UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Diseño, implementación y validación de técnicas de identificación de células infectadas de malaria mediante redes neuronales

Trabajo Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

AUTOR/A: Marín Calvo, Hernán

Tutor/a: Sánchez Salmerón, Antonio José

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO, IMPLEMENTACIÓN Y VALIDACIÓN DE TÉCNICAS DE IDENTIFICACIÓN DE CÉLULAS INFECTADAS DE MALARIA MEDIANTE REDES NEURONALES

AUTOR: HERNÁN MARÍN CALVO

TUTOR: ANTONIO JOSÉ SÁNCHEZ SALMERÓN

Curso Académico: 2021-22

RESUMEN

El objetivo de este trabajo es el diseño, la implementación y la validación de un modelo de clasificación de imágenes, en este caso concreto, de un clasificador que sea capaz de discernir entre una célula en buen estado y otra que este infectada de malaria. Este tipo de modelos se podrían emplear en el sector sanitario para aumentar la precisión y velocidad con la que se detectan las enfermedades parasitarias. No obstante, también podrían ser extrapolables a otro tipo de enfermedades aplicando la misma metodología.

Para la creación de dicho clasificador, se emplean técnicas de inteligencia artificial y *machine learning* como son las redes neuronales o las redes convolucionales. A través de estas, el clasificador aprende de forma totalmente automática a reconocer aquellas características que diferencian una imagen de una célula infectada, de otra que no lo está.

Partiendo de un conjunto de datos ("dataset") de aproximadamente 20.000 imágenes de células infectadas y sin infectar, se diseñará, implementará y validará el modelo utilizando Python, OpenCV y Pytorch.

Finalmente, se evaluarán y optimizarán las distintas propuestas, utilizando como criterio de optimización la tasa de aciertos, con el fin de obtener el clasificador con el mayor porcentaje de aciertos posible.

Palabras clave: Malaria, Clasificador de imágenes, redes neuronales, inteligencia artificial.

RESUM

L'objectiu d'aquest treball és la creació del model d'un classificador d'imatges, en aquest cas concret, d'un classificador que siga capaç de discernir entre una cèl·lula en bon estat i una altra infectada de malària. Aquest tipus de models es podrien emprar en el sector sanitari per a augmentar la precisió i velocitat amb la qual es detecten les malalties parasitàries. No obstant això, també podrien ser extrapolables a una altra mena de malalties aplicant la mateixa metodologia.

Per a la creació d'aquest classificador, s'empren tècniques d'intel·ligència artificial i *machine learning* com són les xarxes neuronals o les xarxes convolucionals. A través d'aquestes, el classificador aprén de forma totalment automàtica a reconéixer aquelles característiques que diferencien una imatge d'una cèl·lula infectada, d'una altra que no ho està.

Partint d'un conjunt de dades ("dataset") d'aproximadament 20.000 imatges de cèl·lules infectades i sense infectar, es dissenyarà, implementarà i validarà el model utilitzant *Python, OpenCV i Pytorch.

Finalment, s'avaluaran i optimitzaran les diferents propostes, utilitzant com a criteri d'optimització la taxa d'encerts, amb la finalitat d'obtindre el classificador amb el major percentatge d'encerts possible.

Paraules clau: Malària, Classificador d'imatges, xarxes neuronals, intel·ligència artificial.

SUMMARY

The objective of this work is the creation of a model of an image classifier, in this particular case, of a classifier that is able to discern between a cell in good condition and another one that is infected with malaria. Such models could be used in the health sector to increase the accuracy and speed with which parasitic diseases are detected. However, they could also be extrapolated to other types of diseases by applying the same methodology.

To create this classifier, artificial intelligence and machine learning techniques such as neural networks or convolutional networks are used. Through these, the classifier automatically learns to recognize those characteristics that differentiate an image of an infected cell from one that is not infected.

Starting from a dataset of approximately 20,000 images of infected and uninfected cells, the model will be designed, implemented and validated using Python, OpenCV and Pytorch.

Finally, the different proposals will be evaluated and optimised, using the hit rate as an optimisation criterion, in order to obtain the classifier with the highest possible hit rate.

Keywords: Malaria, Image classifier, neural networks, artificial intelligence.

ÍNDICE

DOCUMENTOS CONTENIDOS EN EL TFG

1. MEMORIA
2. PRESUPUESTO

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

A lo largo de los últimos años, el ser humano ha ido evolucionando de la mano de la tecnología. Cada vez son más los dispositivos que emplea un ser humano durante su día a día, y es que, cada vez son más, las nuevas tecnologías que aportan novedosas y revolucionarias técnicas digitales.

Uno de los sectores que más se beneficia de esta evolución es el sanitario. En el ámbito de la medicina, han surgido nuevas técnicas que han facilitado el estudio y la visualización del cuerpo humano, con las cuales resulta más sencillo la obtención de datos de los pacientes, el análisis de estos datos, o en general, resulta más sencillo realizar un seguimiento detallado de cualquier enfermedad que se presente en el cuerpo de un paciente.

Hay que destacar, que, en la mayoría de los casos, la utilidad que pueda llegar a tener la imagen de cierto análisis acaba dependiendo de la pericia del profesional sanitario para interpretarla. No obstante, con los avances mencionados previamente, surgen nuevas formas de analizar imágenes facilitando la tarea al sanitario, además de mejorar la calidad y precisión de los diagnósticos.

Existen diferentes metodologías para el tratamiento de imágenes, así como para su estudio e interpretación. Sin embargo, en los últimos años, problemas de esta índole resultan mucho más fácil de abordar gracias a la gran evolución que se ha llevado a cabo en el mundo de la inteligencia artificial, concretamente en las técnicas de *machine learning*, o aprendizaje de la máquina, conocidas como Redes Neuronales Profundas o DNN ('Deep Neuronal Networks').

Empleando el software indicado, así como herramientas como Pytorch entre otras, se pueden obtener estas redes neuronales computacionales, que están compuestas por diferentes capas de 'neuronas'. La idea detrás de estas redes es imitar el funcionamiento de las conexiones neuronales que existen en el cerebro humano, así como los razonamientos e interpretaciones que hace. Estos modelos se han mostrado realmente efectivos en los campos de reconocimiento, segmentación y clasificación de imágenes, siendo este último el que se abordará en el desarrollo del Trabajo Fin de Grado (TFG).

Una de las ventajas de emplear redes neuronales, es el aprendizaje automático de la misma. Es decir, es capaz de modificar y reajustar por sí misma los diferentes parámetros que la definen con el objetivo de lograr la mayor precisión en aquello para lo que se ha definido. Para llevar a cabo este aprendizaje, se emplean diferentes técnicas de retro propagación o '*Backpropagation*', las cuales parten todas de la idea del algoritmo de optimización del Descenso del Gradiente.

De esta forma, una red cuyo objetivo es la clasificación de imágenes, las analiza e interpreta, observando sus características, sus patrones, o aquellos rasgos que permiten identificarlas. Y al mismo tiempo, como se ha mencionado, ajusta sus parámetros de tal forma que capaz de distinguir las diferentes categorías en las que se busca clasificar.

Finalmente cabe destacar que para la realización de este trabajo se ha empleado el conjunto de datos del trabajo titulado "*Deep Learning for Smartphone-Based Malaria Parasite Detection in Thick Blood Smears",* cuyo objetivo es similar al buscado por este trabajo, por lo que también se ha empleado como referencia e inspiración. [1]

1.2. ANTECEDENTES DE VISIÓN ARTIFICIAL

Durante las últimas décadas se han podido resolver multitud de aplicaciones mediante la implantación de sistemas de inspección 2D tanto en aplicaciones industriales como en imagen médica utilizando técnicas de visión por computador tradicionales. El principal problema de estos sistemas es la gran variabilidad de las imágenes capturadas que dificulta la segmentación de los objetos de interés. Para resolver este problema, por un lado, se han propuesto técnicas robustas de segmentación. [2] [3]

Por otro lado, también se han propuesto soluciones aplicando técnicas específicas para captura de imágenes con escáneres 3D [4] [5] [6] [7] [8], sensores hiperespectrales [9] [10] [11]y sistemas de visión activa [12] [13] que ayudan a reducir la variabilidad de las imágenes capturadas y resolver aplicaciones complejas. [14] [15] [16] [17] [18]

Además, los últimos avances producidos en la aplicación de técnicas de clasificación de imágenes basadas en redes neuronales convolucionales; están permitiendo automatizar algunas aplicaciones de clasificación complejas que hace unos años eran impensables. [19] [20]

La clasificación de imagen médica es un problema complejo debido a la gran variabilidad de imágenes que se pueden capturar. En este trabajo se plantea diseñar y validar un modelo de aprendizaje profundo para clasificar imágenes de malaria.

1.3. OBJETIVOS

La finalidad de este trabajo es la creación de un clasificador de imágenes empleando redes neuronales de pequeño y mediano tamaño. Para la realización de este se ha empleado Python, así como las librerías de OpenCV y Pytorch.

En este caso concreto, se ha utilizado como datos de entrada un conjunto de imágenes de células de la sangre que pueden estar infectadas de malaria. Luego, el objetivo final es que el clasificador sea capaz de predecir con fiabilidad que células están sanas y que células están infectadas. Hay que entender que este clasificador es un primer boceto, una primera idea, o una base a partir de la cual podría llegar a surgir un clasificador lo suficientemente fiable como para poder aplicarse en el ámbito sanitario. Esto se debe a que el conjunto de datos con el que se va a trabajar está acotado a 20000 imágenes y por tanto el modelo podría no mostrarse igual de fiable para otros datos de diferente origen.

Es por ello por lo que se quiere recalcar la finalidad académica de este trabajo, pese al pretexto sanitario con el que se trabaja, se busca que el alumno sea capaz de diseñar e implementar un modelo de clasificador de imágenes con solvencia y buenos resultados.

1.4. ORGANIZACIÓN DE LA MEMORIA

Este trabajado sigue una estructura dividida en capítulos, con el objetivo de facilitar al lector el entendimiento de toda la información. De esta forma, la memoria de este trabajo queda dividida de la siguiente forma:

- Introducción: Tal y como se ha leído hasta ahora, en este capítulo se pretende dar a conocer la idea tras este proyecto, así como los objetivos que se persiguen en él.
- Información relativa a la malaria: en este apartado se pone de manifiesto breves nociones de conocimiento sobre la malaria al ser el objeto de estudio y dar contexto al trabajo.
- Estado del arte en redes neuronales: a lo largo de este bloque se muestra el desarrollo del funcionamiento de las redes neuronales, así como las principales características que la definen y el razonamiento que hay detrás de su uso.
- Trabajo desarrollado: en este capítulo se expone de manera extensa aquellos pasos y razonamientos que se han ido llevando a cabo para la creación e implementación de la red neuronal. Del mismo modo, también se muestra las técnicas empleadas en el entrenamiento de la red, así como alternativas al modelo creado.
- Análisis de los resultados: Tras la información obtenida en los apartados previos y sabiendo cómo influyen los diferentes parámetros, en esta sección se revelan los diferentes resultados obtenidos.
- Conclusiones: Finalmente, en vista de los resultados logrados en el apartado anterior se enuncian las conclusiones extraídas tras la realización del trabajo.

2. INFORMACIÓN RELATIVA A LA MALARIA

El paludismo (o malaria) es una enfermedad febril, la cual puede llegar a ser mortal. Esta enfermedad es causada por parásitos que se transmiten a los seres humanos a través de la picadura de hembras infectadas de insectos del género mosquito Anopheles. No obstante, hay que mencionar que esta enfermedad se puede prevenir y curar.

Como se ha mencionado previamente, la malaria es una enfermedad parasitaria. Concretamente, existen cinco tipos de parásitos causantes de esta enfermedad en el ser humano, de los cuales la especie *P. falciparum,* es la más peligrosa de todas, capaz de generar graves consecuencias en el paciente, pudiendo llegar a causar la muerte en apenas 24 horas. [21]

Por este motivo, es necesaria tanto una correcta prevención, como una rápida detección de la infección. Existen diferentes medidas preventivas para evitar la transmisión e infección de los seres humanos. No obstante, en ocasiones, el contagio es inevitable, y es ahí donde entra en juego la detección de los parásitos en sangre.

La velocidad con la que se realiza el diagnóstico y se aplica el tratamiento del paludismo es vital para reducir la incidencia de la enfermedad, para evitar sus efectos mortales, así como para disminuir la transmisión en la medida de lo posible.

Las técnicas para la detección de la enfermedad permiten distinguir entre estas fiebres de origen palúdico, y las ocasionadas por otros motivos. Cuando una persona se infecta, los parásitos pasan al torrente sanguíneo, y da comienzo la reproducción de estos en los glóbulos rojos del paciente, provocando así la enfermedad. Es por ello, que estas técnicas de detección buscan signos de estas infecciones en la sangre.

Las pruebas de detección consisten en análisis de sangre llevados a cabo de diferentes formas. Pueden ser pruebas de diagnóstico rápido, en las cuales se busca detectar proteínas conocidas como antígenos producidos por los parásitos. Este tipo suele ser de rápidos resultados, pero requieren de pruebas más fiables para confirmar el diagnóstico.

Por otro lado, se puede llevar a cabo un frotis de sangre, consistente en analizar una muestra de sangre, a través de un microscopio. En esta prueba, el profesional de laboratorio busca la presencia de parásitos en las células de la sangre. [22]

Es en este momento, donde aparece la idea de un clasificador de imágenes que sea capaz de detectar de forma fiable células infectadas de paludismo. De esta forma se aceleraría el proceso de detección de la enfermedad, además de evitar depender de la pericia y el ojo crítico del personal sanitario.

3. ESTADO DEL ARTE EN REDES NEURONALES

3.1. RED NEURONAL BIOLÓGICA

A pesar de los avances realizados en el ámbito de la neurología, continúa sorprendiendo la forma que tiene el ser humano y su cerebro para procesar la información y reaccionar ante los estímulos entrantes. No obstante, es sabido que el procesamiento de datos que realiza el cerebro está fundamentado en unidades elementales, conocidas como neuronas.

El cerebro humano contiene alrededor de 100 mil millones de neuronas, y a su vez cada una de las neuronas está conectada simultáneamente con hasta otras 50 mil, por lo que el ser humano posee billones de conexiones en el cerebro, que se llevan a cabo en apenas milisegundos. No obstante, cabe destacar que no todas las neuronas se emplean para las mismas funciones, si no que se agrupan para llevar a cabo funciones específicas, como, por ejemplo, las que se encargan de generar movimiento en el cuerpo, otras se encargan de analizar estímulos y enviarlos al cerebro, u otras encargadas de los procesos involuntarios.

Estas conexiones mencionadas se realizan de acuerdo con el siguiente proceso. Una neurona recibe señales provenientes de las neuronas a las que está conectada a través de sus dendritas, en ese momento, si la intensidad de las señales recibidas es superior a un determinado umbral, la neurona se activa generando así, en su cuerpo celular, una señal eléctrica que se prolonga a lo largo de su axón.

