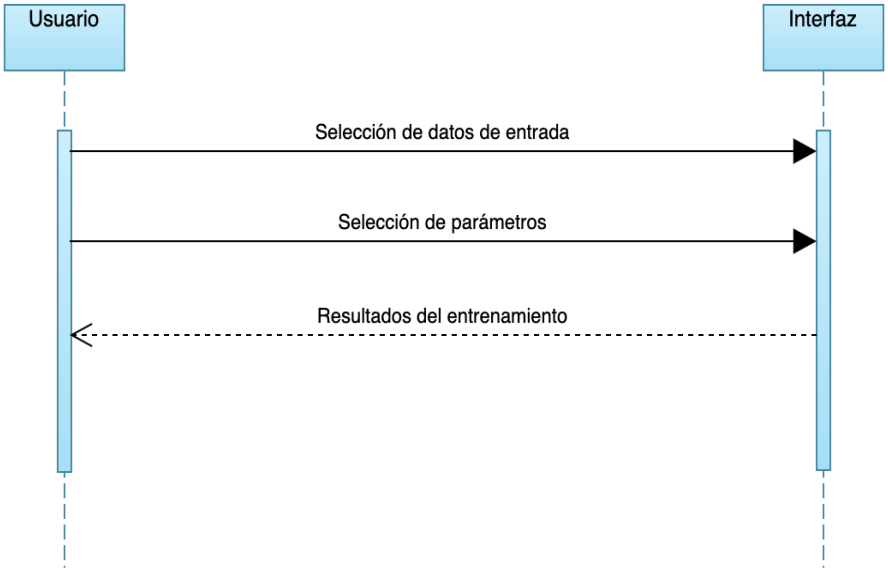


Figura 9.2: Diagrama de secuencia del caso de uso CU-2: Evaluación

### 9.3.2. Comportamiento principal

El comportamiento principal de este diagrama de secuencia es el siguiente:

1. El usuario introduce los datos de entrada.
2. El usuario selecciona los parámetros de configuración para el modelo.
3. La interfaz devuelve los resultados obtenidos en el formato indicado.

## 9.4. Diagrama de secuencia CU-3 Predicción

### 9.4.1. Diagrama de secuencia

El diagrama de secuencia correspondiente al caso de uso de predicción puede ser observado en la Figura 9.3.

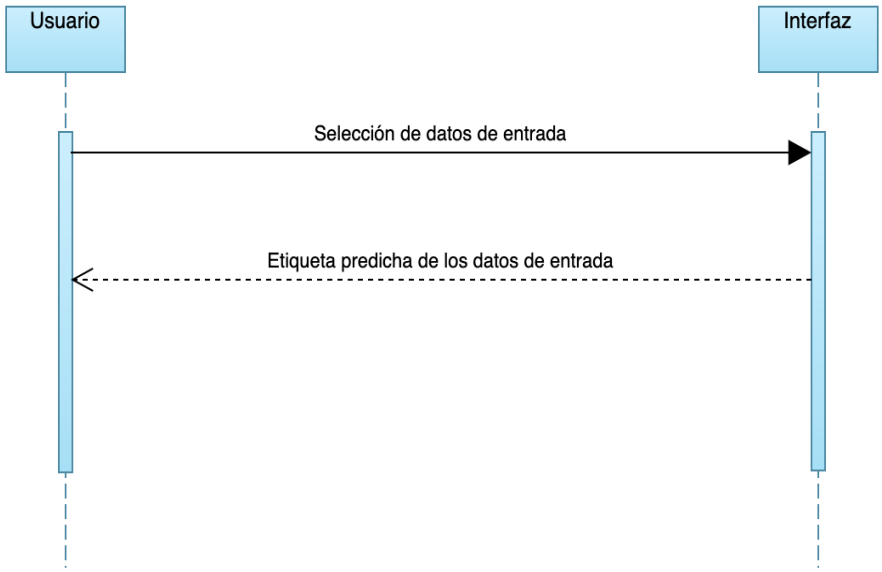


Figura 9.3: Diagrama de secuencia del caso de uso CU-3: Predicción

### 9.4.2. Comportamiento principal

El comportamiento principal de este diagrama de secuencia es el siguiente:

1. El usuario selecciona los datos a utilizar en la predicción.
2. La interfaz devuelve la etiqueta predicha en su formato definido en el caso de uso.

# Parte IV

# Diseño del sistema

# Capítulo 10

# Tecnologías

En este capítulo, comentaremos cuáles han sido las tecnologías que se utilizarán para la creación de este proyecto. Para ello, de cada una de ellas comentaremos un pequeña introducción, un poco de su historia y finalmente las características que presentan.

## 10.1. Python

### 10.1.1. Introducción

*Python* es un lenguaje de programación interpretado, cuya filosofía se basa en facilitar al usuario un código legible. Este lenguaje es multiplataforma, es decir, es capaz de ejecutarse en multitud de sistemas operativos. Además, este lenguaje soporta la orientación a objetos y la programación imperativa. Este lenguaje posee una licencia de código abierto, con lo que cualquier persona podrá utilizarlo sin ningún tipo de inversión. El lenguaje *Python* está administrado por la Python Software Foundation [34].

### 10.1.2. Historia

Este lenguaje de programación fue creado por Guido Van Rossum [35] a finales de los años ochenta en el Centro para las Matemáticas y la Informática en los Países Bajos. Este lenguaje nació como sucesor del lenguaje de programación ABC [36], el cual era capaz de manejar excepciones e interactuar con el sistema operativo Amoeba [37]. El nombre de *Python*, proviene como un guiño a los humoristas *Monty Python* [38], los cuales eran los humoristas preferidos del autor.

En su primera versión, *0.9.0*, la herencia, las excepciones, las funciones y otros ya estaban presentes. Además, en esta versión, el autor decició incluir la cláusula *else*, y no sería hasta 1994 cuando se formó el foro de discusión de *Python*. En este mismo año, se lanzó la versión *1.0*, en la cuál se incluyó las herramientas de programación funcional.

No sería hasta la versión *2.0*, y posteriores, en la que se realizó una actualización del lenguaje para que siguiera la llama *Filosofía Python*. Esta filosofía se compone de varios puntos, los cuales explican cómo un código presenta una legibilidad y transparencia muy alta. Entre esos puntos, caben destacar los siguientes:

- Bello es mejor que feo.
- Plano es mejore que anidado.
- Lo práctico gana a lo puto.
- Los errores nunca deberían dejarse pasar silenciosamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- *etc.*

### 10.1.3. Características

Entre las características más importantes que presenta este lenguaje, cabe destacar que es un lenguaje de programación multiparadigma, es decir, es un lenguaje de programación el cual permite a los programadores adoptar un estilo particular de programación. Además de esta característica, este lenguaje permite la extensión de otros lenguajes, así, permite escribir módulo en otros lenguajes, como *C* o *C++*.

Junto con esto, los elementos que presenta este lenguaje son los siguientes:

- **Comentarios:** En este lenguaje de programación, los comentarios son identificados por encontrase entre tres comillas dobles (*" " "*), finalizando el comentario con otras tres comillas dobles, para comentarios de más de una línea, o # para comentarios de una sóla línea.
- **Variables:** Las variables en este lenguaje de programación se definen de forma dinámica, esto quiere decir que no hace falta identificar el tipo de dato que se está definiendo, con lo que con su propia definición

estamos indicando cuál es el tipo de dato que vamos a utilizar. Los tipos de datos presentes en este lenguaje son los siguientes:

- **str:** Cadena inmutable.
- **unicode:** Cadena, la cuál es una versión *Unicode* de *str*.
- **list:** Secuencia mutable que puede contener varios datos de distinto tipo.
- **tuple:** Secuencia inmutable que puede contener varios tipos de datos distintos.
- **dict:** Grupo de pares de variables, las cuales son identificadas por *clave:valor*.
- **int:** Número entero.
- **bool:** Variable booleana.

Además de las anteriormente nombradas, existen otras muchas con las que se puede realizar una función u otra.

- **Condicionales:** Igual que en otros lenguajes, se caracteriza por una sentencia condicional *if* seguido de un bloque que es ejecutado dependiendo de si esa sentencia ocurre o no. Además, ese bloque puede estar seguido o bien por una sentencia *elif*, en la cuál se vuelve a indicar una nueva sentencia condicional a comprobar, o bien por la sentencia *else*, la cual se ejecutará cuando ninguna de las anteriores sentencias condiconales ocurren.

- **Bucles for y while:** Estos bucles funcionan de manera similar a otros lenguajes, con la diferencia de que con un bucle for, se puede iterar sobre una cadena recogiendo directamente su valor en una cierta posición, es decir, no es necesario ir iterando sobre las posiciones de una variable.

- **Listas y tuplas:** Son grupos de datos que son capaces de almacenar datos de distinto tipo. Al igual que para los vectores en *C*, para acceder al primer elemento de una lista (o tupla), se comienza a iterar desde *0* y no desde *1*. La diferencia entre estos dos tipos de elementos son que, las listas son mutables, es decir, son capaces de cambiar su contenido en tiempo de ejecución. En cambio, las tuplas son inmutables, con lo que no es posible cambiar su contenido una vez han sido creadas.

- **Lista por compresión:** Estas listas son utilizadas para definir una lista en la que se podrá, con tan sólo una línea, rellenar esa lista de manera automática.

- **Interfaz con el sistema operativo:** Para poder utilizar la interfaz de nuestro sistema operativo, deberemos de importar el módulo *os*, el cuál nos permitirá recoger variables introducidas por línea de comando entre otras muchas cosas.

Gracias a la utilización de módulos, este lenguaje de programación aumenta su productividad, ya que al importar el módulo, por ejemplo, *math*, podríamos utilizar funciones previamente declaradas, tales como el número *pi* o calcular el logarítmo de un número.

## 10.2. Tensorflow

### 10.2.1. Introducción

*Tensorflow* es una biblioteca de código abierto utilizada para la creación de redes neuronales y para el uso del aprendizaje automático a través de un rango de tareas. Esta biblioteca ha sido creada por *Google* [39] para satisfacer las necesidades de creación de redes neuronales capaces de detectar patrones sobre un conjunto de datos o bien para poder entender mejor el razonamiento usado por los humanos. Actualmente se encuentra publicado bajo una licencia de código abierto *Apache 2.0*, y esto es así desde el 9 de Noviembre de 2015.

### 10.2.2. Historia

En 2011, el grupo de investigación *Google Brain*, contruyó *DistBelief*, el cual era un sistema de aprendizaje automático, el cuál estaba basado en redes neuronales. Ante su rápido crecimiento, *Google* decidió reunir a varios científicos de la computación para su simplificación y reconstrucción. Este proceso dio paso a la creación de una nueva versión, llamada actualmente como *Tensorflow*. En 2009, uno de los equipos de *Google Brain*, consiguió realizar una red que utilizaba *retropropagación* generalizada, realizado una reducción del 25 % en error del reconocimiento del habla.

El 9 de Noviembre de 2015, nacería oficialmente la biblioteca *Tensorflow*, el cual permite correr en múltiples CPUs y GPUs las redes neuronales creadas. Esta biblioteca está disponible para los sistemas operativos *Linux*, *Mac OS* y para los sistemas operativos móvilos *Android* y *iOS*.

### 10.2.3. Características

Esta biblioteca proporciona una API de *Python*, la cual es la utilizada para la creación de este proyecto. Además de esta API, la biblioteca *Tensorflow* nos permite utilizar lenguajes como *C++*, *C#*, *R* o *Java*.

## 10.3. Keras

### 10.3.1. Introducción

Al igual que *Tensorflow*, *Keras* es una biblioteca de código abierto para la creación de redes neuronales escritas en *Python*. Esta biblioteca puede ser ejecutada sobre *Tensorflow*, *Microsoft Congnitive Toolkit* o *Theano*.

### 10.3.2. Historia

Esta biblioteca fue creada por el ingeniero de *Google François Chollet*, el cual también es el creador del modelo de red neuronal profunda *XCeption* [40]. En 2017, el equipo creador de *Tensorflow* decidió dar soporte a esta biblioteca en el núcleo del mismo. El autor caracteriza a esta biblioteca como una interfaz en la que el programador podrá realizar la creación de redes neuronales profundas de manera muy sencilla, siendo esta biblioteca muy usada, actualmente, por los programadores de redes neuronales profundas.

### 10.3.3. Características

*Keras* se caracteriza por contar con módulos que crean bloques muy utilizados en la creación de redes neuronales, tales como *capas*, *funciones de activación*, *optimizadores*, etc. Además, esta biblioteca contiene muchas herramientas que permiten el procesamiento de imágenes y de texto de manera muy sencilla. Esta biblioteca, además de lo comentado hasta ahora, también permite la creación de modelos de redes neuronales profundas en dispositivos móviles o bien en aplicaciones web.

# Capítulo 11

# Diagrama de clases

En este capítulo se mostrarán las clases creadas para el diseño de la red neuronal profunda residual de este proyecto. Es por ello que especificaremos los diagramas de clases y las especificación de cada una de las clases creadas.

## 11.1. Diagrama de clases

Como para la creación de este proyecto hemos utilizado un lenguaje orientado a objetos, hemos decidido utilizar los diagramas de clases para representar los objetos que se utilizarán en el sistema. Para describir estos objetos, entre los que podremos encontrar los métodos y los datos que componen el sistema, utilizaremos una serie de tablas que seguirán la Tabla 11.1

| **Nombre de la clase** | Nombre de la clase |
|---|---|
| **Descripción** | Descripción del funcionamiento de la clase |
| **Datos** | |
| **dato1** | tipo descripción |
| **Métodos** | |
| **método1** | Descripción de la función |

Tabla 11.1: Plantilla para la descripción de clases

## 11.2. Especificación de clases

### 11.2.1. Clase PrepareData

En esta sección se explicará detalladamente la clase *PrepareData*, la cual se encargará de preparar los datos descargados para tenerlos de manera más ordenada. La especificación de esta clase puede ser observada en la Tabla 11.2.

| **Nombre de la clase** | PrepareData | |
|---|---|---|
| **Descripción** | Esta clase será la encargada se separar los datos descargados en dos tipos, las imágenes de lesiones benignas y las imágenes de lesiones malignas | |
| **Datos** | | |
| **path_images** | String | Identifica la ruta en las que se encuentran las imágenes para su clasificación inicial. |
| **main_path** | String | Identifica la ruta en donde se guardarán los resultados de esta clasificación. |
| **Métodos** | | |
| **createHD5PY** | Guarda las imágenes encontradas en el directorio *path_images* en el directorio *main_path*. El formato de la salida es *hdf5*, en el cual se encontrarán tanto las imágenes encontradas divididas según el tipo de lesión. | |

| | |
|---|---|
| **readDataH5PY** | Lee las imágenes guardadas en el fichero *hdf5* del directorio *main_path*. Esta función devuelve en forma de array las imágenes de tipo benigno y, en otro, las imágenes de tipo maligno. |

Tabla 11.2: Diagrama de la clase PrepareData

### 11.2.2. Clase PrepareTrainTest

Esta clase ha sido creada para preparar los datos de entrada antes de ser entrenados. La especificación de esta clase se puede observar en la Tabla 11.3.

| **Nombre de la clase** | PrepareTrainTest | |
|---|---|---|
| **Descripción** | Esta clase se encargará de separar los datos anteriormente separados en tres partes, una para el entrenamiento (*train)*, una para los test (*test*) y otra para la validación (*validation*). Cada una de estas partes, tendrá una parte que contendrá las imágenes y otra que contendrá la etiqueta de las mismas. | |
| **Datos** | | |
| **main_path** | String | Esta variable guarda la ruta en la que las imágenes para el entrenamiento, junto con su correspondiente etiqueta, son guardadas |

| | | |
|---|---|---|
| **file_name** | String | Nombre del fichero en donde serán guardadas las imágenes con sus etiquetas. |
| **Métodos** | | |
| **createTrainTestH5PY** | Esta función es la encargada de primero, agrupar las imágenes para separarlas en train, test y validación. Una vez ha hecho hecho, transforma las etiquetas en un array categórico y, finalmente, guarda esos vectores en el directorio *main_path* con el nombre *file_name*. | |
| **readDataH5PY** | Esta función se encarga de leer el fichero *file_name* y devolver los vectores correspondientes a cada una de las variables del entrenamiento. | |

Tabla 11.3: Diagrama de la clase PrepareTrainTest

### 11.2.3. Clase ResNet

La clase *ResNet* ha sido creada para almacenar las funciones necesarias para el entrenamiento, evaluación y creación del modelo de red neuronal residual. La especificación de esta clase se puede observar en la Tabla 11.4.

| | |
|---|---|
| **Nombre de la clase** | ResNet |
| **Descripción** | Esta clase contiene una serie de métodos que nos permitirán construir el modelo de red, entrenarlo y finalmente evaluarlo. |
| **Datos** | |

| | | |
|---|---|---|
| **lr** | Decimal | Identifica el ratio de aprendizaje utilizado por la red. |
| **opt** | Entero | Identifica el tipo de optimizador que la red utilizará para compilar el modelo. Su valor será 0, para utilizar el optimizador *Adam*, 1, para el optimizador *RMSprop*, o 2, para el optimizador *SGD*. |
| **batch_size** | Entero | Identifica el tamaño del lote que se entrenará cada iteración. |
| **epochs** | Entero | Identifica el máximo número de iteraciones que nuestro modelo entrenará. |
| **Métodos** | | |
| **trainModel** | Esta función realizará el entrenamiento de nuestra red una vez se hayan comprobado los parámetros de entrada. Nos devolverá el historial del mismo para su posterior uso en la creación de la matriz de confusión. | |

| | |
|---|---|
| **evaluateModel** | Esta función consiste en la evaluación de nuestro modelo una vez entrenado. Esta función nos mostrará por pantalla el valor de precisión y perdida según los datos utilizados para el test. |
| **identityBlock** | Esta función ha sido creada para simular el atajo con el que se identifica el salto entre una capa de la red y otra. Ya explicado en el Capitulo 4.1 |
| **convolutionalBlock** | Esta función ha sido creada para simular unir el atajo creado por la anterior función y el flujo principal de la red. |
| **buildModel** | Esta función es la encargada de unir todo el flujo de capas de nuetra red neuronal residual profunda. Además, nos mostrará por pantalla un resumen de las capas utilizadas y devolverá el modelo para su uso en el resto de funciones. |

Tabla 11.4: Diagrama de la clase ResNet

### 11.2.4. Clase AuxFunctions

Esta clase contendrá las funciones necesarias tanto para la creación del modelo, como para la creación de los ficheros necesarios para la evaluación del modelo. La especificación de esta clase se puede encontrar en la Tabla 11.5.

| | |
|---|---|
| **Nombre de la clase** | AuxFunction |
| **Descripción** | Esta clase tiene al funcionalidad de almacenar las funciones auxiliares necesarias para el sistema. |
| **Datos** | |

| | | |
|---|---|---|
| **model** | Model | Modelo construido. |
| **history** | Dictionary | Es un diccionario que almacena los valores de precisión y pérdida del modelo entrenado. |
| **main_path** | String | Es el directorio principal donde se guardarán los resultado de cada una de las funciones de esta clase. |
| **opt** | Entero | Es el tipo de optimizador utilizado para la compilación del modelo. Sólo puede tomar los valores 0, para utilizar el optimizador *Adam*, 1, para el optimizador *RMSprop*, o 2, para el optimizador *SGD*. |
| **batch_size** | Entero | Es el tamaño del lote que se ha utilizado en cada iteración del entrenamiento. |
| **lr** | Decimal | Es el ratio de aprendizaje utilizado por la red para el entrenamiento. |

| **Métodos** | |
|---|---|
| **create_confusion_matrix** | Este método es el encargado de crear la matriz de confusión obtenida gracias al previo entrenamiento del modelo. La matriz de confusión será guardada en una imagen en la que tendrá como nombre, el optimizador utilizado, el tamaño del lote utilizado y, por último, el valor del ratio de aprendizaje. |
| **create_plots_train_test** | Esta método creará una gráfica en la que se mostrará el valor de pérdida tanto para train como para validación después del entrenamiento a lo largo de las iteraciones. La gráfica tendrá como nombre el optimizador utilizado seguido del valor del tamaño del lote y el valor del ratio de aprendizaje. |
| **saveWeights** | Esta función se encargará de crear un fichero *hdf5* cuyo contenido será el valor de los pesos de la red después del entrenamiento. El nombre que tendrá este fichero será el nombre del optimizador utilizado seguido del valor del tamaño del lote y, finalmente, el valor del ratio de aprendizaje. |

Tabla 11.5: Diagrama de la clase AuxFunction

# Capítulo 12

# Diseño de la interfaz de línea de comandos

En este capítulo se expondrá el diseño de la interfaz de líneas de comandos creada para este proyecto. Para ello, se listarán las diferentes opciones con las que el usuario podrá crear, entrenar y evaluar el modelo residual.

Para que el sistema sea ejecutado correctamente, el usuario tendrá que ejecutar el sistema utilizando las siguientes opciones en la interfaz de línea de comandos:

- **-d, – –datasets TEXT**: Ruta en la que se encuentra el fichero *hdf5* que contiene las imágenes para el entrenamiento.
- **-tt, – –traintest TEXT**: Nombre del fichero *hdf5* contenedor de las imágenes para el entrenamiento.
- **-o, – –optimizer INT**: Tipo de optimizador utilizado para el entrenamiento. Los tipos son:
  - **0**: Optimizador *Adam*.
  - **1**: Optimizador *RSMprop*.
  - **2**: Optimizador *SGD*.
- **-e, – –epochs INT**: Número de épocas que han de transcurrir hasta finalizar el entrenamiento.
- **-lr, – –learningrate FLOAT**: Valor del ratio de aprendizaje a utilizar en el entrenamiento.
- **-b, – –batch INT**: Tamaño del lote a utilizar en cada iteración del entrenamiento.

- **-p, – –patience INT**: Número de épocas que han de pasar sin mejora para dar por finalizado el entrenamiento.
- **-h, – –help**: Muestra la ayuda correspondiente a la ejecución del sistema.

# Parte V

# Pruebas

# Capítulo 13

# Pruebas

Para este tipo de proyecto, la fase de pruebas es la más importante, ya que este fase mostrará la calidad del proyecto. Además de esta razón, la fase de pruebas es muy importante ya que permite al creador adelantar errores antes de terminar con el proyecto, lo que hace ahorrar tiempo y esfuerzo. En el caso de que las pruebas se realizaran al finalizar el proyecto, la acción de tener que borrar elementos intermedios o incluso, realizar cambios de los primeros pasos, hace que la fase de pruebas sea de las más, por no decir la más, importante dentro de todo el proceso de desarrollo de un proyecto.

En este capítulo se abordará el conjunto de pruebas a realizar para comprobar la integridad y calidad de un proyecto. Para ello, dividiremos este capítulo en tres secciones:

- **Pruebas estratégicas.**
- **Pruebas unitarias.**
- **Pruebas del sistema.**

## 13.1. Pruebas estratégicas

A la hora de realizar las pruebas hay que tener mucho cuidado en no dejar ninguna situación sin analizar, ya que no haber realizado una prueba en alguna fase de desarrollo podría implicar tener que volver hacia atrás una vez realizadas las fases siguientes. Para ello, hemos realizado una serie de casos en los que podremos englobar las distintas fases de desarrollo de nuestro proyecto junto con el funcionamiento del mismo.

En el caso de las pruebas efectuadas sobre el sistema para validar su funcionamiento, hemos utilizado las denominadas *pruebas de caja blanca* y *pruebas de caja negra*. Gracias a ellas hemos podido detectar errores sin tener que realizar una *vuelta hacia atrás* en la fase de desarrollo del proyecto.

## 13.2. Pruebas unitarias

Las pruebas unitarias, o *unit testing*, son una fase muy importante en la *metodología ágil* [41]. Estas pruebas consisten en bloques de código que son diseñados para comprobar el correcto comportamiento del programa principal. Estas pruebas presentas distintas características:

1. **Automatizable:** Los resultados obtenidos pueden ser automatizados, con lo que se pueden realizar pruebas individuales o por grupos.

2. **Completas:** Aunque este proceso consta de pequeñas pruebas, siempre hay que comprobar que funcionan en todo el código del proyecto.

