UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

CLASIFICACIÓN DE IMÁGENES MÉDICAS UTILIZANDO REDES NEURONALES

POR

Nicolás Alfonso Bettancourt Matamala

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Biomédico

Profesora Guía:
Dra. Pamela Guevara A.

Comisión:
Dra. Cecilia Hernández R.
Dr. Miguel Figueroa T.

Agosto 2022
Concepción (Chile)

Agradecimientos

A mi familia, por creer en mí. Gracias familia.

A mis amigos, por su compañía en la etapa universitaria y gran parte de mi vida. Ha sido un placer crecer con ustedes. Espero que sigamos creciendo juntos. Gracias amigos.

A la profesora Pamela, por su excelente disposición para ayudarme tanto en el desarrollo de este trabajo como en otros aspectos acádemicos, y por impulsar la motivación de estudiar el área de imágenes médicas en el transcurso de la carrera. Gracias profesora Pamela.

Se agradece financiamiento al Centro ANID-Basal FB0008, Centro Avanzado de Ingeniería Eléctrica y Electrónica (AC3E).

Resumen

Modelos de Redes Neuronales Convolucionales de gran prestigio en la literatura científica fueron implementados para tareas de clasificación de imágenes médicas. El principal objetivo de esta memoria fue generar una base de datos de carácter educativo para el aprendizaje de la implementación y optimización de estos algoritmos. La construcción de esta base de datos, que consiste en tutoriales desarrollados en lenguaje de programación Python utilizando herramientas open-source como Google Colab y bases de datos públicas de imágenes médicas, requirió de un exhaustivo estudio para encontrar los modelos y configuraciones óptimos para cada problema.

Los modelos estudiados fueron VGG-16, ResNet-50 e Inception-V3, y las bases de datos seleccionadas fueron tres. La primera (TC) incluye imágenes de resonancia magnética (MRI) de tres tipos de tumores cerebrales: meningioma, glioma y pituitario. La segunda base de datos (C19) es sobre Covid-19 e incluye imágenes de radiografía de tórax pertenecientes a las clases normal, neumonia y covid. La tercera base de datos (ER) es una colección de imágenes de fondo de retina dividida en dos clases: retinas sanas y enfermas. La clasificación implementada fue de tipo multiclase para las bases de datos TC y C19, y de tipo binaria para ER.

El estudio fue dividido en una serie de etapas que cubrieron el efecto que distintos valores para los hiperparámetros batch-size y learning-rate poseen sobre el rendimiento de la clasificación, además de la aplicación de las técnicas de aumento de datos y fine-tuning para mejorarlo. La evaluación fue ejecutada empleando las métricas recall, precision, F1-score y accuracy, y también matrices de confusión y curvas de aprendizaje. Como resultado, se obtuvo que para la base de datos TC el mejor modelo fue ResNet-50 con batch-size 256. Por otra parte, VGG-16 fue el mejor modelo para las bases de datos C19 y ER, con tamaños de batch-size iguales a 64 y 256, respectivamente. El valor de learning-rate que obtuvo mejores resultados fue de 1e-3 para todas las bases de datos. La aplicación de aumento de datos no ayudó en la mejora de la clasificación en ninguna base de datos, mientras que fine-tuning mejoró el rendimiento en TC y C19, y presentó resultados mixtos en ER. Los puntajes máximos en accuracy fueron de 94.6 %, 98.8 % y 92.7 % para las bases de datos TC, C19 y ER, respectivamente. Estos resultados, en conjunto con otras métricas y consideraciones, ayudaron a definir el diseño y construcción de tres tutoriales —uno por cada base de datos—, que comprenden la información fundamental y pasos necesarios para el desarrollo de un modelo de inteligencia artificial capaz de clasificar imágenes médicas. Con esto se cumple el objetivo de generar un material educativo para la clasificación de imágenes médicas basado en redes neuronales.

Abstract

Scientific literature's renowned Convolutional Neural Network models were implemented for medical image classification tasks. The main objective was to generate a database for educational purposes in the learning of implementing and optimizing these algorithms. The construction of this database, which consists of tutorials developed in Python programming language using open-source tools such as Google Colab and public medical imaging datasets, required an exhaustive study to find out what models and configurations were optimal for each problem.

The studied models were VGG-16, ResNet-50 and Inception-V3, and the selected datasets were three. The first one (TC) includes MRI (magnetic resonance imaging) images of three types of brain tumors: meningioma, glioma and pituitary. The second one (C19) is a Covid-19 dataset which includes chest X-ray images for three classes: normal, pneumonia and covid. The third one (ER) is a collection of retinal fundus images divided into two classes: healthy and unhealthy retinas. Multi-class classification was implemented for the TC and C19 datasets, whilst binary classification was implemented for ER.

The study was divided into a series of stages which covered the effect that different values for the hyperparameters batch-size and learning-rate have on the performance of the classification, as well as the application of data augmentation and fine-tuning techniques in order to improve it. The evaluation was executed using the metrics recall, precision, F1-score and accuracy, plus confusion matrices and learning curves. As a result, it was concluded that the best-performing model for the TC dataset was ResNet-50 with a batch-size of 256. On the other hand, VGG-16 was the best-performing model for C19 and ER datasets, with batch-size values of 64 and 256, respectively. The learning-rate value that achieved the best results was 1e-3 for all the datasets. The application of data augmentation did not help to improve the classification in none of the datasets, while fine-tuning enhanced the performance for datasets TC and C19, and mixed results were presented for ER. The maximum scores in accuracy were 94.6 %, 98.8 % and 92.7 % for TC, C19 and ER datasets, respectively. These results, alongside other metrics and considerations, helped to define the design and construction of three tutorials —one per dataset—, that comprehend the fundamental information and necessary steps to develop an artificial intelligence model capable of classifying medical images. By these means, the objective of producing educative material for medical images classification based on neural networks is met.

Tabla de Contenidos

Índice de Tablas

Índice de Figuras

Siglas

AD Enfermedad de Alzheimer (Alzheimer's Disease)

AI Inteligencia Artificial (Artificial Intelligence)

ANN Red Neuronal Artificial (Artificial Neural Network)

AUC Área Bajo la Curva (Area Under the Curve)

CNN Red Neuronal Convolucional (Convolutional Neural Network)

CT Tomografía Computarizada (Computed Tomography)

DL Aprendizaje Profundo (Deep Learning)

DT Árbol de Decisión (Decision Tree)

GT Ground Truth (Ground Truth)

KNN K-Vecinos más Cercanos (K-Nearest Neighbour)

LR Regresión Logística (Logistic Regression)

MCI Deterioro Cognitivo Leve (Mild Cognitive Impairment)

ML Aprendizaje Automático (Machine Learning)

NB Naive Bayes (Naive Bayes)

ReLU ReLU (Rectified Linear Unit)

RF Bosque Aleatorio (Random Forest)

ROC Característica Operativa del Receptor (Receiver Operating Characteristic)

SL Aprendizaje Supervisado (Supervised Learning)

SSL Aprendizaje Semisupervisado (Semisupervised Learning)

SVM Support Vector Machine (Support Vector Machine)

TL Transferencia de Aprendizaje (Transfer Learning)

UL Aprendizaje no Supervisado (Unsupervised Learning)

1 Introducción

1.1 Introducción General

Las imágenes médicas constituyen una de las herramientas esenciales en el diagnóstico médico, estas son de distinta naturaleza y abarcan una gran cantidad de patologías [1]. La idea de construir instrumentos computacionales que hagan uso de ellas como soporte médico en las etapas de diagnóstico y evaluación de enfermedades, ha sido un desafío que muchos científicos han propuesto e intentado llevar a cabo. El soporte que estos instrumentos entregan radica, principalmente, en la enorme e importante población que no cuenta con la posibilidad de recibir una atención óptima por falta de personal capacitado, ya sea porque su hogar no dispone de centros médicos cercanos o porque simplemente el equipo de profesionales es limitado. Sin embargo, este no es el único propósito de estas tecnologías, ya que su implementación permite complementar y reforzar el veredicto del médico, logrando un aumento en la precisión y minuciosidad del diagnóstico.

Los recientes avances en la tecnología y la naturaleza open-source de herramientas de programación, han permitido expandir la creación de modelos inteligentes a toda persona que desee hacerlo. Como resultado, en la actualidad existen numerosos algoritmos para tareas relacionadas al diagnóstico médico, siendo una de ellas la clasificación.

La clasificación consiste en asignar un elemento a una categoría que lo describa y, dependiendo de las características del dato y el rango de opciones, existen distintos tipos. Por una parte, la clasificación binaria permite asignar una de dos posibles etiquetas al elemento, siendo estas generalmente “verdadero o falso”, “sí o no” o un ejemplo asociado a un diagnóstico, “presencia de tumor o ausencia de tumor”. Por otra parte, la clasificación multiclase permite asignar una de más de dos etiquetas disponibles, por ejemplo “leve, medio o grave”. Finalmente, la clasificación multi-etiqueta, permite clasificar a un elemento en más de una categoría [2].

Considerando la importancia en el desarrollo de estas herramientas de inteligencia artificial, la enorme variedad de patologías en las que se pueden aplicar y el hecho de que la información disponible no siempre existe de forma unificada, se propone la generación de una base de datos de carácter educativo en la clasificación de imágenes médicas utilizando algoritmos de Aprendizaje Profundo, abarcando distintos tipos de clasificación, imágenes y patologías.

La base de datos consiste en tutoriales en formato notebook, desarrollados en lenguaje de programación Python utilizando materiales de libre acceso, como la plataforma Google Colab y bases de datos públicas de imágenes médicas. Para su desarrollo, se realiza un exhaustivo estudio teórico-práctico sobre los componentes esenciales que rigen el funcionamiento de los algoritmos. El estudio incluye la implementación de tres modelos de redes neuronales convolucionales en bases de datos asociadas a las patologías tumores cerebrales, Covid-19 y enfermedades de la retina, con el objetivo de definir los que obtengan mejor rendimiento en cada una de ellas. La evaluación de rendimiento para cada modelo considera numerosas alternativas en la configuración de sus parámetros principales, además de la aplicación de técnicas complementarias con la finalidad de optimizar el nivel de la clasificación.

La articulación de ambos aspectos del trabajo, es decir, los fundamentos teóricos y análisis práctico cubiertos en este informe, junto con los tutoriales desarrollados, propicia una sólida formación en tareas de clasificación de imágenes utilizando redes neuronales convolucionales.

1.2 Objetivos

1.2.1 Objetivo General

Generar tutoriales para la clasificación de imágenes médicas utilizando redes neuronales, con fines educativos.

1.2.2 Objetivos Específicos

- Buscar, seleccionar y estudiar las bases de datos públicas de imágenes a utilizar.
- Para cada base de datos seleccionada, aplicar diferentes modelos de redes neuronales, basándose en la literatura científica y seleccionar los de mejor desempeño en tareas de clasificación.
- Generar un tutorial para cada base de datos, que describa el o los modelos seleccionados, considerando todo el proceso (preprocesamiento, entrenamiento, prueba y evaluación de resultados).

1.3 Alcances y Limitaciones

- El trabajo se realizará empleando herramientas open-source, incluyendo la plataforma Google Colab para la construcción y ejecución de los scripts en lenguaje de programación Python y para la generación de los tutoriales, aprovechando la modalidad notebook y la integración de la plataforma con todas las bibliotecas necesarias para su ejecución.
- Todas las bases de datos involucradas en el trabajo son de carácter público y de libre acceso.
- El trabajo solamente cubre tareas de clasificación, dejando otras aplicaciones del Aprendizaje Profundo relativas a imágenes médicas fuera del estudio.

1.4 Metodología

En primer lugar, se seleccionarán las bases de datos a utilizar y serán estudiadas las patologías asociadas. Las fuentes para la obtención de los datos serán sitios web certificados como Kaggle, Papers With Code, IEEE DataPort, entre otros. Se estudiarán las características de las imágenes en ellas y, de ser necesario, se aplicarán todos los procesamientos para que su uso sea óptimo. También se estudiará la variedad de algoritmos existentes y su rendimiento en base a trabajos previos con objetivos similares.

Para cada base de datos, se implementarán modelos de redes neuronales convolucionales destacados en la literatura, con diversas configuraciones, y se seleccionará la de mejor desempeño. Una vez que los mejores modelos sean definidos, se implementarán técnicas para mejorar el rendimiento. Finalmente, cuando los resultados obtenidos sean satisfactorios, se procederá a la generación de los tutoriales, que incluirán todo lo necesario para la ejecución del código y comprensión del trabajo.

En cuanto a las herramientas computacionales a utilizar, el lenguaje de programación será exclusivamente Python y la plataforma de desarrollo será Google Colab. Finalmente, y en caso de ser necesario, se emplearán otras herramientas de diseño disponibles para la construcción de diagramas y figuras.

1.5 Temario

El desarrollo del trabajo está estructurado de la siguiente manera:

- **Capítulo 2:** Se presenta un estudio bibliográfico referente a las bases del Machine Learning y Deep Learning, enfatizando en este último la Red Neuronal Convolucional, donde se estudian los conceptos y componentes que rigen su funcionamiento. Adicionalmente, se destacan modelos de gran prestigio en tareas de clasificación. Finalmente, se revisan trabajos del estado del arte que emplean redes neuronales convolucionales en tareas afines.

- **Capítulo 3:** Se exponen las bases de datos de imágenes médicas utilizadas en el desarrollo del trabajo, junto con una introducción a las patologías asociadas.

- **Capítulo 4:** Se presenta el procedimiento realizado para la implementación y optimización de los modelos, el cual está estructurado por etapas donde se estudia el efecto que diversas configuraciones tienen sobre el rendimiento de la clasificación.

- **Capítulo 5:** Se exponen los resultados obtenidos en cada uno de los experimentos realizados, además de una comparación de rendimiento y una evaluación del proceso de entrenamiento de los modelos. Por último, se describen los tutoriales construidos.

- **Capítulo 6:** Se presentan las conclusiones finales del trabajo, junto con discusiones y posibles trabajos futuros.

2 Capítulo 2. Estudio Bibliográfico

2.1 Introducción

En este capítulo se revisan brevemente los conceptos y orígenes de la Inteligencia Artificial (Artificial Intelligence, AI), el Aprendizaje Automático (Machine Learning, ML), y la evolución de este último a Aprendizaje Profundo (Deep Learning, DL). Además, se revisan algoritmos clásicos o convencionales del ML y del DL, profundizando en este último la Red Neuronal Convolucional (Convolutional Neural Network, CNN). También, se exponen tres modelos de CNN destacados, junto con sus principales características. Por último, se presentan trabajos previos que implementan modelos de CNN en tareas de clasificación, utilizando distintos tipos de imágenes médicas referentes a diversas patologías.

2.2 Inteligencia Artificial, Machine Learning y Deep Learning

La inteligencia artificial es una rama de las Ciencias de la Computación y se define como la capacidad que tiene un computador (o máquina) de imitar la forma de razonar de un humano, sorteando obstáculos referentes a la capacidad limitada de aprendizaje y al tiempo que conlleva aprender a este último [3]-[4].

El Machine Learning por su parte, cuyo funcionamiento es basado en inteligencia artificial y en la estadística, hace referencia a los diversos algoritmos desarrollados para el aprendizaje de un computador a través del análisis de datos. El origen del ML ocurre en periodos relativamente cercanos a los años de aparición de los primeros computadores. Esto es debido a que uno de los fines para los que un computador fue pensado, era el análisis de grandes cantidades de datos haciendo uso de estos algoritmos [3]. El ML es uno de los campos de la IA que más rápido se ha desarrollado y evolucionado, sobre todo en las últimas décadas [3]-[5].

El Deep Learning es una rama del ML. Los algoritmos de DL se diferencian con los de ML por la profundidad y complejidad de su arquitectura, donde su columna vertebral es una Red Neuronal Artificial (Artificial Neural Network, ANN). El origen del Deep Learning se vio inspirado en el trabajo de Hubel y Wiesel en el año 1959. En él, se describe que el aprendizaje visual está estructurado en base a capas neuronales de variable profundidad, donde las primeras son las encargadas de aprender a

reconocer las características más sencillas —como la forma de un objeto, por ejemplo— y, a medida que se avanza en la profundidad de estas, el reconocimiento es más específico, discerniendo entre propiedades más complejas como texturas, colores y tamaños. En otras palabras, existe jerarquía en el proceso de aprendizaje, yendo de lo menos complejo a lo más complejo [4]-[6].

En cuanto a las herramientas computacionales del DL, estas surgen en el año 2012 con la creación de la base de datos ImageNet [7], que consiste en más de 15 millones de imágenes pertenecientes a distintas clases. Utilizando esta biblioteca de imágenes, Krizhevsky et al. [8] propusieron un modelo de DL, "AlexNet", para la clasificación de estas, marcando un hito en la evolución del ML al DL.

La manera en que los algoritmos de ML y de DL son entrenados para adaptar y mejorar sus resultados a medida que obtienen conocimiento, se realiza a través de Aprendizaje Supervisado (Supervised Learning, SL) o Aprendizaje no Supervisado (Unsupervised Learning, UL). En el primero, el algoritmo conoce cuáles son los resultados reales, lo que se conoce como Ground Truth (Ground Truth, GT) y, a partir de esa certeza, modifica su comportamiento para reducir la diferencia entre lo que su resultado indica y la realidad. Este tipo de aprendizaje es el que se utiliza comúnmente en tareas de clasificación. El aprendizaje no supervisado, por su parte, no cuenta con GT y su uso va enfocado a tareas de agrupación —también conocido como clustering— de datos que comparten características similares. Finalmente, el Aprendizaje Semisupervisado (Semisupervised Learning, SSL) es una combinación de ambos tipos de aprendizaje [5]-[9].

2.3 Algoritmos Convencionales de Machine Learning

A continuación se revisan brevemente algunos algoritmos convencionales de Machine Learning.

- La Regresión Logística (Logistic Regression, LR), es un algoritmo basado en estadísticas y probabilidades. Su nombre es asignado ya que es precisamente la función logística la que es empleada para el cálculo de las probabilidades que permiten la clasificación. Su fortaleza o ventaja se da en tareas que involucran datos que resultan sencillos de dividir de forma lineal. Como método de regularización, el cual es un sistema que permite reducir el error en el entrenamiento [6], usa dos técnicas: "L1" y "L2", ambas utilizadas para obtener una reducción en la cantidad de características. L1 (o Lasso) lleva ciertos coeficientes a cero, por lo que su uso es recomendado si es que la cantidad de características relevantes es pequeña. Por otra parte, L2 (o Ridge) también ejecuta una modificación en los coeficientes pero, en vez de ser llevados a cero, los valores son reducidos. Ambas técnicas contribuyen disminuyendo el impacto de características

irrelevantes en el análisis realizado por el algoritmo. La desventaja de la LR, es que la función logística asume la linealidad entre las variables dependientes e independientes, por lo que su desempeño no es bueno si la complejidad de los datos es alta [2] [10].

- Support Vector Machine (Support Vector Machine, SVM) representa los datos, que pueden ser linealmente separables o no, en espacios de características (feature spaces) de dimensionalidad variable y dependiente del número de estas. Luego de obtener esta representación, realiza una separación de las clases a través de un "hiper-plano" que maximiza la distancia entre ellas [11]. En los casos en los que no resulta sencillo realizar una separación lineal entre dos o más clases, la aplicación de una función no lineal transforma el espacio de características a uno nuevo de dimensión en la que sí es posible lograr dicha separación, destacando de esta forma su buen rendimiento en problemas con alta dimensionalidad. Sin embargo, su desventaja es que no posee un buen rendimiento si los datos presentan mucho ruido [2]-[12].