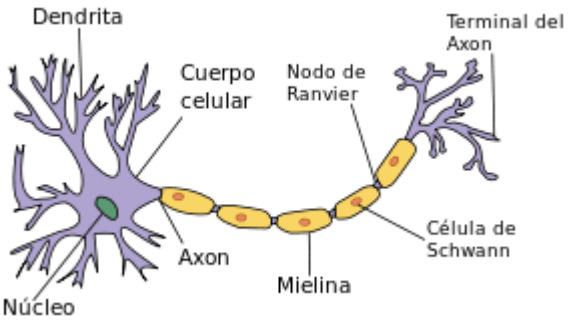

Figura 1: Esquema básico de una neurona biológica [23]

En el momento de transmitir este impulso eléctrico a otra neurona en la sinapsis química (la más habitual), la señal eléctrica se convierte en señal química en el terminal del axón. Para realizar esta transformación, se activa el botón sináptico, que provoca la rotura de las vesículas sinápticas liberando así, neurotransmisores en la sinapsis, siendo esta una interconexión entre dos neuronas. Estos neurotransmisores pasan la señal a las dendritas de la neurona próxima, comenzando de nuevo el proceso de transmisión.

Luego, cuando se habla de aprendizaje en este ámbito, se hace referencia al proceso que realiza el cerebro para modificar el número de neurotransmisores liberados en un botón sináptico, así como del umbral de activación del cuerpo celular. De esta forma, se varia la intensidad de la señal emitida por una neurona, y, por ende, la señal en la neurona receptora.

Es decir, el proceso de aprendizaje en una red neuronal biológica no es más que la variación de la medida en la que una neurona está conectada y se relaciona con las demás, de tal forma que se puedan obtener las respuestas deseadas por el cerebro humano. [24] [25]

3.2. APRENDIZAJE AUTOMÁTICO

Si hay algo que realmente destaque en el campo de las redes neuronales es el *machine learning* (aprendizaje automático). Se entiende por aprendizaje automático como el desarrollo de modelos y sistemas capaces de ajustar su comportamiento, así como los valores que los definen, de forma totalmente autosuficiente, en base a la experiencia y el análisis de los datos que se emplean. Se dice que una red está aprendiendo cuando el error cometido en la tarea que se desempeña se reduce. De esta forma, se destacan dos tipos de aprendizaje:

Aprendizaje No Supervisado (*Unsupervised Learning*): Se introducen en el sistema una numerosa cantidad de datos, cada uno con sus respectivas características, a partir de las cuales, la red trata de aprender y detectar patrones, de tal forma que sea capaz de identificar los diferentes tipos de datos con los que se trabaja.

Aprendizaje supervisado *(Supervised Learning*): En esta ocasión, la red recibe tanto los datos de entrada como una serie de etiquetas que permite identificar cada tipo de datos. De esta forma, el modelo tratara de asociar diferentes patrones o rasgos a cada etiqueta, permitiéndole así, realizar predicciones sobre que etiqueta corresponde a los datos que se han introducido.

Cabe mencionar que estos no son los únicos métodos, ya que también los hay semisupervisados o aprendizaje reforzado entre otros. En cada uno de ellos los algoritmos y la forma de trabajar son diferentes. En el caso concreto de este trabajo, se emplea la técnica del aprendizaje supervisado.

3.3. RED NEURONAL ARTIFICIAL

Como ya se ha mencionado, las redes neuronales son una de las técnicas más empleadas en el ámbito del *machine learning*, destacando por su gran aplicación en el reconocimiento y análisis de datos. Como su propio nombre indica, estas redes pretenden emular el funcionamiento y la capacidad de interpretación que tiene lugar en el sistema nervioso, de tal forma que sean capaces de procesar los diferentes datos de entrada.

En este caso, la unidad básica de estas estructuras es el perceptrón, el cual no es más que un clasificador binario capaz de discernir entre dos categorías.

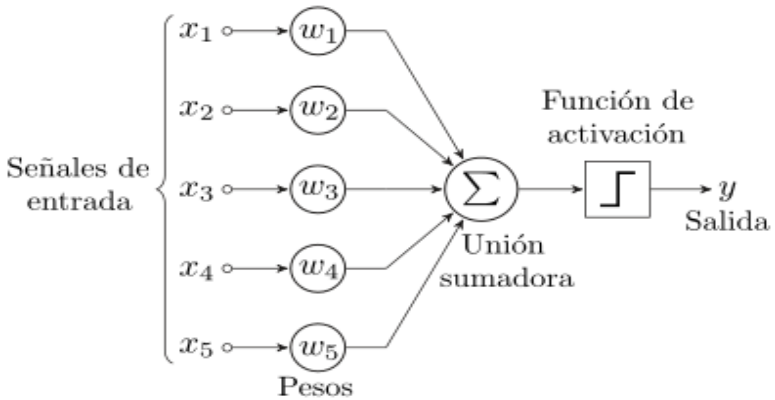

Figura 2: Diagrama de un perceptrón elemental [33]

De forma similar a lo que ocurre en una neurona biológica, el perceptrón evalúa cada una de las señales de entrada sometiéndolas a sus pesos sinápticos, de esta forma hay señales con prioridad sobre otras. A continuación, se realiza una suma ponderada y dependiendo de la función de activación que posea este perceptrón, emitirá una salida u otra. Si aumentamos el número de perceptrones, así como el número de capas en los que se dividen, se obtiene un perceptrón multicapa, que no es sino una red neuronal.

Desde el punto de vista matemático y computacional, se puede entender una red neuronal como una proyección algebraica de vectores de entrada x, de dimensiones arbitrarias, a vectores de salida y, también de dimensiones a definir. Luego, se puede definir la salida de una red neuronal de la siguiente forma:

$$y = f(x) \text{ donde } x \in \mathbb{R}^k,\ y \in \mathbb{R}^{k'};\quad k, k' \geq 1$$

En el mundo de las redes neuronales, existe una gran variedad de arquitecturas y topologías, las cuales permiten al diseñador adaptar su red a sus necesidades, de acuerdo, al problema que trate de resolver o la complejidad de este, los datos con los que se trabaje, o el tiempo de procesamiento del que se disponga. Cierto es que, a mayor número de capas y neuronas, se logran alcanzar un mayor grado de abstracción, no obstante, supone un mayor coste computacional.

No existe ninguna fórmula, o normativa, que especifique qué tipo de red hay que emplear para la resolución de cada problema, por lo que el diseñador de la red deberá seguir un proceso de prueba y error hasta dar con la solución que mejor se adapte a los resultados deseados.

No obstante, todas las redes tienen en común el tipo de capas que la conforman de acuerdo con su posición:

Capa de entrada: Es la capa inicial, y por tanto solo hay una. Es por donde entran los datos iniciales, por lo que deberá haber tantas neuronas como número de datos. Es interesante recalcar que, en el caso de trabajar con imágenes, los datos de entrada son los pixeles de cada imagen, lo que a nivel computacional supone realizar una transformación matriz-vector del conjunto de pixeles.

Capa de salida: Es la encargada de devolver los datos finales, y nuevamente, solo hay una capa de este tipo. La capa de salida tendrá tantas neuronas como salidas se deseen. En el caso de un clasificador de imágenes, como el desarrollado en este trabajo, la capa de salida tendrá tantas neuronas como categorías en las que se quiera clasificar.

Capas ocultas: Al contrario de lo que ocurre con las dos capas anteriores, estas capas no son imprescindibles, por lo que el número de estas, así como las neuronas que las conforman variarán según las necesidades del problema. Estas capas tienen la función de procesar los datos para la obtención de los resultados deseados.

Las operaciones que se llevan a cabo en una red neuronal son un conjunto de cálculos lineales y no lineales. Los datos que se obtienen a la salida de cada neurona, que así es como se llama a los diferentes perceptrones de tal forma que se sigue la analogía con las redes biológicas, se logran gracias a una operación lineal del conjunto de neuronas de la capa anterior y una posterior aplicación de una operación, generalmente no lineal, conocida como función de activación, $A(*)$, sobre la cual se profundizará a continuación. De este modo, se puede definir las operaciones realizadas en cada una de las neuronas de la siguiente manera:

$$h_x^i = A(w^n * X + b^n) \text{ donde } X \in \mathbb{R}^k;\ h_x^i, b^n \in \mathbb{R}^{k'};\ w^n \in \mathbb{R}^{k*k'}$$

Donde w^n hace referencia a la matriz de pesos y b^n al vector bias, de ambos se hablará más adelante. Nótese, que el vector X, no es necesariamente el vector de entrada de la red neuronal, sino que representa cualquier vector de salida de las diferentes capas que conforman la red, a excepción del vector de salida.

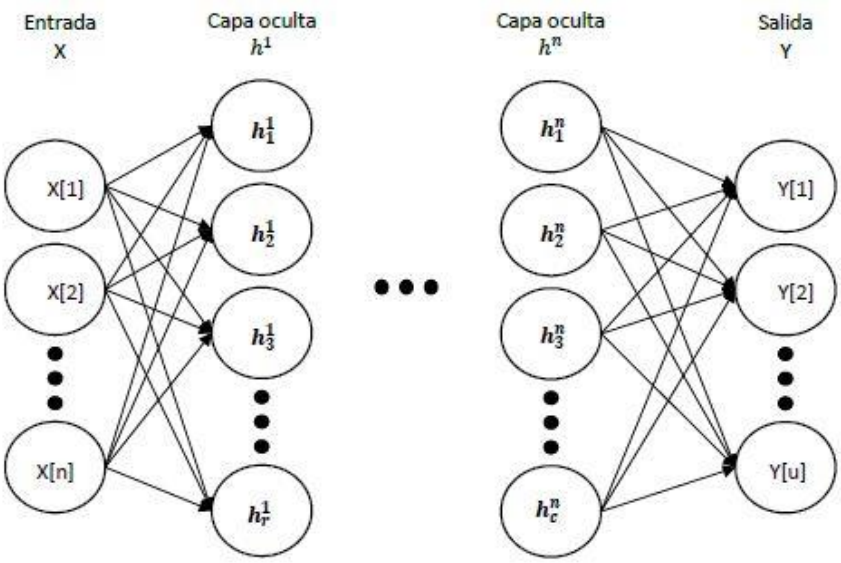

Figura 2: Esquema de una red neuronal genérica [26]

3.3.1. FUNCIONES DE ACTIVACIÓN

Como ya se ha mencionado previamente, las funciones de activación son operaciones que se aplican de acuerdo con los pesos y entradas recibidas para obtener el valor de las salidas. Estas funciones, pretenden simular el umbral de activación de las señales eléctricas en una red biológica. No obstante, existen gran variedad de funciones que permiten definir el comportamiento de las neuronas de acuerdo con las necesidades del diseñador. Algunas de las más empleadas son: [27] [28]

- Función Sigmoide: Es una función no lineal. Se caracteriza por devolver un valor entre 0 y 1. En este caso, los valores altos tienden de manera asintótica a 1 y los valores bajos tienden a 0.

Ofrece un buen rendimiento como función de activación en la capa de salida, pero muestra limitaciones si se emplea de forma recurrente en otras capas. La función sigmoide se formula tal que:

$$A(x_i) = \frac{1}{1+e^{-x_i}} \text{ donde } x_i \in \mathbb{R}^k$$

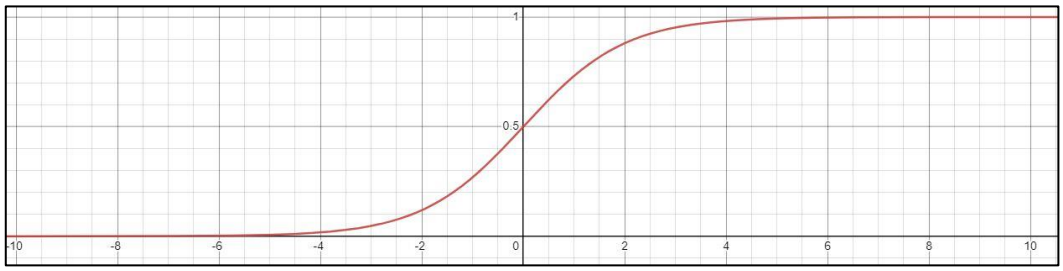

Figura 4: Función Sigmoide

- Función Tangente Hiperbólica: Es otra función no lineal. Tiene un comportamiento muy similar a la función sigmoide por lo que sus características y desempeño también se asemejan. En este caso la diferencia entre ambas es que en este caso los valores bajos, tienden de manera asintótica a -1 y no a 0 como lo hacían en la sigmoide. Se puede representar tal que:

$$A(x_i) = \frac{2}{1+e^{-2x_i}} - 1 \text{ donde } x_i \in \mathbb{R}^k$$

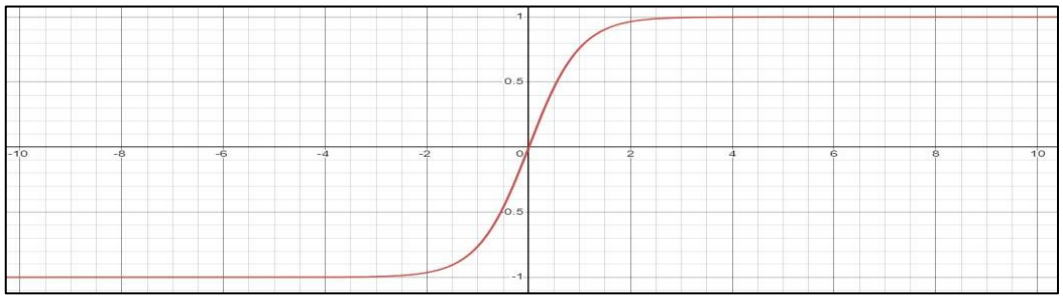

Figura 5: Función Tangente hiperbólica

- Función *ReLU (Rectified Lineal Unit)*: Función no lineal que se tiende a emplear en redes de grandes dimensiones. Esta función transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como están. Es decir, devuelve el valor máximo entre el valor introducido y 0. La función se puede expresar de la siguiente forma:

$$A(x_i) = max(0, x_i) = \begin{cases} 0, & x_i < 0 \\ x_i, & x_i \geq 0 \end{cases} \text{ donde } x_i \in \mathbb{R}^k$$

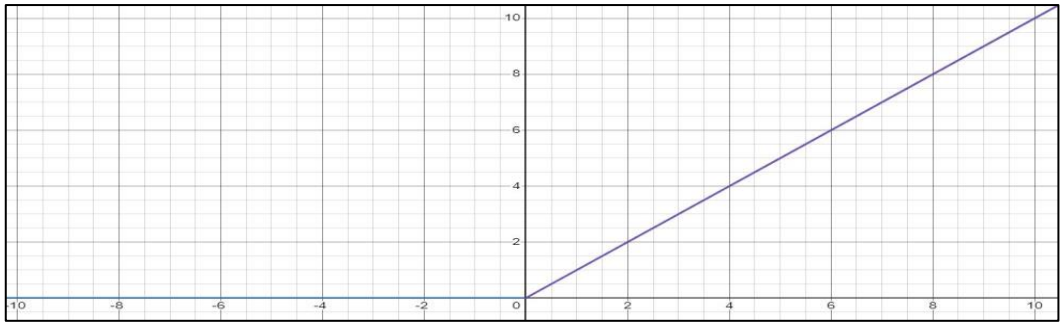

Figura 6: Función ReLU

- Función *Linear*: Al contrario que la mayoría, esta es una función lineal. Devuelve el valor del vector introducido al cual se le aplica esta función. Para ciertos problemas, este tipo de función se emplea a menudo en la capa de salida, ya que, al contrario que las anteriores, esta función no limita los valores positivos a la salida.