3. **Repetibles:** Estas pruebas pueden ser repetidas en todo momento, ya que el resultado ha de ser siempre el mismo dando igual el número de veces que se repita.

4. **Rápido creación:** Este tipo de pruebas han de poder ser creada en menos de cinco minutos, ya que no suelen superar 500 líneas de código.

Estas características son necesarias, pero no suficientes. Es por ello que la creación de unas pruebas que verifiquen el correcto funcionamiento del proyecto, en algunas ocasiones, sea más complicado que la propia creación del proyecto. Es por esto, normalmente, las pruebas unitarias son divididas en dos partes, comentadas anteriormente, *pruebas de caja blanca* y *pruebas de caja negra*.

### 13.2.1. Pruebas de caja blanca

Estas pruebas de caja blanca, también denominadas *pruebas estructurales*, se encargan de la comprobación del correcto funcionamiento del proyecto desde el punto de vista del código, por lo que este tipo de pruebas están muy ligadas al código del proyecto. Se han de realizar en cada uno de los componentes del proyecto y solucionar los errores, si es que ocurren, en cada uno de ellos. Este tipo de pruebas se suele realizar al mismo tiempo que la fase de codificación, ya que en el caso de realizarlas al finalizar esta fase, nos llevaría a un cambio casi completo del proyecto. Además de comprobar el correcto

funcionamiento de cada uno de los módulos que forman el proyecto, también se deben de realizar las pruebas sobre la relación entre los mismos, es decir, comprobar que no se pierde información en la comunicación inter-módulo.

Para este proyecto hemos realizado las siguientes pruebas de caja blanca:

1. Comprobación de que las estructuras de datos almacenadas en cada uno de los módulos no se ha perdido.

2. Realización de pruebas en los límites de los bucles.

3. Comprobación de manera individual de cada una de las sentencias que forman un módulo.

4. Comprobación de la no existencia de pérdida de información a la hora de que un módulo interactúe con otro.

### 13.2.2. Pruebas de caja negra

Las pruebas de caja negra son creadas fuera de la visión del código, es decir, para la creación de este tipo de pruebas nos hemos de fijar en las entradas y salidas del sistema, no siendo necesario conocer cómo funciona la estructura interna del proyecto. Además, son una forma de derivar y seleccionar condiciones, datos y casos de prueba a partir de la documentación de requerimientos del sistema. Una vez dicho esto, se puede añadir que estas pruebas se han de realizar desde el punto de vista del usuario, ya que así podremos ver los distintos errores que pueden surgir si el usuario no realiza un uso adecuado. Como nota, hay que añadir que estas pruebas son creadas una vez finalizada la fase de desarrollo y una vez aceptadas todas y cada una de las pruebas de caja blanca realizadas.

Las pruebas de caja negra realizadas para la comprobación del correcto funcionamiento de este proyecto son las siguientes:

1. Realización de un estudio sobre los valores aceptados por nuestro proyecto, sobre todo de valores límite.

2. Estudio sobre los resultados obtenidos por cada módulo, partiendo de los valores aceptados por nuestro proyecto.

3. Realización de pequeños *programas* que realizaban distintas pruebas sobre el sistema para comprobar su correcto funcionamiento.

## 13.3. Pruebas del sistema

Este tipo de pruebas corresponden a las expuestas en el Capítulo 15. Estas pruebas comprobarán el correcto funcionamiento del sistema a la hora de interpretar los resultados obtenidos, además de comprobar que cada uno de los módulos ha sido integrado correctamente en el sistema.

Este tipo de pruebas realizarán la tarea de comprobar la funcionalidad, usabilidad y rendimiento de cada una de los módulos creados. Es por ello que cada una de estas pruebas llevará consigo una gráfica que demuestre su correcto funcionamiento.

# Capítulo 14

# Casos de prueba

En este capítulo se expondrán una serie de casos de pruebas que corresponden con los descritos en el Capítulo 8. Al ser el uso del sistema por línea de comandos, muchos de los casos de uso expuestos en este capítulo podrán ser comprobados al mismo tiempo. Cada uno de los casos de uso, además, podrá servir como pruebas del sistema, ya que exponen todas las situaciones posibles que pueden darse a la hora de ejecutar el sistema.

Los casos de uso a utilizar para las pruebas son los siguientes:

- **CU-1.1:** Selección de datos de entrada para entrenamiento.
- **CU-1.2:** Selección de parámetros para entrenamiento.
- **CU-2.1:** Selección de datos de entrada para evaluación.
- **CU-2.2:** Selección de parámetros para evaluación.
- **CU-2.3:** Procesamiento de los resultados para evaluación.
- **CU-3.1:** Selección de datos de entrada para predicción.
- **CU-3.2:** Procesamiento de los resultados para predicción.

## 14.1. Caso de prueba CU-1.1

Este caso de prueba puede ser observado en la Tabla 14.1, y hace referencia al caso de uso expuesto en la Tabla 8.6.

| Caso | Descripción | Resultado |
|---|---|---|
| 1 | Introducción de un directorio y nombre de fichero válido | No se produce ningún error y los datos son almacenados correctamente |
| 2 | Introducción de rutas *datasets* y *train-test* | Error por inexistencia del fichero o directorio. Se resolvió poniendo una condición de inexistencia. |

Tabla 14.1: Caso de prueba 1.1

## 14.2. Caso de prueba CU-1.2

Este caso de prueba puede ser observado el la Tabla 14.2, y hace referencia al caso de uso expuesto en la Tabla 8.7

| Caso | Descripción | Resultado |
|---|---|---|
| 1 | Introducción de valores de parámetros válidos | El sistema no devuelve ningún error y se comienza con el entrenamiento de la red |
| 2 | Introducción del parámetro *optimizer* erróneo | El sistema devolvía una excepción. El problema se resolvió utilizando la sentencia *if* acorde a los tres valores posibles. |
| 3 | Introducción del parámetro *epochs*, *batch_size* o *learning_rate* erróneo | Este error se comete introduciendo un valor menor o igual que 0 para cualquiera de estos parámetros. El problema se solucionó introduciendo una sentencia *if*. |

Tabla 14.2: Caso de prueba 1.2

## 14.3. Caso de prueba CU-2.1

Este caso de prueba se puede observar en la Tabla 14.3, y hace referencia al caso de uso expuesto en la Tabla 8.8.

| Caso | Descripción | Resultado |
|---|---|---|
| 1 | Introducción de valores correctos | El sistema almacenará los valores introducidos por el usuario para su posterior evaluación |
| 2 | Gran tamaño de lote introducido | El sistema devolverá un error de exceso de memoria. No se ha podido solucionar ya que no está en mano del programador, si no de la potencia del servidor utilizado. |

Tabla 14.3: Caso de prueba 2.1

## 14.4. Caso de prueba CU-2.2

Este caso de prueba se puede observar en la Tabla 14.4, y hace referencia al caso de uso expuesto en la Tabla 8.9.

| Caso | Descripción | Resultado |
|---|---|---|
| 1 | Introducción de parámetros correctos | El sistema evaluará el modelo entrenado |
| 2 | Creación de un fichero sin directorio | El sistema devuelve un error de directorio no encontrado. Se soluciona automatizando la creación del directorio en caso de no existir. |

Tabla 14.4: Caso de prueba 2.2

## 14.5. Caso de prueba CU-2.3

Este caso de prueba se puede observar en la Tabla 14.5, y hace referencia al caso de uso expuesto en la Tabla 8.10.

| Caso | Descripción | Resultado |
|---|---|---|
| 1 | Procesamiento correcto | El sistema devolverá por pantalla el resultado obtenido por la evaluación |
| 2 | Formato incorrecto de imágenes | El sistema devolverá por pantalla un error. El problema se solucionó mostrando qué error se ha producido para que el usuario sepa la situación |

Tabla 14.5: Caso de prueba 2.3

## 14.6. Caso de prueba CU-3.1

Este caso de prueba se puede observar en la Tabla 14.6, y hace referencia al caso de uso expuesto en la Tabla 8.11.

| Caso | Descripción | Resultado |
|---|---|---|
| 1 | Selección de datos correctos | El sistema almacenará los datos introducidos por el usuario para su posterior predicción |
| 2 | Imágenes de distinto formato introducidas | El sistema devuelve un mensaje de error indicando que la imagen introducida no está en el formato correcto. |
| 3 | Fichero de pesos no válido | El sistema devolverá un mensaje de erro por la no existencia del fichero. Se solucionó introduciendo un mensaje de error más concreto para el conocimiento del usuario. |

Tabla 14.6: Caso de prueba 3.1

## 14.7. Caso de prueba CU-3.2

Este caso de prueba se puede observar en la Tabla 14.7, y hace referencia al caso de uso expuesto en la Tabla 8.12.

| Caso | Descripción | Resultado |
|---|---|---|
| 1 | Procesamiento de resultados | El sistema predecirá la etiqueta resultante de los datos introducidos. |
| 2 | Error en el formato de salida | El resultado es devuelto por el sistema en forma de vector. Este problema se solucionó mostrando sólo la primera posición del vector que almacenaba la etiqueta predicha. |

Tabla 14.7: Caso de prueba 3.2

Como vemos, estos casos de pruebas han recogido todas las posibles situaciones a las que nuestro sistema puede verse sometido, tanto el hecho de introducir datos erróneos, como el hecho de introducir datos válidos. Además, se ha explicado el funcionamiento normal del sistema ante cualquiera de los dos casos.

# Capítulo 15

# Resultados experimentales

En este capítulo mostraremos los distintos resultados obtenidos con el modelo de red neuronal residual profunda creada en este proyecto. El capítulo se dividirá en tres secciones, cada una representando el optimizador utilizado para entrenar el modelo. Cada sección se subdividirá según los tamaños del lote. Hemos decidido utilizar esta división ya que, tras la realización de unas pruebas iniciales realizando cambios sobre el valor del ratio de aprendizaje, observamos que nuestro modelo ofrecía siempre los mejores resultados para un valor de $10^{-6}$, además de considerar una paciencia de 10 épocas antes de considerar como finalizada la fase de entrenamiento. Es por esto por lo que sólo hemos considerado como factores el optimizador y el tamaño del lote.

Dentro de cada sección, los resultados se mostrarán de la siguiente manera:

1. Tabla representativa de los parámetros utilizados.
2. Figura que muestra la precisión en validación a lo largo del entrenamiento.
3. Figura que muestra la pérdida en validación a lo largo del entrenamiento.
4. Comentarios sobre las dos gráficas anteriores explicando las distintas características.
5. Figura comparativa entre la pérdida en validación y la pérdida en el entrenamiento.
6. Figura que representará la matriz de confusión conseguida para ese modelo.

7. Tabla que contendrá los valores de precisión y pérdida sobre el conjunto de test.

Además, mostraremos una tabla resumen en la que se podrá observar los resultados de pérdida y precisión utilizando el conjunto de test como evaluador. Finalmente, se mostrará la tabla comparativa de los mejores resultados obtenidos con las configuraciones siguientes y, los resultados obtenidos por el artículo [7].

## 15.1. Optimizador *Adam*

En esta sección se mostrarán los resultados conseguidos tras utilizar el optimizador *Adam* a la hora de la compilación del modelo.

### 15.1.1. Tamaño del lote: 3

Para esta configuración, el número de épocas ejecutadas, antes de dar como finalizado el entrenamiento, ha sido **78** épocas. La gráfica que representa la precisión a lo largo de las épocas se muestra en la Figura 15.1. Del mismo modo, la pérdida de nuestro modelo a lo largo del entrenamiento se muestra en la Figura 15.2.

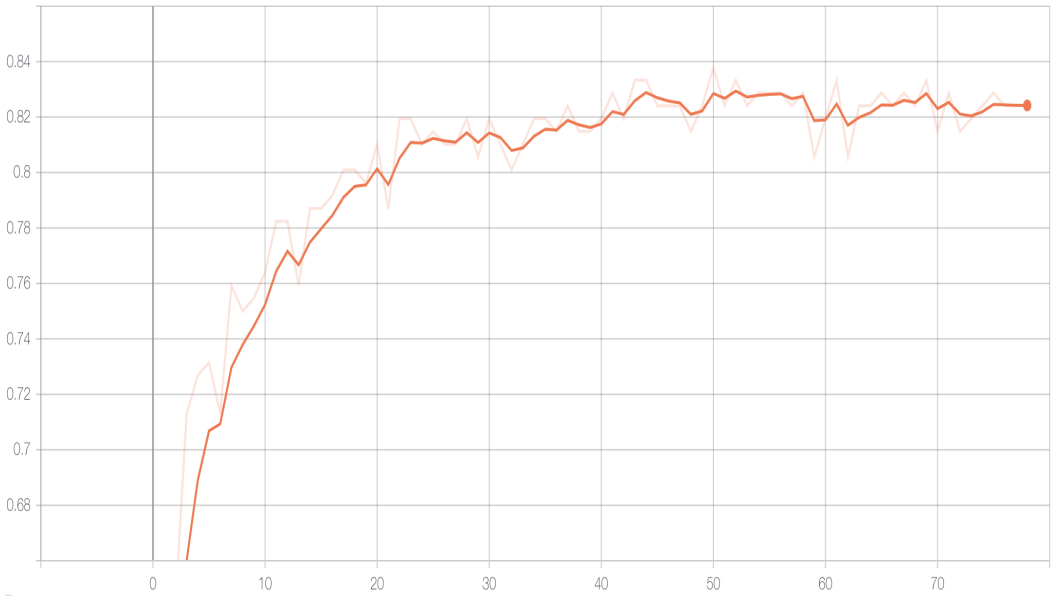


Figura 15.1: Precisión a lo largo del entrenamiento con el optimizador *Adam* para un tamaño de lote de 3

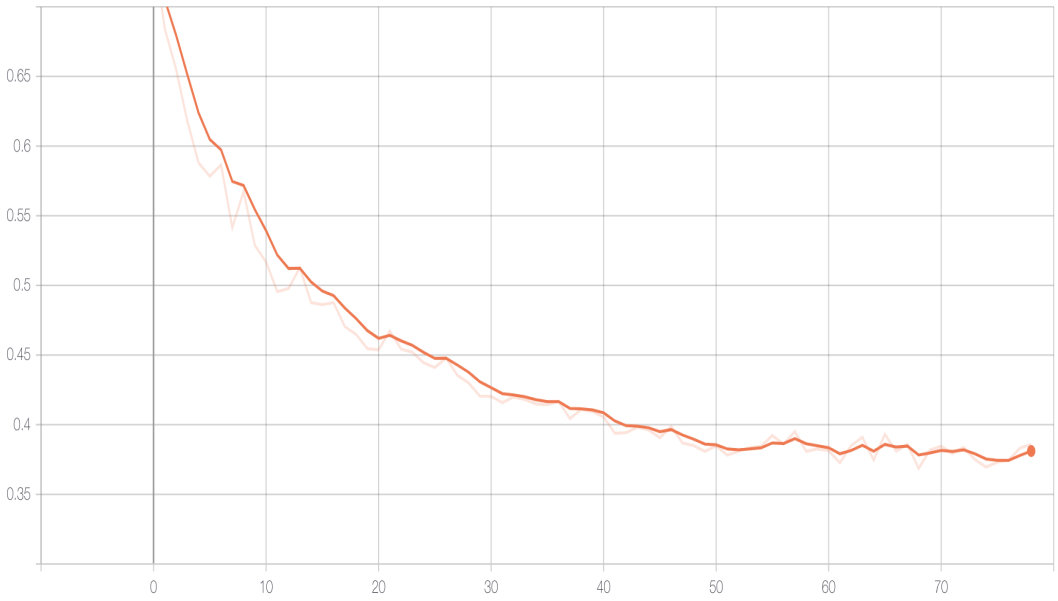


Figura 15.2: Pérdida a lo largo del entrenamiento con el optimizador *Adam* para un tamaño de lote de 3

Como podemos observar, conseguimos un buen resultado en cuanto a valor de precisión (mayor de 0.8). Además, al observar la validación, la pérdida que obtenemos tampoco es mala, siendo menor que 0.45. Seguidamente, como se puede observar en la Figura 15.3, el conjunto de entrenamiento supone un menor valor de pérdida, con lo que estamos ante una situación en la que el modelo no es capaz de generalizar correctamente. Es por ello por lo que el valor de la pérdida en validación es mucho mayor que el conseguido con el conjunto de entrenamiento.

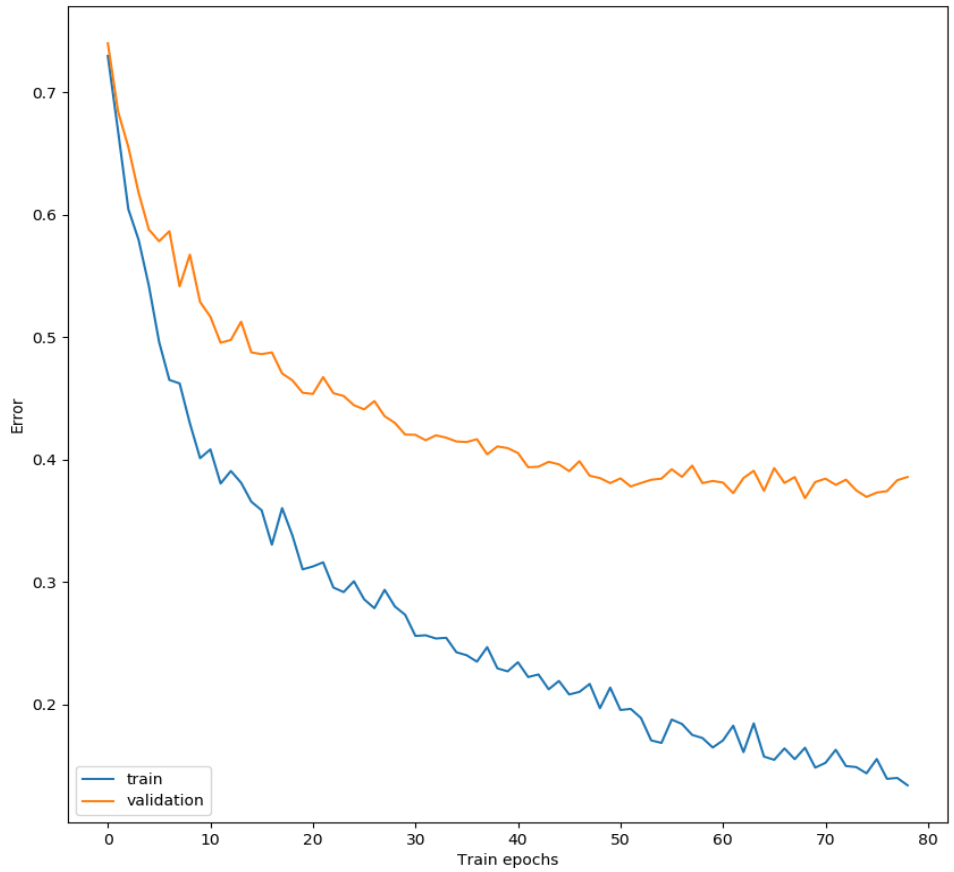


Figura 15.3: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *Adam* junto con un tamaño del lote de 3

A continuación, se mostrará la matriz de confusión conseguida con este modelo (15.4).

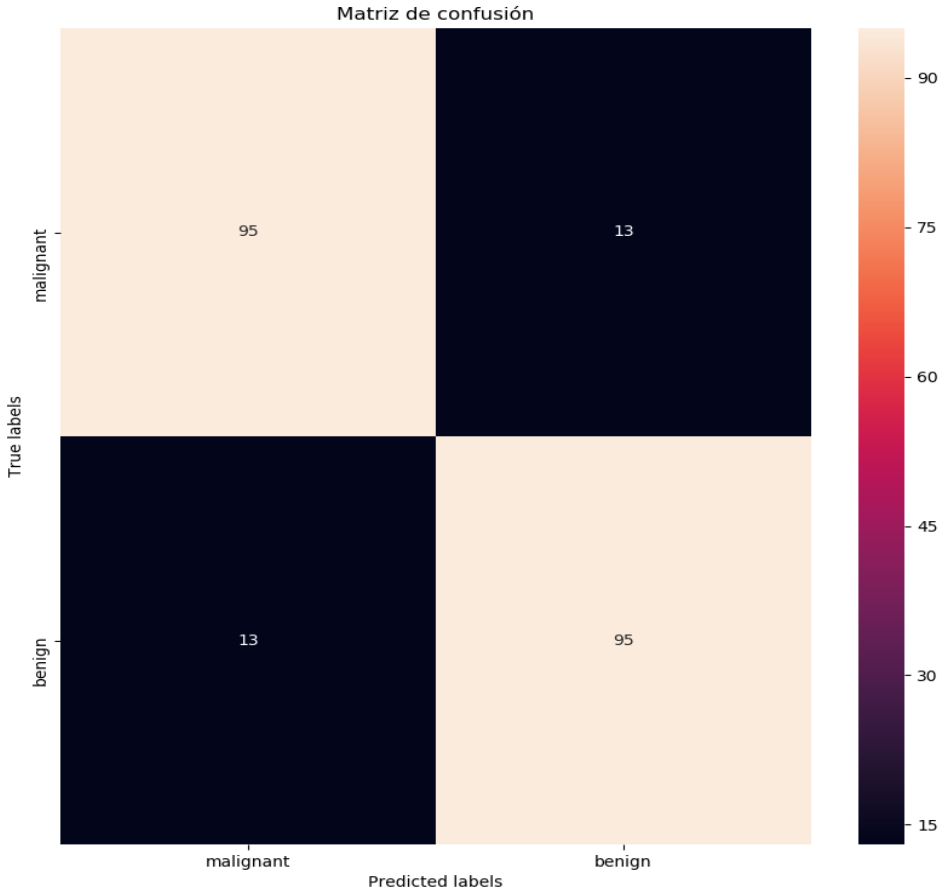


Figura 15.4: Matriz de confusión conseguida con el optimizador *Adam* para un tamaño de lote de 3

Una vez mostrados todos los datos correspondientes al entrenamiento, es hora de mostrar los resultados conseguidos evaluando el conjunto de test con el presente modelo. El resultado se puede observar en la Tabla 15.1.

| | **Valor** |
|---|---|
| **Precisión** | 87,9629 % |
| **Pérdida** | 0,3182 |

Tabla 15.1: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *Adam* y un tamaño del lote de 3

### 15.1.2. Tamaño del lote: 5

Para esta configuración, el número de épocas totales ejecutadas ha sido de **93** épocas. La gráfica correspondiente a la precisión de validación y entrenamiento a lo largo de las épocas se puede observar en la Figura 15.5. De la misma manera, para el valor de pérdida a lo largo del entrenamiento para validación y entrenamiento se muestra en la Figura 15.6.

