Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO DE FIN GRADO

Grado en Ingeniería informática

Estudio y Aplicación de Redes Convolucionales a la Clasificación de Imágenes Estáticas

---

Alumno: Silvia Duque Moro

Tutor: Teodoro Calonge Cano

Índice general

Índice de figuras

Índice de cuadros

Agradecimientos

En primer lugar, gracias a los profesores que me formaron durante estos años, y en especial a mi tutor, Teodoro, no solo por haberme apoyado en la realización de este trabajo de fin de grado, sino por brindarme la oportunidad de poder realizar el mismo sobre el tema propuesto.

Gracias a Jaime, ya que se ha convertido en un gran apoyo este último año, ayudándome siempre que ha podido. Gracias por estar ahí, a pesar de la distancia.

Y sobre todo, gracias a mis padres, por brindarme su apoyo en todo momento, tanto en el desarrollo de los trabajos de fin de grado como a lo largo de estos cinco años de universidad, haciendo que los momentos difíciles fueran siempre más llevaderos y haciéndome ver que al final todo tiene una solución.

Resumen

La visión computacional es una de las áreas que ha avanzado más rápidamente en los últimos años gracias al Aprendizaje Profundo. En este proyecto se aborda uno de sus problemas centrales: la clasificación de imágenes estáticas. En particular, se plantea un estudio de la aplicación de las redes convolucionales en este reto, puesto que es la técnica más usada para ello, de acuerdo con la bibliografía actual sobre este tema. De este modo, no simplemente se profundiza en los fundamentos teóricos, sino que se hace uso del framework *keras* y de la librería *tensorflow* para construir estos modelos.

Cabe destacar la oportunidad que se ha tenido de efectuar este estudio sobre un problema real, la detección de patologías a partir de imágenes oculares.

Como temática principal, se ha abordado el diagnóstico del edema macular y la retinopatía diabética a partir de un imagen de fondo de ojo. En torno a este problema, se ha intentando, no solo diseñar una estructura de red neuronal que proporcione la mayor precisión posible, sino también poner de manifiesto el impacto de la utilización de diversas técnicas recomendadas de cara a mejorar esta precisión. Los resultados obtenidos se consideran satisfactorios, llegándose a unas precisiones del 91,79 % y del 87,86 % respectivamente.

En segundo plano, también se realiza un estudio sobre un pequeño conjunto de datos, el cual ha permitido corroborar el potencial de esta técnica, proporcionando unos buenos resultados a pesar de disponer de pocas muestras.

Finalmente, se ha desarrollado una sencilla aplicación web con *Flask* para permitir que estos modelos puedan ser utilizados por un usuario que disponga de unos conocimientos básicos de Informática. En este sentido se ha considerado de especial interés la utilización de contenedores Docker para favorecer la portabilidad y facilidad de su instalación y despliegue.

Abstract

Computater vision is one of the areas that has advanced most quickly in recent years through Deep Learning. In this project, one of the main problems is covered: static image classification. In particular, a study of the application of convolutional networks is proposed, since it is the most used technique for this, according to the current bibliography on this subject. In this way, one does not simply delve into the theoretical foundations, but uses the *keras* framework and the *tensorflow* library to build these models.

It should be noted the opportunity that this study has had on a real problem, the detection of pathologies from ocular images.

As a main theme, the diagnosis of macular edema and diabetic retinopathy has been covered from an eye fundus image. Concerning this problem, it has been tried, not only to design a neural network structure that provides the highest possible precision, but also to highlight the impact of the use of various recommended techniques in order to improve this accuracy. The results obtained are considered satisfactory, reaching accuracies of 91.79 % and 87.86 % respectively.

In the background, a study is also carried out on a small data set, which has allowed the corroboration of the potential of this technique, providing good results despite having few samples.

Finally, a simple web application has been developed with Flask to allow these models to be used by a user with basic computer skills. In this sense, it has been considered of special interest the use of Docker containers to favor portability and ease of installation and deployment.

Capítulo 1

Introducción

En los últimos años, el auge de la Inteligencia Artificial se ha visto reflejado en todos los ámbitos. Son constantes las menciones relativas a Inteligencia Artificial, Machine Learning y Deep Learning en multitud de artículos, sin ser estos necesariamente de índole tecnológica.

Inicialmente, se considera de interés realizar una breve aclaración entre estos tres términos. En la Figura 1.1 [13] se puede observar cómo la Inteligencia Artificial contiene al Machine Learning, y este a su vez, al Deep Learning; es decir, la Inteligencia Artificial es el concepto general que abarca todo y que surgió inicialmente. Así pues, la Inteligencia Artificial es la rama de la Informática que está relacionada con la automatización del comportamiento inteligente [15], es decir, estudia cómo lograr que las máquinas realicen tareas propias del ser humano. Más adelante, surgió el Aprendizaje Automático (*Machine Learning*) como un conjunto de técnicas, cuyo denominador común era el manejo de conocimiento implícito a través de ejemplos. Por último, surgió el Aprendizaje Profundo (*Deep Learning*), como un paradigma basado en combinar múltiples sistemas de Aprendizaje Automático en forma de capas, ya sean de idéntica o diferente naturaleza.

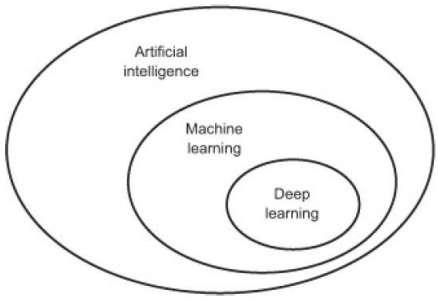


Figura 1.1: Relación entre los conceptos de inteligencia artificial, aprendizaje automático y aprendizaje profundo.

El origen de este nuevo paradigma se remonta a la presentación del modelo de McCullot Pitts en 1943, el cual pretendía simular el funcionamiento de una neurona mediante un modelo matemático. Este modelo fue el detonante de la aparición de las primeras arquitecturas de redes neuronales, aunque esta línea de investigación se paralizó debido a la no disponibilidad de la capacidad computacional necesaria.

De este modo, el incremento de la capacidad computacional, así como el aumento del volumen y tipología de los datos, fue el detonante del apogeo del Aprendizaje Profundo.

En este contexto, en el que el procesamiento de los datos no estructurados (imágenes, textos y audios) se ha convertido en una importante fuente de información, la visión computacional es, hoy en día, una de las áreas que ha avanzado más rápidamente gracias al Aprendizaje Profundo. Vista la importancia adquirida de este área, siendo un tema no tratado de forma directa en mi formación académica, se desarrolla el presente TFG. Así pues, la visión computacional hace referencia al estudio del desarrollo de técnicas que permiten a los ordenadores comprender el contenido de las imágenes, ya sean estáticas o dinámicas como videos.

En particular, el objetivo del presente TFG radica en extraer, analizar y comprender información útil de imágenes. En este caso, se ha decidido optar por la problemática relativa a la clasificación de imágenes estáticas, sustentándose el groso de dicho TFG en este cometido.

Así pues, la clasificación de imágenes consiste básicamente en asignar una de las posibles etiquetas a una imagen dada. Este es uno de los problemas centrales de la visión computacional, sobre el que se basan otros más complejos como la detección de objetos o la segmentación. [9]

En este TFG, se procede a realizar un estudio de la aplicación de las redes convolucionales en este reto, debido a la presencia de estas redes neuronales en las arquitecturas que conforman actualmente el estado del arte.

En este caso concreto, como consecuencia del interés que suscita la realización del estudio en el ámbito de la salud, se aplican las redes convolucionales con el objetivo de ayudar en la determinación del diagnóstico médico, en particular, del diagnóstico de una patología ocular a partir de una imagen.

Cumpliendo con dicha preferencia, este TFG se desarrolla en torno a imágenes oculares. Para ello, se procede a utilizar como temática principal, imágenes del fondo de ojo, a partir de las cuales se pretende predecir el diagnóstico relativo a la retinopatía diabética y al edema macular. Con tal fin, se utiliza el conjunto de datos *Indian Diabetic Retinopathy Image Dataset*, el cual se puede encontrar en [7]. La motivación se basa en el hecho de que la retinopatía diabética es una de las discapacidades visuales más frecuentes, responsable, con el tiempo, de desembocar en ceguera. Es crucial su detección prematura, para detener su avance y paliar sus efectos en el futuro. De esta manera, la aplicación de técnicas de

visión computacional permitiría la identificación masiva, rápida y de bajo coste de dicha enfermedad.

En un segundo plano, también se procede a realizar un breve estudio sobre un pequeño conjunto de datos relativo a imágenes oculares proporcionado por el IOBA (Instituto Universitario de Oftalmobiología Aplicada); en este caso centrando la tarea en el reconocimiento de neovasos y conjuntiva. El objetivo de este breve estudio radica en analizar el alcance del funcionamiento de las redes convolucionales cuando el conjunto de datos es pequeño, así como de explorar la aplicación de estas técnicas a imágenes tomadas de manera diferente y de otra región del ojo.

1.1. Estructura

En vista a los objetivos establecidos, la estructura de este TFG se resume en:

- **Capítulo 2. Gestión del proyecto**. En este apartado se presentan las tareas relativas a la gestión del proyecto realizada, necesarias para garantizar el éxito del proyecto.
- **Capítulo 3. Fundamento teórico**. En él se explica la teoría relativa a las redes convolucionales sobre la que se sustenta este TFG. Para ello, inicialmente se presentan los conceptos fundamentales relativos a las redes neuronales, con el objetivo de que una persona no especializada en dicho campo, sea capaz de comprender el modelo posterior.
- **Capítulo 4. Construcción de una red neuronal convolucional**. Aquí se expone la utilización del framework *keras* sobre *Tensorflow* y sus dos diferentes aproximaciones de aprendizaje proporcionadas por esta librería.
- **Capítulo 5. Descripción y preprocesamiento de los datos**. En esta parte se aborda la descripción de los distintos conjuntos de imágenes que se utilizan en el desarrollo del TFG, así como el preprocesamiento realizado a cada uno de ellos.
- **Capítulo 6. Modelado de los datos**. Se exponen las diferentes estructuras de redes neuronales convolucionales contempladas en la fase de modelado para cada una de las tareas de clasificación, presentando los resultados obtenidos.
- **Capítulo 7. Visualización de las redes neuronales convolucionales**. En esta sección se exponen dos técnicas que permiten que el aprendizaje de las redes convolucionales sea explicable visualmente.
- **Capítulo 8. Aplicación**. En él se aborda el desarrollo de una pequeña aplicación web que permita mostrar los resultados obtenidos, detallando los procesos relativos al análisis y diseño de la misma.
- **Capítulo 9. Conclusiones**. Finalmente, se presentan las conclusiones finales de esta memoria, así como el trabajo futuro a desarrollar.

Capítulo 2

Gestión del proyecto

En este capítulo se exponen las tareas relativas a la gestión de proyectos llevadas a cabo para la realización de este TFG ya que, tal y como se ha enunciado en la formación académica proporcionada, una correcta gestión del proyecto será determinante en la consecución de los objetivos.

2.1. Metodología de trabajo

En el marco de las metodologías relativas al desarrollo del software destacan dos vertientes claramente diferenciadas:

- Metodologías tradicionales: Se sigue un proceso secuencial bien definido, el cual se basa en una planificación predictiva, una documentación exhaustiva, un control estricto y un producto final acorde a las especificaciones definidas.
- Metodologías ágiles: Se caracterizan por su flexibilidad y adaptatividad, estando orientadas al cambio y a las necesidades emergentes.

A pesar del auge de las metodologías ágiles hoy en día, no se puede considerar que un tipo de metodología sea mejor que otra, sino que la elección se debe basar en cada proyecto concreto.

Debido a la familiarización obtenida en los estudios universitarios en el desarrollo de proyectos empleando metodologías tradicionales, se ha optado por utilizar este tipo de metodología. Por otra, se considera que el dinamismo que aporta una metodología ágil sólo se empieza a rentabilizar a partir de un cierto número de miembros del equipo de trabajo, que no es el caso, ya que sólo hay interacción estudiante-tutor.

Dentro de las metodologías tradicionales, existe una amplia variedad de alternativas. Tras realizar un breve estudio, se opta por llevar a cabo un desarrollo por incrementos, que combina el modelo en cascada con la filosofía iterativa de la creación de prototipos.

Esta metodología se caracteriza por la obtención del producto final a través de una serie de incrementos que sucesivamente añaden funcionalidad.

Este enfoque se adapta fácilmente a este proyecto, ya que gracias a su carácter modular, cada incremento puede estar constituido por un módulo. Además, la agregación de nuevas funcionalidades en cada incremento permite un mayor seguimiento de la consecución de los objetivos establecidos al tutor y una mayor flexibilidad dentro de las restricciones propias de una metodología tradicional.

2.2. Entregables del proyecto

Debido a la utilización de un desarrollo por incrementos, los entregables están ligados directamente a los incrementos, de forma que el entregable relativo a cada incremento constituye una versión operativa con ciertas funcionalidades del producto final:

- Incremento 1: Modelo del diagnóstico del edema macular y documentación.
- Incremento 2: Modelo del diagnóstico de la retinopatía diabética y documentación.
- Incremento 3: Modelo de la presencia de neovasos y conjuntivalización y documentación.
- Incremento 4: Aplicación web y documentación.

2.3. Planificación

En esta sección se aborda la planificación del proyecto, siendo un factor clave a la hora de garantizar su finalización en el tiempo estipulado.

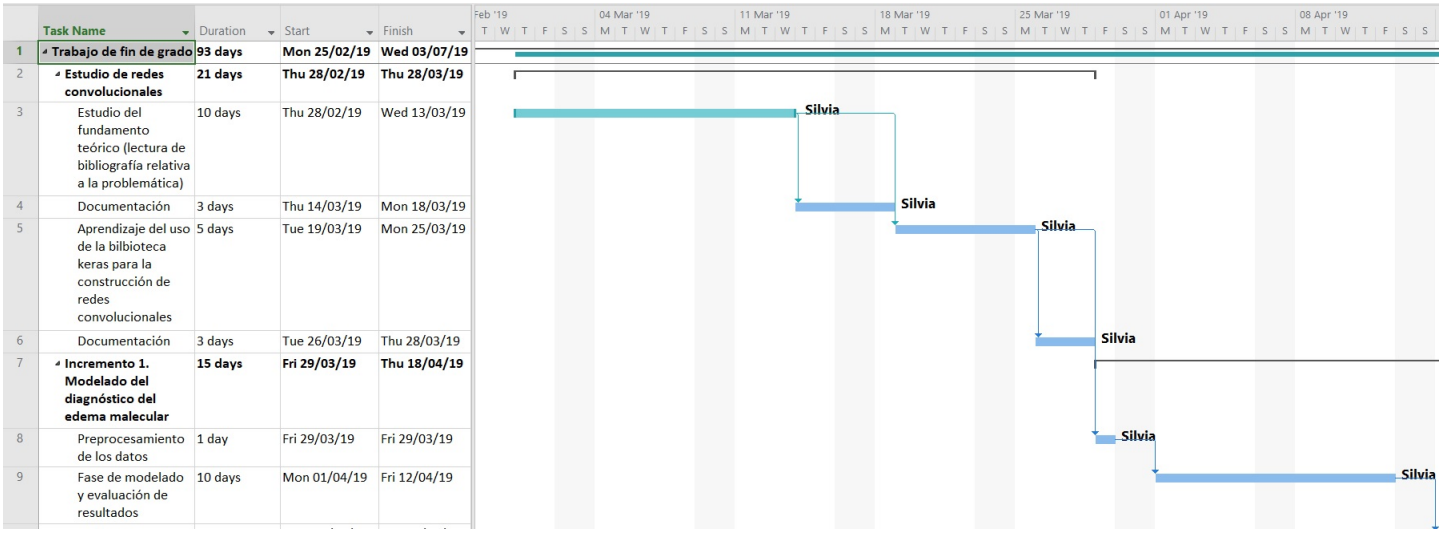
En primer lugar, hay que tener en cuenta que el Trabajo de Fin de Grado tiene una carga de 12 créditos en el Grado de Ingeniería Informática de la Universidad de Valladolid, los cuales equivalen a 300h de trabajo. El proyecto se inicia el 25 de febrero y su fecha de finalización estimada es el 21 de junio, disponiendo de 17 semanas para su elaboración, con una dedicación estimada de 20h a la semana (4h diarias, excluyendo los fines de semana).

A continuación, se expone la planificación inicial realizada y las variaciones acontecidas respecto a esta planificación inicial.

2.3.1. Planificación inicial del proyecto

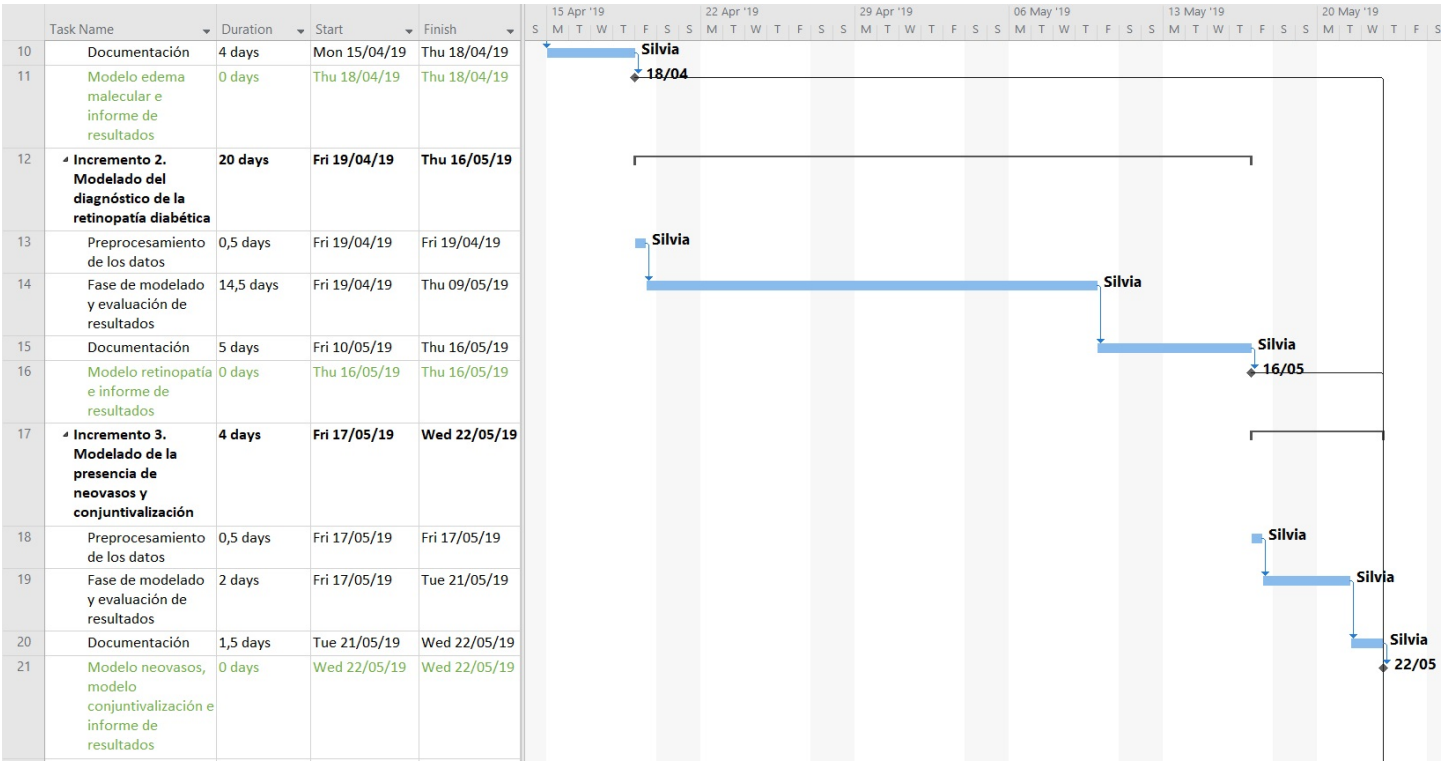
En este apartado, se presenta la planificación inicial del proyecto, en la que se establece el conjunto de tareas que se pretenden abordar, así como la duración estimada de las mismas.

A continuación, se expone en las figuras 2.1, 2.2 y 2.3 el desglose relativo a esta planificación, junto a su diagrama de Gantt correspondiente, el cual permite una visualización más directa de la planificación.



	Task Name	Duration	Start	Finish
1	**Trabajo de fin de grado**	**93 days**	**Mon 25/02/19**	**Wed 03/07/19**
2	**Estudio de redes convolucionales**	**21 days**	**Thu 28/02/19**	**Thu 28/03/19**
3	Estudio del fundamento teórico (lectura de bibliografía relativa a la problemática)	10 days	Thu 28/02/19	Wed 13/03/19
4	Documentación	3 days	Thu 14/03/19	Mon 18/03/19
5	Aprendizaje del uso de la bilbioteca keras para la construcción de redes convolucionales	5 days	Tue 19/03/19	Mon 25/03/19
6	Documentación	3 days	Tue 26/03/19	Thu 28/03/19
7	**Incremento 1. Modelado del diagnóstico del edema malecular**	**15 days**	**Fri 29/03/19**	**Thu 18/04/19**
8	Preprocesamiento de los datos	1 day	Fri 29/03/19	Fri 29/03/19
9	Fase de modelado y evaluación de resultados	10 days	Mon 01/04/19	Fri 12/04/19

Figura 2.1: Planificación del proyecto - parte 1.



	Task Name	Duration	Start	Finish
10	Documentación	4 days	Mon 15/04/19	Thu 18/04/19
11	Modelo edema malecular e informe de resultados	0 days	Thu 18/04/19	Thu 18/04/19
12	**Incremento 2. Modelado del diagnóstico de la retinopatía diabética**	**20 days**	**Fri 19/04/19**	**Thu 16/05/19**
13	Preprocesamiento de los datos	0,5 days	Fri 19/04/19	Fri 19/04/19
14	Fase de modelado y evaluación de resultados	14,5 days	Fri 19/04/19	Thu 09/05/19
15	Documentación	5 days	Fri 10/05/19	Thu 16/05/19
16	Modelo retinopatía e informe de resultados	0 days	Thu 16/05/19	Thu 16/05/19
17	**Incremento 3. Modelado de la presencia de neovasos y conjuntivalización**	**4 days**	**Fri 17/05/19**	**Wed 22/05/19**
18	Preprocesamiento de los datos	0,5 days	Fri 17/05/19	Fri 17/05/19
19	Fase de modelado y evaluación de resultados	2 days	Fri 17/05/19	Tue 21/05/19
20	Documentación	1,5 days	Tue 21/05/19	Wed 22/05/19
21	Modelo neovasos, modelo conjuntivalización e informe de resultados	0 days	Wed 22/05/19	Wed 22/05/19

Figura 2.2: Planificación del proyecto - parte 2.

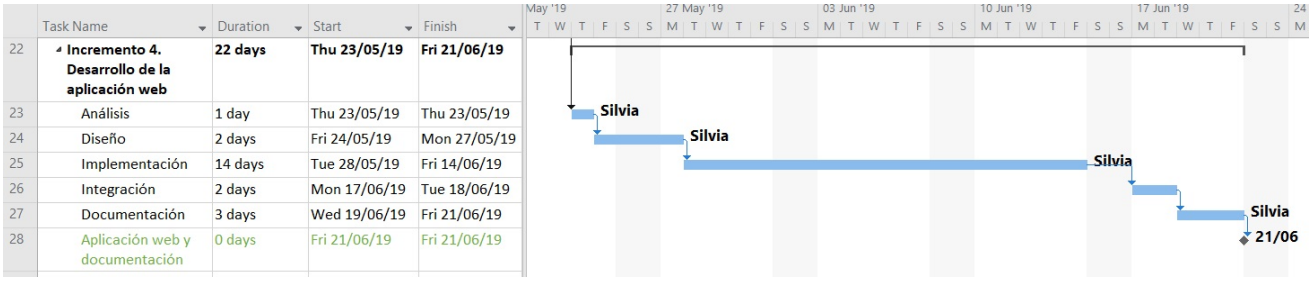


Figura 2.3: Planificación del proyecto - parte 3.

De este modo, se puede observar cómo inicialmente se plantea una fase relativa al estudio de las redes convolucionales, ya que se trata de un tema no impartido en la formación académica. A continuación, se irán abordando las diferentes tareas asociadas a cada uno de los incrementos, obteniendo un producto funcional, que se pueda presentar a cliente, tras cada incremento. Cabe aclarar cómo las reuniones con el tutor están contempladas dentro de las tareas planificadas.

Adicionalmente, se ha estimado una duración de 12 horas en la elaboración de la planificación de este proyecto.

2.3.2. Variaciones respecto a la planificación inicial

Tal y como es de esperar, un seguimiento exhaustivo de la planificación inicial es altamente improbable, ya que es habitual la aparición de imprevistos, los cuales requieren de una adaptación de la planificación para su abordaje.

En el desarrollo de este proyecto, la mayoría de las tareas han sido completadas en los tiempos establecidos, con ligeras desviaciones en las estimaciones de la duración, las cuales se han compensado entre ellas. No obstante, tal y como se comentaba al inicio de la sección, la aparición de imprevistos es habitual, no siendo este proyecto una excepción. Estos imprevistos han requerido de una modificación más notoria en la planificación, siendo los siguientes:

- La obtención de una base de imágenes oculares fue una tarea más ardua de lo esperada, siendo necesario retrasar ligeramente ciertos aspectos de la planificación inicial.

- El tiempo necesario para el entrenamiento de las redes neuronales convolucionales fue superior al esperado, debido a la inexperiencia en el diseño de las mismas. Este imprevisto amplió ligeramente la duración de las tareas relativas a modelado, debido a la extensión de su tiempo de entrenamiento.

- Se detectó una incompatibilidad entre la red neuronal creada y la aplicación diseñada. Para solventar este problema, se optó por utilizar Docker, haciendo uso de una

imagen compatible con ambos. La instalación y configuración de Docker supuso un incremento de 2 días respecto a la planificación inicial.

- Ligado al anterior imprevisto, Docker no se podía ejecutar sobre la máquina virtual proporcionada para el desarrollo de este TFG, ya que la CPU no soportaba las instrucciones *"SSE 4.2"* necesarias para su configuración. Por consiguiente, fue necesario llevar a cabo la instalación de Linux en la máquina personal, añadiendo 1 día a la planificación inicial.

2.4. Gestión de la configuración

Con el objetivo de controlar la evolución del proyecto, se lleva a cabo una gestión de la configuración, la cual nos va a facilitar mantener la integridad del producto desarrollado.

En particular, para abordar esta gestión de la configuración, se ha utilizado la herramienta online GitLab [5], aportando un control de versiones a lo largo del desarrollo del proyecto. De esta manera, no será necesario un control manual de las versiones de los diferentes documentos, sino que dicha herramienta se encargará automáticamente de proporcionar un histórico de cada documento y sus cambios.

Para ello, se ha creado un repositorio que contenga los elementos de configuración, incluyendo tanto los ficheros relativos a los diferentes modelos de redes neuronales como a la aplicación. En la Figura 2.4 se adjunta la estructura del repositorio creado.

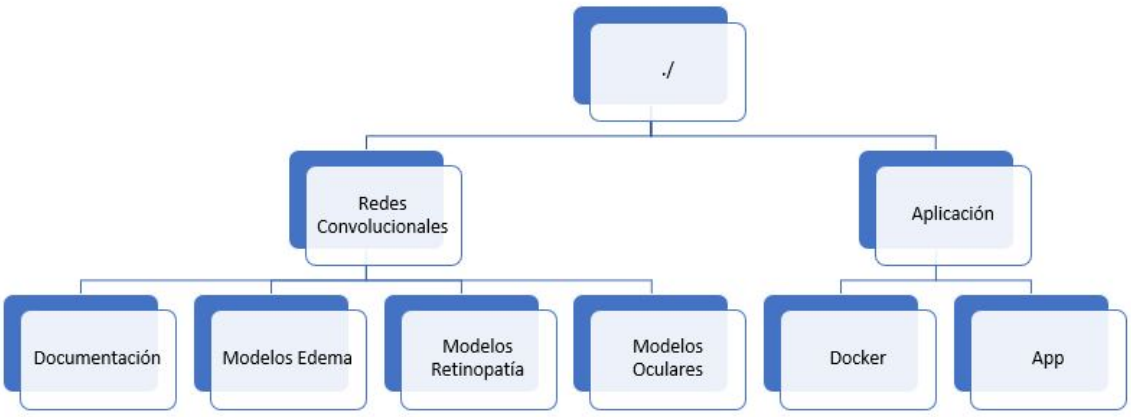


Figura 2.4: Estructura del Repositorio.

De forma más detallada, los elementos de configuración utilizados son:

- Ficheros Dockerfile y .yml incluidos en la carpeta ./Aplicación/Docker (ficheros necesarios para generar y levantar el contenedor).

- Ficheros .py, .html, .css, .js, .png, .h5 incluidos en la carpeta ./Aplicación/App. Se excluyen los ficheros .pyc, ya que se obtienen automáticamente mediante la compilación de los ficheros .py.
- Ficheros .ipynb y .h5 en las carpetas relativas a modelos.

Por otra parte, se opta por hacer uso del editor de LaTeX online *Overleaf* para generar la documentación del proyecto. Al tratarse de una herramienta online, ha posibilitado el trabajar desde diferentes equipos. Con esta herramienta se evita el tedioso proceso instalación de LaTeX, a la vez que incorpora de forma automática las diferentes librerías que se utilizan en el desarrollo de la documentación.

Capítulo 3

Fundamento teórico

En este capítulo, se pretende proporcionar una base sólida de los conceptos teóricos sobre los que se sustentan las redes neuronales convolucionales. De cara a una correcta comprensión de este tipo concreto de red neuronal utilizado en visión computacional, se abordan inicialmente conceptos más genéricos.

3.1. Redes Neuronales Artificiales

Una red neuronal es un modelo computacional inspirado en el funcionamiento del cerebro, es decir, en el sistema nervioso. Tal y como se comentó en el *Capítulo 1: Introducción*, este enfoque no es novedoso, sino que ya fue introducido en 1943 por el neurofisiólogo Warren McCullock y el matemático Walter Pitts, presentando el primer modelo computacional basado en el funcionamiento de una neurona biológica, unidad funcional básica del sistema nervioso. [16]

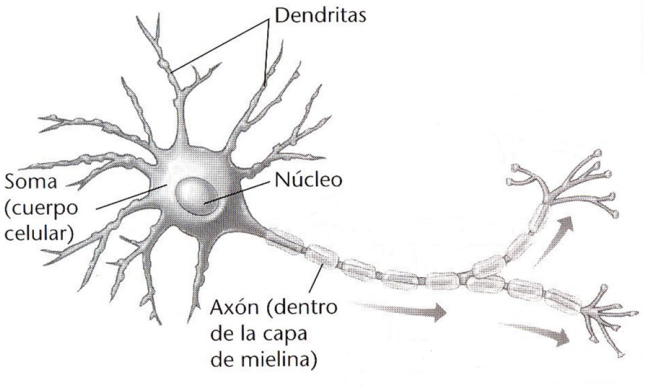


Figura 3.1: Representación simplificada de una neurona biológica.

En la Figura 3.1 se puede visualizar una representación simplificada de una neurona biológica. Básicamente, una neurona recibe las entradas a través de las dendritas, procesan

la información en el soma y distribuye la salida a una o varias neuronas conectadas a través del axón. Así pues, se puede decir que la función principal de las neuronas es la transmisión de los impulsos nerviosos.

El modelo de McCullot Pitts intentó plasmar este comportamiento matemáticamente, dando lugar a las neuronas artificiales. En la Figura 3.2 se puede visualizar una representación gráfica del mismo:

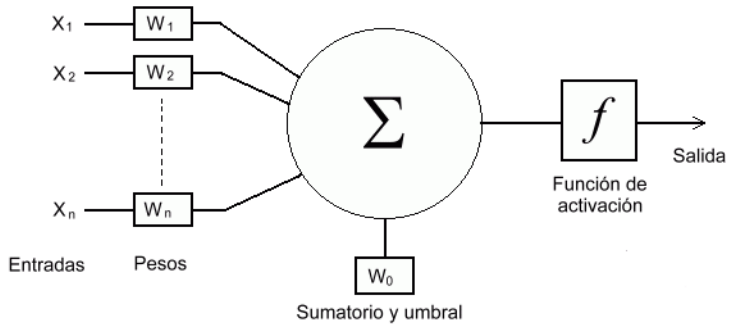


Figura 3.2: Modelo de McCullot y Pitts.

En este modelo de neurona artificial, las entradas $x_1, ..., x_n$ son la analogía de las dendritas; la sinapsis es modeladada a partir de unos pesos $w_1, ..., w_n$, los cuales conforman los parámetros libres del modelo; y el sumatorio representa el soma o núcleo. La salida de este sumatorio se denomina activación o salida analógica (u). De cara a obtener la salida de interés (y), se aplica un filtrado mediante a una determinada función denominada activación. Así pues, la formulación matemática de este modelo es:

$$u = w_0 + \sum_{j=1}^{n} w_j x_j \tag{3.1}$$

$$y = F(u) \tag{3.2}$$

Con esto, se pretende ajustar el valor de los pesos $w_0, ..., w_n$ (parámetros libres del modelo), de acuerdo con la realización de una determinada tarea, la cual condicionará precisamente la elección de la función de activación F.

Tras esta breve explicación del modelo de McCullot Pitts, se puede concluir que las neuronas artificiales se comportan como procesadores simples. De este modo, será su acoplamiento en un sistema distribuido, el que permita que adquieran su verdadero potencial, dando lugar a las redes neuronales.

En 1957, Frank Ronseblatt inventó el perceptrón basándose en el modelo de McCullot Pitts, siendo una arquitectura de red neuronal muy simple, que se convirtió en el primer modelo capaz de aprender de forma autónoma los pesos a partir de un conjunto de ejemplos etiquetados.

Así pues, completando la información relativa al modelo de McCullot Pitts, la función de activación propuesta fue:

$$F(u) = sgn(u) = \begin{cases} +1 & si\ u \geqslant 0 \\ -1 & si\ u < 0 \end{cases} \tag{3.3}$$

Para determinar el valor de los pesos, se propuso un aprendizaje iterativo, que consiste en inicializar los pesos con unos valores aleatorios (siendo recomendable entre [-0.5, 0.5]) e ir modificándolos en cada iteración, si algún valor de la salida no coincide con la salida deseada correspondiente, mediante una adaptación de la regla de Hebb:

$$\triangle\, w_i(t) = d(x)x_i \tag{3.4}$$

Este enfoque se puede generalizar a un problema de clasificación multiclase, poblando la capa de salida con tantas neuronas como clases.

Sin embargo, esta arquitectura presenta muchas limitaciones, siendo la más relevante la necesidad de que el problema sea linealmente separable, para el cual ya existían métodos de resolución.

Estas limitaciones abrieron la veda a planteamientos relativos al aumento del número de capas con el objetivo de aumentar la precisión de las redes neuronales. En 1974, Werbos propuso en su tesis doctoral el uso de nueva metodología en el proceso de entrenamiento las redes neuronales, denominada propagación hacia atrás. Más tarde, en 1986, Hinton, Rumelhart y Willians publicaron el artículo *"Learning representations by back-propagating errors"*, donde introducían el concepto de capas ocultas y popularizaban el algoritmo de propagación hacia atrás, dando lugar al nacimiento del perceptrón multicapa: una generalización del perceptrón simple. Así pues, el perceptrón multicapa está compuesto de una capa de entrada, una capa de salida y una serie de capas ocultas o intermedias (las cuales van a permitir a las redes neuronales aprender características más complejas). En la Figura 3.3 se puede observar esta arquitectura. En esta topología, se puede comprobar cómo las salidas de las neuronas sólo sirven de entrada a las de las capas siguientes [11], hasta llegar a la salida.

Así pues, la propagación hacia atrás es un procedimiento que permite ajustar los pesos de forma iterativa minimizando la diferencia entre la salida obtenida y la deseada. Para ello, como se verá más adelante, se plantea un problema de minimización del error cuadrático medio entre ambas cantidades.

De este modo, se diferencian dos fases claras en el aprendizaje de los pesos de la red neuronal:

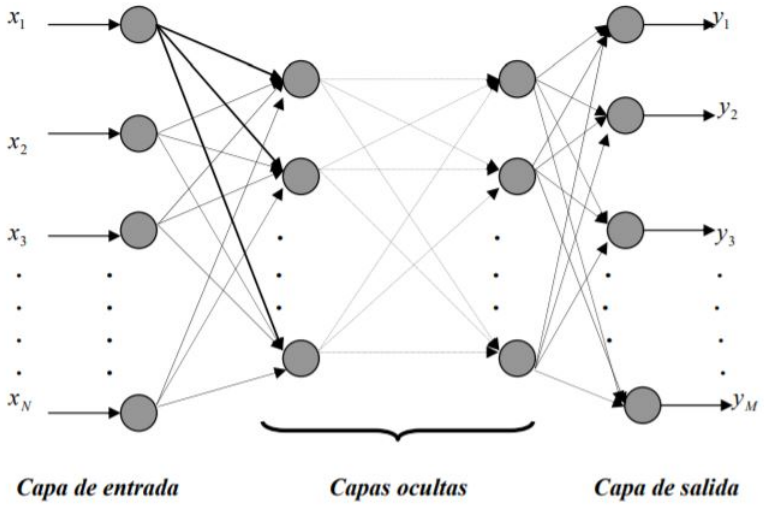


Figura 3.3: Arquitectura del Perceptrón Multicapa.

- Propagación hacia delante, que calcula, a partir de unos valores de entrada, el resultado de la salida de la red.

- Propagación hacia atrás, en la que error cuadrático medio se propaga hacia atrás modificando el valor de los pesos, utilizando para ello el método del descenso del gradiente.

Esta arquitectura asentó las bases de las redes neuronales, aunque el estudio más detallado de las mismas se paralizó debido a la escasa capacidad computacional de aquella época. Con el avance en la fabricación de procesadores cada vez más rápidos y con capacidad de computar en paralelo, esta limitación tecnológica ha ido disminuyendo ostensiblemente. Este incremento de la capacidad computacional ha sido el detonante para que esta línea de investigación se avivase, dando lugar a una gran variedad de arquitecturas de redes neuronales.

3.2. Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales fueron presentadas en 1989 por Yann LeCun como resultado de la experimentación relativa a su posdoctorado sobre redes neuronales conectadas localmente. Demostró que estas conexiones locales proporcionaban un buen rendimiento en el reconocimiento de patrones visuales. La motivación de este estudio se basó en la neurobiología, en particular, en los estudios de Hubel y Wiesel, que determinaron que las neuronas del el cortex visual de un gato era localmente sensibles y de orientación selectiva estática. [17]

Sin embargo, tal y como se expuso en el Capítulo 1, las redes neuronales artificiales han tomado importancia en los últimos años debido a la necesidad de procesar datos no estructurados, siendo considerados una gran fuente de información. En este contexto, las Redes Convolucionales se consideran una de las mejores alternativas de cara a abordar el procesamiento de imágenes. [21]

Debido a que las Redes Convolucionales son redes neuronales conectadas localmente, se describen las mismas como un caso especial del perceptrón multicapa (arquitectura descrita en el apartado 3.1, en la que las neuronas están totalmente conectadas) adecuado para el reconocimiento de patrones, incluso cuando su apariencia varía de alguna manera (traslación, escalado, rotación,...), es decir, casos en los que los datos presentan una topología. En el caso particular de las imágenes, abordado en el presente TFG, se puede considerar cada una de ellas como una cuadrícula bidemensional de píxels.