$$A(x_i) = x_i \text{ donde } x_i \in \mathbb{R}^k$$

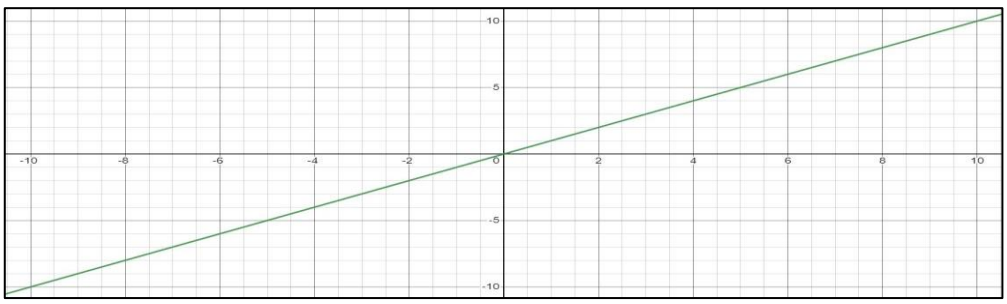

Figura 7: Función Linear

- Función *Softmax*: De nuevo, una función no lineal. Esta función guarda ciertas similitudes con la función Sigmoide. Resulta muy útil en problemas de clasificación de varias clases, por lo que no es representable en dos dimensiones. La función softmax transforma las salidas en representación en forma de probabilidades.

$$A(x_i) = \frac{e^{x_i}}{\sum_{\forall i} e^{x_i}} \text{ donde } x_i \in \mathbb{R}^k$$

3.3.2. FUNCIONES DE COSTE Y BACKPROPAGATION

Una vez ya se ha modelado la arquitectura de la red con sus capas, número de neuronas, funciones de activación..., es el momento de entrenar la red para que sea capaz de lograr el objetivo para la que haya sido diseñada, y es durante esta etapa de entrenamiento donde tiene lugar ese aprendizaje que se ha mencionado en el apartado 3.2. Este aprendizaje viene determinado por las funciones de coste, por lo que es sumamente importante su correcta elección.

Estas son funciones matemáticas que definen la diferencia entre el resultado obtenido por la red a la salida de esta, y el resultado que se desearía haber obtenido, por lo que esta diferencia, a la que a partir de ahora se le hará referencia como el error, deberá ser minimizada.

A este proceso se le conoce como Backpropagation, o regresión, y el objetivo es reducir al mínimo posible el error producido por la red, de esta forma, cuanto menor sea el error, más habrá aprendido la red, y, por tanto, mejor rendimiento desempeñará. Dicho de otra forma, se busca converger en el mínimo error, no obstante, pueden surgir inconvenientes como, por ejemplo, la convergencia en un mínimo local, por lo que no se estará consiguiendo el mejor resultado posible. Hay diferentes parámetros que se pueden ajustar para evitar estos inconvenientes, sobre los cuales se comentará más adelante.

Al igual que con las funciones de activación, existen una gran variedad de funciones de coste, las presentadas a continuación son dos de las más empleadas actualmente: [29]

- RMSE (*Root Mean Square Error* o Mínimo Error Cuadrático Medio): Es la función más empleada en problemas de regresión. Es una medida que se calcula como la raíz cuadrada media de los residuos, entendiendo los residuos como la diferencia entre el valor obtenido y el deseado.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

- *Binary Cross-Entropy* (Entropía Cruzada Binaria): Sigue una estructura bastante diferenciada a la anterior, es una medida de precisión para variables binarias por lo que se destaca en problemas cuyo objetivo es la clasificación en diferentes clases. Al ser el resultado una medida de precisión, la salida estará comprendida entre 0 y 1.

$$\mathcal{L}(\theta) = -\frac{1}{n} * \sum_{i=1}^{n} y_{i * \log(p_i + (1-y) * \log(1-p_i))}$$

Recapitulando, los datos se introducen en la red, son sometidos a diferentes pesos y acaban en la capa de salida donde las funciones de coste indican como de bien se ha realizado la aproximación al objetivo deseado. A partir de aquí, entra en juego la técnica conocida como '*Backpropagation*', o retro propagación, la cual recorre la red neuronal en sentido contrario, del fin al inicio, y teniendo en cuenta el error cometido se encarga de ir reajustando los parámetros del modelo con el objetivo de que durante la siguiente iteración se consiga reducir el error final. Luego, de esta forma se cierra el ciclo de aprendizaje de una red neuronal, que como ya se ha mencionado, es totalmente autónomo ya que es la propia red la que se reajusta para minimizar el error y obtener el mayor desempeño posible.

3.3.3. PESOS SINÁPTICOS, VALORES BIAS, Y OPTIMIZACIÓN

En una red neuronal, no todas las neuronas de una capa interactúan de la misma forma con las neuronas de la capa siguiente. Esta intensidad o grado de interacción viene determinado por la matriz de pesos, a la cual previamente se le ha hecho referencia como w^n. En una red de tipo convencional, pese a que existe la opción de que no todas las neuronas interactúen entre sí, sí que todas tienen conexión, en este caso concreto el peso asociado seria de 0. De esta forma el tamaño de esta matriz viene determinado por la relación entre dos capas consecutivas, es por ello por lo que, si una capa tiene M neuronas y la siguiente N neuronas, la matriz de pesos entre estas dos capas será del tamaño $M\ x\ N$.

Por otro lado, también se encuentra el vector *bias*, b^n, el cual está formado por constantes pertenecientes a cada una de las neuronas de una capa, por lo que el tamaño de este vector se corresponde con la cantidad de neuronas de cada capa. Estas constantes influyen e interactúan con la función de activación, modificando así el valor de salida que se obtiene a la salida de cada neurona.

Es importante destacar que son estos dos conjuntos de parámetros los que con cada iteración se van reajustando de acuerdo con la función de coste y la técnica de *Backpropagation* recientemente definida, con el fin de obtener los mejores resultados. Luego, para obtener una red de calidad que cumpla con los requisitos deseados, esta ha de encontrar la combinación de parámetros que mejor se adapte.

No obstante, esta modificación de parámetros no se realiza de forma aleatoria, ya que encontrar una solución correcta con tantas variables sería prácticamente imposible. Para ello se emplean diferentes algoritmos de optimización que conforme se realizan diferentes iteraciones, permiten la aproximación a la solución ideal. Existe una gran variedad, donde cada algoritmo aporta unas ventajas diferentes, ofrece mejores resultados de acuerdo con el problema tratado, o se implementa de una forma diferente. Aun así, todos ellos parten de la idea común del uso de la variación del error respecto a la variación de los parámetros como en el algoritmo del descenso del gradiente, expresado de la siguiente forma:

$$X_{n+1} = X_n\ -\ \gamma * \frac{\partial E}{\partial X_n}$$

Como se puede observar, tanto en el descenso del gradiente como en el resto de los métodos de optimización, existe un parámetro común a todos, el llamado *learning rate*, o ratio de aprendizaje (γ). Como su nombre indica, este valor indica con qué intensidad o de qué tamaño es el incremento que se realiza en los parámetros en la busca del mínimo error y por tanto para la obtención de una mayor precisión en el modelo.

De esta forma, un *Learning rate* reducido garantiza la convergencia a un mínimo (no necesariamente el mínimo absoluto), no obstante, este bajo valor provocaría un avance lento, requiriéndose así un largo tiempo de entrenamiento. Por el lado contrario, un valor demasiado elevado, nos acercaría rápidamente a las proximidades del mínimo deseado, pero a partir de ese momento, la falta de precisión provocaría que no sea posible acercarnos con exactitud al punto óptimo, además podría llegar a ocurrir que el algoritmo nunca llegara a converger, siendo así imposible la obtención del mejor modelo posible. Por lo que, se debe elegir con precisión el valor de este parámetro de forma que se produzca un avance rápido, pero conservando la precisión necesaria, ya que como se observa, la ratio de aprendizaje es uno de los parámetros más relevantes en el aprendizaje de una red neuronal.

Otra de las alternativas existentes, seria definir un *Learning rate* adaptable, siendo elevado en las primeras iteraciones otorgando una rápida aproximación, pero pasando a un valor reducido en las iteraciones finales para una mayor precisión.

Finalmente, mencionar la inicialización de los parámetros, ya que como se ha comentado estos se modifican tras cada iteración, no obstante, deben comenzar con cierto valor. Existen múltiples opciones, como podría ser inicializarlos todos en 0, o en 1, inicializarlos de forma aleatoria u otra opción seria, partir de los valores de un modelo que ya haya sido entrenado previamente.

3.3.4. BATCH SIZE Y NÚMERO DE EPOCH

Tal y como se viene explicando, la función de coste pretende minimizar el error con cada iteración. Es decir, con cada iteración, todos los datos pasan a través de la red y son sometidos a la función de costes, siempre con el objetivo de reajustar los parámetros de la forma más óptima posible. No obstante, si en cada iteración se emplean todos los datos disponibles individualmente, se podría obtener una mejor precisión, pero sería un proceso demasiado lento. Para solventar este inconveniente nace la idea de los *batches,* o lotes. Estos no son más que pequeñas agrupaciones de datos en las que se divide el conjunto total de los datos. De esta forma, en cada iteración el número de ajuste de los parámetros del modelo es mucho menor, por lo que los tiempos de entrenamiento también se reducen. Por el contrario, al trabajar con menos ajustes la precisión a la hora de optimizar el error podría ser menor.

Si bien en términos globales, con conjuntos de datos sin errores de etiquetado, el tamaño de lote no es algo que repercuta en gran medida en el resultado global, una buena selección de este puede agilizar los tiempos durante los procesos de aprendizaje, por lo que es un valor para tener en cuenta.

Por otro lado, se encuentra el parámetro conocido como, *epochs*, o épocas. Este valor indica el número de iteraciones que se van a llevar a cabo. Dicho de otro modo, se entiende que se ha completado una *epoch* cuando todos los datos disponibles han sido tratados por la red. Es necesario recalcar ahora la diferencia entre una *epoch*, y una iteración, ya que dada la existencia de los *batches*, una iteración, es

cuando uno de estos ha recorrido toda la red, mientras que la *epoch*, es cuando se acaba empleando todo el conjunto de datos.

De esta forma, dentro de cada *epoch*, se pueden llevar a cabo diferentes iteraciones o ajustes de los parámetros. Al mismo tiempo, si se conoce el tiempo que requiere completar una *epoch*, variando el número de *epochs*, se puede variar y determinar cuánto será el tiempo de entrenamiento.

Combinando estos dos parámetros podemos definir diferentes tipos de entrenamiento:

Entrenamiento estocástico (*Stochastic training*): La red toma de forma aleatoria uno de los datos, a partir del cual calcula el error, y se reajustan los parámetros de acuerdo con este. Cada época del entrenamiento constará de tantas iteraciones como datos de entrada se dispongan.

Entrenamiento en lotes *(Batch training*): En este tipo de entrenamiento, se calcula el error para cada dato de entrada, y de acuerdo con la suma de estos se obtiene el error total, el cual se emplea para actualizar el valor de los parámetros. Obtiene mejores resultados que el anterior ya que reduce el error empleando todos los datos disponibles, a pesar de esto, supone un mayor costo computacional, ya que en cada época se emplean todos los datos en una sola iteración, por lo que es necesario un gran número de épocas, para lograr resultados adecuados.

Entrenamiento en lotes pequeños (*Mini-Batch training*): En vez de emplear todos los datos disponibles, el conjunto de datos se divide en pequeños lotes con los que se calcula el error y ajustan los pesos. Como ya se ha mencionado previamente, es necesario elegir correctamente el tamaño de estos lotes, y llegar a una solución de compromiso entre la precisión a la hora de optimizar los parámetros de la red, y el tiempo o coste computacional que se requiere. Notar que el número de iteraciones requeridas por épocas será, el número total de datos entre el tamaño de lote.

3.4. REDES CONVOLUCIONALES

Estas técnicas y métodos explicados hasta ahora ofrecen un gran rendimiento cuando se trabaja o analiza datos numéricos. No obstante, a la hora de trabajar con imágenes, es necesario realizar ciertas variaciones al modelo de la red para lograr obtener la máxima información que este tipo de datos nos ofrece. Una imagen se compone de pixeles, de los cuales es importante conocer su valor, así como se relacionan y de qué manera interactúan con los de alrededor, es decir, si existen patrones detectables que permitan identificar un tipo de imagen concreto.

Para realizar este tipo de detecciones surgen las redes convolucionales. Pese a tener nombre propio, este tipo de redes no es sino una variación de las mencionadas hasta ahora, las cuales se destacan por aportar grandes beneficios en tareas de visión artificial como son la clasificación o detección de imágenes.

La principal ventaja que aportan es que mientras una red completamente conectada trabaja con cada uno de los pixeles, lo cual, cualquier variación en el dato de entrada alteraría el resultado obtenido, una red convolucional, primero trabaja con la imagen como conjunto para obtener patrones o características distintivas, las cuales se emplean a posteriori para el proceso de clasificación. Dentro de las redes convolucionales se encuentran diferentes tipos de capas entre las que destacan:

Capa convolucional: En esta capa la imagen de entrada es sometida a diferentes filtros, los cuales se aplican estudiando píxel a píxel, así como sus vecinos, mediante núcleos (o *kernel*) que pueden variar de tamaño en función de las necesidades del problema. Los diferentes filtros o transformaciones que se realizan se obtienen variando el valor y el peso que se asigna a cada píxel dentro del *kernel*, es decir son matrices cada una diferente a la anterior cuyo objetivo es la obtención de nuevas imágenes donde resulte más sencillo destacar patrones o características.

Hay que destacar que, al igual que ocurre en una red completamente conectada, estos parámetros o pesos de cada uno de los filtros se ajustan de manera automática durante el proceso de entrenamiento, de esta forma aquellos filtros que no hayan aportado gran utilidad al proceso se recalculan para obtener características diferentes.

La cantidad de filtros que se desean aplicar es un valor que se puede definir al crear la capa durante la etapa de diseño. Como observación, en el caso de que la imagen de entrada esté a color, se podrá aplicar la misma transformación a cada uno de los tres canales que conforman los colores básicos (*RGB, Red Green Blue*), por lo que la red estará realizando el triple de las convoluciones establecidas. No obstante, el resultado obtenido al final de la capa podrá combinar las tres salidas tras aplicar un filtro unidimensional, por lo que sí que se acaba obteniendo la cantidad predefinida.