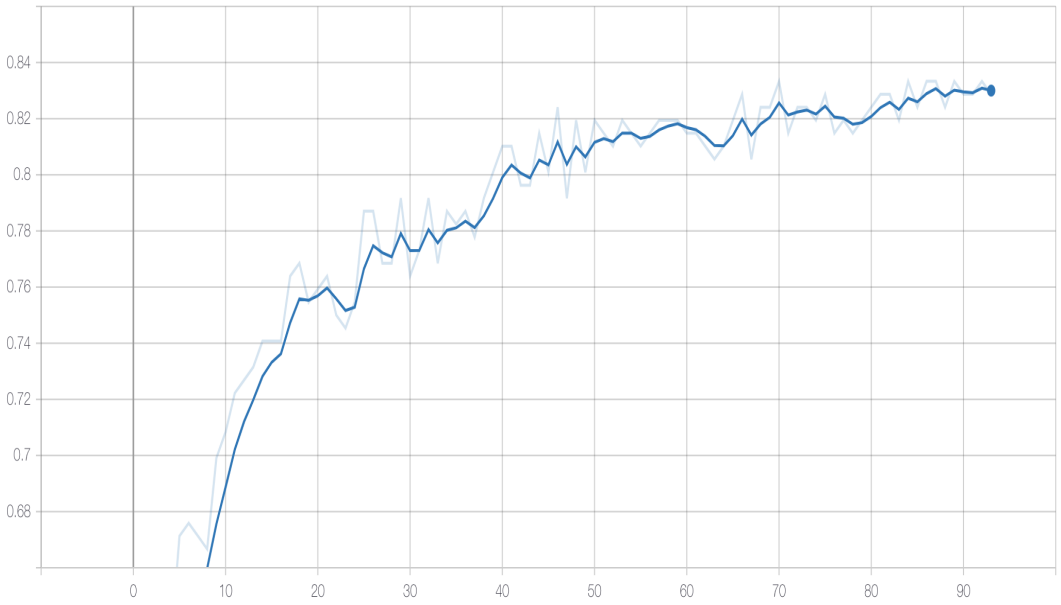


Figura 15.5: Precisión a lo largo del entrenamiento con el optimizador *Adam* para un tamaño de lote de 5

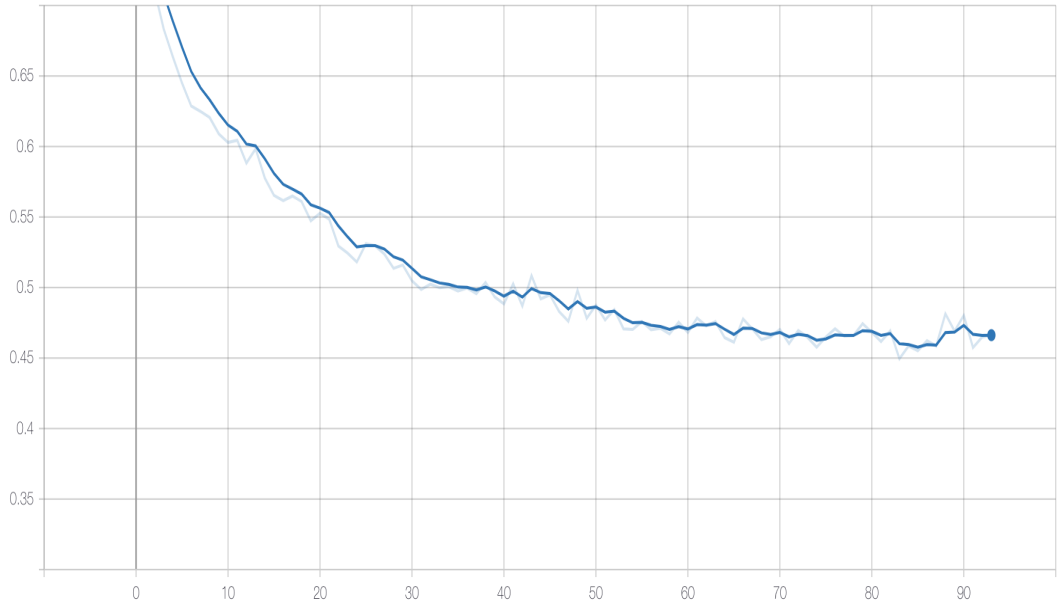


Figura 15.6: Pérdida a lo largo del entrenamiento con el optimizador *Adam* para un tamaño de lote de 5

Como podemos observar en las dos anteriores figuras, los resultados son muy parecidos a los que hemos obtenido utilizando un tamaño de lote de 3, pero, a diferencia de éste, cuando hemos utilizado un tamaño de lote de 5, el valor de pérdida para el conjunto de validación es mucho mayor, siendo éste mayor que 0.45. A continuación, en la Figura 15.7, se muestra una comparación entre la pérdida del conjunto de entrenamiento con el conjunto de validación.

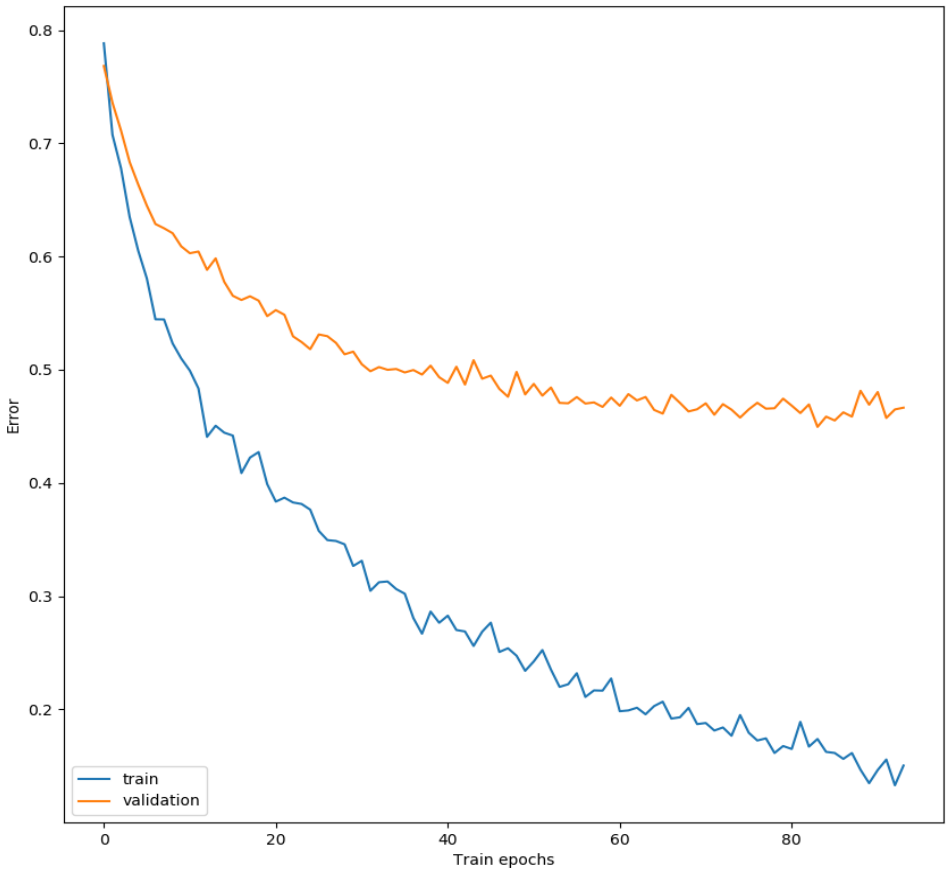


Figura 15.7: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *Adam* junto con un tamaño del lote de 5

A continuación, se mostrará la matriz de confusión conseguida con este modelo (ver 15.8).

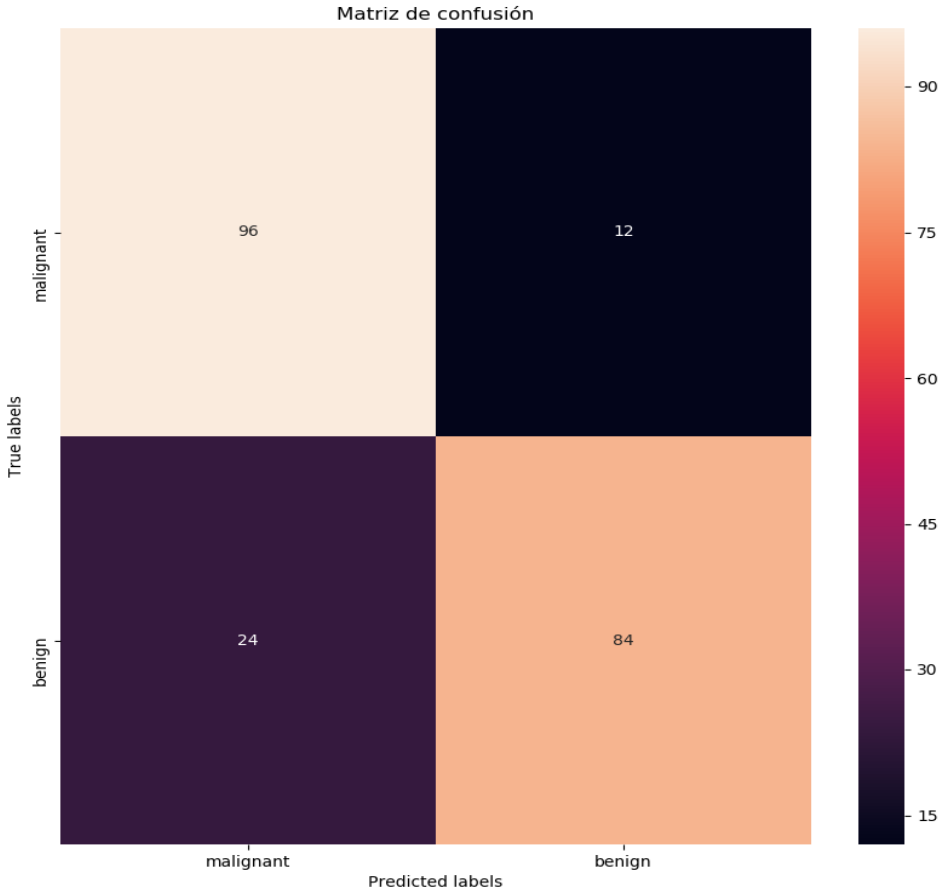


Figura 15.8: Matriz de confusión conseguida con el optimizador *Adam* para un tamaño de lote de 5

Finalmente, en la Tabla 15.2, mostraremos cuales han sido los resultados de precisión y de pérdida al utilizar esta configuración junto con el conjunto de test.

| | **Valor** |
|---|---|
| **Precisión** | 83,3333 % |
| **Pérdida** | 0,3884 |

Tabla 15.2: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *Adam* y un tamaño del lote de 5

### 15.1.3. Tamaño del lote: 7

Con esta configuración, el número de épocas totales ejecutadas ha sido de **55** épocas. Seguidamente, se puede observar, Figura 15.9, la precisión del modelo con el conjunto de validación. En la Figura 15.10 se muestra, igualmente, la pérdida del modelo con el conjunto de validación durante la fase de entrenamiento.

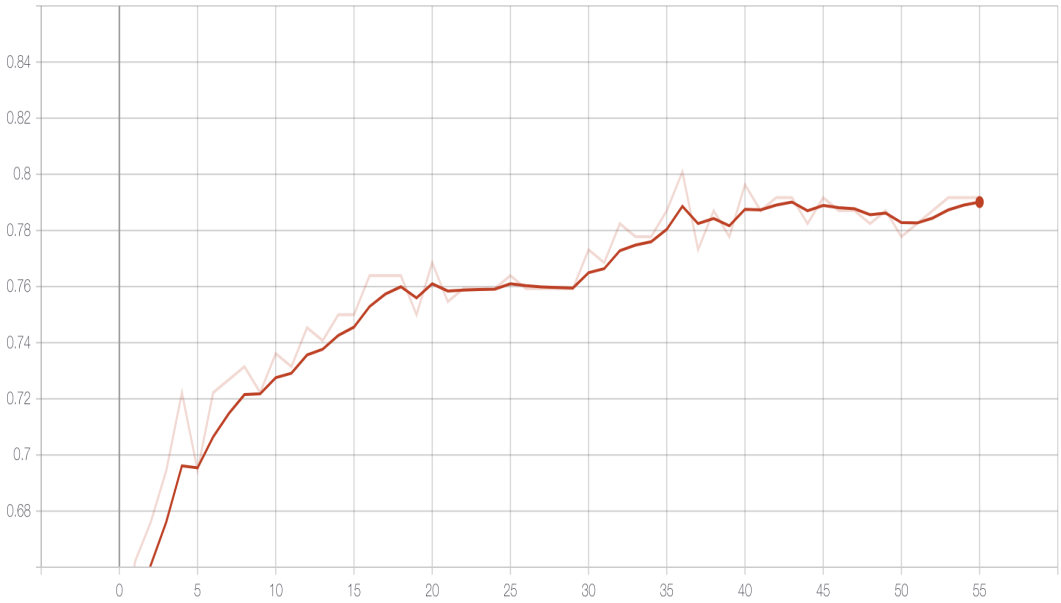


Figura 15.9: Precisión a lo largo del entrenamiento con el optimizador *Adam* para un tamaño de lote de 7

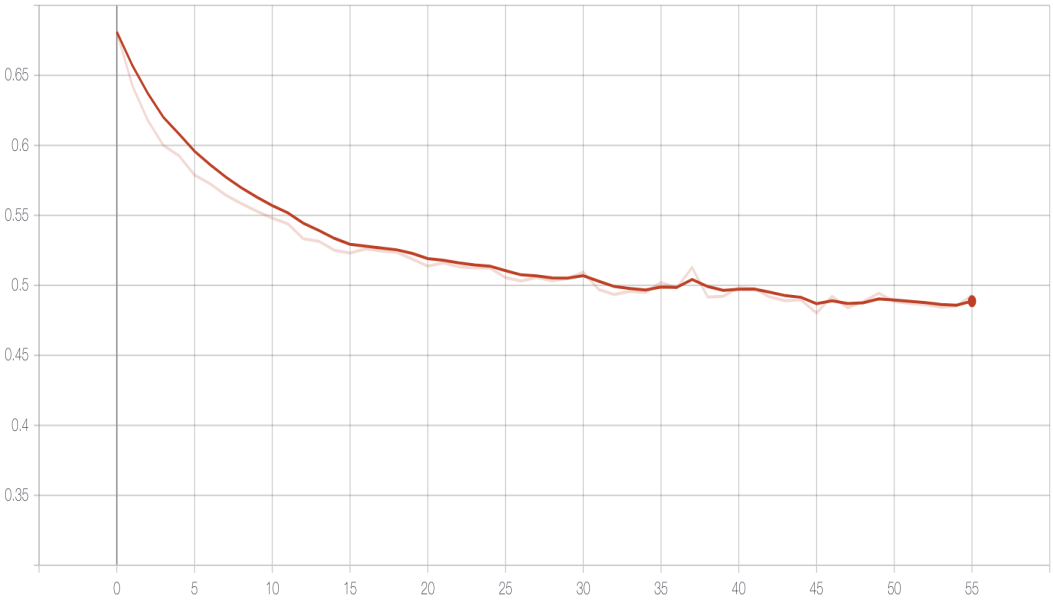


Figura 15.10: Pérdida a lo largo del entrenamiento con el optimizador *Adam* para un tamaño de lote de 7

Esta configuración no es muy exitosa, ya que no se consigue llegar ni a un 0.8 de precisión en el conjunto de validación. Además de esto, la pérdida durante el entrenamiento no es capaz de bajar del 0.45, con lo que podemos decir que esta configuración es la peor hasta el momento. Seguidamente, se muestra una comparativa entre la pérdida del conjunto de entrenamiento con el conjunto de validación durante toda la fase de entrenamiento (ver 15.11).

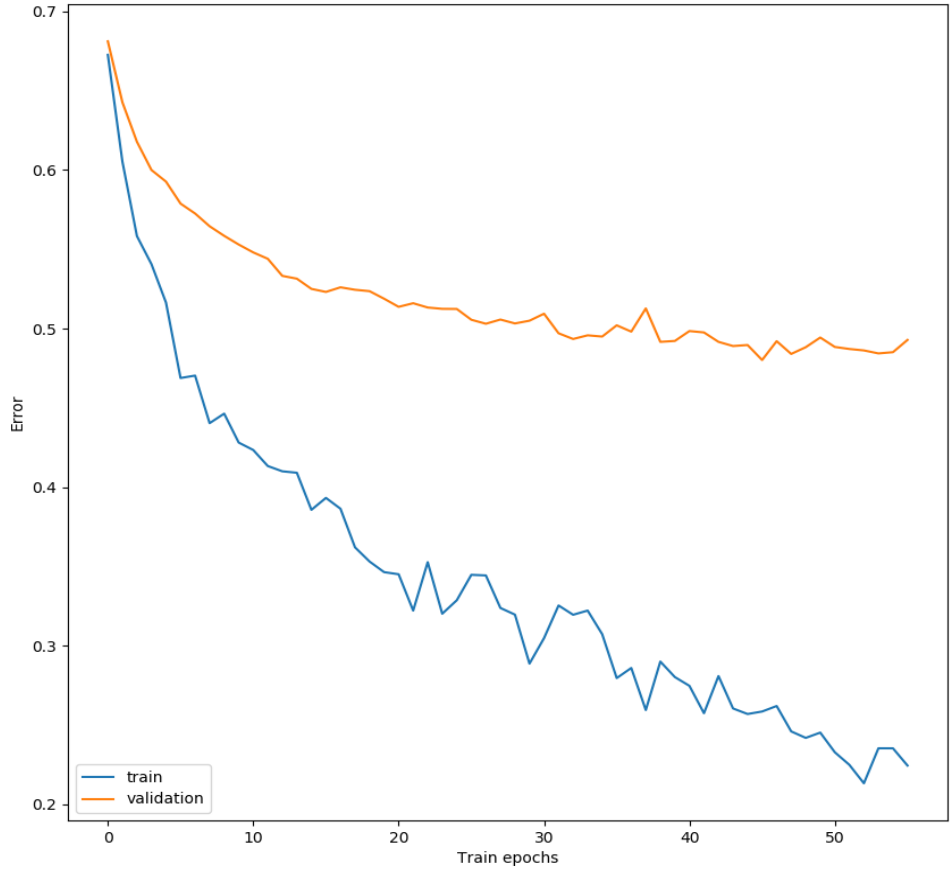


Figura 15.11: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *Adam* junto con un tamaño del lote de 7

A continuación, se mostrará la matriz de confusión conseguida con este modelo (ver 15.12).

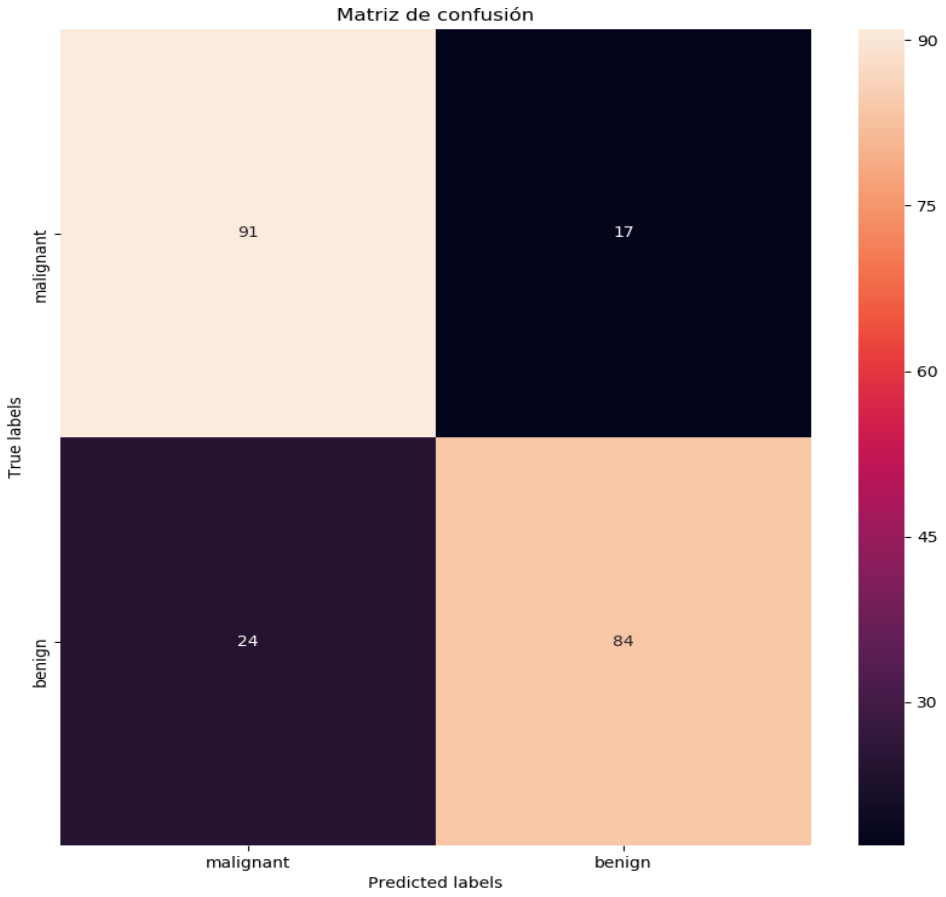


Figura 15.12: Matriz de confusión conseguida con el optimizador *Adam* para un tamaño de lote de 7

Finalmente, una vez mostrados los datos relativos al entrenamiento del modelo y su evaluación, pasaremos a mostrar los resultados de precisión y pérdida evaluando el modelo con el conjunto de test. Estos resultados se pueden observar en la Tabla 15.3.

| | **Valor** |
|---|---|
| **Precisión** | 81,0185 % |
| **Pérdida** | 0,4082 |

Tabla 15.3: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *Adam* y un tamaño del lote de 7

### 15.1.4. Tamaño del lote: 10

Al utilizar esta configuración, el número de épocas totales ejecutadas ha sido de **57** épocas. Seguidamente se mostrará la precisión del modelo con el conjunto de validación durante toda la fase de entrenamiento (ver 15.13). De igual manera, en la Figura 15.14 se muestra la pérdida del modelo con el conjunto de validación durante la fase de entrenamiento con esta configuración.

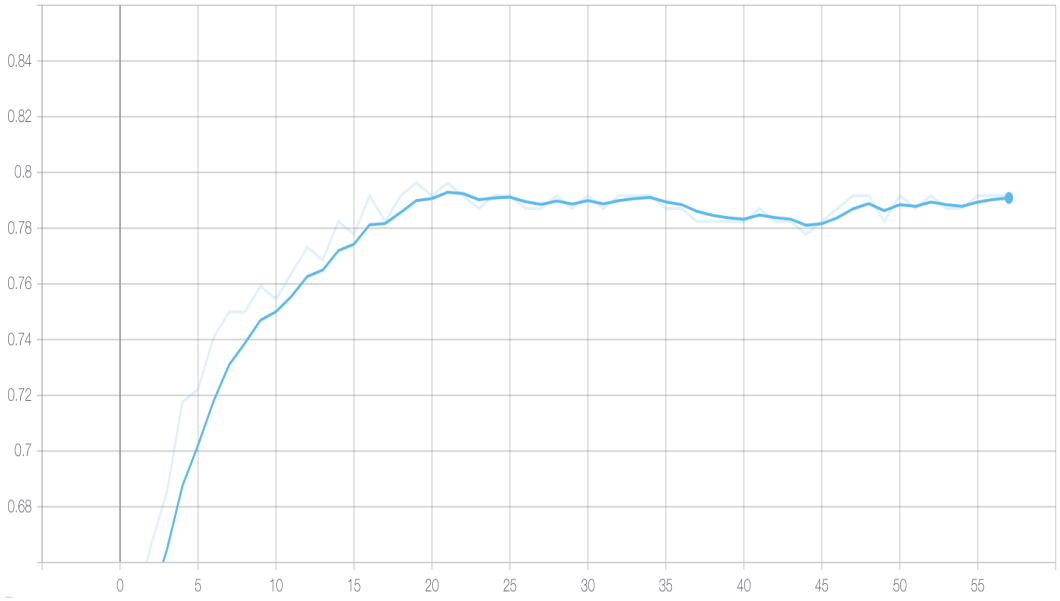


Figura 15.13: Precisión a lo largo del entrenamiento con el optimizador *Adam* para un tamaño de lote de 10

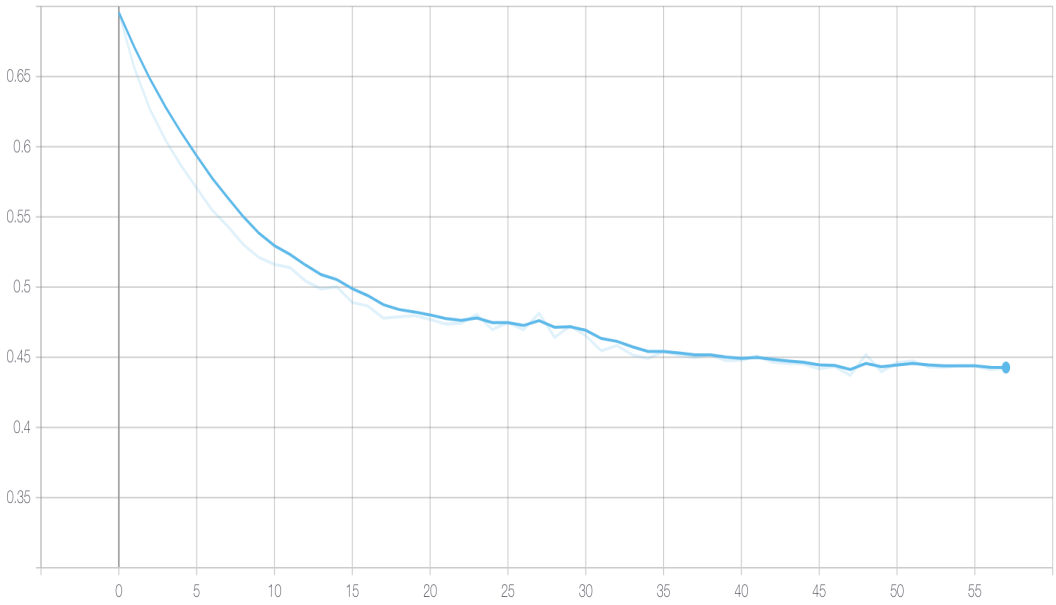


Figura 15.14: Pérdida a lo largo del entrenamiento con el optimizador *Adam* para un tamaño de lote de 10

Como se puede observar, con esta configuración conseguimos mejores resultados, siempre hablando de la validación, que con un tamaño del lote de 7. Aún así, estos resultados no son muy buenos ya que no se supera una precisión del 0.8 ni tampoco se baja de una pérdida del 0.4 en toda la fase de entrenamiento. A continuación, en la Figura 15.15, se muestra una comparativa de la pérdida del conjunto de validación con el conjunto de entrenamiento.