- Por otra parte, uno de los algoritmos más sencillos en su implementación e interpretación es el Árbol de Decisión (Decision Tree, DT). Su funcionamiento consiste en evaluar sucesivamente si ciertas condiciones se cumplen o no. La primera condición es evaluada en el nodo raíz, que diverge a través de dos ramas en dos nodos hijos, también conocidos como nodos internos. En cada uno de estos nodos hijos, el proceso es repetido hasta que se llega a una clasificación o condición final, lo que se conoce como nodo hoja [2]. Los parámetros empleados para establecer la decisión son los Coeficientes de Gini y la Entropía [12]. Una de las ventajas importantes de este algoritmo, es que los parámetros mencionados permiten al humano comprender cómo se da el proceso de la clasificación. A partir del algoritmo DT se desprende uno nuevo, conocido como Bosque Aleatorio (Random Forest, RF), el cual junta o ensambla más de un árbol. Su funcionamiento involucra la división de los datos en los distintos árboles y realiza el análisis de forma paralela. La clasificación final se logra considerando la mayoría simple. Su ventaja es que, debido a la división de los datos en los distintos árboles, reduce el riesgo de obtener un aprendizaje específico y suele aumentar la exactitud [2]. Sin embargo, un bosque muy grande afecta de forma negativa el tiempo de ejecución ya que requiere de una mayor cantidad de datos [13].

- Naive Bayes (Naive Bayes, NB) es un algoritmo construido en base a la naturaleza probabilística que rige al Teorema de Bayes. Su ventaja, es que el conjunto de entrenamiento no requiere una gran cantidad de datos, debido a su arquitectura simple que solo involucra probabilidades. Su desventaja, sin embargo, recae en que realiza la suposición "ingenua" de que las características son independientes, lo cual no es cierto en la mayoría de problemas del mundo real [2]-[12].

- Uno de los primeros algoritmos de clasificación supervisada fue el K-Vecinos más Cercanos

(K-Nearest Neighbour, KNN). A partir de la representación en un plano de datos cuya clase es conocida, un nuevo dato puede ser asignado a alguna de ellas basado en la distancia. El valor de K es asignado manualmente e indica el número de vecinos que se va a considerar. Por ejemplo, al seleccionar un valor de 5 para K, se identifican los 5 vecinos más próximos al dato que se quiere clasificar. Para esos 5, se revisan su clases y aquella que más se repite es asignada al nuevo dato. Además del valor de K, es importante asignar el tipo de distancia a considerar, siendo una de las más utilizadas la Euclídea. La principal ventaja de KNN, es que no requiere un proceso de entrenamiento. Por otra parte, la simplicidad del modelo y lo poco representativo que es para problemas complejos, se convierte en una desventaja. Además, si los datos analizados son muchos, el algoritmo es lento porque debe calcular la distancia para todos los datos etiquetados [11]-[13].

- Por último, la Red Neuronal Artificial es un algoritmo inspirado en el cerebro humano, específicamente en la forma en que la unión de distintas neuronas realizan asociaciones e interpretaciones complejas. Es un modelo compuesto por capas de neuronas interconectadas, donde dichas capas poseen un coeficiente de peso modificable a medida que la información es procesada. La función de transferencia es la encargada de la activación de ciertas neuronas e inhibición de otras en el proceso de entrenamiento, obteniendo las características importantes y, a partir de ellas, la posibilidad de conseguir una clasificación como salida del sistema [11]-[14]. Su gran ventaja es que es un algoritmo adaptativo o, en otras palabras, su modificación y optimización lo realiza en base a la experiencia, lo cual otorga versatilidad al permitir el trabajo sobre diferentes tipos de datos [15].

A modo de síntesis, la Tabla 2.1 expone las principales ventajas y desventajas de los algoritmos de ML revisados.

Tabla 2.1: Ventajas y desventajas de algoritmos convencionales del Machine Learning.

Algoritmo	Ventajas	Desventajas
Regresión logística	Buen desempeño en datos linealmente separables.	Mal desempeño en datos complejos.
Support Vector Machine	Buen desempeño en problemas con alta dimensionalidad.	Mal desempeño en alta presencia de datos ruidosos.
Árbol de Decisión	Fácil de entender e implementar.	Suele presentar sobreajuste.
Bosque Aleatorio	Presenta menor riesgo de sobreajuste y mayor exactitud que un árbol de decisión.	Mayor tiempo de ejecución y requiere más datos.
Naive Bayes	Simple de implementar y requiere pocos datos para su entrenamiento.	Asume la independencia entre datos.
K-Vecinos más Cercanos	Simple de implementar y no requiere un proceso de entrenamiento.	Alto tiempo de ejecución al usar grandes cantidades de datos o de alta complejidad.
Red Neuronal Artificial	Posee un aprendizaje adaptativo en base a la experiencia y tiene alta tolerancia a datos imprecisos o faltantes.	Alto costo computacional al usar muchos datos y complejo de comprender debido a su naturaleza ”black-box”.

2.4 Red Neuronal Convolucional

La Red Neuronal Convolucional es uno de los algoritmos más populares del DL y su funcionamiento es basado en una ANN. Sin embargo, la gran diferencia entre una CNN y los algoritmos de ML convencional, es que la entrada al modelo es la imagen en sí. Otra diferencia es que la CNN usa la operación convolución para la extracción de características en vez del sistema de pesos que usa la ANN, esto es porque la convolución es una operación que involucra filtros que por sí entregan peso. Una ventaja de la CNN, es que esta se encarga de obtener las características por sí misma y no requiere de un proceso previo de selección manual. Esto significa que su aprendizaje lo logra de forma adaptativa en base a la experiencia obtenida en el proceso de entrenamiento, al igual que la ANN [6].

2.4.1 Arquitectura de una CNN

La arquitectura por defecto de una CNN consta de una serie de capas que se encargan de obtener las características relevantes de la imagen de forma automática y adaptativa. La entrada o input al sistema es la imagen, mientras que la salida o output es la clasificación. Las capas entre el input y output son esencialmente de tres tipos y corresponden a las capas profundas, capas ocultas o hidden layers. Estas capas son: la capa de convolución, la capa de agrupación (o pooling) y la capa densa o totalmente conectada (dense o fully-connected). El conjunto de estas se conoce como bloque y una CNN involucra la unión de varios de ellos. La salida de cada una de las capas se convierte en la entrada de la siguiente, de forma sucesiva, hasta alcanzar el final de la red [16]. La Fig. 2.1 muestra la arquitectura genérica de una red neuronal convolucional.

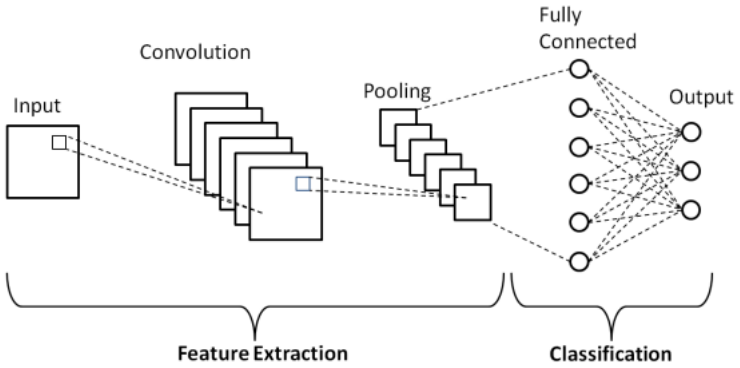


Fig. 2.1: Arquitectura genérica de una CNN. Las capas de convolución y agrupación pertenecen a la etapa de extracción de características, mientras que las capas totalmente conectadas forman parte de la clasificación [17].

2.4.1.1 Capa de Convolución

En esta capa la operación protagonista es la convolución, una operación lineal que se realiza aplicando un filtro de convolución (o kernel) sobre un arreglo de números que, en este caso, es la imagen y recibe el nombre de tensor. Esta operación es la encargada de realizar la extracción de características del tensor y su resultado es conocido como mapa de características o feature map [16]. La Fig. 2.2 ejemplifica este procedimiento.

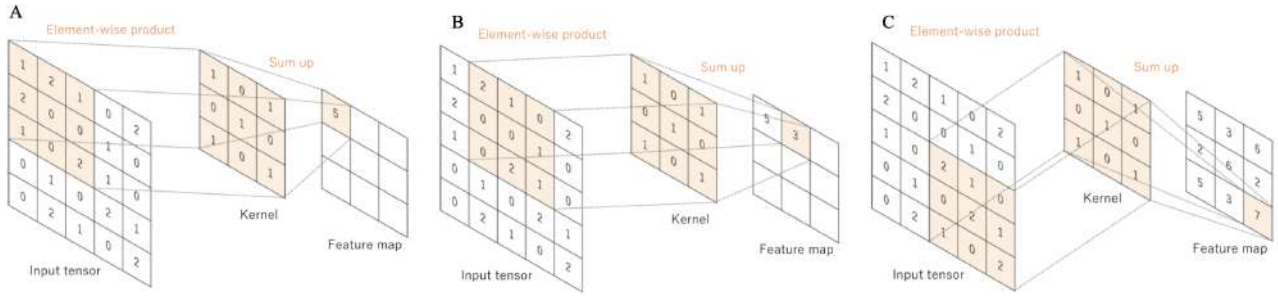


Fig. 2.2: Ejemplo de operación de convolución para un kernel de dimensión 3x3 sobre un tensor de dimensión 5x5. A) Convolución con el primer elemento del tensor. B) Convolución con el segundo elemento del tensor. C) Resultado final de la convolución [16].

2.4.1.2 Capa de Agrupación

La capa de agrupación o capa de pooling recibe este nombre porque su trabajo es, a partir de una vecindad de valores en un arreglo, seleccionar tan solo uno de ellos. Esta selección se realiza en base a diversas operaciones predefinidas. Una de ellas, en la cual se obtiene el promedio de los valores a seleccionar, recibe el nombre de agrupación por valor promedio o average pooling. Otra operación popular es la obtención del valor mayor, proceso conocido como agrupación por el valor mayor o max pooling. El objetivo de esta capa es lograr una reducción en la dimensión del mapa de características, para que el coste computacional sea menor, sin pérdida de información relevante [16]. Una ejemplificación de este proceso se describe en la Fig. 2.3.

2.4.1.3 Capa Densa

La capa densa o fully-connected es aquella que se obtiene luego de un proceso de aplanamiento del arreglo de entrada. Este proceso consiste en reducir su dimensión de dos a una o, en otras palabras, convertir el arreglo de dos dimensiones a un vector. La tarea de esta capa es trazar un mapa de conexión entre las características extraídas para diseñar la salida del sistema, o sea, obtener la clasificación. La

capa final, que también es una capa densa, contiene una cantidad de nodos igual a la cantidad de clases disponibles [16]. La Fig. 2.4 ilustra un ejemplo en el cual, a partir de un arreglo de dimensiones 3x3, se obtiene un vector de dimensión 1x9.

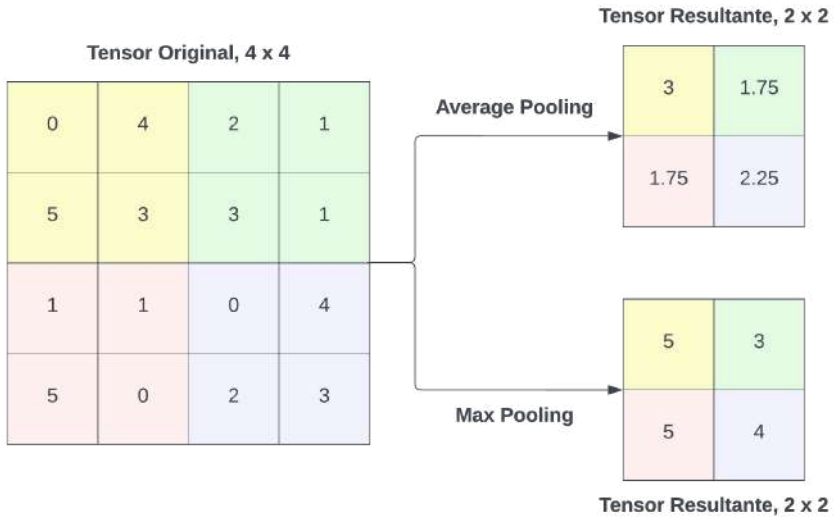


Fig. 2.3: Ejemplo de operación de agrupación utilizando un valor de ventana 2x2 para un tensor con dimensión original 4x4.

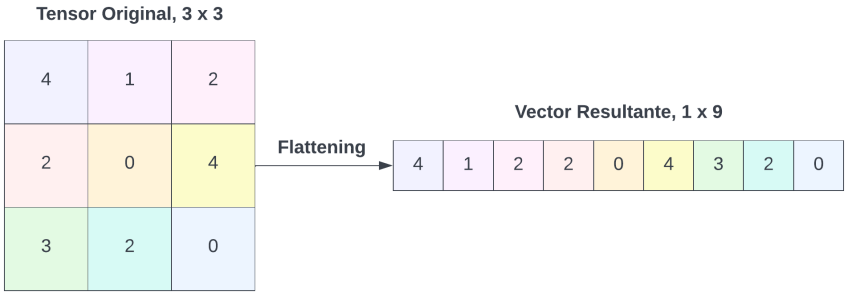


Fig. 2.4: Ejemplo de operación flattening o aplanamiento. A partir de un arreglo bidimensional de dimensión 3x3 se obtiene un arreglo unidimensional.

2.4.1.4 Funciones de Activación

Dependiendo de la arquitectura, luego de operaciones de convolución, agrupación o aplanamiento, se suele aplicar una función de activación para reducir los costos computacionales y aumentar la no-linealidad del problema. De esta forma, se facilita la clasificación al obtener una función capaz de separar los datos de forma correcta. La función de activación empleada generalmente es la función

rectificador ReLU (Rectified Linear Unit, ReLU). ReLU (ecuación 2.4.1), se encarga de llevar todos los valores negativos a cero y mantener los positivos sin modificación.

Por otra parte, la capa final también cuenta con una función de activación, cuya labor es normalizar los valores obtenidos en los nodos finales y representarlos como probabilidades, a partir de las cuales se designa la clase predicha. Para clasificación binaria, la función de activación empleada es la función sigmoide (ecuación 2.4.2), mientras que para la clasificación multiclase, es la función softmax (ecuación 2.4.3) la que realiza dicha tarea [14]-[16]-[18].

$$Relu(z) = max(0, z) \tag{2.4.1}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4.2}$$

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad para\ i = 1, 2, \ldots, K \tag{2.4.3}$$

Donde z: valor observado (en ecuación 2.4.1) y probabilidad predicha (en ecuaciones 2.4.2 y 2.4.3); i: dato observado; K: número total de datos.

2.4.2 Entrenamiento de una CNN

2.4.2.1 Hiperparámetros

El proceso de entrenamiento considera la configuración de hiperparámetros previo a su inicio. Uno de ellos es el hiperparámetro Epochs (o épocas), que es el número de veces que el proceso es realizado. En otras palabras, un Epoch es cuando todas las imágenes cruzan la red neuronal una vez. Este hiperparámetro toma valores enteros y su valor óptimo depende de cada problema. Por otra parte, el hiperparámetro Batch-Size (o tamaño del lote), configura la cantidad de datos que el algoritmo considera previo a la actualización de pesos y suele ser ajustado a potencias de dos.

En cuanto al proceso de aprendizaje de la red, este involucra la actualización automática de los valores que conforman los kernels presentes en las capas de convolución, y de los pesos en las capas densas, pues el fin es lograr valores óptimos en estos elementos, y así obtener el mejor desempeño tanto en la extracción de características relevantes, como en la clasificación. Este trabajo de modificación de

pesos es realizado por un optimizador junto a una función de pérdida (loss function), ambos también hiperparámetros. La función de pérdida calcula un valor de error, capaz de medir la diferencia entre la salida de la red y la realidad o, en otras palabras, la diferencia entre el output y el ground truth. Por otra parte, al fin de cada iteración, o sea, cuando un batch de datos ha recorrido la red, el optimizador actualiza los pesos del modelo con el objetivo de minimizar el error. Esta actualización la hace en base a un gradiente que, a su vez, es obtenido a partir de la función de pérdida. Dicho de otro modo, el optimizador supervisa que los pesos sean actualizados a favor de la minimización del valor de pérdida. Finalmente, la función de pérdida utilizada depende del tipo de clasificación que se desea obtener, o sea, si es binaria o multiclase. "Binary Cross-entropy" y "Categorical Cross-entropy" son las funciones usualmente utilizadas en cada caso, respectivamente.

Por último, el Learning-rate (lr) —que también es parte del optimizador— es considerado el hiperparámetro más importante. Consiste en un valor númerico, generalmente configurado entre 1e-1 y 1e-7, que decreta qué tan rápido o qué tan lento el modelo aprende. Más específicamente, establece qué tan significativa es la alteración de los pesos por parte del optimizador para obtener la convergencia a un valor de pérdida o de error mínimo. Un lr muy pequeño puede significar un entrenamiento sumamente lento y, por lo tanto, inviable, mientras que un lr demasiado grande generalmente implica que el valor de pérdida no converja a un mínimo. Por lo tanto, se suelen estudiar distintos valores de lr para hallar el óptimo, pues dependerá de la complejidad de cada problema [16]-[19].

Las ecuaciones matemáticas para el optimizador Adam, las funciones de pérdida y las funciones de activación mencionadas, se exponen en las secciones siguientes.

2.4.2.2 Optimizador

Como se mencionó anteriormente, el optimizador es el componente encargado de actualizar los pesos presentes en las capas de la red. Adam (Adaptative Moment Estimation), es uno de los optimizadores más utilizados en el DL. Su popular uso se debe a que es eficiente computacionalmente, pues requiere pocos recursos de memoria, converge rápidamente y porque tiene buen rendimiento en tareas que involucran una gran cantidad de datos y parámetros.

Adam es un optimizador que combina elementos de otros dos algoritmos destacados: AdaGrad (Adaptive Gradient Algorithm) y RMSProp (Root Mean Square Propagation). Específicamente, acopla los sistemas de actualización de pesos basados en gradiente que ambos algoritmos presentan. La actualización de pesos en Adam utiliza la medias móviles del gradiente (denotado por g): $m = \beta_1 m_{t-1} + (1 - \beta_1) g_t$ y del cuadrado del gradiente $v = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ en cada iteración t. β_1 y

β_2 son índices que aportan en el decrecimiento exponencial de las médicas móviles y sus valores son 0.9 y 0.999, respectivamente. Debido a que m y v son inicializadas en cero, en las primeras iteraciones del entrenamiento se presenta una aproximación al valor mínimo muy distante del óptimo, por lo cual sufren una corrección a través de un sesgo. Luego, las médias móviles del gradiente y del cuadrado del gradiente resultan en $\hat{m}_t = m_t/(1-\beta_t^1)$ y $\hat{v}_t = v_t/(1-\beta_t^2)$, respectivamente. Finalmente, la actualización de pesos se da según la ecuación 2.4.4 [20].

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.4.4}$$

Donde *t* denota la iteración actual, θ representa el peso, α es el learning-rate, $\hat{m}$ y $\hat{v}$ son las medias móviles del gradiente (corregidas), y ϵ es una constante configurada por defecto en 1e-8.

2.4.2.3 Funciones de Pérdida

Las expresiones matemáticas para las funciones de pérdida utilizadas para tareas de clasificación son expuestas a continuación.

Para clasificación binaria, se tiene la "Binary Cross-Entropy":

$$Loss = -(y \log(p) + (1-y) \log(1-p)) \tag{2.4.5}$$

Donde y: valor binario indicando la clase observada; p: probabilidad de que la clase observada corresponda a la clase real.

Para clasificación multiclase, se tiene la "Categorical Cross-Entropy":

$$Loss = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2.4.6}$$

Donde o: clase observada; c: clase real; p: probabilidad de que o corresponda a c; y: valor binario indicando si o y c coinciden; M: número de clases.

2.4.3 Conjuntos de Datos

Para la implementación del modelo de CNN, se necesitan al menos dos conjuntos de datos. El primero, conocido como conjunto o set de entrenamiento, es el que contiene los datos que utiliza la

red para aprender a identificar las características relevantes y extraerlas. El segundo, conocido como conjunto o set de prueba, contiene los datos para evaluar el rendimiento del modelo. Adicional a los dos conjuntos mencionados, puede existir un tercero: el conjunto de validación. Este conjunto permite obtener información, mientras el modelo es entrenado, sobre qué tan bien está aprendiendo y desempeñándose sobre datos nuevos y así fijar los hiperparámetros óptimos. Es de suma importancia que los elementos que componen los conjuntos sean distintos para lograr un entrenamiento adecuado y un buen nivel de generalización. De esta manera, se consigue un buen rendimiento sobre el conjunto de prueba [16].