Cabe mencionar que igual que se aplican capas convolucionales a elemento de dos dimensiones como es el caso de las imágenes, también se podrían aplicar a volúmenes de tres dimensiones o incluso más. De esta forma, al incluir un mayor número de dimensiones, surgen patrones espaciales que permiten obtener modelos de aprendizaje mucho más robustos, aunque, por el contrario, el costo computacional es mucho más elevado, así como el espacio de memoria debido a la gran cantidad de parámetros con los que se trabaja. De igual modo, pese a ser imágenes el objeto de trabajo de este proyecto, se han empleado filtros en 3 planos diferenciados como se indicará más adelante.

Un factor que hay que tener en cuenta durante el proceso de convolución es que cuando se emplean filtros de dimensiones mayores que $1\ x\ 1$, se reduce el tamaño de la imagen a causa de los bordes de la imagen. Esto se debe a que, al aplicarse estas matrices píxel a píxel, y a los colindantes, como los pixeles de los bordes no tienen vecinos, estos no se emplean como centro de la matriz del filtro. No obstante, como se explicará más adelante, esta reducción de tamaño no es necesariamente una desventaja.

Capa de submuestreo o agrupación (*Pooling Layer):* Esta capa tiene como datos de entrada los resultados de la convolución, y tiene como objetivos, la reducción del tamaño de imagen, así como del número de parámetros calculados y destacar las características más importantes que encuentre en cada imagen. Hay varios tipos de submuestreo en la capa de agrupación, pero la más empleada actualmente es la llamada agrupación máxima (*maxpooling*). En este caso, se va reduciendo la imagen empleando una matriz de un tamaño que puede definir el diseñador, de este conjunto de pixeles que se encuentra en la matriz se escogerá el de mayor valor, descartando el resto.
También existen otros tipos como el muestreo por mínimos (*minipooling*), o el muestreo de media (*average pooling*), no obstante, es fácilmente demostrable que el método por máximos es el que mejor resultados ofrece en las redes convolucionales.
Mientras que, en la capa convolucional, la matriz de filtros avanzaba píxel a piel, en este caso avanza según el tamaño de la matriz. En este caso, el entrenamiento de la red no varía ningún parámetro de estas capas, sino que permanecen todos constantes.

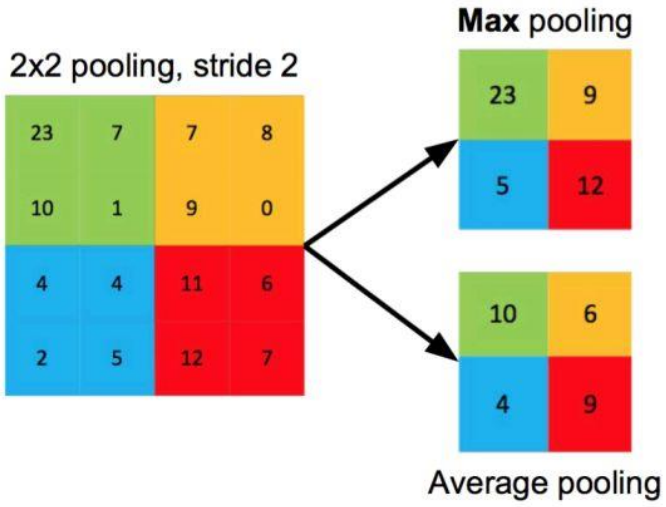

Figura 8: Ejemplo capas de agrupación máxima y agrupación de media

Como se ha mencionado previamente, el dato de entrada a una red neuronal debe ser un vector de pixeles, pero gracias a las capas de agrupación, y la correspondiente reducción del tamaño de imagen, esta transformación matriz-vector resulta mucho más sencilla, es por este motivo por lo que la reducción durante la convolución a causa de los bordes de la imagen puede ser un tema interesante, no obstante, no es recomendable utilizar filtros muy grandes ya que si se elimina demasiado borde, se estará perdiendo demasiada información.

Capa de interpolación (*Upsampling layer*): El proceso que se lleva a cabo es el contrario que se realiza en las capas de agrupación, es decir, en esta capa se aumentan los pixeles de las imágenes con las que se trabaja. La idea de este concepto es poder obtener al final de la red una imagen con las mismas dimensiones de entrada, lo cual es un aspecto interesante en tareas de segmentación.

Capa completamente conectada (*Fully-connected layer*): Esta es una capa generalmente empleada en labores de detección o en procesos de clasificación de imágenes. Se suelen ubicar estas capas al final de la red, reciben como dato de entrada las imágenes obtenidas tras uno o varios procesos de convolución, evalúan los valores de los píxeles que conforman cada una de las imágenes, y en función de estos, devuelven un único valor por imagen. Este valor representa la probabilidad que tiene esta imagen de pertenecer a una de las diferentes categorías en las que se quiera clasificar. Estas capas son las que conforman lo que comúnmente se conoce como perceptrón multicapa.

4. TRABAJO DESARROLLADO

Como se puede observar, la *Figura 9* es una representación visual de los pasos que hay que llevar a cabo en el diseño de una red neuronal independientemente de su funcionalidad final. Al mismo tiempo sirve de introducción al trabajo que se ha que realizado durante este proyecto. En primer lugar, se determina bajo qué criterios y qué arquitectura se ha diseñado la red neuronal, así como las diferentes funciones que la definen. A continuación, se establecen los hiperparámetros para el correcto entrenamiento del modelo con el fin de obtener unos niveles de aprendizaje satisfactorios. De esta forma, constituido el modelo del clasificador se procede a su entrenamiento, y una vez finalizado este, se evalúan los resultados. Si estos valores obtenidos distan de los deseados se procede a realizar las modificaciones o bien de la estructura de la red o bien de los hiperparámetros de entrenamiento. No obstante, si se da con una estructura de red adecuada, solo se harán reajustes en los hiperparámetros de entrenamiento. Es por ello por lo que a lo largo de este capítulo y el siguiente, *4. Trabajo desarrollado y 5. Análisis de los resultados*, se procede a describir cómo se ha diseñado, implementado y validado el modelo de un clasificador de imágenes mediante redes neuronales.

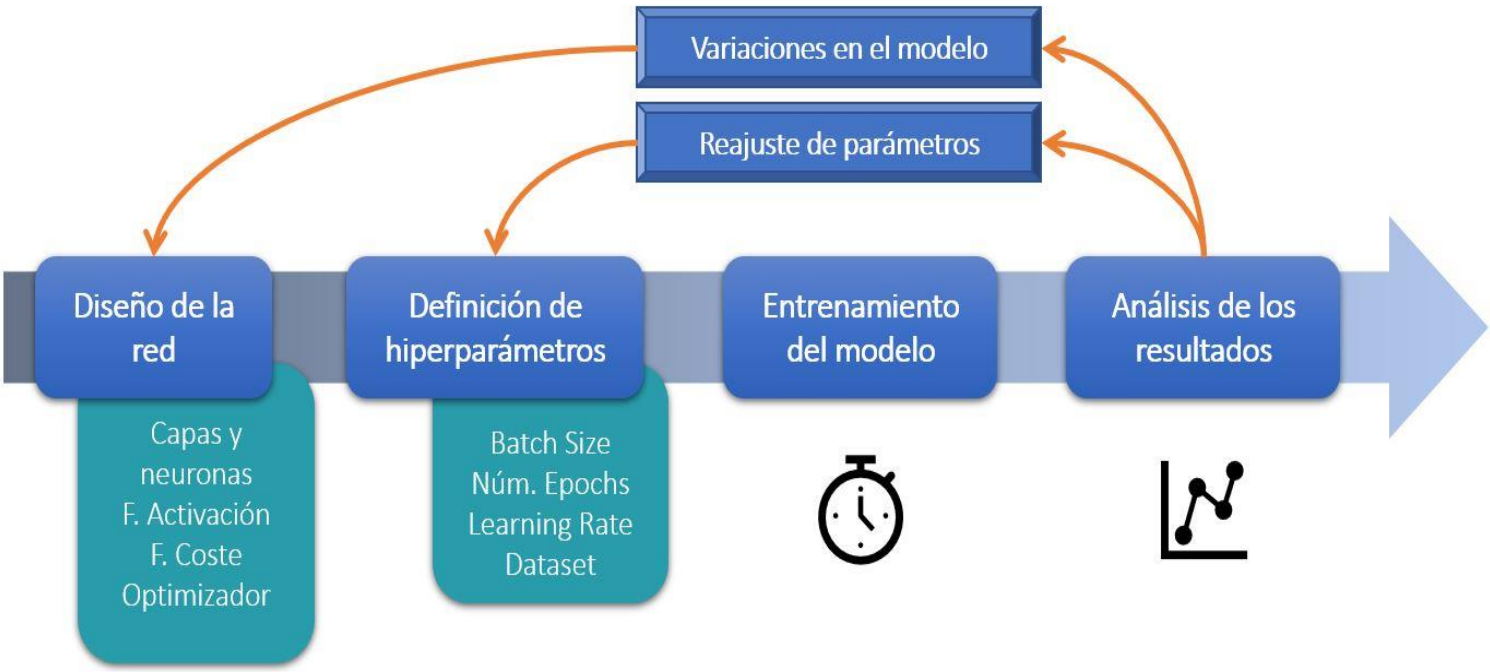

Figura 9: Representación gráfica del trabajo elaborado

4.1. CRITERIOS PARA ELECCIÓN DEL MODELO

Para la realización de este trabajo se ha empleado un dataset, o conjunto de datos, compuesto por 20000 imágenes diferentes de células de la sangre. Estas células se pueden dividir en dos categorías células infectadas de los parásitos de la malaria, y células sanas en buenas condiciones. Al mismo tiempo, este dataset contiene etiquetas para todas las imágenes las cuales permiten identificar y diferenciar de qué tipo de células se trata. De esta forma, combinando estos dos tipos de datos, se ha llevado a cabo un aprendizaje supervisado como se explicó en el Apartado 3.2.

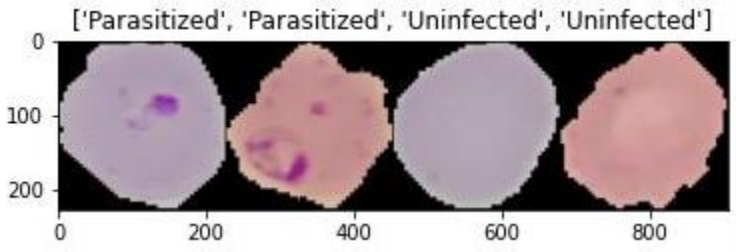

Figura 10: Ejemplo de las células empleadas para el diseño del modelo

Tal y como se mencionó en los objetivos del trabajo, se busca implementar un clasificador que permita la distinción y clasificación de estos dos tipos de células, con el mejor rendimiento posible. No obstante, tras la creación del modelo se requiere algún tipo de prueba o validación que demuestre que el modelo funciona correctamente. Con este objetivo en mente, previo a cualquier otro paso, se lleva a cabo la división del dataset en dos conjuntos. El primer conjunto representa el 70% del total de imágenes, las cuales serán empleadas en el entrenamiento del modelo del clasificador.

El segundo conjunto, el 30% restante, está conformado por las imágenes que se emplearan tras el entrenamiento del modelo, para la validación de este.

Esta separación empleada no ha de ser exacta, aun así, sí que es recomendable una proporción 70-80% del total para el entrenamiento de la red y 20-30% para la validación. Esto se debe a que, si se emplea un conjunto de entrenamiento reducido, puede ocurrir que la red no aprenda todo lo que debería, suponiendo así un clasificador de dudosa calidad. Por el contrario, un conjunto de entrenamiento excesivo supondría que resta un número reducido de datos para la validación, por lo que, a pesar de haberse realizado un buen entrenamiento, podría no verse reflejado en los resultados al tener pocos datos para la validación del modelo. Recalcar también, que independientemente de la relación que se escoja, en ambos conjuntos debe haber la misma cantidad, aunque sea de forma aproximada, de datos de cada categoría a clasificar.

De esta forma, a lo largo del desarrollo del clasificador se obtienen dos valores relevantes que permiten comprobar la validez de este. Por un lado, se encuentra la precisión del entrenamiento, este porcentaje indica como de bien se está llevando a cabo el entrenamiento del modelo, no obstante, como se

detallará más adelante, hay que tener cuidado con este valor ya que un alto porcentaje no es necesariamente sinónimo de éxito. Por el otro lado, está la precisión de la validación, que indica lo bien que clasifica el modelo empleando unas imágenes que no ha visto durante el entrenamiento. Es por ello por lo que el indicador de lo bueno que es el clasificador es está segunda precisión.

De esta forma, todos los parámetros que se han explicado hasta ahora, y como se hará a continuación, han sido elegidos con el fin de maximizar este valor en la medida de lo posible, siempre con la finalidad de obtener el clasificador con la mejor calidad.

4.2. ARQUITECTURA DE LA RED

El proceso de creación, entrenamiento y validación de la red neuronal se ha llevado a cabo, empleando la herramienta gratuita de Google Colab, la cual permite conectarse con una GPU, facilitando así el cálculo matricial que realiza la computadora. Esto es muy interesante ya que al fin y al cabo las imágenes son matrices de pixeles con lo que, si se logra más velocidad con este tipo de cálculos, se acelera el procesamiento de las imágenes, dando como resultado menores tiempos en el entrenamiento de las redes. Además, se ha empleado el lenguaje computacional conocido como Python, así como librerías de OpenCV y Pytorch, la cuales se destacan por su utilidad en el ámbito del proceso de imágenes.