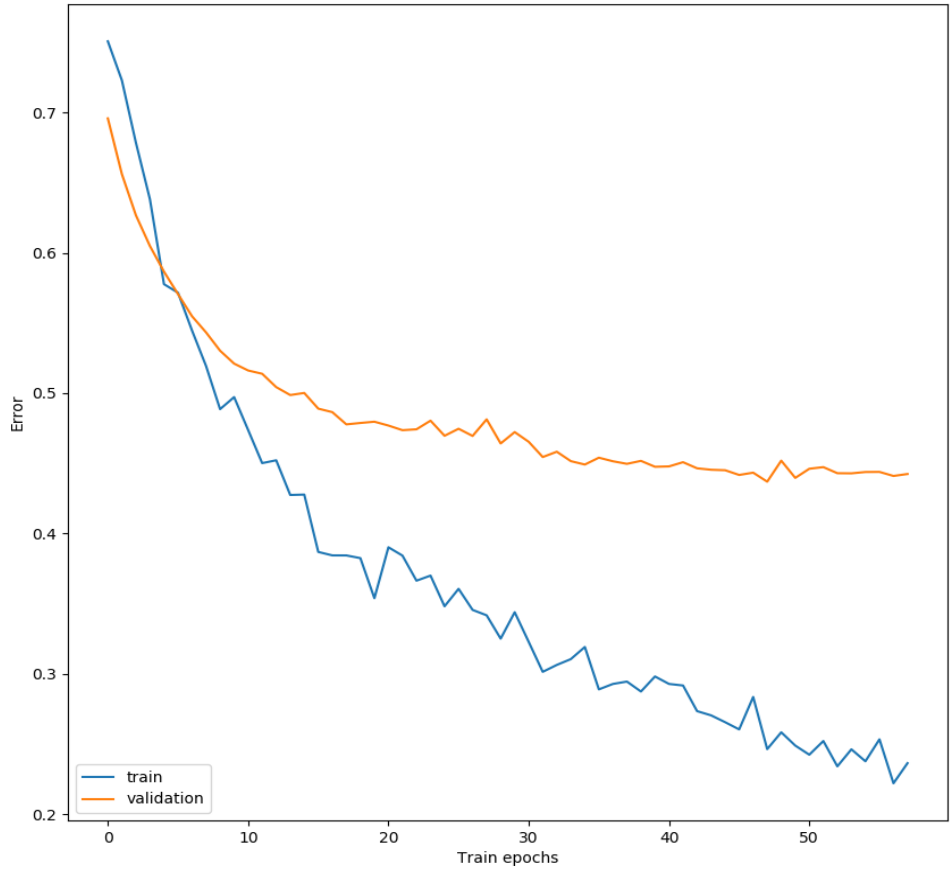


Figura 15.15: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *Adam* junto con un tamaño del lote de 10

A continuación, se mostrará la matriz de confusión conseguida con este modelo (ver 15.16).

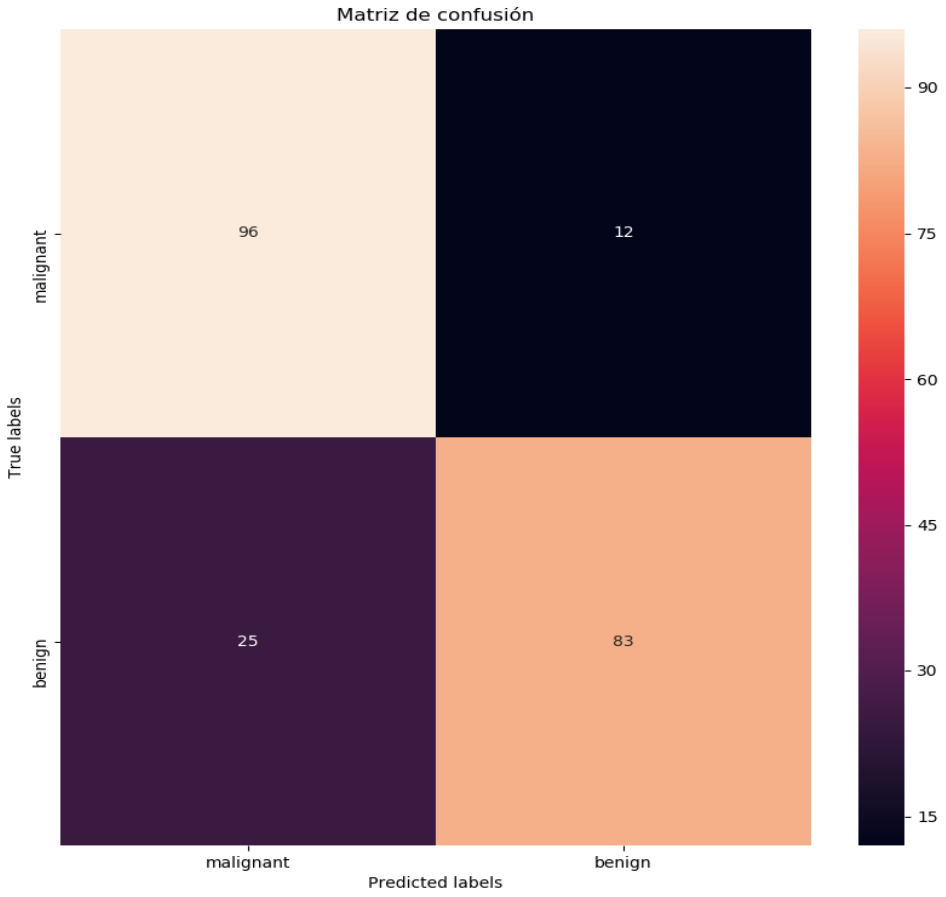


Figura 15.16: Matriz de confusión conseguida con el optimizador *Adam* para un tamaño de lote de 10

Finalmente, en la Tabla 15.4 se puede observar los resultados de precisión y pérdida, obtenidos con esta configuración, al evaluar el modelo con el conjunto de test.

| | **Valor** |
|---|---|
| **Precisión** | 82,8703 % |
| **Pérdida** | 0,3927 |

Tabla 15.4: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *Adam* y un tamaño del lote de 10

## 15.2. Optimizador *RMSprop*

En esta sección se mostrarán los resultados conseguidos tras utilizar el optimizador *RMSprop* a la hora de compilar el modelo.

### 15.2.1. Tamaño del lote: 3

El número de épocas totales ejecutadas para esta configuración ha sido de **19** épocas. Seguidamente, en la Figura 15.17, se muestra la precisión obtenida a lo largo del entrenamiento con el conjunto de validación. Además, en la Figura 15.18, se puede observar la pérdida a lo largo del entrenamiento para el conjunto de validación.

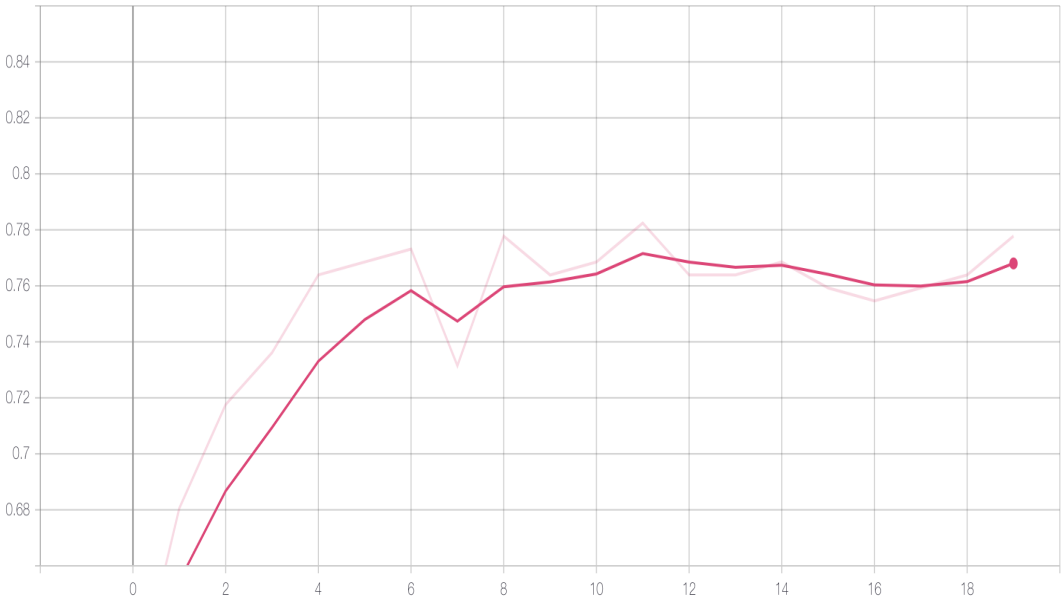


Figura 15.17: Precisión a lo largo del entrenamiento con el optimizador *RMSprop* para un tamaño de lote de 3

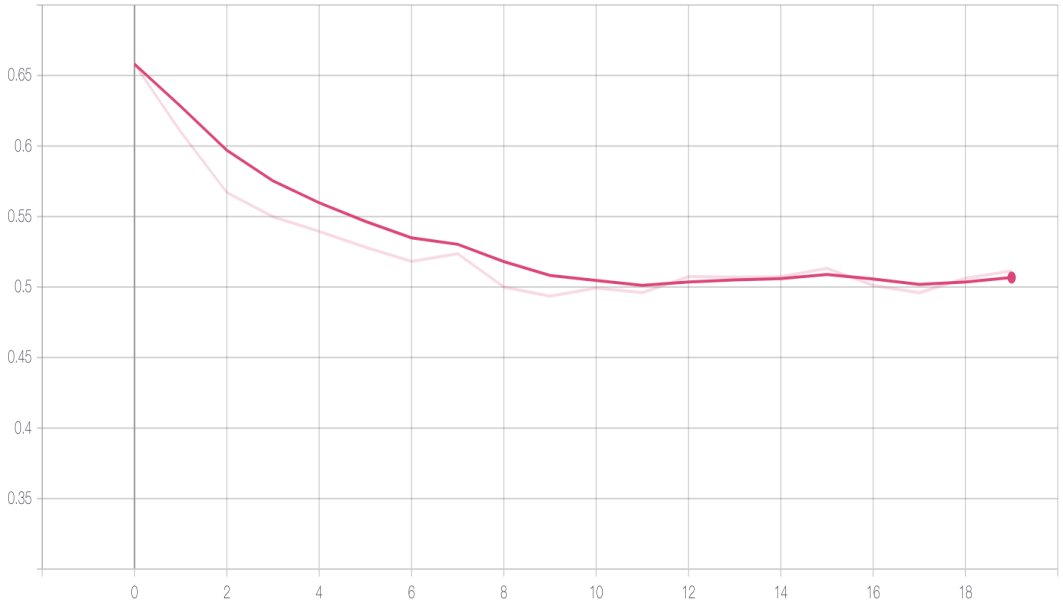


Figura 15.18: Pérdida a lo largo del entrenamiento con el optimizador *RMSprop* para un tamaño de lote de 3

Se puede observar que esta configuración es la peor hasta ahora, ya que no se consigue superar el 0.78 de precisión en la validación. Además, si nos fijamos en la pérdida del modelo, podemos ver que en la iteración 18 comienza a aumentar, siendo esto un síntoma de sobre-aprendizaje. Seguidamente, se muestra una figura comparativa entre las pérdidas durante el entrenamiento de los conjuntos de entrenamiento y de validación (ver 15.19).

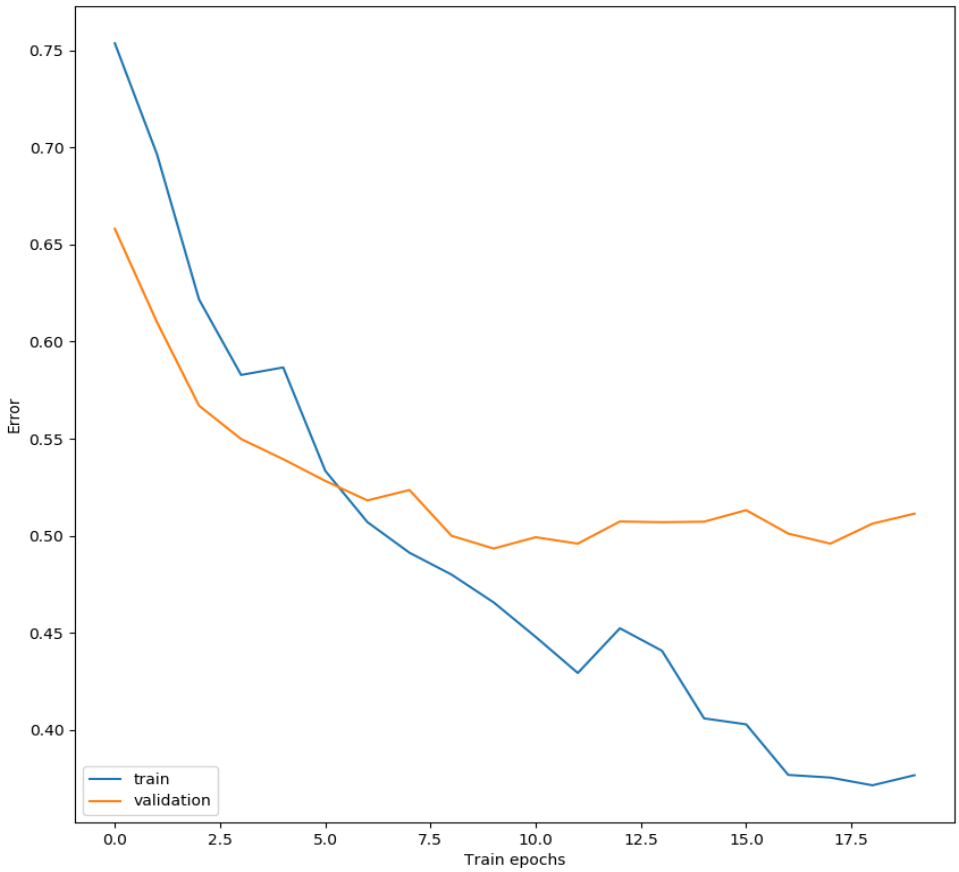


Figura 15.19: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *RMSprop* junto con un tamaño del lote de 3

A continuación, se mostrará la matriz de confusión conseguida con este modelo. Ver Figura 15.20.

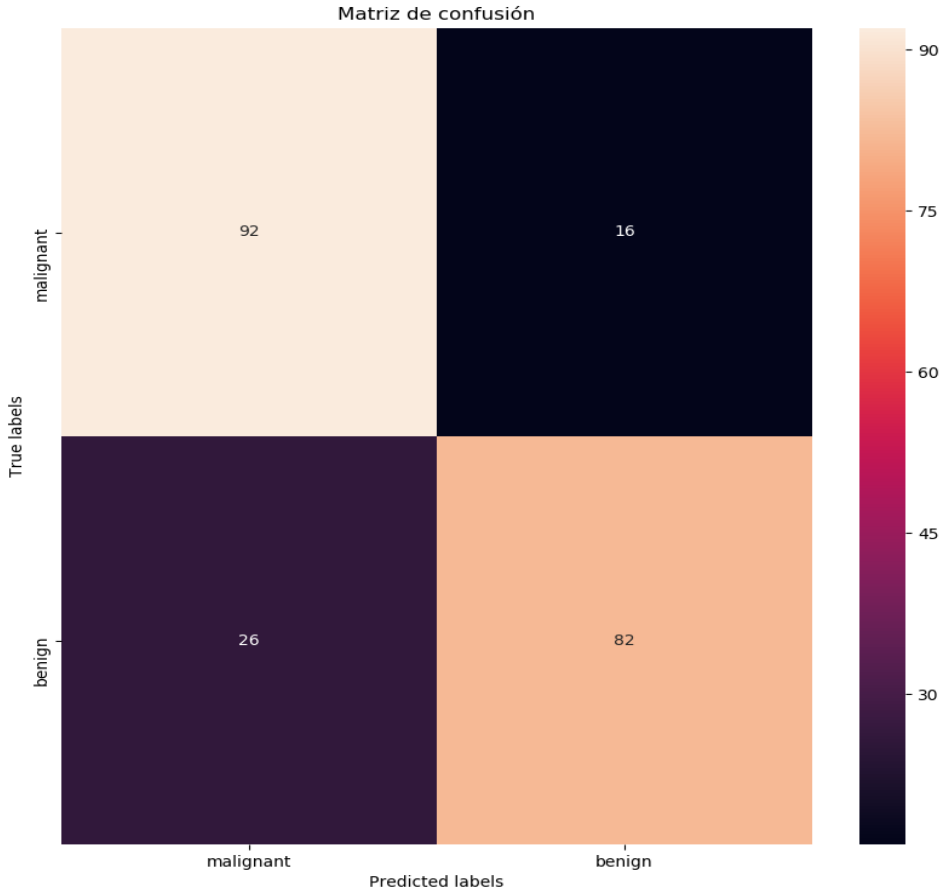


Figura 15.20: Matriz de confusión conseguida con el optimizador *RMSprop* para un tamaño de lote de 3

Finalmente, mostraremos los resultados de precisión y validación del modelo con esta configuración utilizando el conjunto de test para su evaluación. Estos resultados se pueden observar en la Tabla 15.5.

| | **Valor** |
|---|---|
| **Precisión** | 80,5555 % |
| **Pérdida** | 0,4436 |

Tabla 15.5: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *RMSprop* y un tamaño del lote de 3

### 15.2.2. Tamaño del lote: 5

El número de épocas totales ejecutadas para esta configuración ha sido de **19** épocas. Seguidamente, en la Figura 15.21, se muestra la precisión obtenida a lo largo del entrenamiento con el conjunto de validación. Además, en la Figura 15.22, se puede observar la pérdida a lo largo del entrenamiento para el conjunto de validación.

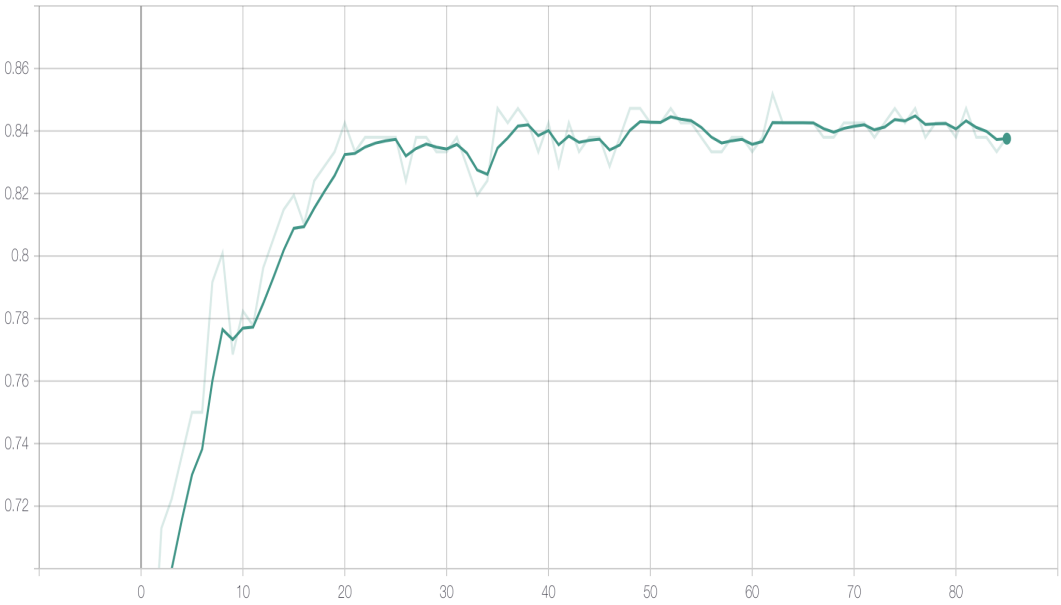


Figura 15.21: Precisión a lo largo del entrenamiento con el optimizador *RMSprop* para un tamaño de lote de 5

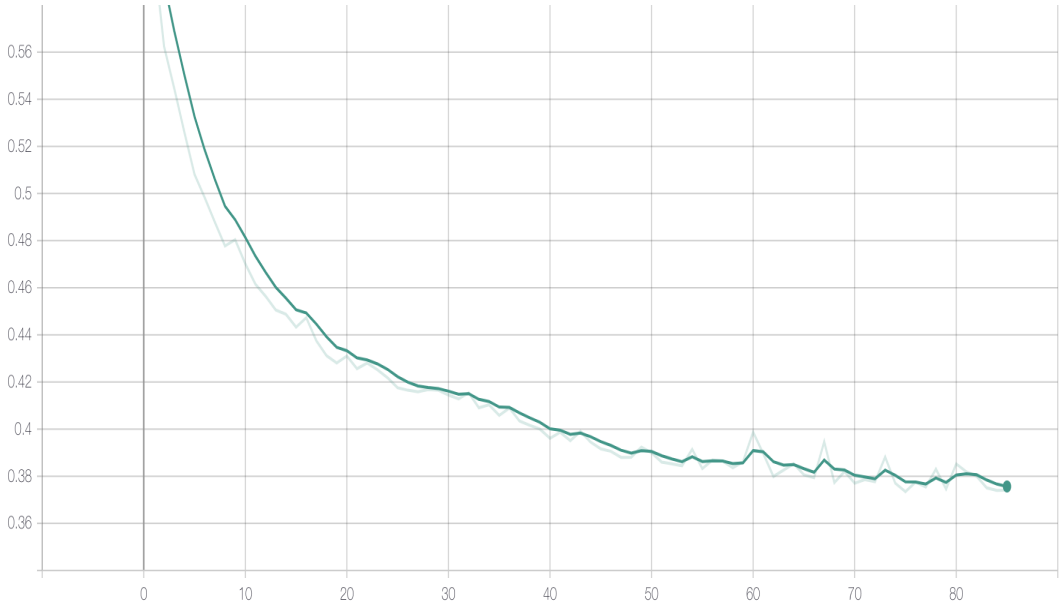


Figura 15.22: Pérdida a lo largo del entrenamiento con el optimizador *RMSprop* para un tamaño de lote de 5

Como vemos en las dos Figuras anteriores, Figuras 15.21 y 15.22, hemos

conseguidos los mejores resultados hasta ahora, ya que hemos conseguido llegar por encima del 0.85 de precisión sobre el conjunto de validación y además, la pérdida hemos conseguido reducirla. Seguidamente, podremos observar una comparativa entre la pérdida en entrenamiento y validación durante la fase de ejecución (ver 15.23).