Para obtener una generalización adecuada, es importante que el conjunto de entrenamiento sea lo suficientemente representativo, tanto en la cantidad de imágenes, como en las características presentes en ellas, permitiendo una clara diferenciación entre las clases [4]-[9]. Al trabajar con imágenes médicas, no está garantizado que esto sea así, ya que existe una cantidad limitada de ellas, lo cual es uno de los principales inconvenientes en el DL [19]. Debido a esto, el conjunto de entrenamiento no es lo suficientemente robusto y, si bien el modelo logra aprender las características presentes en las imágenes, lo hace de manera muy específica. Esto concluye en una mala generalización y, en consecuencia, en un mal desempeño con el conjunto de prueba. Este fenómeno es conocido como sobreajuste u overfitting. Por otra parte, puede ocurrir infrajuste o underfitting, en el que las características relevantes presentes en las imágenes no son suficientes y no permiten que el modelo aprenda de ellas. En otras palabras, el overfitting resulta en un aprendizaje específico de los datos mientras que el underfitting resulta en un aprendizaje incompleto [6]-[16]. La Fig. 2.5 muestra una forma de evaluar si el entrenamiento del modelo se está dando de forma correcta o no, en base a curvas de aprendizaje.

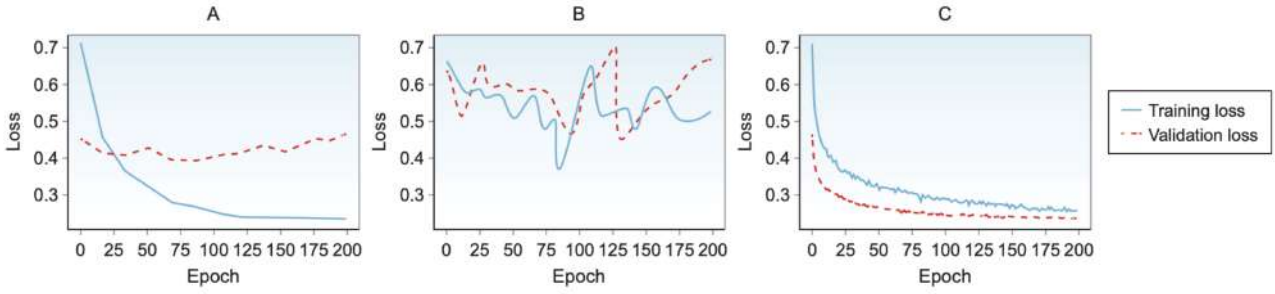


Fig. 2.5: Curvas de aprendizaje. A) el modelo presenta overfitting y mala generalización en el conjunto de validación. B) el modelo presenta underfitting y mala generalización en ambos conjuntos. C) el modelo presenta buen ajuste y buena generalización en el set de validación [21].

2.4.4 Técnicas de Regularización

Existen técnicas de regularización que permiten sobrellevar, en cierto grado, los fenómenos de overfitting y underfitting, permitiendo una mejora en la capacidad de generalización del modelo [22]. Algunas de estas técnicas son mencionadas a continuación.

2.4.4.1 Dropout

La técnica dropout —o abandono, en español— es popularmente empleada en modelos de CNN y consiste en la desconexión aleatoria de uniones en la red, o sea, la inhibición de neuronas. La selección de neuronas que son desconectadas es realizada de forma aleatoria, a partir de un parámetro númerico que va desde 0 a 1, y que representa el porcentaje de neuronas que se inhiben. La aplicación de dropout resulta en la omisión de algunos pesos en la fase de entrenamiento y, por lo tanto, en una disminución del aprendizaje específico y un aumento en la generalización. La Fig. 2.6 muestra un ejemplo de la aplicación de esta técnica [6]-[23].

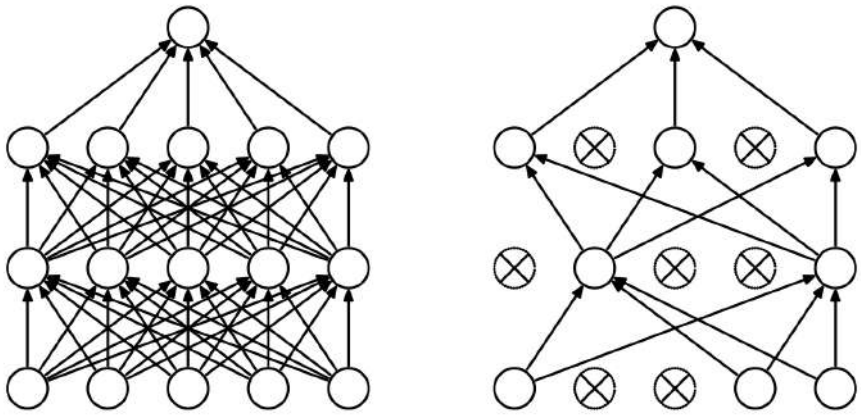


Fig. 2.6: Efecto sobre la red luego de la aplicación de dropout. a) red previa a la aplicación de dropout. b) red posterior a la aplicación de dropout [23].

2.4.4.2 Normalización de Batch

La normalización de batch o normalización por lotes, consiste en una capa suplementaria que, al ser situada en alguna capa profunda de la red, aplica una normalización a los valores de entrada de esta. Esta técnica permite una reducción en la cantidad de iteraciones necesarias para el entrenamiento

óptimo del modelo, lo que se traduce en una disminución en el tiempo de ejecución de esta fase. Por otra parte, también posee un efecto sobre la capacidad de generalización, similar a la que otorga la implementación de dropout, por lo que en ocasiones suprime la incorporación de este último o permite la reducción de su valor [16]-[24].

2.4.4.3 Aumento de Datos

Es una técnica que consiste en aplicar transformaciones a las imágenes originales, resultando en un aumento en la cantidad de información. Algunas de las transformaciones que se aplican son: rotación en un ángulo determinado, traslación en el eje horizontal y/o eje vertical, flip o reflejo respecto a un eje y acercamiento o alejamiento (zoom). Otras operaciones que suelen ser aplicadas incluyen el suavizado, la inversión o reajuste de canales, la adición de ruido aleatorio y modificaciones en los niveles de brillo y contraste. La aplicación de esta técnica permite aumentar la variabilidad de los datos, lo cual es beneficioso en la fase de entrenamiento del modelo [16]-[25]. La Fig. 2.7 expone algunos ejemplos de transformaciones aplicadas a una imagen. Es importante mencionar que las transformaciones a aplicar dependederán de las imágenes a utilizar, puesto que las imágenes resultantes, al ser de naturaleza médica, deben ser realistas.

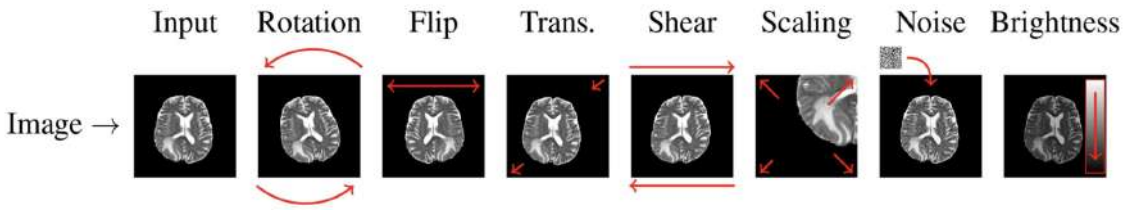


Fig. 2.7: Ejemplos de operaciones sobre imágenes en la técnica aumento de datos [26].

2.4.4.4 Callbacks

La biblioteca Keras [27] cuenta con funciones auxiliares, llamadas callbacks, que permiten mantener estable el entrenamiento de la red, monitoreando en cada momento el desarrollo de este. A continuación se mencionan tres de estas funciones que son implementadas en el trabajo.

- La primera función es ModelCheckpoint. Esta función permite guardar los modelos en un archivo cada cierta cantidad de epochs. De esta forma, si el entrenamiento es interrumpido por alguna razón, es posible reanudarlo desde el archivo respaldado. También, en vez de guardar el modelo completo, permite guardar solo los pesos para que puedan ser utilizados nuevamente en una red que presente la misma arquitectura que la red original [27].

- Otra función utilizada es Early-Stopping. Como su nombre lo indica, esta función permite detener el entrenamiento de forma anticipada. Gracias a los parámetros presentes en la función, se pueden monitorear métricas de interés y, si en una cantidad de epochs determinada —parámetro que recibe el nombre de "paciencia"— no hay una mejora en dichas métricas, el entrenamiento se detiene. La ventaja es que evita que el modelo presente sobreajuste, ya que si en las últimas iteraciones se comienza a manifestar este fenómeno, la función permite, además de detener el entrenamiento, restaurar los pesos óptimos que fueron encontrados en iteraciones anteriores [27].

- Finalmente, la función ReduceLROnPlateau opera de forma análoga a Early-Stopping, pero en vez de detener el entrenamiento, el hiperparámetro learning-rate es reducido si la métrica monitoreada no presenta mejoras en la cantidad de epochs establecidos. La ventaja de esta función, es que permite modificar este hiperparámetro sin la necesidad de detener el modelo y comenzar de nuevo [27].

2.5 Transferencia de Aprendizaje y Modelos Destacados del Deep Learning

La Transferencia de Aprendizaje (Transfer Learning, TL) consiste en emplear arquitecturas de CNN que en su fase de construcción fueron entrenadas y evaluadas en una gran cantidad de datos. Si bien la tarea de clasificación original puede ser distinta a la que se desea implementar, esto es posible gracias a que los pesos obtenidos en el proceso de entrenamiento, son lo suficientemente cualificados para identificar características comunes o genéricas de distintos objetos o elementos. Por lo tanto, emplear esta técnica resulta conveniente cuando la cantidad de datos en el problema de interés es limitada, o cuando el objetivo es aumentar el desempeño del modelo [16].

En la práctica, el TL consiste en implementar el modelo original, pero inhibiendo el entrenamiento de las capas encargadas de la extracción de características y entrenando solo las capas finales que concluyen en la clasificación. Estas últimas capas suelen ser modificadas para adecuarse a la tarea de clasificación específica que se desea lograr. Una vez que el modelo ha sido entrenado, se pueden deshinibir algunas capas y entrenar nuevamente, pero esta vez con valores de learning-rate y epochs disminuidos, para que el impacto sobre los pesos originales sea pequeño y así el rendimiento alcanzado previamente no sea alterado de forma drástica, pues la idea de este proceso es mejorar el rendimiento del modelo, no perjudicarlo. Esta técnica recibe el nombre "fine-tuning". Otra forma de aplicar TL, es utilizar el modelo original solo en la tarea de extracción de características, y luego proporcionar este resultado a un algoritmo de clasificación ajeno a la CNN, por ejemplo, a uno de los algoritmos

convencionales del ML, como DT, KNN o SVM [16].

Existen diversas arquitecturas destacadas en tareas de clasificación. Si bien la naturaleza de estas considera las capas convencionales de una CNN, muchas varían en el orden en el que son posicionadas, en la cantidad de bloques implementados y en el tipo de técnicas de regularización. Además, modelos más complejos involucran otras técnicas, como conexiones alternativas entre bloques o capas de clasificación auxiliares. Algunas de las arquitecturas destacadas son VGG-16 [28], ResNet-50 [29] e Inception-V3 [30].

2.5.1 VGG-16

La arquitectura VGG utiliza filtros de convolución de dimensión pequeña, siendo esta su principal diferencia respecto al resto de modelos estudiados. Mientras que otros modelos suelen utilizar filtros 5x5, 7x7, 9x9 e incluso de mayor dimensión, VGG utiliza filtros 3x3. Utilizar estos filtros permite, por una parte, aumentar la profundidad de la red reduciendo la cantidad de parámetros y el coste computacional, debido a que dos filtros 3x3 reemplazan a un filtro 5x5 y tres filtros 3x3 reemplazan a un filtro 7x7. Por otra parte, es posible aumentar el uso de la función de activación ReLU, lo que permite mejorar la capacidad de discriminación al momento de la clasificación. De hecho, algunas versiones de VGG también hacen uso de capas de convolución con filtros de dimensión 1x1, con el único fin de aumentar la no-linealidad al incorporar, luego de la convolución, la función ReLU. Por último, el trabajo de reducción de dimensión, es llevado a cabo por capas de agrupación por valores máximos.

El modelo VGG-16, que es el utilizado en este trabajo, tiene 224x224 píxeles como dimensión de la imagen de entrada. La siguiente sección está compuesta por cinco bloques convolucionales. El primer bloque posee dos capas de convolución con 64 filtros (o kernels), seguido por una operación max-pooling de ventana 2x2. El segundo bloque es de similares características, excepto por la cantidad de filtros, que esta vez son 128. El tercer bloque cuenta con tres capas de convolución con 256 filtros cada una, seguidos por una capa de max-pooling de ventana 2x2. El cuarto y quinto bloque son idénticos: tres capas de convolución con 512 filtros, seguidos por una capa de max-pooling de ventana 2x2. Finalmente, tres capas fully-connected de 4096, 4096 y 1000 nodos respectivamente, son las encargadas de generar la salida de la red [28]. El modelo VGG-16 se presenta en la Fig. 2.8.

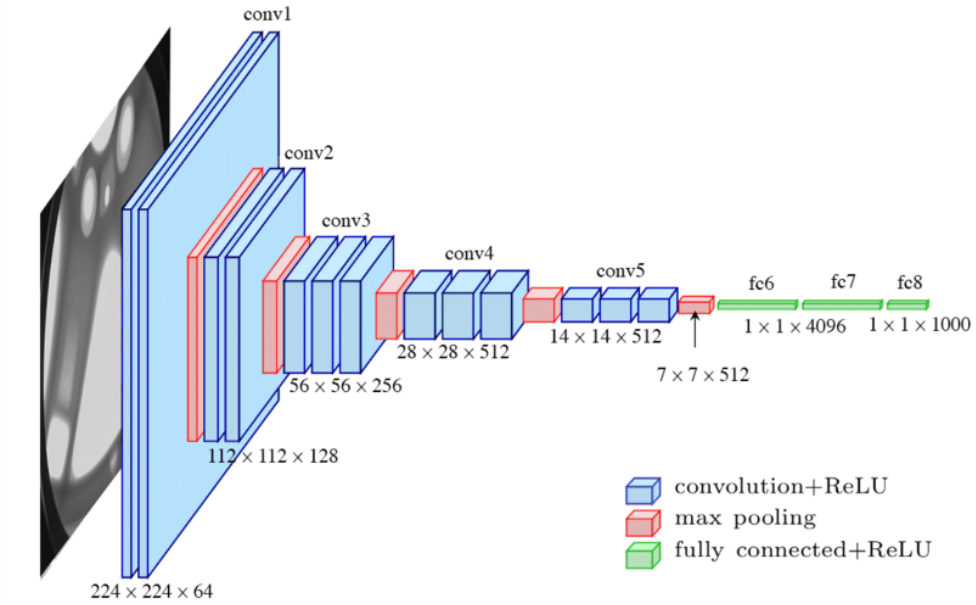


Fig. 2.8: Arquitectura del modelo VGG-16 [31].

2.5.2 ResNet-50

La cantidad de capas presentes en una red, conocida como “profundidad”, generalmente se comporta de forma directamente proporcional a la exactitud obtenida en el conjunto de entrenamiento. Sin embargo, en redes con una gran cantidad de capas, llega un punto en el que esta métrica alcanza un límite y comienza a decaer, lo que es descrito como un “problema de degradación”.

La Red Residual (ResNet) permite superar este inconveniente y lograr un entrenamiento óptimo en arquitecturas profundas. A partir de un modelo convencional, donde la información viaja directamente de capa en capa, las capas presentes en ResNet incluyen, además de las conexiones comunes, conexiones atajo (Fig. 2.9 a). Estas últimas, tomando ventaja del principio de la función residual, permiten agrupar dos o más capas realizando “identity mapping”, con el objetivo de aumentar la información que reciben las capas más profundas a partir de las capas iniciales. Dicho de otra forma, la información recibida por una capa no es solo la presente en la capa anterior, sino también la presente en el origen de la conexión atajo. Realizar este tipo de conexiones no presenta ningún problema cuando las dimensiones de las capas conectadas son las mismas. Sin embargo, cuando las dimensiones no coinciden, una operación de convolución adicional es aplicada para hacerlas coincidir.

Existen diversas versiones de la arquitectura ResNet, donde sus dos principales diferencias son la cantidad de capas y el tipo de bloques residuales o conexiones atajo que utilizan. Mientras que ResNet-18 y ResNet-34, con 18 y 34 capas respectivamente, utilizan el modelo de bloque expuesto en

la Fig. 2.9 b, ResNet-50, 101 y 152 utilizan el modelo Bottleneck, expuesto en la Fig. 2.9 c. El modelo Bottleneck, en vez de utilizar dos capas para la agrupación, utiliza tres.

Los modelos ResNet permiten demostrar la eficacia de la función residual sobre el problema de degradación y, si bien todos los modelos manifiestan mejoras sobre el problema, son aquellas con mayor cantidad de capas las que destacan [29]. El modelo ResNet-50 se expone en la Fig. 2.10.

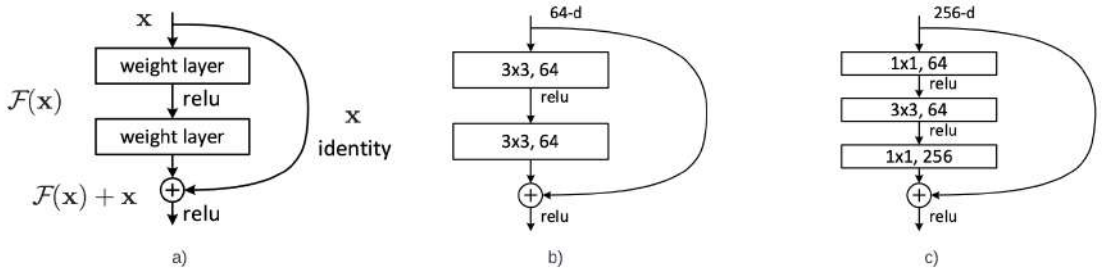


Fig. 2.9: Bloque residual. a) bloque residual genérico. b) bloque residual de dos capas. c) bloque residual de tres capas (Bottleneck) [29].

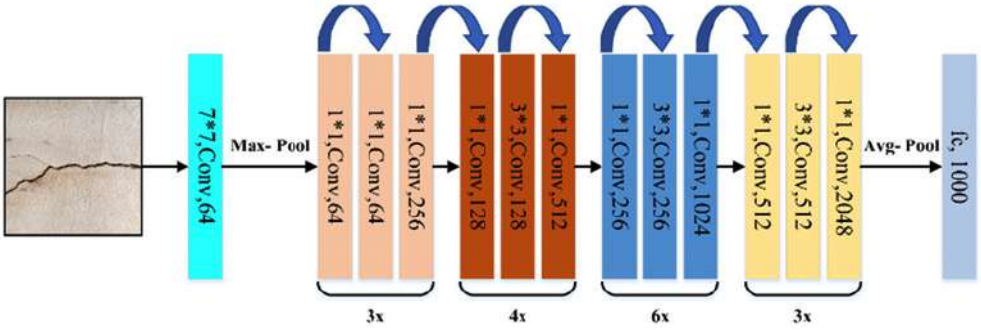


Fig. 2.10: Arquitectura de modelo ResNet-50 [32].

2.5.3 Inception-V3

El componente en el cual el modelo Inception basa su arquitectura, es llamado Módulo Inception. Este elemento se encarga de aplicar diferentes cantidades y tamaños de filtros de convolución, al igual que capas de pooling, a distintas secciones de un mismo input. Luego, los resultados son concatenados para obtener un output. Esto resulta conveniente cuando no se sabe con certeza qué tamaño de filtro se desempeña mejor en una situación específica, o cuántos de estos resultan más eficientes, permitiendo aplicar distintas alternativas y obtener beneficios de cada una de ellas. Sin embargo, esto significa un alto costo computacional. Para contrarrestar este problema, incluir una convolución de tamaño 1x1 antes de las operaciones principales es la técnica empleada. Esta operación posibilita disminuir la dimensión del input, específicamente su profundidad o cantidad de canales y, por lo tanto,

reducir el número de operaciones que un módulo Inception requiere en un 90 %, aproximadamente. Esto es viable ya que al aplicar una convolución con filtro de tamaño 1x1 a un input multicanal, es posible obtener una representación simplificada pero lo suficientemente representativa de este. Luego, al aplicar convoluciones con filtros de mayor orden sobre esta representación simplificada, el costo es reducido significativamente.