Durante la etapa de diseño de la red se ha buscado la máxima precisión posible, valorando y analizando los resultados que ofrecían las diferentes alternativas que se han estudiado. Al estar trabajando con imágenes como datos de entrada, se ha decidido introducir una serie de capas convolucionales previas a la etapa del perceptrón multicapa, con el fin de obtener la mayor cantidad de patrones a analizar, de esta forma el modelo creado quedaría de la siguiente forma:

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=6400, out_features=3000, bias=True)
  (fac1): ReLU()
  (fc4): Linear(in_features=3000, out_features=2, bias=True)
)
```

Figura 11: Implementación de la red diseñada en Python

Como se puede apreciar, se ha aplicado la técnica ya mencionada en el Apartado 3.4., en la cual se aplican varias capas de convolución tras varias capas de submuestreo. De esta forma, se observa como en cada etapa convolucional se obtiene el doble de mapas de características que en la etapa anterior.

También hay que destacar que el tamaño de la matriz de filtros seleccionada es de dimensiones $3\ x\ 3$, ya que es un tamaño suficiente para los objetivos buscados, se podrían emplear tamaños más grandes, pero esto aumentaría el coste computacional. Como ya se introdujo en el apartado teórico al emplear este tamaño de matriz, se puede considerar que se está perdiendo información, aun así, esto no supone ningún problema ya que no es una pérdida crítica. De hecho, este fenómeno concuerda con el objetivo perseguido al emplear capas de *Pooling*, que no es otro que reducir la información con la que se trabaja.

Igualmente, si fuese necesario se podría aplicar técnicas de *Padding* y se evitaría este inconveniente, pero en este caso concreto, por lo recientemente expuesto, no es necesaria su aplicación.

Tal y como se ha mencionado, tras cada capa convolucional, sigue una capa de submuestreo, que, al emplear imágenes como datos de entrada, dos dimensiones son suficientes. El método de agrupación empleado en estas capas es a partir de los máximos (*maxpooling*), destacando así las características más interesantes. Para la agrupación de pixeles se ha empleado una matriz de tamaño $2\ x\ 2$, en estos casos es recomendable evitar matrices de grandes tamaños ya que al prevalecer solo el máximo y descartar el resto, se podría estar descartando valores también influyentes en el conjunto.

Si bien es cierto que, con dos etapas de convolución ya se obtienen buenos resultados, al añadir un par más, el modelo se ha mostrado mucho más sólido, alcanzando una alta precisión con facilidad. Es por ello por lo que se ha decidido añadir estas dos últimas etapas convolucionales, las cuales provocan un aumento en la duración del entrenamiento de la red, pero que se compensa con el notable aumento de precisión obtenido.

Para el diseño del perceptrón multicapa se han realizado diferentes pruebas variando el número de capas ocultas del mismo. Con los valores obtenidos en estos entrenamientos, los cuales se analizarán más en detalle en el apartado de *Resultados*, finalmente se ha decidido emplear solo dos capas, que se podrían considerar una capa de entrada y otra de salida. Es decir, vistos los resultados, no es necesario añadir capas ocultas al modelo ya que, al hacerlo, no se obtiene una gran mejoría, pero sí que se añaden más parámetros con los que se trabajan siendo así una propuesta más ineficiente.

Para la capa de entrada del perceptrón, se ha empleado la función de activación *ReLU*. Esta decisión se ha tomado ya que es la función que mejores resultados ofrece en problemas de clasificación ya que ante entradas negativas se producen salidas con valor cero lo que simplifica el modelo y acelera la etapa de entrenamiento. Así mismo, también se destaca por su buen comportamiento con el tratado de imágenes y un buen desempeño en redes convolucionales.

Por otro lado, hay que aclarar que es necesario una buena elección de las funciones de activación en las capas internas de la red ya que una mala elección puede ocasionar problemas durante los entrenamientos complicando así el aprendizaje.

Esto no es así en la capa de salida, ya que al ser la última y no haber ninguna capa a continuación, esta elección es menos delicada que en las anteriores por lo que se ha decidido emplear la función de activación *Linear* la cual es mucho más sencilla de procesar.

A continuación, es necesario definir los mecanismos que se van a llevar a cabo para el aprendizaje de la red, así, como función de coste del modelo se ha empleado la Entropía Cruzada Binaria, que como ya se mencionó en el apartado teórico, es la más empleada en problemas de esta índole, aportando grandes resultados cuando se trata de clasificar entre diferentes categorías de imagen.

Tal y como se mencionó, existen diferentes modelos para la optimización en el reajuste de parámetros. En este caso, se ha empleado el optimizador Adam, el cual destaca por ser fruto de la combinación de otros tantos optimizadores, obteniendo las ventajas de estos. Como consecuencia, este algoritmo de optimización se presenta como uno de los más sólidos de los que se disponen. Otra de las alternativas que se han empleado ha sido el algoritmo SGD, pero rápidamente se ha descartado al no poder alcanzar los resultados deseados.

Cabe mencionar que uno de los parámetros más importantes que se pueden modificar en el optimizador es el learning rate, del cual se hará un desarrollo más adelante, ya que es un parámetro que influye de manera directa en la obtención del resultado, por lo que se requiere una buena toma de decisiones para la elección de este parámetro.

Luego, combinando lo mencionado hasta ahora, ya se dispondría de una red con todo lo necesario para ser entrenada, o lo que es lo mismo, ya se puede dar por finalizado el diseño de la arquitectura de la red. A continuación, solo resta dar con la combinación correcta de hiperparámetros que permitan obtener el máximo rendimiento de la red, para ello, se decide mantener la estructura sin variación alguna, ya que, como se ha observado, podría llegar a influir, y perderíamos la noción de cuál es la causa real de bien el aumento o reducción de precisión final.

4.3. HIPERPARÁMETROS E INFLUENCIA EN EL RESULTADO

4.3.1. BATCH SIZE Y NÚMERO DE EPOCH

Un *Batch*, como ya se ha explicado previamente, es una agrupación de datos dentro del total del que se dispone. Por lo que se debe elegir de qué tamaño se quieren definir estos lotes. Hay que mencionar que a lo largo del desarrollo del proyecto se ha podido comprobar que la variación de la cantidad de datos que se incluyen en cada agrupación es uno de los parámetros menos influyentes en el resultado final de todo el modelo. Por este motivo, siempre y cuando no se empleen valores excesivamente elevados, se deberían conseguir resultados igualmente satisfactorios.

De forma más concreta, se ha comprobado que con batches de un tamaño de 64 o 128 imágenes, si bien los resultados no se pueden considerar malos, sí que son ligeramente inferiores a los obtenidos con 16 o 32 imágenes.

Esta diferencia apenas es del 2-3%, no obstante, en un modelo donde se busca la mayor precisión posible, se opta por trabajar con los parámetros que suponen una mejor combinación. Finalmente, se ha decidido emplear un "*Batch size*" de 16 imágenes por lote, aunque como se ha mencionado previamente, una elección de 32 sería también correcta.

Definido así la cantidad de iteraciones que se van a realizar en cada época, se debe definir el número de épocas que se quieren llevar a cabo. Esto es un modo aproximado de determinar cuánto tiempo se desea que dure el entrenamiento, ya que, a mayor cantidad de épocas, más durará el entrenamiento, y más riesgo hay de sufrir sobre entrenamiento, problema del que se hablará más adelante.

Sea pues, el número de épocas es un parámetro que influye en el modelo, pero con ciertas limitaciones, no de forma directa como ocurre con otros hiperparámetros. Podría ser que el modelo con el que se trabaja llegue a su máxima precisión de acierto mucho antes de completar todas las épocas, por lo que, a partir de ese momento se estaría entrenando la red en balde. Por el contrario, podría darse el caso de que el número de épocas definidas sea inferior al número que necesita el modelo para alcanzar su máxima precisión, por lo que se podría pensar que se ha hecho una mala elección en el diseño de la red, cuando realmente solo era necesario permitir un mayor entrenamiento a esta. Es decir, si el único criterio de decisión se trata del número de épocas, se debe seleccionar un valor que permita alcanzar el mejor resultado posible, sin entrenar de más el modelo.

No obstante, como se ha mencionado, existen ciertas limitaciones, y es que, en función del resto de hiperparámetros seleccionados, podría ser que se necesiten más o menos épocas, al haber hiperparámetros que permiten la aceleración del entrenamiento.

Por lo que, en primer lugar, se debería elegir la combinación de hiperparámetros que ofrecen el mejor resultado para, a continuación, ajustar el número de épocas que permiten obtenerlo.

En este caso, se ha elegido un número de 25 épocas. También se han realizado pruebas con 30 épocas, aunque estas últimas cinco no han aportado ninguna mejoría, lo cual se ve reflejado en una tendencia constante en la precisión del clasificador, por lo que se ha descartado esta opción. Otra alternativa ha sido emplear 20 épocas, pero hay ocasiones en las que la red todavía podía continuar aprendiendo ya que al finalizar el entrenamiento se observa una tendencia en la precisión ascendente.

Es preciso destacar que existen otras alternativas a la hora de definir un entrenamiento, por ejemplo, en lugar de definir un número concreto de épocas, se podría haber entrenado la red hasta que se cumpliese una determinada condición.

En este trabajo en concreto, esta opción podría ser complicada de implementar ya que se busca la máxima precisión posible, por lo que sería contraproducente entrenar la red solo hasta determinada precisión. Así mismo, sería igual de problemático forzar a la red a entrenar hasta alcanzar un 100% de precisión, ya que podría no alcanzar esa precisión, y por tanto no finalizar nunca el entrenamiento.

Es por ello por lo que se ha definido un número limitado de épocas, de este modo, el entrenamiento se lleva a cabo en un tiempo razonable, obteniendo resultados satisfactorios.

4.3.2. AUMENTO DEL DATASET Y OVERFITTING

Para llevar a cabo los entrenamientos, se ha empleado un dataset de 20000 imágenes, que es una buena cantidad para obtener buenos resultados en este trabajo, aunque es necesario destacar que para otros problemas puede ser que fuesen demasiadas o por el contrario insuficientes, a modo de ejemplo, una red de grandes dimensiones se entrena con cientos de miles de imágenes. Por otro lado, si se decide trabajar con las imágenes originales sin transformación alguna, se producen dos fenómenos a destacar.

Por un lado, obtenemos una precisión en el entrenamiento del 99,99% y, por otro lado, obtenemos un coste del entrenamiento del 0,04%. Estos valores tienen una explicación detrás, y es que, al ser todas las imágenes de características similares, la red aprende a identificarlas perfectamente. Aun así, pese a estos datos, en ningún momento se llega a obtener un resultado en la validación que los permita dar por satisfactorios.

Luego, si se quiere obtener un modelo sólido que sea capaz de clasificar correctamente cualquier tipo de imagen, así como, el mejor resultado posible se debe aumentar el dataset, aumentando la diversidad y variabilidad de las imágenes, ya que como se ha mencionado previamente, el modelo aprende estudiando las diferentes propiedades, características y patrones de las imágenes que se introducen como datos. Para la ampliación del dataset se aplican a las imágenes de entrada ciertas transformaciones que modifiquen los datos de entrada.

Para variar los datos de entrada, es decir, las imágenes, se han empleado transformaciones como, volteos tanto en vertical como en horizontal aleatorios, rotaciones de la imagen, ajustes de color…Aun así, hay muchas otras técnicas de transformación que se podrían haber aplicado. Igualmente, se ha aplicado una normalización de los datos que no debe considerarse una transformación, pero es de gran utilidad en el procesamiento de los datos. [30]

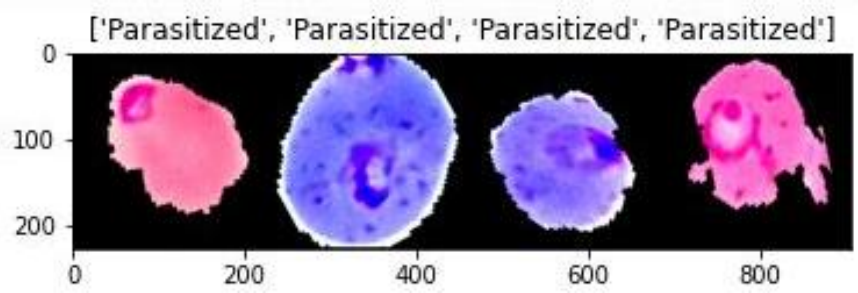

Figura 12: Ejemplo de imágenes tras aplicar las transformaciones

Hay que destacar que no aplicar ninguna transformación podría provocar un problema conocido como *"Overfitting"* o sobre entrenamiento. Este problema es fácilmente detectable cuando la precisión de los datos de entrenamiento continúa aumentando a medida que se realizan las diferentes iteraciones, mientras que la de los datos de validación llega un punto en el que permanece estancada y es claramente inferior a la de entrenamiento. Lo que ocurre es que, en cierto modo, la red está memorizando el conjunto de datos de entrenamiento más que aprendiendo, pero luego no es capaz de reflejarlo en la validación, es decir, durante el entrenamiento se está aprendiendo ruido, que debe entenderse como información no generalizable a otras imágenes. A causa de esto, es por lo que la validación no consigue alcanzar la precisión del entrenamiento.

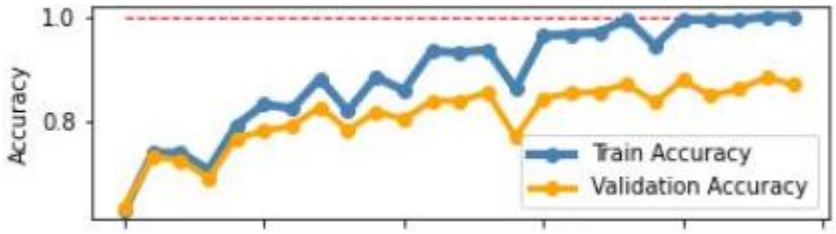

Figura 13: Caso de Overfitting

Por el lado contrario, aumentar de forma excesiva las transformaciones que se aplican a las imágenes supone un problema añadido a los existentes, ya que al aumentarlas se está aumentando la información que debe aprender la red y por tanto más información a procesar. Ante esta situación no aparecerían problemas de Overfitting, pero lo que si ocurriría sería una clara perdida de precisión del modelo.

Osease, un *dataset* de entrada con más variabilidad permite obtener un modelo más generalizable a cualquier entrada, pero tal y como se ha explicado, se debe tener en cuenta que los datos de entrenamiento y de la validación deben guardar cierta similitud, llegando así a una relación de compromiso, aumentando el *dataset* lo justo y necesario para poder aprender todas las características relevantes sin que aparezca ruido que altere el resultado.