El propio nombre de estas redes neuronales, Redes Convolucionales, hace referencia a la operación matemática característica de estas redes, la *convolución*. En [10] se definen las Redes Convolucionales como "redes neuronales simples que usan la convolución en lugar de la típica multiplicación de matrices en al menos una de sus capas".

3.2.1. Convolución

Matemáticamente, la convolución es un operador matemático sobre dos funciones, cuyo resultado es una tercera función. En el caso discreto, este cálculo se resumiría como:

$$s(t) = (I * K)(t) = \sum_{a=-\infty}^{\infty} I(a)K(t-a) \tag{3.5}$$

donde:

- I hace referencia a los datos de entrada, siendo estos habitualmente un array multidimensional de los datos.
- K es el kernel, siendo éste un array multidimensional de los parámetros cuyos valores se desea aprender; habitualmente de menor tamaño al relativo a los datos de entrada.

Tanto el array relativo a I como a K, se suelen denominar tensores, ya que cada elemento de estos se debe almacenar por separado explícitamente. De este modo, el valor de dichas funciones para los puntos que no estén almacenados explícitamente se consideran 0. Como consecuencia, la suma anteriormente descrita se reduce a ejecutarla sobre un conjunto finito de números. [10]

La aplicación de esta operación calcula, en cierto sentido, cómo de similar es el kernel respecto a la porción de entrada evaluada en el momento. Así pues, esta salida tomará valores altos, cuando el núcleo sea similar a la parte de la entrada. Esta salida, suele denominarse *mapa de características*.

Esta operación se puede extender a dos dimensiones, abordando así la problemática relativa a este TFG. De este modo, la entrada I simplemente será una imagen bidimensional (objeto de estudio), pasando el kernel K a ser también bidimensional. En este caso, la operación es:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.6}$$

Esta operación satisface la propiedad conmutativa, resultando la siguiente equivalencia más fácil de implementar, debido al menor rango de posibles valores de m y n:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{3.7}$$

El signo negativo que se observa en la operación anterior, se puede reemplazar dando lugar a la operación *correlación cruzada*:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{3.8}$$

En la Figura 3.4 [10] se ilustra la operación convolución (en su definición base) en el caso bidimensional. Tal y como se comentó, el kernel es de menor tamaño que el array bidimensional relativo a los datos de entrada, por lo que el kernel se va deslizando sobre los datos de entrada. En consecuencia, los parámetros del kernel se comparten, teniendo que ser sus valores únicos en la totalidad de la operación.

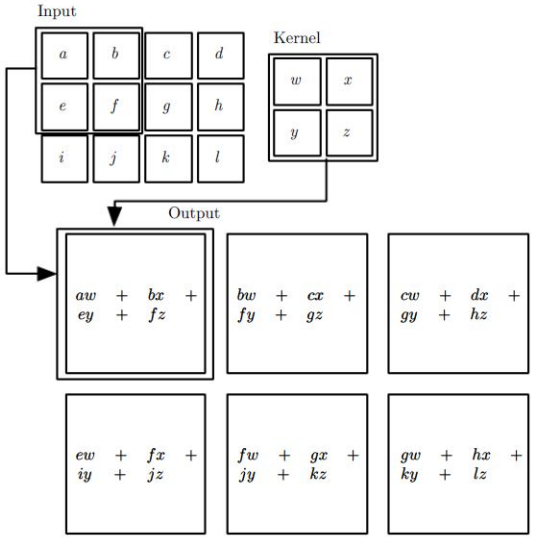


Figura 3.4: Ejemplo de la operación convolución en el caso bidimensional.

Una vez explicada la convolución, se exponen las tres ideas fundamentales por las que se apostó sustituir la típica multiplicación de matrices utilizada hasta el momento en las redes neuronales por la operación convolución.

- **Conectividad dispersa**: En las redes neuronales tradicionales, tomando como ejemplo el perceptrón multicapa explicado en el apartado 3.1, cada neurona de una determinada capa interactúa con todas las de la capa anterior, estando definida cada una de estas interacciones por un parámetro distinto. En el caso de las redes convolucionales, el número de interacciones es menor, debiéndose esto a la utilización de un kernel de menor tamaño que la entrada. En el procesamiento de imágenes, esta idea juega un papel muy importante, ya que permite detectar características simples como pueden ser los bordes, dando lugar la composición de éstas a la detección de características complejas, que pueden ser determinantes en tareas como la clasificación de imágenes. Asimismo, la utilización de un menor número de parámetros lleva asociada una reducción de los requisitos de memoria del modelo y una mejora de su eficiencia. En la Figura 3.5 se muestra gráficamente este concepto. En la representación superior, relativa a la conectividad dispersa, se puede observar cómo a la neurona *s3* sólo le afectan tres neuronas de entrada (el tamaño del kernel utilizado es 3). Sin embargo, en la representación inferior, se puede observar cómo esta neurona se ve influenciada por todas las neuronas de entrada. No obstante, cabe destacar cómo las neuronas de las capas más profundas pueden conectar indirectamente con la mayoría de la entrada, aunque las conexiones directas en una red convolucional son muy dispersas.

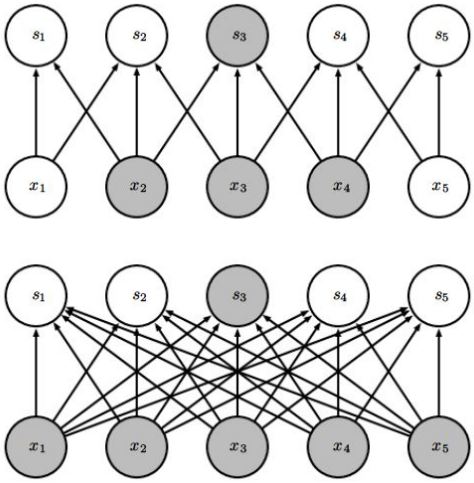


Figura 3.5: Comparativa Conectividad Dispersa (arriba) con Conectividad Densa (abajo).

- **Parámetros compartidos**: En las redes neuronales tradicionales, cada elemento de la matriz de pesos se utiliza una única vez, sin embargo, en las convolucionales,

cada peso del kernel se usa en cada operación relativa al deslizamiento sobre los datos de entrada; de este modo, se aprende exclusivamente un conjunto de pesos.

- **Representaciones equivalentes**: Esta propiedad hace referencia a que si la entrada sufre un cambio de traslación, la salida obtenida lo hará del mismo modo. Matemáticamente, una función f es equivariante a una función g si

$$f(g(x)) = g(f(x)) \tag{3.9}$$

  Sin embargo, la convolución no es equivariante a otros cambios como la rotación o el escalado, siendo necesario utilizar otros métodos en estos casos.

3.2.2. Restricciones estructurales

Con el objetivo concretar la estructura de las Redes Convolucionales, Yann LeCun y Yoshua Bengio publicaron en 2003 el artículo "Convolutional Networks for Images, Speech, and Time Series", en el cual se presentaban las siguientes restricciones estructurales [17], algunas de ellas haciendo referencia a las ideas fundamentales que sustentan el uso de las redes Convolucionales comentadas en el apartado anterior:

- **Extracción de características**: Cada neurona debe tomar sus entradas de un campo receptivo local de la capa anterior, extrayendo así características locales. La situación exacta de las características no es importante, siempre y cuando se mantenga su posición relativa con otras.
- **Mapeo de características**: Cada capa convolucional debe estar compuesta por varios mapas de características.
- **Submuestreo**: A cada capa convolucional la debe seguir otra que realice un submuestreo (promedios locales) con el objetivo de reducir la sensibilidad del mapa de características a cambios.

Según estas restricciones, la estructura típica de una capa convolucional se puede observar en la Figura 3.6 [10]. Así pues, una capa convolucional está constituida por 3 partes perfectamente diferenciables. La primera de ellas se corresponde con la aplicación de la operación convolución una o varias veces. En segundo lugar, se emplea una función de activación no lineal sobre el resultado de la primera parte, como una especie de corrector. Por último, tal y como indica la restricción 3, se usa una función pooling que modifica la salida reduciendo su tamaño.

3.2.3. Pooling

La función *pooling* tiene como objetivo principal reducir el tamaño del mapa de características disminuyendo, como consecuencia, el número de parámetros y los cálculos que se deben efectuar en la red. Así pues, la salida se reemplaza por un nuevo resultado obtenido a partir de una operación estadística aplicada sobre valores de la salida cercanos. Las dos funciones *pooling* más conocidas son:

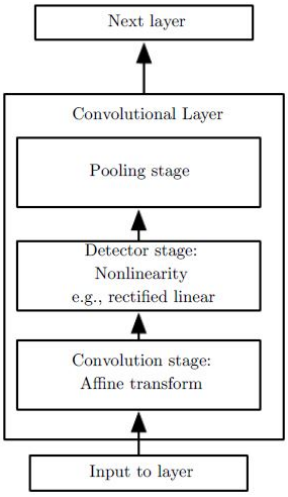


Figura 3.6: Estructura de una capa convolucional.

- Max pooling: En este caso la operación estadística es el máximo; de este modo, se calcula el máximo sobre subconjuntos de valores cercanos entre sí, o en otras palabras, identifica cuál es el número mayor entre los números que conforman cada subconjunto. En la Figura 3.7 se puede visualizar esta operación con el objetivo de clarificar el concepto. Aquí, se consideran cuatro subconjuntos de valores excluyentes (sombreados en distintas tonalidades), obteniéndose el máximo de cada uno de ellos.

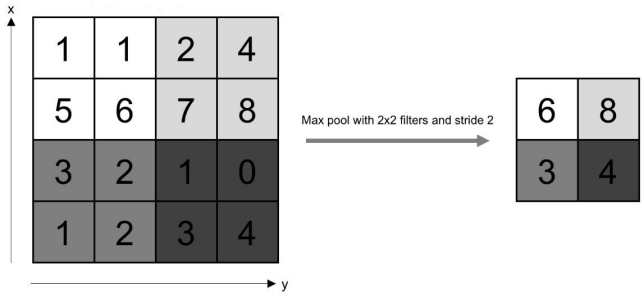


Figura 3.7: Operación Max Pooling.

- Average pooling: En este caso, la operación estadística es la media aritmética. Lo que se calcula es el promedio sobre los valores que conforman un subconjunto.

No obstante, el transfondo de la aplicación de esta operación no se basa exclusivamente en una reducción de la dimensionalidad, sino que esta reducción se efectúa con el objetivo de capacitar a la red neuronal de la habilidad de ser invariante frente a pequeños cambios de traslación, siendo esta invarianza una de las características principales que definen a

las redes convolucionales. De este modo, se capacita a la red neuronal para clasificar las imágenes correctamente, incluso cuando la posición de los objetos en la imagen varíen. Es decir, al utilizar este tipo de redes nos importa la presencia o ausencia de ciertas características, no la localización de las mismas.

Sin embargo, estas redes neuronales, tal y como se comentó al inicio del capítulo, no son únicamente invariantes a la traslación. La extensión de la invarianza a otras formas de distorsión se consigue aplicando dicha operación sobre distintas capas convolucionales anteriores (cuyos parámetros son independientes). De este modo se pueden aprender distorsiones contempladas en las diferentes convoluciones, como puede ser la rotación o el escalado.

Entre las dos funciones de *pooling* descritas anteriormente, se recomienda utilizar la función *max pooling*. Esta elección se sustenta en la operación estadística utilizada, ya que en el caso de la función *max pooling* se hace uso del máximo, calculando los mayores valores de unos determinados subconjuntos, extrayendo en consecuencia los valores (o características) más relevantes. En el caso de la función *average pooling* se hace uso de la media aritmética extrayendo valores (o características) promedios, los cuales agrupan toda la información proporcionada, pudiendo propiciar que los agrupamientos obtenidos no sean lo suficientemente significativos. Es decir, la información proporcionada al utilizar el máximo es más informativa. No obstante, se pueden encontrar diversos artículos en los que se ha optado por utilizar esta función de agrupamiento.

3.2.4. Estructura de una Red Convolucional en la Clasificación de Imágenes

En el apartado 3.2.2 se expusieron una serie de restricciones estructurales, las cuales debían satisfacerse en el diseño de la estructura de la red neuronal convolucional. Basado en esto, se mostraba en la Figura 3.6 la estructura típica de una capa convolucional. En este apartado se pretende especificar la estructura de una red convolucional sencilla para tratar de llevar a cabo la clasificación de imágenes.

En primer lugar, el conjunto de datos de entrada está constituido por un conjunto de imágenes, que están constituidas por tres canales, siguiendo el esquema de codificación RGB. Se trata de un modelo aditivo tal que, un color se representa como la suma de los tres primarios (rojo, azul y verde). Así pues, el conjunto de imágenes de entrada se descompone en estos tres canales: uno por cada color primario. En la Figura 3.8 se puede observar esta descomposición gráficamente. De este modo, la operación convolución se aplica de forma independiente sobre cada uno de estos canales. Las salidas obtenidas se suman para obtener el resultado final. Bajo esta perspectiva, se utiliza un kernel distinto para cada canal y posteriormente se resumen estos resultados intermedios a partir de la operación suma.

En la Figura 3.9 se presenta un pequeño ejemplo aclaratorio de esta metodología [8]. De esta manera, se puede observar cómo una imagen perteneciente a los datos de entrada

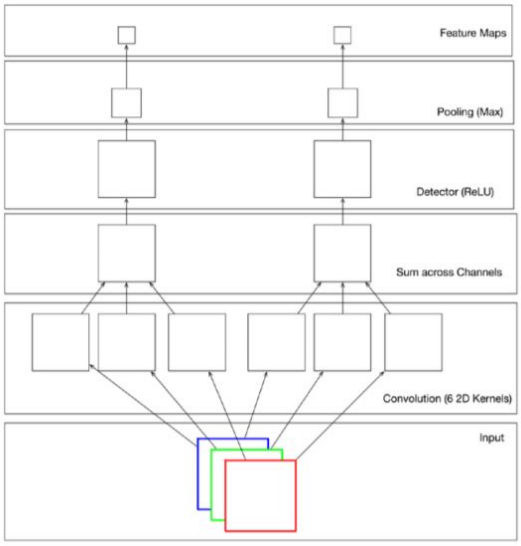


Figura 3.8: Estructura de la primera capa convolución cuando los datos de entrada son imágenes.

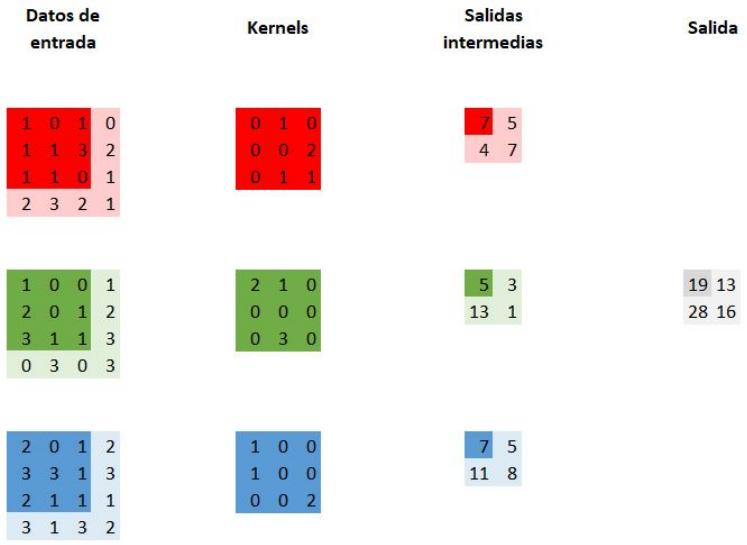


Figura 3.9: Ejemplo de la aplicación de la operación convolucional sobre una imagen.

se desglosa en los tres canales relativos a los colores primarios. En esta Figura, también se puede observar cómo cada canal tiene su propio kernel, de tal forma, que se aplica cada kernel a su canal correspondiente, mediante la operación convolución proporcionando las salidas intermedias. Así pues, para obtener cada uno de los valores de la salida intermedia

se desplaza el kernel (tanto en horizontal como en vertical) hasta abarcar la totalidad de los datos de entrada relativos al canal correspondiente. Tal y como se comentó en el apartado 3.2.1, en cada deslizamiento, se multiplican los elementos correspondientes y se suman. Por último, para obtener la salida final de la aplicación de este filtro, se suman las salidas intermedias.

Retomando la explicación relativa a la Figura 3.8, se puede observar cómo se aplica la función de activación *ReLu* sobre las salidas obtenidas tras la operación convolución. Por último, se aplica la función de agrupamiento *max pooling*, tal y cómo se expuso en el apartado 3.2.3. Las salidas obtenidas se denominan *mapas de características* o *filtros*, siendo habitualmente las entradas de una nueva capa (convolucional o totalmente conectada).

En esta Figura 3.8, también se pone de manifiesto la posibilidad de aplicar múltiples secuencias convolución-activación-pooling en paralelo. Así, a partir de los mismos datos de entrada, se obtienen múltiples salidas diferentes.

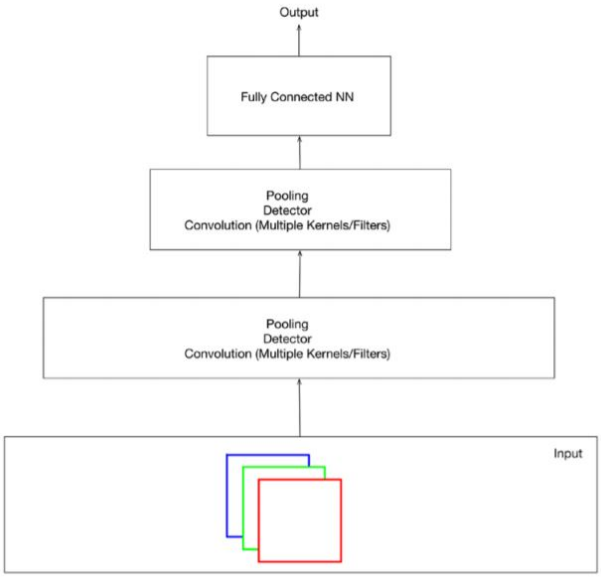


Figura 3.10: Estructura de una red convolucional sencilla cuando los datos de entrada son imágenes.

Tras haber abordado en detalle los elementos constituyentes de una red neuronal convolucional, se ilustra en la Figura 3.10 la integración de estos, dando lugar a una estructura sencilla de red convolucional. En particular, se puede observar cómo está compuesta por dos bloques convolución-activación-pooling y una capa final totalmente conectada, la cual proporciona la categoría predicha.

En cuanto a la generalización de dicha estructura, una red convolucional está compuesta por el número preciso de bloques relativos a las operaciones convolución/*pooling* y

un último bloque relativo a una capa totalmente conectada.

3.2.5. Variantes de la Operación Convolución

En la exposición de la estructura general de una red convolucional, se expuso que una red de esta tipología está conformada por varios bloques relativos a la aplicación de las operaciones convolución y *pooling*, y por una capa final totalmente conectada. No obstante, de cara a la clasificación de imágenes, se requiere de la extracción de más de un mapa de características. Por tanto, es necesario utilizar más de un kernel en cada capa de la red, con el fin de no extraer únicamente una característica por capa. Para ello, generalmente se opta por aplicar la operación convolución de forma paralela, lo que da lugar a un planteamiento ligeramente modificado de la operación convolución descrita anteriormente. Hasta ahora, dicha operación se definía acorde a proporcionar una única característica a partir de diversos canales de entrada relativos a los datos de entrada. Con esta modificación, es necesario que el kernel sea un tensor de 4 dimensiones, haciendo una referencia al mapa de características que se va a obtener.

Con esta modificación del kernel se propone una nueva definición para la obtención de la salida Z para un determinado canal, de modo que el elemento $Z_{i,j,k}$ haga referencia al elemento *(j,k)* del mapa de características i:

$$Z_{i,j,k} = \sum_{l,m,m} V_{l,j+m-1,+n-1} K_{i,l,m,n} \tag{3.10}$$

donde:

- V se corresponde con los datos de entrada, siendo $V_{i,j,k}$ el elemento *(j,k)* del canal i de los datos de entrada.
- K se corresponde con el kernel (4-D tensor), donde el elemento $K_{i,j,k,l}$ hace referencia al elemento *(k,l* relativo al canal de los datos de entrada j y relativo al canal de salida j.

De este modo, se puede observar cómo dicha operación se encarga de sumar las resultados obtenidos, tras la aplicación de la operación convolución sobre cada uno de los canales relativos a la entrada del bloque convolucional.

No obstante, existen diferentes variantes de esta operación que se suelen utilizar en la determinación de la estructura de la red convolucional. La mayoría de ellas se sustentan en dos conceptos principalmente:

- *Strides*: Hasta ahora, se ha expuesto cómo en la aplicación de la operación convolución, el kernel se desplazaba (de izquierda a derecha y de arriba a abajo) a través de los diferentes canales de la imagen para llevar a cabo la multiplicación entre los elementos, realizándose estos desplazamientos de columna a columna y de fila en fila. Bajo este contexto, surge el concepto de *stride*, definiéndose como la cantidad

de desplazamiento entre las sucesivas aplicaciones del kernel a la imagen. Así pues, bajo la especificación proporcionada hasta el momento de la operación convolución, el valor por defecto del *stride* es (1,1). No obstante, si, por ejemplo, el valor del *stride* fuera (2,2), indicaría que por cada movimiento del kernel en la dirección horizontal, se debería desplazar este kernel 2 pixels a la derecha y, por cada movimiento del kernel en la dirección vertical, 2 pixels hacia abajo. Matemáticamente, se expresa la inclusión de este concepto en la operación convolución, siendo denotado como s:

$$Z_{i,j,k} = c(K, V, s)_{i,j,k} = \sum_{l,m,m} V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \tag{3.11}$$

Tras la definición de este nuevo concepto, se puede deducir que, además de influir en la aplicación de la operación convolución, se puede emplear de cara a reducir la dimensión del mapa de características resultante. Esta reducción en el número de operaciones lleva asociada una reducción en el coste computacional. No obstante, no es la opción más recomendable, si el objetivo es reducir la dimensión del mapa de características. En la parte superior de la Figura 3.11 [10] se puede visualizar una representación gráfica de la operación convolución en una dimensión con un valor de *stride* igual a dos. Este nuevo concepto es equivalente a aplicar la operación convolución propiamente dicha y realizar, posteriormente, una reducción de la dimensión seleccionando unas determinadas salidas, tal y como se puede observar en la parte inferior de la Figura 3.11.

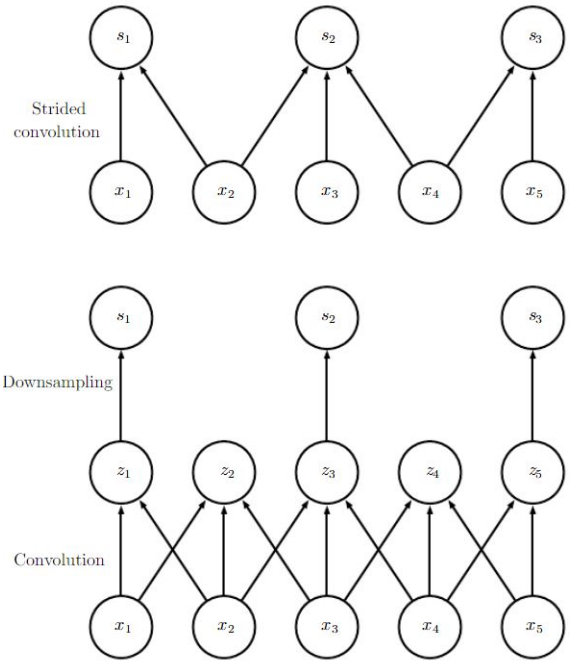


Figura 3.11: Operación convolucional con un valor de *stride* igual a 2.

- *Padding*: Evaluando los resultados proporcionados por la aplicación de la operación convolución sobre sus respectivos datos de entrada, se puede percatar cómo la dimensión del mapa de características resultante es menor que la del mapa de características de entrada. Esta reducción de la dimensionalidad se denomina *efecto de bordes*, ya que esta disminución es causa de la interacción del kernel con el borde de los datos de entrada. En particular, esta reducción de la dimensionalidad del mapa de características resultante, se corresponde con un aminoramiento de la dimensionalidad del kernel menos 1 sobre la dimensionalidad del mapa de características de entrada. Así pues, no se puede obviar esta reducción, ya que en caso de que la red neuronal convolucional sea muy profunda, esta disminución podría provocar que, en un determinado punto, no se disponga de datos de entrada. Con el objetivo de solventar esta problemática, surge el *padding*, el cual consiste en añadir el número necesario de filas y columnas en cada lado del mapa de características de entrada, para que se ajuste una ventana de convolución en torno a cada una de las celdas del mapa de características. Con esta medida, se consigue que la dimensión del mapa de características de salida sea igual a la del mapa de características de entrada. De cara a la implementación, esta acción se traduce en añadir filas y columnas de 0's, haciendo el conjunto de datos de entrada más ancho.

  Acorde a este nuevo concepto, se distinguen tres casos posibles:

  - *valid convolution*: Hace referencia al caso en el que no se aplica *padding* (caso expuesto hasta el momento), reduciéndose la dimensionalidad de los mapas de características en cada capa.
  - *same convolution*: Se añaden tantas filas y columnas de 0's como sea necesario, para que el tamaño de la salida sea igual al tamaño de la entrada. No obstante, esta medida correctiva acarrea que los pixeles cercanos a los bordes tengan menos influencia que los céntricos.
  - *full convolution*: Esta alternativa surge para solventar el problema de que unos píxeles tengan mayor influencia que otros. Así pues, se opta por añadir tantas filas y columnas de 0's como sean necesarias, para que cada pixel relativo al mapa de características de entrada sea visitado el mismo número de veces.

No obstante, estos conceptos no constituyen las únicas variantes posibles de la operación convolución. En la práctica, se puede encontrar, entre otras:

- *unshared convolution*: Esta variante relaja las condiciones de dicha operación. En particular, mantiene la conectividad local de las capas, pero cada conexión tiene su propio peso asociado, es decir, no se comparten los parámetros. En la Figura 3.12 [10] se puede visualizar esta variante, pudiendo observar cómo cada conexión lleva asociada un peso diferente.

- *tiled convolution*: Esta operación surge como una combinación de la convolución propiamente dicha y la variante *unshared convolution*. Así pues, se dispone de varios kernels que se aplican alternativamente. En la Figura 3.13 [10] se ilustra su

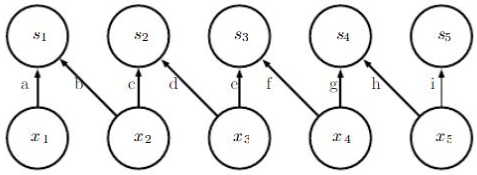


Figura 3.12: Variante de la operación convolución - *unshared convolution*.

funcionamiento. En este caso, se dispone de dos kernels: el primero compuesto por los pesos a y b y, el segundo, por los pesos c y d.

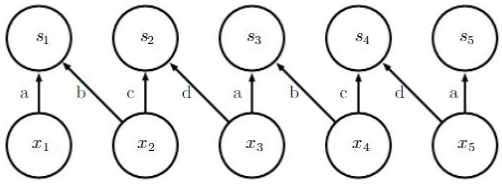


Figura 3.13: Variante de la operación convolución - *tiled convolution*.

3.2.6. Entrenamiento de la Red Neural Convolucional

El entrenamiento de una red neuronal convolucional radica en aprender el valor de los pesos de la red en cuestión, siendo necesario un algoritmo que nos permita ajustar estos valores. No obstante, el algoritmo empleado no difiere del comúnmente utilizado en el aprendizaje supervisado de otras redes neuronales, como por ejemplo, el método de *back-propagation* o *retroprogración del error*. Análogamente, se puede diferenciar claramente dos fases:

- Propagación hacia delante: Tiene como objetivo generar la salida predicha Z por la red para una determinada muestra. Para ello, se propaga la salida de una capa a la siguiente [11] hasta generar la salida. A continuación, se calcula la función de coste *J(V,K)*, es decir, se computa el error cuadrático medio para esa muestra a partir de la salida obtenida y deseada.

- Propagación hacia atrás: Aquí es donde se produce el aprendizaje, es decir, la actualización de los pesos. Para ello, se hace uso del método del gradiente. [10] Debido a que la dependencia funcional con los pesos no es directa, se aplica la regla de la cadena. Con este planteamiento, se sigue un desarrollo basado fundamentalmente en los mismos pasos que los del algoritmo de retropropagación del error del perceptrón multicapa.

Capítulo 4

Construcción de una red neuronal convolucional

En el capítulo anterior, se han presentado los diferentes conceptos matemáticos subyacentes a las redes neuronales convolucionales. Aquí, se aborda la puesta en práctica de la creación de esta tipología de redes, mediante la utilización de la biblioteca *keras*. Asimismo, se expondrán las diferentes metodologías de aprendizaje que facilita la biblioteca y su utilización para llevar a cabo la creación de una red neuronal.

4.1. Python

Hoy en día, existe una gran cantidad de lenguajes de programación, algunos de propósito general y otros orientados a un determinado fin. Aun dentro de los primeros, no cabe duda que suelen estar especialmente indicados para ciertos ámbitos específicos. Esto es lo que sucede con *python* y su uso en análisis de datos. No es el único lenguaje en este campo. Es frecuente encontrar trabajos desarrollados en *R*, pero su propósito es más específico que *python*. Desde el punto de vista de la programación, ofrece una ventaja básica frente a R; es multiparadigma, lo que permite explotar todos los avances introducidos por la POO, que en R no es posible.

En cuanto a la creación del código con implementación multinúcleo y multihilo de *python* se alcanzan velocidades sensiblemente superiores a *R* en general.

4.2. Keras

Keras es una API de alto nivel de código abierto escrita en *python* para construir modelos de redes neuronales. La razón principal por la cual se decide utilizar esta biblioteca es que es una de las API's líderes en la construcción de redes neuronales. Se caracteriza por su facilidad de uso, ya que, cómo propiamente dicen los creadores de la API, fue diseñada para que el ser humano reduzca, en la medida de lo posible, la carga cognitiva. Para ello, esta API proporciona una construcción de modelos de redes neuronales a partir

de bloques de alto nivel, es decir, proporciona módulos independientes cómo pueden ser diferentes tipos de capas neuronales (convolucionales, densamente conectadas,...), optimizadores, funciones de activación, etc, que se combinan para crear el modelo deseado. [18]

Keras no se encarga de las operaciones de bajo nivel, sino que requiere utilizar un motor *back-end*, permitiendo que se puedan conectar diferentes de estos motores a la API, es decir, no fijando el uso de un determinado marco de manipulación de tensores, sino proporcionando un amplio grado de integración.

De entre los posibles motores *back-end* con los que se puede integrar esta API, el que se utiliza principalmente (proporcionado por defecto) es *tensorflow*, siendo esta una librería ampliamente utilizada de aprendizaje automático desarrollada por Google. Debido al alto grado de utilización de la API *keras* sobre esta librería, ella misma proporciona un módulo relativo a ella, *tf.keras*.

4.3. Creación de un modelo en Keras

Para llevar a cabo la construcción de una red neuronal utilizando esta API, se hace uso de la estructura de datos *model* proporcionada por *keras*, como si fuera una secuencia de capas. En particular, *keras* proporciona dos modelos:

- El modelo *Sequential*
- La clase *model* utilizada a partir de la *API* funcional

Entre estas dos posibilidades, el modelo *Sequential* es el más simple y común, consistiendo en un ensamblaje lineal de capas. Para construir redes neuronales más complejas (múltiples entradas/salidas, capas compartidas, reutilización de determinadas capas, etc) es necesario utilizar la clase *model*. [12]

En este TFG, debido a la topología propia de las redes neuronales convolucionales (sucesión lineal de capas) se procede a utilizar el modelo *Sequential*. Con ello, se asume que, tanto la entrada como la salida del modelo, son únicas, lo que puede suponer una restricción muy fuerte. Si fuera el caso, con la utilización de la clase *Model*, se podría abarcar múltiples entradas.

4.3.1. Modelo *Sequential*

Un modelo *sequential* es una secuencia de capas, creándose capa a capa, es decir, agregando una a una las capas de la topología deseada. No obstante, en primer lugar, es necesario crear un objeto de tipo *Sequential*, sobre el que ir agregando las capas (*model = Sequential()*). A continuación, se usa el método *add()*, para ir añadiendo progresivamente las capas de la red convolucional.

En cuanto a las capas, *keras* proporciona una variedad de tipos predefinidos, a la par que posibilita la creación de capas personalizadas. Hay que tener en cuenta, que este modelo requiere especificar el tamaño de los datos de entrada en la primera capa (y sólo en ella) de la red neuronal, indicándose mediante el argumento *input_ shape*. En la construcción de modelos relativos a imágenes, este argumento debe indicar tanto la altura y anchura de las imágenes como el número de canales que la constituyen (3 en el caso de que el sistema de codificación sea RGB) en formato tupla. En el campo relativo a las redes neuronales convoluciones se utilizan las siguientes:

- *Conv2D*: Hace referencia a una capa de convolución sobre un espacio bidimensional. Entre sus argumentos de configuración resultan de especial interés:
  - *filters*: Permite especificar el número de mapas de características (filtros) que se desean obtener, es decir, especifica la dimensionalidad de la salida de esta capa, siendo ésta obligada. Habitualmente, en la práctica, el valor de este argumento se irá aumentando a medida que las capas sean más profundas, aprendiendo las primeras capas menos filtros y, las últimas, más.
  - *kernel_ size*: Hace referencia al tamaño de los kernels que se emplearán a la hora de realizar la operación convolución, siendo también obligatorio. Su valor se especifica a través de una tupla, en particular, indicando tanto el ancho como el alto del kernel (debiendo ser enteros). En la práctica, se recomienda que, estas dimensiones sean impares y no muy grandes.
  - *strides*: Se refiere al concepto presentado en el apartado 2.2.5, siendo su valor por defecto (1,1), de tal manera, las sucesivas aplicaciones del kernel al mapa de características de entrada se realizan por cada desplazamiento de 1 pixel, tanto en la dirección horizontal como vertical.
  - *padding*: Hace referencia a la posibilidad de añadir el número necesario de filas y columnas, en los mapas de características de entrada, de modo que sea posible ajustar la operación convolución en torno a cada celda. El valor por defecto de este argumento es *valid*, no aplicándose *padding*. Mediante la opción *same*, el *padding* se aplica.
  - *activation*: Permite indicar la función de activación, que se desea aplicar a la salida proporcionada por la operación convolución. Por defecto, no se aplica ninguna función de activación. Otra opción plausible, sería la especificación de dicha función a través de una capa *Activation*. En este TFG, se opta por indicar dicha función a través de este parámetro, con la finalidad de manejar un código más compacto, que sea más fácilmente interpretable. *Keras* posibilita la utilización de una amplia variedad de funciones de activación:
    - *softmax*
    - *softplus*
    - *tanh*
    - *elu*
    - *softsign*
    - *sigmoid*
    - *selu*
    - *relu*
    - *hard_ sigmoid*

- *exponential*
- *linear*
- *PReLU*
- *LeakyReLU*

La elección de la función de activación depende del problema. Así pues, en redes convolucionales (visión artificial), la función de activación más comúnmente utilizada, de cara a su aplicación sobre la salida de la operación convolución, es *ReLU* (Rectified Linear Unit):

$$f(x) = max(0, x) \tag{4.1}$$

En la Figura 4.1 [20] se puede visualizar esta función.

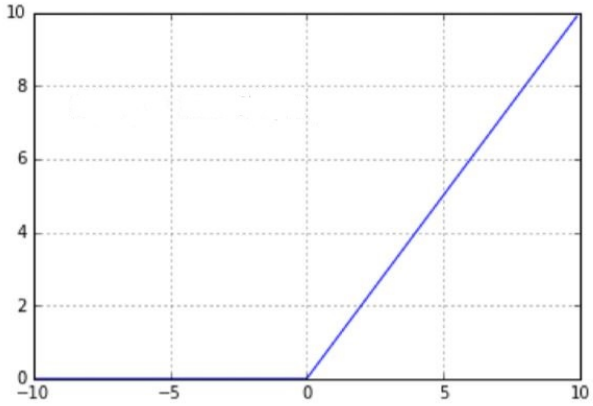


Figura 4.1: Representación gráfica de la función ReLU.

Ofrece una implementación sencilla, a la vez que proporciona una convergencia acelerada del descenso del gradiente. Sin embargo, la conversión de todos los valores negativos a cero puede disminuir su capacidad de ajuste.

– *kernel_ regularizer, bias_ regularizer, y activity_ regularizer*: Estos argumentos permiten especificar el tipo de regularización, que se desea aplicar a la capa convolucional. Por defecto, no se aplica ninguna regularización.

- *MaxPooling2D*: Hace referencia a una capa de agrupamiento, en particular, a la función *max pooling*. Esta no es la única capa de reducción de la dimensionalidad proporcionada por *keras*, sino que también se podría utilizar la función *average pooling* mediante una capa *AveragePooling2D*. Entre los argumentos de configuración de esta capa, el más relevante es *pool_ size*, el cual permite indicar el tamaño de la ventana sobre la que se aplicará la función de agrupamiento. Esto se especifica en formato tupla, indicando tanto el número de pixels horizontales como verticales, que abarcará la ventana, aunque también se podría indicar un único número, de forma que se sobreentienda, que el ancho y alto de la ventana es el mismo.

- *Flatten*: Este tipo de capa, como el propio nombre indica, se encarga de aplanar los datos proporcionados: los transforma a un vector de una sola dimensión. Si por ejemplo, las dimensiones del mapa de características de entrada de dicha capa fueran (3, 3, 64), la salida proporcionada sería un vector de dimensión (576,).

- *Dense*: Hace referencia a capas completamente conectadas, donde cada neurona de esta capa recibe información de todas las neuronas de la capa anterior. Entre los argumentos de configuración resultan de interés:

  – *units*: Permite especificar la dimensionalidad de la salida de la capa, siendo el argumento más relevante. En el ámbito de la clasificación de imágenes, tal y como se comentó en el capítulo anterior, la última capa debe ser de esta tipología, correspondiéndose esta dimensionalidad al número de categorías posibles en la clasificación. En el caso particular, de que la clasificación sea binaria, resulta suficiente que este valor sea igual a 1.

  – *activation*: Igual que en la capa *Conv2D*, permite especificar la función de activación que se desea aplicar a la salida proporcionada. En caso de no seleccionarse ninguna, la función por defecto es la lineal, no sufriendo modificación alguna. Tal y como se comentó, la elección de esta función depende del propio problema. En la clasificación de imágenes, son dos las funciones de activación que se suelen emplear habitualmente:

    ◦ *sigmoid*: Se recomienda la utilización de esta función, tanto en los problemas de clasificación binaria, como en los de clasificación multietiqueta. Esta función es:

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{4.2}$$

En la Figura 4.2 [20] se puede observar una representación gráfica de esta función. En ella, se puede visualizar cómo sus valores están comprendidos entre 0 y 1, siendo especialmente útil cuando se desean predecir probabilidades.