La arquitectura Inception consiste en bloques o módulos inception apilados de forma sucesiva. Inception-V3, utilizada en este trabajo, ha sufrido algunas modificaciones respecto a sus versiones anteriores y son mencionadas a continuación.

En primer lugar, busca encontrar un equilibrio entre la profundidad de la red y su grosor. La profundidad hace referencia a la cantidad de capas presentes en la red, mientras que el grosor es la cantidad de operaciones utilizadas en cada uno de los módulos. Convoluciones con un filtro de tamaño 5x5 presentes en la versión original (Inception-V1), fueron reemplazados por factorizaciones de estos en la versión siguiente (Inception-V2), ya que dos filtros de tamaño 3x3, o dos filtros de tamaño 1x3 y 3x1, permiten una representación similar a la del filtro original con un costo computacional reducido. Además de los módulos Inception descritos, en Inception-V2 se incluyen clasificadores auxiliares en algunas secciones de la red. Estos clasificadores son esencialmente funciones de activación “softmax” o “sigmoid” intermedias para monitorear que la clasificación se esté dando de forma óptima. En Inception-V3, si bien no hubo grandes modificaciones en cuanto a la estructura de los módulos, se optó por optimizar lo más posible los parámetros y capas auxiliares. Esta vez, fueron las convoluciones con filtros de tamaño 7x7 las reemplazadas por su respectiva factorización. Además, los clasificadores auxiliares son precedidos por normalización de batch. Por último, se empleó Label Smoothing, un mecanismo de regularización que reduce el riesgo de sobreajuste haciendo a los clasificadores más “ingenuos”, al reducir la distancia entre puntajes de probabilidad otorgado a las clases [33]-[30]. La arquitectura para Inception-V3 se muestra en la Fig. 2.11.

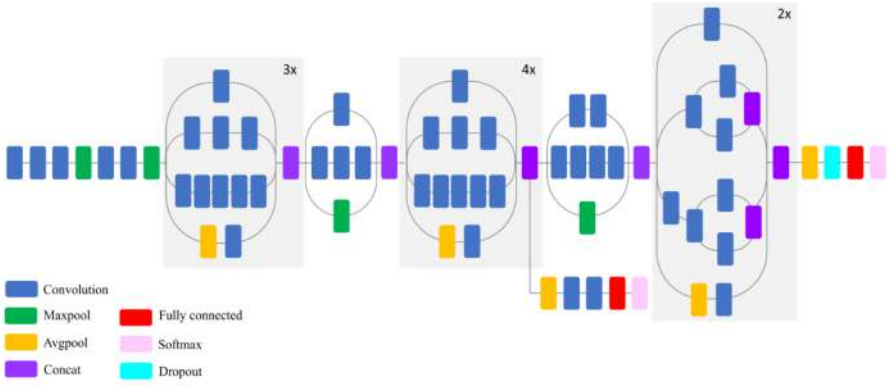


Fig. 2.11: Arquitectura de modelo Inception-V3 [34].

2.6 Métricas de Evaluación

Son diversas las métricas empleadas para la evaluación del rendimiento de los modelos. Estas, permiten realizar un análisis específico por cada clase y también de forma generalizada. Para cada una de las métricas descritas a continuación:

- True positives o verdaderos positivos (TP): clases identificadas como positivas cuando la etiqueta real también es positiva.
- True negatives o verdaderos negativos (TN): clases identificadas como negativas cuando la etiqueta real también es negativa.
- False positives o falsos positivos (FP): clases identificadas como positivas cuando la etiqueta real es negativa.
- False negatives o falsos negativos (FN): clases identificadas como negativas cuando la etiqueta real es positiva.

Las métricas empleadas para la evaluación fueron las siguientes:

- Recall o Exhaustividad (también llamada Sensibilidad): indica la proporción de clases correctamente identificadas como positivas sobre el total de clases que efectivamente son positivas.

$$Recall = \frac{TP}{TP + FN} \tag{2.6.1}$$

- Precision o Precisión: indica la proporción de clases que son correctamente identificadas como positivas sobre el total de clases identificadas como positivas.

$$Precision = \frac{TP}{TP + FP} \tag{2.6.2}$$

- F1-Score o Valor-F: combina recall y precision en una métrica.

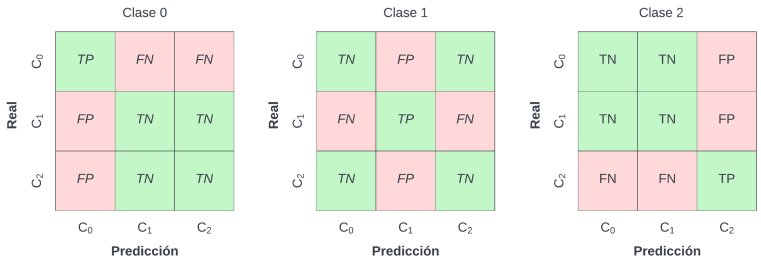
$$F1 - Score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.6.3}$$

- Accuracy o Exactitud: indica el porcentaje de clasificaciones correctas entre la totalidad de datos.

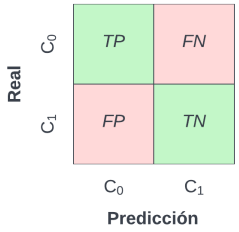
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.6.4}$$

- Matrices de confusión: expone las imágenes con sus clases reales en filas y las predicciones del modelo en columnas. Dependiendo de la clase observada, los valores para TP, TN, FP y FN pueden ser obtenidos según se muestra a continuación.

  Para clasificación multiclase (se ejemplifica para tres clases: C_0, C_1 y C_2):



  Para clasificación binaria:



2.7 Trabajos Previos

En la actualidad existe un sinfín de trabajos en tareas de clasificación, cubriendo distintas patologías y tipos de imágenes médicas. Algunos de ellos son expuestos a continuación.

Un tema relevante ha sido la clasificación relativa al cáncer de mamas. En un trabajo, T. Araújo et al. [35], haciendo uso de una base de datos de acceso público, compuesta por imágenes de histología de alta resolución, desarrollaron un modelo para dos tipos de clasificación: uno binario y otro multiclase. El primer modelo permitía identificar la presencia o no de carcinoma, mientras que el segundo permitía clasificar un tejido entre cuatro clases: normal, lesión benigna, carcinoma invasivo y carcinoma in situ. Lo relevante de este trabajo, es que la arquitectura de la CNN fue diseñada de tal forma que permitiera

la clasificación de imágenes tanto por regiones específicas, en este caso a nivel de célula, como por la totalidad del tejido presente en la imagen. Además de la clasificación propia del modelo de CNN, las características extraídas por el algoritmo fueron utilizadas en la clasificación a través de SVM, donde los resultados obtenidos por ámbos clasificadores son similares: un promedio de 75.68 % en accuracy al usar solo CNN y 75.75 % al usar CNN y SVM.

En el mismo tema, Y. Wang et al. [36] trabajaron en un modelo para el soporte clínico en el diagnóstico de lesiones benignas y malignas utilizando 316 imágenes de ultrasonido. La metodología empleó un modelo de CNN basado en la arquitectura Inception-V3, la cual fue incorporada por TL. Además, existió un proceso de evaluación que consideró un grupo de cinco médicos, de los cuales cuatro aumentaron su desempeño en el diagnóstico luego de tomar en consideración los resultados entregados por el modelo. La métrica utilizada para la evaluación de desempeño en este trabajo, es el Área Bajo la Curva (Area Under the Curve, AUC) de la curva Característica Operativa del Receptor (Receiver Operating Characteristic, ROC). Lo que hace la curva ROC, es relacionar las métricas especificidad (specificity) y sensibilidad (sensitivity o recall). Al calcular el AUC de la curva ROC, se puede discriminar qué tan bueno es el desempeño de una clase sobre la otra. Mientras más cercano a uno sea el valor, mejor es el modelo clasificando esa clase. El valor promedio AUC de la curva ROC para el modelo fue de 0.919.

Siguiendo el enfoque anterior, B. Ehteshami et al. [37] realizaron un estudio comparativo entre modelos de DL y un grupo de 11 especialistas patólogos, en la detección y clasificación de metástasis en ganglios linfáticos en mujeres con cáncer de mama, a partir de 399 imágenes de microscopía. Los modelos estudiados son el resultado del desafío “Cancer Metastases in Lymph Nodes Challenge 2016"(CAMELYON16). El objetivo en clasificación, que fue de naturaleza binaria, era identificar si existía o no metástasis en los ganglios linfáticos. Los patólogos obtuvieron un promedio AUC de 0.810 considerando un límite de tiempo de dos horas. Por otra parte, de los modelos presentados por los equipos, el algoritmo con mejor desempeño, que fue basado en la arquitectura GoogLeNet, obtuvo un valor AUC de 0.994, superando a todos los patólogos. Teniendo esto en consideración, el desafío CAMELYON16 permitió demostrar la superioridad de algunos algoritmos de DL frente a profesionales patólogos en la tarea de clasificación descrita.

T. Tran et al. [38] realizaron una clasificación en células con presencia de leucemia y sanas. Parte de los datos utilizados, específicamente aquellos que representaban células anormales, pertenecían a cuatro tipos de la enfermedad. Además de estos grupos, se contó con imágenes de células normales o sanas, siendo el objetivo del clasificador lograr una distinción binaria. El número inicial de imágenes disponibles en la base de datos era de 108, 59 células normales y 49 anormales. Dado que esto no era

una cantidad elevada de datos y que por lo tanto un buen desempeño del modelo no estaba garantizado, aplicaron técnicas de aumento de datos para sobrellevar este inconveniente. Estas operaciones, que fueron aplicadas solo a un subtipo de leucemia, lograron aumentar las imágenes a 1188. Luego de entrenar el modelo con el 70 % de los datos y evaluar el desempeño con el 30 % restante, se obtuvo un accuracy de 96.6 %, identificando correctamente 345 imágenes de las 357 evaluadas. Este resultado fue comparado con un trabajo previo que solo consideró la base de datos en su estado original y donde el algoritmo de clasificación obtuvo un accuracy de 88.8 %, o sea, un 7.8 % menos. El desarrollo del trabajo permitió concluir que si bien emplear técnicas de transformaciones en imágenes es un proceso sencillo, el efecto en los resultados es notable.

Por último, la clasificación referente a enfermedades y condiciones del cerebro es un tema que ha permitido la realización de varios trabajos. C. Boo-Kyeong et al. [39], a partir de una arquitectura original, utilizaron imágenes de resonancia magnética compuestas por tres clases: Enfermedad de Alzheimer (Alzheimer's Disease, AD), Deterioro Cognitivo Leve (Mild Cognitive Impairment, MCI) y pacientes sanos (o "normal control", NC). La cantidad total de datos, considerando el proceso de aumento, fueron 10.396 imágenes correspondientes a 180 sujetos, 60 por cada clase descrita. Previo al entrenamiento del modelo, las imágenes originales pasaron por una etapa que involucraba pre-procesamiento, registro y segmentación por región de interés (ROI). Dado que el modelo fue construido para predecir resultados en forma binaria, la tarea de clasificación se dividió en tres grupos: el primero contenía imágenes de AD y NC, el segundo de MCI y NC y el tercero de AD y MCI. La evaluación del primer grupo arrojó como resultado un accuracy de 92.3 % considerando 2.176 imágenes, el segundo 85.6 % considerando 2.172 imágenes y el tercero 78.1 % considerando 2.222 imágenes. Esto se tradujo en un promedio de 85.3 %. Además de lo ya mencionado, el modelo fue evaluado considerando las imágenes sin la etapa de segmentación por ROI, reduciendo su desempeño a un 76.9 % en accuracy para el promedio de las tres clases. Por último, se evaluó sin ROI y con las imágenes sin pre-procesar, donde el resultado se redujo aún más, específicamente a un 67.6 % promedio. Esto permitió destacar la relevancia de los pasos de pre-procesamiento en el desempeño del modelo, al menos para el set de datos implementado.

En otro estudio relativo a enfermedades del cerebro, A. Çinar y M. Yildirim [40] utilizaron TL para la creación de un modelo capaz de identificar la presencia o ausencia de un tumor cerebral, también a partir de imágenes de resonancia magnética. Hubo una disponibilidad de 253 imágenes, donde 155 de estas contaban con la presencia de tumor y 98 no. Implementaron una serie de arquitecturas destacadas como base para generar una comparación en los desempeños de evaluación entregados por cada modelo. Aquel que obtuvo un mejor desempeño fue el que incluyó la arquitectura ResNet-50. Obtuvieron un accuracy de 97.01 %, clasificando correctamente 65 imágenes de las 67 incluidas en la

fase de evaluación. Este trabajo permitió, una vez más, demostrar el efecto positivo en el rendimiento de un modelo híbrido, pues el modelo original por sí solo obtuvo un accuracy de 92.54 %.

Los trabajos mencionados son algunos de los más destacados en sus respectivas tareas, sin embargo, la cantidad de otras publicaciones en el área de la medicina es muy elevada. Así lo demuestra Geert Litjens, et al. [18] quienes realizaron un extenso y detallado estudio en el uso de algoritmos de ML y DL en aplicaciones médicas, concentrando el enfoque en los de CNN y en distintos tipos de imágenes como MRI, radiografías y Tomografía Computarizada (Computed Tomography, CT), entre otros. En trabajos que involucran al cerebro se destacan la clasificación y, en algunos casos detección, de las enfermedades Alzheimer, esquizofrenia, deterioro cognitivo leve y lesiones o tumores cerebrales. Respecto a imágenes de retina, ejemplifica tareas relativas a glaucoma, retinopatía diabética y presencia o no de hemorragias. En cuanto a imágenes que involucran el estudio del tórax, sean estas radiografías o CT, destaca tareas en la detección y clasificación de nódulos y tuberculosis. Por otra parte, en estudios con imágenes de células a partir de microscopía u otras técnicas, destaca la clasificación de núcleos, leucocitos, mitocondrias y de cáncer presente en múltiples tejidos, por ejemplo en aquellos pertenecientes al colon. En imágenes para el análisis de abdomen, se destacan estudios sobre detección y clasificación de lesiones en varios órganos, por ejemplo en hígado, riñones, páncreas y próstata. De la totalidad de trabajos que consideró este estudio (308), la mayoría utiliza modelos de CNN, donde algunos son arquitecturas desarrolladas desde cero, otros involucran algoritmos convencionales del ML y otros implementan arquitecturas destacadas del DL a través de la metodología TL.

La Tabla 2.2 expone un resumen de los trabajos analizados en la sección, considerando el tipo de imagen, la cantidad de muestras, el algoritmo de clasificación empleado, si hubo implementación de TL o no y los resultados en las métricas utilizadas para la evaluación.

Tabla 2.2: Sumario de resultados en trabajos previos.

	Tipo de Imagen	Cantidad de Imágenes	Clasificador	TL	Accuracy	AUC
T. Araújo et al. (a) (2017)	WSI	285	CNN	No	75.68 %	-
T. Araújo et al. (b) (2017)	WSI	285	CNN+SVM	No	75.75 %	-
Y. Wang et al. (2020)	US	316	CNN	Inception-V3	-	0.919
B. Ehteshami et al. (2017)	WSI	399	CNN	GoogLeNet	-	0.994
T. Tran et al. (2018)	Microscopía	108	CNN	No	96.6 %	-
C. Boo-Kyeong et al. (2020)	MRI	180	CNN	No	85.3 %	-
A. Çinar y M. Yildirim (2020)	MRI	253	CNN	ResNet-50	97.01 %	-

Donde WSI: Whole Slide Image; US: Ultrasonido; MRI: Magnetic Resonance Image.

2.8 Discusión

La armonía entre los componentes de una red neuronal convolucional, además de la naturaleza adaptativa que poseen, demuestran el potencial del algoritmo al trabajar sobre imágenes de distinta complejidad. Además, la variedad de modelos destacados estudiados y las ventajas que cada arquitectura presenta, otorgan alternativas que ayudan a combatir la variabilidad de problemas del mundo real.

Por otra parte, la literatura demuestra resultados de gran prestigio en trabajos que emplean imágenes médicas. Incluso, el nivel alcanzado en varios de ellos es comparable al de profesionales de la salud. En muchos de estos trabajos, se ha demostrado la relevancia de las técnicas de regularización en el rendimiento de los modelos, como el aumento de datos y la transferencia de aprendizaje.

Si bien las técnicas de regularización permiten la mejora en el desempeño general del modelo, cuando se trata del aumento de datos con imágenes médicas, las transformaciones aplicadas deben ser sutiles y de un impacto reducido, ya que las imágenes resultantes deben preservar las características naturales de las patologías.

El rendimiento de los modelos presentados, junto con lo relacionado a hiperparámetros, técnicas de regularización y funciones auxiliares, será estudiado en detalle en las siguientes secciones con el objeto de hallar lo óptimo para su incorporación en los tutoriales.

3 Capítulo 3. Bases de Datos y Patologías Asociadas

3.1 Introducción

Luego de un proceso de revisión de bases de datos públicas de imágenes médicas disponibles en la web, fueron tres las seleccionadas. La primera, posee imágenes de resonancia magnética del cerebro con la presencia de tumores cerebrales y permite una clasificación multiclase. La segunda, incluye imágenes de radiografía de tórax referentes a Covid-19, y también permite una clasificación multiclase. La tercera, contiene imágenes de fondo de retina con retinas sanas y enfermas, y permite una clasificación binaria. En este capítulo, se hace una breve introducción a las patologías asociadas y se detallan las imágenes disponibles en cada base de datos.

3.2 Tumores Cerebrales

Un tumor cerebral se origina debido a la presencia anormal de células cancerígenas y se puede clasificar en primario o secundario. En el primario, dichas células se originan en el cerebro o estructuras adyacentes, mientras que en el secundario, estas viajan desde otros órganos hacia él [41]. Meningioma y glioma corresponden a los tipos de tumores cerebrales más comunes. El primero se origina en las meninges, es decir, las capas que protegen al cerebro y a la médula espinal, mientras que el segundo lo hace en las células gliales [42]-[43]. Por otra parte, el tumor pituitario es el principal mal que afecta a la glándula pituitaria [44].

Los síntomas varían según el tipo de tumor, sin embargo, es común que la persona presente malestares neurológicos usuales como dolor de cabeza, agotamiento o fatiga constante. De forma más precisa, en cuanto a los síntomas del tumor pituitario, estos involucran el mal funcionamiento del sistema endocrino [41]-[44].

Los estudios en cuanto a los motivos de aparición, son aún poco concluyentes, pero existe cierta evidencia que indica que la genética, o la frecuente exposición a radición ionizante, podrían ser algunos de ellos. [41].

Una forma de diagnóstico es a través de la imagen de resonancia magnética, la cual es una técnica no invasiva que permite obtener imágenes transversales del cuerpo humano. La imagen es generada

a través de un equipo llamado resonador, en el cual el paciente es introducido. El resonador produce campos magnéticos que, con la ayuda de radiofrecuencia, genera movimientos en los átomos presentes en el cuerpo, y la imagen es construida a partir de la energía recibida y luego liberada. La ventaja de esta técnica, es que utiliza radiación no ionizante, por lo que su uso es seguro para el paciente y provechoso para la investigación relativa a la medicina, ya que las imágenes producidas son de alta calidad [45]-[46].

La base de datos seleccionada está disponible en [47]. Consiste en 3.064 imágenes de resonancia magnética de tres tipos de tumores cerebrales: meningioma, glioma y pituitario. Posee 708 imágenes para meningioma, 1.426 para glioma y 930 para pituitario, y corresponden a 233 pacientes. Las imágenes están en formato JPEG de tres canales (RGB), con una dimensión de 512x512 píxeles. Además, se dispone de una lista de clases o etiquetas para cada una de ellas donde clase 0, clase 1 y clase 2 corresponden a meningioma, glioma y pituitario, respectivamente. Por último, la base de datos cuenta con una lista con los IDs de cada paciente. La Fig. 3.1 expone imágenes de distintos sujetos por clase y plano anatómico.