4.3.3. LEARNING RATE

El *learning rate, o* ratio de aprendizaje, es el hiperparámetro que más influye en el avance del entrenamiento, ya que como se especificó en el apartado teórico, un valor excesivamente grande puede ocasionar problemas, pero igualmente, un valor reducido no permite alcanzar los objetivos deseados. Luego, de acuerdo con los entrenamientos realizados, la elección de un buen valor es crucial para obtener los mejores resultados.

Es por ello por lo que dar con el valor adecuado ha sido uno de los principales objetivos de este trabajo. En el apartado de Resultados, que se encuentra más adelante, se especificará todos los entrenamientos y alternativas que han sido sometidas a estudio, sin embargo, a continuación, se realizara una breve introducción.

En primer lugar, se han llevado a cabo numerosos entrenamientos con ratios de aprendizaje constantes, es decir, que permanece invariable para todas y cada una de las diferentes épocas. Además, se han realizado pruebas con diferentes valores como son, $10^{-3}, 10^{-4}$, 10^{-5} y 10^{-6}, con el fin de encontrar el valor que ofrezca mejores resultados.

Seguidamente, se ha procedido a emplear un learning rate adaptativo, es decir, que varíe en función de una determinada condición. En este caso concreto, la condición seleccionada ha sido el número de épocas, por lo que al llegar a un número determinado de épocas la ratio de aprendizaje varia.

De acuerdo con esta teoría se han implementados dos métodos diferentes. En el primero, la ratio solo varia una vez a lo largo de todo el entrenamiento, mientras que en el segundo varia en dos ocasiones. Al haberse implementado la ratio adaptativa tras el estudio de los valores constantes, se han empleado aquellos valores que mejor desempeño mostraron.

Existen otras alternativas a la hora de definir estas condiciones de cambio, por ejemplo, otra solución que se valoró fue en base al coste, o al error producido durante los entrenamientos, de esta forma, si este valor no se reduce en un determinado número de épocas, se reduce el learning rate, para aportar más precisión. Con esta alternativa, se evita que la red se quede estancada y continúe con el aprendizaje, no obstante, en la red que se ha empleado, el coste se reduce de forma natural de acuerdo con el paso de las épocas por lo que al emplear esta condición de cambio no sucedería ningún efecto.

4.4. TRANSFER LEARNING

Este tipo de redes y modelos que se han explicado hasta ahora se pueden guardar y emplear para otros proyectos, es decir, se puede aprovechar todo el conocimiento aprendido a lo largo del entrenamiento para un tipo de datos concreto, para posteriormente emplear otros datos. Este concepto es lo que se conoce como transfer learning, o transferencia de aprendizaje. [31]

A primera vista parece una técnica muy atractiva ya que te ahorra gran parte del diseño, aun así, tiene ciertas limitaciones para tener en cuenta. En primer lugar, se debe tener en cuenta que, tal y como se ha mencionado varias veces, las redes neuronales aprenden a identificar patrones propios de los datos que se introducen, por lo que una red de pequeño tamaño y con un dataset reducido, se limitará a aprender las características específicas. A modo de rápido ejemplo, un clasificador diseñado para diferenciar entre tipos de verduras tendrá problemas para clasificar entre razas de perros.

Pero para solventar este problema, existen diferentes redes ya creadas y de sencilla implementación, cuyas dimensiones son generalmente grandes, además de haber sido entrenadas con una gran cantidad de imágenes, de esta forma, las características aprendidas son básicas en la identificación de cualquier imagen por lo que son modelos fácilmente extrapolables.

Al ser estructuras ya diseñadas simplemente se debe importar el modelo, y realizar un cambio en la o las, depende de las necesidades, última capa de la red. Este cambio es importante, además de necesario, ya que es el que reajusta la red para la función específica que se desea que cumpla. Además, esta técnica es muy útil también para casos en los que no se pueda disponer de una gran cantidad de datos, porque la red ya es capaz de asimilar patrones, por lo que el aprendizaje que se necesita es mucho menor, obteniendo así, resultados igualmente buenos.

Para este proyecto se ha empleado la familia de redes conocidas como ResNet, la cual consta de estructuras de diferentes tamaños. Concretamente, se ha utilizado la más pequeña de todas ResNet18, ya que modelos más grandes como ResNet34 o ResNet50 no parecían aportar muchas mejoras.

Hay que mencionar también que estas redes se han entrenado con un formato de imagen concreto, por lo que para poder emplear el *dataset* de la malaria, previo al trabajo realizado se requiere hacer un reescalado para hacer coincidir las imágenes con el formato requerido.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 14: Estructuras de la familia de redes ResNet [32]

Dentro del mundo del *transfer learning* existen dos conceptos que son necesarios conocer para entender el funcionamiento, y extraer el máximo beneficio del aprendizaje previo.

En primer lugar, tenemos el concepto de "*Pretrained*", o preentrenamiento. Esta opción permite al usuario obtener la red con los parámetros ya calculados sin necesidad de llevar a cabo ningún entrenamiento. Si no se selecciona esta opción, entonces al implementar el modelo solo se obtiene la estructura de la red.

El segundo concepto es el de "*Freeze*", o congelar, el cual hace referencia a dejar invariables los parámetros ya calculados de la red. Esto nos permite dos opciones diferentes, o bien congelar los pesos ya entrenados y entrenar solo la última capa que se ha modificado, o bien, partiendo de los parámetros ya calculados, seguir entrenando toda la red, esta técnica se conoce como "*Fine Tuning*". De forma más concreta, en este trabajo se han empleado los parámetros ya entrenamos y se han realizado pruebas tanto congelándolos como continuando el entrenamiento.

Es importante destacar que en las redes preentrenadas se han empleado unos hiperparámetros concretos por lo que puede ser más complicado intentar obtener un mejor porcentaje. No obstante, sí que se pueden emplear las técnicas explicadas hasta ahora, como variar el *learning rate* tras un número de épocas, o finalizar el entrenamiento al llegar a una condición concreta...

5. ANÁLISIS DE LOS RESULTADOS

Antes de proceder al desarrollo de los resultados es necesario mencionar que existe cierto carácter de aleatoriedad a lo largo de todo el proceso de entrenamiento, es por ello por lo que pese a dar con una buena combinación de hiperparámetros es complicado repetir el mismo resultado exactamente. No obstante, sí que se puede considerar que hay regularidad por lo que, aunque existan diferencias en los decimales, si una combinación ofrece buenos resultados, está casi siempre lo hará.

Para la elección de la arquitectura de la red, concretamente del perceptrón multicapa, se han llevado varios entrenamientos con diferentes capas cada vez, así mismo, se han repetido bajo las mismas condiciones los entrenamientos varias veces, con el fin de mostrar la aleatoriedad nombrada recientemente:

Número de capas	Máxima precisión alcanzada	Precisión media	Desviación típica	Desviación típica de las medias
2	95,49%	95,50%	$6{,}16x10^{-4}$	$2{,}38x10^{-4}$
	95,43%			
	95,58%			
3	95,63%	95,54%	$6{,}68x10^{-4}$	
	95,52%			
	95,47%			
4	95,53%	95,56%	$1{,}08x10^{-3}$	
	95,70%			
	95,44%			

Tabla 1: Análisis de aleatoriedad y diseño de perceptrón multicapa

De esos entrenamientos se pueden obtener varias conclusiones. En primer lugar, se confirma que existe cierto carácter aleatorio que provoca que dos resultados no han de ser iguales por tener la misma combinación de hiperparámetros predefinidos. Aun así, como ya se mencionó, la variación entre resultados es de apenas decimales por lo que se puede considerar que existe regularidad en los valores obtenidos.

Seguidamente se puede comprobar a simple vista en la *Tabla 1*. cómo no se logra una clara mejoría al añadir más capas al modelo, de esta forma se puede concluir que un modelo de dos capas es el indicado para la resolución de este problema. De lo contrario, se estarían añadiendo más capas y por extensión más variables con las que trabajar, para no lograr mejores resultados.

Dicho esto, los entrenamientos se han llevado a cabo en una herramienta gratuita que ofrece un gran abanico de opciones, no obstante, hay que tener en cuenta que esta depende de la conexión a internet, por lo que, si se desea realizar comparativas entre tiempos de entrenamiento, u observar las diferentes relaciones precisión-tiempo para los diferentes modelos, es necesario que todos los ensayos se realicen bajo las mismas condiciones. Esto también implica tener una buena conectividad con la GPU, ya que en Google Colab existen limitaciones en su tiempo de uso, y si no se dispone de esta opción, los tiempos de entrenamiento pueden ser de gran duración por lo que los resultados no estarían reflejando fielmente las diferencias existentes entre los diferentes modelos.

Dicho esto, como ya se introdujo anteriormente, el objetivo de estos entrenamientos será dar con la combinación de hiperparámetros óptima que permita obtener un clasificador con la mejor precisión posible. A lo largo del capítulo anterior se ha desarrollado las diferentes alternativas que se han seleccionado para la realización del modelo, a excepción de una, la ratio de aprendizaje. Ya se ha incidido en varias ocasiones sobre la relevancia directa que existe entre este hiperparámetro y el resultado, por lo que a continuación, se procede a analizar las diferentes opciones que se han sometido a estudio y los resultados que se han podido obtener de cada una de ellas.

5.1. RESULTADOS EMPLEANDO LEARNING RATE CONSTANTE

En primer lugar, se comienza usando un learning rate de valor contante, lo que significa que la intensidad con la que se realizará el aprendizaje a lo largo de todo el entrenamiento será siempre la misma. Estos valores ofrecen además un primer acercamiento sobre el orden de magnitud con el que se está trabajando. Sea pues, se han realizado los siguientes casos de estudio:

Casos de estudio	Valor del LR empleado	Maxima precision obtenida	Duración del entrenamiento (min)	Tiempo / epoch (min)
A	10^{-3}	94,48%	121,203	4,85
B	10^{-4}	95,58%	103,47	4,14
C	10^{-5}	92,99%	132,74	5,31
D	10^{-6}	82,19%	118,76	4,75

Tabla 2: Resultados obtenidos empleando learning rate constante

Como se puede observar en la *Tabla 2* se han realizado cuatro casos de estudio donde se ha comparado tanto la máxima precisión que se ha obtenido a lo largo del proceso como el tiempo requerido para el completo entrenamiento de los diferentes modelos.

De esta forma, si se realiza un primer escrutinio a los tiempos en los que han tenido lugar los entrenamientos, se comprueba que, a excepción del Caso de estudio "C", son valores relativamente similares por lo que para la elección de la mejor alternativa habrá que emplear otro criterio que si permita discernir de forma clara entre las diferentes opciones.

También hay que aclarar que, como se ha introducido al comienzo del entrenamiento, los tiempos dependen de la conectividad a la red, por lo que ese valor distintivo del Caso C en otro momento podría haber sido inferior y quizás más similar al resto. No obstante, no debe considerarse como valor anómalo ya que, tras los diferentes ensayos realizados, es un valor que entra dentro de los límites de lo que se ha considerado como un buen entrenamiento. Es por ello por lo que no se debe descartar este valor como posible alternativa, en base al tiempo de entrenamiento que ha requerido.

Es importante destacar que pese a tomarse como dato de análisis, los tiempos de entrenamiento no son un valor critico a la hora de decidir la combinación de parámetros. Básicamente esto se debe a que lo más interesante de un clasificador es su precisión y velocidad en la predicción de imágenes, pero el tiempo de entrenamiento es relativamente irrelevante.

Continuando con el análisis de los valores obtenidos, el otro criterio que se puede emplear es el de la precisión obtenida. En base a esto, para cada caso de estudio se obtienen los diferentes resultados:

Caso de estudio A: Inicialmente, evaluando exclusivamente el máximo valor obtenido se podría considerar como una buena selección de los hiperparámetros ya que los resultados son satisfactorios. Sin embargo, este valor aislado no muestra toda la magnitud del entrenamiento, por lo que de seleccionar este valor se estaría cometiendo un error. Este hecho es fácilmente detectable si se observa la *Figura 15*, donde se comprueba que, pese a un buen comienzo y una correcta evolución en el aprendizaje, tras un cierto número de épocas, la precisión se desploma.

Como se explicó en el apartado teórico, este fenómeno se debe a que el valor del *learning rate* es demasiado grande por lo que carece de la precisión necesaria para converger en el mínimo error absoluto. De hecho, ocurre todo lo contrario, la función diverge y se producen una serie de errores en el ajuste de parámetros que provocan esa caída de la precisión del modelo.

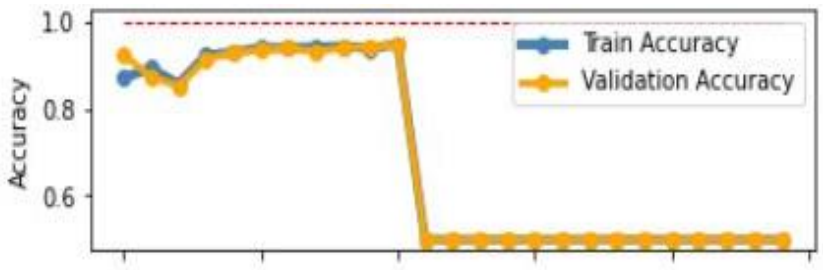

Figura 15: Evolución de la precisión Caso de estudio A

Caso de estudio B: Tras los datos obtenidos en el Caso anterior, se procede a reducir el valor del learning rate. Si bien ya se ha explicado que los tiempos no son un factor determinante, hay que destacar que con este valor se ha obtenido el menor tiempo de entrenamiento de todos.

Así mismo, también se observa que se obtiene la máxima precisión de todos los casos de estudio, postulándose como la mejor alternativa para el diseño del clasificador. Para evitar cometer los errores del caso anterior se comprueba la *Figura 16*, que muestra en este caso una correcta evolución lo cual indica que se puede dar por válido el valor de la ratio de aprendizaje que se ha empleado en este Caso de estudio.

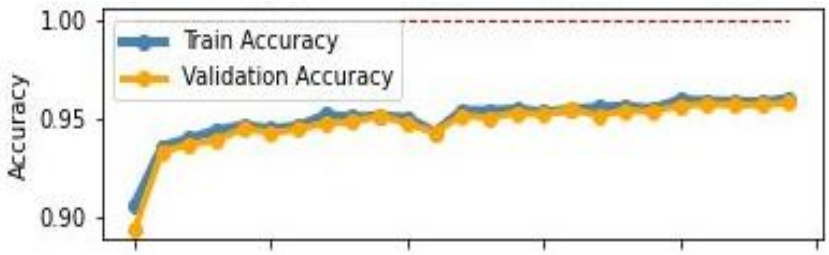