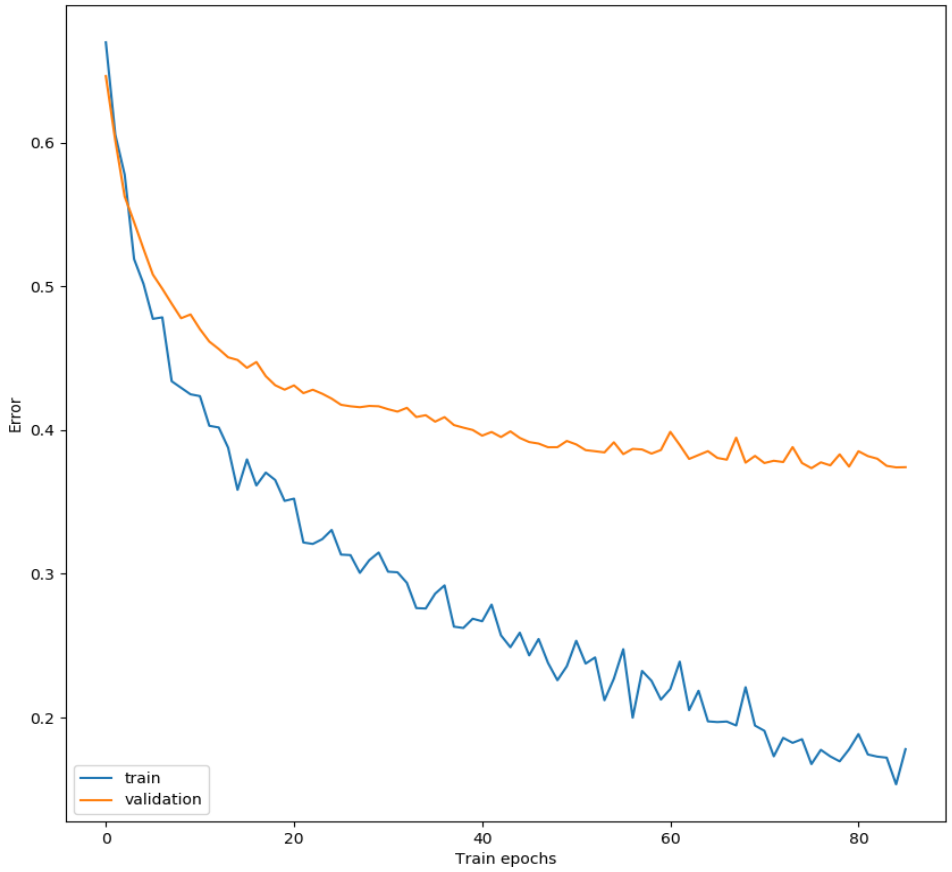


Figura 15.23: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *RMSprop* junto con un tamaño del lote de 5

A continuación, se mostrará la matriz de confusión conseguida con este modelo (ver 15.24).

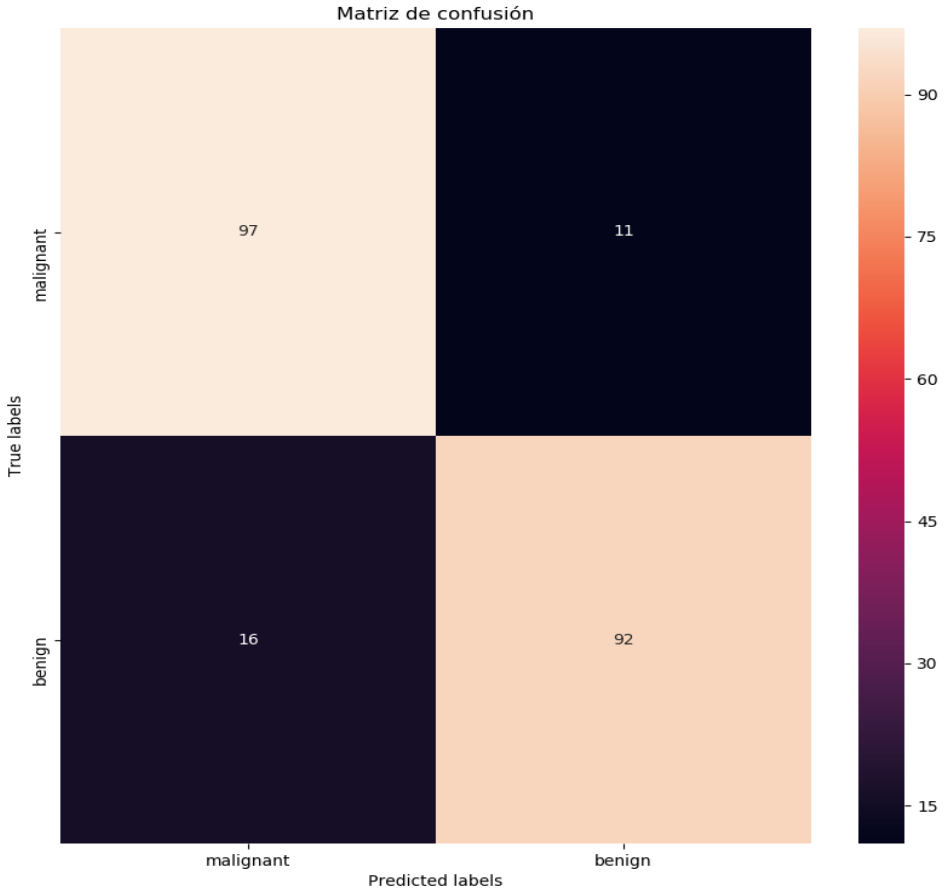


Figura 15.24: Matriz de confusión conseguida con el optimizador *RMSprop* para un tamaño de lote de 5

Finalmente, mostraremos los resultados de precisión y validación del modelo con esta configuración utilizando el conjunto de test para su evaluación. Estos resultados se pueden observar en la Tabla 15.6.

| | **Valor** |
|---|---|
| **Precisión** | 87'5 % |
| **Pérdida** | 0,3276 |

Tabla 15.6: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *RMSprop* y un tamaño del lote de 5

### 15.2.3. Tamaño del lote: 7

Con esta configuración, el número total de épocas ejecutadas ha sido de **38**. En la Figura 15.25, se muestra la precisión obtenida a lo largo del entrenamiento con el conjunto de validación. Además, en la Figura 15.26, se puede observar la pérdida a lo largo del entrenamiento para el conjunto de validación.

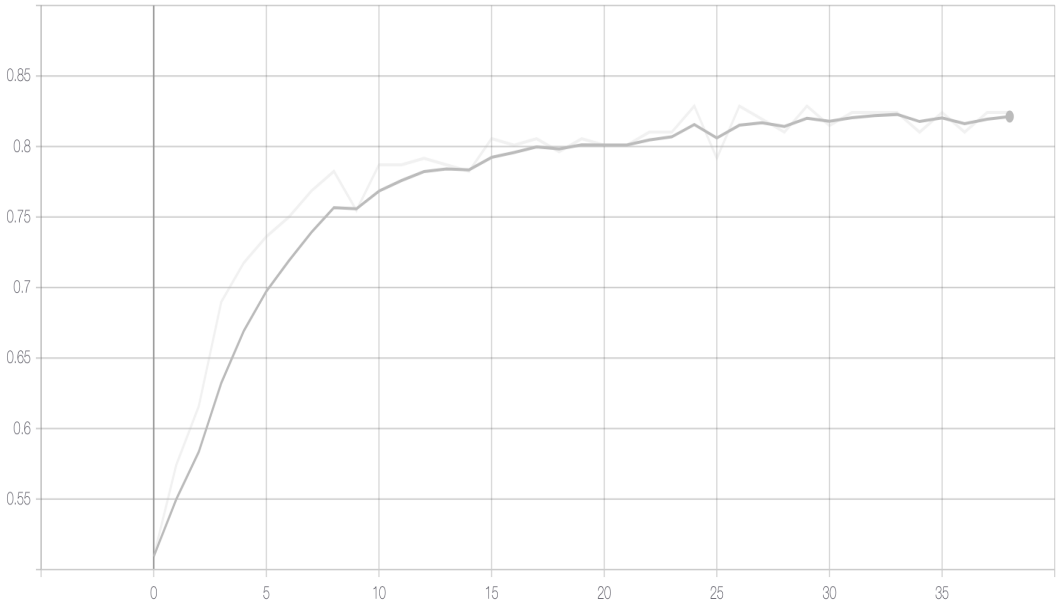


Figura 15.25: Precisión a lo largo del entrenamiento con el optimizador *RMSprop* para un tamaño de lote de 7

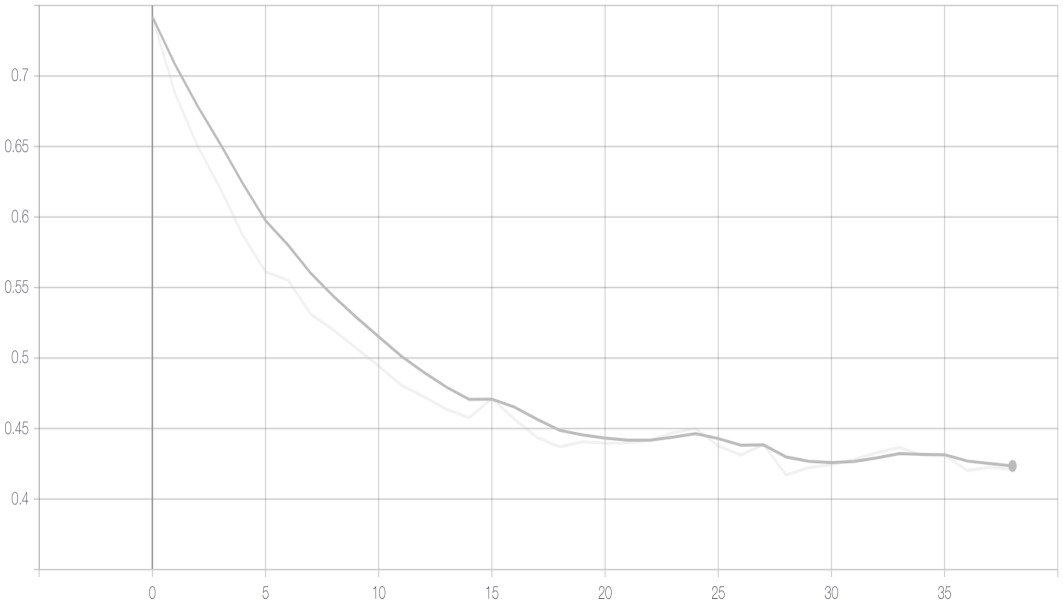


Figura 15.26: Pérdida a lo largo del entrenamiento con el optimizador *RMSprop* para un tamaño de lote de 7

Como vemos en las anteriores figuras, con esta configuración conseguimos buenos datos para la validación, llegando hasta 0.82 en predicción. En cambio, en relación a la pérdida del modelo, no hemos conseguido reducirla menos de 0.4. A continuación, mostraremos una comparativa entre la predicción y la pérdida durante la fase de entrenamiento (ver 15.27).

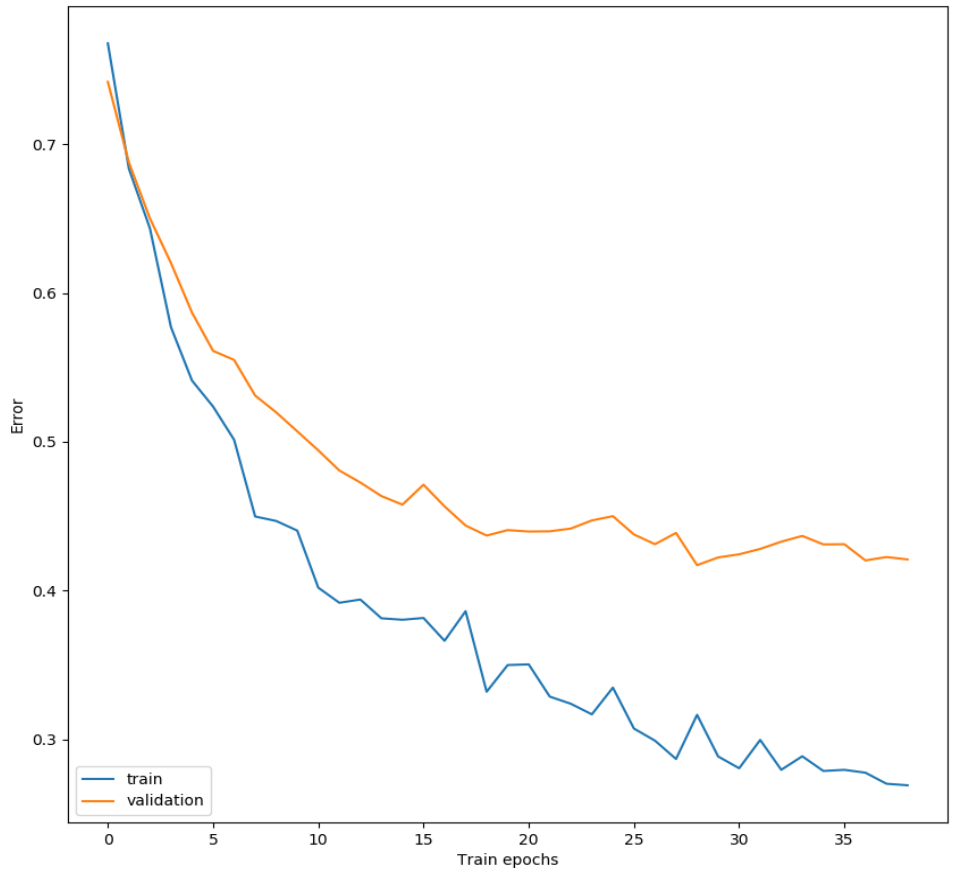


Figura 15.27: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *RMSprop* junto con un tamaño del lote de 7

A continuación, se mostrará la matriz de confusión conseguida con este modelo (ver 15.28).

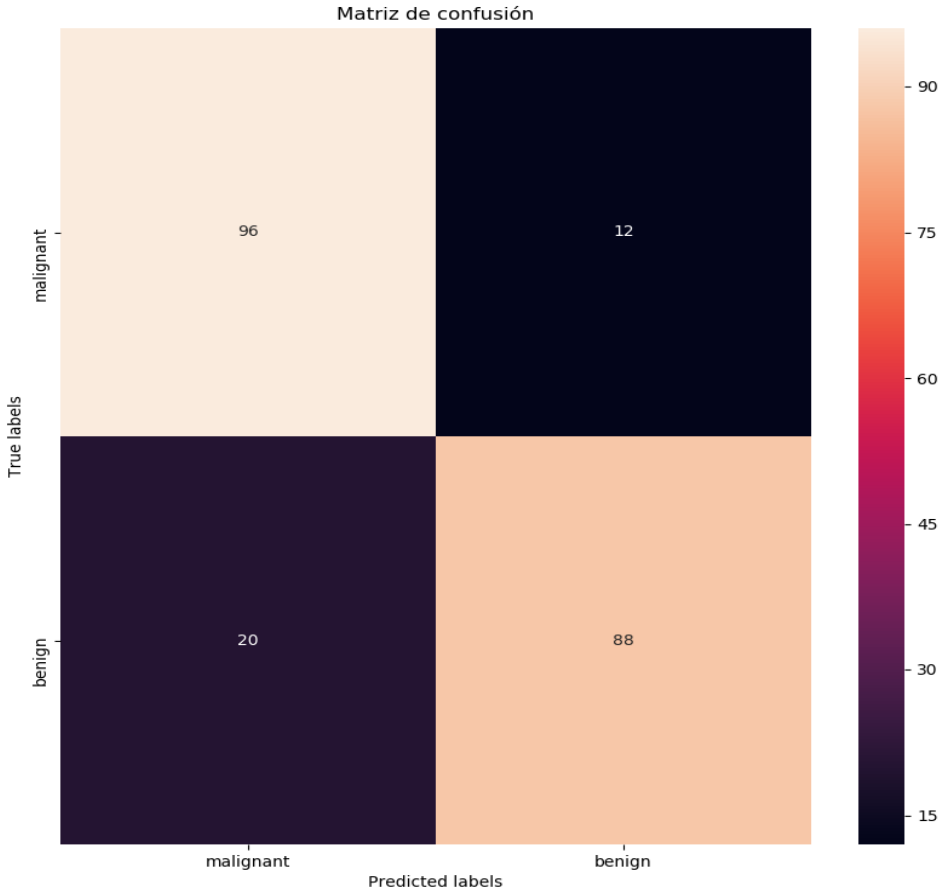


Figura 15.28: Matriz de confusión conseguida con el optimizador *RMSprop* para un tamaño de lote de 7

Finalmente, mostraremos los resultados de precisión y validación del modelo con esta configuración utilizando el conjunto de test para su evaluación. Estos resultados se pueden observar en la Tabla 15.7.

| | **Valor** |
|---|---|
| **Precisión** | 85'1851 % |
| **Pérdida** | 0,3652 |

Tabla 15.7: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *RMSprop* y un tamaño del lote de 7

### 15.2.4. Tamaño del lote: 10

Con esta configuración, el número total de épocas ejecutadas ha sido de **60**. En la Figura 15.29, se muestra la precisión obtenida a lo largo del entrenamiento con el conjunto de validación. Además, en la Figura 15.30, se puede observar la pérdida a lo largo del entrenamiento para el conjunto de validación.

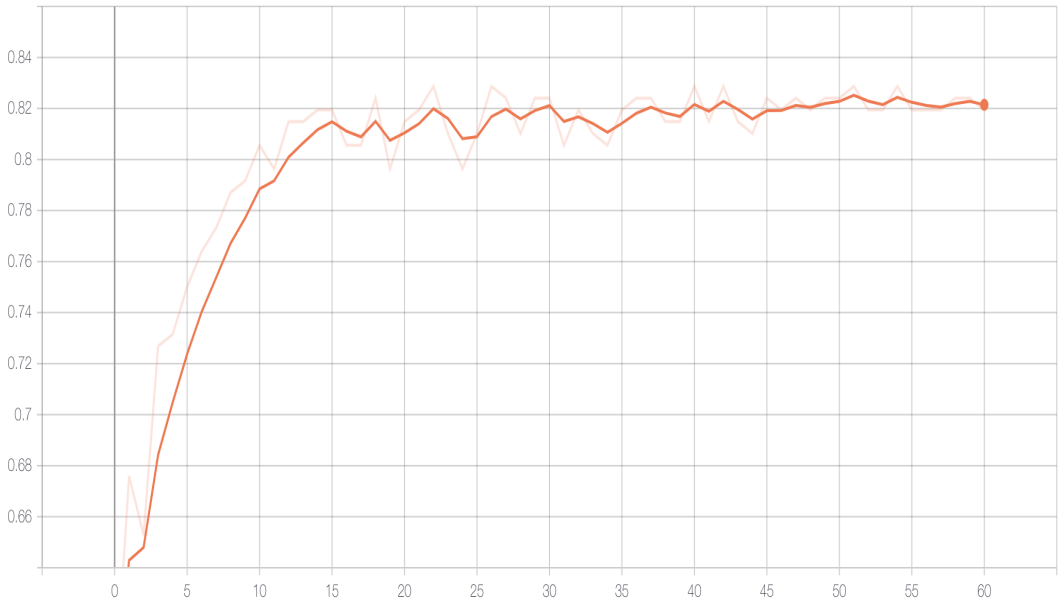


Figura 15.29: Precisión a lo largo del entrenamiento con el optimizador *RMSprop* para un tamaño de lote de 10

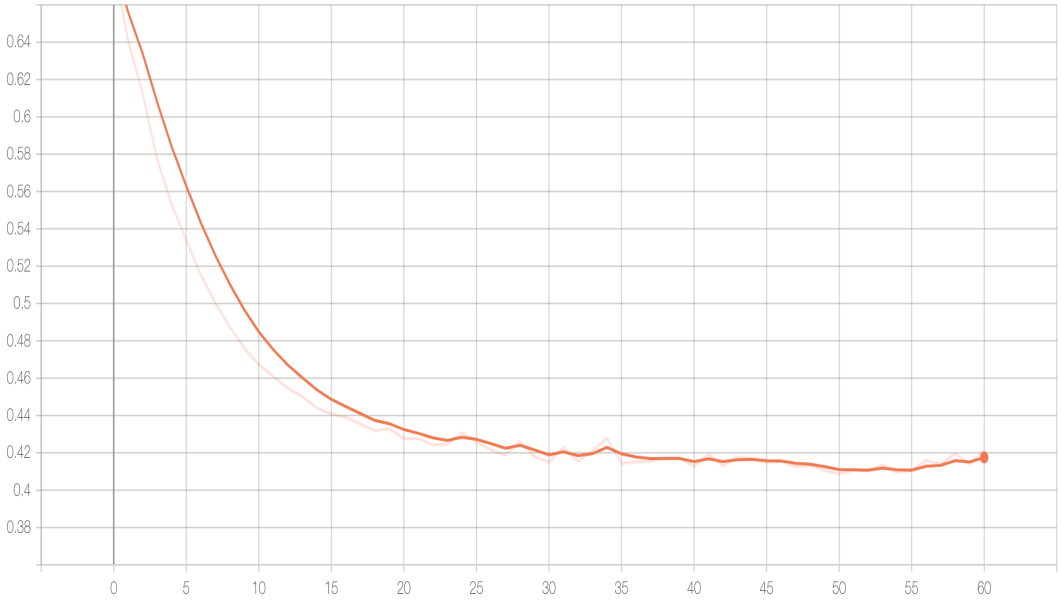


Figura 15.30: Pérdida a lo largo del entrenamiento con el optimizador *RMSprop* para un tamaño de lote de 10

Observando las dos figuras anteriores, podemos observar que para esta configuración conseguimos llegar a un 0.82 para la predicción, aunque no hemos conseguido rebajar del 0.4 en perdida. A continuación, Figura 15.31, podemos observar una gráfica comparativa entre la precisión y pérdida, durante la fase de entrenamiento, de los conjuntos de entrenamiento y validación.

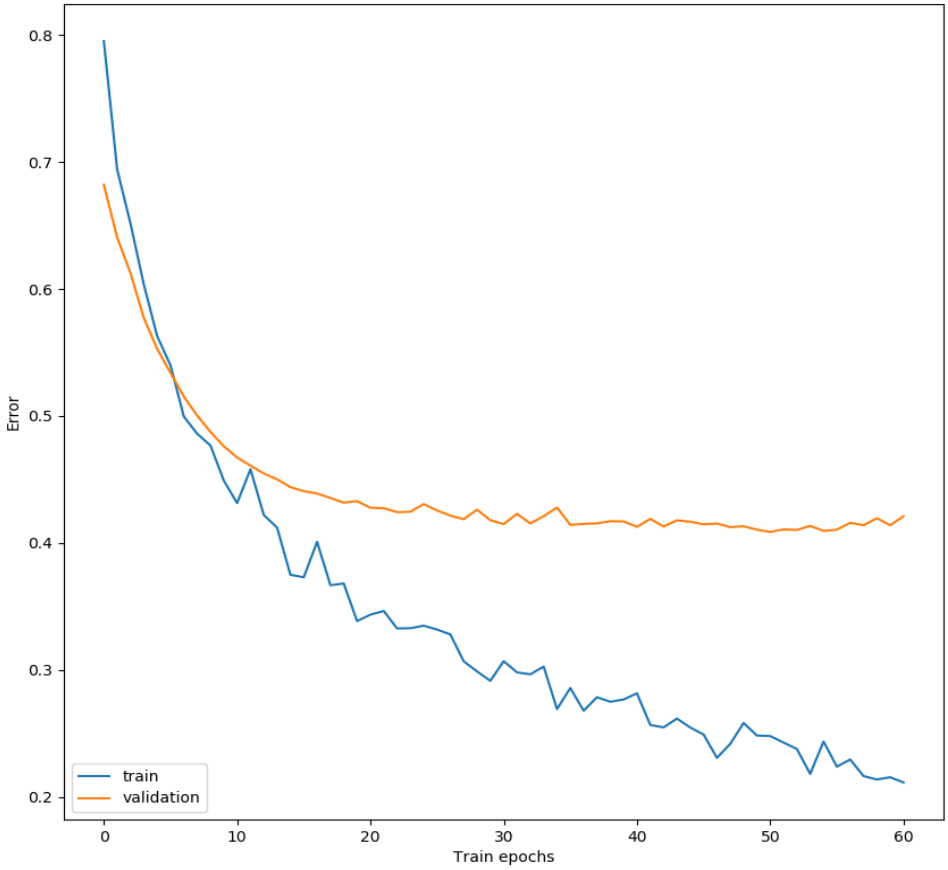


Figura 15.31: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *RMSprop* junto con un tamaño del lote de 10

A continuación, se mostrará la matriz de confusión conseguida con este modelo (ver 15.32).