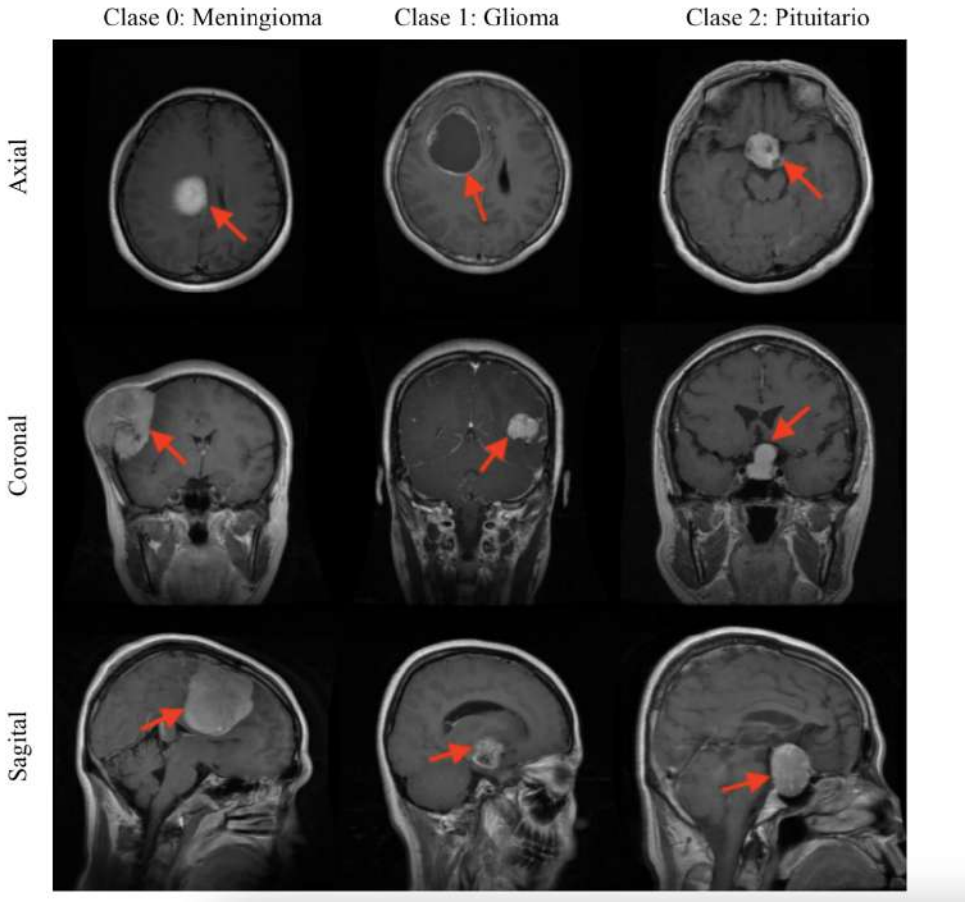


Fig. 3.1: Algunas imágenes disponibles en la base de datos Tumores Cerebrales. Se exponen los tres tipos de tumores en distintos planos anatómicos [47].

3.3 Covid-19

El 31 de Diciembre del año 2019, la Organización Mundial de la Salud reportó el primer paciente infectado con coronavirus en Wuhan, China. La enfermedad adquirió la característica de Pandemia el 11 de Marzo del 2020. Sin embargo, los coronavirus han sido estudiados desde hace ya medio siglo, por Tyrell y Bynoe en primera instancia.

El diagnóstico de la enfermedad involucra una serie de síntomas, siendo uno de los principales y más severos la neumonía, que afecta directamente al parénquima pulmonar. Además, otros síntomas, como complicaciones de naturaleza gastrointestinal, dolores de cabeza, fiebre, congestión nasal y dificultad para respirar son recurrentes en pacientes infectados. Asimismo, existen pacientes asintomáticos [48]. Los pacientes que suelen presentar mayores complicaciones de salud a raíz de la infección, son aquellos que cuentan con su sistema inmunológico comprometido, que presentan comorbilidades, o que pertenecen a la tercera edad [49].

Si bien la mayoría de los síntomas se presentan en una etapa prematura, la neumonía suele manifestarse luego de aproximadamente 10 días, lo cual dificulta la detección temprana a través de técnicas de imagenología, como radiografía de tórax o CT, siendo la técnica más precisa e inmediata la reacción en cadena de la polimerasa (PCR). De hecho, un estudio que consideró a 64 pacientes demostró que la radiografía de tórax presentó una sensibilidad de 69 % al momento de identificar la presencia del virus, no en una etapa necesariamente inicial. Esto, debido a que las características propias de la enfermedad, como las opacidades presentes en los pulmones, especialmente en la periferia y las zonas más bajas, se manifiestan en una etapa avanzada. Lo anterior es desalentador, ya que la radiografía es un método con alto alcance en la población, sencillo de obtener y de bajo costo, lo que la convierte en la herramienta principal para diagnosticar patologías con síntomas respiratorios. A pesar de que los hallazgos de esta técnica son deficientes en etapas iniciales de la enfermedad, y de no ser recomendada para pacientes asintomáticos o con síntomas leves debido a su baja sensibilidad, sí resulta de gran utilidad en etapas avanzadas. Cuando un paciente requiere de un monitoreo constante durante la evolución de la enfermedad, y se encuentra en riesgo de que su situación empeore, la radiografía de tórax es un método provechoso y apropiado [50]-[51]-[52].

La base de datos para Covid-19 cuenta con 15.153 imágenes de Radiografía de Tórax, en formato PNG, en escala de grises y con dimensiones 299x299 píxeles. Estas, están dividas en las clases normal, neumonía viral y covid, con 10.192, 1.345 y 3.616 imágenes para cada una, respectivamente. La Fig. 3.2 expone tres imágenes por cada clase. La base de datos está disponible en el siguiente enlace: COVID-19 Radiography Database. [53]-[54].

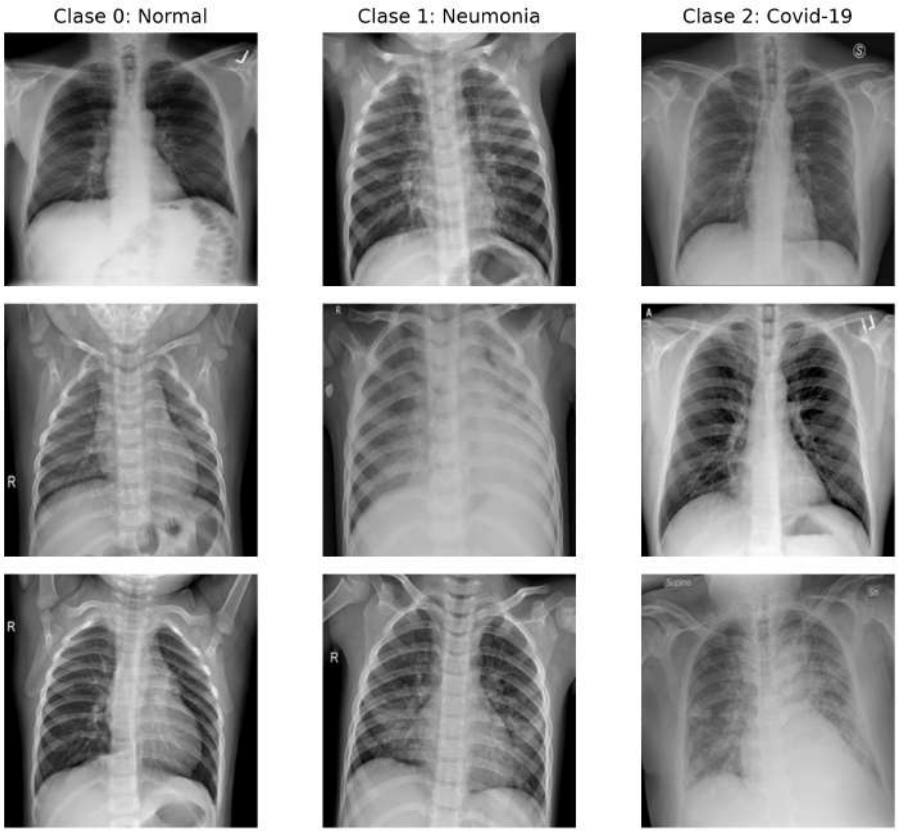


Fig. 3.2: Algunas imágenes disponibles en la base de datos Covid-19. Se exponen las clases normal, neumonía y covid en columnas [53]-[54].

3.4 Enfermedades de la Retina

La retina, que corresponde a la superficie interna del ojo, es ideal para la observación no invasiva de los vasos sanguíneos, permitiendo el estudio del Sistema Nervioso Central. Esta estructura, de forma esférica y que mide alrededor de 22 milímetros en un humano adulto, está compuesta por neuronas y células gliales. Dado que el sistema circulatorio de la retina presenta un flujo sanguíneo eficiente, es posible identificar anormalidades cuando este no es el caso. A veces, estas anormalidades reflejan la existencia de patologías crónicas que en el momento de la evaluación no presentan otros síntomas. Esto se explica debido a que los vasos sanguíneos de la retina son parte del sistema circulatorio del cerebro y contemplan similitudes en el comportamiento al presentarse alteraciones [55].

Diversas técnicas de imagen son utilizadas al momento del diagnóstico. La técnica de Imagen de Fondo de Retina permite generar una imagen en dos dimensiones de la superficie interior del ojo, a partir de la cual se pueden detectar cambios minúsculos en la retina, que en un examen de rutina realizado por un oftalmólogo podrían pasar desapercibidos. Respecto a otras técnicas utilizadas, la Tomografía

de Coherencia Óptica permite la obtención de imágenes de alta resolución de las distintas capas de la retina, mientras que la Autoflorescencia del fondo de retina usa la capacidad de fluorescencia natural que tiene esta estructura para identificar anormalidades. Por último, la Angiografía con fluoresceína es una técnica invasiva en la cual, a través de la inyección de un tinte y posterior estimulación por una cámara especial para el examen, se obtienen imágenes de alta calidad de los vasos sanguíneos y de cualquier anormalidad [55]-[56].

Respecto a las patologías, la retinopatía diabética es muy común en personas que padecen Diabetes, ya sea Tipo 1 o 2. Es más, es la principal causa de pérdida de visión en pacientes adultos. Puede ser clasificada en retinopatía diabética proliferativa y no-proliferativa, diferenciadas principalmente por la proliferación moderada o severa de vasos sanguíneos en la primera. La enfermedad ocurre producto de una alta y prolongada exposición a hiperglucemia, lo que produce un daño a los capilares de la retina, generando complicaciones como oclusión en las venas (retinal vein occlusion), en las arterias (retinal artery occlusion) y posible isquemia macular (ischemic maculopathy). Al no ser tratada, y debido a la gran presión intraocular, se puede presentar pérdida de visión o desprendimiento de la retina (retinal detatchment). Otras patologías que interfieren en la visión a través de pequeñas manchas luminosas o granulares, son las fotopsias, síndrome de puntos blancos y nieve visual [56]-[57].

La base de datos se encuentra disponible en [58]. Cuenta con 3.200 imágenes de fondo de retina, que corresponden a 669 retinas sanas y a 2.531 retinas enfermas que, a su vez, cubren 46 patologías. Las dimensiones son de 300x300 píxeles, el formato de archivo es PNG y cuentan con canales RGB. Además, se dispone de archivos en formato CSV con la información referente a cada conjunto de datos y a cada clase. La Fig. 3.3 muestra algunas imágenes que conforman la base de datos.

3.5 Discusión

Las bases de datos seleccionadas presentan características compatibles con el uso de redes neuronales convolucionales para la clasificación, permitiendo abarcar dos tipos de esta: binaria y multiclase. Por otra parte, la cantidad de imágenes disponibles es adecuada para conformar conjuntos de entrenamiento, validación y prueba lo suficientemente representativos para cada una de las clases. Asimismo, las características de las imágenes posibilitan la aplicación de las técnicas transferencia de aprendizaje y aumento de datos y, por lo tanto, estudiar el efecto de estas en el rendimiento de la clasificación.

Cada una de las bases de datos considerada en este estudio se encuentra disponible para su implementación en los tutoriales.

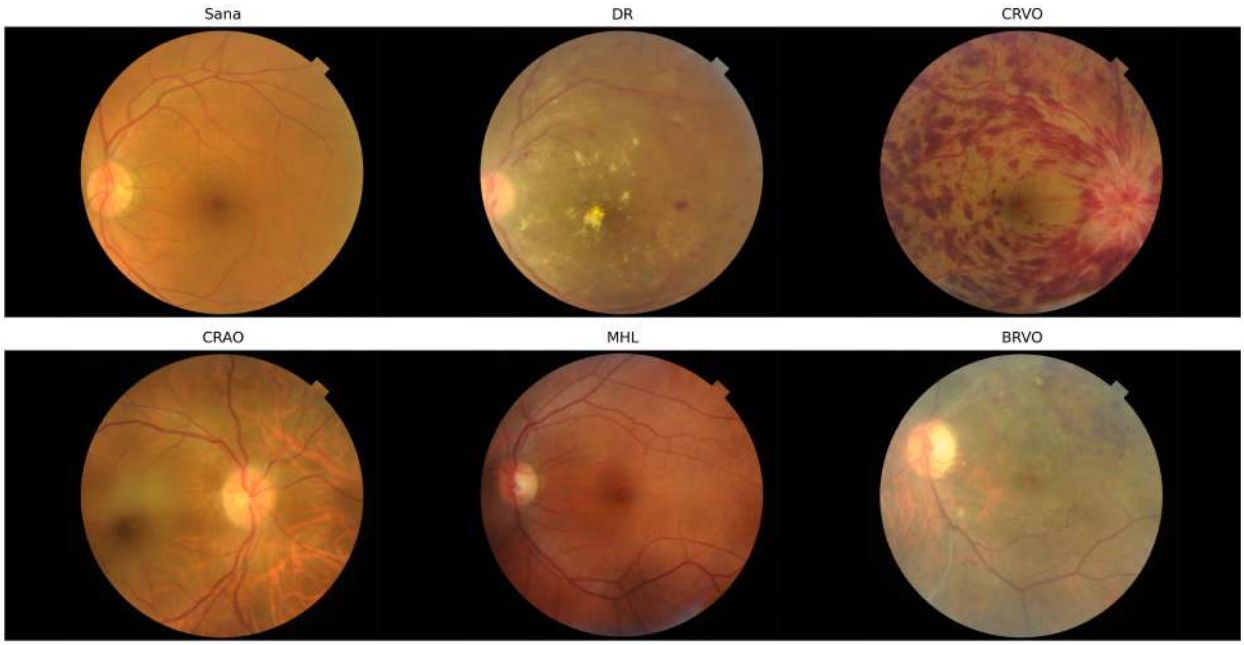


Fig. 3.3: Algunas imágenes disponibles en la base de datos Enfermedades de la Retina. Se expone una retina sana perteneciente a la clase 0 y cinco patologías distintas, pertenecientes a la clase 1. DR: retinopatía diabética; CRVO: oclusión de vena retinal central; CRAO: oclusión de arteria retinal central; MHL: agujero macular; BRVO: oclusión de ramificación de vena retinal [58].

Otras bases de datos revisadas incluyen imágenes de lesiones benignas y malignas de la piel, de histología para la identificación de linfoma, de microscopía para la clasificación de distintos tipos de células, entre otras. Sin embargo, no fueron consideradas debido a que algunas de ellas están en formatos que implican mucho costo computacional y, por lo tanto, son incompatibles con la plataforma de desarrollo, o porque otras presentan escasa información. De todas formas, las bases de datos que sí fueron consideradas son suficientes para el estudio.

4 Capítulo 4. Implementación y Optimización de Modelos

4.1 Introducción

En este capítulo se describe el procedimiento realizado para llevar a cabo la clasificación de imágenes médicas y la generación de tutoriales (Fig. 4.1). En primer lugar, para cada base de datos, se expone cómo se construyeron los conjuntos de datos y cuántas imágenes por clase forman parte de cada uno. Posteriormente, se detalla el preprocesamiento aplicado a las imágenes. Luego, se describe cómo fueron configurados los modelos VGG-16, ResNet-50 e Inception-V3, junto a algunos hiperparámetros que se mantuvieron fijos para todas las pruebas. Estos hiperparámetros son el optimizador y las funciones de pérdida. Además, se detalla la configuración establecida para las funciones auxiliares, es decir, los callbacks. Finalmente, se describe una serie de experimentos realizados que permitieron obtener una evaluación de desempeño de los modelos en la clasificación. Estos experimentos fueron divididos en cuatro etapas que comprenden el efecto que distintas configuraciones tienen sobre el desempeño de los modelos, además de la aplicación de técnicas para mejorarlo. Adicionalmente, se presenta una última etapa que cubre la generación de tutoriales. La plataforma de desarrollo utilizada fue Google Colab y las principales bibliotecas fueron Tensorflow [59] y Keras [27], para la implementación de los modelos de redes neuronales y funciones auxiliares, al igual que para la lectura, carga y preprocesamiento de las imágenes.

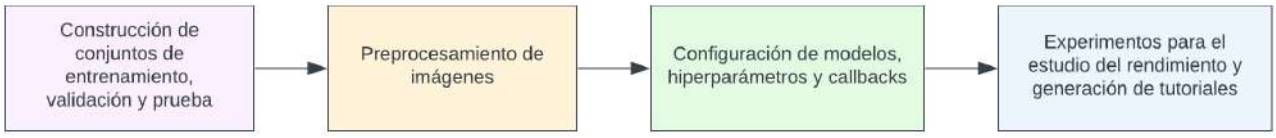


Fig. 4.1: Procedimiento propuesto para la implementación de la clasificación y generación de tutoriales.

4.2 Procedimiento para la Implementación de la Clasificación

4.2.1 Construcción de Conjuntos de Entrenamiento, Validación y Prueba.

Para la base de datos Tumores Cerebrales, de un total de 3.064 imágenes, se consideró aproximadamente el 70 % para el conjunto de entrenamiento, el 15 % para el conjunto de validación y el 15 %

para el conjunto de prueba. La clasificación implementada fue de tipo multiclase, y permitió identicar a qué tipo de tumor pertenecen las imágenes del conjunto de prueba.

Por otra parte, para la base de datos Covid-19, de un total de 15.153 imágenes, se consideró aproximadamente el 60 % para el conjunto de entrenamiento, el 20 % para el conjunto de validación y el 20 % para el conjunto de prueba. Para este caso, la clasificación también fue de tipo multiclase y permitió catalogar las imágenes del conjunto de prueba en las clases normal, neumonía y covid.

Por último, para la base de datos Enfermedades de la Retina, del total de 4.480 imágenes disponibles, se consideró aproximadamente el 70 % para el conjunto de entrenamiento, el 15 % para el conjunto de validación y el 15 % para el conjunto de prueba. En este caso, la clasificación implementada fue de tipo binaria y permitió identificar si la retina estudiada presenta alguna enfermedad (clase 1, "sí") o no (clase 0, "no").

Para la construcción de los conjuntos, debido que se contaba con una cantidad desbalanceada de imágenes por clase, se realizó una división estratificada de estas para mantener la proporción de imágenes pertenecientes a cada clase en los distintos conjuntos. De esta forma, se pudo garantizar una buena representatividad para todas las clases. Además, para la base de datos Tumores Cerebrales, se tuvo en consideración la información del ID del paciente, para que imágenes de una misma persona no estuvieran en distintos conjuntos. El detalle de la cantidad de imágenes por cada clase en cada conjunto, se muestra en la Tabla 4.1, para las bases de datos Tumores Cerebrales, Covid-19 y Enfermedades de la Retina.

4.2.2 Preprocesamiento de Imágenes

Las imágenes originales presentes en las bases de datos Tumores Cerebrales, Covid-19 y Enfermedades de la Retina, tienen dimensiones de 512x512, 299x299 y 300x300 píxeles, respectivamente. Cada una posee tres canales (RGB), y los valores de los píxeles van de 0 a 255.

Dependiendo del modelo implementado, las imágenes pasaron por un preprocesamiento específico. Funciones de la biblioteca Keras se encargaron de este proceso, aplicando las mismas operaciones que fueron empleadas en el entrenamiento original de cada modelo en la base de datos ImageNet.

Para los modelos VGG-16 y ResNet-50, se modificó la dimensión a 224x224, los canales fueron invertidos de RGB a BGR y los valores de los píxeles fueron centrados en cero. Para Inception-V3, se obtuvo una imagen de dimensión 299x299 y los valores de los píxeles fueron escalados entre -1 y 1, sin realizar modificaciones en el orden de los canales.