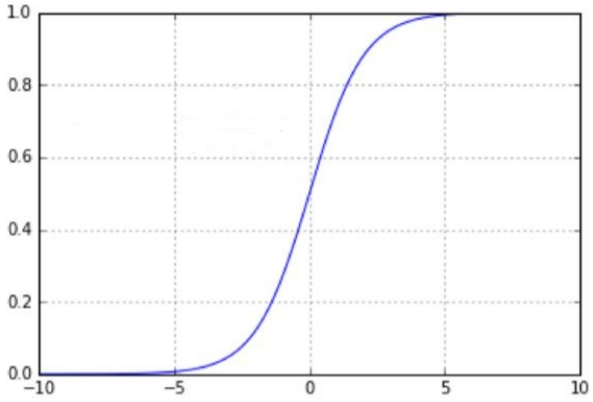


Figura 4.2: Representación gráfica de la función sigmoide.

- *softmax*: Se recomienda la utilización de esta función en los problemas de multiclasificación (siendo su etiqueta única). Se puede considerar una generalización de la función sigmoide, proporcionando la probabilidad de pertenencia a cada una de las categorías candidatas. Así pues, esta función se expresa como:

$$f(x_i) = \frac{\exp(x_i)}{\sum_{c=0}^{k} \exp(x_c)} \tag{4.3}$$

  En la Figura 4.3 se puede visualizar una representación gráfica del funcionamiento de esta función de activación.

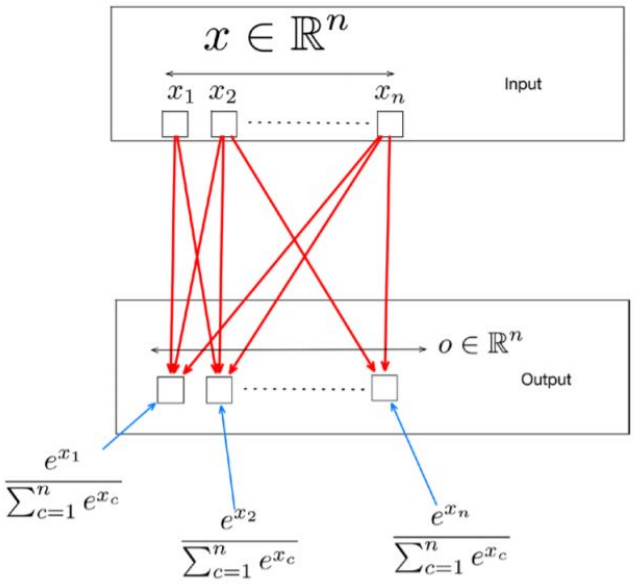


Figura 4.3: Representación gráfica de la función softmax.

- *Dropout*: Hace referencia a una técnica de regularización que se basa en imponer, de forma aleatoria, el valor cero, a un porcentaje especificado de las unidades de entrada (sería como desconectar esas neuronas). La utilización de estas capas tiene como motivación la generalización de la red y, a su vez, evitar caer en el sobreajuste. El parámetro de mayor interés, en esta tipología de capa, es *rate*, el cual indica el porcentaje de unidades que se deben establecer a 0, teniendo que estar este valor comprendido en el rango (0,1).

Una vez definida la estructura de la red neuronal deseada, se lleva a cabo el entrenamiento del modelo. Para ello, en primer lugar, se debe especificar la configuración del proceso de aprendizaje mediante el método *compile*. Entre sus argumentos de configuración resultan de especial interés:

- *loss*: Permite especificar la función de pérdida que se desea minimizar durante el aprendizaje del modelo, actualizándose los pesos acorde a la evaluación proporcionada por esta función. La elección de esta función, debe sustentarse en la tipología

del problema, es decir, si se trata de regresión o clasificación. En el caso de que la clasificación sea binaria, se recomienda optar por la función *binary_ crossentropy*, siendo necesario, para su utilización, que la capa de salida de la red especifique el uso de la función *sigmoid* como función de activación, y que la dimensionalidad de la salida sea 1. De este modo, el valor de esta función aumenta basándose en la diferencia entre la probabilidad predicha y la categoría real. La entropía cruzada se calcula como:

$$H(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \tag{4.4}$$

En el caso de que la clasificación fuese multiclase, se recomienda utilizar una generalización de esta entropía cruzada binaria, denominándose en *keras* esta función *categorical_ crossentropy*. La formulación asociada a esta generalización es:

$$H(y, \hat{y}) = \sum_{j=1}^{C} -(y_{i,j} \log(\hat{y_{i,j}})) \tag{4.5}$$

Se calcula, para cada observación, la diferencia promedio entre la probabilidad predicha y la probabilidad real (el valor asociado a la categoría real toma valor 1, el resto 0) para todas las clases.

- *optimizer*: Permite indicar el algoritmo de optimización que se desea utilizar a la hora de actualizar los pesos, dependiendo, en cierta medida, de esta elección la convergencia del modelo. Entre los diversos algoritmos disponibles, el algoritmo del descenso de gradiente es el más conocido y simple; no obstante, el aprendizaje con este algoritmo puede ser en algunos casos muy lento. Por ello, en el presente TFG, entre la variedad de algoritmos proporcionados por *keras*, se baraja la utilización de dos métodos adaptativos:

  - *RMSProp (Root Mean Square Propogation)*: Se asemeja al conocido descenso de gradiente, pero proporcionando una convergencia más rápida, ajustando automáticamente la tasa de aprendizaje con el objetivo de evitar oscilaciones.
  - *Adam (Adaptive Moment)*: Intenta combinar las ventajas del algoritmo anterior, junto con las proporcionadas por el algoritmo del descenso del gradiente con impulso (el cual tiene en cuenta los gradientes calculados en pasos anteriores, para determinar de forma más precisa la dirección de la optimización). [19]

- *metrics*: Permite indicar medidas que se consideran de interés para evaluar el rendimiento del modelo; no obstante, los resultados de estas medidas no se consideran en el aprendizaje. En este TFG, de cara a evaluar los modelos, se opta por utilizar la precisión (*accuracy*), siendo esta la medida más utilizada, haciendo referencia al porcentaje de aciertos del modelo en cuestión: se suma la totalidad de los aciertos y se divide entre el número total de los ejemplos.

Una vez configurado el proceso de aprendizaje, se lleva a cabo el ajuste del modelo a los datos. En función de la tipología de los datos de entrada, es necesario utilizar una de estas dos funciones:

- *fit*: Se utiliza cuando el formato de los datos de entrada son matrices *Numpy* y, cuando los conjuntos de datos son pequeños, ya que con este método, el conjunto de entrenamiento se almacena en memoria. Para utilizar esta función en la tarea relativa a la clasificación de imágenes, es necesario convertir las imágenes a su representación numérica. Entre sus argumentos de configuración, resultan de interés:
  - x e y: A partir de estos argumentos, se especifican las matrices relativas a las imágenes y sus etiquetas asociadas respectivamente en formato *Numpy*; correspondiéndose al conjunto de datos de entrenamiento sobre el que se desea ajustar el modelo.
  - *epochs*: Hace referencia al número de épocas en el entrenamiento del modelo, siendo una época una iteración que recorre todos los elementos del conjunto de entrenamiento.
  - *batch_ size*: Cuando se entrena el modelo, con el objetivo de acelerar este proceso, es habitual dividir el conjunto de entrenamiento en lotes, de forma que el modelo se entrena para cada uno de los subconjuntos de muestras, ajustando los pesos del modelo después de la propagación por la red de las ejemplos correspondientes a un lote. Este argumento permite especificar el tamaño de los lotes.
  - *validation_ data*: Permite especificar un conjunto test, a partir del cual se pretende evaluar el modelo, proporcionando una aproximación más real de las prestaciones del sistema.
- *fit_ generator*: Se utiliza cuando los datos de entrada se obtienen a partir de un generador, formándose automáticamente los lotes. La necesidad de utilizar esta opción, radica en la tenencia de un conjunto de datos demasiado grande como para almacenar en memoria. O al contrario, debido a que el conjunto de datos es muy pequeño, necesitando realizar un aumento de los datos disponibles a partir de este generador (con el objetivo de lograr una mayor generalización del modelo). En este caso, los argumentos de configuración de interés son:
  - *generator*: Especifica el identificador del generador que se encarga de proporcionar de forma indefinida lotes.
  - *steps_ per_ epoch*: Especifica el número de muestras que se deben extraer del generador, antes de declarar una época finalizada (ya que como se comentó, el generador proporcionará muestras de forma indefinida).
  - *epochs*: Hace referencia al número de épocas en el entrenamiento del modelo.
  - *validation_ data*: Especifica el generador relativo a un conjunto de datos de validación, de cara a obtener una evaluación del modelo durante su entrenamiento.

– *validation_ steps*: Especifica el número de lotes que debe generar el generador, para obtener la medida de evaluación tras completar cada época de entrenamiento.

Una vez entrenado el modelo, se pueden obtener las predicciones de nuevos datos a partir de la función *predict*. También resulta de interés el uso de las funciones *evaluate* y *evaluate_ generator*, las cuales devuelven el valor de la función de pérdida y el valor de las medidas especificadas, para un conjunto test especificado, evaluando el rendimiento del modelo creado. La utilización, de una función u otra, depende del formato de los datos de entrada (igual que en las funciones relativas al ajuste del modelo).

4.3.2. Clase *model* de la *API* funcional

En la subsección anterior, se ha podido observar cómo el modelo *Sequential* proporciona una metodología sencilla para modelar la estructura de la red neuronal. Esta topología de red es restringida, debiendo consistir en una pila de capas. Así pues, para crear estructuras de redes neuronales más complejas, se requiere hacer uso de la clase *model* proporcionada por la API. En el desarrollo de este TFG, se considera de interés explorar esta alternativa para abordar la problemática relativa a entradas múltiples.

Esta API utiliza, de la misma manera, las capas descritas en el apartado anterior, relativo al modelo *Sequential*, pero proporcionando una mayor flexibilidad en su ensamblaje. En lugar de crear exclusivamente secuencias de capas, esta clase permite crear grafos de capas.

Más concretamente, esta API funcional trabaja directamente con tensores, actuando las capas como funciones que toman como argumentos tensores y que devuelven otros tensores. Bajo este enfoque, se definen, en primera instancia las capas y, posteriormente se crea el modelo a partir del método *Model*, para el cual es necesario especificar sus entradas y salidas, a partir de los argumentos *inputs* y *outputs*. Tal y como se comentó, una de las ventajas de esta API funcional reside en la posibilidad de manejar entradas y salidas múltiples, especificando para ello simplemente una lista con las diferentes entradas o salidas en los argumentos. Esto conlleva, que se deba definir una capa *Input* para cada una de las entradas del modelo, constituyendo cada entrada un tensor.

Bajo estas especificaciones, este modelo incluirá todas las capas necesarias para obtener las salidas dadas las entradas; recuperando cada capa involucrada en la obtención de la salida a partir de la entrada, creando así el modelo subyacente (la estructura de red neuronal). La recuperación de esta estructura se obtiene acorde a la especificación de las capas, ya que actúan como funciones, que van transformando la entrada hasta obtener la salida final. Por ello, en cada capa, es necesario especificar su entrada (entre paréntesis al final de la definición de la capa), siendo el resultado de la aplicación de dicha capa, la salida de la misma.

Una vez creada la estructura de la red neuronal, es necesario compilar (configurar el proceso de aprendizaje) y entrenar el modelo; utilizándose las mismas funciones que en el modelo *Sequential*, descritas en el apartado anterior. En cuanto a la predicción y evaluación del modelo, también se llevan a cabo del mismo modo que en el modelo *Sequential*, utilizando las mismas funciones. Se puede observar cómo el proceso de entrenamiento es el mismo, lo único que cambia es la especificación del modelo, con el objetivo de abarcar modelos más complejos.

Modelo de entrada múltiple

Tras exponer de forma general el funcionamiento de la API funcional de *keras* para la creación de modelos más complejos, se aborda a continuación, de forma algo más detenida, el supuesto relativo a la entrada múltiple; ya que se explora esta alternativa en este TFG.

El groso de este supuesto reside en la especificación de las diversas fuentes de datos de entrada (las cuales se especifican en la creación del modelo mediante una lista) y. la combinación de estas entradas en algún punto de la estructura de la red neuronal.

Para llevar a cabo esta combinación de diferentes ramas, se suele utilizar una capa que sea capaz de combinar tensores; siendo las más conocidas:

- *add*: Esta capa toma como argumento una lista de tensores de la misma dimensionalidad y devuelve su suma elemento a elemento.

- *concatenate*: Esta capa toma como argumento una lista de tensores, los cuales deben ser de la misma dimensión excepto en el eje en que se va a concatenar (siendo necesaria su especificación) y, devuelve su concatenación.

4.3.3. Estrategias proporcionadas por *keras* a la hora de entrenar una red neuronal

Hasta ahora, en este capítulo se ha contemplado la creación de la estructura de la red neuronal desde cero, especificando capa a capa la topología de la red y, entrenándola posteriormente, con el objetivo de ajustar sus pesos a partir del conjunto de datos de entrenamiento. No obstante, si el conjunto de datos es muy pequeño, es probable que la red neuronal sea propensa al sobreajuste, no logrando que la red extraiga todas las características subyacentes a las imágenes que serían determinantes en su clasificación.

Una solución, ampliamente utilizada en el campo del aprendizaje profundo, radica en la utilización de una red previamente entrenada, estrategia conocida como *transfer learning*. Estas redes pre-entrenadas han sido entrenadas a partir de conjuntos de datos muy grandes, pudiendo actuar como un modelo genérico del mundo visual en general. La propia librería *keras* proporciona una serie de modelos entrenados para la clasificación de imágenes.

Dentro de esta estrategia, se pueden diferenciar dos enfoques a la hora de utilizar el modelo pre-entrenado [13]:

- *Extracción de características*: Se eliminan las capas totalmente conectadas, manteniendo el resto de la red (capas de convolución y agrupación), denominándose base convolucional. Este bloque se utiliza para extraer características generales; y a continuación, se añade un nuevo clasificador, el cual se entrena desde cero en base a los datos disponibles. Cabe recalcar, cómo los pesos relativos a la base convolucional se congelan, no siendo modificados en el entrenamiento. Si la nueva tarea difiere mucho de la tarea relativa al modelo pre-entrenado, se recomienda usar solo los primeras capas del modelo; ya que como se vio en capítulos anteriores, las primeras capas extraen características generales y cuanto más profundas son, extraen características más específicas.

- *Fine tuning*: Es complementario a la extracción de características, de forma que se descongelan las capas más profundas de la base convolucional, para entrenarlas conjuntamente con el clasificador añadido a esta base. Mediante esta técnica, se pretende reajustar las características más específicas del modelo, con el objetivo de que sean más relevantes en la nueva tarea de clasificación.

En la Figura 4.4 [23] se adjunta una representación gráfica de estas alternativas, mostrándose, de izquierda a derecha, un modelo pre-entrenado, uno obtenido mediante la utilización de extracción de características y, por último, otro aplicando *fine-tuning*.

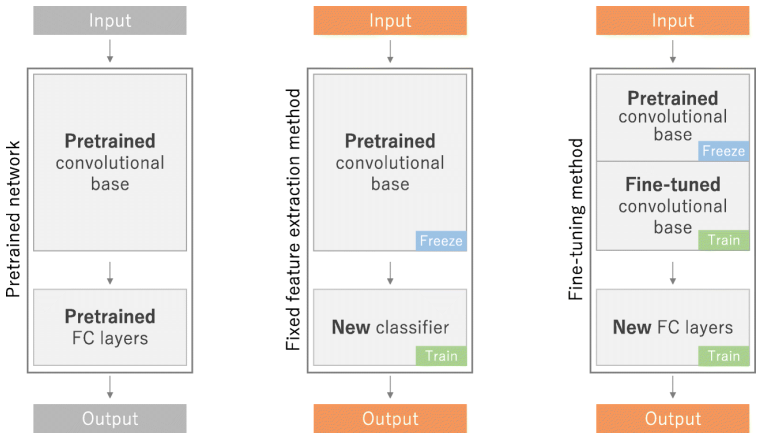


Figura 4.4: Alternativas relativas a *transfer learning*.

Actualmente, esta estrategia está siendo altamente usada a la hora de entrenar un modelo. A pesar de ser muy útil, hay que evaluar si esta técnica es correctamente aplicable a la tarea en cuestión. En esta línea, es de vital importancia, evaluar si el dominio de nuestro conjunto de datos es similar al de los datos con el que la red fue pre-entrenada. En el caso

de que estos conjuntos no sean similares, no debería aplicarse *transfer learning*, a pesar de que el conjunto de datos del que se disponga sea pequeño, ya que las características aprendidas, no serían válidas para el problema en cuestión.

Capítulo 5

Descripción y preprocesamiento de los datos

Tras efectuar un estudio de las redes neuronales convolucionales en los dos capítulos anteriores, se proporciona, en este capítulo, una descripción de los datos, sobre los que se aplican estos conocimientos adquiridos. A su vez, se aborda su preprocesamiento, haciendo hincapié en su importancia en cualquier proyecto de Minería de Datos, para obtener así unos buenos resultados en la fase de modelado.

5.1. Descripción de los datos

En el presente TFG, se pretende abordar la aplicación de redes neuronales convolucionales a un problema de clasificación de imágenes, siendo de interés personal que dicho problema sea de aplicación real en este caso, perteneciendo al ámbito de la medicina. Satisfaciendo este interés, se lleva a cabo este estudio en el campo de la Oftalmología. Para ello, se elaboran una serie de modelos, que permitan predecir diferentes enfermedades en base a imágenes oculares.

En particular, se opta por trabajar con dos conjuntos de imágenes distintos; el primero de ellos, relativo a imágenes de fondo de ojo y el segundo, a imágenes de superficie ocular. La decisión de utilizar dos conjuntos de imágenes se fundamenta en las descripciones de los mismos. A continuación, se presenta una descripción más detallada de ambos conjuntos.

5.1.1. Conjunto de datos 1: Imágenes de fondo de ojo

El primer conjunto de datos que se utiliza en este TFG, y sobre el cual se sustenta la mayor parte del estudio de las redes convolucionales, hace referencia a una base de datos orientada a la investigación de la detección de la retinopatía diabética, obteniéndose a partir de una competición promovida por el *IEEE International Symposium on Biomedical Imaging*. En particular, la denominación de este conjunto de datos es *Indian Diabetic*

Retinophaty Image Dataset (IDRID), estando conformado por un conjunto de imágenes de fondo de ojo. Según la propia descripción de este conjunto de datos, las imágenes fueron tomadas en una clínica oftalmológica ubicada en la India, por un especialista en retina, estando todas centradas en torno a la mácula. Este *dataset* está conformado por 516 imágenes de fondo de ojo, tamaño apropiado para un estudio de este tipo.

Este *dataset* consiste en:

- Un conjunto de imágenes de fondo de ojo (516 imágenes), las cuales se proporcionan distribuidas en un conjunto de entrenamiento (413 imágenes) y, en un conjunto test (103 imágenes), contando estas imágenes con una alta resolución (4288 $\times$ 2848).

- Las etiquetas, correspondientes a estas imágenes, se proporcionan en dos archivos en formato *.csv*, uno relativo al conjunto de entrenamiento y otro, al conjunto test. En ellos, se proporciona el grado de la enfermedad del edema macular diabético y de la retinopatía diabética correspondiente a cada imagen. Cada imagen se identifica con el nombre de la propia imagen, actuando como identificador en los archivos *.csv*. Para la retinopatía diabética, los valores de las etiquetas están comprendidos en el rango [0,4]. El valor 0 hace referencia a que no se padece esa enfermedad, y el resto de valores indican el grado de gravedad de dicha enfermedad. Por otra parte, para el edema macular diabético, los valores de las etiquetas están comprendidos en el rango [0,2]. Del mismo modo, el 0 hace referencia al no diagnóstico de esta enfermedad.

En las tablas 5.1, 5.2 y 5.3, se adjuntan las frecuencias (número de imágenes) asociadas a cada grado de la enfermedad en el conjunto de entrenamiento, en el conjunto test y en la totalidad de los datos para la enfermedad edema macular diabético, es decir, se proporciona la descripción de la variable *Risk of macular edema*.

Conjunto de entrenamiento			
Grado	0	1	2
Frecuencia	177	41	195

Cuadro 5.1: Descripción de la variable *Risk of macular edema* en el conjunto de entrenamiento.

Conjunto test			
Grado	0	1	2
Frecuencia	45	10	48

Cuadro 5.2: Descripción de la variable *Risk of macular edema* en el conjunto test.

Conjunto total			
Grado	0	1	2
Frecuencia	222	51	243

Cuadro 5.3: Descripción de la variable *Risk of macular edema* en la totalidad de los datos.

Se puede observar, cómo los datos no están balanceados, teniendo estos conjuntos de datos una menor representación del grado 1, no constituyendo ni el 5 % de la totalidad de las imágenes.

Del mismo, en las tablas 5.4, 5.5 y 5.6, se adjuntan las frecuencias relativas a la retinopatía diabética. En vista a estas descripciones de la variable, se puede observar cómo las diferentes clases tampoco están equilibradas.

Conjunto de entrenamiento					
Grado	0	1	2	3	4
Frecuencia	134	20	136	74	49

Cuadro 5.4: Descripción de la variable *Retinopathy grade* en el conjunto de entrenamiento.

Conjunto test					
Grado	0	1	2	3	4
Frecuencia	34	5	32	19	13

Cuadro 5.5: Descripción de la variable *Retinopathy grade* en el conjunto test.

Conjunto total					
Grado	0	1	2	3	4
Frecuencia	168	25	168	93	62

Cuadro 5.6: Descripción de la variable *Retinopathy grade* en la totalidad de los datos.

5.1.2. Conjunto de datos 2: Imágenes oculares

El estudio de este segundo conjunto de datos se efectúa de forma complementaria, justificando el mismo en evaluar la potencia de las redes neuronales convolucionales, incluso cuando el conjunto de datos proporcionado sea muy pequeño.

Este segundo conjunto de datos hace referencia a una pequeña base de imágenes oculares (40 imágenes), siendo proporcionada por el IOBA (Instituto Universitario de Oftalmobiología Aplicada). Para esta tipología de imágenes, las tareas relativas a su clasificación radican en detectar la presencia de neovasos y el reconocer la conjuntivalización. Este conjunto está conformado por:

- Un conjunto de imágenes oculares constituido por 40 imágenes, siendo por ello un conjunto de datos muy pequeño, tanto en su formato original como en formato normalizado.

- Un archivo *.xlsx* que proporciona las etiquetas correspondientes a las imágenes, para las dos enfermedades a tratar. Sin embargo, estas etiquetas hacen referencia a diferentes estadíos de ambas enfermedades. En el caso de la conjuntivalización, las imágenes se etiquetan en tres posibles estados (II, IIC y III). En la tabla 5.7 se puede observar la distribución de esta variable, pudiendo detectar cómo, debido al escaso número de muestras disponibles, el estadío II es muy poco representativo, teniendo en cuenta que, este conjunto de imágenes se deberá particionar en un conjunto de entrenamiento y un conjunto test. Por otra parte, en el caso de la neovasculización, el número de posibles etiquetas es mucho mayor (I, IA, IB, IC, IIA, IIB, IIC y III). En la tabla 5.8, se muestra la frecuencia de cada uno de estos estadíos, pudiendo observar cómo este caso es más extremo que el anterior, ya que algunas categorías disponen de un única observación. Cabe recordar que el objetivo no es determinar la gradualización de las enfermedades, sino la presencia de ellas, siendo estas transformaciones llevadas a cabo en la fase de preprocesamiento.

Estadío	II	IIC	III
Frecuencia	3	17	20

Cuadro 5.7: Descripción de la variable *Estadío* relativa a la conjuntivalización.

Estadío	I	IA	IB	IC	IIA	IIB	IIC	III
Frecuencia	2	5	6	8	1	9	8	1

Cuadro 5.8: Descripción de la variable *Estadío* relativa a la neovasculización.

5.2. Preprocesamiento de los datos

En todo proyecto relativo a la Minería de Datos, es de vital importancia llevar a cabo una preparación de los datos, de cara a su posterior uso en la fase de modelado, siendo una etapa fundamental en el proceso de descubrimiento de información (*KDD*).

La calidad de los resultados obtenidos, posteriormente en la fase de modelado, depende en gran medida de la calidad propia de los datos, la cual depende directamente del preprocesamiento efectuado. Por consiguiente, se ha de prestar especial atención a esta etapa, ya que será un factor condicionante en la obtención de unos buenos resultados.

En las descripciones relativas a los dos conjuntos de datos, que se emplean en el desarrollo de este TFG, se puede percibir cómo no se ajustan de forma exacta al cometido de la clasificación. Por este motivo, el primer paso del preprocesamiento radica en realizar una conversión de las etiquetas relativas a las diferentes enfermedades a etiquetas binarias, indicando simplemente la presencia o no de la enfermedad, sin especificar el grado de la misma.

5.2.1. Preprocesamiento relativo al conjunto de datos 1

Como se comentó en la sección anterior, sobre el conjunto de datos 1 se enfoca el groso de este TFG. Por ello, sobre este conjunto de imágenes, se pretende explorar la importancia del preprocesamiento mediante la comparación de las resultados obtenidos tras el modelado, utilizando diferentes preprocesamientos.

Tras realizar las respectivas conversiones a clases binarias, se opta por crear una estructura de ficheros, tal que sea capaz de soportar un generador. De este modo, automáticamente se leen los imágenes, se decodifican en formato *RGB*, se convierten en tensores y se reescalan los valores de los pixeles al intervalo [0,1], rango de funcionamiento recomendado. No obstante, la utilización de un generador aumenta el tiempo de cómputo, por lo que se explora la alternativa de realizar un preprocesamiento puramente manual, convirtiendo manualmente las imágenes en arrays.

A continuación, se describe de forma más detallada el preprocesamiento efectuado sobre el conjunto de imágenes, para cada una de las enfermedades.

Edema macular diabético

Los valores de las etiquetas, relativas al edema macular diabético, están comprendidas en el *dataset* inicial en el rango [0,2]. Sin embargo, el objetivo establecido radica en determinar la presencia o ausencia de dicha enfermedad, siendo de este modo un problema de clasificación binaria. Por ello, es necesario convertir las etiquetas en una clase binaria. Tal y como se comentó, el valor 0 indica la ausencia de la enfermedad y, el resto de valores hacen referencia al grado de gravedad de la enfermedad. Para realizar la conversión requerida, simplemente se establecen a valor 1 todos los valores distintos a 0, indicando ahora este valor 1 la presencia de la enfermedad, independientemente de su grado. En las tablas 5.9 y 5.10 se muestra la descripción de las nuevas variables binarias, creadas tanto para el conjunto de entrenamiento como test.

Se opta por agrupar estos conjuntos predefinidos (agregando tanto las imágenes como las etiquetas) y crear, a posteriori, una partición en conjunto de entrenamiento y test

Conjunto de entrenamiento		
Grado	0	1
Frecuencia	177	236

Cuadro 5.9: Descripción de la variable binaria *Macular edema* en el conjunto de entrenamiento.

Conjunto test		
Grado	0	1
Frecuencia	45	58

Cuadro 5.10: Descripción de la variable binaria *Macular edema* en el conjunto test.

propia, asegurando la estratificación de las clases, es decir, preservando las proporciones entre las clases. En la tabla 5.11 se muestra la descripción de la variable binaria con la información relativa al entrenamiento y test agregada.

Conjunto total		
Grado	0	1
Frecuencia	222	294

Cuadro 5.11: Descripción de la variable *Macular edema* en la totalidad de los datos.

De este modo, se utiliza la función *train_test_split* del paquete *sklearn* para realizar esta partición del conjunto de datos total, indicando que el tamaño del conjunto test estará conformado por $\frac{1}{3}$ de la totalidad (siguiendo la típica división $\frac{2}{3}$, $\frac{1}{3}$). En este punto, cabe recordar la importancia de la división del conjunto de datos original en un conjunto de entrenamiento y, otro de prueba; dado que, para evaluar el ajuste del modelo, es necesario emplear un conjunto de datos independiente al utilizado en el entrenamiento, permitiendo obtener una aproximación real del error cometido. De no realizarse así, esta aproximación sería optimista, el modelo aprendería a clasificar expresamente los datos del conjunto de entrenamiento. En las tablas 5.12 y 5.13 se muestra la descripción de esta variable en los nuevos conjuntos de entrenamiento y test, en los que se puede observar cómo se mantiene la proporción de las clases.

Nuevo conjunto de entrenamiento		
Grado	0	1
Frecuencia	196	148

Cuadro 5.12: Descripción de la variable binaria *Macular edema* en el nuevo conjunto de entrenamiento.

Nuevo conjunto test		
Grado	0	1
Frecuencia	98	74

Cuadro 5.13: Descripción de la variable binaria *Macular edema* en el nuevo conjunto test.

Con el objetivo de poder hacer uso de un generador en la fase de modelado, es necesario crear una estructura de ficheros. Para ello, se utiliza el módulo *os* de *python*, que permite crear los directorios necesarios; y el módulo *shutil*, que permite copiar ficheros de un directorio a otro. Así pues, esta estructura está constituida, en primer nivel, por un directorio relativo al conjunto de entrenamiento y otro relativo al conjunto test. Cada uno de estos directorios, a su vez, deben contener tantos directorios como clases se desea predecir; en este caso, contienen dos clases (relativas a la presencia o ausencia de la enfermedad). Por último, en cada uno de estos directorios se encuentran las imágenes correspondientes.

Tal y como se comentó al inicio de esta sección, se ha optado por hacer uso de un generador en el preprocesamiento relativo a esta enfermedad. De esta manera, *keras* se encargará de realizar automáticamente las conversiones pertinentes para obtener los tensores correspondientes a las imágenes. El módulo relativo al preprocesamiento de las imágenes utilizado es *ImageDataGenerator*, contenido en *keras.preprocessing.image*. Esta clase permite especificar las transformaciones y operaciones de normalización que se desea aplicar a los datos. En primera instancia, simplemente se debe usar el argumento *rescale* para reescalar los valores de los pixel al intervalo [0,1].

A continuación, es necesario hacer uso del método *flow_from_directory*, para especificar el directorio en el que se encuentran las imágenes sobre las que se ajustará el modelo. En este caso, se especifica la ruta de la carpeta relativa al conjunto de datos de entrenamiento. Entre los argumentos de dicho método, también se utilizan los siguientes:

- target_size: Permite especificar las dimensiones a las que se quieren redimensionar todas las imágenes.
- *batch_size*: Permite especificar el tamaño de los lotes (indicando cada cuántas muestras se van a ajustar los pesos del modelo).
- *class_mode*: Permite especificar el tipo de etiquetas con el que se va a tratar. En este caso, el valor adecuado es *"binary"*, indicando que la clasificación a efectuar es binaria (2 etiquetas).

Al utilizar este preprocesamiento automático proporcionado por *keras*, se debe hacer uso del método *fit_generator* en la fase de modelado..

Hasta este punto, se contempla el uso de un generador, proporcionado por *keras*, con el objetivo de que la conversión a tensores de las imágenes se realice de forma automática, aunque el potencial de los generadores no reside en esta utilidad.

En la descripción de este conjunto de imágenes, se puede percatar cómo se trata de un conjunto no muy grande (516 imágenes), donde es habitual no conseguir una generalización correcta de los datos, sufriendo el modelo sobreajuste. En este contexto, una de las técnicas más empleadas para intentar solventar, en cierta medida, este problema y sacar el máximo valor, es utilizar la técnica *Data Augmentation*. A rasgos generales, esta técnica consiste en generar nuevas imágenes de entrenamiento a partir de las imágenes disponibles, utilizando una serie de transformaciones aleatorias, que sean capaces de producir imágenes de apariencia real. La aplicación de esta técnica propiciará que el modelo contemple más aspectos de las imágenes, consiguiendo una mejor generalización evitando, en cierta medida, el sobreajuste. [13]

En *keras*, se pueden configurar estas transformaciones aleatorias a través del generador (previamente sólo se reescalaban los valores de los píxeles). A continuación, se presentan algunas de las transformaciones disponibles, las cuales se han empleado en este preprocesamiento:

- *rotation_range*: Se especifica un valor en grados, de tal forma, que una imagen sufrirá una rotación comprendida entre 0 grados y el valor especificado, en este caso, de 40 grados sexagesimales.

- *width_shift_range*: Hace referencia a transformaciones relativas a desplazamientos en horizontal, estableciendo un rango entre el que se pueden trasladar la imágenes horizontalmente. Este valor se fija como una fracción de la anchura total de la imagen. En este caso, se establece este valor a 0.2.

- *heigth_shift_range*: Hace referencia a transformaciones relativas a desplazamientos en vertical, siendo su especificación igual que en el caso de los desplazamientos horizontales. También se opta por fijar este valor a 0.2.

- *shear_range*: Hace referencia a transformaciones relativas a recortar la imagen. Para ello se especifica un ángulo de corte en radianes. Este valor se establece en 0.2.

- *zoom_range*: Hace referencia a transformaciones relativas a aplicar zoom sobre una imagen. Se especifica un valor, y se aplica, para cada imagen, un zoom comprendido entre el rango [1-zoom_range, 1+zoom_range]. En este caso, se establece este valor a 0.2.

- *horizontal_flip*: Hace referencia a la posibilidad de voltear las imágenes. En este caso, es una opción que se desea tener en cuenta por lo que se establece su valor a True.

- *fill_mode*: Hace referencia a la estrategia utilizada para rellenar los pixeles necesarios tras aplicar las transformaciones. En este caso se opta por utilizar la estrategia *"nearest"*, siendo esta la opción por defecto.

No obstante, al tratarse de un conjunto de datos muy pequeño, probablemente no se consiga evitar el sobreajuste por completo. Para contribuir a solventar esta problemática, se exploran otras opciones en la fase de modelado.

Retinopatía diabética

Tal y como se observó en la descripción de los datos, se proporcionaba un mayor grado de especificación de la gravedad de esta enfermedad, estando comprendidos los valores de las etiquetas relativas a la retinopatía diabética en el rango [0,5]. Al igual que en el preprocesamiento relativo a la enfermedad anterior, es necesario convertir estas etiquetas en una clase binaria, ya que la tarea establecida subyacente es determinar la presencia o ausencia de la enfermedad. La estrategia seguida es la misma. En las tablas 5.14 y 5.15 se puede visualizar la descripción de estas nuevas variables binarias.

Conjunto de entrenamiento		
Grado	0	1
Frecuencia	134	279

Cuadro 5.14: Descripción de la variable binaria *Retinopathy* en el conjunto de entrenamiento.

Conjunto test		
Grado	0	1
Frecuencia	34	69

Cuadro 5.15: Descripción de la variable binaria *Retinopathy* en el conjunto test.

Del mismo modo que en el preprocesamiento relativo al edema macular, se opta por agrupar estos conjuntos predefinidos y llevar a cabo una partición del mismo, obteniendo unos nuevos conjuntos de entrenamiento y test, cuyos tamaños sigan las proporciones $\frac{2}{3}$, $\frac{1}{3}$, asegurando de la misma manera la estratificación de las clases. En la tabla 3.16, se proporciona la distribución de esta variable binaria en la totalidad agregada de los datos. Por otra parte, en las tablas 3.17 y 3.18 se adjuntan las descripciones relativas a esta variable en los nuevos conjuntos definidos.

Conjunto total		
Grado	0	1
Frecuencia	168	348

Cuadro 5.16: Descripción de la variable binaria *Retinopathy* en la totalidad de los datos.

En estas dos últimas tablas, se puede observar cómo al efectuar estas particiones de forma estratificada, en ambos conjuntos hay aproximadamente el doble de imágenes relativas a la presencia de la retinopatía diabética que a su ausencia.

Conjunto de entrenamiento		
Grado	0	1
Frecuencia	112	232

Cuadro 5.17: Descripción de la variable binaria *Retinopathy* en el nuevo conjunto de entrenamiento.

Conjunto test		
Grado	0	1
Frecuencia	56	116

Cuadro 5.18: Descripción de la variable binaria *Retinopathy* en el nuevo conjunto test.

Siguiendo la misma metodología descrita en el apartado anterior, se opta por crear una estructura de ficheros, organizando de este modo las imágenes, de tal manera que nos permita utilizar un generador en la fase de modelado. También se efectúa el preprocesamiento relativo a la aplicación de la técnica *Data Augmentation* descrito en el apartado anterior, haciendo uso de la misma configuración.

Por otra parte, en esta tarea de clasificación se ha elegido explorar posteriormente el preprocesamiento relativo a las imágenes de forma manual, sin hacer uso de generadores. En la fase de modelado, se compara respecto a la utilización de un generador, que se encargue de realizar el preprocesamiento básico de forma automática, respecto a realizar el mismo de forma manual.

Con el objetivo de llevar a cabo este preprocesamiento de forma manual, se hace uso de la librería *Pillow*, en particular de su módulo *Image*, el cual posibilita cargar y manipular imágenes. Inicialmente, se opta por crear un array, en el cual se irán almacenando las decodificaciones de las imágenes en formato *RGB*, obteniendo una matriz de pixeles para cada canal. Se opta por cargar las sucesivas imágenes haciendo uso del método *open* y, redimensionando cada una de ellas mediante el método *resize*, estableciendo las mismas dimensiones que las especificadas en el generador (429,285). Una vez redimensionadas, se debe convertir cada imagen a formato *NumPy* con el objetivo de obtener la decodificación correspondiente. Para ello, se emplea simplemente el método *array* de la librería *NumPy*, especificando como argumento la imagen que se desea codificar. De este modo, se van almacenando las respectivas conversiones en el array creado inicialmente. Como paso final de este preprocesamiento básico (el cual realiza de forma automática el generador), se escalan los valores de las decodificaciones para que estén comprendidas en el intervalo [0,1], dividiendo para ello entre 255 cada uno de los valores (intensidad máxima de cada color básico).

Este preprocesamiento manual conlleva utilizar el método *fit* para ajustar el modelo, ya que el formato de los datos de entrada son matrices. Por ello, también se debe proporcio-

nar, a dicho método, las etiquetas correspondientes a las diferentes imágenes en formato *NumPy*, ya que requiere de la especificación explícita de la variable respuesta, obteniendo esta mediante un proceso de binarización.

Por último, siguiendo la línea del preprocesamiento manual y tras la exploración manual de las imágenes, se opta por testear, en esta tarea de clasificación, el preprocesamiento relativo a recortar las imágenes. Este planteamiento radica en la propia fisionomía de las imágenes de fondo de ojo, en las cuales se visualiza el fondo de ojo sobre un fondo negro, abarcando este fondo parte de los laterales de la imagen. Por ello, se plantea recortar estos bordes laterales, para que la red neuronal no malgaste recursos en explorar dicha área.