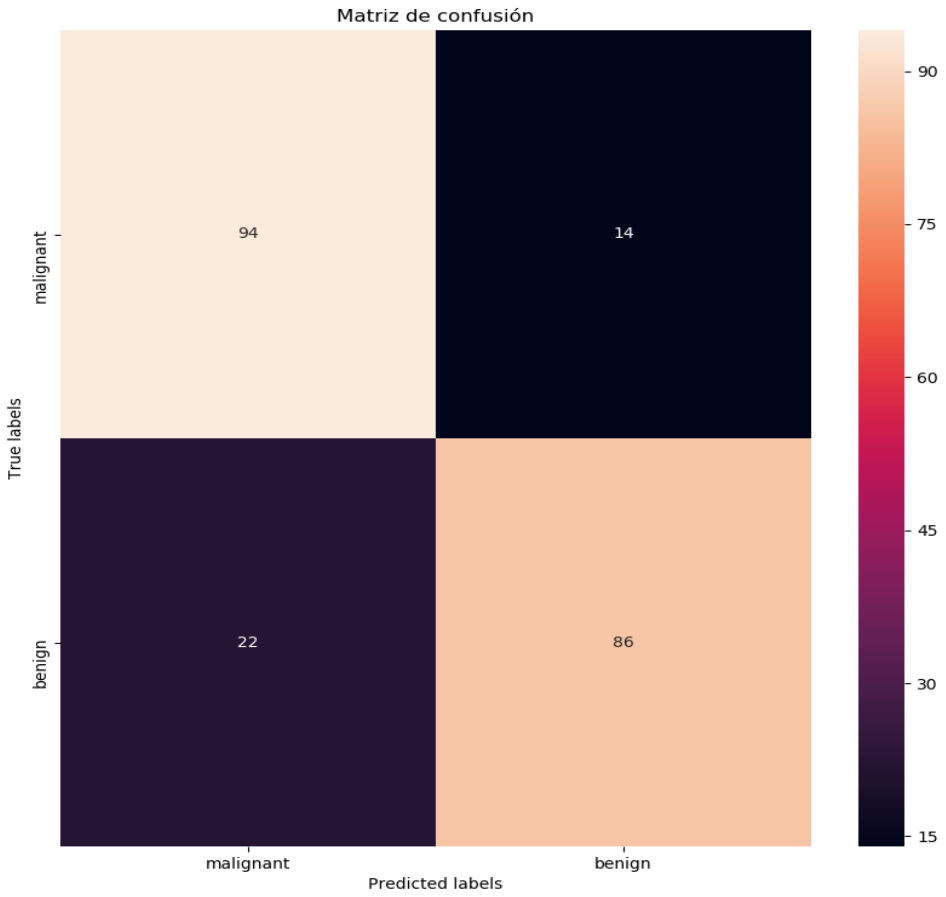


Figura 15.32: Matriz de confusión conseguida con el optimizador *RMSprop* para un tamaño de lote de 10

Finalmente, mostraremos los resultados de precisión y validación del modelo con esta configuración utilizando el conjunto de test para su evaluación. Estos resultados se pueden observar en la Tabla 15.8.

| | **Valor** |
|---|---|
| **Precisión** | 83,3333 % |
| **Pérdida** | 0,4362 |

Tabla 15.8: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *RMSprop* y un tamaño del lote de 10

## 15.3. Optimizador *SGD*

En esta sección se mostrarán los resultados obtenidos tras utilizar el optimizador *SGD* a la hora de construir el modelo.

### 15.3.1. Tamaño del lote: 3

Con esta configuración, el número total de épocas ejecutadas ha sido de **135**. En la Figura 15.33, se muestra la precisión obtenida a lo largo del entrenamiento con el conjunto de validación. Además, en la Figura 15.34, se puede observar la pérdida a lo largo del entrenamiento para el conjunto de validación.

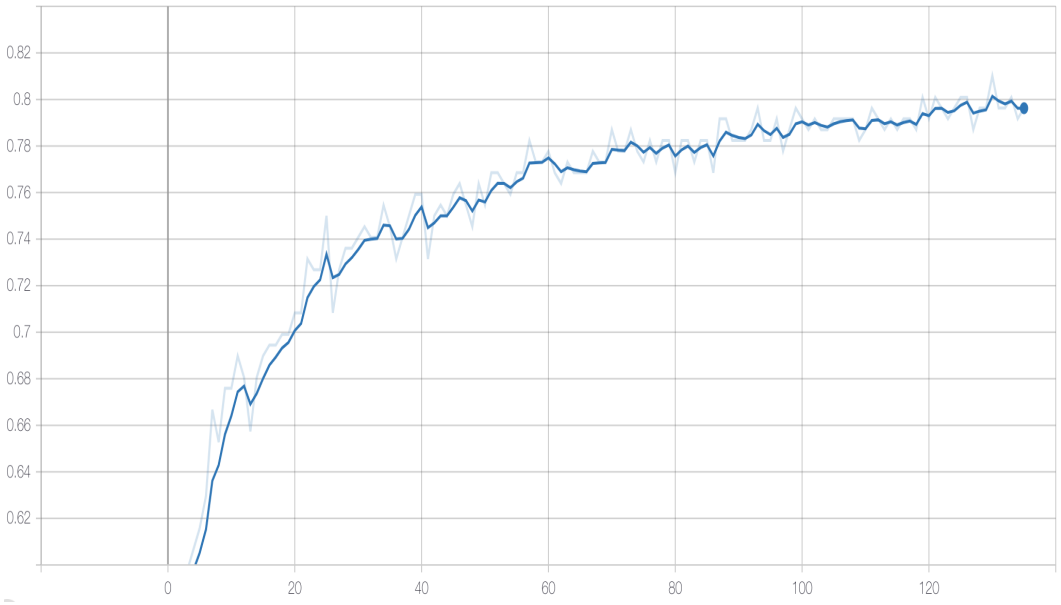


Figura 15.33: Precisión a lo largo del entrenamiento con el optimizador *SGD* para un tamaño de lote de 3

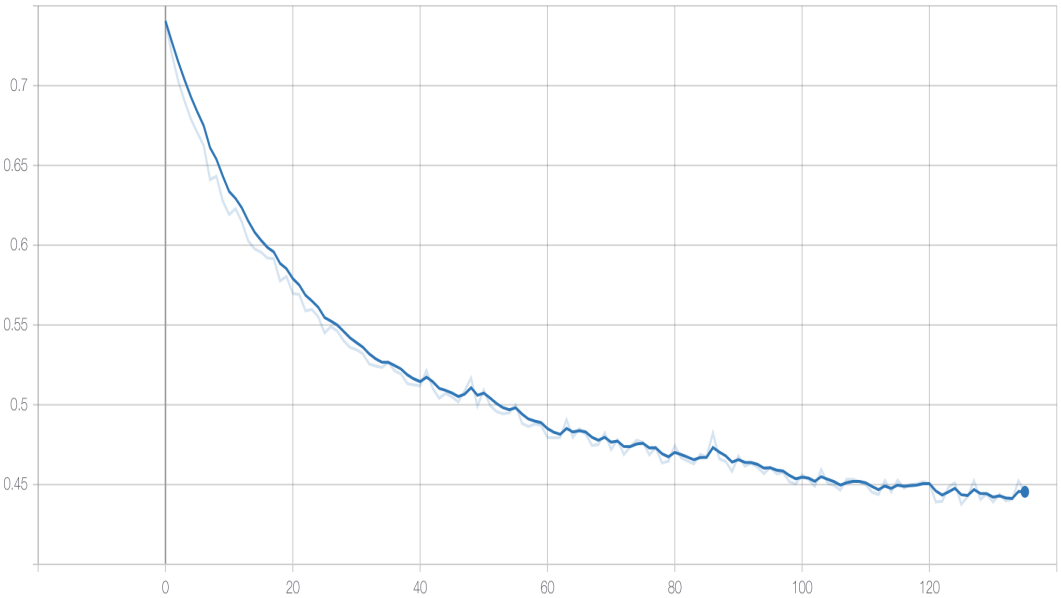


Figura 15.34: Pérdida a lo largo del entrenamiento con el optimizador *SGD* para un tamaño de lote de 3

Como podemos ver en las dos figuras anteriores, el conjunto se queda estancado en un 0.8 de precisión, mientras que para la pérdida, los resultados meramente bajan del 0.45. A continuación, se mostrará una comparativa entre el conjunto de entrenamiento y el conjunto de validación correspondiente a la pérdida durante la fase de entrenamiento (ver 15.35).

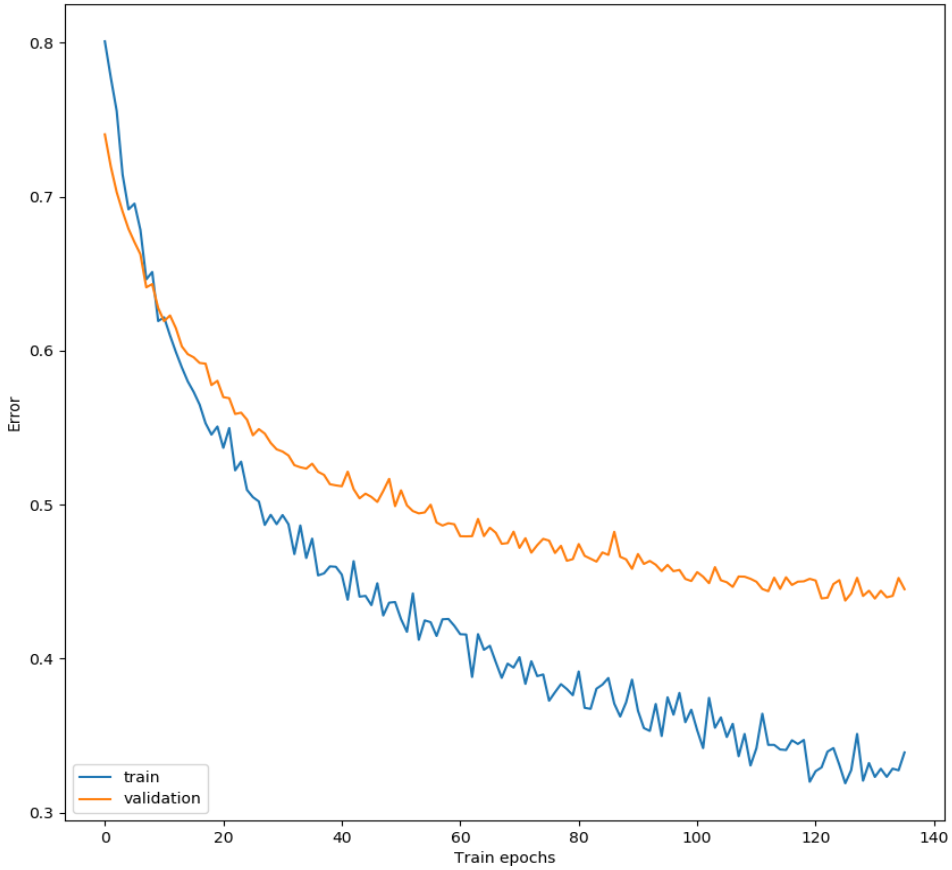


Figura 15.35: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *SGD* junto con un tamaño del lote de 3

A continuación, se mostrará la matriz de confusión conseguida con este modelo (ver 15.32).

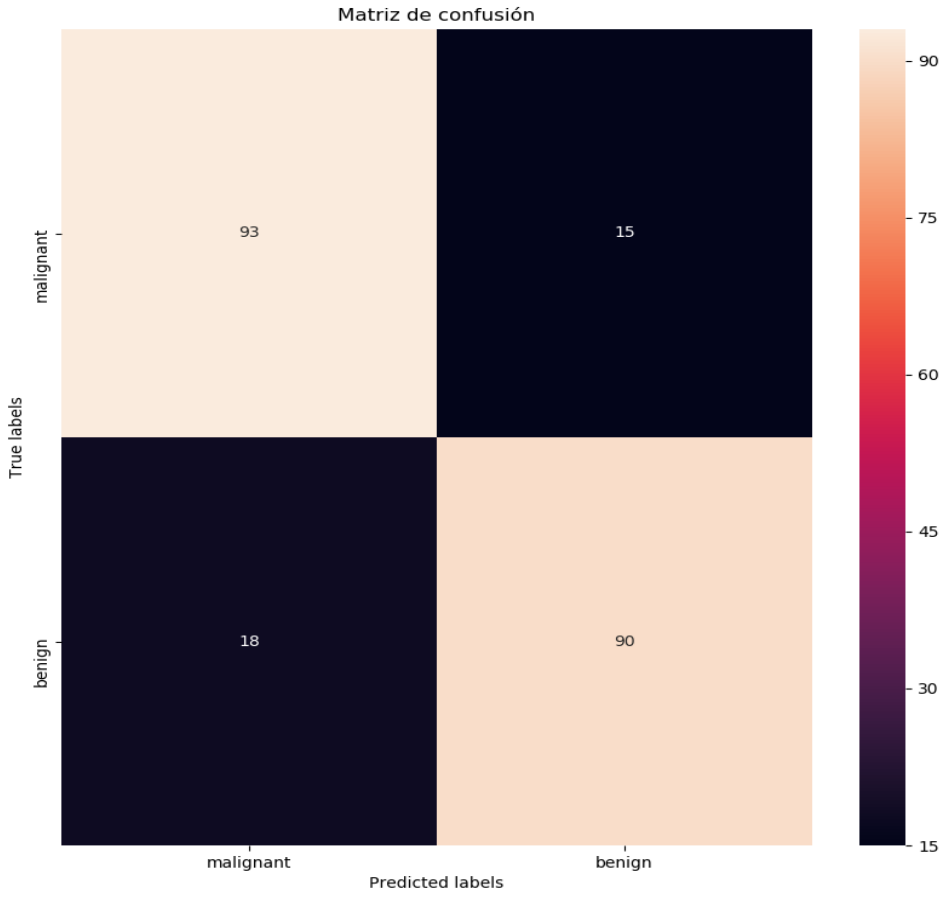


Figura 15.36: Matriz de confusión conseguida con el optimizador *SGD* para un tamaño de lote de 3

Finalmente, mostraremos los resultados de precisión y validación del modelo con esta configuración utilizando el conjunto de test para su evaluación. Estos resultados se pueden observar en la Tabla 15.9.

| | Valor |
|---|---|
| **Precisión** | 84,7222 % |
| **Pérdida** | 0,3811 |

Tabla 15.9: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *SGD* y un tamaño del lote de 3

### 15.3.2. Tamaño del lote: 5

Con esta configuración, el número total de épocas ejecutadas ha sido de **140**. En la Figura 15.37, se muestra la precisión obtenida a lo largo del entrenamiento con el conjunto de validación. Además, en la Figura 15.38, se puede observar la pérdida a lo largo del entrenamiento para el conjunto de validación.

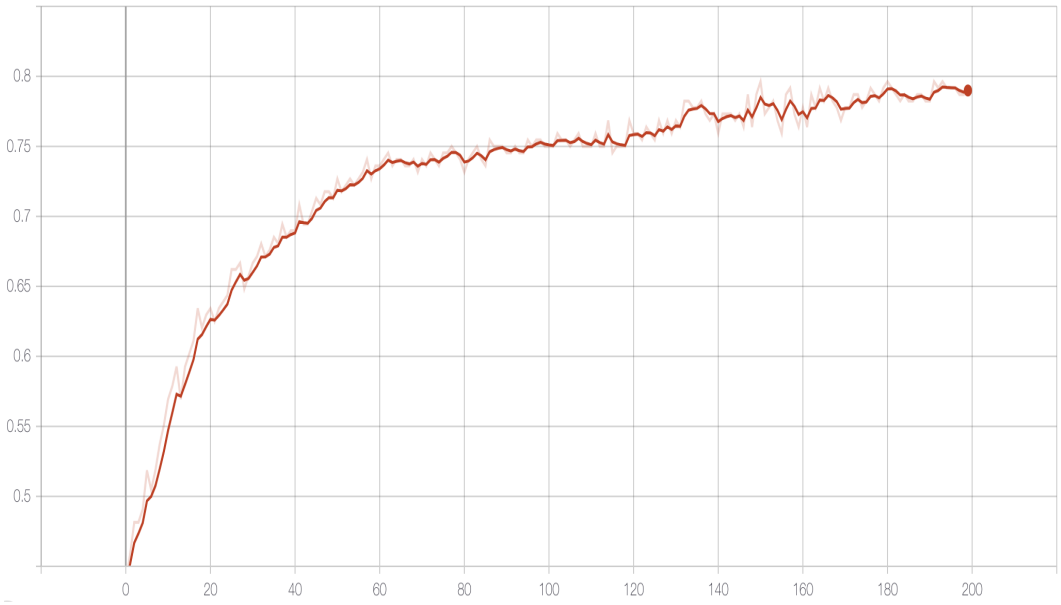


Figura 15.37: Precisión a lo largo del entrenamiento con el optimizador *SGD* para un tamaño de lote de 5

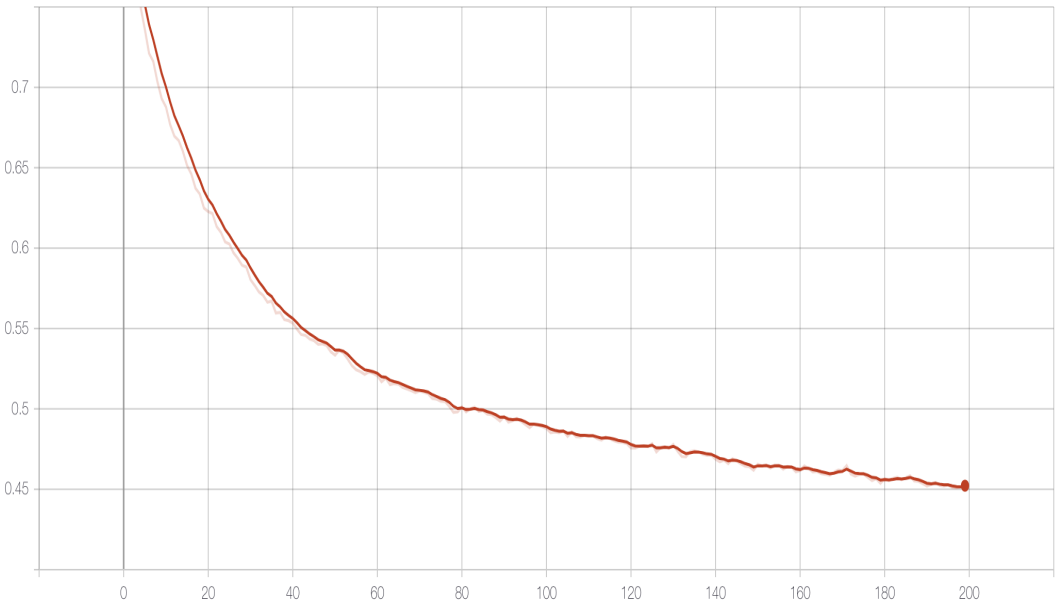


Figura 15.38: Pérdida a lo largo del entrenamiento con el optimizador *SGD* para un tamaño de lote de 5

Observando las anteriores dos gráficas, podemos ver que esta configuración no es muy buena, ya que en precisión sobre el conjunto de validación, en la iteración **199** comienza a disminuir, con respecto a la pérdida, no conseguimos bajar del 0,45. A continuación, se mostrará una comparativa entre el conjunto de entrenamiento y el conjunto de validación correspondiente a la pérdida durante la fase de entrenamiento (ver 15.39).

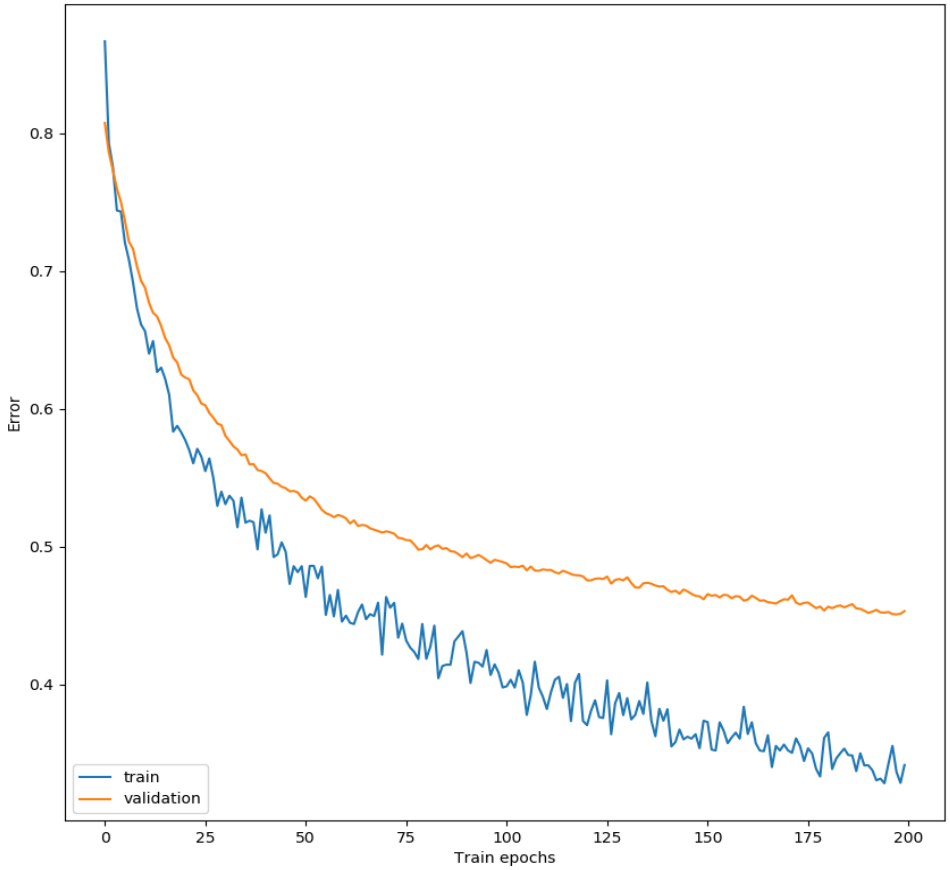


Figura 15.39: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *SGD* junto con un tamaño del lote de 5

A continuación, se mostrará la matriz de confusión conseguida con este modelo (ver 15.40).