Tabla 4.1: Detalle de imágenes por clase y por conjunto en cada base de datos.

Tumores Cerebrales				
Conjunto	Clase 0: Meningioma	Clase 1: Glioma	Clase 2: Pituitario	Total
Entrenamiento	527	944	650	2.121
Validación	64	227	193	484
Prueba	117	255	87	459
Covid-19				
Conjunto	Clase 0: Normal	Clase 1: Neumonía	Clase 2: Covid	Total
Entrenamiento	6.150	822	2.119	9.091
Validación	2.020	251	760	3.031
Prueba	2.022	272	737	3.031
Enfermedades de la Retina				
Conjunto	Clase 0: No	Clase 1: Sí		Total
Entrenamiento	401	1.519		3.200
Validación	134	506		640
Prueba	134	506		640

4.2.3 Configuración de Modelos, Hiperparámetros y Callbacks.

Los modelos VGG-16, ResNet-50 e Inception-V3, fueron implementados con los pesos pre-entrenados en la base de datos ImageNet. Para ello, se congeló la actualización de pesos en las capas encargadas de la extracción de características, y se modificó la sección de la red encargada de la clasificación. Esto último, para adaptar el modelo a las tareas de clasificación específicas a resolver. La modificación realizada se expone en la Fig. 4.2.

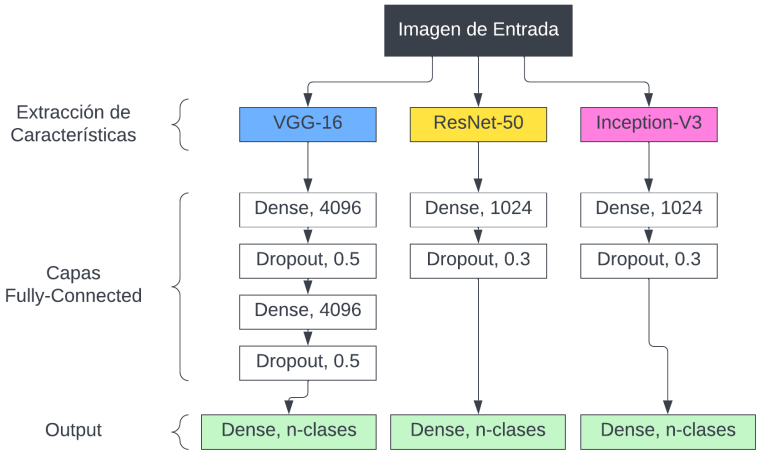


Fig. 4.2: Modificación implementada en capas finales de cada modelo para la tarea de clasificación específica.

En cuanto a los hiperparámetros utilizados en el entrenamiento del modelo, el optimizador fue Adam, las funciones de pérdida fueron binary cross-entropy para clasificación binaria y categorical cross-entropy para clasificación multiclase, y la cantidad máxima de epochs fue 100.

Respecto a la configuración de los callbacks, para Early-Stopping se consideró como métrica de monitoreo la pérdida del conjunto de validación (validation loss). El parámetro de paciencia, o sea, cuánto esperar por una mejora en el entrenamiento antes de detenerse, fue ajustado en 10. La mejora mínima que debe ocurrir dentro de este intervalo de paciencia fue ajustado en 0.0001. Por último, se activó la opción de restaurar los mejores pesos en caso de que el entrenamiento del modelo hubiese sido perjudicado dentro del periodo de paciencia. Por otra parte, ReduceLROnPlateau, fue también configurada con la pérdida de validación como métrica de monitoreo. Sin embargo, esta vez el parámetro paciencia fue configurado en 5, y la mejora mínima requerida fue ajustada en 0.001. El factor de disminución del learning-rate considerado fue de 0.1, y el valor mínimo posible fue de 1e-7. Por último, la función ModelCheckpoint fue configurada para almacenar el avance del entrenamiento del modelo cada vez que existiese una mejora en la actualización de pesos.

4.2.4 Experimentos para el Estudio del Efecto de Distintas Configuraciones sobre el Rendimiento de los Modelos y Generación de Tutoriales.

Los experimentos fueron divididos en etapas, que son descritas a continuación y resumidas en la Fig. 4.3. La evaluación de rendimiento fue realizada en base a las métricas recall, precision, f1-score, accuracy y matrices de confusión, detalladas en la sección 2.6.

- Etapa 1: efecto del batch-size. Para cada base de datos se probaron los modelos VGG-16, ResNet-50 e Inception-V3 para los valores de batch-size 32, 64, 128 y 256, manteniendo el learning-rate fijo en el valor por defecto, 0.001. Este proceso produjo 36 resultados, de los cuales se seleccionó el modelo y el batch-size que presentó mejor desempeño en cada base de datos.

- Etapa 2: efecto del learning-rate. Los modelos seleccionados en la etapa anterior fueron evaluados con valores de learning-rate inicial de 1e-1, 1e-2, 1e-3, 1e-4 y 1e-5, esta vez manteniendo fijo el valor de batch-size seleccionado en la etapa anterior. Este proceso produjo 15 resultados, de los cuales se selecciónó el valor de learning-rate con mejor rendimiento.

- Etapa 3: efecto del aumento de datos. Se aplicó la técnica aumento de datos al modelo con mejor desempeño en cada base de datos. Esta vez, se mantuvo fijos los valores de batch-size y learning-rate encontrados en los experimentos anteriores. Las operaciones realizadas, y que

fueron aplicadas solo a las imágenes del conjunto de entrenamiento, se detallan en Anexo A, Tabla A.1.

- Etapa 4: efecto del fine-tuning. Se alteró la configuración de las capas de los modelos para hacer algunas entrenables y otras no. La primera mitad mantuvo inhibido el entrenamiento, mientras que la segunda mitad lo mantuvo activado. El modelo, con su nueva configuración, fue compilado con un learning-rate igual a 1e-5 y entrenado por 5 épocas.
- Etapa 5: generación de tutoriales. Luego de estudiar la influencia que diversas configuraciones tienen sobre los modelos al implementar la clasificación, se generaron tutoriales teniendo en consideración la información obtenida. Para cada base de datos se desarrolló un tutorial en formato Notebook, que cuenta con las bases de datos de imágenes médicas utilizadas, y con todos los archivos necesarios para su ejecución.

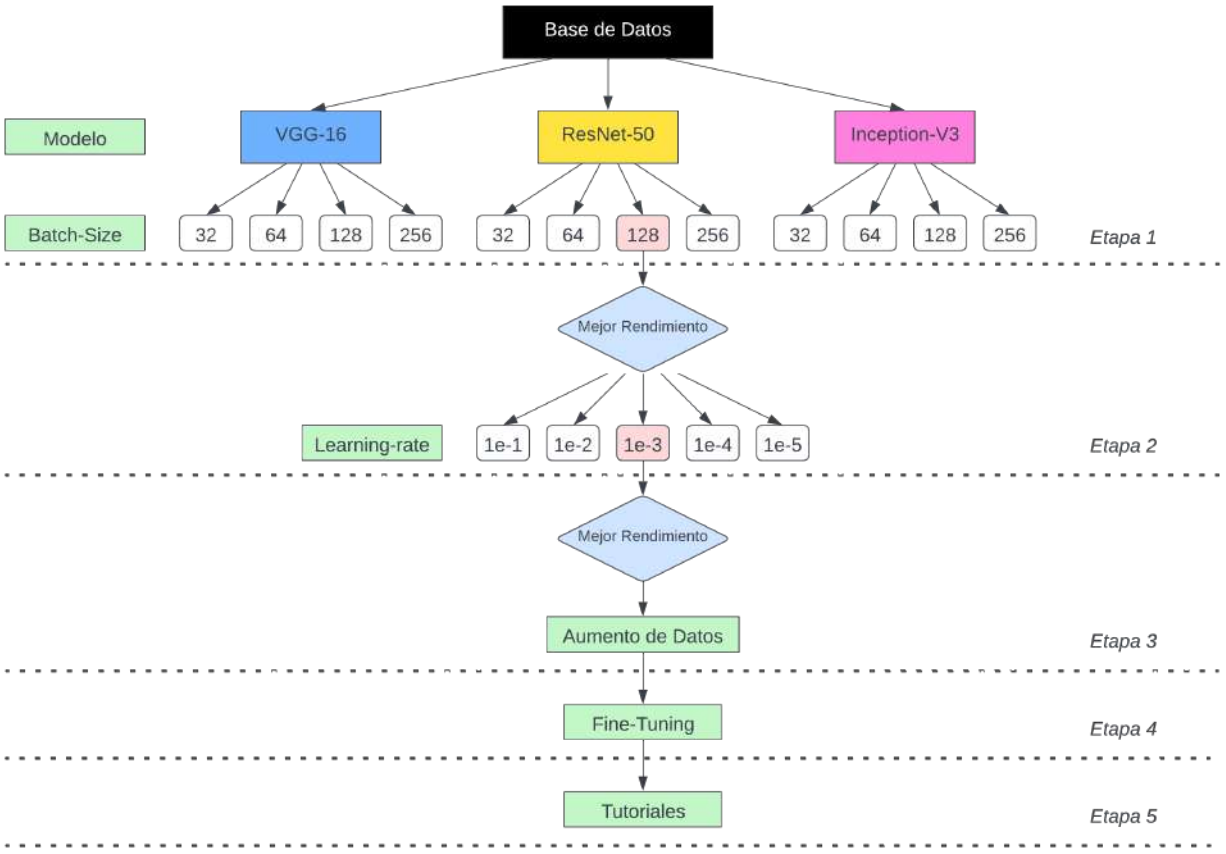


Fig. 4.3: Procedimiento implementado para la evaluación de rendimiento de los modelos en cada base de datos. Etapa 1: efecto del batch-size. Etapa 2: efecto del learning-rate. Etapa 3: efecto del aumento de datos. Etapa 4: efecto del fine-tuning. Etapa 5: generación de tutoriales.

4.3 Discusión

El desbalance de clases presentado en las bases de datos requirió de una división estratificada para garantizar que cada conjunto fuese lo más proporcional y representativo posible.

Con la configuración establecida para cada modelo, se obtuvo una cantidad total de 134.272.835 parámetros para VGG-16, de los cuales 119.558.147 son entrenables. De forma similar, para ResNet-50 se obtuvo 126.352.259 parámetros y 102.764.547 entrenables. Por último, para Inception-V3 se obtuvo 156.024.611 parámetros en total, de los cuales 134.221.827 son entrenables. De esta forma se pudo concluir que el modelo que presenta mayor complejidad en la actualización de parámetros, considerando la cantidad de estos que son entrenables, es Inception-V3 y el que presenta menos complejidad es ResNet-50.

Por otra parte, el estudio propuesto permite un completo análisis sobre el efecto que los modelos y sus respectivas configuraciones presentan en el rendimiento de la clasificación. La totalidad del procedimiento implementado logra cubrir lo revisado en los capítulos anteriores, es decir, la generación de distintos conjuntos de imágenes, el ajuste de hiperparámetros sustanciales, la implementación de funciones auxiliares, y las técnicas para aumentar el nivel de la clasificación.

Por último, las métricas de evaluación utilizadas permiten obtener resultados desde distintas perspectivas. Por ejemplo, un resultado general, es la cantidad de imágenes clasificadas correctamente e incorrectamente, mientras que resultados más minuciosos permiten observar, dentro de una misma base de datos, qué clases desempeñan mejor que otras y cuáles generan más confusión. De esta manera, mediante las conclusiones obtenidas según distintos criterios, se puede tomar una decisión fundamentada al momento de seleccionar los mejores modelos y configuraciones, e integrarlos en el tutorial respectivo a cada base de datos.

5 Capítulo 5. Resultados

5.1 Introducción

En este capítulo se presentan, en etapas, los resultados obtenidos para cada uno de los experimentos realizados. Se incluyen gráficos de barra, tablas, matrices de confusión y curvas de aprendizaje cuando el resultado presentado lo requiera. Además, se realiza una comparación entre el rendimiento que poseen los modelos al clasificar los datos con y sin la aplicación de técnicas para su mejora. Por último, se definen los mejores modelos y configuraciones para cada problema y se detalla su incorporación en los tutoriales

5.2 Efecto del Batch-Size sobre el Rendimiento

Para la base de datos Tumores Cerebrales, la Fig. 5.1 muestra gráficamente el desempeño de los modelos en cada métrica para cada batch-size. En a) se muestra la métrica recall, en b) la métrica precisión, en c) la métrica f1-score y en d) la métrica accuracy. El mejor desempeño se logró con ResNet-50 y un tamaño de batch igual a 256. Esto, considerando todas las métricas utilizadas para la evaluación. De un total de 459 imágenes disponibles en el conjunto de prueba, el modelo logró clasificar de forma correcta 426. El mismo modelo, pero esta vez con el resto de valores para batch-size, logró obtener resultados similares en cuanto a métricas e imágenes correctamente identificadas y, además, superar a todos los otros modelos evaluados. Se puede apreciar que el modelo VGG-16, con un tamaño de batch igual a 128, es el que más se aproxima al mejor resultado, clasificando correctamente 418 imágenes. Por último, Inception-V3 logró su mejor rendimiento con un batch-size de 32 y 414 imágenes clasificadas correctamente. Los mejores resultados son resumidos en la Tabla 5.1. El detalle para todos los resultados se presenta en Anexo B, Tabla B.1.

Tabla 5.1: Resultados para el efecto del batch-size sobre el rendimiento de los modelos en la base de datos Tumores Cerebrales. Las métricas presentadas corresponden al promedio entre la totalidad de clases.

Modelo	Batch-Size	Recall	Precision	F1-Score	Accuracy	Correcto	Incorrecto	Total
VGG-16	128	0.922	0.893	0.906	0.911	418	41	459
ResNet-50	256	0.931	0.904	0.915	0.928	426	33	459
Inception-V3	32	0.897	0.889	0.892	0.902	414	45	459

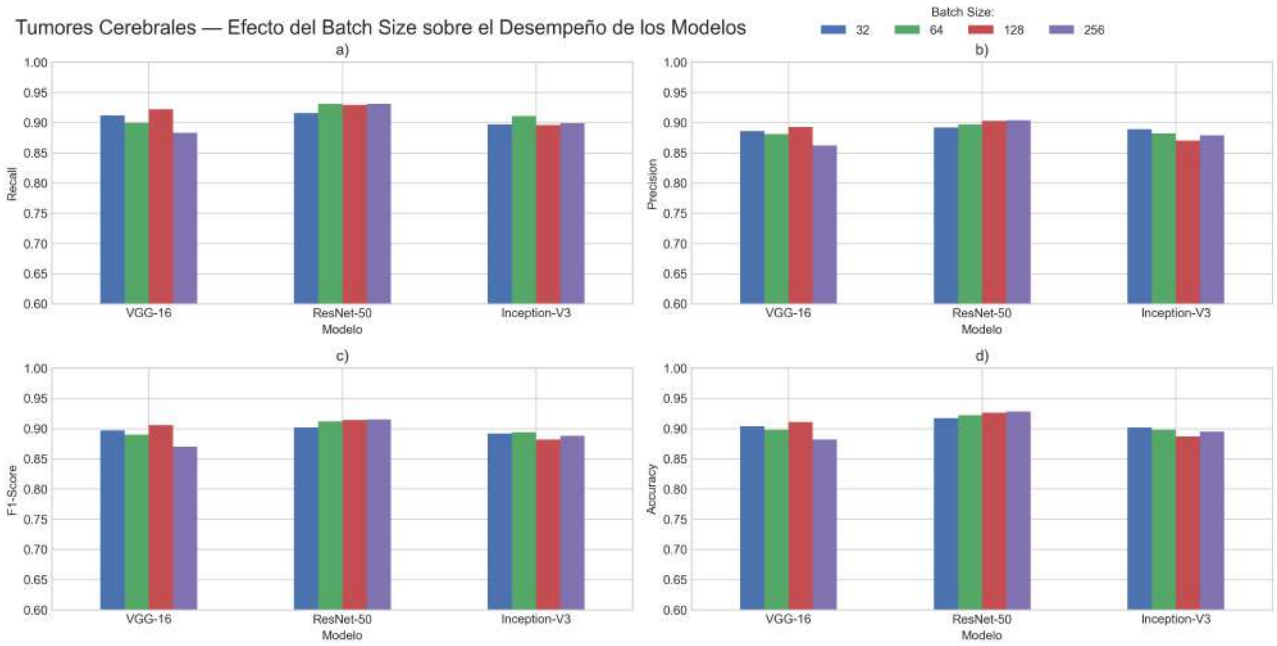


Fig. 5.1: Efecto del batch-size sobre el rendimiento de los modelos VGG-16, ResNet-50 e Inception-V3 para la base de datos Tumores Cerebrales. a) Recall; b) Precision; c) F1-Score; d) Accuracy.

Para la base de datos Covid-19, los modelos fueron evaluados en 3.031 imágenes. La Fig. 5.2 expone gráficos de barra con los resultados del rendimiento. VGG-16, con un tamaño de batch igual a 64, fue quien obtuvo la mayor cantidad de imágenes clasificadas correctamente. Logró identificar 2.982 imágenes de forma correcta, y 49 de forma incorrecta. Obtuvo los mejores valores para las métricas recall, precision y accuracy, pero fue superado en f1-score por ResNet-50 con un tamaño de batch de 128. Sin embargo, este último clasificó 2.975 imágenes de forma correcta, por lo que VGG-16 es, de todas formas, el modelo con mejor rendimiento. Inception-V3, por otra parte, fue el modelo con peor desempeño, y la máxima cantidad de imágenes que logró clasificar correctamente fue de 2.946, con un tamaño de batch igual a 256. Los mejores resultados se resumen en la Tabla 5.2. El detalle para todos los resultados se presentan en Anexo B, Tabla B.2.

Tabla 5.2: Resultados para el efecto del batch-size sobre el rendimiento de los modelos en la base de datos Covid-19. Las métricas presentadas corresponden al promedio entre la totalidad de clases.

Modelo	Batch-Size	Recall	Precision	F1-Score	Accuracy	Correcto	Incorrecto	Total
VGG-16	64	0.980	0.980	0.970	0.984	2.982	49	3.031
ResNet-50	128	0.976	0.974	0.975	0.982	2.975	56	3.031
Inception-V3	256	0.962	0.967	0.965	0.972	2.946	85	3.031

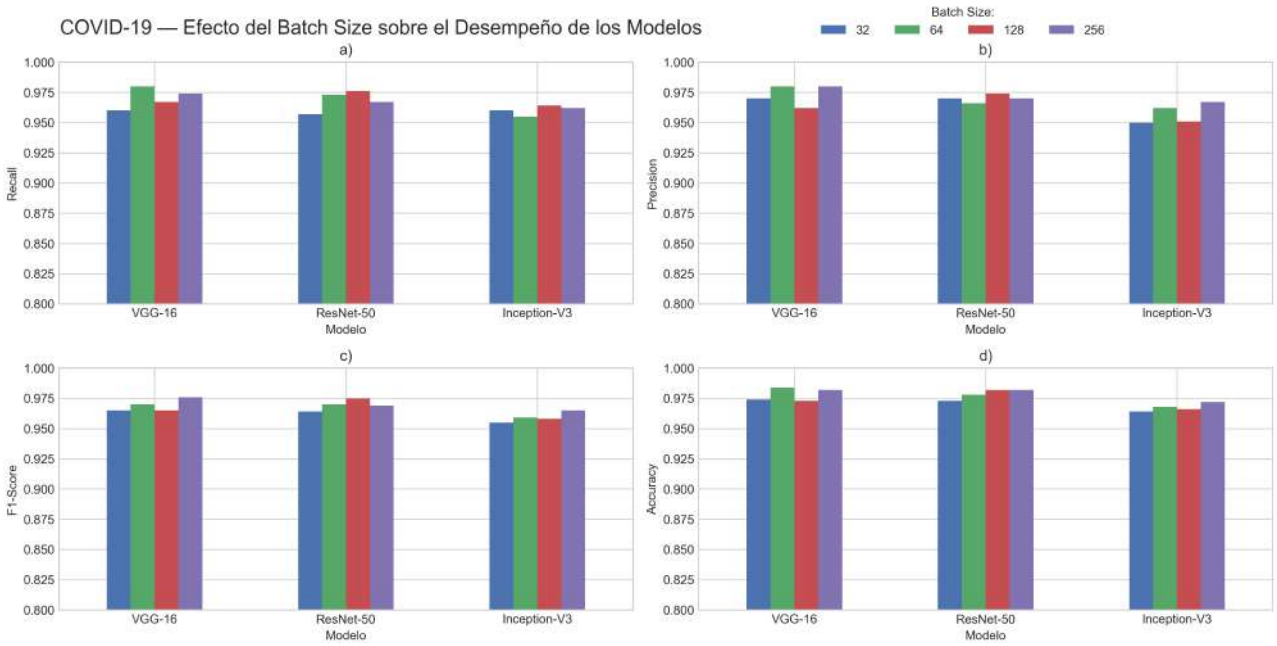


Fig. 5.2: Efecto del batch-size sobre el rendimiento de los modelos VGG-16, ResNet-50 e Inception-V3 para la base de datos Covid-19. a) Recall; b) Precision; c) F1-Score; d) Accuracy.