Para llevar a cabo este cometido, se vuelve a hacer uso del módulo *Image*, reduciéndose este preprocesamiento a determinar el intervalo relativo a la anchura de la imagen correspondiente a la información de interés, es decir, excluyendo en la medida de lo posible el fondo negro, en función del cual se recorta la imagen original.

Para determinar el intervalo correspondiente a cada imagen, en primer lugar se carga la imagen de interés mediante el método *open*, y se realiza su conversión a la escala de grises, mediante el método *convert* especificando el modo *L*; convirtiendo la imagen en blanco y negro en una matriz de formato *NumPy* mediante el método *array*. Ahora bien, sobre esta matriz se debe determinar el inicio y el fin del intervalo. Para agilizar los cálculos, se procede a buscar el inicio del intervalo entre la primera mitad de la totalidad de las columnas y el fin en la segunda mitad. Así pues, para obtener el valor del inicio del intervalo se sigue el siguiente procedimiento:

- Se obtiene el máximo valor relativo a cada columna (de las columnas contempladas).
- Se evalúa cuales de estos valores son menores a un umbral establecido que hace referencia al color negro (en este caso, este umbral toma el valor 25).
- Se obtienen los índices de las columnas relativos a los valores que cumplen la condición anterior.
- Se establece como inicio de intervalo el índice máximo entre el conjunto de índices anterior.

Por otra parte, el valor del fin del intervalo se calcula siguiendo un procedimiento muy similar aplicado a las columnas relativas a la segunda mitad. Los tres primeros pasos se mantienen iguales. Finalmente, se selecciona el valor mínimo entre el conjunto de índices obtenido y se le suma el valor relativo a la mitad de las columnas, correspondiéndose este valor al final del intervalo deseado.

Una vez obtenido el intervalo horizontal, se procede a utilizar el método *crop* para recortar la imagen. Obviamente, no se realiza ningún recorte en torno al eje vertical. Finalmente, se guarda la foto recortada mediante el método *save*.

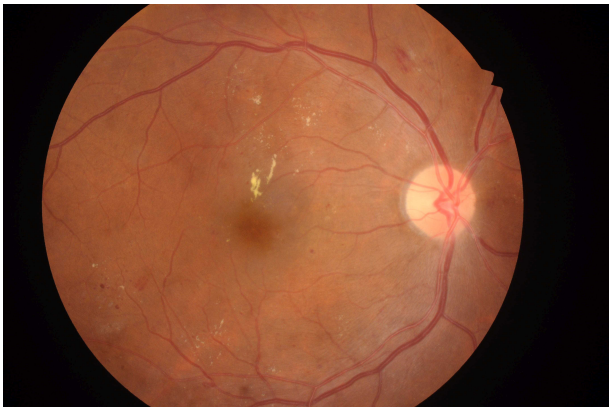


Figura 5.1: Ejemplo de imagen de fondo de ojo.

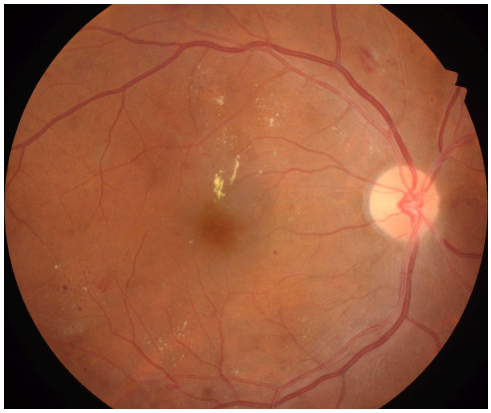


Figura 5.2: Ejemplo de imagen de fondo de ojo recortada.

Para corroborar el buen funcionamiento de esta técnica de recorte, se procede a calcular la anchura del intervalo obtenido. En el caso de que esta anchura sea muy pequeña, esta imagen se rechaza. Con el umbral establecido, no se efectúa un mal recorte sobre ninguna de las imágenes, pudiendo utilizar la totalidad de las imágenes. No obstante, si el valor de este umbral se establece en un valor algo más elevado, experimentalmente se detecta que algunas imágenes sufrirían un mal recorte y deberían ser descartadas.

En la Figura 5.1 se puede visualizar una imagen del fondo de ojo, y en la Figura 5.2 el resultado del recorte eliminando una gran parte de la zona negra.

5.2.2. Preprocesamiento relativo al conjunto de datos 2

Tal y como se comentó en la propia descripción de este conjunto de datos, el estudio de esta base de imágenes oculares se efectúa con el objetivo de mostrar la capacidad de aprendizaje proporcionada por las redes neuronales, incluso cuando se dispone de muy pocos datos. En este contexto, el preprocesamiento que se efectúa no es excesivo, sino que contempla únicamente las necesidades básicas con el fin de aplicar un modelo de redes neuronales convolucionales a posteriori.

Al igual que en el conjunto de datos anterior, este conjunto también proporciona información relativa a dos enfermedades distintas: conjuntivalización y neovasculización. En este caso, el preprocesamiento realizado es el mismo para ambas enfermedades, consistiendo en realizar en primer lugar la conversión de las etiquetas a etiquetas binarias; y en segundo lugar, convertir manualmente las imágenes en arrays (decodificación de las imágenes en formato RGB), tal y como se abordó en el preprocesamiento relativo a la retinopatía diabética.

A continuación, se describe de forma algo más detallada el preprocesamiento relativo a la conversión, en tareas de clasificación binarias, de ambas enfermedades.

Conjuntivalización

En el caso de la conjuntivalización, se puede observar en la descripción cómo se proporcionan tres posibles posibles estados relativos a esta enfermedad. De cara a su conversión en un problema de clasificación binaria, se agregan los estadíos II y IIC para conformar la clase 0 y se hace uso del estadío III como clase 1. En la Tabla 5.19, se puede visualizar la descripción de esta nueva variable binaria, pudiendo resultar de interés cómo en este nuevo problema planteado, los datos están balanceados.

Estadío	0	1
Frecuencia	20	20

Cuadro 5.19: Descripción de la variable binaria *Estadío* relativa a la conjuntivalización.

A continuación, se efectúa la partición de este conjunto de imágenes, en un conjunto de entrenamiento y un conjunto test, cuyos tamaños siguen las proporciones $\frac{2}{3}$, $\frac{1}{3}$, con el objetivo de poder obtener una aproximación real a la eficacia del modelo. En las tablas 5.20 y 5.21 se adjuntan las distribuciones de dicha variable, tanto en el conjunto de entrenamiento como en el conjunto test.

Estadío	0	1
Frecuencia	13	13

Cuadro 5.20: Descripción de la variable binaria *Estadío* relativa a la conjuntivalización en el conjunto de entrenamiento.

Estadío	0	1
Frecuencia	7	7

Cuadro 5.21: Descripción de la variable binaria *Estadío* relativa a la conjuntivalización en el conjunto test.

Neovasculización

Tal y como se observó en la descripción de la variable relativa a dicha enfermedad en el conjunto de datos, sus posibles etiquetas son mucho mayores que en el resto de casos abordados (8 etiquetas posibles). Según la significación de cada uno de los estadíos proporcionados, se opta por agrupar los estadíos I, IA, IB y IC en una clase (clase 0) y los estadíos IIA, IIB, IIC en otra clase (clase 1). Debido a que el estadío III dispone de una única observación y difiere totalmente de los anteriores (representando un nivel

de gravedad mayor), se opta por prescindir de él en el experimento. En la Tabla 5.22 se puede observar la descripción de la variable binaria establecida.

Estadío	0	1
Frecuencia	21	18

Cuadro 5.22: Descripción de la variable binaria *Estadío* relativa a la neovasculización.

Al igual que en los casos anteriores, se procede a dividir este conjunto de datos en un conjunto de entrenamiento y un conjunto test, garantizando la estratificación de los mismos, y bajo la razón de proporciones señalada. En las tablas 5.23 y 5.24 se adjuntan las descripciones relativas a esta variable en los conjuntos obtenidos.

Estadío	0	1
Frecuencia	14	12

Cuadro 5.23: Descripción de la variable binaria *Estadío* relativa a la neovasculización en el conjunto de entrenamiento.

Estadío	0	1
Frecuencia	7	6

Cuadro 5.24: Descripción de la variable binaria *Estadío* relativa a la neovasculización en el conjunto test.

Capítulo 6

Modelado de los datos

En este capítulo, se pretende abordar la fase de modelado relativa a cualquier proyecto referente a Minería de Datos. Consiste en la aplicación de diversas técnicas para una correcta configuración de los valores de los parámetros requeridos, con el objetivo de obtener un modelo que se ajuste lo mejor posible a los datos. Todo ello con vistas a la detección de patrones y características ocultas para obtener a posteriori una buena precisión en las predicciones.

En dicho TFG la selección de las técnicas de modelado está prefijada de antemano. Tal y como se expuso en el apartado 3.2.4, una estructura de red neuronal está compuesta por varios bloques relativos a las operaciones convolución/pooling y por una última capa totalmente conectada. En este contexto, la fase de modelado se centra en diseñar una estructura de red neuronal, que sea capaz de afrontar con la mayor precisión posible las tareas de clasificación. De este modo, se exponen las diferentes estructuras de redes convolucionales valoradas para cada uno de los problemas de clasificación binaria.

En las descripciones de ambos conjuntos de datos, se puede observar cómo están conformados por un número muy pequeño de observaciones. Por ello, se intenta poner de manifiesto el impacto de la utilización de diversas estrategias recomendadas de cara a mejorar la precisión de un modelo entrenado con un pequeño conjunto de datos. Entre estas técnicas se encuentran:

- *Data augmentation*
- Utilización de una capa de tipología *Dropout*
- Utilización de un modelo más complejo

En este punto, se considera de interés especificar que la totalidad de los modelos probados se han ejecutado sobre una máquina virtual proporcionada por el Departamento de Informática, estando dotada con 8 núcleos, ya que el tiempo de entrenamiento de los modelos está condicionado a las características de la máquina en la que se ejecuta.

Por otra parte, en el capítulo 4 se expusieron las diferentes estrategias proporcionadas por la librería *keras* a la hora de entrenar una red neuronal, donde se recalcó la importancia de evaluar la adecuación de la aplicación de la técnica *transfer learning* en cada caso concreto. Explorando los diferentes modelos pre-entrenados, se puede observar como ninguno de ellos ha sido entrenado con un conjunto de imágenes similares al utilizado en este TFG; es decir, ninguno de los dominios empleados es similar al requerido (imágenes oculares). Por ello, se concluye que la aplicación de esta técnica no es adecuada en los problemas tratados en este proyecto.

6.1. Modelización del diagnóstico del edema macular diabético

El objetivo de esta primera tarea de clasificación consiste en crear una red neuronal convolucional, que sea capaz de detectar automáticamente y, con la mayor precisión posible, el edema macular diabético basándose sólo en una imagen del fondo de ojo.

Tras llevar a cabo el preprocesamiento correspondiente a esta tarea explicado en el apartado 5.2.1, la dinámica empleada de cara a obtener una estructura de red neuronal satisfactoria es incremental, es decir, en primer lugar se implementa una estructura sencilla comúnmente utilizada y se procede a evaluar el impacto de diversas mejoras sobre el modelo inicial.

6.1.1. Modelo inicial

En el contexto descrito, la primera aproximación a esta tarea de clasificación consiste en elaborar una estructura de red neuronal convolucional sencilla que sirva de punto de partida. Inicialmente, estará conformada por:

- Cuatro bloques compuestos por una capa convolucional con función de activación *ReLu* y una capa de agrupamiento *max pooling*.
- Una capa encargada de aplanar la salida obtenida.
- Dos capas totalmente conectadas, la última compuesta por una única unidad y utilizando la función de activación sigmoide, que proporcione la probabilidad de pertenencia a cada clase.

En la Figura 6.1 se adjunta esta estructura descrita de forma detallada, observando en la primera columna el tipo de capa, en la segunda columna el tamaño de la salida y en la última columna el número de parámetros a ajustar en cada capa. Se puede observar cómo hasta la utilización de la capa *Flatten*, las salidas se presentan como mapas de características y, tras la utilización de la misma, la salida se reduce a un vector unidimensional. Por otro lado, se puede observar cómo se ha optado por un incremento del número de filtros

en la especificación de cada capa convolución (32, 64 y 128), aunque en la última capa convolucional se utiliza el mismo número de filtros. Por último, cabe destacar el número de parámetros ajustables en esta estructura relativamente sencilla, siendo un número bastante elevado, el cual no se podría afrontar de no ser por el aumento de la capacidad computacional.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 427, 283, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 213, 141, 32)	0
conv2d_2 (Conv2D)	(None, 211, 139, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 105, 69, 64)	0
conv2d_3 (Conv2D)	(None, 103, 67, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 51, 33, 128)	0
conv2d_4 (Conv2D)	(None, 49, 31, 128)	147584
max_pooling2d_4 (MaxPooling2	(None, 24, 15, 128)	0
flatten_1 (Flatten)	(None, 46080)	0
dense_1 (Dense)	(None, 512)	23593472
dense_2 (Dense)	(None, 1)	513

Total params: 23,834,817
Trainable params: 23,834,817
Non-trainable params: 0

Figura 6.1: Edema macular - Estructura básica de la red convolucional.

Tal y como se comentó en el capítulo 4, tras definir la estructura se lleva a cabo el entrenamiento de la red, siendo necesaria la configuración del mismo previamente. Se opta por utilizar la función *loss* como función de pérdida y el algoritmo RMSProp para la optimización. De cara a evaluar el rendimiento del modelo, se especifica el uso de la precisión, la cual hace referencia al porcentaje de aciertos.

Por otra parte, debido a la creación de la estructura de ficheros que se realizó durante el preprocesamiento, los datos de entrada se obtienen a partir de un generador (pero en este caso sin realizar más transformaciones que la pertinente al reescalado de los pixeles), estando especificados en esta estructura de ficheros los conjuntos de entrenamiento y test. En cuanto al generador de los datos de entrada, se establecen las dimensiones a las que se desea redimensionar las imágenes y el tamaño del *batch*, fijado este último valor a 20.

Finalmente, para realizar el ajuste se opta por utilizar 100 épocas en el entrenamiento, estando conformada una época por el entrenamiento relativo a 100 imágenes extraídas por el generador. En las Figuras 6.2 y 6.3 se puede observar la evolución de la función de pérdida y de la precisión, tanto para el conjunto de entrenamiento como para el conjunto test, a lo largo del entrenamiento del modelo (siendo elaborados estos gráficos a partir de la librería *matplotlib*).

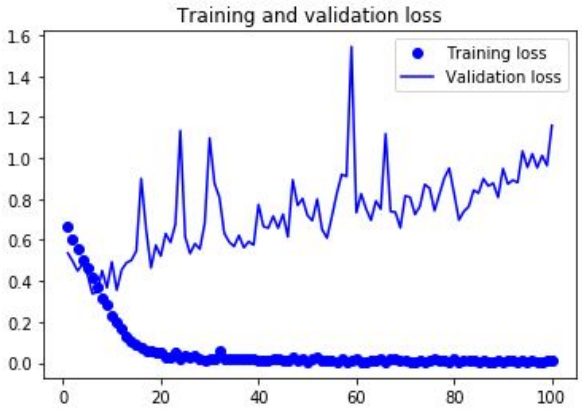


Figura 6.2: Edema macular - Evolución de la función de pérdida del modelo básico.

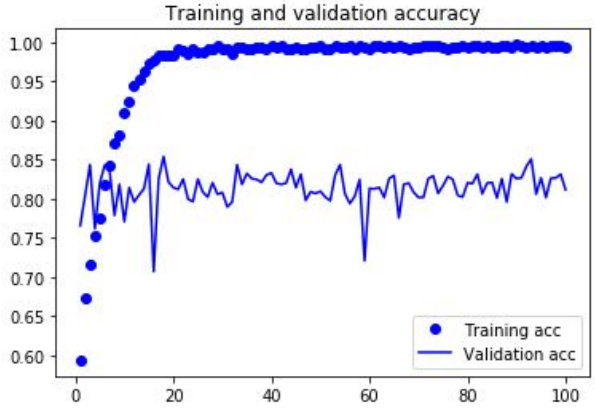


Figura 6.3: Edema macular - Evolución de la precisión del modelo básico.

En la gráfica relativa a la función de pérdida, se puede observar cómo este valor se va minimizando en el transcurso del ajuste sobre el conjunto de entrenamiento, llegando a ser prácticamente 0. En cuanto al gráfico relativo a la precisión, se puede observar cómo en las primeras épocas aumenta notoriamente, hasta alcanzar el 100 % de precisión aproximadamente, sobreajustando en este punto los datos (debiéndose a que el conjunto de datos está conformado por pocas imágenes). En cuanto a la precisión relativa al conjunto test, se puede observar como en la mayoría de las épocas se alcanza una precisión mayor al 80 %, alcanzando en algunas de ellas un 85 % de precisión, un resultado muy prometedor teniendo en cuenta que se trata de un modelo de partida.

Tras finalizar el entrenamiento de esta estructura, se evalúa el rendimiento final de la misma sobre el conjunto test. En la Figura 6.4 se puede observar la matriz de confusión relativa a este modelo, siendo su precisión (*accuracy*) del **81,395 %**.

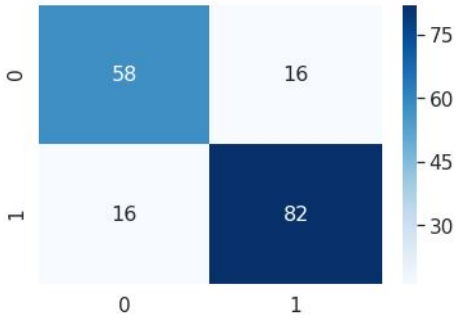


Figura 6.4: Edema macular - Matriz de confusión asociada al modelo inicial.

A pesar de que esta medida es la más conocida y utilizada, la precisión trata a todos los ejemplos del conjunto test por igual, por lo que dependiendo del problema concreto puede que esta medida no sea la más adecuada, existiendo otras que pueden resultar de interés para evaluar la capacidad predictiva del modelo. En el ámbito de las pruebas diagnóstico son de uso frecuente dos medidas que tratan de proporcionar una aproximación más detallada:

- Sensibilidad: Hace referencia a la proporción de enfermos que la prueba diagnóstico es capaz de identificar correctamente, es decir, evalúa su capacidad de detección de la enfermedad. Considerando la clase P (positivos) como la clase relativa a la presencia de la enfermedad, esta medida se define como:

$$Sensibilidad = \frac{VP}{VP + FN} \tag{6.1}$$

- Especificidad: Hace referencia a la proporción de sanos que la prueba diagnóstico identifica correctamente, es decir, evalúa su capacidad de detección de la ausencia de la enfermedad. Esta medida se define como:

$$Especificidad = \frac{VN}{VN + FP} \tag{6.2}$$

En un ámbito más genérico, también se considera de interés valorar la precisión (distinta a *accuracy*). Esta medida hace referencia a la fracción de ejemplos correctamente clasificados del total de ejemplos clasificados en una determinada clase, es decir, cuántos de los predichos son verdaderamente de la categoría de interés. Su definición es:

$$Precision = \frac{VP}{VP + FP} \tag{6.3}$$

Por último, en situaciones en la que se está interesado en tener en cuenta un único valor numérico considerando estas medidas más específicas, buscando un equilibrio entre ambas, se hace uso de la medida *F1 Score*, siendo la media armónica de la sensibilidad y la precisión:

$$F1\ Score = \left(\frac{Precisin^{-1} + Sensibilidad^{-1}}{2}\right)^{-1} = 2 \times \frac{Sensibilidad \times Precisin}{Sensibilidad + Precisin} \tag{6.4}$$

En la Tabla 6.1 se adjuntan los valores de este conjunto de medidas para este primer modelo planteado, los cuales se pueden obtener de forma directa a partir de la función *classification_report* de la librería *sklearn*.

Se puede observar cómo los valores relativos a la sensibilidad y a la especificidad son iguales, debido a que el número de falsos positivos y falsos negativos es el mismo, tal y como se puede observar en la matriz de confusión. Acorde a los valores de estas medidas, se puede volver a concluir que los resultados de este modelo son prometedores.

Accuracy	0.81
Sensibilidad	0.84
Especificidad	0.78
Precisión	0.84
F1 score	0.84

Cuadro 6.1: Resultados de las métricas de evaluación del modelo básico.

6.1.2. Modelo básico aplicando la técnica *Data augmentation*

Una vez evaluado el rendimiento de un modelo básico, se considera de interés valorar, en primer lugar, el impacto de la técnica *Data augmentation*, ya que debido a que el conjunto de imágenes utilizado es relativamente pequeño, el modelo básico sufría sobreajuste.

Se hace uso de la misma estructura de red neuronal convolucional descrita en la Figura 6.1, modificando exclusivamente la parte relativa al preprocesamiento de los datos de entrada, configurando en la definición del generador una serie de transformaciones aleatorias, que van a permitir generar nuevas imágenes a partir del conjunto de imágenes disponibles. En la subsección relativa al edema macular del apartado 5.2.1, se proporciona una descripción más detallada de esta técnica, así como de las transformaciones aleatorias empleadas.

Haciendo uso de la misma configuración del proceso de aprendizaje definida para el modelo básico, se procede a entrenar el modelo. En las Figuras 6.5 y 6.6 se pueden visualizar las trayectorias de la evolución a lo largo del entrenamiento de esta red.

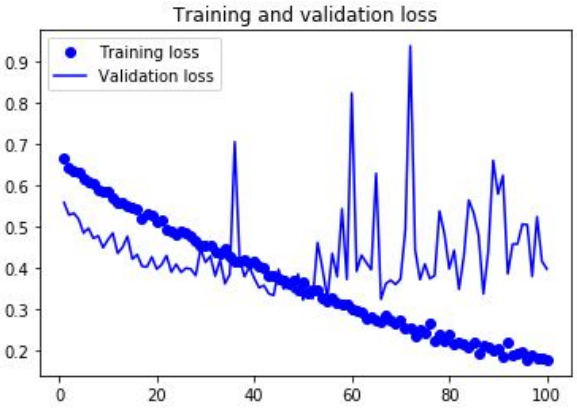


Figura 6.5: Edema macular - Evolución de la función de pérdida del modelo básico utilizando *Data augmentation*.

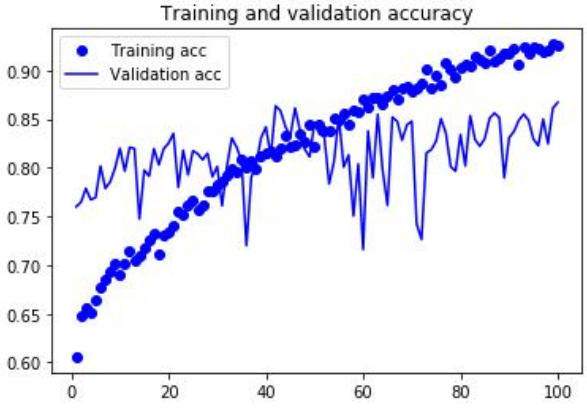


Figura 6.6: Edema macular - Evolución de la precisión del modelo básico utilizando *Data augmentation*.

En este caso, se puede observar como la función de pérdida se minimiza más lentamente, siendo consecuencia directa de la inclusión de nuevas imágenes a través de la técnica *data augmentation*. También se puede observar su influencia en la evolución de la precisión,

que no alcanza su máximo valor en iteraciones tempranas, sino que sigue una progresión a lo largo de la fase de entrenamiento, disminuyendo de esta forma el sobreajuste. Por consiguiente, se puede determinar que gracias a la aplicación de esta técnica, el modelo se ajusta de forma más general. En cuanto a la precisión del modelo evaluada en el conjunto test, se puede observar una ligera mejoría respecto al modelo anterior, llegando a tener una mayor estabilidad en torno al 85 %, siendo la precisión final del modelo ajustado del **86,768 %**, aumentando esta técnica la precisión del modelo aproximadamente en un 5 %.

Se valora, de forma algo más detallada, el rendimiento del modelo según la matriz de confusión presentada en la Figura 6.7, donde se puede observar cómo el número de imágenes correctamente clasificadas ha incrementado en ambas categorías, notando una mejoría algo mayor en el diagnóstico de la ausencia de la enfermedad.

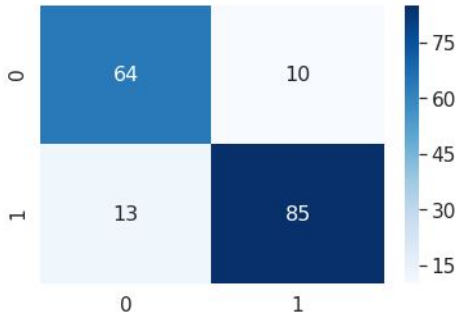


Figura 6.7: Edema macular - Matriz de confusión asociada al modelo básico utilizando *Data augmentation*.

En esta línea, se adjuntan en la Tabla 6.2 las medidas de las métricas expuestas anteriormente, utilizadas en la tarea de evaluar el rendimiento.

Accuracy	0.87
Sensibilidad	0.87
Especificidad	0.86
Precisión	0.89
F1 score	0.88

Cuadro 6.2: Resultados de las métricas de evaluación del modelo básico utilizando *Data augmentation*.

Respecto al modelo anterior, a primera vista se puede observar cómo el valor relativo a todas las medidas se ha visto incrementado en mayor o menor grado, contando no sólo con una mayor *accuracy*, sino con una mayor sensibilidad y especificidad, mejorando el modelo en la detección de ambas clases. Por otro lado, la precisión se ve incrementada notoriamente, de modo que las imágenes en las que se detecte la enfermedad, tendrán

una probabilidad de casi el 90 % de que dicho diagnóstico sea correcto. No obstante, de cara a pruebas diagnósticas resulta de mayor interés tener un valor más alto en la medida relativa a la sensibilidad, ya que el costo de no detectar la enfermedad es mayor que el relativo a la detección errónea de la enfermedad.

De acuerdo con estos resultados, se puede concluir que la técnica *Data augmentation* ayuda a que el modelo generalice mejor, proporcionando mejores resultados en la predicción de nuevas imágenes (aproximadamente un 5 % más de precisión). Por esta razón, se considera satisfactoria la utilización de esta técnica en esta tarea de clasificación y se opta por su aplicación en posteriores intentos de optimizar el modelo. Así pues, empleando esta técnica se ha podido corroborar la importancia de la fase de preprocesamiento para obtener unos buenos resultados.

6.1.3. Modelo complejo utilizando la técnica *Data augmentation*

Con el objetivo de obtener unos mejores resultados, se opta por evaluar el impacto de utilizar un modelo más complejo. En el ámbito de la oftalmología, la detección del edema macular por parte de profesionales en la materia a partir de una imagen no es considerada una tarea sencilla, por lo que un modelo más complejo puede proporcionar una mayor extracción de características, las cuales pueden resultar útiles en esta clasificación.

Esta nueva estructura también está compuesta por cuatro bloques que engloban las operaciones convolución y *max pooling*. La complejidad añadida radica en la utilización de dos capas convolucionales en cada bloque (utilizando ambas la función de activación *ReLu* sobre sus salidas) en lugar de una. Con esta adicción de cuatro capas convolucionales al modelo, el número de parámetros se cuadriplica aproximadamente, por lo que la complejidad del modelo resulta mayor, disponiendo de un mayor número de parámetros por ajustar. Esta estructura se puede visualizar íntegramente en la Figura 6.8.

Una vez definida esta nueva estructura, se procede a realizar el entrenamiento de la red. Se mantiene la misma configuración del proceso de aprendizaje, haciendo uso de la técnica de *Data augmentation* como parte del preprocesamiento.

Bajo estas especificaciones, se lleva a cabo el entrenamiento de esta estructura más compleja. En las Figuras 6.9 y 6.10 se puede visualizar tanto la evolución de la función de pérdida (función que determina la actualización de los valores de los pesos) como la evolución de la precisión. En cuanto a la función de pérdida, su trayectoria relativa al conjunto de entrenamiento va disminuyendo conforme aumenta el número de iteraciones, con una mayor reducción, mucho más rápida, en las primeras actualizaciones (sobre todo en las 15 primeras). Por el contrario, en la evolución de dicha función sobre el conjunto test se puede observar como para determinadas iteraciones finales, el valor de la función de pérdida se ve incrementado.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 427, 283, 32)	896
conv2d_2 (Conv2D)	(None, 425, 281, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 212, 140, 32)	0
conv2d_3 (Conv2D)	(None, 210, 138, 64)	18496
conv2d_4 (Conv2D)	(None, 208, 136, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 104, 68, 64)	0
conv2d_5 (Conv2D)	(None, 102, 66, 128)	73856
conv2d_6 (Conv2D)	(None, 100, 64, 128)	147584
max_pooling2d_3 (MaxPooling2	(None, 50, 32, 128)	0
conv2d_7 (Conv2D)	(None, 48, 30, 256)	295168
conv2d_8 (Conv2D)	(None, 46, 28, 256)	590080
max_pooling2d_4 (MaxPooling2	(None, 23, 14, 256)	0
flatten_1 (Flatten)	(None, 82432)	0
dense_1 (Dense)	(None, 1024)	84411392
dense_2 (Dense)	(None, 1)	1025

Total params: 85,584,673
Trainable params: 85,584,673
Non-trainable params: 0

Figura 6.8: Edema macular - Estructura más compleja de la red convolucional.

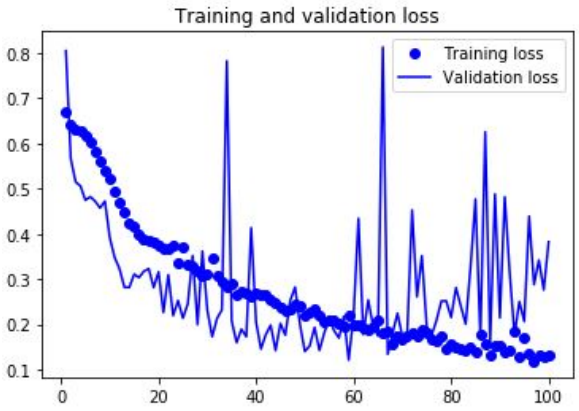


Figura 6.9: Edema macular - Evolución de la función de pérdida del modelo complejo.

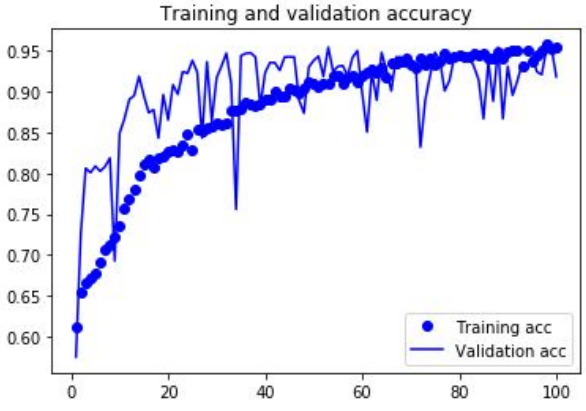


Figura 6.10: Edema macular - Evolución de la precisión del modelo complejo.

En la función relativa a la precisión del modelo, este valor aumenta rápidamente en las primeras iteraciones de la fase de entrenamiento y, en el resto de iteraciones, se produce

un aumento más sosegado.

Respecto a la evaluación del modelo sobre el conjunto test, la precisión del modelo se ha visto incrementada, llegando a alcanzar en ciertas iteraciones una precisión mayor al 95 %. Por desgracia, el entrenamiento es un proceso iterativo y aleatorio, llevando en esta ejecución a reducir en un 4 % la precisión del modelo respecto a su iteración anterior, siendo del **91,79 %**. Este entrenamiento se podría repetir con el objetivo de conseguir una actualización más satisfactoria de los pesos, pero el tiempo de esta fase de entrenamiento es muy elevado, excediendo el periodo de varios días, debiéndose estos largos periodos a la utilización del generador y su almacenaje en memoria de las fotos.

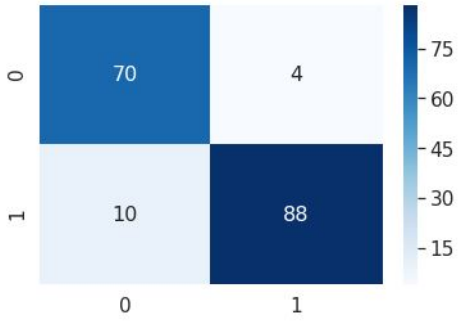


Figura 6.11: Edema macular - Matriz de confusión asociada al modelo básico utilizando *Data augmentation*.

Con el objetivo de obtener una visualización más representativa de los buenos resultados proporcionados por este modelo, se muestra en la Figura 6.11 la matriz de confusión asociada, en la que tanto el número de falsos positivos como falsos negativos se ha reducido, aunque en mayor medida el número de falsos positivos. En líneas generales, el modelo clasifica únicamente 14 imágenes de forma errónea sobre el total de las imágenes que componen el conjunto test.

En relación a la matriz de confusión, en la tabla 6.3 se adjuntan los valores relativos a las medidas utilizadas para la medición del rendimiento del modelo.

Accuracy	0.92
Sensibilidad	0.90
Especificidad	0.95
Precisión	0.96
F1 score	0.93

Cuadro 6.3: Resultados de las métricas de evaluación del modelo complejo.

A primera vista, todas ellas presentan un valor igual o mayor a 0.9, es decir, numéricamente el rendimiento del modelo ha mejorado (tal y como se observaba tanto en las

gráficas como en la matriz de confusión), siendo los valores relativos a algunas medidas mayores al 0.95. El valor de la sensibilidad es el más bajo de todos (0.9), siendo considerada una métrica importante en el ámbito de las pruebas diagnóstico. No obstante, la evaluación del rendimiento según este conjunto de métricas se considera muy satisfactorio.

Vistos estos resultados, se puede determinar rotundamente que la utilización de un modelo más complejo de cara a obtener unos mejores resultados ha propiciado una mejora considerable en los resultados proporcionados por el modelo, considerando que una precisión mayor al 90 % es un resultado lo suficientemente bueno, como para seleccionarlo para llevar a cabo el diagnóstico del edema macular.

En el desarrollo de esta tarea de clasificación, se abordó posteriormente un proceso de afinación de la estructura de la red neuronal (ya que siempre y cuando los resultados sean similares se considera mejor utilizar el modelo más sencillo posible).Esta premisa no se cumplió por lo que la especificación de los modelos así como los resultados obtenidos no se considera necesaria.

Como se comentó anteriormente, esta enfermedad en el campo de la Oftalmología es de difícil detección, siendo los resultados obtenidos muy satisfactorios. La razón de la buena capacidad de predicción de la enfermedad puede radicar en que la red neuronal aprendió características que habitualmente no son valoradas o perceptibles para el ser humano.

6.2. Modelización del diagnóstico de la retinopatía diabética

El objetivo de esta segunda tarea de clasificación está ligada a la primera tarea abordada, ya que la detección de ambas enfermedades se realiza en base a la misma imagen de fondo de ojo. Además, en el campo de la Oftalmología, se considera que el edema macular es un factor de riesgo de la retinopatía diabética, de modo que si en una imagen se detecta el edema macular hay una probabilidad mayor de que ese paciente también sufra retinopatía diabética. En la misma línea que el problema anterior, esta tarea consiste en crear una red neuronal convolucional capaz de detectar automáticamente y con la mayor precisión posible la retinopatía diabética.

El enfoque que se decide utilizar en esta tarea de clasificación también es incremental. En primer lugar se opta por implementar una estructura sencilla que sirva como punto de partida, aplicando y evaluando en los modelos posteriores diversas prácticas recomendadas. En este caso, el estudio efectuado es algo más exhaustivo, ya que se explora la diferencia entre el uso de un generador y la realización de un preprocesamiento manual de las imágenes, encontrando este preprocesamiento detallado el apartado 5.2.1. Tras una

exploración de las imágenes de fondo de ojo, también se procede a evaluar el impacto del preprocesamiento relativo a llevar a cabo un recorte de estas imágenes.

Por último, se considera de interés abordar la posibilidad de incluir en este modelo la información relativa a la detección del edema macular, ya que esta enfermedad es un factor de riesgo de la retinopatía diabética, trabajando en este supuesto con la creación de un modelo más complejo.

6.2.1. Modelo inicial obteniendo los datos de entrada de un generador

En primer lugar, se opta por elaborar una estructura de red neuronal sencilla la cual va a permitir establecer una línea base de los resultados que se pueden lograr en esta tarea de clasificación. La estructura básica por la que se opta es la misma que la utilizada en el modelo inicial de la tarea relativa al edema macular, pudiendo observarla en la Figura 6.1. Se recuerda que esta estructura está compuesta por cuatro bloques conformados por una capa convolucional y una capa de agrupamiento *max pooling*.

Tal y como indica este subsección, se procede a entrenar este modelo utilizando un generador, el cual proporciona los datos de entrada a lo largo del entrenamiento. Por otra parte, se mantiene la configuración del proceso de aprendizaje fijada desde el inicio del capítulo. En este punto, cabe destacar como la única diferencia existente entre este modelo inicial y el relativo al diagnóstico del edema macular reside en los datos de entrada, en particular, en la estructura de ficheros creada en base a las etiquetas, ya que las imágenes que se utilizan son las mismas en ambas tareas.

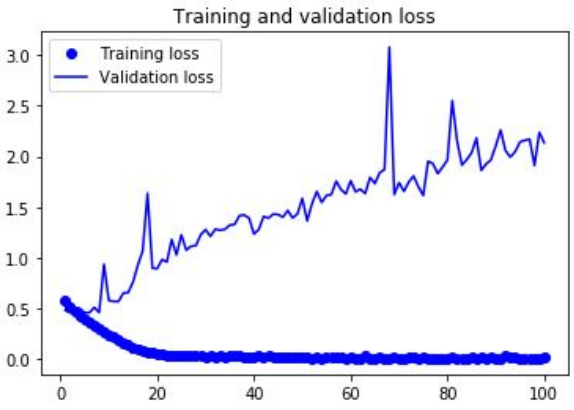


Figura 6.12: Retinopatía diabética - Evolución de la función de pérdida del modelo básico.

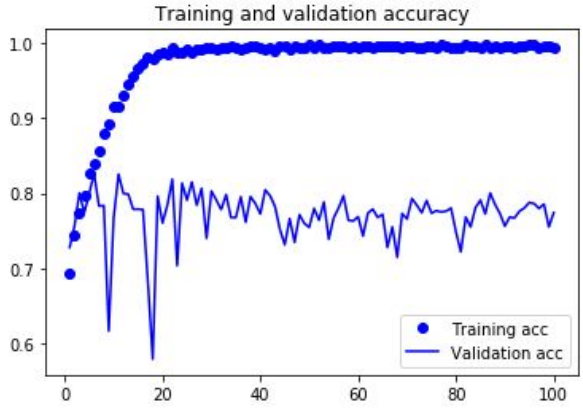


Figura 6.13: Retinopatía diabética - Evolución de la precisión del modelo básico.

Bajo estas especificaciones se lleva a cabo el entrenamiento del modelo. En las Figuras 6.12 y 6.13 se puede observar la evolución de la función de pérdida y de la precisión

a lo largo del proceso de aprendizaje. En la gráfica relativa a la función de pérdida, la minimización en el entrenamiento del modelo converge muy rápidamente al valor 0 (aproximadamente en la iteración 20). A su vez, en torno a esta misma iteración, en la gráfica relativa a la evolución de la precisión se alcanza el 100 % de la precisión aproximadamente, siendo estas características claros indicios de sobreajuste.