Figura 16: Evolución de la precisión Caso de estudio B

Caso de estudio C: Como se observa en la *Tabla* 2. al reducir el valor del *learning rate* ha disminuido la máxima precisión alcanzada. Este suceso es fácilmente justificable, al tener un menor *learning rate*, el avance es mucho más lento por lo que para alcanzar los valores obtenidos en el Caso B sería necesario un mayor número de épocas, pero al dejar constante el número de épocas, el modelo no logra igualar los valores obtenidos hasta ahora.

De forma aislada se podría considerar estos resultados como buenos, ya que una precisión del 93% generalmente se concibe como satisfactoria. No obstante, en la búsqueda por la mayor precisión posible, se debería descartar este valor ya que en el Caso anterior se superó la precisión obtenida en este.

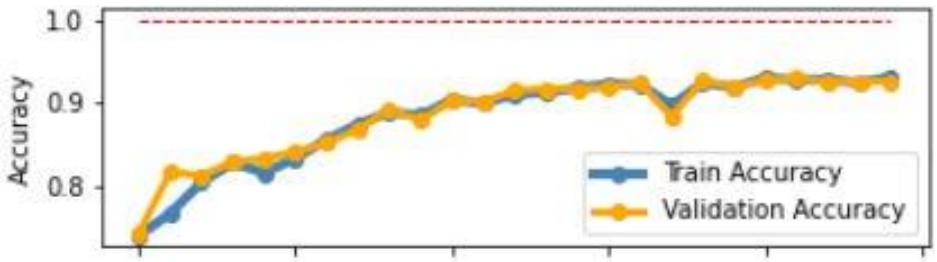

Figura 17: Evolución de la precisión Caso de estudio C

Caso de estudio D: Comparando los resultados, se podría considerar que el *learning rate* empleado en este caso es el que devuelve unos peores resultados, por lo que se puede concluir que no tiene sentido continuar reduciendo este parámetro ya que va a empeorar la precisión final. Estos resultados son fruto de lo explicado en el Caso anterior, al llevar un avance más lento, cuanto más se reduce el learning rate menor será la precisión para un número de épocas constantes.

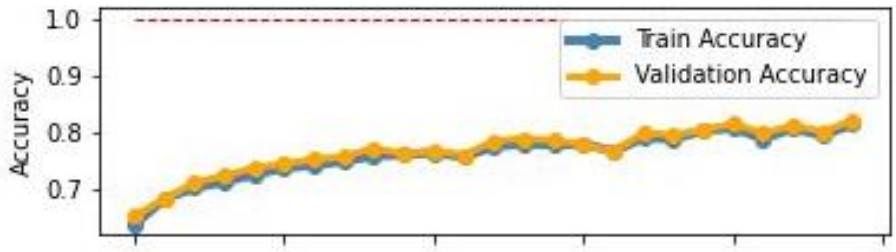

Figura 18: Evolución de la precisión Caso de estudio D

Tras analizar los diferentes casos de estudio, se pueden resaltar los resultados obtenidos en el Caso B, no obstante, es necesario destacar que la máxima precisión que se logra se alcanza en la última etapa del entrenamiento. A causa de esto, se decide llevar a cabo un nuevo entrenamiento con los mismos parámetros, pero en esta ocasión se decide ampliar hasta las 35 épocas para determinar si el modelo puede seguir mejorando.

Caso de estudio	Valor del LR empleado	Máxima precisión obtenida	Duración del entrenamiento (min)	Tiempo / epoch (min)
B.2	10^{-4}	95,44%	189,35	5,41

Tabla 3: Resultados logrados en el caso B.2

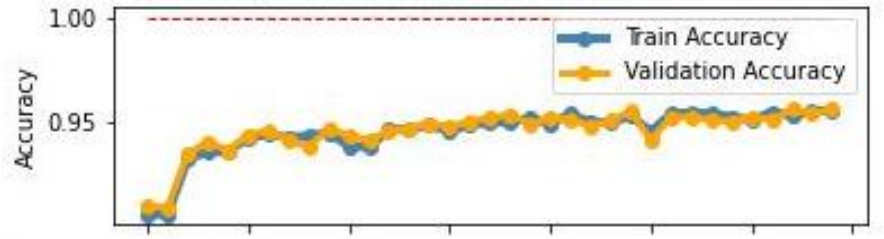

Figura 19: Evolución de la precisión Caso de estudio B.2

Por razones obvias, a más épocas, mayor es el tiempo total del entrenamiento. Pero lo importante de este entrenamiento es la precisión que se alcanza, no solo no continúa mejorando, sino que además es inferior a la ya obtenida. Esto demuestra varias cosas, por un lado, se refleja el carácter aleatorio de estos entrenamientos, donde una misma combinación de hiperparámetros devuelve resultados diferentes. Por otro lado, como se ha podido comprobar, no necesariamente aumentar el tiempo de entrenamiento va a ofrecer mejores resultados. En este caso concreto, se ha requerido casi el doble de tiempo para completar el entrenamiento para obtener unos resultados inferiores.

5.2. RESULTADOS EMPLEANDO LEARNING RATE ADAPTATIVO

En base a los datos obtenidos en el apartado anterior se ha decidido emplear varios valores de learning rate a lo largo de un mismo entrenamiento. De esta forma se buscan combinar las ventajas que ofrecen cada uno, es decir, obtener una buena aproximación a la máxima precisión posible, y tener la minuciosidad necesaria para alcanzarla. Para ello se han realizado los siguientes casos:

Casos de estudio	Tipo y época/s de cambio	Valores de LR empleados	Máxima precisión obtenida	Duración del entrenamiento (min)	Tiempo / epoch (min)
E	Step - 15	$10^{-4}\ y\ 10^{-5}$	95,23%	119,23	4,77
F	Step - 17	$10^{-4}\ y\ 10^{-5}$	95, 48%	120,59	4,8236
G	Step - 22	$10^{-4}\ y\ 10^{-5}$	95,42%	118,58	4,74
H	Multistep - 15-20	$10^{-4}, 10^{-5}\ y\ 10^{-6}$	95,38%	126,87	5,08
I	Multistep - 17-22	$10^{-4}, 10^{-5}\ y\ 10^{-6}$	95,14%	102,96	4,11

Tabla 4: Resultados obtenidos empleando learning rate adaptativo

De nuevo se comprueba que la duración de los entrenamientos apenas varía de una alternativa a otra, incluso comparando con los valores constante, por lo que tampoco se puede discernir entre casos según lo que dura su entrenamiento.

Si se observa ahora la máxima precisión alcanzada, en todos los casos que se han realizado, se ha logrado superar el 95% siendo todos resultados realmente buenos. Es aquí donde entra en juego la aleatoriedad de la que se hablaba al inicio del capítulo, y es que, si se repitiesen todos los casos de estudio, es probable que las precisiones varíen y por tanto unas alternativas pasarían a ser mejores que otras. Lo que está claro es que todas aportan los mismos resultados por lo que puede parecer indiferente emplear unas alternativas u otras.

En caso de continuar analizando más a fondo, se localiza un fenómeno que merece la pena destacar, para ello es necesario destacar en la época concreta en la que se alcanza la máxima precisión.

Este dato para los valores de *learning rate* constante era ciertamente indiferente, ya que generalmente se alcanzaba el máximo en las últimas épocas fruto de un continuo aprendizaje, sin embargo, los máximos alcanzados en estos últimos casos son los siguientes:

Casos de estudio	Época donde se alcanza el máximo	Época de cambio del learning rate
E	16	15
F	20	20
G	24	22
H	18	$15 - 20$
I	20	$17 - 22$

Tabla 5: Información adicional para los casos empleando learning rate adaptativo

Como se puede observar, en los Casos de estudio E, F y G, es decir, los de tipo Step, la máxima precisión alcanzada tiene lugar en épocas próximas al cambio si no en esa época. Visto de otra forma, lo que reflejan estos datos es que el cambio de valor no aporta beneficio alguno ya que no hay mejoría una vez se alcanza el máximo. Concretamente en el Caso E, tras alcanzar el mejor resultado aún tiene por delante un tercio del entrenamiento durante el cual no parece mejorar.

Por otro lado, en los casos H e I, sucede algo similar, el máximo se alcanza en las épocas próximas al primer cabio de valor por lo que no solo el primer cambio, sino que el segundo es completamente innecesario.

Al mismo tiempo, se observa que las gráficas que muestran la evolución de la precisión en los diferentes casos de learning rate adaptable, resultan muy similares a la que ofrece el Caso de estudio B. Luego, dados los resultados, resulta razonable argumentar que los cambio que ofrecen los valores adaptativos son innecesarios, ya que no aportan ninguna mejoría respecto al valor constante, incluso ofreciendo este un valor por debajo de la media de los variables.

5.3. RESULTADOS EMPLEANDO TRANSFER LEARNING

Durante este capítulo se va a desarrollar los diferentes métodos de *Transfer Learning* que se han empleado. Si bien estos métodos se presentan como una alternativa al modelo diseñado, es decir, no van a producir mejorías en el modelo estudiado hasta ahora ya que es precisamente en esto en lo que consiste el *transfer learning*, en importar modelos ya diseñados.

Aun así, estas técnicas son muy destacadas en el mundo de la inteligencia artificial y del *machine learning*, por lo que dados los resultados que se han obtenido, merece la pena analizarlos y desarrollarlos.

En primer lugar, se ha importado el modelo ResNet18, con los parámetros ya entrenados con la opción "*Pretrained*", la cual ya se explicó en el *Apartado 4.4.*, y antes de modificar la última capa que permita la clasificación de imágenes en dos categorías, se realiza el congelado de los parámetros de toda la red, de esta forma, a lo largo del entrenamiento, los únicos parámetros que se reajusten serán los de esta última capa añadida.

Así pues, tras realizar este entrenamiento cuya duración es de tan solo 5 minutos y 58 segundos se consigue alcanzar un 87% de máxima precisión. Rápidamente se comprueba la diferencia existente entre modelos en cuanto a lo que duración del entrenamiento se refiere. No obstante, la precisión que se obtiene es bastante inferior a la que se ha obtenido con el diseño de red original, por lo que esta combinación no parece aportar mejoría.

Siguiendo con la misma estrategia, de nuevo se importa el modelo con los valores ya entrenados, pero en esta ocasión se decide descongelar la red. Esto supone que, a partir de los valores entrenados previamente, cuando comience el entrenamiento, se reajustaran todos los parámetros de la red.

Numero de épocas	Máxima precisión obtenida	Época en la que se obtiene	Duración del entrenamiento (min)
25	97,22%	12	5,87
15	97,22%	11	3,57
5	97,11%	2	1,18
3	96,89%	2	0,71

Tabla 6: Resultados obtenidos empleando la técnica de Fine Tuning

Para estos entrenamientos se han respetado los parámetros con los que se entrenó la red importada, no obstante, se ha querido comenzar por el mismo número de épocas para obtener una referencia y poder comparar de forma directa con el resto de las alternativas y combinación de parámetros.

Tal y como ocurría con el método de los valores congelados, se puede ver reflejado en la *Tabla 6* que los tiempos de entrenamiento son muy inferiores luego ya se comienza obtenido mejoras respecto a los modelos originales. Al mismo tiempo, es fácilmente destacable la mejoría existente en la precisión lograda, la cual es ligeramente superior a la máxima que se había logrado hasta ahora.

A continuación, observando las épocas en las que se alcanza el máximo, se decide exprimir los tiempos de entrenamiento por lo que se procede a reducir el número de épocas que se llevan a cabo, pero siempre con la intención de mantener una buena precisión final.

Se comprueba que es posible mantener los buenos resultados, con un menor número de épocas lo que implica un menor tiempo de entrenamiento. Eso sí, si se reducen en exceso, la red no tiene tiempo de alcanzar su máximo potencial, por lo que es necesario permitirle cierta holgura que le permita aprender.

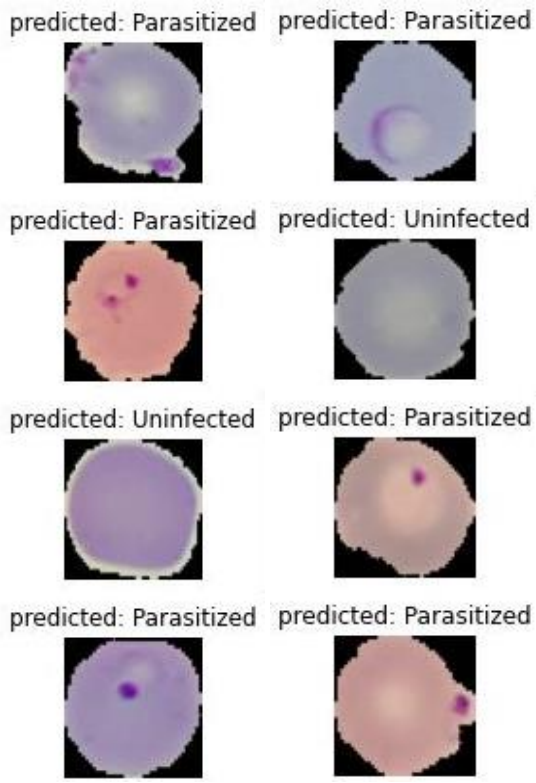

Figura 20: Predicciones realizadas por el mejor clasificador diseñado

Como se puede observar en la *Figura 20*, el clasificador realiza predicciones con un gran porcentaje de acierto. Aun así, esto es un simple ejemplo gráfico, ya que, en caso de haberse realizado una predicción errónea, y haber clasificado mal una de las ocho imágenes que se muestran, sería incorrecto asumir entonces que la precisión se limita a un $87{,}5\%$. Hay que tener en cuenta que esta precisión de la clasificación se aplica a cada imagen, y mientras esta no sea del 100%, en el mundo probabilístico siempre va a existir un riesgo, por pequeño que sea, de que esta diferenciación de clases se realice de forma incorrecta. Por tanto, para evitar este riesgo es por lo que se busca aumentar al máximo la precisión del modelo.

6. CONCLUSIONES

Dado el objetivo de diseñar e implementar un clasificador desde cero, se ha logrado obtener buenos resultados. El diseño de la red, así como su arquitectura, sus funciones de coste y de activación han demostrado un buen comportamiento frente a las necesidades requeridas.

En lo que al *learning rate* se refiere, ya que es el parámetro que más estudio ha requerido, los resultados han mostrado que el valor más indicado para la correcta resolución del problema es 10^{-4}, ya que es el que permite obtener el mayor porcentaje de acierto en la clasificación de las diferentes imágenes. Al mismo tiempo, se ha podido demostrar que incluir valores de la ratio de aprendizaje adaptativos que vayan variando durante el entrenamiento no aportan ningún valor añadido, ya que, si bien los resultados que se obtienen no difieren mucho de los que se alcanzan con el valor constante, los tiempos requeridos, así como el procesamiento es mucho mayor. Por lo que se puede concluir que, con el uso del valor constante ya mencionado, el modelo logra sus mejores resultados con un $95{,}58\%$ de precisión en el acierto de la clasificación de las imágenes.

De esta manera, se puede concluir que se ha logrado el objetivo con resultados satisfactorios. Aun así, hay que destacar las mejorías que aportan los modelos empleados mediante *Transfer Learning* en los problemas de esta índole. Gracias a estos métodos se logran grandes resultados, sin requerir apenas conocimientos sobre el diseño de una red, ya que se importan los modelos ya completados. Además, no requieren *datasets* muy amplios para sus entrenamientos por lo que se pueden emplear en problemas más concretos y específicos donde no se disponga de muchas imágenes para analizar.

Este último punto abre las posibilidades a nuevos trabajos de cara al futuro, como podría ser, el análisis de las capacidades y los límites que tienen las técnicas del *Transfer Learning*.

Finalmente hay que añadir que se ha logrado obtener un clasificador con una precisión de $95 - 97\%$, por lo que se da por conseguidos los objetivos. Esto demuestra que es factible como vía de desarrollo para un clasificador de aplicación práctica. No obstante, como ya se introdujo al comienzo de este trabajo, el modelo diseñado es sencillo y se ha entrenado para unos datos concretos por lo que pese a ser un buen punto de comienzo, aún dista de ser un método aplicable en el día a día de un centro médico.

6.1. PROBLEMAS ENCONTRADOS

A lo largo de este trabajo han surgido diferentes inconvenientes que han dificultado el avance del desarrollo de este. Aun así, se han podido solventar y como se ha indicado en las Conclusiones, se ha logrado el objetivo propuesto. Los problemas que se han encontrado han sido los siguientes:

Dataset adecuado: A lo largo del trabajo se ha mencionado en varias ocasiones la relevancia de un buen dataset. En primer lugar, debe estar bien etiquetado de forma que se evite una mala distinción de clases por motivos ajenos a la red neuronal. Seguidamente, debe tener una buena cantidad de imágenes con las que trabajar.

Además, se debe entender que cada dataset se puede encontrar en diferentes formatos dependiendo de donde se obtenga, por lo que la forma de implementarlo para poder operar con él es diferente según el caso. De esta forma encontrar un conjunto de imágenes que cumpla todas las restricciones mencionadas fue el primer obstáculo antes de comenzar a trabajar.