Para la base de datos Enfermedades de la Retina, los modelos fueron evaluados en 640 imágenes. Los resultados se exponen en forma gráfica en la Fig. 5.3. Nuevamente fue VGG-16 quien logró la mayor cantidad de aciertos en la clasificación, con 593 imágenes clasificadas correctamente. Estos resultados fueron obtenidos con un tamaño de batch igual a 256. A este resultado le sigue, en segundo lugar, el modelo Inception-V3 con un bach-size igual a 128, que logró clasificar de forma correcta 586 imágenes y de forma incorrecta 54. ResNet-50 fue, esta vez, el modelo con peor desempeño, siendo su mejor resultado el obtenido con un tamaño de batch igual a 64, con 582 imágenes correctamente clasificadas. Los mejores resultados se resumen en la Tabla 5.3. El detalle para todos los resultados se presentan en Anexo B, Tabla B.3.

Tabla 5.3: Resultados para el efecto del batch-size sobre el rendimiento de los modelos en la base de datos Enfermedades de la Retina. Las métricas presentadas corresponden al promedio entre la totalidad de clases.

Modelo	Batch-Size	Recall	Precision	F1-Score	Accuracy	Correcto	Incorrecto	Total
VGG-16	256	0.891	0.888	0.890	0.927	593	33	640
ResNet-50	64	0.877	0.858	0.867	0.909	582	49	640
Inception-V3	128	0.866	0.876	0.871	0.916	586	47	640

Del análisis anterior, se concluyó que para la base de datos Tumores Cerebrales, ResNet-50 con un tamaño de batch igual a 256 logró los mejores resultados. Para la base de datos Covid-19, fue VGG-16 con un tamaño de batch igual a 64. Por último, para la base de datos Enfermedades de la Retina, fue

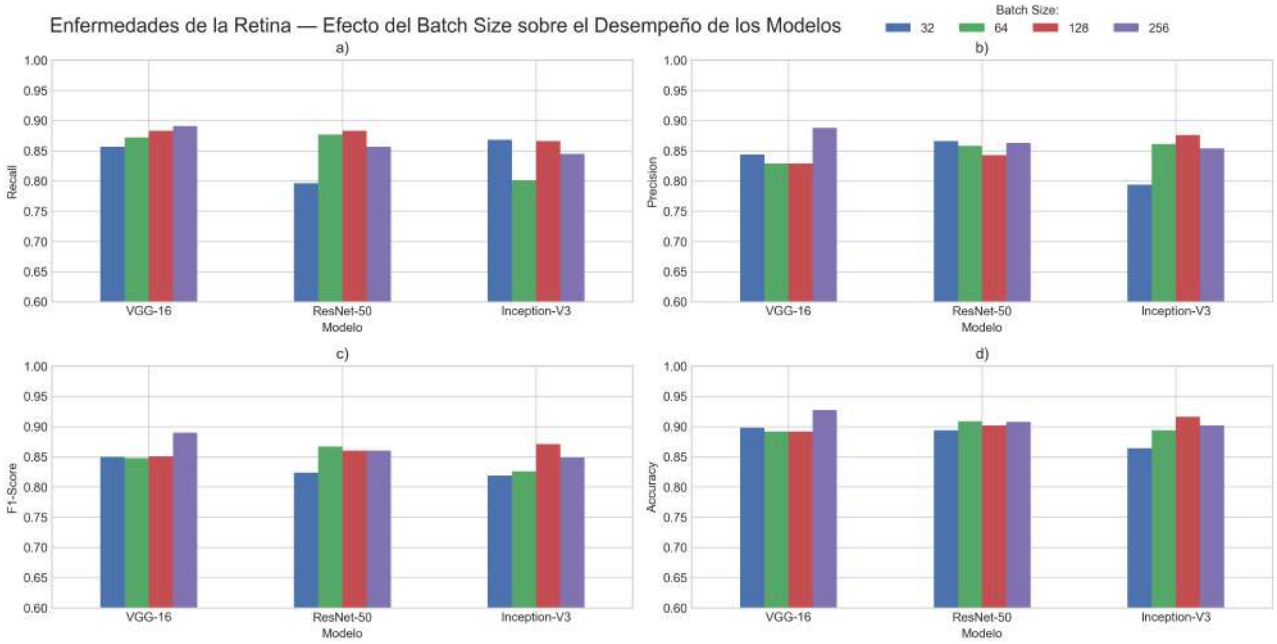


Fig. 5.3: Efecto del Batch-Size sobre el rendimiento de los modelos VGG-16, ResNet-50 e Inception-V3 para la base de datos Enfermedades de la Retina. a) Recall; b) Precision; c) F1-Score; d) Accuracy.

también VGG-16 el modelo con mejor desempeño, esta vez con un tamaño de batch igual a 256. Estos modelos fueron seleccionados para la evaluación en la siguiente etapa.

5.3 Efecto del Learning-Rate sobre el Rendimiento

Luego de evaluar los modelos seleccionados para distintos valores del hiperparámetro learning-rate (LR), se obtuvieron los resultados expuestos en la Fig. 5.4. Los mejores resultados son presentados en la Tabla 5.4 y el detalle para todas las métricas en Anexo B, Tabla B.4.

Para todas las bases de datos, los mejores resultados se obtuvieron con un LR igual a 1e-3, valor que coincide con el utilizado en la etapa anterior. Por otra parte, los peores resultados para todas las bases de datos se obtuvieron con un LR igual a 1e-1. El alto número de clasificaciones incorrectas en estos últimos resultados, además del mal desempeño en las métricas obtenidas para cada uno de ellos, indica que los pesos del modelo fueron ajustados de forma drástica y errada al no encontrar valores adecuados que orienten al optimizador en la minimización de la pérdida.

Con un LR igual a 1e-2, los resultados mejoraron bastante, pero aún así distan de los óptimos. Esta mejora se atribuye al uso del callback ReduceLROnPlateau, quien logró notar el estancamiento en la mejora de los pesos y redujo el LR a 1e-3, por lo que el modelo mejoró su entrenamiento.

Para un LR igual a 1e-4 y 1e-5, los resultados obtenidos en las bases de datos Covid-19 y Enfermedades de la Retina, son los segundos y terceros mejores pero, aún así, la cantidad de imágenes clasificadas de forma correcta dista considerablemente de las obtenidas con un LR igual a 1e-3.

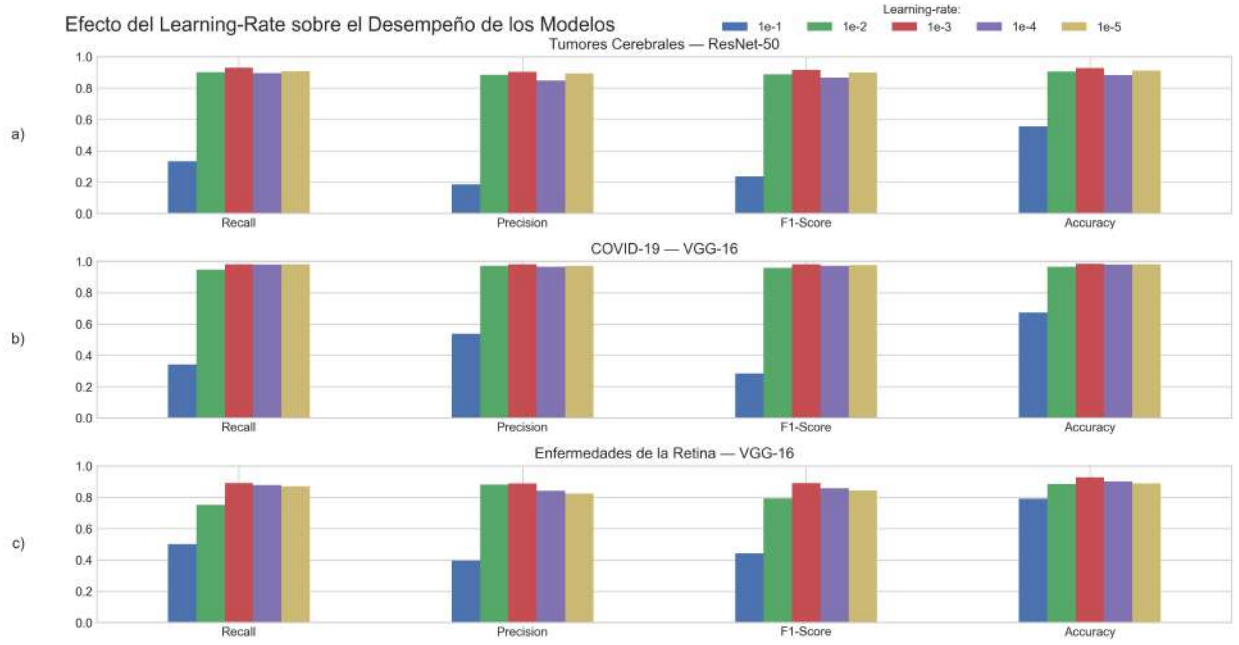


Fig. 5.4: Efecto del learning-rate sobre el desempeño de los modelos en cada base de datos. a) Tumores Cerebrales con ResNet-50; b) Covid-19 con VGG-16; c) Enfermedades de la Retina con VGG-16.

Tabla 5.4: Resultados para el efecto del learning-rate sobre el rendimiento de los modelos en cada base de datos. Las métricas presentadas corresponden al promedio entre la totalidad de clases.

BD	Modelo	BS	LR	Recall	Precision	F1-Score	Accuracy	Correcto	Incorrecto	Total
TC	ResNet-50	256	1e-3	0.931	0.904	0.915	0.928	426	33	459
C19	VGG-16	64	1e-3	0.980	0.980	0.980	0.984	2.982	49	3.031
ER	VGG-16	256	1e-3	0.891	0.888	0.890	0.927	593	47	640

Donde BD: base de datos; TC: tumores cerebrales; C19: covid-19; ER: enfermedades de la retina; BS: batch-size; LR: learning-rate.

5.4 Efecto del Aumento de Datos sobre el Rendimiento

Para el aumento de datos, las transformaciones aplicadas fueron seleccionadas teniendo en cuenta preservar las características naturales de la imagen original y, por lo tanto, se obtuvieron imágenes realistas para cada patología, y son ejemplificadas en Anexo A, Fig. A.1. Los resultados para el efecto del aumento de datos sobre el rendimiento de la clasifcación se presentan en la Tabla 5.5. En ella, se

muestra una comparación de las métricas promedio obtenidas al evaluar el modelo antes y después del aumento de datos. Además, se presenta una extensión de los resultados obtenidos por cada clase en el Anexo B, Tabla B.5.

El aumento de datos demostró no ser efectivo en la mejora del rendimiento de los modelos para ninguna base de datos. Al contrario, el desempeño en la clasificación fue perjudicado.

En la base de datos Tumores Cerebrales, la métrica precision aumentó en la clase 0, y se redujo en la clases 1 y 2. Por otra parte, la métrica recall disminuyó en las clases 0 y 1, y mantuvo el puntaje perfecto para la clase 2. La combinación de estas dos métricas es presentada en f1-score, donde todas las clases se vieron perjudicadas. En cuanto al accuracy, que toma en cuenta todas las imágenes, disminuyó de 92.8 % a 90.6 %. Considerando el desempeño de manera general, todas las métricas promedio empeoraron y se obtuvo una disminución en diez unidades en la cantidad de imágenes clasificadas correctamente, nueve para la clase meningioma y una para la clase glioma.

Para la base de datos Covid-19 la métrica precision empeoró considerablemente en la clase 0, pasando de 98.9 % a 92.2 %. Sin embargo, mejoró levemente en el resto de las clases, con un aumento de 97.8 % a 98.7 % para la clase 1, y de 97.3 % a 98.2 % para la clase 2. Respecto a la métrica recall, esta presentó un aumento de 98.9 % a 99.4 % en la clase 0, pero una disminución en las clases 1 y 2. En la clase 1, la reducción fue de 97.8 % a 96.4 %, mientras que en la clase 0 el descenso fue más considerable, pasando de 97.8 % a 81.8 %. Estos resultados en las métricas precision y recall, significaron una reducción del f1-score en todas las clases. Por último, la métrica accuracy presentó una importante baja de 98.4 % a 93.9 %. Como la disminución en todas las métricas promedio fue de una alta magnitud, el rendimiento empeoró en la clasificación correcta de más de 100 imágenes. Si bien la clase normal fue beneficiada en 10 unidades, las clases neumonía y covid fueron perjudicadas en 31 y 114 imágenes, respectivamente.

Por último, para la base de datos Enfermedades de la Retina, la métrica precision empeoró de 82.2 % a 75.7 % en la clase 0, y aumentó de 95.4 % a 96.1 % en la clase 1, resultados que significaron un descenso de 88.8 % a 85.9 % en el promedio. En recall, el promedio aumentó levemente de 89.1 % a 89.3 %, debido a la mejora de 82.8 % a 85.8 % en la clase 0 y al empeoramiento de 95.3 % a 92.7 % en la clase 1. Como resultado de la variación en ambas métricas, el valor promedio de f1-score pasó de 89.0 % a 86.6 %. Por último, la métrica accuracy decayó de 92.7 % a 91.3 %. Debido al efecto del aumento de datos, se obtuvo un incremento en 4 imágenes clasificadas correctamente para la clase 0, pero una disminución de 13 en la clase 1, lo que permitió concluir que la aplicación de esta técnica a la base de datos Enfermedades de la Retina no es beneficiosa.

Tabla 5.5: Resultados para el efecto del aumento de datos sobre el rendimiento en las bases de datos Tumores Cerebrales, Covid-19 y Enfermedades de la Retina.

BD	P-a	P-b	R-a	R-b	F-a	F-b	Acc-a	Acc-b	C-a	C-b	I-a	I-b	T
TC	0.904	0.887	0.931	0.904	0.915	0.892	0.928	0.906	426	416	33	43	459
C19	0.980	0.964	0.980	0.925	0.970	0.924	0.984	0.939	2.982	2.847	49	184	3.031
ER	0.888	0.859	0.891	0.893	0.890	0.866	0.927	0.913	593	584	47	56	640

Donde BD: base de datos; TC: tumores cerebrales; C19: covid-19; ER: enfermedades de la retina; P: precision; R: recall; F: F1-score; Acc: accuracy; C: imágenes clasificadas correctamente; I: imágenes clasificadas incorrectamente; T: cantidad total de imágenes; a: previo al aumento de datos; b: posterior al aumento de datos.

5.5 Efecto del Fine-Tuning sobre el Rendimiento

Para el estudio del efecto del fine-tuning, debido a que los resultados en la etapa de aumento de datos no fueron positivos, se consideraron los modelos obtenidos al final de la etapa 2. Esto, dado que el objetivo es lograr una mejora en el rendimiento de los modelos con mejores resultados. Para cada modelo, luego de habilitar el entrenamiento en las capas de la segunda mitad, se reanudó el entrenamiento por cinco épocas más con un learning-rate reducido de 1e-5. Como resultado, el rendimiento aumentó para las bases de datos Tumores Cerebrales y Covid-19, y empeoró para Enfermedades de la Retina. Los resultados promedio se presentan en la Tabla 5.6, mientras que los resultados por cada clase son expuestos en el Anexo B, Tabla B.6.

Para la base de datos Tumores Cerebrales, la métrica precision mejoró en las clases 0 y 2, y empeoró en la clase 1. La métrica recall mejoró solo en la clase 1, manteniendo el resultado en las clases 0 y 2. Las mejoras descritas permitieron un aumento en f1-score en todas las clases. Por último, la métrica accuracy aumentó de 92.8 % a 94.6 %. El fine-tuning permitió aumentar todas las métricas promedio, lo que influyó en la obtención de ocho unidades más en las imágenes clasificadas correctamente.

Para la base de datos Covid-19 las métricas también aumentaron en la mayoría de los casos, pero de forma menos significativa que en la base de datos anterior. Por una parte, la métrica precision logró aumentar levemente en todas las clases. Por otra parte, el valor para recall aumentó en las clase 0 y 2, y no presentó modificación en la clase 1. Estos resultados influyeron positivamente sobre f1-score, aumentando en todas las clases. La métrica accuracy logró un aumento leve de 0.4 %. Para esta base de datos, a pesar de que las métricas promedio no presentaron aumentos tan significativos, la aplicación de fine-tuning logró obtener doce imágenes más clasificadas de forma correcta.

Para la base de datos Enfermedades de la Retina, los resultados del fine-tuning fueron mixtos. La métrica precision disminuyó en la clase 0 pero aumentó en la clase 1. El recall, por su parte, aumentó en la clase 0 pero disminuyó en la clase 1. Sin embargo, al considerar la métrica f1-score, los valores disminuyeron en ambas clases. Por último, la métrica accuracy presentó un decrecimiento de 92.7 % a 90.8 %. Teniendo en cuenta los resultados promedio para cada métrica, y la disminución de doce imágenes clasificadas correctamente, se pudo concluir que el fine-tuning no es fructuoso para esta base de datos.

Tabla 5.6: Resultados para el efecto del fine-tuning sobre el rendimiento en las bases de datos Tumores Cerebrales, Covid-19 y Enfermedades de la Retina.

BD	P-a	P-b	R-a	R-b	F-a	F-b	Acc-a	Acc-b	C-a	C-b	I-a	I-b	T
TC	0.904	0.932	0.931	0.941	0.915	0.935	0.928	0.946	426	434	33	25	459
C19	0.980	0.986	0.980	0.982	0.980	0.984	0.984	0.988	2.982	2.994	49	37	3.031
ER	0.888	0.850	0.891	0.898	0.890	0.870	0.927	0.925	593	581	47	59	640

Donde: BD: base de datos; TC: tumores cerebrales; C19: covid-19; ER: enfermedades de la retina; P: precision; R: recall; F: f1-score; Acc: accuracy; C: imágenes clasificadas correctamente; I: imágenes clasificadas incorrectamente; T: cantidad total de imágenes; a: previo al fine-tuning; b: posterior al fine-tuning.

5.6 Comparación de Rendimiento

A modo de conclusión general para las etapas implementadas, se presenta una comparación para el rendimiento de la clasificación antes y después de aplicar las técnicas para su mejora, es decir, una comparación entre los resultados obtenidos al final de las etapas 2 y 4. La Fig. 5.5 expone las matrices de confusión al finalizar la etapa 2 (izquierda) y al finalizar la etapa 4 (derecha) para cada base de datos.

En la base de datos Tumores Cerebrales, la clase meningioma no presentó mejoras en la métrica recall pero sí en precision y, por lo tanto, en f1-score, aumentando de 86.2 % a 92.6 % y de 85.8 % a 88.9 % en cada una, respectivamente. La clase glioma presentó mejoras en recall, aumentando de 93.7 % a 96.9 %. Sin embargo, la precision disminuyó de 98.8 % a 97.2 %. De todas formas, esto significó un leve aumento en f1-score, incrementando de 96.2 % a 97.1 %. Por último, la clase pituitario mantuvo su puntaje perfecto en recall y presentó un aumento de 86.1 % a 89.7 % en precision, lo que resultó en un crecimiento de 92.6 % a 94.6 % en f1-score.