En cuanto a la trayectoria de estas funciones valoradas sobre el conjunto test, la función de pérdida toma valores cada vez mayores. Por el contrario, la precisión del modelo no aumenta, siendo menor en las últimas iteraciones que en las iniciales. Asimismo, la precisión del modelo, evaluada en el conjunto test, es mucho menor que en el de entrenamiento, corroborando la hipótesis de que el modelo sobreajusta los datos.

Una vez finalizado el entrenamiento, se evalúa el rendimiento del modelo final sobre el conjunto test. En la Figura 6.14, se adjunta la matriz de confusión asociada, siendo la precisión correspondiente del **77,25 %**. En la Tabla 6.4, se recogen esta y otras medidas que pueden resultar útiles en la evaluación del modelo. En vista a estas medidas, se puede concluir que la precisión (*accuracy*) no es muy elevada. Este valor se debe a que el modelo no capta correctamente la ausencia de la enfermedad, siendo los indicadores asociados a su detección más prometedores.

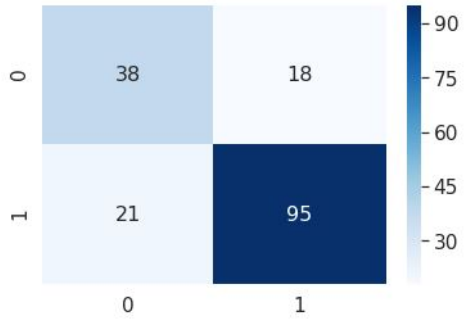


Figura 6.14: Retinopatía diabética - Matriz de confusión asociada al modelo básico.

Accuracy	0.77
Sensibilidad	0.82
Especificidad	0.68
Precisión	0.84
F1 score	0.83

Cuadro 6.4: Retinopatía diabética - Resultados de las métricas de evaluación del modelo básico.

6.2.2. Modelo básico aplicando la técnica *Data augmentation*

Debido al notable impacto de la aplicación de la técnica *Data augmentation* en la detección del edema macular, se opta por investigar su impacto sobre el modelo básico del apartado anterior, con expectativas de que la precisión del modelo se vea incrementada. Esta hipótesis se basa en el hecho de que el modelo básico sobreajusta los datos, tal y como se podía observar en la gráfica relativa a la evolución de la función de pérdida.

Respecto al modelo anterior, simplemente se modifica el preprocesamiento empleado, incorporando la aplicación de esta técnica. En las Figuras 6.15 y 6.16, se puede observar la evolución de la función de pérdida y de la precisión durante el entrenamiento de la red.

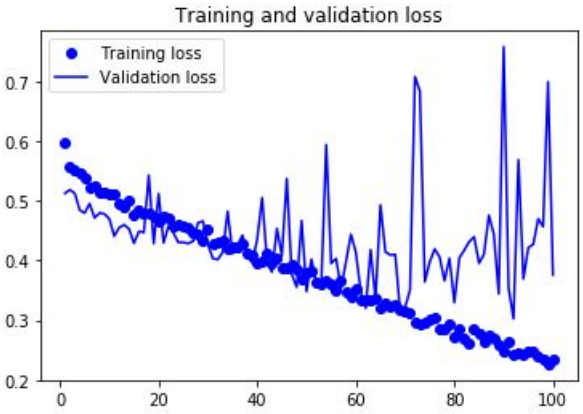


Figura 6.15: Retinopatía diabética - Evolución de la función de pérdida del modelo básico utilizando *Data augmentation*.

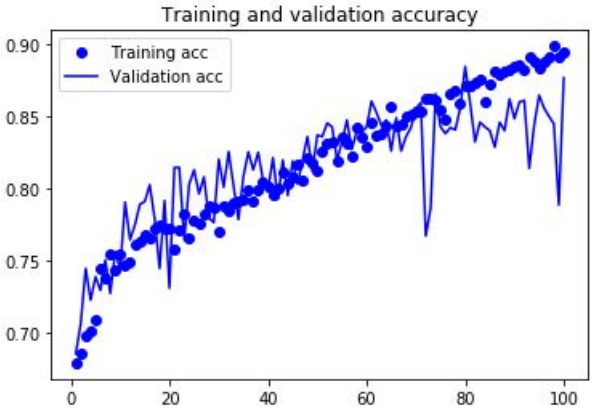


Figura 6.16: Retinopatía diabética - Evolución de la precisión del modelo básico utilizando *Data augmentation*.

En el caso de la función de pérdida, se puede percibir cómo disminuye mucho más lentamente a lo largo del proceso de aprendizaje, siendo su trayectoria prácticamente lineal, siendo su valor mínimo obtenido próximo a 0,2.

En cuanto a la gráfica relativa a la precisión, se puede observar cómo esta va incrementando a lo largo del entrenamiento del modelo. Cabe destacar, cómo la evolución relativa al conjunto test sigue una trayectoria muy similar, siendo un claro indicio de que esta técnica ha conseguido paliar en gran medida el sobreajuste.

Con el objetivo de proporcionar una evaluación más detallada del modelo entrenado, se adjunta, en la Figura 6.17, la matriz de confusión correspondiente. En comparación con la matriz relativa al modelo básico, a primera vista se observa como el número de ejemplos bien clasificados ha aumentado notoriamente, en ambas categorías.

En la Tabla 6.5, se plasman las métricas del rendimiento del modelo, pudiendo concluir que el rendimiento del modelo ha mejorado, ya que todas las métricas han aumentado

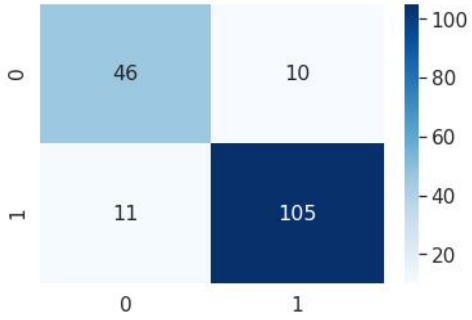


Figura 6.17: Retinopatía diabética - Matriz de confusión asociada al modelo básico utilizando *Data augmentation*.

su valor. En cuanto al valor de la sensibilidad se observa que es mayor que el relativo a la especificidad, por lo que el modelo detecta mejor la enfermedad, que su ausencia. Este resultado es satisfactorio, ya que en pruebas diagnóstico el costo relativo a un falso negativo es mucho mayor.

Accuracy	0.88
Sensibilidad	0.91
Especificidad	0.82
Precisión	0.91
F1 score	0.91

Cuadro 6.5: Retinopatía diabética - Resultados de las métricas de evaluación del modelo básico utilizando *Data augmentation*.

Se puede concluir que la mejora es sustancial, ya que la precisión se ha visto aumentada en algo más de un 10 %, alcanzando el **87,86 %**, lo que reafirma la importancia de realizar un correcto preprocesamiento que permita sacar el mayor valor posible a los datos disponibles. Por lo tanto, se considera que este modelo es una buena opción.

6.2.3. Modelos más complejos aplicando la técnica *Data augmentation*

Otra técnica, comúnmente utilizada, para intentar mejorar la precisión de un modelo, es la utilización de una estructura de red neuronal más compleja, la cual podría permitir una mayor extracción de características.

En esta sección, se exploran diferentes modelos, incrementando gradualmente la complejidad del modelo básico, siendo el más complejo el mostrado en la Figura 6.8. No obstante, ninguno de estas estructuras proporciona mejores resultados que los obtenidos utilizando el modelo básico con la técnica *data augmentation*, considerándose no satisfactorios.

A continuación, se exponen brevemente la complejidad añadida en cada una de las estructuras:

- Aumento del número de filtros en la última capa convolución del modelo básico, estableciéndolo a 256. El proceso de aprendizaje de esta estructura es muy similar al obtenido con el modelo básico aplicando *data augmentation*, pudiendo concluir que el aumento del número de filtros no ha sido muy influyente. La precisión del modelo sobre el conjunto test es **81,22 %**, aunque en las iteraciones anteriores se alcanzaban precisiones mayores.

- Estructura compuesta por tres bloques que engloban las operaciones convolución y *max pooling*, pero utilizando dos capas convolucionales en cada bloque. La precisión final del modelo, evaluada sobre el conjunto test, es **86,45 %**.

- Estructura compuesta por cuatro bloques que engloban las operaciones convolución y *max pooling*, utilizando dos capas convolucionales en cada bloque. Esta se puede visualizar en la Figura 6.8. La precisión final del modelo es **67,57 %**, valor muy por debajo incluso del modelo básico sin utilizar *data augmentation*. Este pésimo resultado nos indica que el modelo utilizado es demasiado complejo, posiblemente aprende características demasiado específicas, las cuales no son útiles a la hora de clasificar.

Se concluye que la utilización de un modelo más complejo, no aporta mejoras respecto al modelo anterior, descartando, por consiguiente, su utilización.

6.2.4. Modelo inicial realizando el preprocesamiento de forma manual

Tal y como se comentó en el preprocesamiento relativo a esta tarea de clasificación, se pretende abordar la realización manual del procesamiento automático efectuado por el generador, llevando a cabo una comparación de los resultados con los relativos a la utilización del generador.

Tras la aplicación de este preprocesamiento manual, explicado de forma detenida en el capítulo 4, los datos de entrada del modelo son:

- Imágenes convertidas a formato *Numpy*, haciendo referencia a la codificación *RGB* de las imágenes iniciales.

- Un array *Numpy* que especifica las etiquetas de las imágenes.

En cuanto a la estructura de la red neuronal, se elige la del modelo básico. La configuración del proceso de aprendizaje es la misma que en la utilización del generador. No obstante, a la hora de entrenar el modelo, es necesario hacer uso de la función *fit*, debido al formato de los datos de entrada.

En las Figuras 6.18 y 6.19, se puede observar, respectivamente, la evolución de la función de pérdida y, de la precisión a lo largo del entrenamiento del modelo. A pesar de que se esperaría un proceso de aprendizaje similar al obtenido utilizando un generador, se pueden percibir algunas diferencias en las gráficas (Figuras 6.12 y 6.13). En este caso, la minimización de la función de pérdida no converge tan rápidamente a 0 y, por consiguiente, el aumento de la precisión es más progresivo, aunque finalmente la precisión en el conjunto de entrenamiento sea del 100 %. En cuanto a la precisión evaluada en el conjunto test, se puede observar cómo no se ve incrementada al igual que en el conjunto de entrenamiento, manteniéndose mucho menor, coincidiendo esta característica con la utilización del generador. Igualmente, parece que la precisión disminuye levemente a partir de la iteración 60, ya que en iteraciones anteriores se obtuvo más del 80 % de la precisión, mientras que la precisión final es de **74,99 %**. Comparando este resultado, con el obtenido utilizando el generador, se puede concluir que la diferencia no es excesivamente significativa, pudiendo deberse la variación a la aleatoriedad del proceso de aprendizaje.

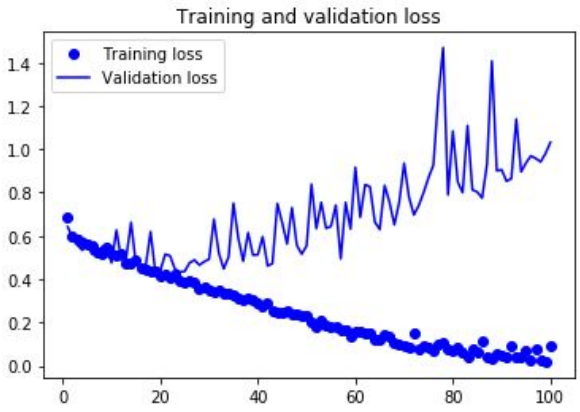


Figura 6.18: Retinopatía diabética - Evolución de la función de pérdida del modelo básico utilizando un preprocesamiento manual.

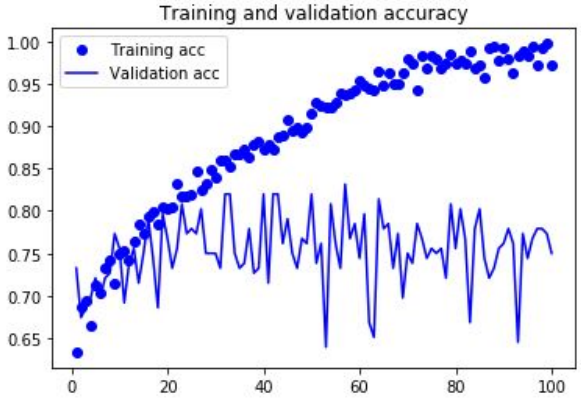


Figura 6.19: Retinopatía diabética - Evolución de la precisión del modelo básico utilizando un preprocesamiento manual.

Con el objetivo de indagar en el rendimiento del modelo, se adjunta en la Figura 6.20 la matriz de confusión y, en la Tabla 6.6, una tabla resumen con diferentes métricas de evaluación.

En primera instancia, se puede observar, en la matriz de confusión, cómo la mayoría de las instancias han sido predichas en la categoría relativa a la presencia de la enfermedad. Esto conduce a que, de la totalidad de las imágenes correspondientes a la ausencia de la enfermedad, más de la mitad estén mal clasificadas. Se puede decir, que este modelo tiene una ligera tendencia a sobrecategorizar la presencia de la enfermedad.

Explorando las métricas de evaluación, se pueden corroborar las conclusiones anteriores. El alto porcentaje de sensibilidad hace referencia a la capacidad del modelo para detectar

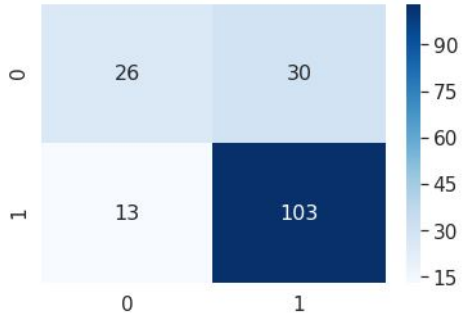


Figura 6.20: Retinopatía diabética - Matriz de confusión asociada al modelo básico utilizando un preprocesamiento manual.

Accuracy	0.75
Sensibilidad	0.89
Especificidad	0.46
Precisión	0.77
F1 score	0.83

Cuadro 6.6: Retinopatía diabética - Resultados de las métricas de evaluación del modelo básico utilizando un preprocesamiento manual.

la enfermedad, mientras que el bajo porcentaje de especificidad, a su mala capacidad para detectar su ausencia, siendo peor que el aleatorio. El valor de relativo a la precisión es menor debido a aquellos ejemplos categorizados erróneamente en la enfermedad.

En conclusión, a pesar de que la precisión sea aproximadamente similar, se ha podido observar como tanto el proceso de aprendizaje como el modelo obtenido son bastante diferentes, aunque podría deberse a la aleatoriedad del aprendizaje. Al margen, cabe destacar la reducción en el tiempo de ejecución respecto a la utilización de un generador.

Haciendo uso de este preprocesamiento, también se ha explorado la utilización de un modelo más complejo (Figura 6.8). No obstante, los resultados obtenidos no se consideran de suficiente interés como para aparecer reflejados en la memoria, no aumentando la precisión del modelo (75 %), al igual que ocurría realizando el preprocesamiento automáticamente. Se reafirma que los modelos complejos no producen resultados satisfactorios en la detección de esta enfermedad, siendo preferible la utilización de un modelo sencillo.

6.2.5. Modelo básico utilizando una capa *Dropout* realizando el preprocesamiento de forma manual

En esta sección, se pretende evaluar el impacto de la utilización de una capa *Dropout*, mediante su agregación al modelo básico expuesto en el apartado anterior.

Esta tipología de capa es considerada una técnica de regularización, que permite reducir el sobreajuste de la red neuronal. Su funcionalidad consiste en desactivar aleatoriamente un número determinado de neuronas. De este modo, en cada iteración se desactivan diferentes neuronas, no teniéndose en cuenta ni en la propagación hacia delante ni hacia atrás y, por consiguiente, no actualizándose los pesos correspondientes.

Este método se sustenta en el hecho de que, habitualmente, durante la fase de entrenamiento, las neuronas próximas tienden a adaptarse entre sí, apoyándose entre ellas de cara a ajustarse mejor al conjunto de entrenamiento, dependiendo unas de otras. El método *Dropout* intenta evitar que unas neuronas dependan de otras, obligándolas a aprender a proporcionar una respuesta válida incluso en la ausencia de sus neuronas vecinas.

En la Figura 6.21 se adjunta la estructura de la red convolucional empleada, en la cual se integra la capa *Dropout* tras la capa *Flatten*, desactivándose el 50 % de las neuronas tras el aplanamiento de los datos.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 427, 283, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 213, 141, 32)	0
conv2d_2 (Conv2D)	(None, 211, 139, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 105, 69, 64)	0
conv2d_3 (Conv2D)	(None, 103, 67, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 51, 33, 128)	0
conv2d_4 (Conv2D)	(None, 49, 31, 128)	147584
max_pooling2d_4 (MaxPooling2	(None, 24, 15, 128)	0
flatten_1 (Flatten)	(None, 46080)	0
dropout_1 (Dropout)	(None, 46080)	0
dense_1 (Dense)	(None, 1024)	47186944
dense_2 (Dense)	(None, 1)	1025

Total params: 47,428,801
Trainable params: 47,428,801
Non-trainable params: 0

Figura 6.21: Estructura básica agregando una capa *Dropout*.

En las Figuras 6.22 y 6.23, se adjuntan las gráficas relativas a la evolución de la función

de pérdida y de la precisión durante el proceso de aprendizaje, siendo la configuración del entrenamiento igual a la del apartado anterior. Comparando estas gráficas con las del anterior modelo, se puede observar cómo no existen grandes diferencias, siguiendo trayectorias similares; no consiguiendo evitar el sobreajuste. Aún así, se puede observar cómo la precisión alcanza valores algo mayores utilizando la capa *Dropout*.

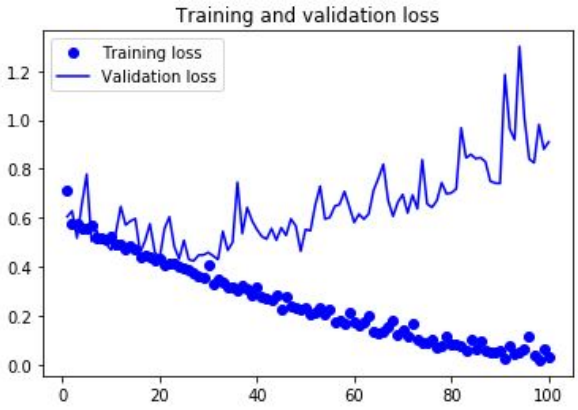


Figura 6.22: Retinopatía diabética - Evolución de la función de pérdida del modelo básico utilizando utilizando una capa *Dropout* realizando un preprocesamiento manual.

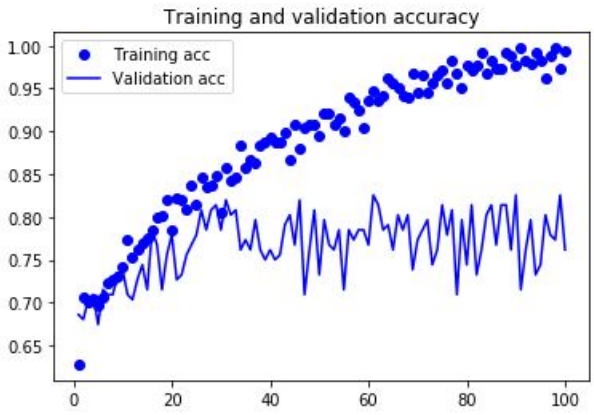


Figura 6.23: Retinopatía diabética - Evolución de la precisión del modelo básico utilizando una capa *Dropout* realizando un preprocesamiento manual.

Con el objetivo de profundizar algo más en el rendimiento del modelo, se adjunta, en la Figura 6.24, la matriz de confusión asociada. Respecto al anterior modelo, la predicción relativa a la ausencia de la enfermedad ha mejorado, aunque su precisión final es similar, alcanzando el **76,16 %**.

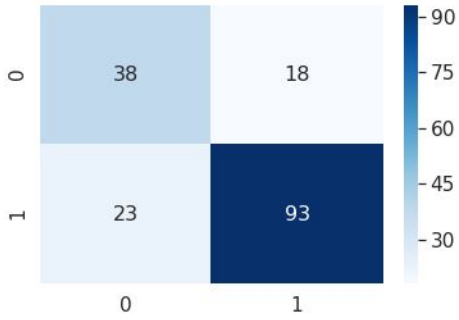


Figura 6.24: Retinopatía diabética - Matriz de confusión asociada al modelo básico utilizando una capa *Dropout* realizando un preprocesamiento manual.

En cuanto al resto de medidas utilizadas en la evaluación del modelo, la sensibilidad se ha visto reducida, mientras que la especificidad y precisión han mejorado, ya que el

anterior modelo tenía una ligera tendencia a detectar la enfermedad. Por esta razón, se proporciona unos resultados relativamente buenos, en la medida en que el conjunto de datos estuviese conformado por más muestras relativas a la presencia de la enfermedad.

Accuracy	0.76
Sensibilidad	0.80
Especificidad	0.68
Precisión	0.84
F1 score	0.82

Cuadro 6.7: Retinopatía diabética - Resultados de las métricas de evaluación del modelo básico utilizando un preprocesamiento manual.

A pesar de que el impacto de la aplicación de esta técnica no ha sido muy notorio, no siendo lo suficientemente potente como para evitar o reducir el sobreajuste, se considera que la clasificación efectuada por este modelo es más lógica que la anterior.

6.2.6. Modelo inicial utilizando las imágenes recortadas

Realizando un breve análisis exploratorio de las imágenes de fondo de ojo disponibles, se puede observar cómo se visualiza sobre un fondo prácticamente negro. En el proceso de aprendizaje, las neuronas deben aprender que esta zona de la imagen no aporta información útil en la detección de la enfermedad.

En este apartado, se considera de interés valorar el impacto relativo al preprocesamiento consistente en recortar las imágenes, acotando la zona de la imagen que resulta de interés, el cual se expone detenidamente en el apartado 5.2.1.

En cuanto a la estructura de la red neuronal a emplear, se opta por hacer uso del modelo básico inicial, realizando el preprocesamiento de forma manual para llevar a cabo este análisis comparativo. Para ello, se varía respecto al modelo del apartado 6.2.4. únicamente las imágenes de entrada.

En las Figuras 6.25 y 6.26 se puede visualizar la evolución de la función de pérdida y de la precisión a lo largo del proceso de entrenamiento, siendo estas gráficas muy similares a las relativas a la utilización de las imágenes sin recortar, pudiéndose intuir que el impacto del recorte de las imágenes no es significativo.

Profundizando en la evaluación del modelo en el conjunto test, se adjunta, en la Figura 6.27, la matriz de confusión. Comparándola con la obtenida utilizando las imágenes originales, este modelo se ajusta mejor a la ausencia de la enfermedad, a pesar de que la precisión del modelo es inferior, **73,25 %**. En la Tabla 6.8, se pueden visualizar unas métricas acordes a los comentarios realizados, aumentando el valor relativo a la especificidad y disminuyendo la sensibilidad.

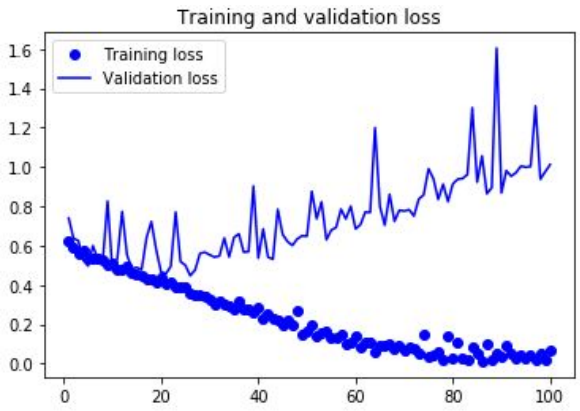


Figura 6.25: Retinopatía diabética - Evolución de la función de pérdida del modelo básico utilizando las imágenes recortadas.

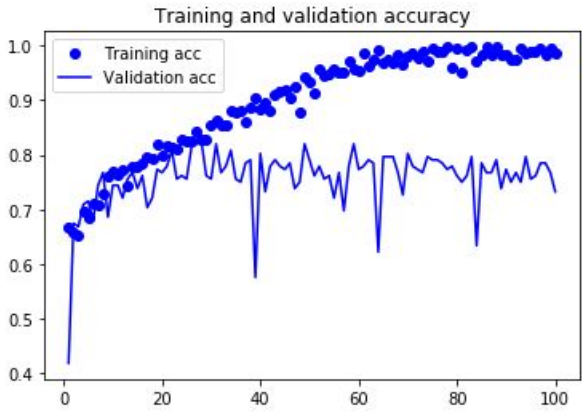


Figura 6.26: Retinopatía diabética - Evolución de la precisión del modelo básico utilizando las imágenes recortadas.

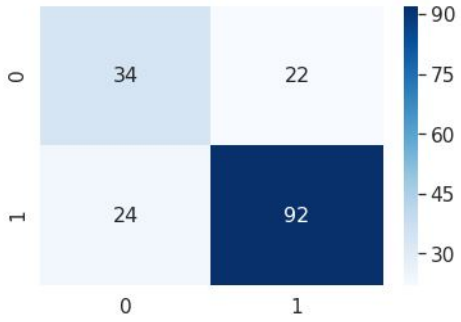


Figura 6.27: Retinopatía diabética - Matriz de confusión asociada al modelo básico utilizando una capa *Dropout* realizando un preprocesamiento manual.

Accuracy	0.73
Sensibilidad	0.79
Especificidad	0.61
Precisión	0.81
F1 score	0.80

Cuadro 6.8: Retinopatía diabética - Resultados de las métricas de evaluación del modelo básico utilizando las imágenes recortadas.

Haciendo uso de este preprocesamiento relativo al recorte de las imágenes, también se ha explorado la utilización de un modelo más complejo (Figura 6.8). Sin embargo, los resultados obtenidos no se consideran lo suficientemente interesantes, siendo la mejora de la precisión del modelo no significativa, ya que a pesar de un ligero aumento del

valor (75 %) respecto al modelo básico, constituye un ajuste peor que el de otros modelos expuestos. Por ello, no se presentan en detalle los resultados en esta memoria.

6.2.7. Modelo multientrada incluyendo la información relativa a la detección del edema macular

Tal y como se comentó anteriormente, el edema macular es un factor de riesgo en la detección de la retinopatía diabética. En esta sección, se pretende abordar la inclusión de esta información en el modelado, dando lugar a la necesidad de un modelo de entrada múltiple. De esta manera, el modelo no genera sus predicciones exclusivamente en base a una imagen de fondo de ojo, sino que utiliza la información relativa al diagnóstico del edema macular, que conformará la segunda entrada del modelo de entrada múltiple planteado.

Para crear esta estructura de red neuronal más compleja es necesario hacer uso de la clase *Model* de la API funcional, cuya utilización fue expuesta en el apartado 4.3.2.

En primer lugar, se decide hacer uso del procesamiento manual de las imágenes utilizado en los apartados 6.2.4, 6.2.5 y 6.2.6, con el objetivo de poder realizar un análisis comparativo respecto a los resultados obtenidos en estos modelos.

Con el objetivo de diseñar la estructura de la red neuronal, se opta por valorar los resultados obtenidos en los modelos citados en el párrafo anterior y, tomar como estructura base la relativa al modelo que proporciona el mejor ajuste, que es el *modelo básico utilizando una capa Dropout realizando el preprocesamiento de forma manual (apartado 6.2.5)*.

De este modo, se procede a modificar ligeramente esta estructura base, incluyendo la información relativa al modelo molecular. Estos nuevos datos de entrada se corresponden con la variable respuesta propia en las tareas relativas al diagnóstico del edema macular, siendo un array formato *Numpy* cuyos posibles valores son 0 ó 1, indicando la presencia o ausencia de esta enfermedad.

De acuerdo con estos datos, se considera que el punto de inclusión óptimo de la información en la estructura base es tras efectuar el aplanamiento de los mapas de características, es decir, concatenando este valor indicador del diagnóstico del edema macular con la salida de la capa *Flatten*, siendo esta concatenación la entrada de la capa totalmente conectada.

Cabe recordar como la especificación del diseño de la estructura será diferente debido a la utilización de la clase *Model*, siendo necesario satisfacer el enfoque establecido por esta clase. En la Figura 6.28 se adjunta la estructura de la red neuronal creada, donde se puede visualizar este enfoque basado en la definición de capas como tensores que toman como entrada tensores y que devuelven otros tensores, pudiendo considerarse funciones. La columna denominada *Connected to* hace referencia a los argumentos (tensores) que

recibe cada capa. También se puede observar la definición de las dos capas de entrada *input_1* e *input_2*, haciendo referencia *input_1* a la descomposición de la imagen bajo la codificación *RGB* e *input_2* al diagnóstico relativo al edema, estando conformado por un único valor. Asimismo, destaca la capa *concatenate_1*, siendo un tipo de capa propio de la clase *Model*, que se encarga de concatenar los tensores proporcionados, pudiendo observar como recibe el resultado de la capa *flatten_1* y los datos de entrada *input_2*.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 429, 285, 3)	0	
conv2d_1 (Conv2D)	(None, 427, 283, 32)	896	input_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 213, 141, 32)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 211, 139, 64)	18496	max_pooling2d_1[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 105, 69, 64)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 103, 67, 128)	73856	max_pooling2d_2[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 51, 33, 128)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 49, 31, 128)	147584	max_pooling2d_3[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 24, 15, 128)	0	conv2d_4[0][0]
flatten_1 (Flatten)	(None, 46080)	0	max_pooling2d_4[0][0]
dropout_1 (Dropout)	(None, 46080)	0	flatten_1[0][0]
input_2 (InputLayer)	(None, 1)	0	
concatenate_1 (Concatenate)	(None, 46081)	0	dropout_1[0][0] input_2[0][0]
dense_1 (Dense)	(None, 512)	23593984	concatenate_1[0][0]
dense_2 (Dense)	(None, 1)	513	dense_1[0][0]

Total params: 23,835,329
Trainable params: 23,835,329
Non-trainable params: 0

Figura 6.28: Retinopatía diabética - Estructura de red convolucional con entrada múltiple.

Con el objetivo de clarificar esta estructura, se presenta en la Figura 6.29 la representación de la estructura en forma de grafo, donde las conexiones entre las capas se pueden observar de forma directa y sencilla.

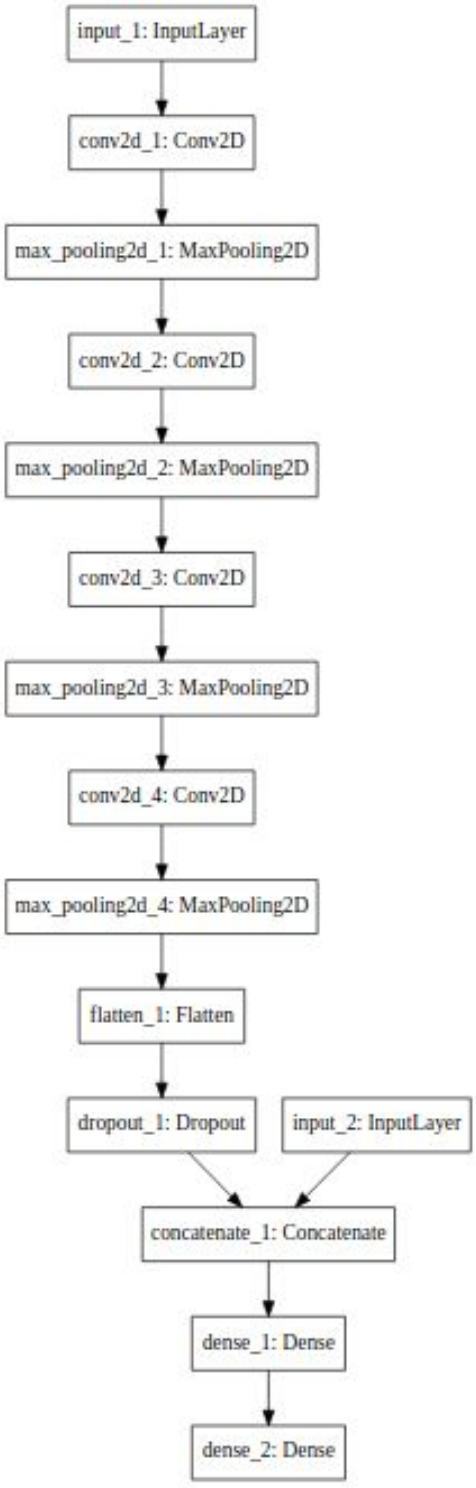


Figura 6.29: Retinopatía diabética - Representación en forma de grafo de la estructura del modelo de red convolucional con entrada múltiple.

Tras definir la estructura de la red, se procede a entrenar la red, haciendo uso de la misma configuración del proceso de aprendizaje. En las Figuras 6.30 y 6.31 se presentan las trayectorias relativas a la evolución de la función de pérdida y a la de la precisión. En la primera gráfica, la minimización de la función de pérdida es algo más pronunciada en las primeras iteraciones en comparación con el entrenamiento relativo al modelo base empleado. En la segunda gráfica, el aumento de la precisión también es más notorio en las primeras iteraciones. Sin embargo, en ambas gráficas se puede percibir cómo el modelo sobreajusta los datos. En cuanto a la trayectoria relativa al conjunto test, se puede observar cómo la precisión aumenta en las primeras iteraciones, manteniéndose posteriormente en torno al 85 %. En esta línea, la precisión final del modelo es **86,73 %**, que se ve aumentada en torno a un 10 % respecto al modelo base, es decir, sin la inclusión de la información relativa al edema.

En este punto, se considera de interés realizar una pequeña aclaración relativa a esta información. En el entrenamiento del modelo, se ha optado por hacer uso de la variable respuesta binaria relativa al edema macular. No obstante, de cara a realizar nuevas predicciones, esta información no estará disponible, por lo que se utiliza la predicción obtenida mediante el modelo del apartado 6.1.2, el cual es seleccionado para modelar el diagnóstico del edema macular.

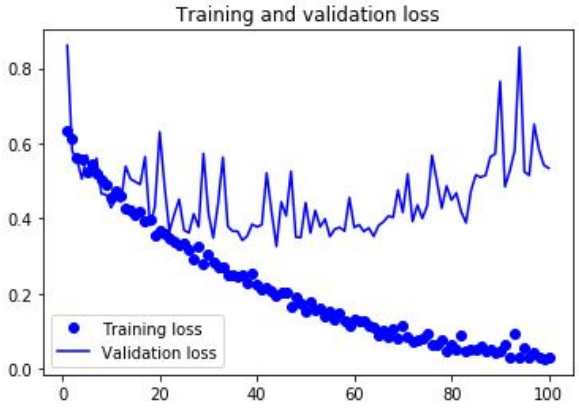


Figura 6.30: Retinopatía diabética - Evolución de la función de pérdida del modelo multientrada.

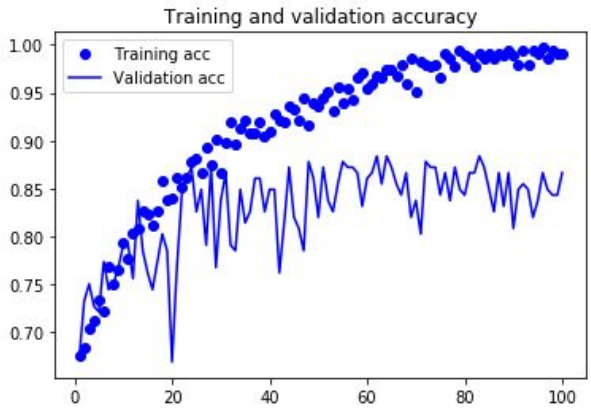


Figura 6.31: Retinopatía diabética - Evolución de la precisión del modelo multientrada.

Para evaluar de forma más detenida el rendimiento del modelo, se adjunta en la Figura 6.32 la matriz de confusión referente a este modelo. Respecto al modelo anterior, la capacidad de predicción ha incrementando, tanto en la detección de la presencia como de la ausencia de la enfermedad. En la Tabla 6.9 se adjuntan los valores de las métricas de evaluación correspondientes, pudiendo observarse una mejora de todas ellas. También se puede concluir que la capacidad de detección de la enfermedad es mayor (0.91), viéndose también incrementada la precisión del modelo (0.9).

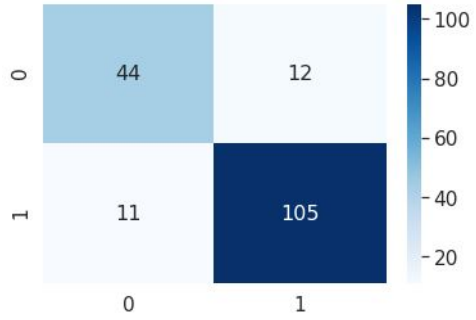


Figura 6.32: Retinopatía diabética - Matriz de confusión asociada al modelo básico utilizando una capa *Dropout* realizando un preprocesamiento manual.

Accuracy	0.87
Sensibilidad	0.91
Especificidad	0.79
Precisión	0.90
F1 score	0.90

Cuadro 6.9: Retinopatía diabética - Resultados de las métricas de evaluación del modelo básico utilizando las imágenes recortadas.

En vista a estos resultados, se puede concluir que la inclusión de la información relativa al edema macular contribuye notoriamente en la obtención de unos buenos resultados, corroborando su influencia como factor de riesgo.

Recapitulando los modelos evaluados en la modelización del diagnóstico de la retinopatía diabética, se observa que este modelo se encuentra entre los dos con mejor capacidad predictiva, junto con el del apartado 6.2.2 *Modelo básico aplicando la técnica Data augmentation*, donde la aplicación de esta técnica daba lugar a que generalizase mejor los datos.

Realizando una comparación entre estos dos modelos, los modelos proporcionados por ambos son bastantes similares, pudiendo seleccionar cualquiera de los dos mencionados. No obstante, el modelo básico aplicando la técnica Data augmentation proporciona un ligero mejor ajuste, por lo que se opta por seleccionar este modelo, siendo más simple que el expuesto en este apartado.

6.3. Modelización del diagnóstico de la conjuntivalización

El objetivo de esta sección radica en poner de manifiesto la potencia de las redes neuronales en las tareas relativas a clasificación de imágenes incluso cuando el conjunto

de datos es muy pequeño.

Bajo esta perspectiva, el planteamiento de esta sección varía respecto a las dos anteriores, ya que no se pretende mostrar en detalle el impacto de diferentes técnicas recomendadas, sino simplemente poner de manifiesto los buenos resultados que se pueden obtener haciendo uso de una estructura de red neuronal convolucional sencilla sobre este conjunto de datos descrito en el apartado 5.1.2.

Cabe destacar que al tratarse de un conjunto de datos tan pequeño, el entrenamiento de la red neuronal se efectúa mucho más rápido, pudiendo realizar varias pruebas en poco tiempo. A continuación, se adjunta en el Cuadro 6.10 una tabla que muestra la precisión (*accuracy*) obtenida para diferentes modelos.