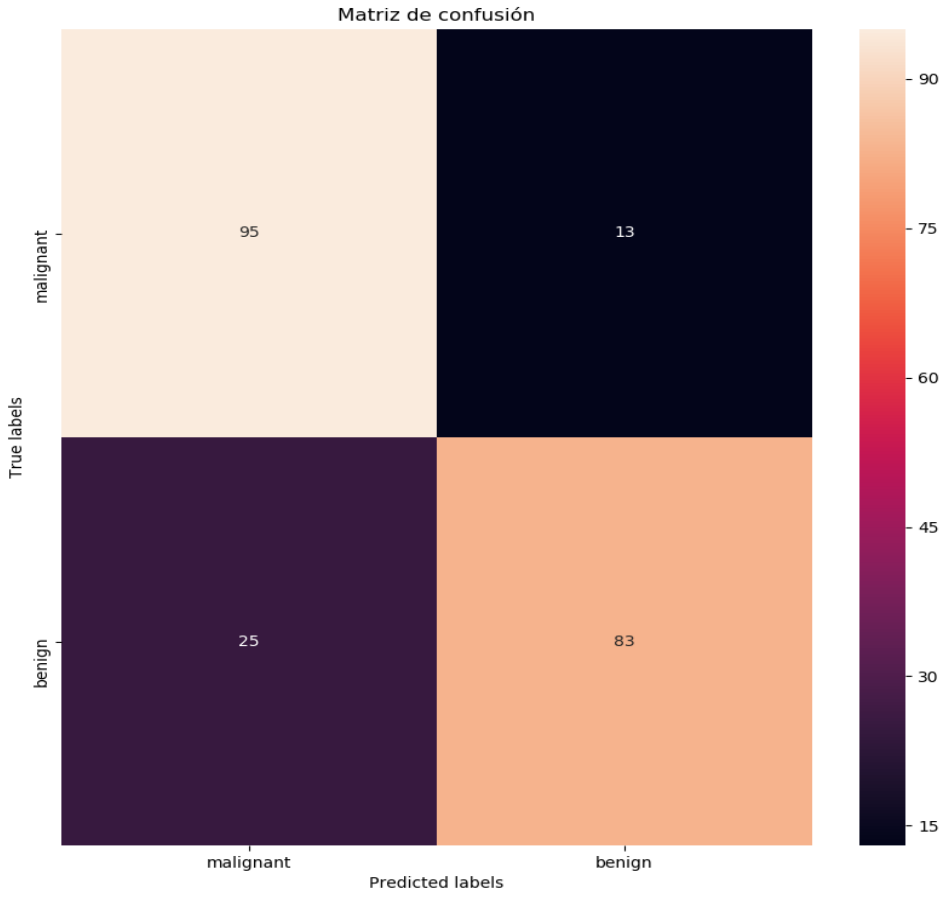


Figura 15.40: Matriz de confusión conseguida con el optimizador *SGD* para un tamaño de lote de 5

Finalmente, mostraremos los resultados de precisión y validación del modelo con esta configuración utilizando el conjunto de test para su evaluación. Estos resultados se pueden observar en la Tabla 15.10.

| | Valor |
|---|---|
| **Precisión** | 82,4074 % |
| **Pérdida** | 0,4330 |

Tabla 15.10: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *SGD* y un tamaño del lote de 5

### 15.3.3. Tamaño del lote: 7

Con esta configuración, el número total de épocas ejecutadas ha sido de **209**. En la Figura 15.41, se muestra la precisión obtenida a lo largo del entrenamiento con el conjunto de validación. Además, en la Figura 15.42, se puede observar la pérdida a lo largo del entrenamiento para el conjunto de validación.

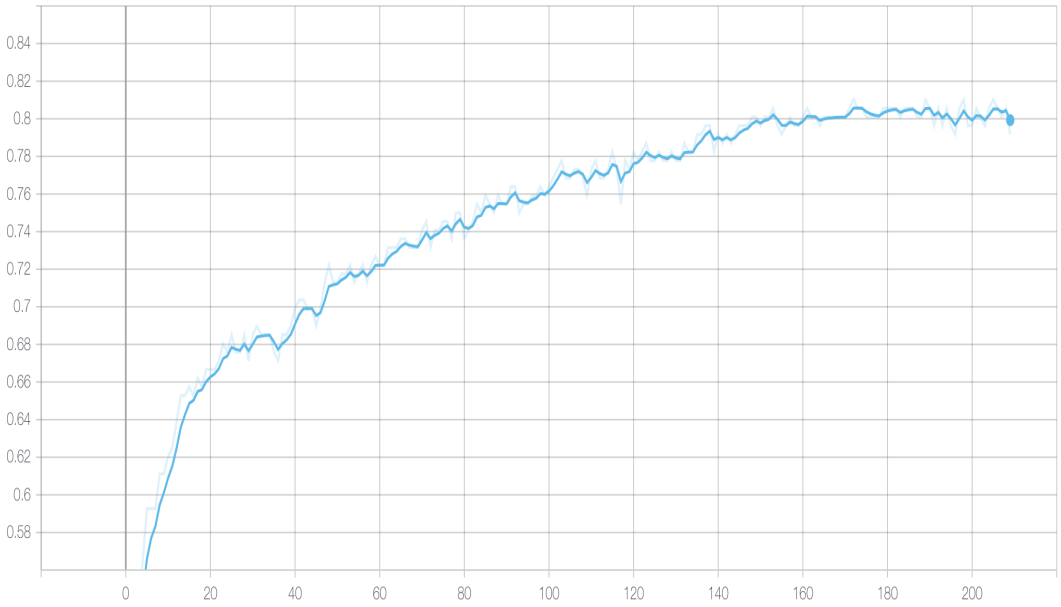


Figura 15.41: Precisión a lo largo del entrenamiento con el optimizador *SGD* para un tamaño de lote de 7

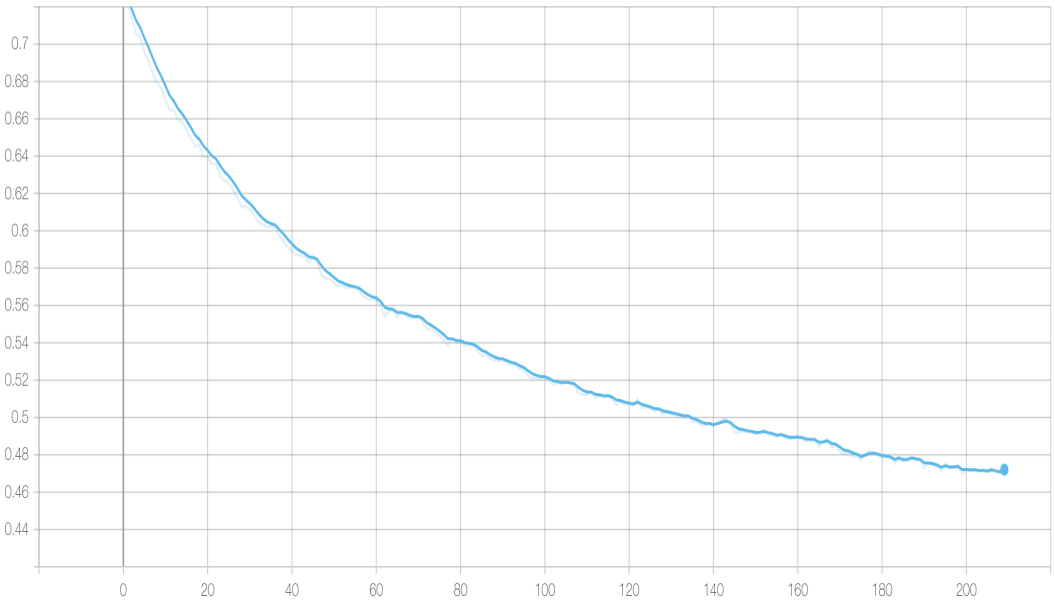


Figura 15.42: Pérdida a lo largo del entrenamiento con el optimizador *SGD* para un tamaño de lote de 7

Con esta configuración, el número de épocas necesarias para no conseguir mejorar ninguna de las configuración anteriores, precisión, 0,8 y, pérdida, superior a 0,45. A continuación, se mostrará una comparativa entre el conjunto de entrenamiento y el conjunto de validación correspondiente a la pérdida durante la fase de entrenamiento (ver 15.43).

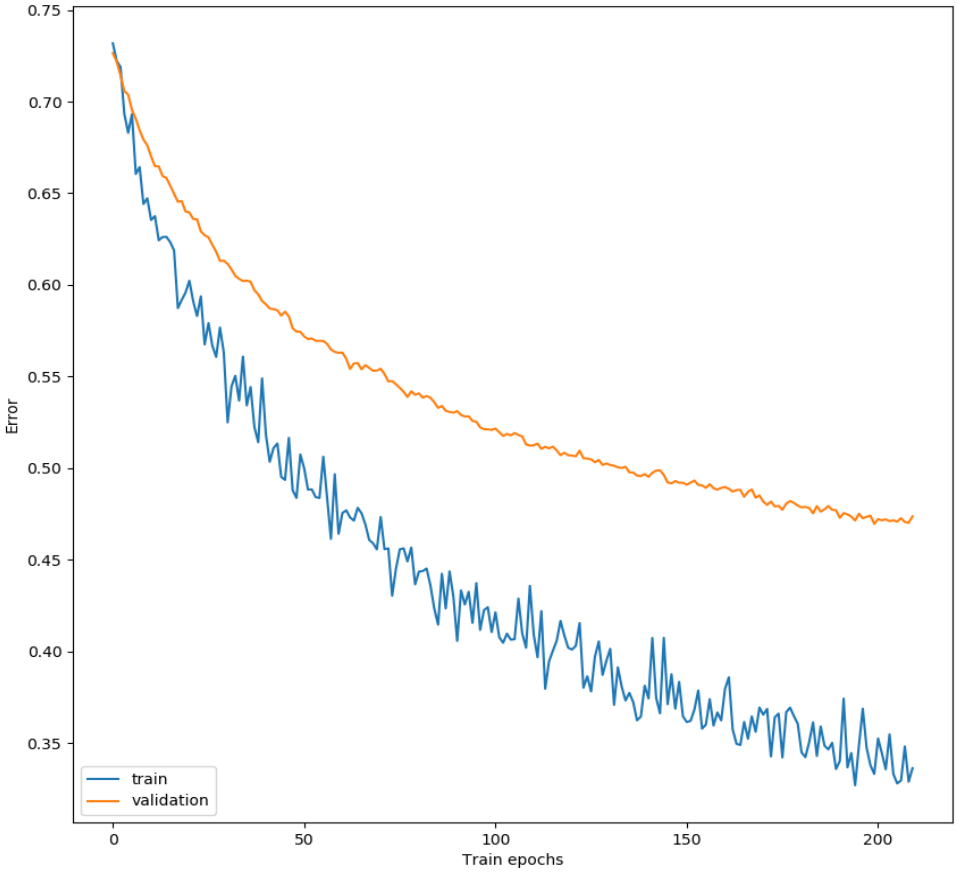


Figura 15.43: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *SGD* junto con un tamaño del lote de 7

A continuación, se mostrará la matriz de confusión conseguida con este modelo (ver 15.44).

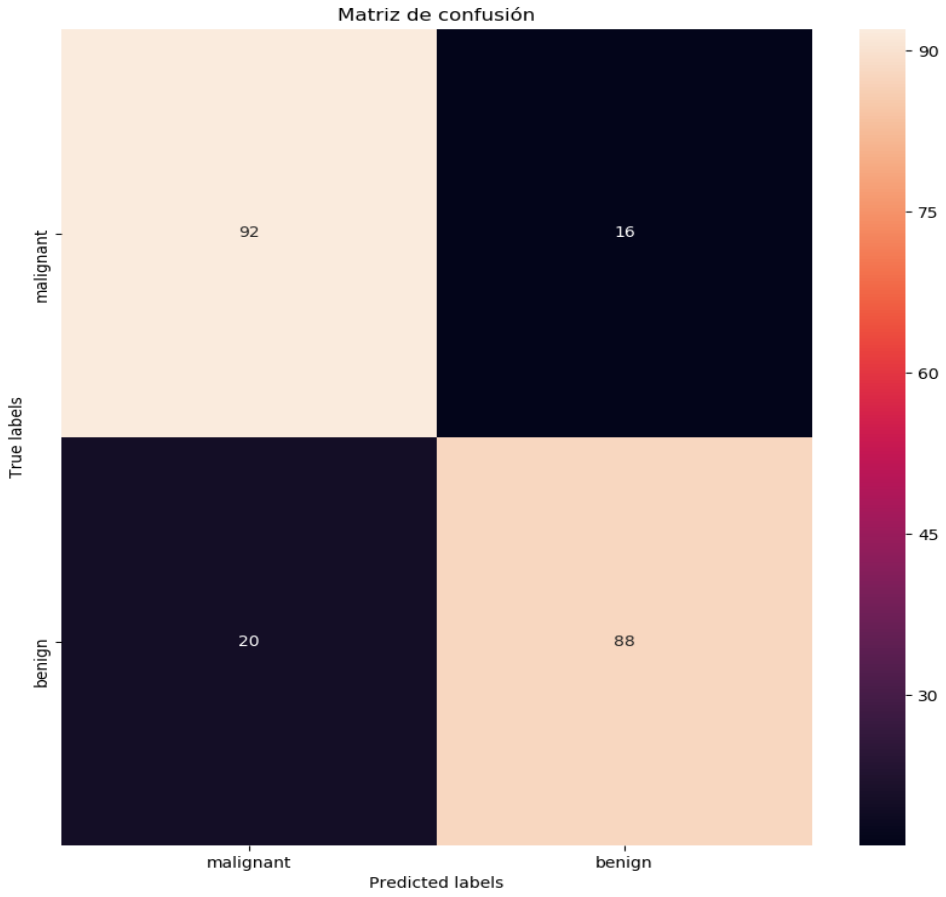


Figura 15.44: Matriz de confusión conseguida con el optimizador *SGD* para un tamaño de lote de 7

Finalmente, mostraremos los resultados de precisión y validación del modelo con esta configuración utilizando el conjunto de test para su evaluación. Estos resultados se pueden observar en la Tabla 15.11.

| | Valor |
|---|---|
| **Precisión** | 83,3333 % |
| **Pérdida** | 0,3901 |

Tabla 15.11: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *SGD* y un tamaño del lote de 7

### 15.3.4. Tamaño del lote: 10

Esta configuración ha necesitado un total de **314** épocas en la fase de entrenamiento. En la Figura 15.45, se muestra la precisión obtenida a lo largo del entrenamiento con el conjunto de validación. Además, en la Figura 15.46, se puede observar la pérdida a lo largo del entrenamiento para el conjunto de validación.

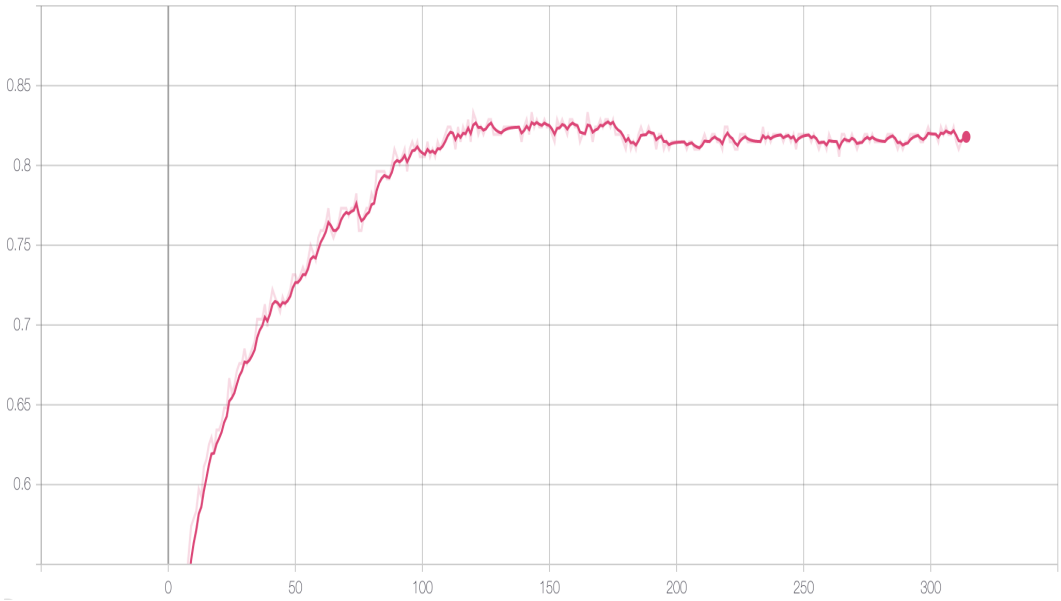


Figura 15.45: Precisión a lo largo del entrenamiento con el optimizador *SGD* para un tamaño de lote de 10

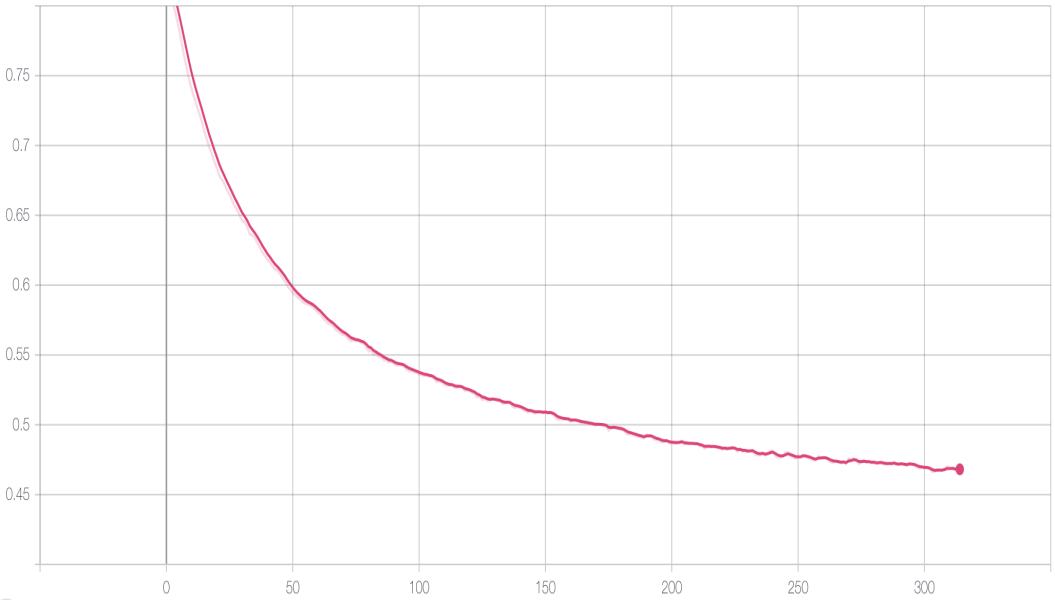


Figura 15.46: Pérdida a lo largo del entrenamiento con el optimizador *SGD* para un tamaño de lote de 10

Esta configuración ha sido la peor que se ha podido encontrar según la división realizada para esta sección. No sólo ha necesitado un gran número de épocas, sino que, además, los resultados obtenidos no son mejores que los conseguidos por otra configuración. A continuación, se mostrará una comparativa entre el conjunto de entrenamiento y el conjunto de validación correspondiente a la pérdida durante la fase de entrenamiento (ver 15.47).

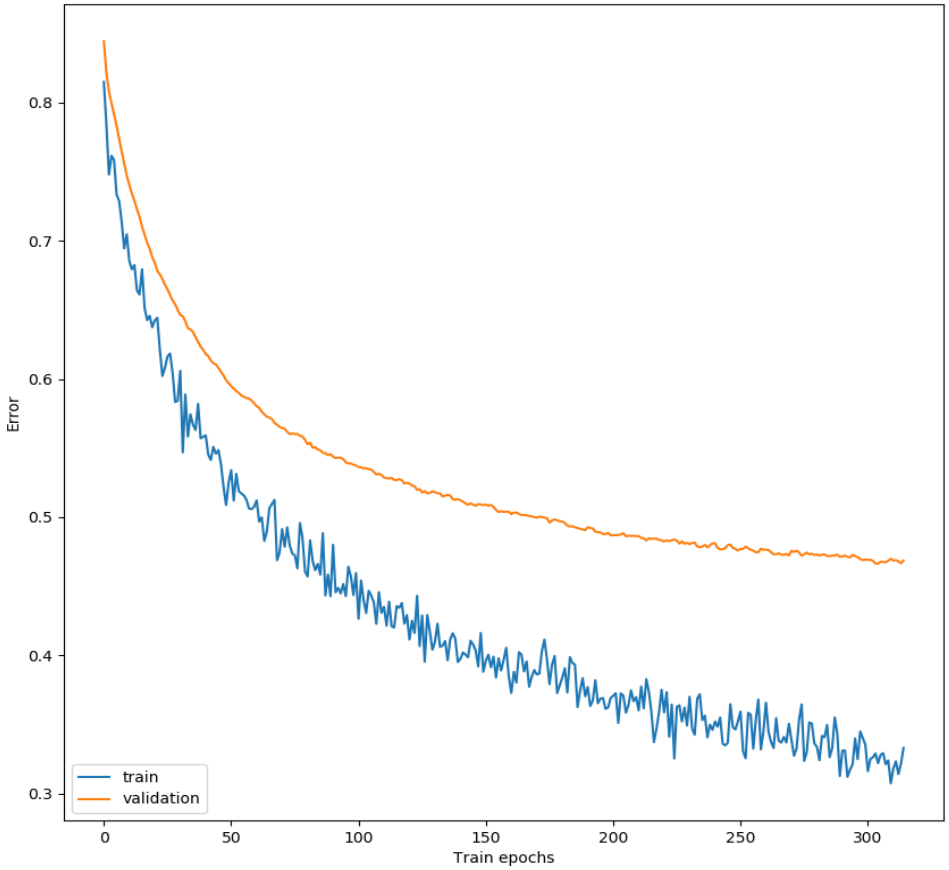


Figura 15.47: Comparación entre la pérdida en validación con la de entrenamiento utilizando el optimizador *SGD* junto con un tamaño del lote de 10

A continuación, se mostrará la matriz de confusión conseguida con este modelo (ver 15.48).

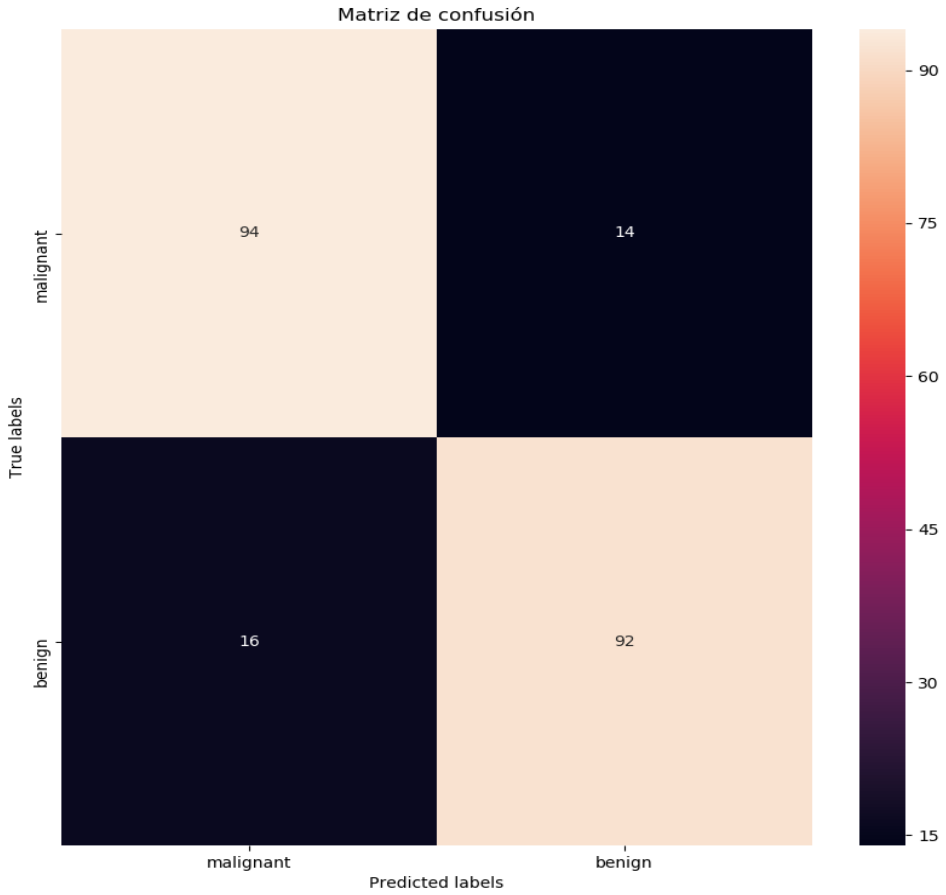


Figura 15.48: Matriz de confusión conseguida con el optimizador *SGD* para un tamaño de lote de 10

Finalmente, mostraremos los resultados de precisión y validación del modelo con esta configuración utilizando el conjunto de test para su evaluación. Estos resultados se pueden observar en la Tabla 15.12.

| | **Valor** |
|---|---|
| **Precisión** | 86,1111 % |
| **Pérdida** | 0,3651 |

Tabla 15.12: Valores obtenidos por este modelo al aplicar el conjunto de test utilizando el optimizador *SGD* y un tamaño del lote de 10

## 15.4. Tablas comparativas

En esta sección se mostrará una tabla comparativa entre la precisión y la pérdida de todas las configuraciones realizadas sobre el conjunto test (ver 15.13).

| Optimizador | Precisión | Pérdida |
|---|---|---|
| ***Adam* - Lote 3** | **87,9629 %** | **0,3182** |
| ***Adam* - Lote 5** | 83,3333 % | 0,3884 |
| ***Adam* - Lote 7** | 81,0185 % | 0,4082 |
| ***Adam* - Lote 10** | 82,8703 % | 0,3927 |
| ***RMSprop* - Lote 3** | 80,5555 % | 0,4436 |
| ***RMSprop* - Lote 5** | **87,5000 %** | **0,3276** |
| ***RMSprop* - Lote 7** | 85,1851 % | 0,3652 |
| ***RMSprop* - Lote 10** | 83,3333 % | 0,4362 |
| ***SGD* - Lote 3** | 84,7222 % | 0,3811 |
| ***SGD* - Lote 5** | 82,4074 % | 0,4330 |
| ***SGD* - Lote 7** | 83,3333 % | 0,3901 |
| ***SGD* - Lote 10** | 86,1111 % | 0,3651 |

Tabla 15.13: Tabla comparativa general de resultados

Como podemos observar, los mejores resultados obtenidos para el conjunto de test se obtienen utilizando un optimizador *Adam* y con un tamaño del lote de 3. Esto tiene sentido, ya que conseguimos un resultado similar, en toda la fase de entrenamiento, usando el conjunto de validación. Además de esta configuración, utilizando el optimizador *RMSprop* y un tamaño del lote de 5, también conseguimos buenos resultados. De igual manera, este resulta-

do se ve reflejado en las Figuras 15.21 y 15.22.