Limitaciones de hardware: Precisamente, al tratar una elevada cantidad de imágenes se requieren unas ciertas características de la CPU para llevar a cabo el tratamiento de estas. Por lo que si se desea evitar problemas relacionados con estas limitaciones se debería trabajar con una computadora con una tarjeta gráfica de calidad.

Al surgir este problema durante el trabajo, se decidió emplear Google Colab que es una herramienta que permite trabajar con una GPU de forma remota facilitando así el procesamiento de datos matriciales, así como su operatividad, independientemente de la calidad y los recursos que se dispongan en cada procesador.

Tiempos de entrenamiento: Durante un entrenamiento se procesan un gran número de imágenes, igual que se llevan a cabo una amplia variedad de operaciones y cálculos, por lo que cada entrenamiento requiere una larga duración, siempre teniendo en cuenta las magnitudes del problema. Este hecho ha provocado un lento avance del trabajo ya que para el análisis de los diferentes hiperparámetros se ha requerido de numerosos entrenamientos.

Al emplear la GPU se han acelerado los entrenamientos, no obstante, el acceso gratuito tiene sus limitaciones. Concretamente, limitaciones de tiempo, es decir que, si se usa demasiado tiempo, se desconecta el servicio y se debe esperar un tiempo hasta volver a disponer de él. Para evitarlo, se ha adquirido el servicio Google Colab Plus, para poder disponer de un libre acceso continuado a diferentes GPU's de calidad. Aun así, la duración del tiempo del que se puede disponer de la conexión a una GPU sigue sin ser ilimitado, por lo que en ocasiones no se ha podido realizar entrenamientos al no disponer de esta conexión. Pese a estos inconvenientes, se han podido llevar a cabo todos los entrenamientos necesarios para alcanzar los objetivos.

7. BIBLIOGRAFÍA

[1] F. Yang, «Deep Learning for Smartphone-Based Malaria Parasite Detection in Thick Blood Smears,» *Journal of Biomedical and Health Informatics,* vol. 24, nº 5, pp. 1427-1438, 2020.

[2] J. V. Benlloch, M. Agusti, A. Sanchez y A. Rodas, «Colour segmentation techniques for detecting weed patches in cereal crops,» de *Proc. of Fourth Workshop on Robotics in Agriculture and the Food-Industry*, October, 1995, pp. 30-31.

[3] A. J. Sanchez, W. Albarracin, R. Grau, C. Ricolfe y J. M. Barat, «Control of ham salting by using image segmentation,» de *Food Control*, 2008, pp. 19(2), 135-142.

[4] A. J. Sanchez y J. A. Marchant, «Fusing 3D information for crop/weeds classification,» *In Proceedings 15th International Conference on Pattern Recognition ICPR-2000,* vol. 4, pp. 295-298, 2000a.

[5] A. J. Sanchez-Salmeron y C. Ricolfe-Viala, «Optimal conditions for camera calibration using a planar template,» *In 2011 18th IEEE International Conference on Image Processing,* pp. 853-856, 2011, Septiembre.

[6] E. Ivorra, S. Amat, A. J. Sanchez, J. M. Barat y R. Grau, «Continuous monitoring of bread dough fermentation using a 3D vision Structured Light technique,» *Journal of Food Engineering,* vol. 130, pp. 8-13, 2014.

[7] E. Ivorra, A. J. Sanchez, J. G. Camarasa, M. P. Diago y J. Tardaquila, «Assessment of grape cluster yield components based on 3D descriptors using stereo vision,» *Food Control,* vol. 50, pp. 273-282, 2015.

[8] S. Verdú, E. Ivorra, A. J. Sanchez, J. M. Barat y R. Grau, «Relationship between fermentation behavior, measured with a 3D vision Structured Light technique, and the internal structure of bread,» *Journal of Food Engineering,* vol. 146, pp. 227-233, 2015a.

[9] R. Grau, A. J. Sanchez, J. Girón, E. Iborra, A. Fuentes y J. M. Barat, «Non-destructive assessment of freshness in packaged sliced chicken breasts using SW-NIR spectroscopy,» *Food Research International,* vol. 44(1), pp. 331-337, 2011.

[10] E. Ivorra, A. J. Sánchez, S. Verdú, J. M. Barat y R. Grau, «Shelf-life prediction of expired vacuum-packed chilled smoked salmon based on a KNN tissue segmentation method using hyperspectral images,» *Journal of Food Engineering,* vol. 178, pp. 110-116, 2016.

[11] S. Verdú, E. Ivorra, A. J. Sánchez, J. M. Barat y R. Grau, «Study of high strength wheat flours considering their physicochemical and rheological characterisation as well as fermentation capacity using SW-NIR imaging,» *Journal of Cereal Science,* vol. 62, pp. 31-37, 2015b.

[12] J. Puchalt, A. J. Sánchez-Salmeron, P. Martorell Guerola y S. Genovés Martínez, «). Active backlight for automating visual monitoring: An analysis of a lighting control technique for Caenorhabditis elegans cultured on standard Petri plates,» *PloS one,* vol. 14(4), 2019.

[13] J. C. Puchalt, J. F. González-Rojo, A. P. Gómez-Escribano, R. P. Vázquez-Manrique y A. J. Sánchez-Salmerón, «Multiview motion tracking based on a cartesian robot to monitor Caenorhabditis elegans in standard Petri dishes,» *Scientific reports,* vol. 12(1), pp. 1-11, 2022.

[14] J. C. Puchalt, A. J. Sánchez-Salmerón, E. Ivorra, S. G. Martínez y P. M. Guerola, «Improving lifespan automation for Caenorhabditis elegans by using image processing and a post-processing adaptive data filter,» *Scientific Reports,* vol. 10(1), pp. 1-14, 2020.

[15] J. Puchalt, A. J. Sánchez-Salmerón y P. E. Layana Castro, «Reducing Results Variance in Lifespan Machines: An Analysis of the Influence of Vibrotaxis on Wild-Type Caenorhabditis elegans for the Death Criterion,» *Sensors,* vol. 20(21), nº 5981, 2020a.

[16] J. C. Puchalt, A. J. Sánchez-Salmerón, E. Ivorra y e. al, «Small flexible automated system for monitoring Caenorhabditis elegans lifespan based on active vision and image processing techniques,» *Scientific Reports,* vol. 11, nº 12289, 2021.

[17] P. E. Layana Castro, J. C. Puchalt, A. García Garví y A. J. Sánchez-Salmerón, «Caenorhabditis elegans Multi-Tracker Based on a Modified Skeleton Algorithm,» *Sensors,* vol. 21(16), p. 5622, 2021.

[18] J. C. Puchalt, A. J. Sánchez-Salmerón y P. E. L. Castro, «Improving skeleton algorithm for helping Caenorhabditis elegans trackers,» *Scientific Reports,* vol. 10(1), pp. 1-12, 2020.

[19] A. García Garví, J. C. Puchalt, P. E. Layana Castro, F. Navarro Moya y A. J. Sánchez-Salmerón, «Towards Lifespan Automation for Caenorhabditis elegans Based on Deep Learning: Analysing Convolutional and Recurrent Neural Networks for Dead or Live Classification,» *Sensors,* vol. 21(14), p. 4943, 2021.

[20] F. Navarro Moya, J. C. Puchalt, P. E. Layana Castro, A. García Garví y A. J. Sánchez-Salmerón, «A new training strategy for spatial transform networks (STN's),» *Neural Computing and Applications,* pp. 1-12, 2022.

[21] Organizacion Mundial de la Salud, «Organizacion Mundial de la Salud / Paludismo,» 6 Diciembre 2021. [En línea]. Available: https://www.who.int/es/news-room/fact-sheets/detail/malaria#. [Último acceso: 10 Mayo 2022].

[22] National Library of Medicine, «MedlinePlus,» 2021 Marzo 4. [En línea]. Available: https://medlineplus.gov/spanish/pruebas-de-laboratorio/pruebas-de-malaria/#:~:text=Si%20sospecha%20que%20usted%20tiene,tubo%20de%20ensayo%20o%20frasco.. [Último acceso: 10 Mayo 2022].

[23] Colaboradores de Wikipedia, «Wikipedia, La enciclopedia libre,» 15 Diciembre 2021. [En línea]. Available: https://es.wikipedia.org/wiki/Dendrita. [Último acceso: 11 Mayo 2022].

[24] M. J. Mas, «Neuronas en crecimiento,» 4 Junio 2014. [En línea]. Available: https://neuropediatra.org/2014/06/04/sinapsis-neuronal/#:~:text=Se%20calcula%20que%20un%20cerebro,conectarse%20hasta%20con%20otras%2050.000.. [Último acceso: 11 Mayo 2022].

[25] F. Izaurieta y C. Saavedra, «Redes Neuronales Artificiales,» Departamento de Fisica, Universidad de Chile, Concepción,Chile, 2000.

[26] C. Lamas Álvarez, «Mejoras de voz mediante redes neuronales profundas,» Trabajo Fin de Grado, Universidad Autónoma de Madrid, Junio 2017.

[27] J. Maroñas Molano, «Adversarial Learning with Ladder Network,» Departamento de sistemas informaticos y computacion, Proyecto Fin de Master, Universidad Politecnica de Valencia, 18 Septiembre 2016.

[28] D. Calvo, «Diego Calvo: Análisis de Datos,» 7 Diciembre 2018. [En línea]. Available: https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/. [Último acceso: 12 Mayo 2022].

[29] C. M. Bishop, «Neural Networks for Pattern Recognition,» Oxford, Clarendon Press, 1995, pp. 195-200.

[30] PyTorch, «PyTorch / Transforming and augmenting images,» 2017. [En línea]. Available: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html. [Último acceso: Mayo 2022].

[31] PyTorch, «PyTorch / Transfer learning for computer vision tutorial,» 2022. [En línea]. Available: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html. [Último acceso: Mayo 2022].

[32 K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» 10 Diciembre 2015.]

[33 Colaboradores de Wikipedia, «Wikipedia, La enciclopedia libre,» 12 Noviembre 2021. [En línea].] Available: https://es.wikipedia.org/wiki/Perceptr%C3%B3n. [Último acceso: 14 Mayo 2022].

8. GLOSARIO

TFG: Trabajo Fin de Grado

DNN: Deep Neuronal Network

ReLU: Rectified Lineal Unit

RMSE: Root Mean Square Error

BCE: Binary Cross-Entropy

LR: Learning Rate

CPU: Central Processing Unit

GPU: Graphic Processing Unit

9. PRESUPUESTO

9.1. DESCRIPCIÓN

En el presente capítulo se procede a valorar el presupuesto del trabajo llevado a cabo con el fin de determinar su viabilidad económica a través del cálculo de los costes necesarios para completar el proyecto. Para evaluar esta viabilidad se han presupuestado la mano de obra, el material empleado, así como las licencias de software requeridas, cada cual con su respectivo cuadro de costes parciales.

A partir de estos cuadros se han obtenido los diferentes presupuestos necesarios en cualquier proyecto de ingeniería como son: el presupuesto de ejecución material, el presupuesto de ejecución por contrata y, para concluir, el presupuesto general del proyecto.

Previo al desarrollo del presupuesto se requiere una aclaración de los códigos que se han empleado para la elaboración de este recogidos en la siguiente tabla.

Código	Descripción del código
MO	Mano de obra
MM	Material
LS	Licencia de Software

Tabla 7: Tabla de códigos empleados en el desarrollo del presupuesto

9.2. MANO DE OBRA

A continuación, se definen los precios de la mano de obra en el desarrollo del trabajo. Para contabilizar las horas se han tenido en cuenta las diferentes etapas del proyecto, como son la búsqueda de un dataset adecuado, la implementación del modelo en Python, el entrenamiento y el análisis de los resultados obtenidos. Así mismo también se han contabilizado las horas de seguimiento y corrección por parte del tutor del TFG, por lo que también se ha considerado esta figura como mano de obra del trabajo.

Código	Descripción	Importe		
		Coste (€/h)	Cantidad (h)	Precio (€)
MO. 1	Graduado en Ingeniería de Tecnologías Industriales	18	300	5400
MO. 2	Tutor Responsable del Proyecto	30	30	900
Precio Total Mano de Obra				6300

Tabla 8: Cuadro de precios de mano de obra

9.3. MATERIAL EMPLEADO

Para el material que se ha empleado en este proyecto, se ha calculado la amortización de este de acuerdo con los periodos que se observan en la *Tabla 9*. En este caso, para la realización de todo el trabajo se ha empleado un único ordenador portátil el cual se ve reflejado en el presupuesto de la siguiente forma:

Código	Descripción	Importe			
		Coste (€)	Amortización (años)	Tiempo (meses)	Precio (€)
MM. 1	Ordenador portátil Lenovo	475	5	2	15,85
	Precio Total Material				15,85

Tabla 9: Cuadro de precios de material empleado

La amortización se ha calculado según la siguiente ecuación:

$$C_A = \frac{C_{mat} * t_{uso}}{Per_{amor}}$$

Las variables empleadas son:

C_{mat}: Coste de los materiales empleados

t_{uso}: Tiempo de uso del material

Per_{amor}: Periodo de amortización del material

9.4. LICENCIAS DE SOFTWARE

Como se ha mencionado a lo largo de la memoria, se ha empleado una herramienta (Google Colab) de acceso gratuito para la implementación del modelo computacional. Así mismo, el lenguaje de Python como las librerías empleadas también son de libre acceso. No obstante, como se indica en el apartado de *6.1. Problemas encontrados*, se ha adquirido la versión de Google Colab +, con la finalidad de agilizar el trabajo llevado a cabo. De esta forma, este coste debe quedar registrado en el presupuesto de la siguiente forma:

Código	Descripción	Importe		
		Coste (€/mes)	Cantidad (meses)	Precio (€)
LS. 1	Software Google Colab+	10,99	1	10,99
Precio Total Licencias Adquiridas				10,99

Tabla 10: Cuadro de precios de licencias de software empleadas

9.5. PRESUPUESTO FINAL

El primer presupuesto que se obtiene es el de ejecución material, que se obtiene tras la suma de los precios parciales de la mano de obra, el material empleado y las licencias adquiridas, además hay que tener en cuenta que hay que aplicar unos costes indirectos que para este trabajo se estiman en un 15% del total de costes directos.

A continuación, se obtiene el presupuesto de ejecución por contrata, el cual no es más que el presupuesto de ejecución material más unos gastos generales y unos beneficios industriales. Para este proyecto se han valorado en un 13% los gastos generales y en un 6% el beneficio industrial.

Finalmente, se obtiene el presupuesto base de licitación o presupuesto general el cual resulta de aplicar el porcentaje del IVA al último presupuesto calculado.

Presupuesto de Ejecución Material, por Contrata y Base de Licitación

1.Presupuesto Parcial Mano de Obra	6.300,00 €
2.Presupuesto Parcial Material Empleado	15,85 €
3.Presupuesto Parcial Licencias Adquiridas	10,99 €
Total Parcial	6.326,84 €
Costes Indirectos (15%)	949,03 €
Total Ejecución Material	**7.275,87 €**

El presupuesto de Ejecución Material asciende a la expresada cantidad de **SIETE MIL DOSCIENTOS SETENTA Y CINCO EUROS Y OCHENTA Y SIETE CÉNTIMOS**

Costes Generales (13%)	945,86 €
Beneficio Industrial (6%)	436,55 €
Total Ejecución por Contrata	**8.658,28 €**

El presupuesto de Ejecución por Contrata asciende a la expresada cantidad de **OCHO MIL SEISCIENTOS CINCUENTA Y OCHO EUROS Y VEINTIOCHO CÉNTIMOS**

IVA (21%)	1.818,24 €
Total Base de Licitación	**10.476,52 €**

El presupuesto de Ejecución por Contrata asciende a la expresada cantidad de **DIEZ MIL CUATROCIENTOS SETENTA Y SEIS EUROS Y CINCUENTA Y DOS CÉNTIMOS**