Estructura de red convolucional	Precisión
Modelo básico (Fig. 6.1)	85.71 %
Modelo básico + Dropout (Fig. 6.21)	71.43 %
Modelo complejo (Fig. 6.8)	78.57 %
Modelo complejo + Dropout	64.29 %

Cuadro 6.10: Conjuntivalización - Resultados obtenidos con diferentes estructuras de red convolucional.

En vista a estos resultados, la utilización de estructuras de redes neuronales convolucionales más densas no proporciona mejores resultados, resultando ser modelos demasiado complejos para el conjunto de datos proporcionado. De este modo, el modelo básico es aquel que generaliza mejor los datos, proporcionando un mayor rendimiento sobre el conjunto test. Cabe destacar como debido al escaso número de muestras empleadas, estos valores pueden variar en función de las particiones de entrenamiento y test de los datos, así como del propio ajuste de la red.

La precisión del modelo básico seleccionado es del **85.71 %**, considerando este resultado satisfactorio. Profundizando en este resultado, en la descripción del conjunto de datos se expuso que el conjunto test estaba conformado por 14 imágenes, de modo que este porcentaje hace referencia la hecho de que sólo 2 imágenes se clasifican erróneamente. En la Figura 6.33 se adjunta la matriz de confusión relativa a este modelo básico, donde se puede observar la existencia de un falso negativo y un falso positivo.

Este resultado es una clara muestra de la potencia de las redes convoluciones y de su capacidad de aprender las características inherentes a las imágenes a pesar de que el conjunto de datos sea muy pequeño.

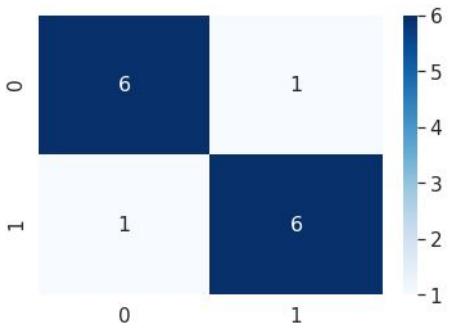


Figura 6.33: Conjuntivalización - Modelo básico.

6.4. Modelización del diagnóstico de la neovasculización

Esta sección, al igual que la anterior, pretende poner de manifiesto la potencia de las redes convolucionales, siguiendo el mismo planteamiento que el anteriormente mostrado. En la Tabla 6.11 se adjunta la precisión relativa a diferentes modelos.

Estructura de red convolucional	Precisión
Modelo básico (Fig. 6.1)	69.92 %
Modelo básico + Dropout (Fig. 6.21)	76.92 %
Modelo complejo (Fig. 6.8)	61.54 %

Cuadro 6.11: Neovasculización - Resultados obtenidos con diferentes estructuras de red convolucional.

En esta tarea de clasificación, la calidad de los modelos es inferior respecto a la anterior, lo cual puede deberse a que el conjunto de imágenes no consiga ser lo suficientemente representativo como para extraer las características inherentes a esta enfermedad.

Según los resultados proporcionados en la tabla 6.11, el modelo básico utilizando una capa *Dropout* es aquel que nos proporciona mejores resultados, con una precisión del **76.92 %**, valor que a priori puede no resultar muy satisfactorio. Sin embargo, el conjunto relativo a esta tarea de clasificación estaba conformado por 13 imágenes, indicando que 3 imágenes se clasifican erróneamente. Este desglose muestra que los resultados son más satisfactorios de lo que se puede intuir en función del valor numérico de la precisión.

En la Figura 6.34 se adjunta la matriz de confusión relativa a este modelo, en la que se puede observar cómo este modelo tiende a diagnosticar la patología, correspondiéndose las 3 imágenes etiquetadas erróneamente a falsos positivos. En consecuencia, la sensibilidad

de este modelo es del 100 % en el conjunto test, ya que identifica a la totalidad de los enfermos.

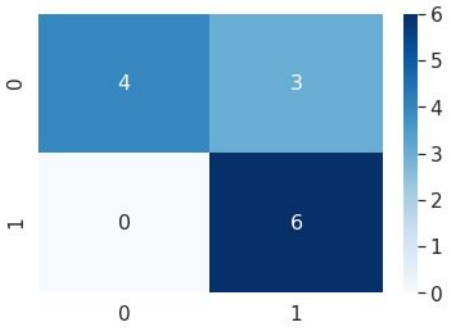


Figura 6.34: Conjuntivalización - Modelo básico.

De esta manera, en esta tarea de clasificación, también se puede vislumbrar el potencial de las redes neuronales convolucionales cuando el conjunto de datos es muy pequeño.

Capítulo 7

Visualización de las redes neuronales convolucionales

En el capítulo anterior se ha podido percibir el potencial de la aplicación de las redes neuronales convolucionales en los problemas de clasificación de imágenes, pero su funcionamiento interno no es fácilmente interpretable para el ser humano, siendo una tarea compleja determinar en qué se basan para tomar sus decisiones.

Esta carencia de interpretabilidad se debe principalmente a la utilización del método de propagación hacia atrás en el entrenamiento de la red, ya que no resulta sencillo conocer el nivel de importancia de cada pixel en la imagen. Por esta razón, habitualmente se indica que los modelos de redes neuronales son algoritmos de caja negra (aquellos que se estudian exclusivamente en función de sus entradas y salidas, siendo desconocido el funcionamiento interno del algoritmo).

Respondiendo a esta crítica, en la bibliografía se han intentado plantear varios enfoques que permitan comprender y visualizar las redes neuronales. En el caso concreto de las redes convolucionales, se han desarrollado diversas técnicas desde 2003, que permiten que el aprendizaje de las redes sea explicable visualmente [13].

En este TFG se exponen dos técnicas comúnmente utilizadas, que se aplican sobre el modelo explicado en el apartado 6.1.3, que fue el seleccionado para diagnosticar el edema macular diabético.

7.1. Visualización de las salidas intermedias

Esta técnica radica en mostrar las activaciones (salida de cualquier capa, mapas de características) de la red durante su propagación hacia delante. Esta visualización permite comprender cómo las sucesivas capas de la red neuronal van a ir transformando la imagen de entrada, es decir, cómo se descompone la imagen aplicando los diferentes filtros.

Hay que tener en cuenta que la dimensionalidad de un mapa de características hace referencia a la anchura, altura y número de canales, por lo que se puede considerar que la visualización más apropiada consiste en representar cada uno de los canales, reduciéndose a una representación en 2 dimensiones.

Para obtener las activaciones de la red convolucional, es necesario crear un nuevo modelo a partir de la clase *Model* proporcionada por *keras*, que posibilita la creación de un modelo de múltiples salidas. La entrada de este modelo es la propia entrada de la red neuronal que se pretende interpretar y la salida está conformada por las salidas de cada una de las capas del modelo, pudiendo acceder a ellas a través de la función *output*. Más concretamente, se utiliza un bucle para recorrer las diferentes capas e ir obteniendo la salida correspondiente mediante la función citada.

De este modo, proporcionando una imagen de entrada al modelo, este devolverá las salidas de cada una de las capas del modelo a interpretar.

En este apartado, se presenta la representación visual de todas las capas intermedias del modelo seleccionado para abordar el diagnóstico del edema macular. Cabe destacar cómo al tratarse de un modelo complejo, el número de salidas intermedias es elevado.

Tal y como se indicó, se obtienen las salidas de las capas intermedias para una imagen proporcionada, por lo que se opta por mostrar las salidas intermedias de una imagen que presenta edema macular. En la Figura 7.1 se puede visualizar la imagen tomada como ejemplo.

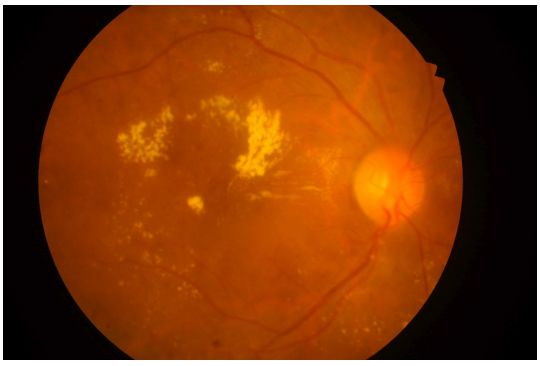


Figura 7.1: Imagen tomada como ejemplo de un fondo de ojo que padece edema macular.

Debido al elevado número de salidas intermedias del modelo, se presenta en el Apéndice A el desglose de la totalidad de las activaciones obtenidas en cada una de las capas de la estructura de la red, para la imagen mostrada en la Figura 7.1.

Las diferentes capas intermedias intentan resaltar diferentes partes de la imagen, pudiéndose apreciar este comportamiento en las primeras capas fácilmente, ya que a medida

que aumenta la profundidad de la red, las activaciones son más dispersas y localizadas, es decir, presentan una mayor complejidad.

En las Figuras 7.2 y 7.3 se adjuntan dos ejemplos de salidas intermedias que pretenden clarificar esta característica. En la primera de ellas, se puede percibir la imagen original resaltando ciertas zonas, que considera de interés. Por el contrario, en la Figura 7.3 ya no se percibe la imagen, ya que hace referencia a una zona localizada, que considera que aportará información en la tarea de clasificación.

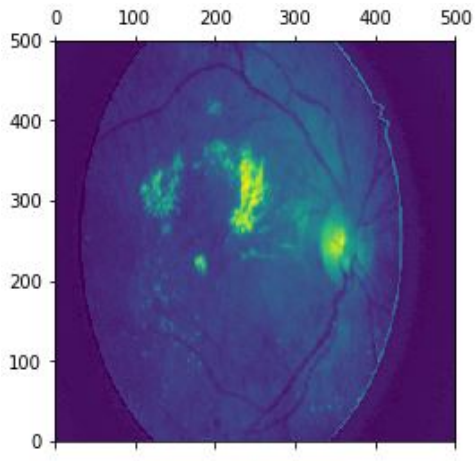


Figura 7.2: Ejemplo de una salida intermedia de la primera capa convolución.

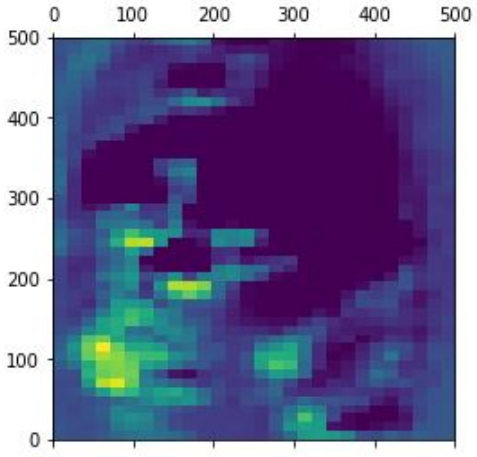


Figura 7.3: Ejemplo de una salida intermedia de la última capa convolución.

En esta línea, también se puede observar como la dispersión aumenta con la profundidad. Mientras en las primeras capas todos los filtros se activan, el número de filtros no activados en las siguientes capas se va incrementando. Esto se debe a que el patrón capturado es más específico, pudiendo no encontrarse en la imagen, no activándose ninguna neurona y proporcionando una salida negra.

Por último, cabe destacar como esta visualización permite mostrar el proceso de transformación de los datos de entrada que se efectúa, obviando la información irrelevante y magnificando la información útil en la toma de decisiones.

7.2. Visualización de los filtros de la red convolucional

La visualización de los filtros se considera de gran utilidad a la hora de interpretar la red neuronal, ya que va a permitir mostrar los patrones a los que hace referencia cada filtro, resultando ser una representación complementaria a la visualización anterior.

Estos filtros se obtienen mediante la aplicación del descenso del gradiente, intentando maximizar los valores de los mismos. Para ello, se define una función de pérdida que maximiza el valor de un filtro, a la cual se le aplica este método para ajustar estos valores de forma que maximicen su salida.

Al igual que en el apartado anterior, se adjunta la representación de la totalidad de los filtros aprendidos en el Apéndice 2, debido a su extensión.

Acorde a las visualizaciones de las capas intermedias, los filtros aumentan también su complejidad según la profundidad de la capa. Se puede observar como los filtros relativos a las primeras capas son sencillos, representando texturas simples como pueden ser colores o bordes. A medida que la profundidad de la red aumenta, los patrones contemplados adquieren complejidad, consecuencia directa de que estos filtros aprenden características más abstractas, las cuales serán representativas de las diferentes clases contempladas en la tarea de clasificación.

En las Figuras 7.4 y 7.5 se muestran las representaciones relativas a un filtro de la primera capa convolución y a un filtro de las capas finales respectivamente. El filtro relativo a la Figura 7.4 es sencillo, siendo fácilmente interpretable su referencia al color verde. En cambio el filtro relativo a la Figura 7.5 es complejo, no sabiendo identificar qué tipo de filtro se aplica exactamente.

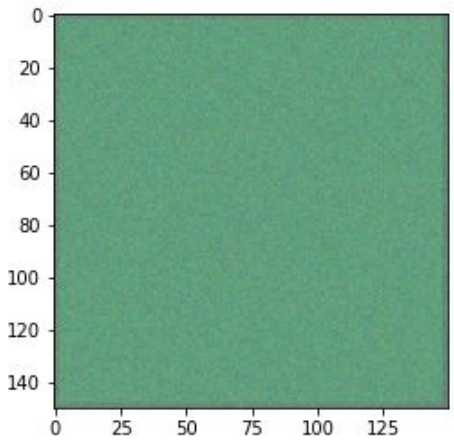


Figura 7.4: Filtro de la primera capa.

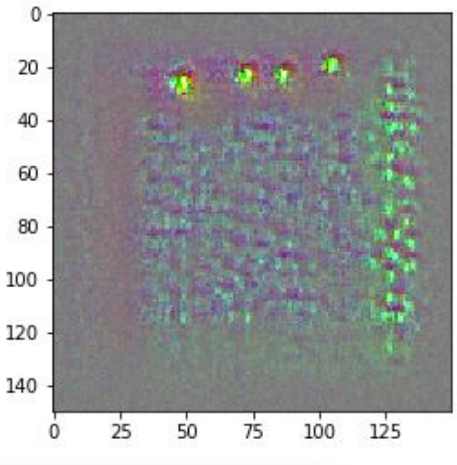


Figura 7.5: Filtro relativo a las últimas capas.

Debido al aumento de la complejidad de estos filtros más profundos, en la visualización anterior se observaba cómo algunos filtros no activaban ninguna de sus neuronas, haciendo referencia esos filtros a características propias de la otra clase. De este modo, dichos filtros son los responsables de las salidas de activación mostradas en el apartado anterior.

Capítulo 8

Aplicación

Aunque el TFG se centra en el estudio y la aplicación de redes neuronales convolucionales a la clasificación de imágenes estáticas, se realiza una pequeña aplicación web que permita mostrar los resultados obtenidos y, a su vez, que haga posible la utilización de los modelos relativos al diagnóstico del edema macular y de la retinopatía diabética a un usuario básico con conocimientos mínimos en Informática.

8.1. Análisis

8.1.1. Requisitos funcionales

Identificador	Nombre	Descripción
RF-01	Subir imagen	El sistema debe permitir subir imágenes
RF-02	Diagnosticar	El sistema debe diagnosticar el edema macular y la retinopatía diabética a partir de una imagen mediante los modelos almacenados
RF-03	Visualizar resultados	El sistema debe mostrar el diagnóstico obtenido dada una imagen

Cuadro 8.1: Tabla de requisitos funcionales.

8.1.2. Requisitos no funcionales

Identificador	Nombre	Descripción
RNF-01	Facilidad de uso	El sistema debe ser usable por un usuario básico con conocimientos mínimos de informática

RNF-02	Lenguaje de programación	El sistema debe desarrollarse en python 2.7
RNF-03	Formato de imágenes	El sistema debe soportar al menos los formatos .png y .jpg

Cuadro 8.2: Tabla de requisitos no funcionales.

8.1.3. Requisitos de información

Identificador	Nombre	Descripción
RINF-01	Modelos	El sistema debe almacenar los modelos relativos al diagnóstico del edema macular y de la retinopatía diabética
RINF-02	Sesiones	El sistema debe almacenar sesiones de usuario, las cuales incluirán la imagen a diagnosticar y las predicciones relativas a ambas enfermedades

Cuadro 8.3: Tabla de requisitos de información.

8.1.4. Matriz de trazabilidad

En este apartado se proporciona la matriz de trazabilidad, que permite mostrar de forma sencilla las relaciones existentes entre los requisitos funcionales y los de información.

	RINF-01	RINF-02
RF-01		⌟↑
RF-02	⌟↑	⌟↑
RF-03		⌟↑

Cuadro 8.4: Matriz Casos de uso - Requisitos de información.

8.1.5. Casos de uso

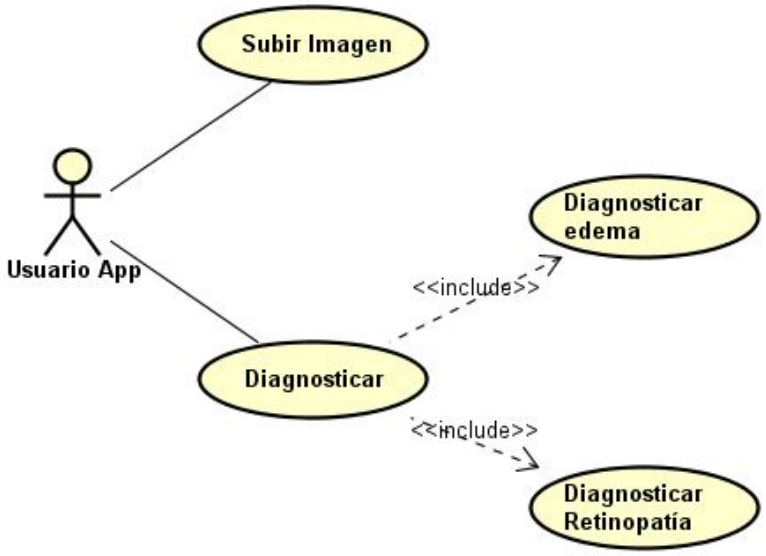


Figura 8.1: Diagrama de Casos de Uso.

CU-001

CU-001	Subir Imagen
Descripción	El sistema debe permitir al usuario subir imágenes almacenadas en su máquina local.
Actores	Usuario App
Precondición	La aplicación debe estar levantada en el servidor y el usuario debe haber accedido a través de la url a la página web.
Secuencia Normal	1 - El usuario selecciona la imagen que desea importar 2 - El sistema comprueba la extensión de la imagen 3 - El sistema almacena la imagen en el servicio
Postcondición	La imagen se encuentra almacenada en el servidor
Secuencia Alternativa	4.a - El sistema comprueba la extensión y detecta que se trata de un formato no contemplado 4.b - El sistema notifica al usuario que el formato de la imagen seleccionada es inválido

CU-002

CU-002	Diagnosticar
Descripción	El sistema debe permitir diagnosticar las diferentes enfermedades contempladas a partir de la imagen previamente proporcionada.
Actores	Usuario App
Precondición	La imagen a analizar debe haberse subido previamente al sistema.
Secuencia Normal	1 - El usuario selecciona la opción relativa a efectuar el diagnóstico 2 - Se realiza el caso de uso <Diagnosticar Edema> 3 - Se realiza el caso de uso <Diagnosticar Retinopatía> 4 - El sistema muestra por pantalla los resultados obtenidos
Postcondición	El sistema muestra por pantalla el diagnóstico predicho
Secuencia Alternativa	2.a - Se produce un error al procesar la imagen y se devuelve un mensaje de error 3.a - Se produce un error al procesar la imagen y se devuelve un mensaje de error

CU-003

CU-003	Diagnosticar Edema
Descripción	El sistema debe permitir diagnosticar el edema macular mediante el modelo almacenado relativo a esta tarea.
Actores	Usuario App
Precondición	La imagen a analizar debe haberse subido previamente al sistema y el caso de uso CU-002 ha sido iniciado.
Secuencia Normal	1 - El sistema predice el diagnóstico relativo al edema macular (presencia o ausencia) a partir de la imagen proporcionada 2 - El sistema almacena este resultado en el servidor
Postcondición	La predicción relativa al edema ha sido almacenada correctamente
Secuencia Alternativa	1.a - Se produce un error al procesar la imagen y se devuelve un mensaje de error

CU-004

CU-004	Diagnosticar Retinopatía
Descripción	El sistema debe permitir diagnosticar la retinopatía diabética mediante el modelo almacenado asociado a esta tarea.
Actores	Usuario App
Precondición	La imagen a analizar debe haberse subido previamente al sistema y el caso de uso CU-002 ha sido iniciado
Secuencia Normal	1 - El sistema predice el diagnóstico relativo a la retinopatía diabética (presencia o ausencia) a partir de la imagen proporcionada 2 - El sistema almacena este resultado en el servidor
Postcondición	La predicción relativa a la retinopatía ha sido almacenada correctamente
Secuencia Alternativa	1.a - Se produce un error al procesar la imagen y se devuelve un mensaje de error

8.2. Diseño

Tras el análisis efectuado en la sección anterior, en este apartado se presenta el diseño de la solución finalmente aportada, describiendo las tecnologías utilizadas y la arquitectura elegida. Asimismo se aporta el diagrama de clases y los de secuencia de cada caso de uso.

Cabe destacar cómo, acorde al requisito no funcional RNF-01, se ha considerado que la mejor solución posible radicaba en el desarrollo de una aplicación web, ya que este requisito hacia referencia a la facilidad de uso, indicando que debía ser fácilmente utilizable por un usuario básico con conocimientos mínimos de Informática. En esta línea, el desarrollo de una página web permitirá evitar que el usuario tenga que realizar una instalación de la aplicación, encargándose un administrador de sistemas de la misma. El usuario accederá a través de una URL en el navegador sin necesidad de instalar software relativo a la aplicación en su equipo.

8.2.1. Tecnologías utilizadas

En primer lugar, tal y como recoge el requisito RNF-02, el servidor web debía desarrollarse en *python* debido a que la creación de los modelos de redes neuronales convolucionales se abordó utilizando este lenguaje de programación. Tras una breve exploración de diferentes frameworks, se ha optado por utilizar *Flask* [2] ya que, además de ser uno de los frameworks más utilizados en el desarrollo de aplicaciones web en *python*, en diversos estudios comparativos se expone que su curva de aprendizaje es poco elevada, es decir, es de fácil aprendizaje. Esta característica resulta de gran interés debido a la restricción

temporal del presente TFG, teniendo en cuenta que previamente no se ha trabajado en proyectos de características similares.

Por su parte, *Flask* permite manejar sesiones, almacenando la información relativa a cada petición. No obstante, debido a la restricción del tamaño del almacenamiento en la parte cliente, es decir, en el navegador del usuario, se ha optado por utilizar *Redis* [6] para almacenar dichas sesiones en el servidor, sin límite, siendo un "motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes", presentando *Flask* soporte para su integración.

No obstante, hay que tener en cuenta que *Flask* suministra un servidor para el despliegue de la aplicación en su fase de desarrollo, no siendo recomendable utilizar el mismo en la fase de producción. Por ello, se opta por el servidor de despliegue *Gunicorn* [3], debido a su compatibilidad con el framework y a su mayor facilidad de integración, permitiendo atender múltiples peticiones de forma paralela a través de hilos.

Por último, cabe señalar que también se ha utilizado *JavaScript* para facilitar el desarrollo de la interfaz web.

8.2.2. Arquitectura

En este subapartado se exponen los diferentes patrones de diseño utilizados en la aplicación web, los cuales contribuyen a que el diseño sea más comprensible estandarizando el código.

Patrón MVC

En primer lugar, cabe destacar el uso del patrón Modelo-Vista-Controlador, que permite separar la lógica de negocio de la interfaz de usuario. De esta manera, el diseño de la aplicación se descompone en vistas, que permiten mostrar la información al usuario (es decir, presentar los datos), un controlador, que se encarga de gestionar las peticiones realizadas por el usuario y un modelo que representa los datos.

Patrón Singleton

La aplicación de este patrón asegura que sólo se cree una instancia de ciertas clases, de forma que estas instancias únicas sean compartidas por toda la aplicación, no creando instancias específicas para cada cliente. En esta aplicación concreta, se crea una única instancia relativa al modelo de edema macular y otra única relativa al modelo de retinopatía diabética.

Patrón Vista Compuesta

En el desarrollo de esta aplicación se contemplan dos vistas principales (una relativa a la página de inicio y otra relativa a mostrar los resultados) y una tercera vista que se utiliza

simplemente para informar al usuario de que la aplicación está procesando la petición, es decir, de que se está efectuando la predicción, siendo una vista más sencilla. En este contexto, se utiliza el patrón Vista Compuesta, con el objetivo de evitar dependencias entre las vistas, evitando que cambios en la vista secundaria afecten a la principal. De este modo, la vista principal contendrá a la tercera vista informativa.

Patrón Fachada

La utilización de este patrón se basa en la idea de evitar que el controlador llame directamente al modelo, siendo necesario que se acceda a él a través de la fachada. En este caso, el patrón permite desacoplar los modelos relativos al edema y a la retinopatía, reduciendo el número de dependencias. La utilización de este patrón contribuirá a que la aplicación sea fácilmente extensible.

8.2.3. Diagrama de clases

En esta sección se presenta el diagrama de clases, en el cual se pueden percibir los diferentes patrones de diseño descritos anteriormente.

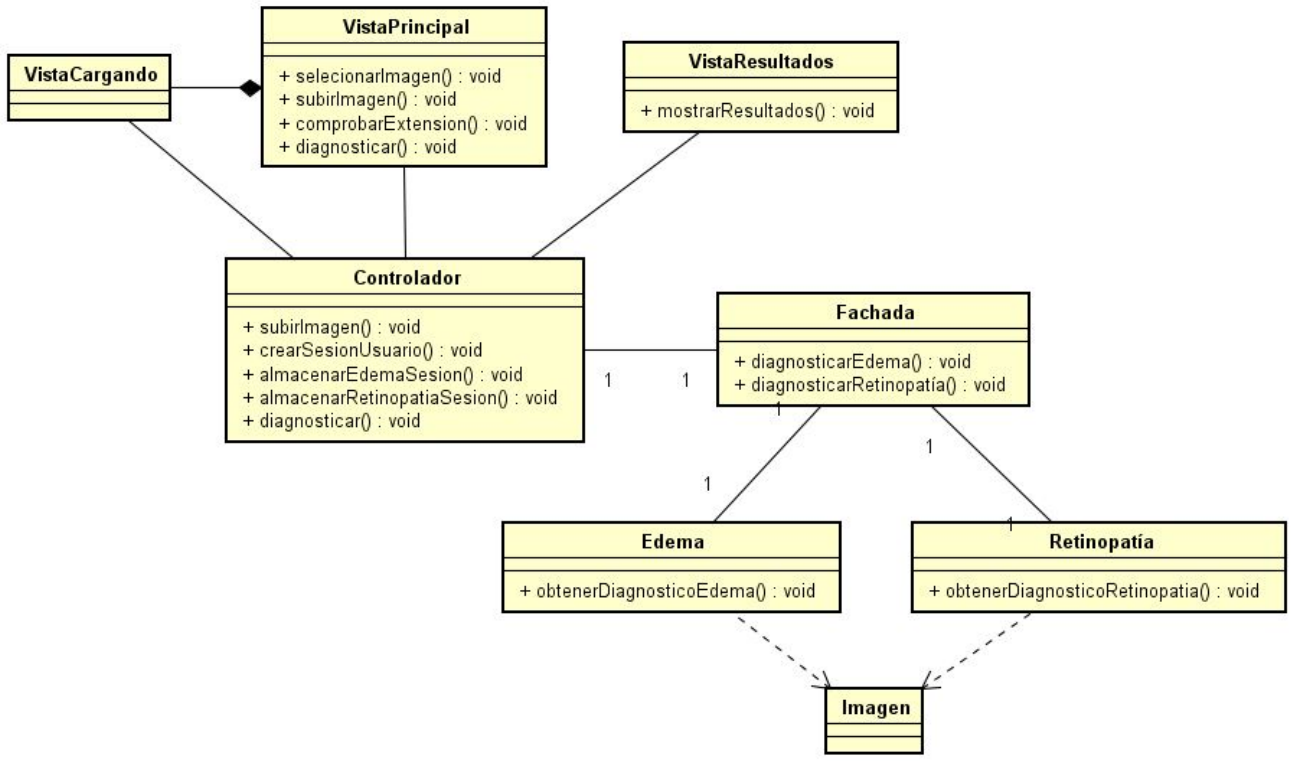


Figura 8.2: Diagrama de clases.

8.2.4. Diagramas de secuencia

En este apartado se presentan los diagramas de secuencias asociados a los casos de uso, con el objetivo de clarificar el funcionamiento de la aplicación a través del diseño.

CU-001

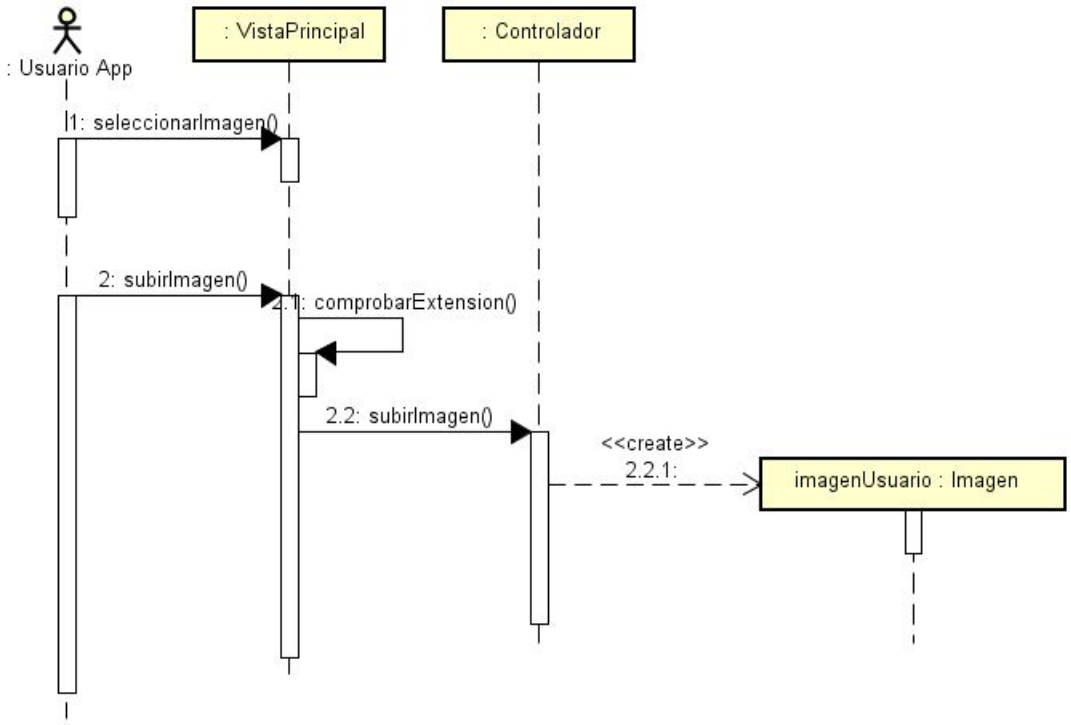


Figura 8.3: Diagrama de secuencia CU-001.

CU-002

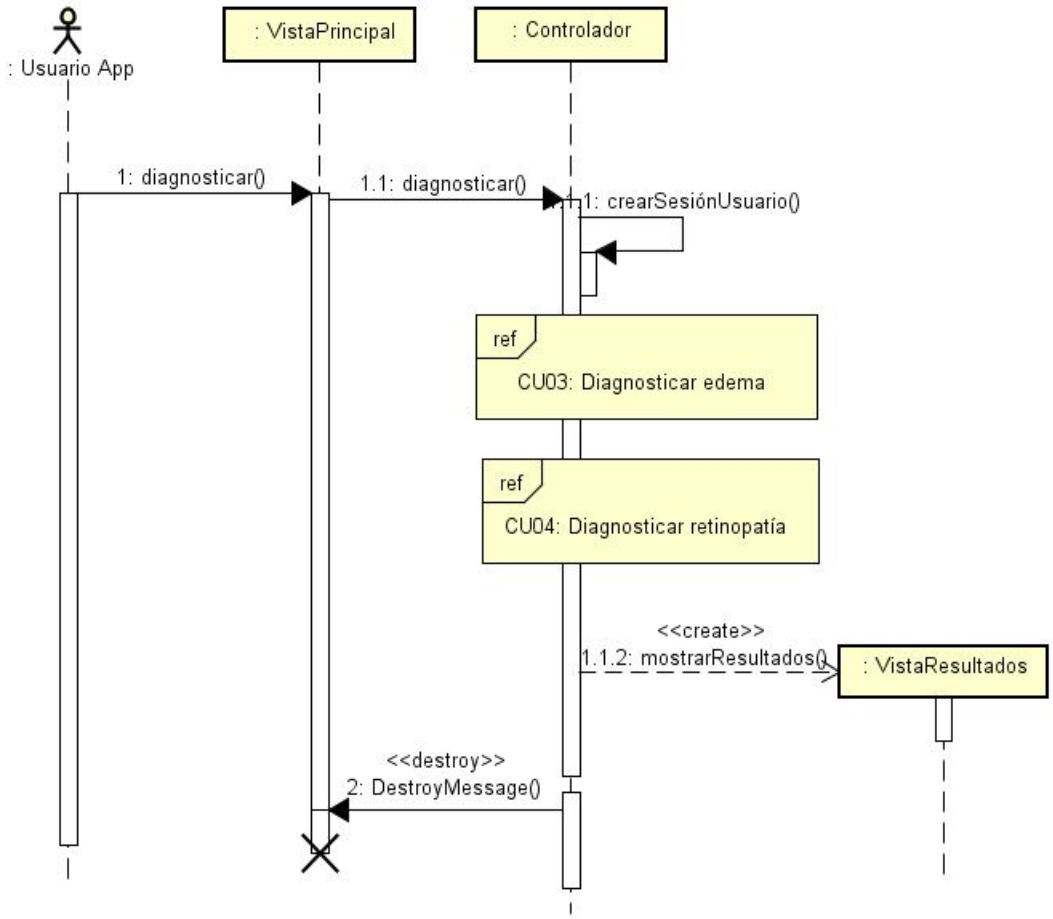


Figura 8.4: Diagrama de secuencia CU-002.

CU-003

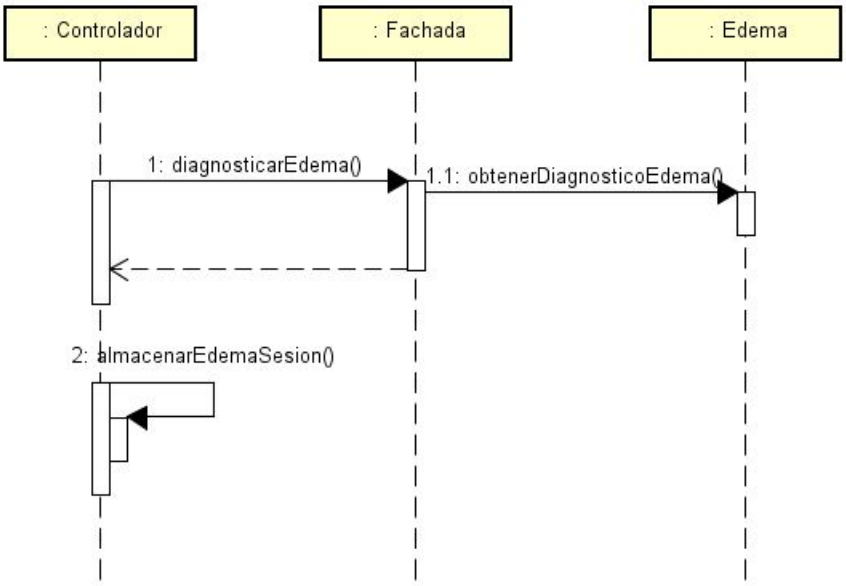


Figura 8.5: Diagrama de secuencia CU-003.

CU-004

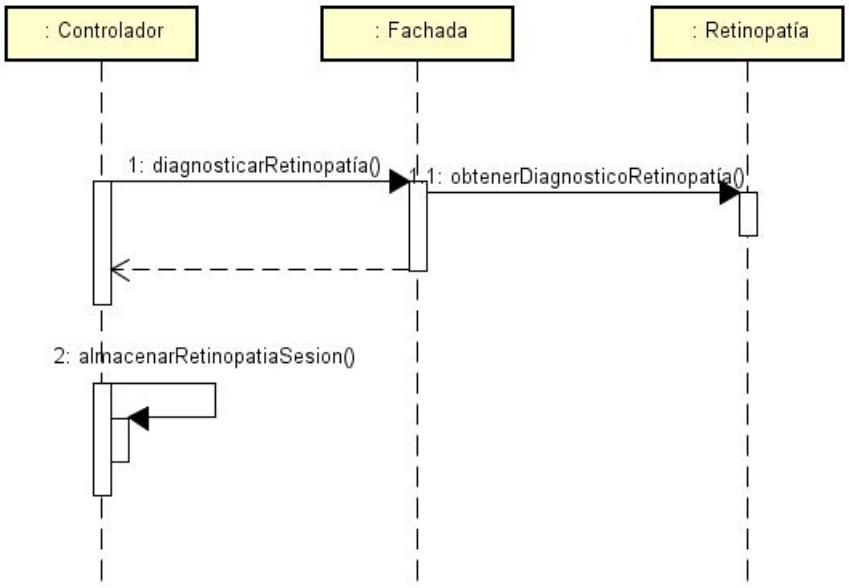


Figura 8.6: Diagrama de secuencia CU-004.

8.2.5. Gestión de la seguridad

En este TFG los aspectos relativos a seguridad no han sido abordados en gran profundidad, al encontrarse fuera de alcance debido a las restricciones temporales. No obstante, se han tomado dos decisiones al respecto:

- Comprobar la extensión de las imágenes
- Cifrar las sesiones de los usuarios

8.2.6. Configuración del servidor *Gunicorn*

Tal y como se expuso en el apartado 8.2.1, se opta por utilizar el servidor *Gunicorn* para desplegar la aplicación web, que permite atender múltiples peticiones de forma paralela, mediante el manejo de múltiples procesos e hilos. Con tal fin, *Gunicorn* crea un proceso maestro que se bifurcará en secundarios, denominados *workers*, que son los encargados de manejar las peticiones HTTP. El proceso maestro se encarga de que el número de *workers* sea igual al especificado en la configuración del servidor [22].

Otra forma de concurrencia contemplada en este servidor, se basa en el manejo de hilos, de forma que permite que cada *worker* tenga varios hilos asociados.

Por esta razón, entre las opciones de configuración del servidor es necesario especificar el número de *workers* y el número de *hilos*. Para tomar esta decisión se ha hecho uso de la documentación de Gunicorn [4], donde se recomienda usar un único *worker* y varios hilos, resultando más eficiente que la utilización de varios *workers*. Asimismo, se recomienda que el número de hilos especificado en la configuración debe estar condicionado por la cantidad de núcleos del servidor en el que se despliega la aplicación, según esta función:

$$N^{o}\ de\ hilos = 2 \times N^{o}\ de\ núcleos + 1$$

8.2.7. Utilización de contenedores Docker

En el diseño de esta aplicación web, se ha optado finalmente por utilizar Docker, siendo una plataforma software que permite empaquetar software en unidades estandarizadas denominadas contenedores, que incluyen todo lo requerido por la aplicación para su ejecución: bibliotecas, código, etc.