Finalmente, como observamos en la Tabla 15.14, mostraremos una comparativa de los resultados conseguidos con la mejor configuración para nuestro modelo, y los resultados obtenidos por el modelo del artículo [7].

| **Modelo** | **Propio** | **Artículo** |
|---|---|---|
| **Precisión** | 87,9629 % | 85,55 % |

Tabla 15.14: Tabla comparativa de resultados propios y de artículo

Los resultados obtenidos por este modelo con la mejor configuración encontrada, han conseguido mejor precisión que la conseguida por los autores del artículo. El único problema que podemos encontrar aquí es que los autores del artículo no proporcionan el valor de entropía cruzada para calcular la pérdida del modelo, es por ello por lo que no estamos tan seguros de que el resultado obtenido por nuestro propio modelo sea mejor que el conseguido por los autores del artículo en cuanto a valores de pérdida. Es importante reseñar que en el citado artículo aplica un preprocesamiento para segmentar la lesión previo al uso de la red neuronal, que nosotros no hemos llegado a aplicar.

# Parte VI

# Conclusiones

# Capítulo 16

# Conclusiones del autor

En este capítulo se hablará de los objetivos alcanzados por parte del alumno durante la realización de este proyecto. Finalizado la fase de desarrollo del proyecto y tras haber superado las pruebas del sistema, podemos determinar que hemos cumplido con los requisitos declarados al comienzo del proyecto.

Los objetivos tras la finalización de este proyecto son los siguientes:

- Se ha mejorado el uso del lenguaje de programación orientado a objetos *Python*.
- Se han adquirido conocimientos de creación de entornos virtuales y del uso de servidores.
- Se han adquirido conocimientos en el uso de las bibliotecas del lenguaje *Python*, tales como *NumPy* o *Pandas*.
- Se han adquirido las bases necesarias para la creación de un modelo de red neuronal residual profunda, y de las bibliotecas necesarias para su creación.
- Se ha utilizado los conocimientos adquiridos a lo largo de la carrera para la creación de documentos oficiales de un proyecto, como el manual técnico o el manual de código.
- Se han mejorado los conocimientos sobre la herramienta de generación de documentos LaTeX.
- Se ha creado una interfaz de líneas de comandos que permite realizar el entrenamiento del modelo.

- Se han adquirido conocimientos del uso de los ficheros *HDF5*.
- Se han creado distintos modelos para la clasificación de los dos tipos de lesiones en la piel considerados en este proyecto.
- Se ha implementado con éxito un modelo de red neuronal residual con su correspondiente inicialización.

# Capítulo 17

# Futuras mejoras

En el presente capítulo se intentarán definir las posibles mejoras que se podrían realizar en un futuro al sistema, ya sea aumentando su funcionalidad o aumentando su fiabilidad.

La primera posible mejora a realizar, es un preprocesamiento a las imágenes de entrada, ya que el intentado para este proyecto no ha sido satisfactorio. Una posible opción sería la utilización de un modelo de red neuronal profundo denominado *UNet* [17]. Este modelo, en vez de devolvernos un entero, nos devolvería una aproximación a la perfecta máscara de la imagen, denominada *ground truth*. Esta máscara conseguiría asilar por completo la lesión, consiguiendo así una mejora muy notable.

Otra futura mejora, una vez conseguido un mejor procesamiento de las imágenes de entrada, sería intentar clasificar un mayor número de tipos de lesiones, consiguiendo abarcar un mayor rango de enfermedades. Por último, otra mejora de este proyecto sería realizar un sistema que nos permita realizar la clasificación con varios tipos de modelos de red neuronal, como por ejemplo, realizar la clasificación con una *ResNet* o con una *CNN*. Además, se podría realizar una optimización del código o utilizar un mayor número de capas.

Por último, este proyecto se podría implementar para su uso en aplicaciones web o móviles, para que así los usuarios de la misma puedan analizar de forma autónoma lesiones en su piel.

# Bibliografía


[1] Computing Machinery. Computing machinery and intelligence-am turing. *Mind*, 59(236):433, 1950.

[2] Ethem Alpaydin. *Machine learning: the new AI*. MIT press, 2016.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[4] American cancer society. https://www.cancer.org/es/. Último acceso 27-09-2019.

[5] Estadísticas sobre cancer en la piel. https://www.cancer.org/es/cancer/cancer-de-piel-tipo-melanoma/acerca/estadisticas-clave.html. Último acceso 27-09-2019.

[6] Distintos estados de un melanoma. https://www.cancer.org/es/cancer/cancer-de-piel-tipo-melanoma/deteccion-diagnostico-clasificacion-por-etapas/clasificacion-por-etapas-el-cancer-de-piel-tipo-melanoma.html. Último acceso 27-09-2019.

[7] Lequan Yu, Hao Chen, Qi Dou, Jing Qin, and Pheng-Ann Heng. Automated melanoma recognition in dermoscopy images via very deep residual networks. *IEEE transactions on medical imaging*, 36(4):994–1004, 2016.

[8] 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.

[9] Framework de aprendizaje profundo. https://caffe.berkeleyvision.org. Último acceso 27-09-2019.

[10] Keras. https://keras.io. Último acceso 22-08-2019.

[11] Tensorflow. https://www.tensorflow.org. Último acceso 22-08-2019.

[12] Python3. https://www.python.org. Último acceso 22-08-2019.

[13] International skin imaging classification, *Dataset* isic. https://www.isic-archive.com/#!/topWithHeader/onlyHeaderTop/gallery. Último acceso 24-08-2019.

[14] Tensorboard. https://www.tensorflow.org/guide/summaries_and_tensorboard. Último acceso 19-07-2008.

[15] Adria Romero Lopez, Xavier Giro-i Nieto, Jack Burdick, and Oge Marques. Skin lesion classification from dermoscopic images using deep learning techniques. In *Biomedical Engineering (BioMed), 2017 13th IASTED International Conference on*, pages 49–54. IEEE, 2017.

[16] A comprehensive guide to convolutional neural networks — the eli5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Último acceso 22-08-2019.

[17] Unet. https://towardsdatascience.com/u-net-b229b32b4a71. Último acceso 22-08-2019.

[18] Style guide for python pep 8. https://www.python.org/dev/peps/pep-0008/. Último acceso 27-09-2019.

[19] Latex. https://www.latex-project.org. Último acceso 23-08-2019.

[20] Tensorflow c++ reference. https://www.tensorflow.org/api_docs/cc/. Último acceso 24-08-2019.

[21] Haskell bindigns for tensorflow. https://github.com/tensorflow/haskell. Último acceso 24-08-2019.

[22] Tensorflow java reference. https://www.tensorflow.org/api_docs/java/reference/org/tensorflow/package-summary. Último acceso 24-08-2019.

[23] Página web personal del autor. https://fchollet.com. Último acceso 24-08-2019.

[24] Microsoft cognitive services. https://azure.microsoft.com/es-es/services/cognitive-services/. Último acceso 24-08-2019.

[25] Theano documentation web. http://deeplearning.net/software/theano/. Último acceso 24-08-2019.

[26] Official web page of numpy. https://www.numpy.org. Último acceso 25-08-2019.

[27] Creator of numpy. https://es.wikipedia.org/wiki/Travis_Oliphant. Último acceso 27-09-2019.

[28] Offical web page of jupyter. https://jupyter.org. Último acceso 27-09-2019.

[29] Visual studio code's official web page. https://code.visualstudio.com. Último acceso 27-09-2019.

[30] Hierarchical data format. https://www.hdfgroup.org/solutions/hdf5/. Último acceso 27-09-2019.

[31] National center for supercomputing applications. https://en.wikipedia.org/wiki/National_Center_for_Supercomputing_Applications. Último acceso 27-09-2019.

[32] Pedro antonio gutiérrez peña and juan carlos fernández caballero. grupo de invstigación y redes neuronales artificiales ayrna. https://uco.es/ayrna/. Último acceso 27-09-2019.

[33] Official web page of uml. https://www.uml.org. Último acceso 27-09-2019.

[34] Python software foundation web page. https://www.python.org/psf/. Último acceso 27-09-2019.

[35] Guido van rossum. https://es.wikipedia.org/wiki/Guido_van_Rossum. Último acceso 27-09-2019.

[36] Programming language abc. https://es.wikipedia.org/wiki/ABC_(lenguaje_de_programacion). Último acceso 27-09-2019.

[37] Amoeba operative system. https://en.wikipedia.org/wiki/Amoeba_(operating_system). Último acceso 27-09-2019.

[38] Monty python. https://es.wikipedia.org/wiki/Monty_Python. Último acceso 27-09-2019.

[39] Google. https://es.wikipedia.org/wiki/Google. Último acceso 27-09-2019.

[40] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[41] Metodología ágil. https://es.wikipedia.org/wiki/Desarrollo_\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{a\global\mathchardef\accent@spacefactor\spacefactor}\accent19a\egroup\spacefactor\accent@spacefactor\futurelet\@let@token\penalty\@M\hskip\z@skipgil_de_software. Último acceso 27-09-2019.

# Parte VII

# Apéndices

# Apéndice A

# Manual de usuario

Este documento hace referencia al manual de usuario de la aplicación creada en *Python* para el Trabajo de Fin de Grado ***Clasificación de melanomas en la piel a través de redes neuronales residuales***.

Este apéndice comenzará con la descripción del producto desarrollado.

## A.1. Descripción del producto

El producto que ha sido especificado en este manual, se utiliza mediante una interfaz de línea de comandos a través de la cual, el usuario podrá entrenar y evaluar el modelo que el seleccione. Esta línea de comandos es flexible, ya que da la oportunidad al usuario de realizar una plena configuración del modelo.

Las características que nos podemos encontrar en el sistema creado son:

- Ejecución de un modelo de red neuronal residual.
- Automatización de la fase de obtención de resultados.
- Interfaz de línea de comandos flexible.
- Desarrollado con el lenguaje de programación a objetos *Python*.

## A.2. Requisitos del Sistema

Esta aplicación puede ser utilizada sobre cualquier sistema que incorpore *Python*, *Tensorflow* y *Keras*. La aplicación puede ser utilizada en sistemas

con un sistema operativo basado en *UNIX* o en *MS-DOS*.

### A.2.1. Requisitos Recomendados

- 32GB de RAM.
- Tarjetas con soporte para CUDA.
- Espacio disponible en el almacenamiento del sistema para poder almacenar los datos del entrenamiento mayor de 20GB.
- Procesador de la familia Intel i7 Coffe Lake u otro equivalente.

### A.2.2. Requisitos Mínimos

- 8GB de RAM.
- Espacio disponible en el almacenamiento del sistema para poder almacenar los datos del entrenamiento mayor de 5GB.
- Procesador de la familia Intel i5 Skylake u otro equivalente con un mínimo de 3,5GHz y 4 núcleos.

### A.2.3. Requisitos Software

- Sistema operativos *GNU/Linux*, *Mac OS* o *Windows*.

## A.3. Instalación de la aplicación

### A.3.1. Instalación de Python

#### Linux

*Python* se puede instalar en sistemas *GNU/Linux* utilizando el administrador de paquetes *apt-get*. El siguiente mandato muestra cómo puede realizarse:

```
sudo apt-get install python
```

#### Mac Os

*Python* se puede instalar en sistemas *Mac OS* utilizando el administrador de paquetes *brew*. El siguiente mandato muestra cómo puede realizarse:

```
brew install python
```

#### Windows

El lenguaje de programación *Python* puede ser instalado en el sistema operativo *Windows* descargándolo desde su página web https://www.python.org/downloads/windows/.

### A.3.2. Instalación de Tensorflow

#### Linux

*Tensorflow* se puede instalar en sistemas *GNU/Linux* utilizando el administrador de paquetes *pip*. El siguiente mandato muestra cómo puede realizarse:

```
pip install --upgrade tensorflow
```

#### Mac Os

Igualmente que con el sistema operativo *Linux*, *Tensorflow* puede ser instalado en sistemas operativos *Mac OS* haciendo uso del comando *pip*. El siguiente mandato muestra cómo realizar la instalación:

```
pip install --upgrade tensorflow
```

#### Windows

*Tensorflow* puede ser instalado en sistemas operativos *Windows* haciendo uso del comando *pip*. El siguiente mandato muestra cómo realizar la instalación:

```
pip install --upgrade tensorflow
```

### A.3.3. Instalación de Keras

#### Linux

*Keras* se puede instalar en sistemas *GNU/Linux* utilizando el administrador de paquetes *pip*. El siguiente mandato muestra cómo puede realizarse:

```
pip install keras
```

#### Mac Os

Igualmente que con el sistema operativo *Linux*, *Keras* puede ser instalado en sistemas operativos *Mac OS* haciendo uso del comando *pip*. El siguiente mandato muestra cómo realizar la instalación:

```
pip install keras
```

#### Windows

*Keras* puede ser instalado en sistemas operativos *Windows* haciendo uso del comando *pip*. El siguiente mandato muestra cómo realizar la instalación:

```
pip install keras
```

### A.3.4. Instalación de NumPy

Esta biblioteca puede ser instalada en cualquiera de los sistemas operativos nombrados hasta ahora haciendo uso del gestor de paquetes *pip*. El siguiente mandato muestra cómo debe de realizarse:

```
pip install numpy
```

### A.3.5. Instalación de TQDM

Al igual que con la biblioteca *NumPy*, la biblioteca *TQDM* puede ser instalada en cualquier sistema operativo utilizando el gestor de paquetes *pip*. El siguiente mandato muestra cómo debe de realizarse la instalación:

```
pip install tqdm
```

### A.3.6. Instalación de Matplotlib

Esta biblioteca puede ser instalada a través del gestor de paquetes *pip* en cualquier sistema operativo. Para ello, hay que introducir el comando mostrado en el siguiente mandato:

```
pip install matplotlib
```

### A.3.7. Instalación de Scikit-Learn

La biblioteca *Scikit-Learn* puede ser instalada en cualquier sistema operativo haciendo uso del gestor de paquetes *pip*. El siguiente mandato muestra cómo realizar la instalación:

```
pip install sklearn
```

## A.4. Desinstalación de la aplicación

### A.4.1. Desinstalación de Python

La desinstalación de *Python* en los sistemas operativos *GNU/Linux* y *Mac OS* no se puede realizar por completo, ya que estos sistemas operativos tienen la necesidad de utilizar este lenguaje para poder funcionar. En el caso del sistema operativo *Windows*, bastaría con ir a la sección *Inicio* → *Panel de Control* → *Programas y características*.

### A.4.2. Desinstalación de Tensorflow

Para la desinstalación de esta biblioteca en cualquiera de los sistemas operativos nombrados, bastaría con utilizar el gestor de paquetes *pip* y utilizar el comando mostrado en el siguiente mandato:

```
pip uninstall tensorflow
```

### A.4.3. Desinstalación de Keras

Para la desinstalación de esta biblioteca en cualquiera de los sistemas operativos nombrados, bastaría con utilizar el gestor de paquetes *pip* y utilizar el comando mostrado en el siguiente mandato:

```
pip uninstall keras
```

### A.4.4. Desinstalación de NumPy

Para la desinstalación de esta biblioteca en cualquiera de los sistemas operativos nombrados, bastaría con utilizar el gestor de paquetes *pip* y utilizar el comando mostrado en el siguiente mandato:

```
pip uninstall numpy
```

### A.4.5. Desinstalación de TQDM

Para la desinstalación de esta biblioteca en cualquiera de los sistemas operativos nombrados, bastaría con utilizar el gestor de paquetes *pip* y utilizar el comando mostrado en el siguiente mandato:

```
pip uninstall tqdm
```

### A.4.6. Desinstalación de Matplotlib

Para la desinstalación de esta biblioteca en cualquiera de los sistemas operativos nombrados, bastaría con utilizar el gestor de paquetes *pip* y utilizar el comando mostrado en el siguiente mandato:

```
pip uninstall matplotlib
```

### A.4.7. Desinstalación de Scikit-Learn

Para la desinstalación de esta biblioteca en cualquiera de los sistemas operativos nombrados, bastaría con utilizar el gestor de paquetes *pip* y utilizar el comando mostrado en el siguiente mandato:

```
pip uninstall sklearn
```

## A.5. Interfaz de Línea de Comandos

La utilización de este sistema se realiza a través de la interfaz de línea de comandos. Este sistema se centra en la creación, entrenamiento y evaluación de una red neuronal residual profunda. Para ello, bastaría con ejecutar el comando mostrado en el siguiente mandato:

```
    python main.py -d [ruta-imagenes] -tt [nombre-fichero
-hdf5] -lr [ratio-aprendizaje] -e [ pocas ] -p [paciencia]
 -o [tipo-optimizador] -b [tama o-lote]
```

Cada opción tiene una funcionalidad, seguidamente mostraremos las distintas opciones que nos ofrece el sistema:

- **Opción -d**: Nos permite especificar la ruta en la que se encontrarán las imágenes para el entrenamiento.
- **Opción -tt**: Esta opción nos permite especificar el nombre del fichero donde se guardarán los conjuntos de entrenamiento, validación y test junto a sus etiquetas correspondientes.
- **Opción -lr**: Nos permite especificar un valor para el ratio de aprendizaje. Este valor no puede ser menor o igual a 0.
- **Opción -e**: Esta opción nos permite especificar el número máximo de épocas que el modelo va a poder llegar en la fase de entrenamiento. Este valor no ha de ser menor o igual a 0.
- **Opción -p**: Esta opción nos permite especificar el número de épocas sin mejora que permitiremos antes de dar como finalizada la fase de entrenamiento. Este valor no ha de ser menor o igual a 0.

- **Opción -o**: Esta opción se ha creado para seleccionar el tipo de optimizador que deseamos a la hora de crear el modelo. Las opciones son *0*, para el optimizador *Adam*, *1* para el optimizador *RMSprop* y, *2* para utilizar el optimizador *SGD*.
- **Opción -b**: Esta opción ha sido creada para que el usuario pueda seleccionar el tamaño del lote usado en la fase de entrenamiento. Este valor no puede ser menor o igual que 0.

Para facilitar el uso del sistema al usuario, éste podrá ejecutar el siguiente comando que le mostrará las diferentes opciones de ejecución de las que dispone el sistema:

```
python main.py -h
```

En relación al formato de salida de los resultados, el sistema guardará automáticamente y nombrado con el tipo de optimizador seguido del tamaño del lote utilizado una serie de archivos en los que el usuario podrá evaluar el entrenamiento del modelo. Entre estos ficheros se encuentran:

- Una gráfica en la que se mostrará una comparativa de la pérdida del modelo tras su ejecución entre el conjunto de entrenamiento y el conjunto de evaluación. Esta gráfica estará en formato *PNG*.
- Una imagen en formato *PNG* en la que se guardará la matriz de confusión producida por la evaluación del modelo.
- Ficheros *checkpoint* en los que se guardará tanto la precisión como la pérdida del modelo sobre los conjuntos de validación y de entrenamiento. Para poder observarlos, el usuario tendrá que introducir el siguiente comando. El fichero *log* también será creado automáticamente por el sistema. En este se podrá encontrar varios directorios creados con el nombre del optimizador y tamaño de lote utilizados para el entrenamiento.

  ```
  tensorboard --logdir="fichero log"
  ```

- Los valores de precisión y pérdida del modelo evaluados con el conjunto de test se mostrarán por pantalla una vez la fase de entrenamiento haya finalizado.
- El valor de precisión y pérdida tanto para el conjunto de entrenamiento como el de validación, se irá mostrado época por época por pantalla.

## A.6. Ejemplo de ejecución

Una vez hemos ejecutado el comando tal y como hemos indicado en el apartado A.5, el sistema nos mostrará una serie de apartados que explicarán lo que nuestro sistema está realizando en ese momento. Como se muestra en la Figura A.1, lo primero que realiza el sistema es leer las imágenes y crear el modelo con la configuración introducida.

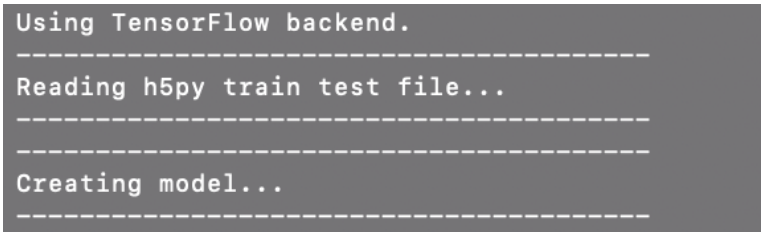


Figura A.1: Ejecución inicial del sistema

Una vez creado, como se muestra en la Figura A.2, el sistema nos mostrará un resumen del modelo, diciendo los parámetros que contiene nuestro modelo. Como nota, hemos decidido no mostrar todas las capas ya que es un modelo demasiado grande.

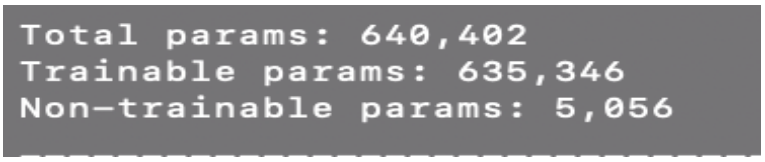


Figura A.2: Resumen del modelo

Una vez el entrenamiento ha comenzado, el sistema nos irá mostrando época por época, como se muestra en la Figura A.3, la precisión y pérdida tanto del conjunto de entrenamiento, como del conjunto de validación.

```
Epoch 85/500
648/648 [==============================] - 51s 78ms/step - loss: 0.1716 - acc: 0.9491 - val_loss: 0.4259 - val_acc: 0.8519
Epoch 86/500
648/648 [==============================] - 51s 78ms/step - loss: 0.1584 - acc: 0.9552 - val_loss: 0.4414 - val_acc: 0.8333
Epoch 87/500
648/648 [==============================] - 51s 78ms/step - loss: 0.1644 - acc: 0.9491 - val_loss: 0.4287 - val_acc: 0.8426
```

Figura A.3: Fase de entrenamiento

Finalmente, se muestra a modo resumen los parámetros utilizados para esa configuración, ya que, como el entrenamiento puede requerir una o dos horas, el usuario puede haber olvidado la configuración con la que había entrenado el modelo (ver Figura A.4).

```
----------------------------------------
Summary:
        optimizer -> Adam
        lr ->  1e-06
        epochs ->  500
        batch size ->  5
        patience ->  10
----------------------------------------
----------------------------------------
Evaluating model...
----------------------------------------
----------------------------------------
Test loss->  0.36031323450582997
Test accuracy->  0.8611111133186905
----------------------------------------
----------------------------------------
Creating confusion matrix...
----------------------------------------
----------------------------------------
Creating train test plot...
----------------------------------------
----------------------------------------
Saving weights...
----------------------------------------
```

Figura A.4: Fin del entrenamiento