Gracias a estas unidades estandarizadas, Docker ofrece un sistema de ejecución de entornos parcialmente aislado. En este punto, se podría considerar que Docker ofrece los mismos beneficios que una máquina virtual. No obstante, a pesar de la similitud en sus beneficios, funcionan de manera totalmente diferente.

Con el objetivo de ilustrar esta diferencia, se adjuntan en las Figuras 8.3 y 8.4 un esquema de ambas arquitecturas.

En la figura relativa a los contenedores se puede visualizar cómo Docker actúa a modo de una capa intermedia entre una aplicación (contenida en Docker) y el sistema

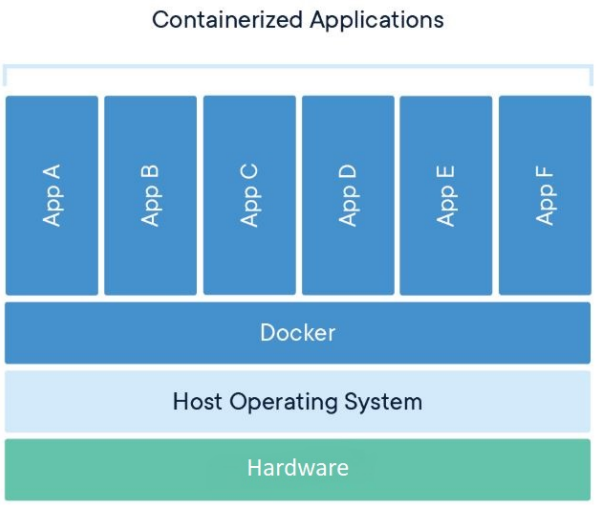


Figura 8.7: Estructura de la arquitectura utilizando contenedores.

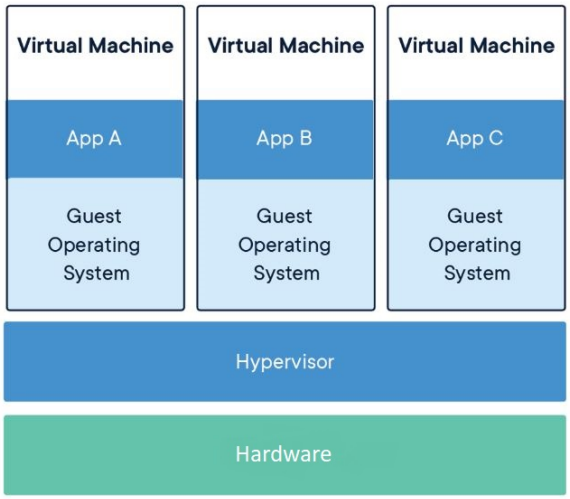


Figura 8.8: Estructura de la arquitectura utilizando máquinas virtuales.

operativo. Por el contrario, una máquina virtual es un sistema operativo que funciona de forma independiente sobre otro sistema operativo, es decir, esta tecnología permite compartir el hardware para que sea utilizado por varios sistemas operativos, tratándose de una abstracción del hardware, mientras que los contenedores son una abstracción del sistema operativo (aislando aplicaciones y no sistemas operativos).

Por consiguiente, la utilización de contenedores presenta múltiples ventajas, siendo la principal su mayor portabilidad y eficiencia. Gracias a esta portabilidad, sería suficiente con instalar Docker en un sistema operativo Unix para poder ejecutar la aplicación sin instalar las dependencias requeridas.

Igualmente cabe destacar que Docker dispone de un repositorio de imágenes oficial, definiéndose una imagen como una instancia de un contenedor, las cuales pueden ser descargadas e incorporadas en el propio contenedor. Por ejemplo, en el desarrollo de este proyecto se ha descargado la imagen oficial de Redis, lo cual ha ahorrado la instalación y configuración del mismo (siendo el detonante de la utilización de Docker).

Finalmente, de cara a levantar los contenedores, se ha hecho uso de la herramienta *Docker Compose*[1], la cual permite a través de un único comando (*docker-compose up*) crear e iniciar los servicios de la aplicación configurados previamente en un archivo.

Capítulo 9

Conclusiones

En primer lugar, cabe destacar cómo la elaboración de este trabajo de fin de grado ha brindado la oportunidad de tratar con imágenes, que ha permitido iniciarme en la disciplina de la visión computacional, siendo un tema escasamente tratado en la carrera. No obstante, diversas materias como *Minería de Datos*, *Técnicas de Aprendizaje Automático*, *Fundamentos de Inteligencia Artificial*, *Ingeniería del Conocimiento*, etc, han proporcionado unos conocimientos básicos para abordar esta problemática.

Sin embargo, estas materias no han sido las únicas que han contribuido en la elaboración de este trabajo, considerándose que este proyecto presenta un carácter integrador de numerosas asignaturas de diversa índole. De este modo, se han aplicado conocimientos relativos al análisis y diseño de software, planificación y gestión de proyectos, computación e inteligencia artificial entre otros. Cabe destacar cómo el estudio simultáneo del Grado en Estadística ha supuesto un gran aporte, siendo las lecciones aprendidas directamente aplicables en este trabajo gracias a su enfoque en el análisis de datos.

En particular, en este TFG se ha planteado la tarea de clasificación de imágenes estáticas mediante la utilización de redes convolucionales, puesto que es la técnica más usada para ello, de acuerdo con la bibliografía actual sobre este tema. A pesar de haber tratado en varias asignaturas diversas estructuras de redes neuronales, el estudio de las convolucionales ha resultado novedoso, permitiendo no sólo profundizar en su estructura, sino reforzar los conocimientos genéricos de redes neuronales anteriormente aprendidos.

En el plano de la programación, este proyecto me ha aportado una primera toma de contacto con las bibliotecas *keras* y *tensorflow*, ya que son de las más utilizadas en la construcción de modelos de aprendizaje profundo, por lo que se considera un gran aporte a nivel formativo.

Asimismo se quiere destacar la aproximación de dicho trabajo a un problema de aplicación real, que ha permitido abordar un reto actual, la detección de diferentes patologías a partir de imágenes oculares. Gracias a la contextualización de este estudio en el ámbito de la salud, y en vista a los resultados obtenidos, se pretende poner de manifiesto el

importante papel que puede ocupar la utilización de la tecnología en este campo, facilitando y mejorando los procesos de prevención, diagnóstico, tratamiento y monitorización de pacientes.

Tras este estudio, la aplicación de las redes neuronales se ha considerado exitosa, obteniendo resultados muy satisfactorios en la detección de las patologías contempladas. En cuanto a la predicción del diagnóstico del edema macular a partir de una imagen de fondo de ojo, se ha obtenido una precisión mayor al 90 %, y para la retinopatía diabética una precisión mayor del 87 % en su predicción. A pesar de ser este resultado algo inferior al obtenido en el diagnóstico del edema, se considera satisfactorio, superando la precisión obtenida por un estudio relativo a la misma problemática realizado por el IOBA, IMUVA, Hospital Clínico y INCYL, publicado recientemente en *el día de Valladolid* [14], aunque cabe aclarar cómo estos resultados están sesgados a la utilización de una base de imágenes ocular distinta.

En cuanto a la detección de la presencia de neovasos y conjuntivalización sobre un pequeño conjunto de imágenes, ha permitido tomar consciencia del potencial de esta técnica, proporcionando buenos resultados, incluso cuando se realiza un entrenamiento con pocas instancias.

Durante la fase de modelado de estas tareas de clasificación, cabe destacar la apreciación relativa a la importancia de un correcto preprocesamiento de los datos, y a su influencia en los resultados proporcionados por el modelo. Asimismo, cabe remarcar cómo un modelo más complejo no va a suponer siempre un mejor resultado, teniendo que tener en cuenta que no se debe añadir más complejidad de la necesaria.

En este contexto, ha resultado sorprendente la alta precisión con el que una red convolucional correctamente diseñada y entrenada puede realizar inferencias, aunque el desarrollador no disponga de un conocimiento específico en la materia de objeto estudio.

Aunque el groso de este TFG se haya centrado en las redes convolucionales, se quiere destacar las lecciones aprendidas tras la realización de la versión web. A pesar de tratarse de una aplicación sencilla, ha permitido aprender los conocimientos básicos del framework *Flask*, así como hacer frente a su integración con otros servicios, al tratarse finalmente de una web de fácil uso para cualquier usuario con conocimientos básicos en Informática.

Por último, es interesante destacar la utilización de contenedores Docker, que favorecen la portabilidad, la facilidad de instalación y su despliegue.

Obviamente, tras este proyecto quedan abiertas muchas líneas de trabajo, las cuales no se han podido abordar debido a la restricción temporal. Entre ellas se encuentran:

- Abordar un preprocesamiento más exhaustivo de las imágenes de fondo de ojo.

- Relativo al modelo de la retinopatía diabética, explorar la posibilidad de un modelo que incluya la técnica de preprocesamiento *data augmentation*, junto con la inclusión de la información relativa al edema macular.
- Investigar otras alternativas posibles en la clasificación de imágenes, con el objetivo de obtener un análisis comparativo de diferentes metodologías.

Apéndice A

Ejemplo ilustrativo de la visualización de las capas intermedias de una red convolucional

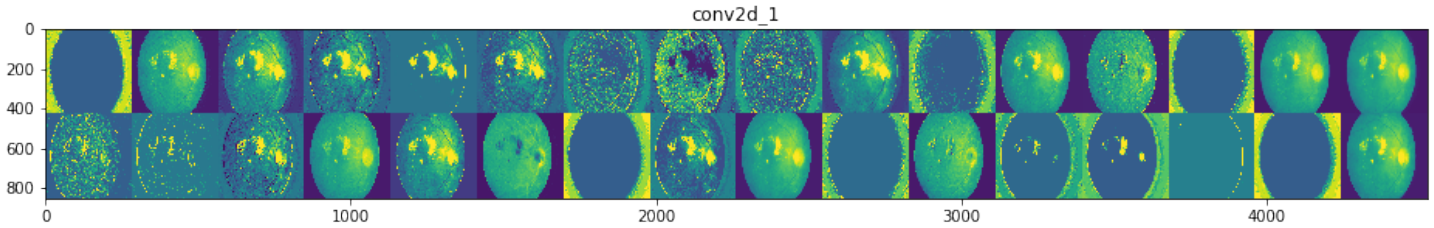


Figura A.1: Salida de la capa conv2d_1 para la imagen ejemplo.

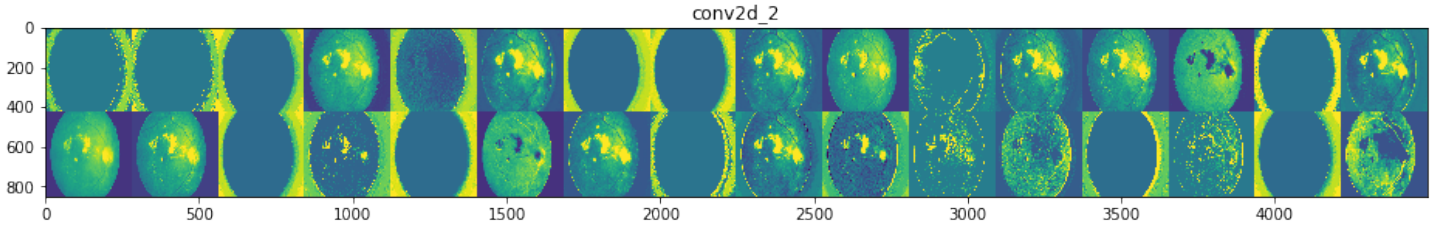


Figura A.2: Salida de la capa conv2d_2 para la imagen ejemplo.

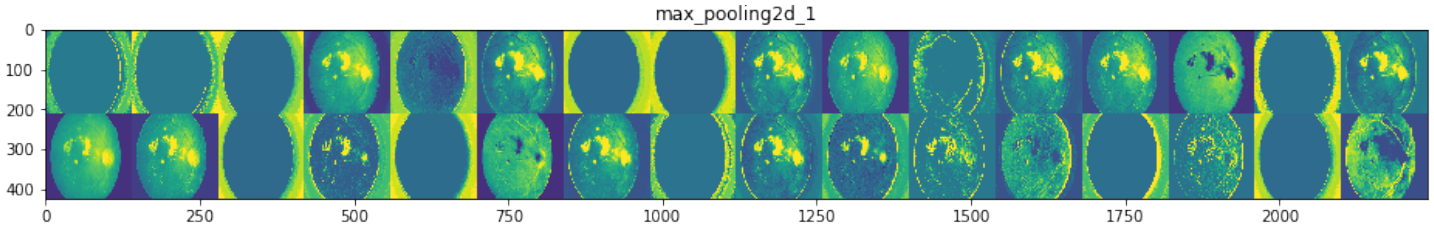


Figura A.3: Salida de la capa maxpooling2d_1 para la imagen ejemplo.

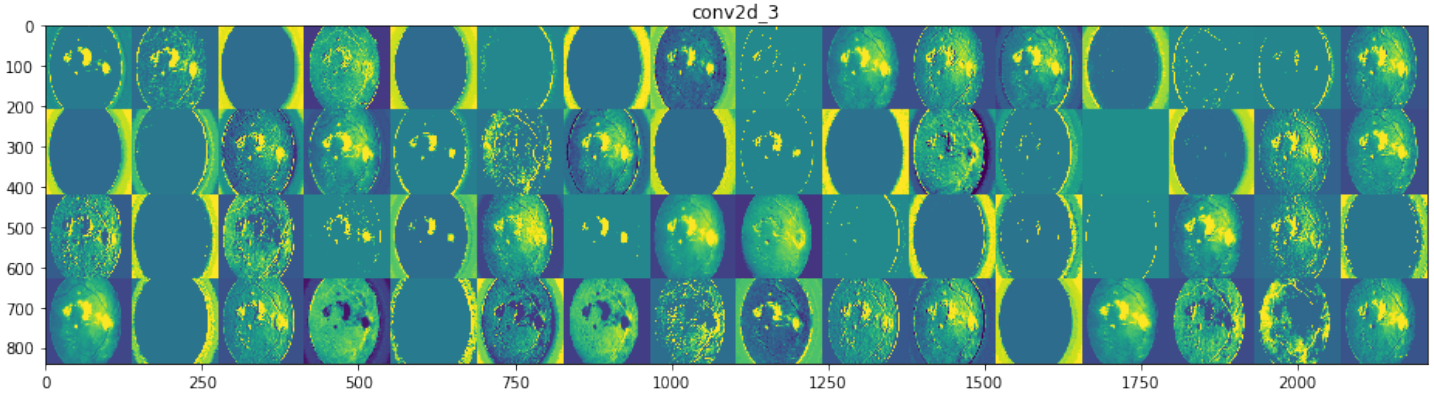


Figura A.4: Salida de la capa conv2d_3 para la imagen ejemplo.

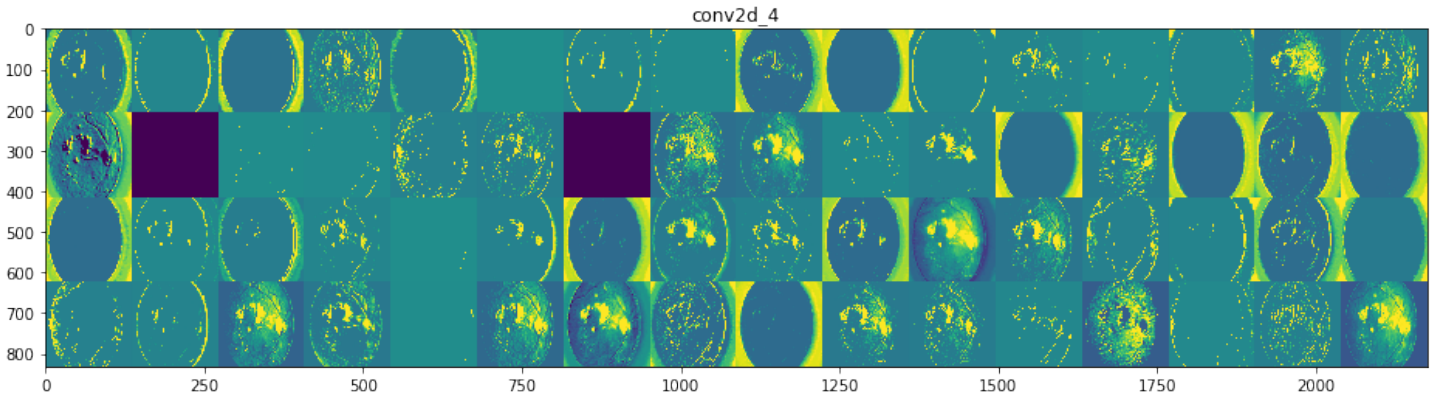


Figura A.5: Salida de la capa conv2d_4 para la imagen ejemplo.

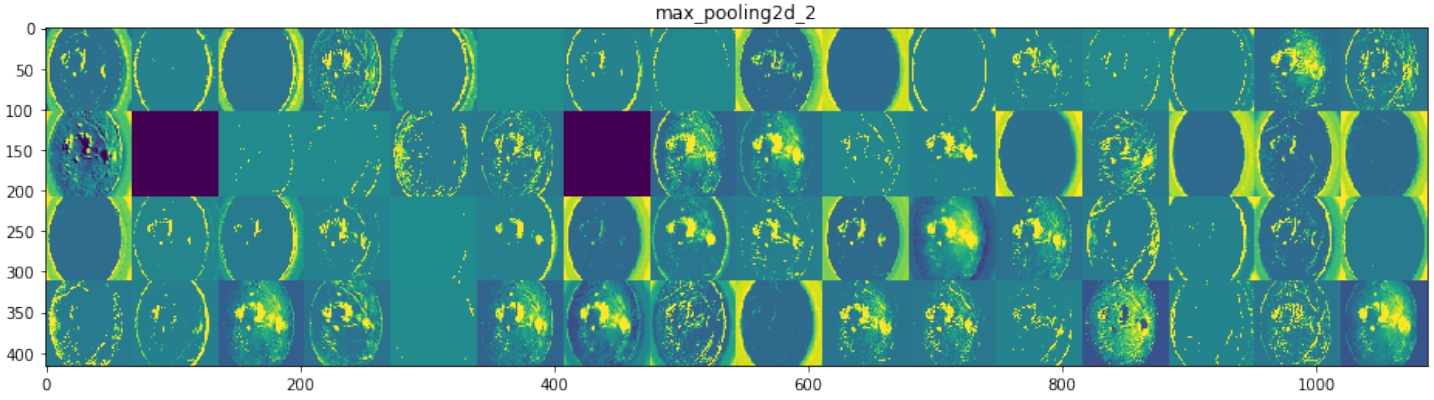


Figura A.6: Salida de la capa maxpooling2d_2 para la imagen ejemplo.

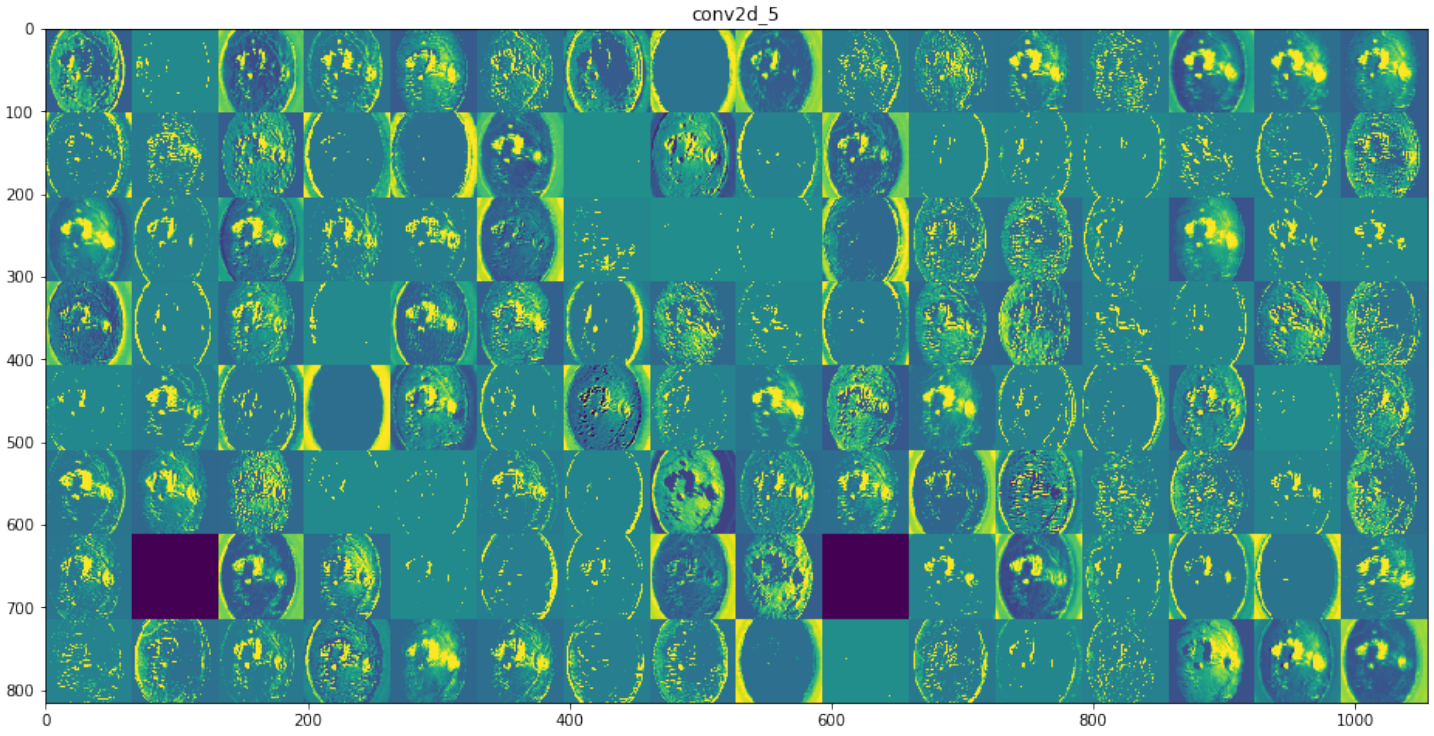


Figura A.7: Salida de la capa conv2d_5 para la imagen ejemplo.

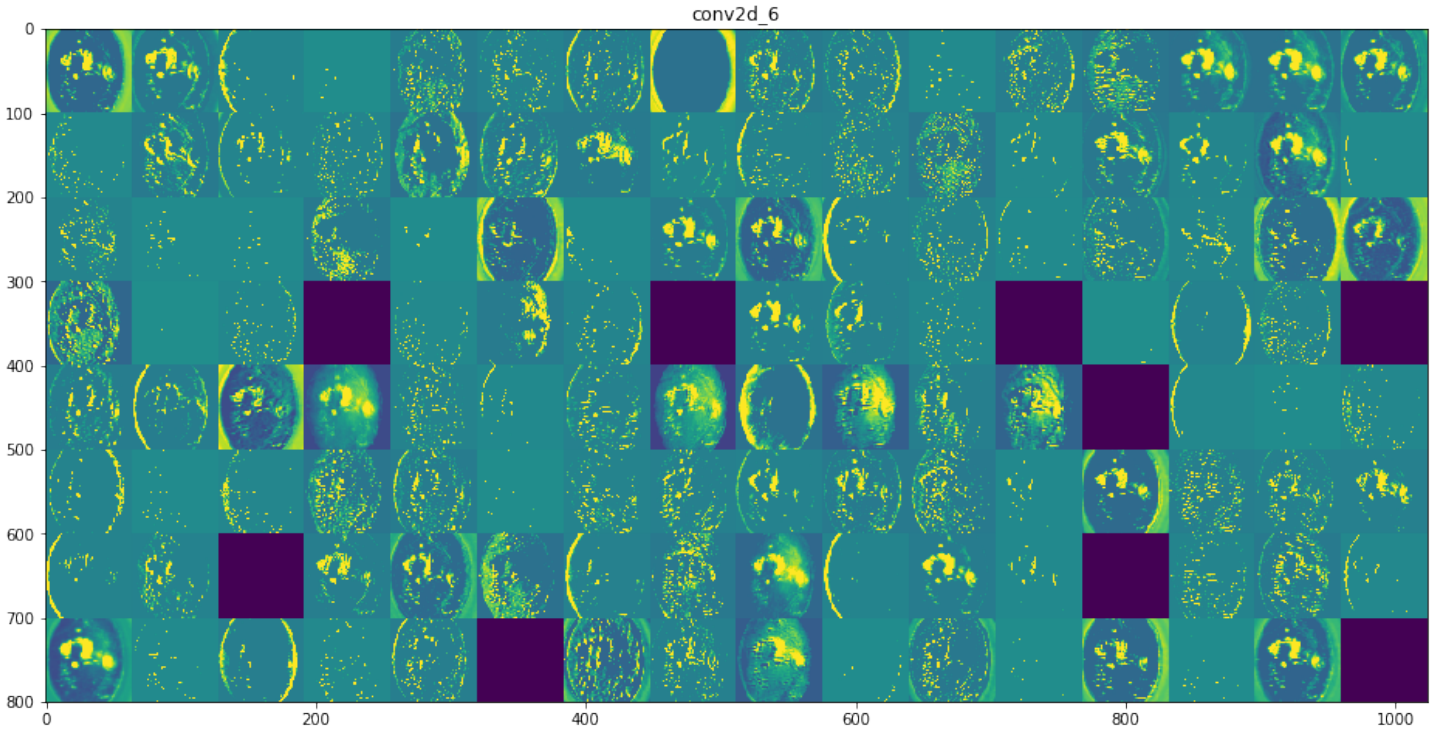


Figura A.8: Salida de la capa conv2d_6 para la imagen ejemplo.

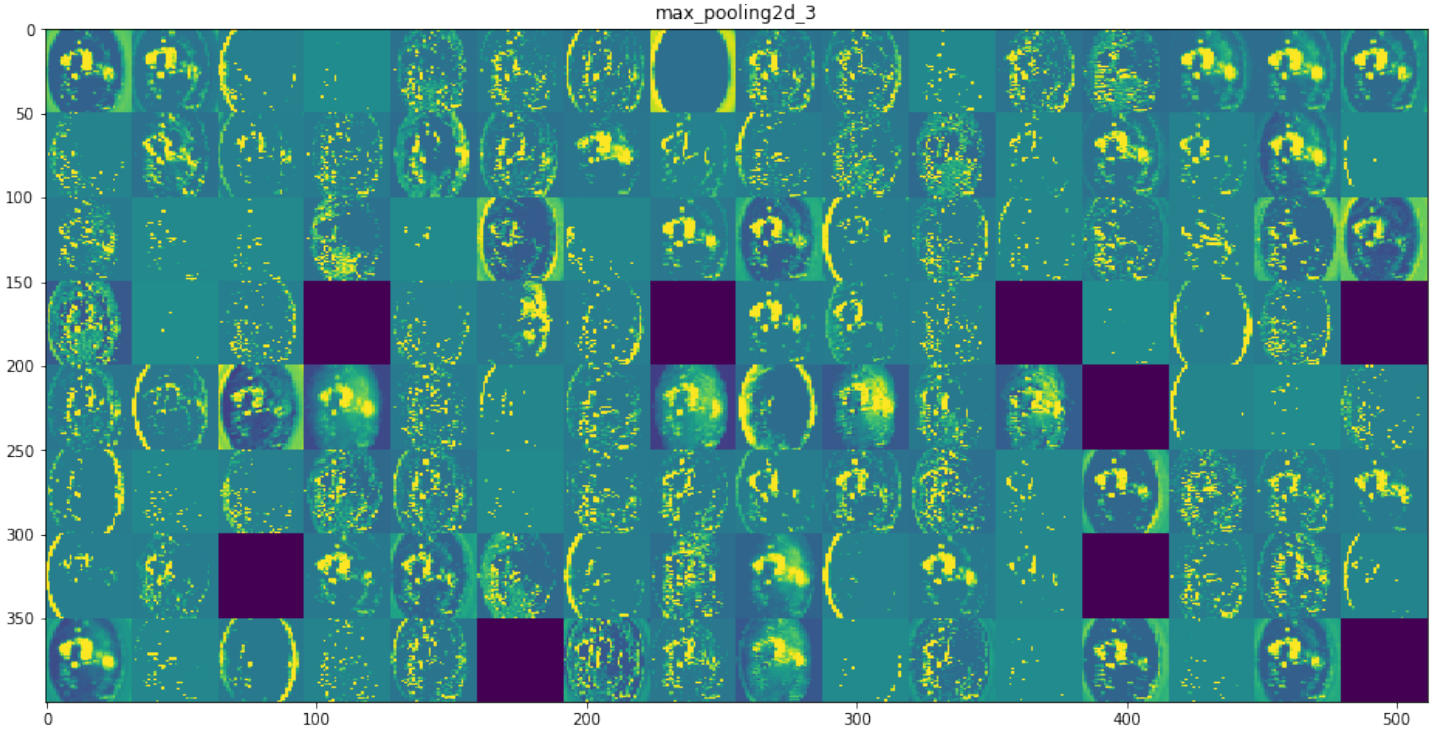


Figura A.9: Salida de la capa maxpooling2d_3 para la imagen ejemplo.

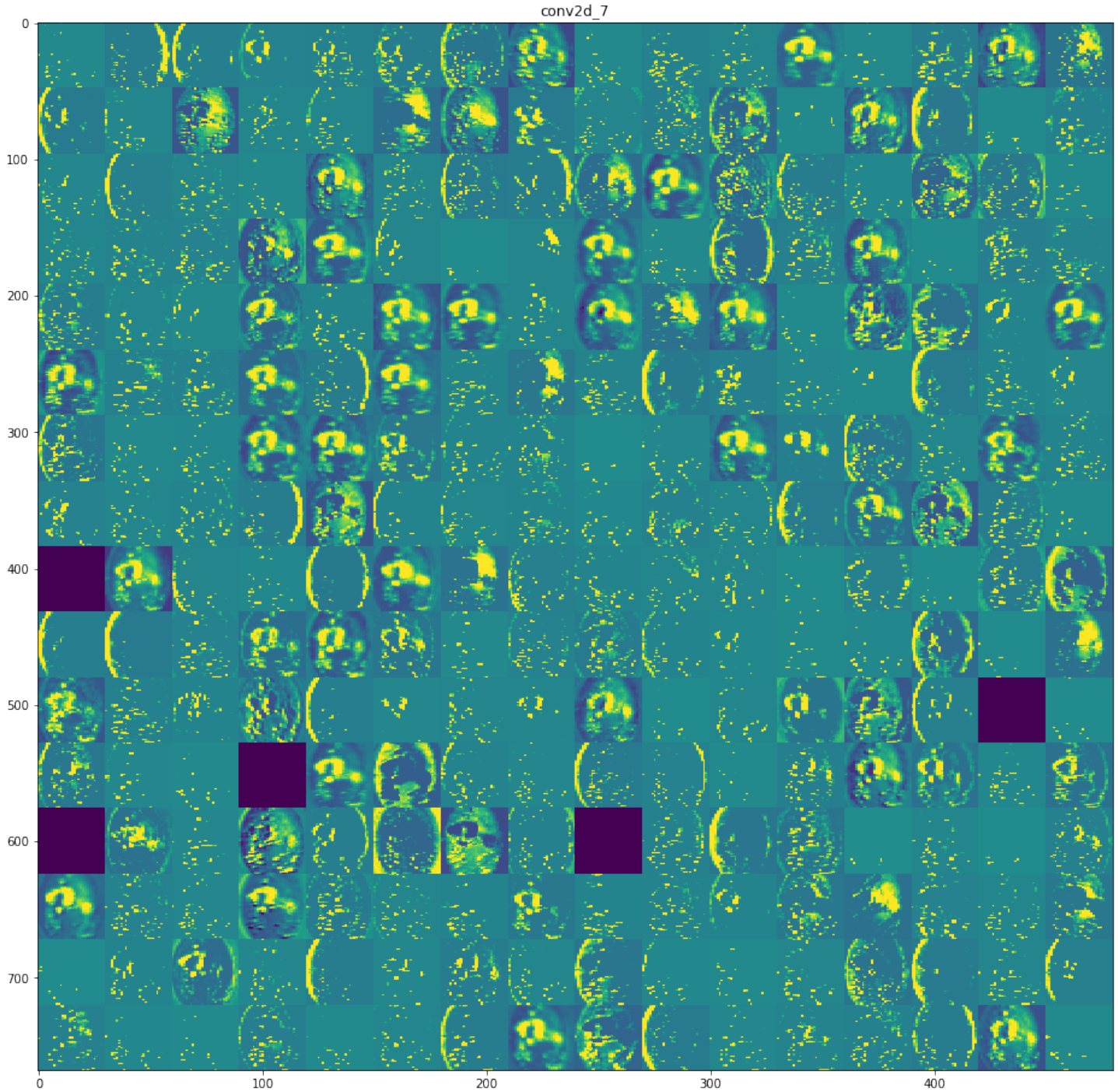


Figura A.10: Salida de la capa conv2d_10 para la imagen ejemplo.

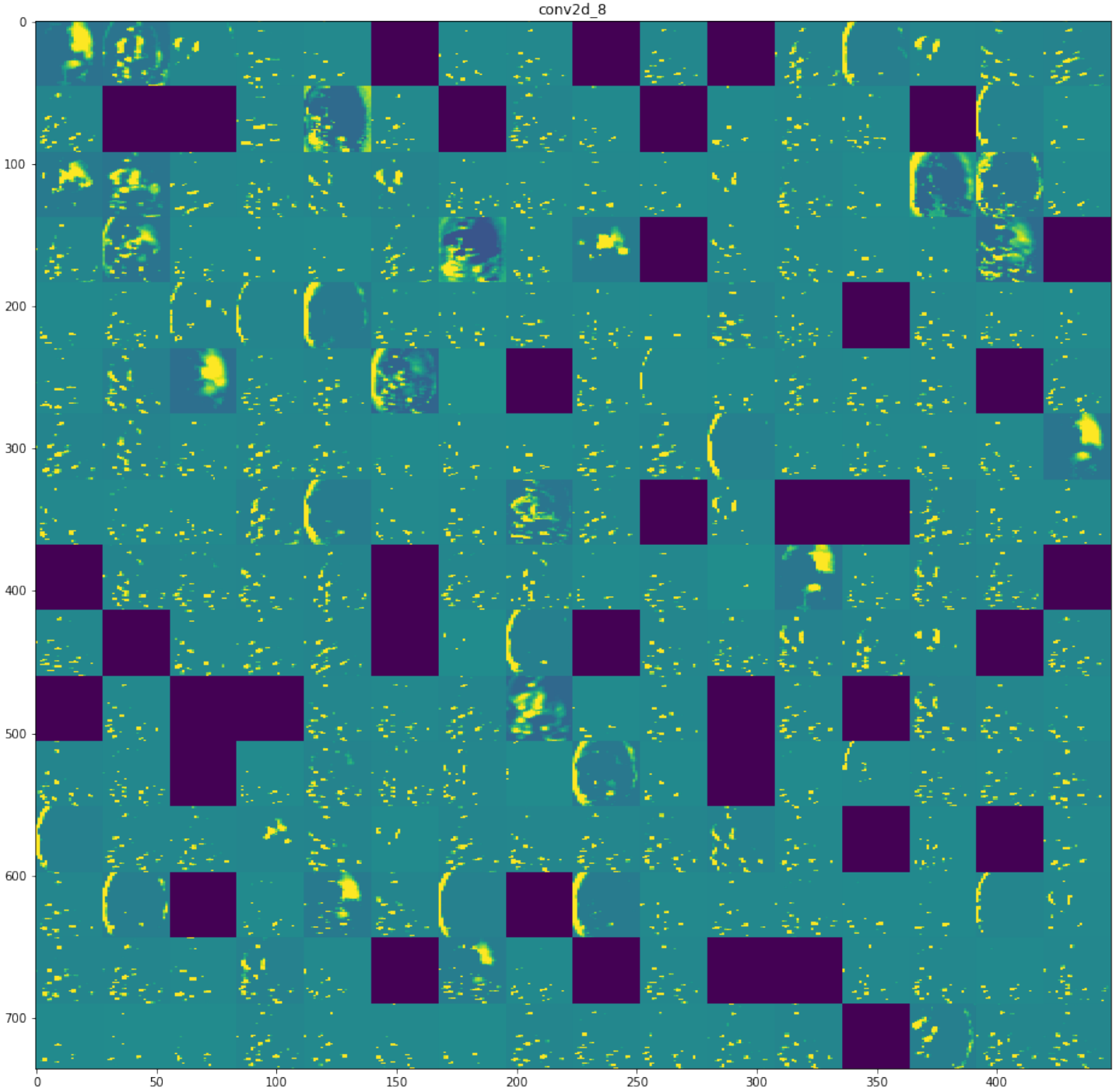


Figura A.11: Salida de la capa conv2d_11 para la imagen ejemplo.

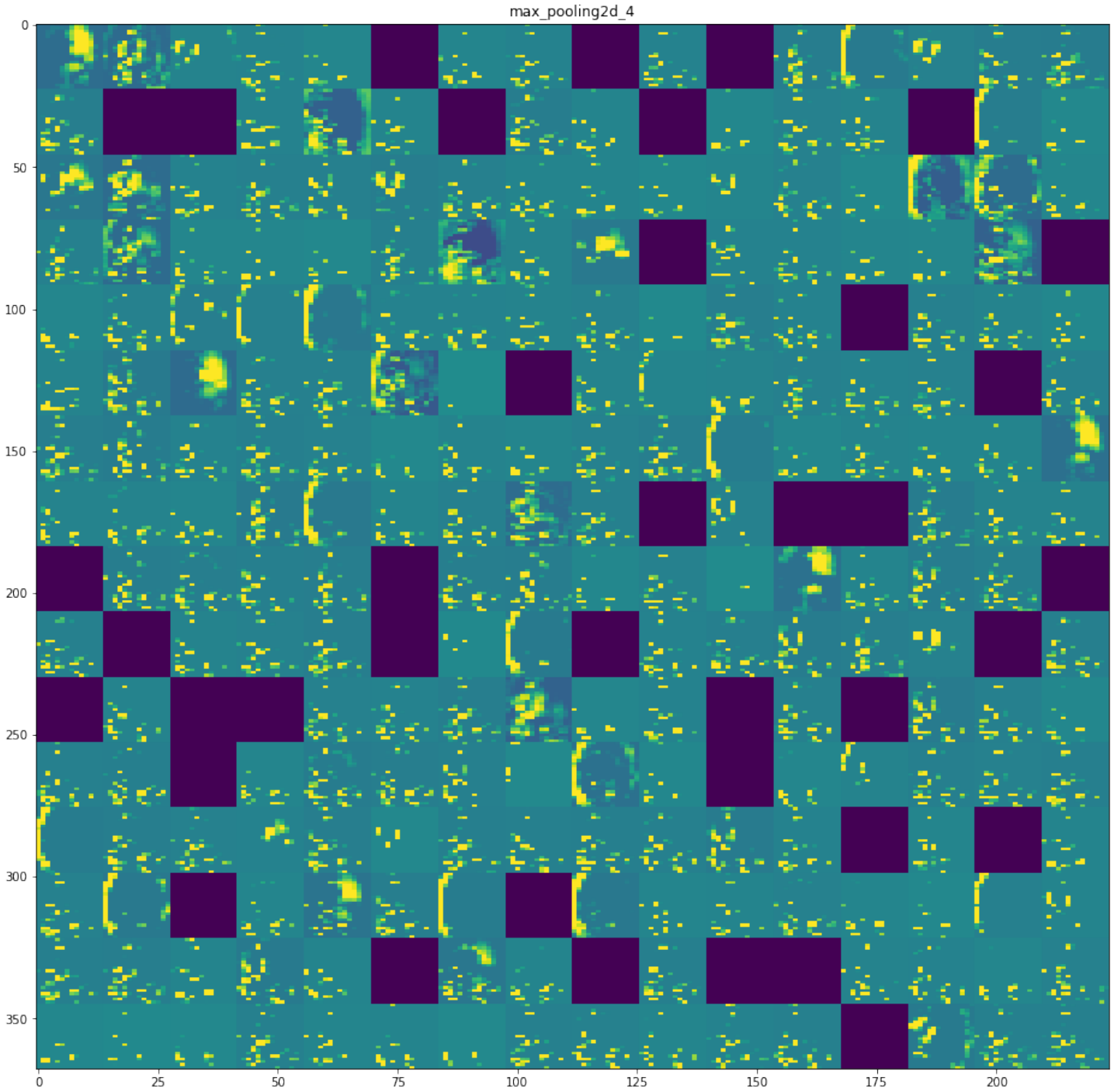


Figura A.12: Salida de la capa maxpooling2d_4 para la imagen ejemplo.

APÉNDICE A. EJEMPLO ILUSTRATIVO DE LA VISUALIZACIÓN DE LAS CAPAS INTERMEDIAS DE UNA RED CONVOLUCIONAL

Apéndice B

Ejemplo ilustrativo de la visualización de los filtros de una red convolucional

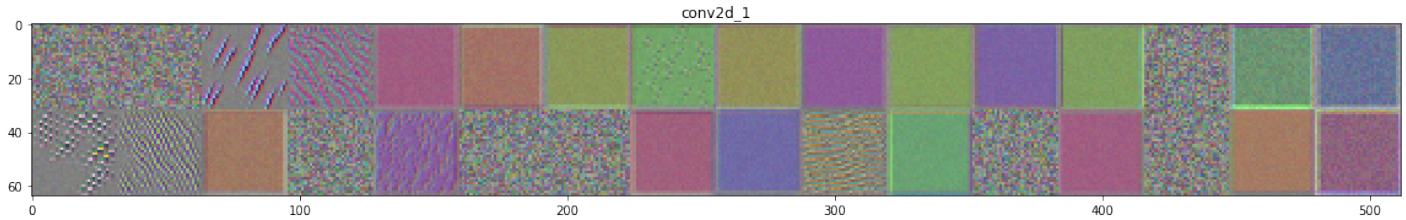


Figura B.1: Filtros de la capa conv2d_1.

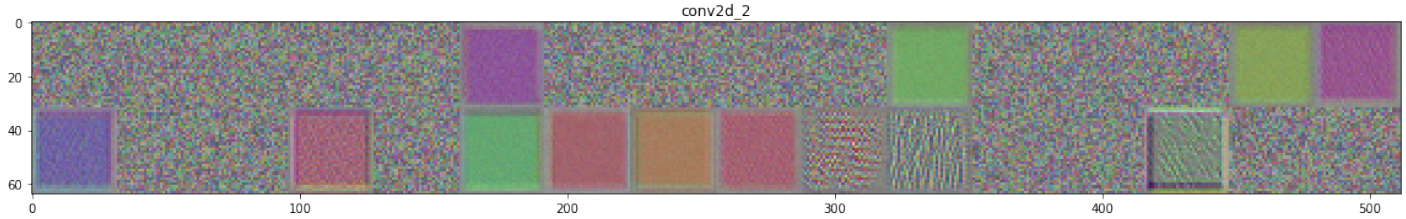


Figura B.2: Filtros de la capa conv2d_2.

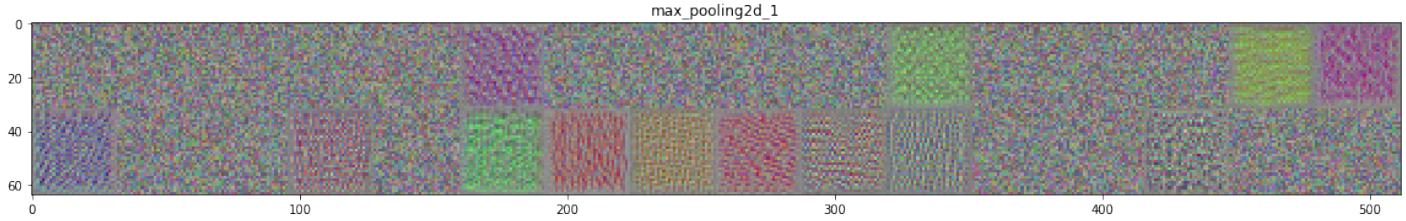


Figura B.3: Filtros de la capa maxpooling2d_1.

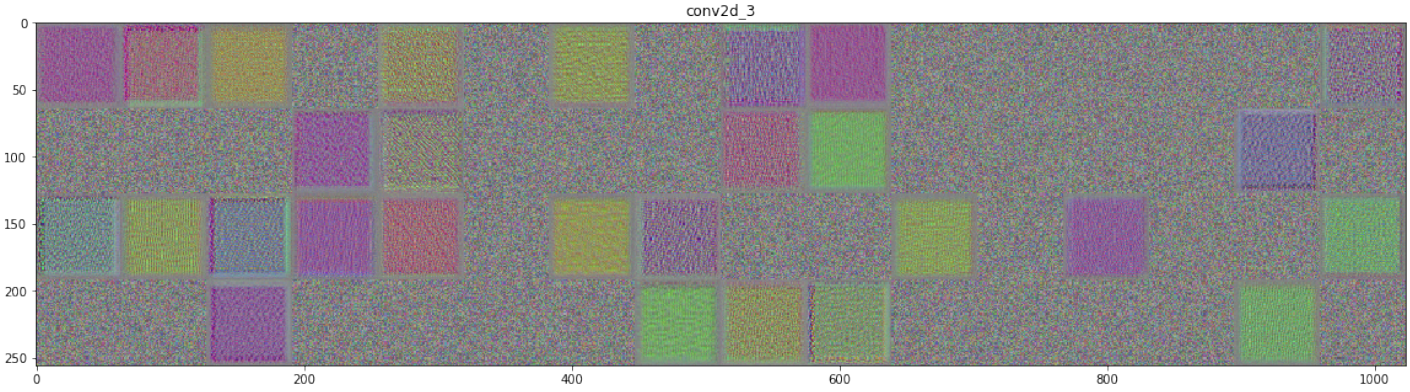


Figura B.4: Filtros de la capa conv2d_3.

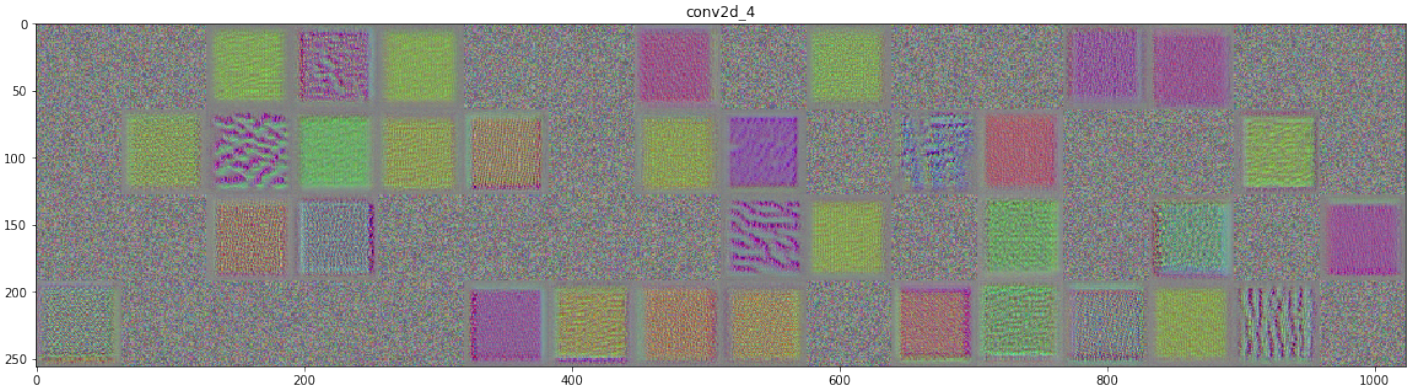


Figura B.5: Filtros de la capa conv2d_4.

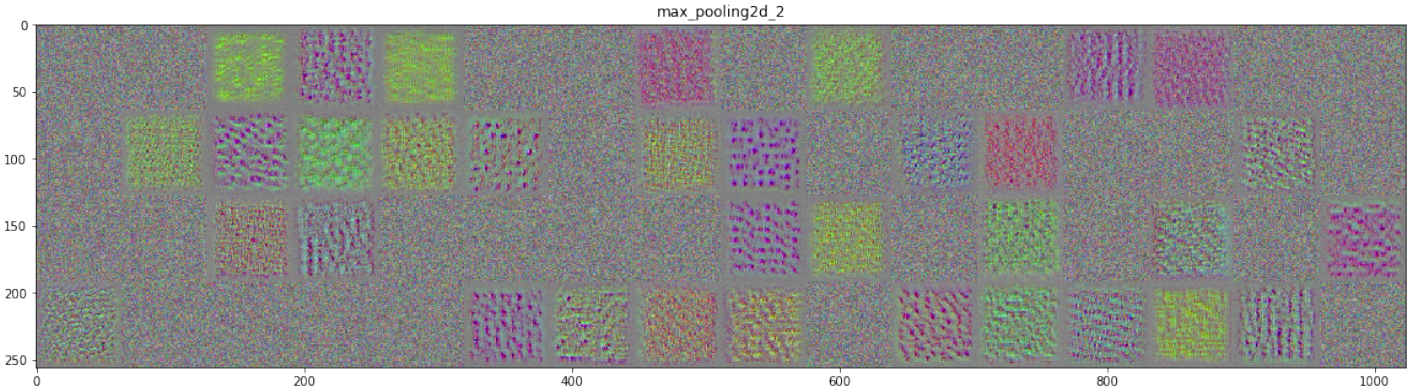


Figura B.6: Filtros de la capa maxpooling2d_2.

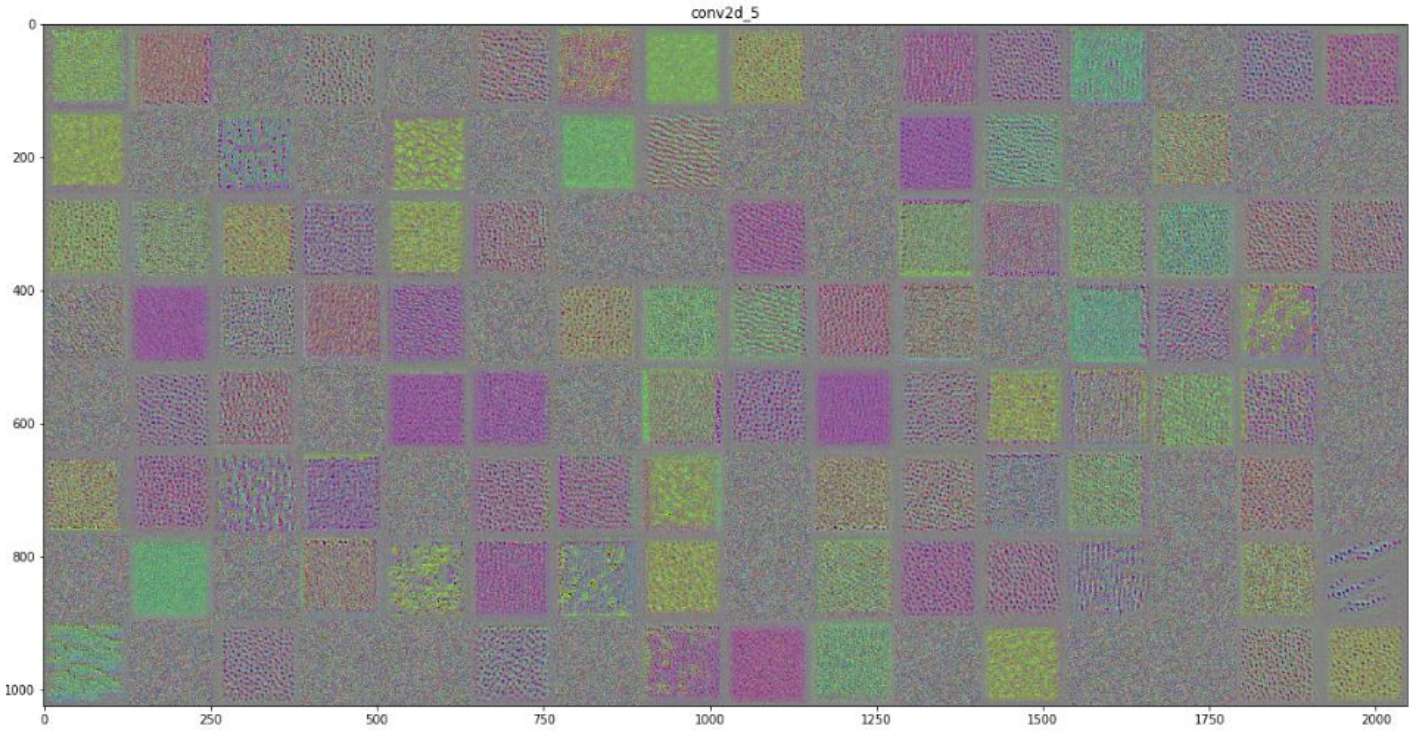


Figura B.7: Filtros de la capa conv2d_5.

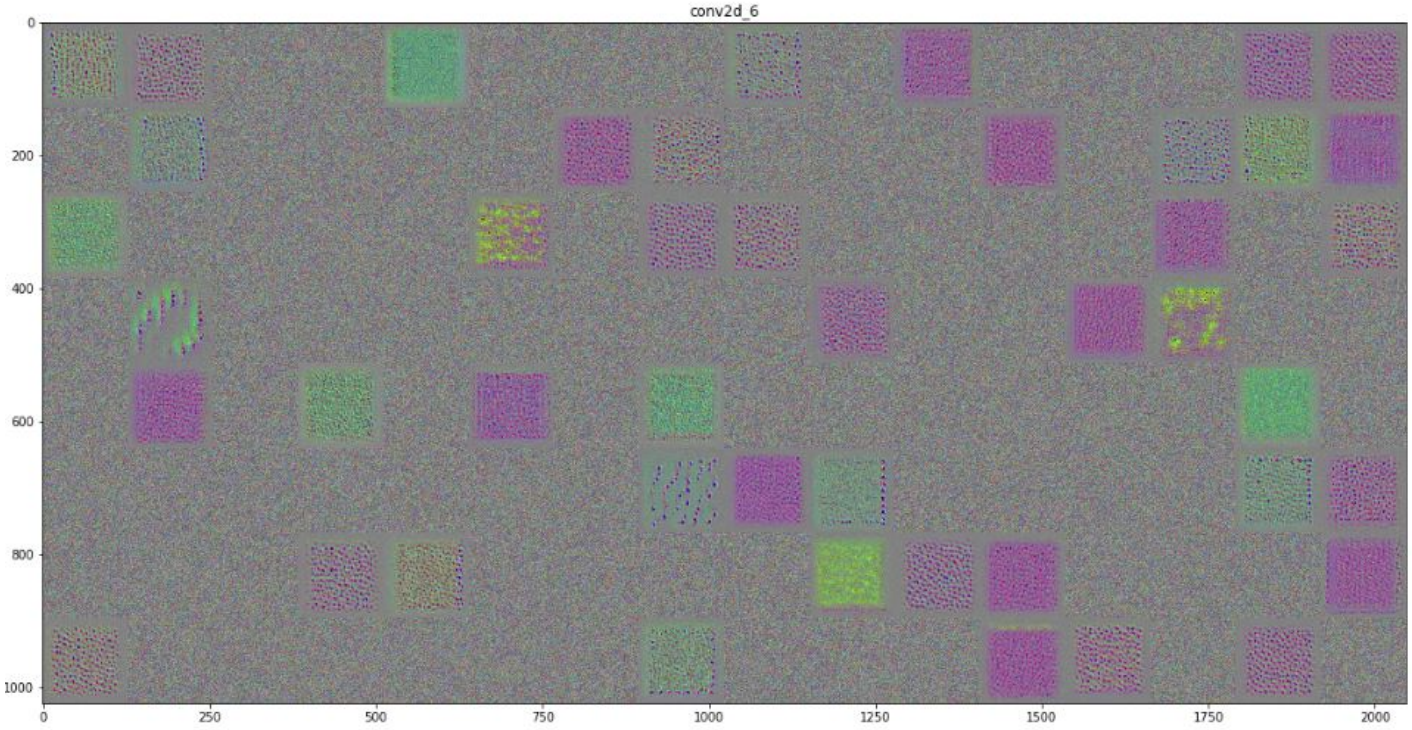


Figura B.8: Filtros de la capa conv2d_6.

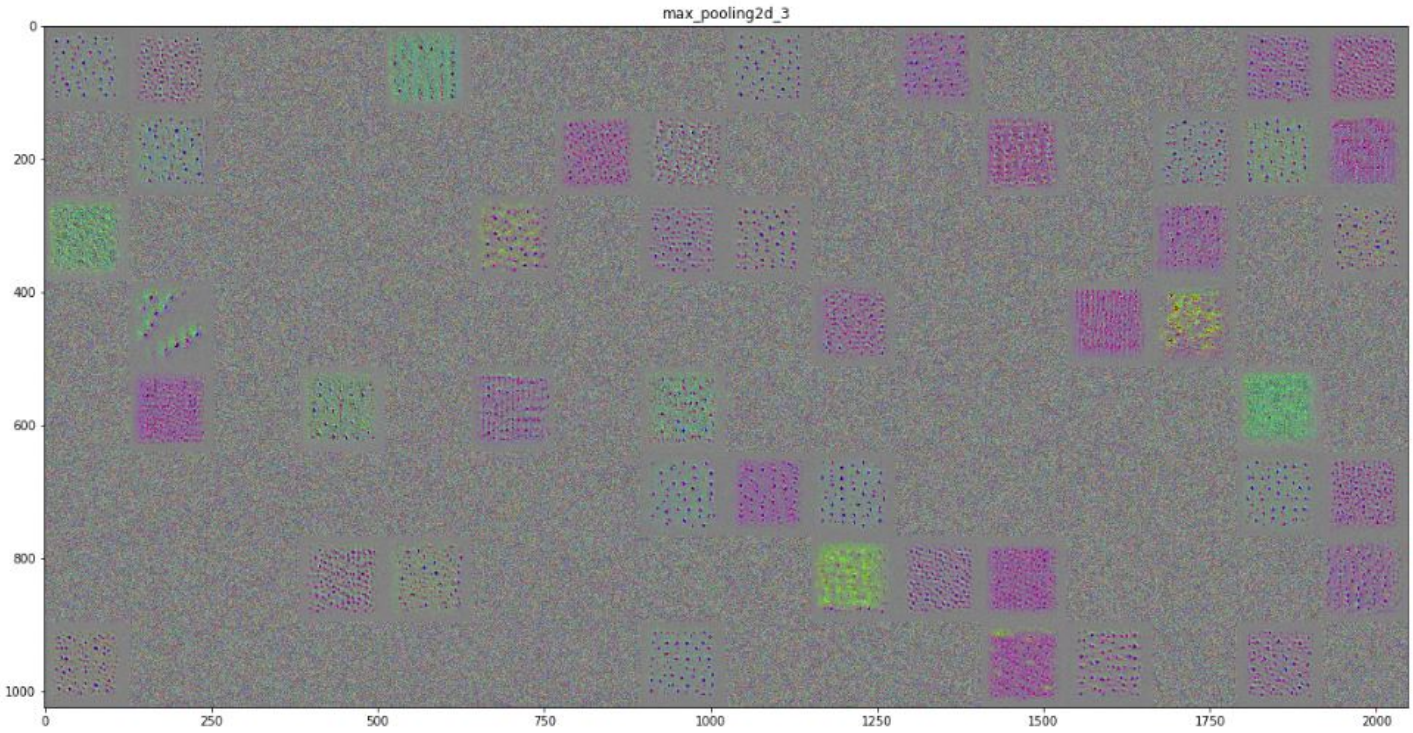


Figura B.9: Filtros de la capa maxpooling2d_3.

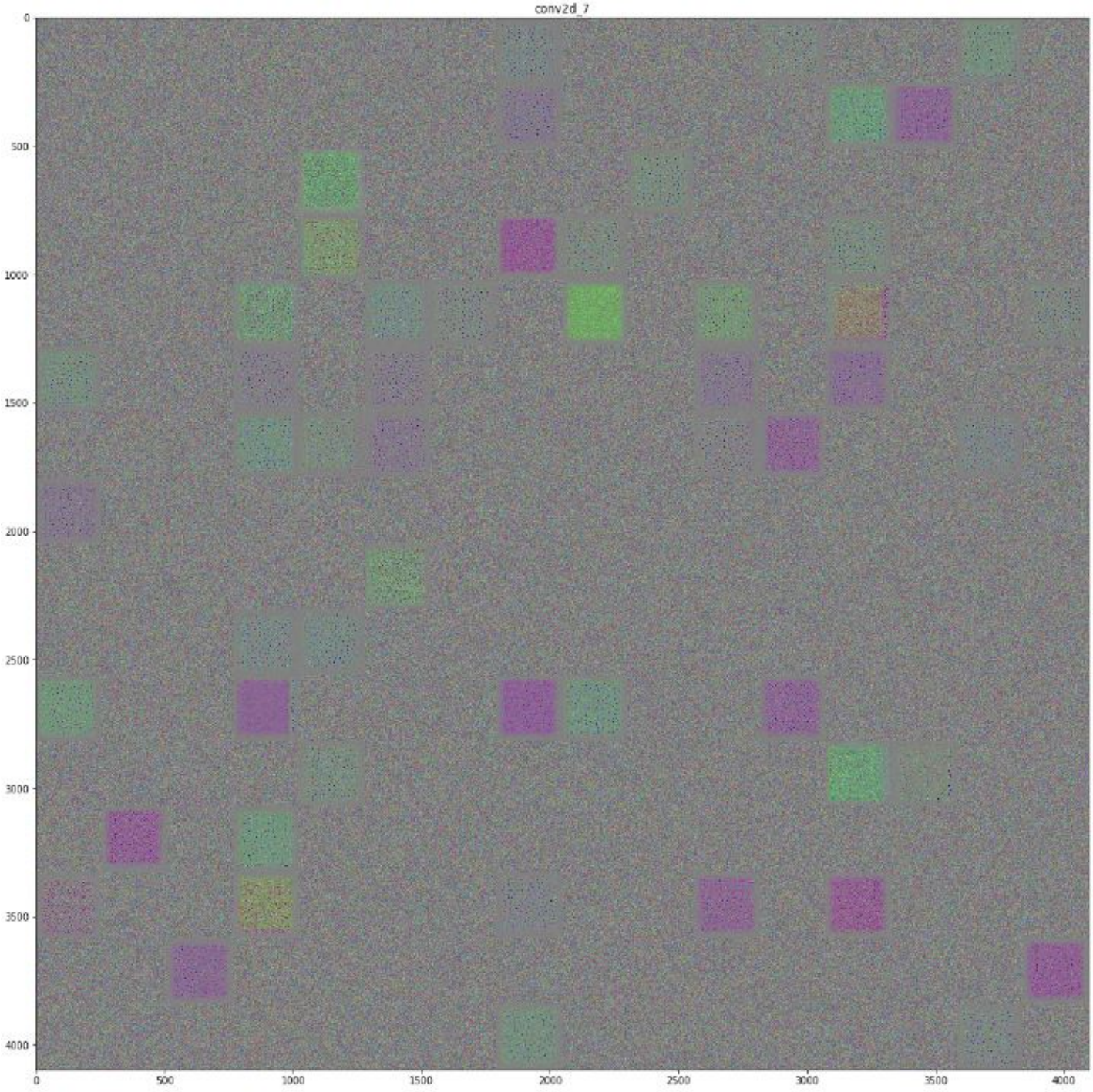


Figura B.10: Filtros de la capa conv2d_10.

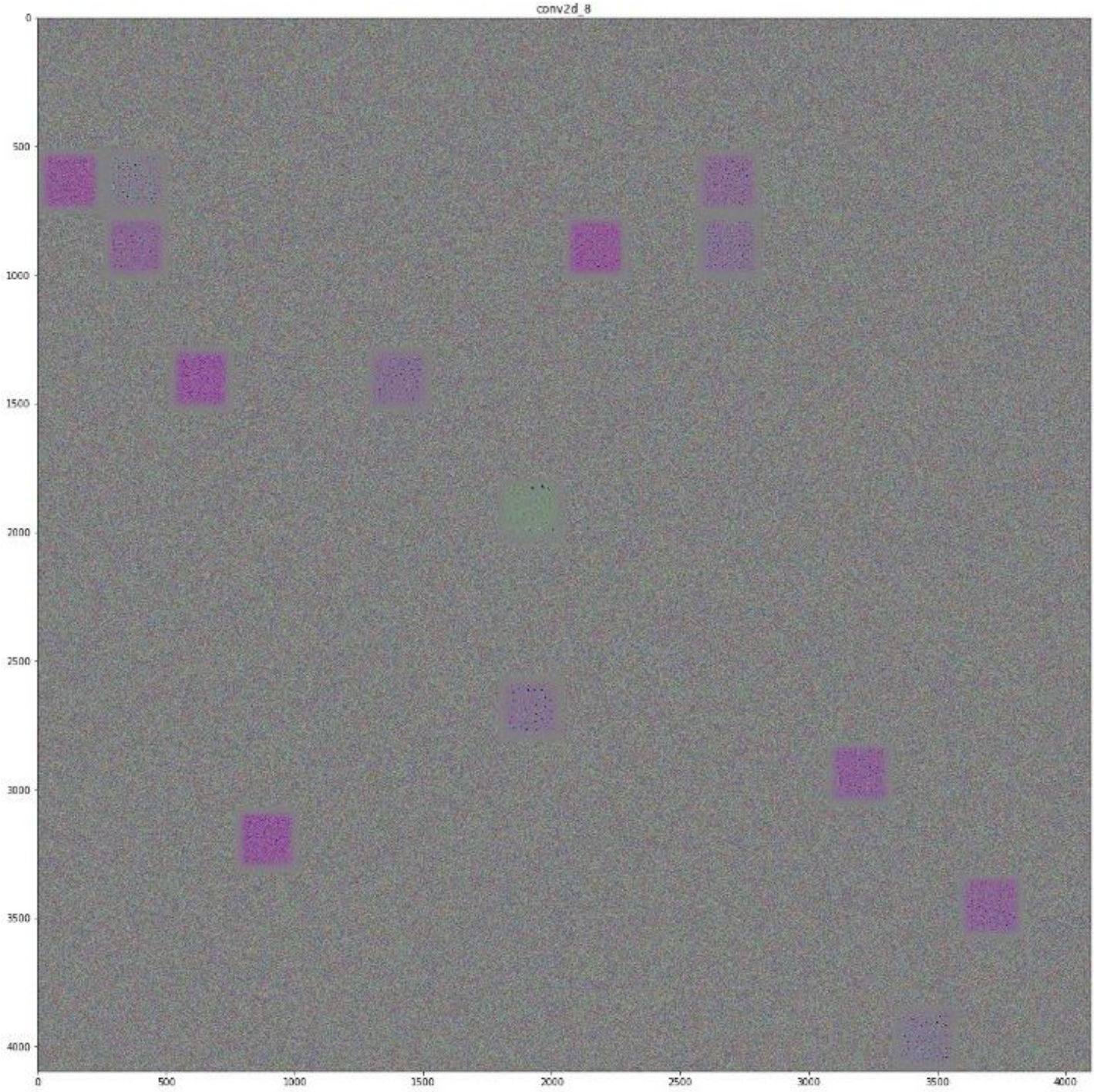


Figura B.11: Filtros de la capa conv2d_11.

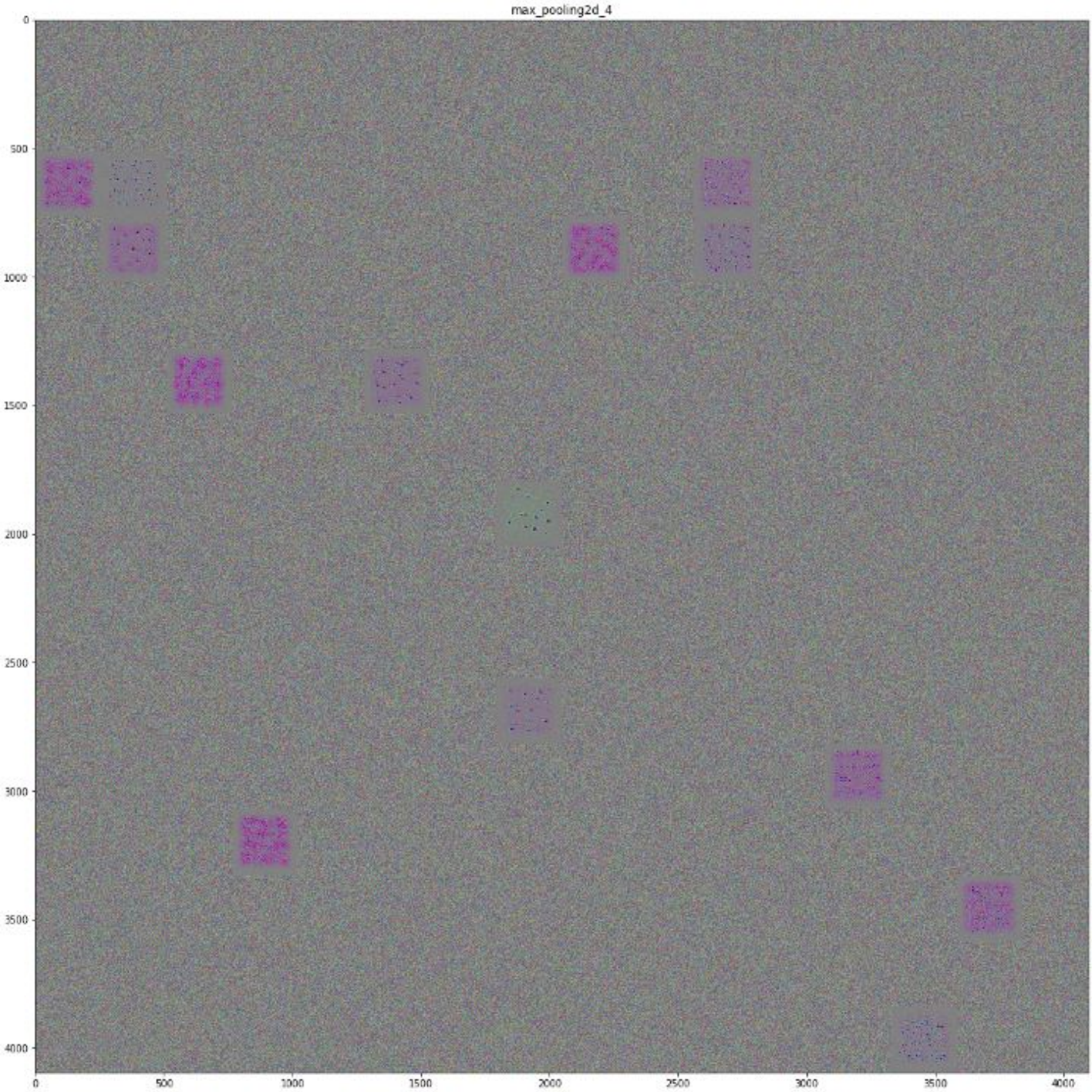


Figura B.12: Filtros de la capa maxpooling2d_4.

APÉNDICE B. EJEMPLO ILUSTRATIVO DE LA VISUALIZACIÓN DE LOS FILTROS DE UNA RED CONVOLUCIONAL

Apéndice C

Manual de instalación

En este anexo se presentan los pasos a seguir para llevar a cabo la instalación y configuración de la aplicación web a nivel de servidor, es decir, orientadas al administrador de la aplicación.

Requisitos

Para poder desplegar la aplicación se deben de satisfacer los siguientes requisitos:

- Sistema operativo GNU/Linux x86_64 con una versión de kernel $>$ 3.10
- Docker
- Docker-compose

La instalación de Docker puede efectuarse siguiendo las instrucciones que se especifican en el siguiente enlace:

`https://docs.docker.com/install/linux/docker-ce/ubuntu/`

La instalación de Docker-compose puede efectuarse siguiendo las instrucciones que se especifican en el sigiente enlace:

`https://docs.docker.com/compose/install/`

Organización de la aplicación

Con el objetivo de clarificar las instrucciones relativas al despliegue de la aplicación, se adjunta a continuación la estructura de los directorios de la aplicación:

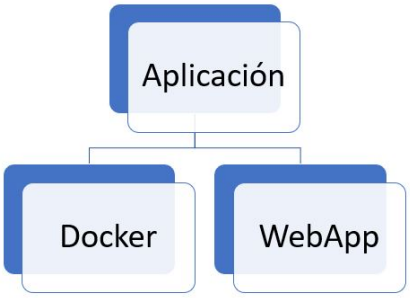


Figura C.1: Organización de la aplicación.

Instalación y despliegue de la aplicación

Para llevar a cabo la instalación y despliegue de la aplicación, simplemente se debe acceder desde terminal a la carpeta Docker contenida en la aplicación y ejecutar el comando:

```
docker-compose up
```

Una vez ejecutado el comando, para acceder a la aplicación desde el navegador, debe especificarse la IP y el puerto de la máquina en la que se ha desplegado la aplicación web (http://IP:Puerto), siendo posible la configuración de estos. Por defecto, este puerto es el 5000.

Si se desea parar la aplicación, simplemente se debe acceder nuevamente desde terminal a la carpeta Docker y ejecutar el comando:

```
docker-compose down
```

Apéndice D

Manual de usuario

Este manual pretende proporcionar a los usuarios el conocimiento necesario para el correcto funcionamiento de la aplicación web.

Acceso a la página web

Para acceder a la aplicación web desde el navegador se debe especificar la IP y el puerto proporcionados por el administrador de la siguiente manera:

http://IP:Puerto

Tras el acceso, se obtiene una vista como la presentada en la Figura D.1.

Subida de una imagen

Una vez ha accedido a la aplicación web, el siguiente paso radica en subir la imagen que se desea diagnosticar. Para ello hay dos alternativas posibles:

- Hacer uso del botón *Examinar*:
  - Se debe hacer click sobre el botón *Examinar*.
  - Se mostrará en una ventana emergente el gestor de sus archivos locales.
  - Se debe seleccionar la imagen que se desee.
- Hacer uso de la opción *Drag & Drop*:
  - Se arrastra la imagen que se desea subir a la caja que indica *Arrastrar imagen aquí*.

Diagnosticar las enfermedades

Una vez especificada la imagen, se debe hacer click sobre el botón *Diagnosticar* para obtener el diagnóstico asociado. Debido a que esta opción puede tardar ligeramente, se muestra una vista indicando que se está efectuando la predicción como la que se adjunta en la Figura D.2.

Figura D.1: Vista inicial tras acceder a la página web.

Figura D.2: Vista relativa a espera.

Finalmente, se muestran por pantalla los resultados obtenidos, pudiendo visualizar la imagen subida, así como la información relativa a la presencia o ausencia del edema

macular y la retinopatía diabética. En la Figura D.3 se puede visualizar un ejemplo ilustrativo de esta vista.

Figura D.3: Vista relativa a mostrar resultados.

En el caso de que al hacer click sobre el botón *Diagnosticar* emerja una alerta con el mensaje *"Error extensión invalida"*, se le estará informando de que el formato de su imagen no está soportado por la aplicación. Los formatos soportados son : *jpg, JPG, PNG, png, jpeg y JPEG.*

Si desea efectuar otra predicción, simplemente debe hacer click sobre el botón *Efectuar otra predicción* y será redirigido a la página inicial.

Bibliografía

[1] Documentación docker compose. https://docs.docker.com/compose/.

[2] Documentación flask. http://flask.pocoo.org/docs/1.0/.

[3] Documentación gunicorn. https://docs.docker.com/compose/.

[4] Documentación gunicorn - diseño. http://docs.gunicorn.org/en/stable/design.html.

[5] Gitlab. https://about.gitlab.com/.

[6] Redis. https://redis.io/.

[7] Indian diabetic retinopathy image dataset. https://ieee-dataport.org/open-access/indian-diabetic-retinopathy-image-dataset-idrid, 2018.

[8] Multi-channel convolutions explained with... ms excel! https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108, 2018.

[9] Notes cs231n: Convolutional neural networks for visual recognition. http://cs231n.github.io/, 2019.

[10] Yosua Bengio, Aaron Courville, and Ian Goodfellow. *Deep Learning*. MIT Press, 2015.

[11] Teodoro Calonge Conde. Perceptrón multicapa - apuntes técnicas de aprendizaje automático, 2017-2018.

[12] François Chollet et al. Keras. https://keras.io, 2015.

[13] François Chollet. *Deep Learing with Python*. Manning, 2017.

[14] El día de Valladolid. Un sistema automático para ver lesiones rojas en la retina. https://www.eldiadevalladolid.com/Noticia/ZF59FCB3D-B0BB-6ECD-3513666CB17CD0B8/201907/Un-sistema-automatico-para-ver-lesiones-rojas-en-la-retina.

[15] George F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Adisson Wesley, 2008.

[16] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools and Tecniques to Build Intelligent Systems*. O'reilly, 2019.

[17] Simon Haykin. *Neural Networks and Learning Machines*. Pearson, 1993.

[18] Martin Heller. What is keras? the deep neural network api explained. `https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html`, 2019.

[19] Ayoosh Kathuria. Intro to optimization in deep learning: Momentum, rmsprop and adam. `https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/`, 2018.

[20] Nikhil Ketkar. *Deep Learing with Python: A Hands-on Introduction*. Apress, 2017.

[21] Santanu Pattanayak. *Pro Deep Learning with Tensorflow: A Mathematical Approach to Advanced Artificial Intelligence in Python*. Apress, 2017.

[22] Omar Rayward. Better performance by optimizing gunicorn config. `http://docs.gunicorn.org/en/stable/design.htmlhttps://medium.com/building-the-system/gunicorn-3-means-of-concurrency-efbb547674b7`, 2018.

[23] Rikiya Yamashita, Mizuho NishioRichard, Kinh Gian, and DoKaori Togashi. *Convolutional neural networks: an overview and application in radiology*. Springer, 2018.