Para la base de datos Covid-19, la clase normal aumentó el recall de 98.9 % a 99.4 % producto de 11 imágenes más clasificadas correctamente. La precision para esta clase aumentó de forma minúscula: de 98.9 % a 99.0 %. El aumento en ambas métricas significó también un aumento en f1-score, acrecentando el puntaje de 98.9 % a 99.2 %. Para la clase neumonía, el recall se mantuvó en 97.8 % y la precision aumentó de 97.8 % a 98.2 %, influyendo en la leve mejora de 97.8 % a 98.0 % en f1-score. Finalmente, la clase covid presentó mejoras en todas las métricas, con aumentos de 97.3 % a 97.4 % para recall, de 97.3 % a 98.5 % para precision, y de 97.3 % a 98.0 % para f1-score. Para esta base de datos, el aumento en la mayoría de las métricas fue ligero debido a que el alto rendimiento obtenido previamente es complejo de mejorar.

Para la base de datos Enfermedades de la Retina, los resultados en cada métrica fueron positivos para una clase y negativos para la otra. En la clase 0, el recall aumentó de 82.8 % a 88.1 % mientras que precision disminuyó de 82.2 % a 73.3 %, lo que produjo un resultado negativo sobre la métrica f1-score, la cual descendió de 82.5 % a 80.0 %. Por otra parte, en la clase 1 el recall presentó una caída de 95.3 % a 91.5 %, mientras que la precision subió de 95.4 % a 96.7 % y, por consecuencia, se obtuvo una alteración negativa en f1-score, decayendo de 95.4 % a 94.0 %.

Finalmente, los valores promedio para cada métrica en cada base de datos, son presentados en la Tabla 5.7. Para las bases de datos Tumores Cerebrales y Covid-19, todas las métricas promedio aumentaron y, si bien la aplicación de aumento de datos no fue útil, fine-tuning influyó sustancialmente y de forma positiva en el rendimiento de la clasificación. Para la base de datos Enfermedades de la Retina, ninguna de las técnicas lograron mejorar el rendimiento.

Tabla 5.7: Comparación de resultados antes y después de la aplicación de técnicas para mejorar el rendimiento de la clasificación. Los resultados presentados corresponden a las métricas promedio entre la totalidad de clases.

BD	P-a	P-b	R-a	R-b	F-a	F-b	Acc-a	Acc-b	C-a	C-b	I-a	I-b	T
TC	0.904	0.932	0.931	0.941	0.915	0.935	0.928	0.946	426	434	33	25	459
C19	0.980	0.986	0.980	0.982	0.970	0.984	0.984	0.988	2.982	2.994	49	37	3.031
ER	0.888	0.884	0.891	0.893	0.890	0.888	0.927	0.925	593	592	47	48	640

Donde BD: base de datos; TC: tumores cerebrales; C19: Covid-19; ER: enfermedades de la retina; P: precision; R: recall; F: f1-score; Acc: accuracy; C: imágenes clasificadas correctamente; I: imágenes clasificadas incorrectamente; T: cantidad total de imágenes; a: previo al aumento de datos y fine-tuning; b: posterior al aumento de datos y fine-tuning.

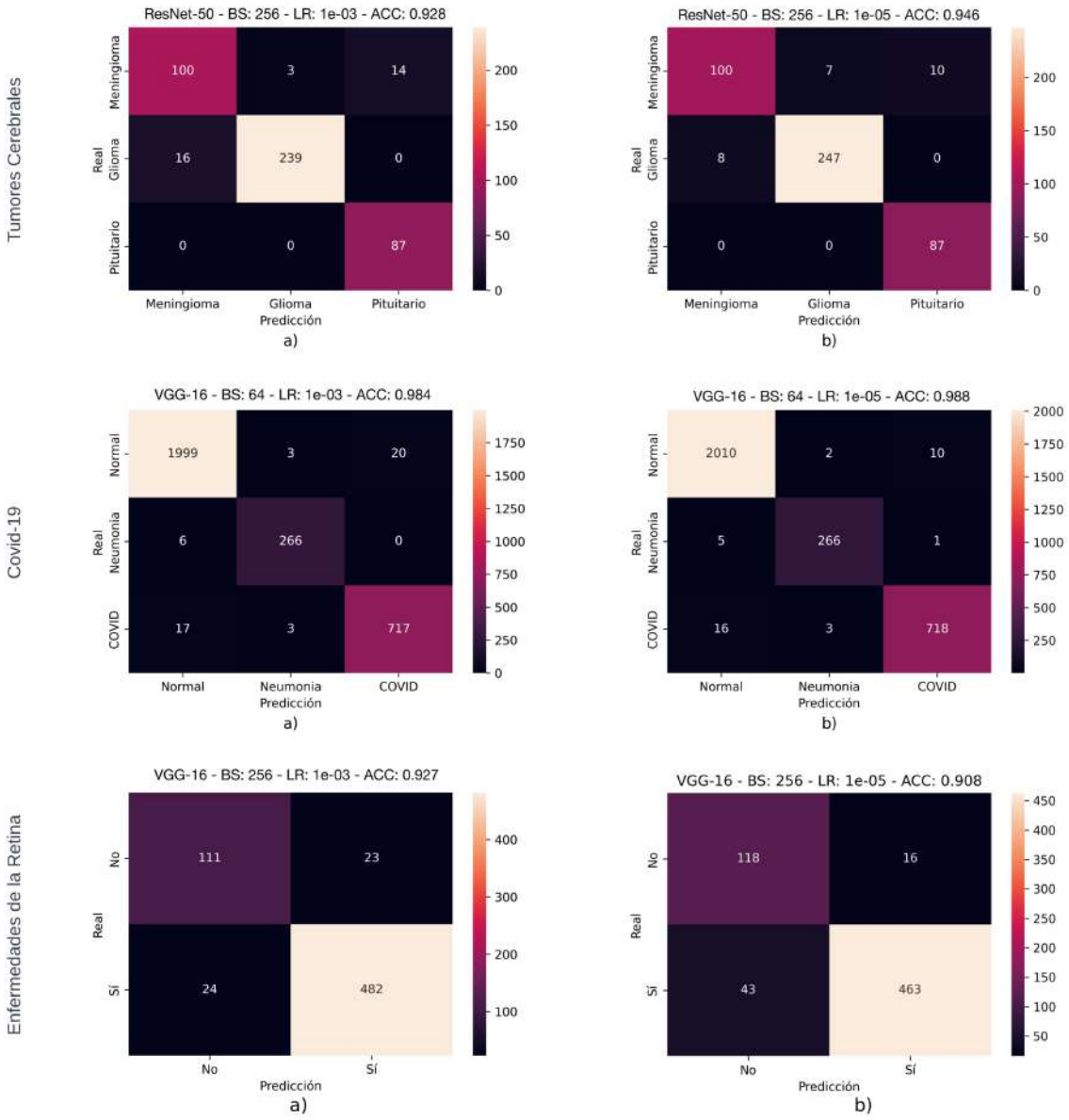


Fig. 5.5: Matrices de confusión comparativas para cada base de datos, antes y después de la aplicación de técnicas para mejorar el rendimiento. a) matriz de confusión luego de las etapas 1 y 2. b) matriz de confusión luego de las etapas 3 y 4. BS: batch-size; LR: learning-rate; ACC: accuracy.

5.7 Definición de Mejores Modelos

Gracias al proceso de análisis realizado en las etapas anteriores, se logró obtener cuáles fueron los mejores modelos junto con las óptimas configuraciones para cada base de datos. Para la base de datos Tumores Cerebrales, el mejor modelo fue ResNet-50, con un batch-size de 256 y un learning-rate de 1e-3. En este caso, el aumento de datos perjudicó el rendimiento pero el fine-tuning logró incrementar las métricas y cantidad de imágenes clasificadas correctamente. Por otra parte, para la

base de datos Covid-19, el mejor modelo fue VGG-16, con un batch-size de 64 y un learning-rate de 1e-3. Nuevamente la técnica de aumento de datos influyó negativamente y fine-tuning, por el contrario, logró optimizar el rendimiento. Por último, la base de datos Enfermedades de la Retina obtuvo su mejor rendimiento con el modelo VGG-16, un batch-size de 256 y un learning-rate de 1e-3. En este caso, tanto el aumento de datos como el fine-tuning perjudicaron el desempeño de la clasificación. Esta información es resumida en la Tabla 5.8.

Tabla 5.8: Resumen de modelos y configuraciones con mejor rendimiento para cada base de datos

Base de Datos	Modelo	Batch-Size	Learning-Rate	Aumento de Datos	Fine-Tuning
Tumores Cerebrales	ResNet-50	256	1e-3	No	Sí
Covid-19	VGG-16	64	1e-3	No	Sí
Enfermedades de la Retina	VGG-16	256	1e-3	No	No

Para finalizar el estudio, se evaluó el proceso de entrenamiento de los mejores modelos para garantizar una buena capacidad de generalización sobre datos nuevos. La Fig. 5.6 expone las curvas de aprendizaje para los valores de pérdida de los conjuntos de entrenamiento y validación, para cada base de datos. Para la base de datos Tumores Cerebrales, el valor de pérdida en los conjuntos de entrenamiento (curva azul) y validación (curva magenta) comenzó su estabilización alrededor de la quinta época. En la base de datos Covid-19, el valor de pérdida tuvo un comportamiento volátil en las primeras épocas, pero logró su estabilización en ambos conjuntos en tiempos similares. Para la base de datos Enfermedades de la Retina, el valor de pérdida para el conjunto de validación comenzó su estabilización antes que en el conjunto de entrenamiento. Alrededor de la cuarta época, las pérdidas comenzaron a divergir, lo cual es un signo de sobreajuste. Si bien los valores de pérdida en los conjuntos de validación para las bases de datos Tumores Cerebrales y Covid-19 no se redujeron más tras el paso de las primeras épocas, tampoco presentaron un aumento y se mantuvieron estables por el resto del entrenamiento. Esto es un resultado positivo, pues no se presentó el fenómeno de sobreajuste dado que la brecha entre ambos conjuntos se mantuvo estable por el resto de las épocas. Finalmente, al no presentarse mejoras en la pérdida luego de diez épocas, la función auxiliar early-stopping detuvo el entrenamiento y restauró los pesos óptimos encontrados en épocas anteriores.

De forma similar, se pudo evaluar el comportamiento de la métrica accuracy a medida que aumentaron las épocas (Fig. 5.7). En la base de datos Tumores Cerebrales la curva del conjunto de entrenamiento alcanzó un accuracy superior al 95 % alrededor de la quinta época, donde también comenzó su estabilización la curva del conjunto de validación. Por el resto de épocas, la curva para el conjunto de entrenamiento siguió creciendo mientras que la del conjunto de validación se mantuvo relativamente estable, presentando pequeñas variaciones ocasionalmente. En la base de datos Covid-19, se presentó un comportamiento similar. Ambas curvas superaron el 95 % alrededor de la quinta época

y luego comenzaron su estabilización. Por último, en la base de datos Enfermedades de la Retina, los valores obtenidos en ambos conjuntos fueron menores. Alrededor de la quinta época ambas curvas alcanzaron valores cercanos al 85 % y, a partir de ese momento, solo hubo mejoras en la curva de entrenamiento, mientras que la de validación se estabilizó. La evidente divergencia en las curvas para esta base de datos es congruente con la presentada en las curvas para valores de pérdida, reforzando la conclusión de que se presentó algún grado de sobreajuste.

De estos gráficos también se pudo concluir que el conjunto de imágenes de entrenamiento que presentó mayor complejidad para identificar y aprender las características, corresponde a la base de datos Enfermedades de la Retina. Los valores alcanzados en accuracy, para este conjunto, fueron de aproximadamente el 95 %. Por otra parte, para la base de datos Tumores Cerebrales y Covid-19, el accuracy en el conjunto de entrenamiento alcanzó valores cercanos al 99 %, resultado que indica que las características presentes en las imágenes para estas bases de datos resultaron más sencillas de aprender que para la base de datos Enfermedades de la Retina. Respecto a los valores de accuracy para los conjuntos de validación y prueba, se pudo concluir que aquellas bases de datos que presentan mayor dificultad al clasificar cada imagen son, en orden decreciente, Enfermedades de la Retina, Tumores Cerebrales y Covid-19.

Los resultados presentados en estos gráficos, son un fiel reflejo del desempeño obtenido en la clasificación de las imágenes pertenecientes a los conjuntos de prueba, consiguiendo modelos competentes para el trabajo con datos nuevos.

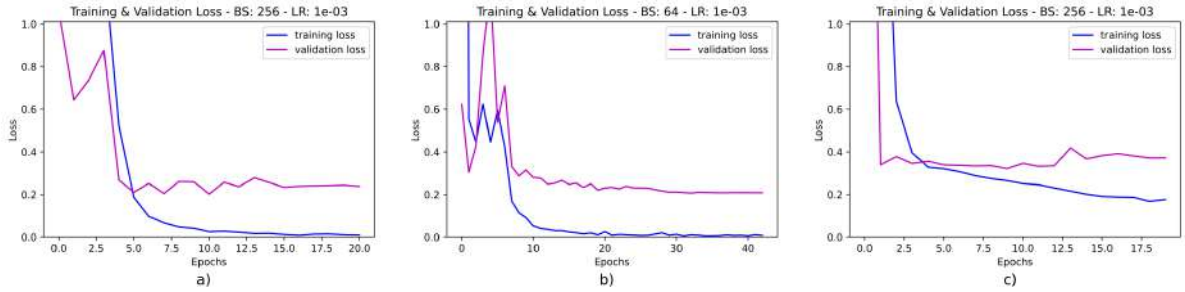


Fig. 5.6: Curvas de aprendizaje para los mejores modelos. Se presentan los valores de pérdida en los conjuntos de entrenamiento y validación para cada base de datos. a) Tumores Cerebrales; b) Covid-19; c) Enfermedades de la Retina; BS: batch-size; LR: learning-rate; ACC: accuracy.

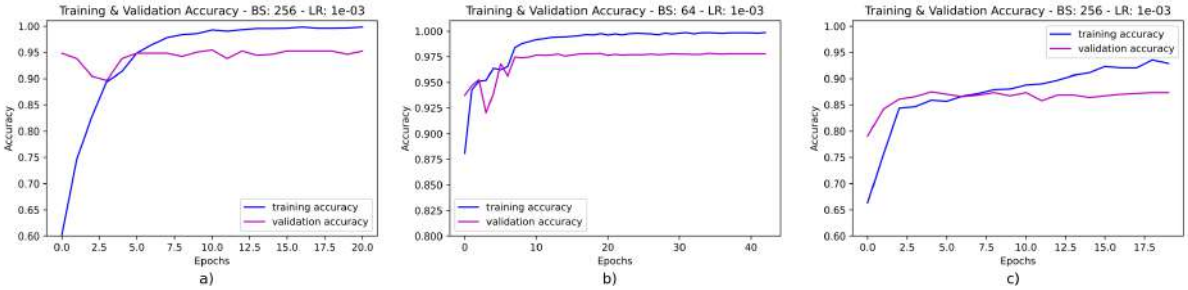


Fig. 5.7: Curvas de aprendizaje para los mejores modelos. Se presentan los valores de accuracy en los conjuntos de entrenamiento y validación para cada base de datos. a) Tumores Cerebrales; b) Covid-19; c) Enfermedades de la Retina; BS: batch-size; LR: learning-rate; ACC: accuracy.

5.8 Tutoriales

Se construyó un tutorial para cada base de datos considerando la información obtenida en el estudio. Estos tutoriales fueron pensados para su ejecución en la plataforma Google Colab que, dentro de sus principales beneficios, están la posibilidad de utilizar GPU y la integración con todas las bibliotecas de programación necesarias para su ejecución.

Cada tutorial sigue la misma estructura, la cual es detallada a continuación, indicando los principales tópicos cubiertos en cada una.

1. **Introducción.** En ella, se introduce la tarea de clasificación a realizar y el modelo de red neuronal convolucional que será implementado. Además, se explica cómo configurar la plataforma para hacer uso de la GPU.

2. **Listado e importación de bibliotecas.** Se describe la utilidad de cada biblioteca necesaria para la implementación de la clasificación y la forma en que son importadas en el código.

3. **Descripción de base de datos.** Se estudia la base de datos de imágenes utilizada, considerando la cantidad total de imágenes disponibles y su división en las respectivas clases. Además, de ser necesario, se revisan los archivos con información adicional.

4. **Carga de imágenes.** Se describe el procedimiento para la carga de imágenes a la plataforma, el cual puede ser realizado como una carga directa del archivo, o a través de la articulación entre la plataforma Google Drive y Google Colab.

5. **Generación de conjuntos de datos.** En esta sección se explica cómo construir los tres conjuntos de datos para la implementación del algoritmo, es decir, entrenamiento, validación y prueba.

Dependiendo de la base de datos utilizada, su construcción requiere de un procedimiento diferente. Además, se instruye en el uso de las funciones "Generadores"de la biblioteca Keras que permiten, además de aplicar la técnica aumento de datos, cargar las imágenes de forma progresiva para alivianar los costos computacionales y no presentar errores referentes a límites de memoria.

6. **Configuración de Callbacks.** Se explica cómo configurar las funciones auxiliares Early Stopping, ReduceLROnPlateau y ModelCheckpoint. Para cada una, se expone cuál es la tarea que desempeñan y cómo ajustar sus parámetros principales.

7. **Carga de modelo pre-entrenado y modificación para la tarea específica de clasificación.** Se explica, dependiendo de cada base de datos, cómo implementar el modelo de CNN a utilizar junto a la carga de pesos pre-entrenados, y cómo llevar a cabo la modificación de la arquitectura para hacerla compatible con la tarea de clasificación específica a resolver.

8. **Compilación y entrenamiento del modelo.** Se explica cómo realizar la compilación del modelo con los hiperparámetros optimizador y función de pérdida. Además, para el entrenamiento del modelo, se muestra cómo configurar los hiperparámetros epochs y batch-size, y el paso de conjuntos de entrenamiento y validación, además de callbacks, al modelo. Por último, se explica cómo evaluar el rendimiento del modelo a medida que está siendo entrenado. Esto, con ayuda de la información presentada en pantalla respecto a los conjuntos de entrenamiento y validación.

9. **Fine-tuning.** Dependiendo de la base de datos y el modelo de CNN utilizado, se instruye en la aplicación de la técnica fine-tuning. Se muestra cómo alterar la configuración de las capas presentes en la red para activar o inhibir su entrenamiento y, una vez hecho esto, re-entrenar parte del modelo con el objetivo de aumentar el rendimiento.

10. **Predicción sobre el conjunto de prueba.** Una vez que el modelo haya sido entrenado y pasado por el proceso de fine-tuning, se muestra cómo aplicarlo al conjunto de prueba y cómo interpretar los valores obtenidos.

11. **Despliegue de métricas y evaluación de resultados.** Se explica cómo obtener las métricas principales para la evaluación de rendimiento. Estas son: reporte de clasificación, matrices de confusión y curvas de aprendizaje. Además, se indica cómo interpretar cada una de ellas con el fin de obtener conclusiones sobre el rendimiento que posee el modelo en la clasificación.

12. **Conclusión.** Se realiza una recapitulación de lo cubierto en el tutorial y se indican conclusiones generales.

El conjunto de etapas descritas cubre todo lo estudiado a lo largo del informe, y posibilita la obtención de un modelo de inteligencia artificial capaz de clasificar imágenes médicas.

Los tutoriales se encuentran disponibles en formato Notebook en los siguientes enlaces:

- Tutorial Tumores Cerebrales
- Tutorial Covid-19
- Tutorial Enfermedades de la Retina

Además, como ejemplo, el tutorial para Covid-19 se presenta en el Anexo C.

6 Capítulo 6. Conclusiones

6.1 Discusión

Las bases de datos utilizadas demostraron ser efectivas para una buena clasificación. Si bien todas presentaron desbalance de clases, esto no significó mayor problema, ya que la representatividad fue suficiente para la obtención de buenos resultados. Por otra parte, el uso de pesos pre-entrenados en la base de datos ImageNet, demostró su idoneidad para capturar las características fundamentales presentes en las imágenes, destacando el potencial que posee la transferencia de aprendizaje.

De los modelos evaluados, fueron ResNet-50 y VGG-16 quienes demostraron ser los más competentes al trabajar con los datos disponibles y, si bien Inception-V3 no obtuvo en ningún caso el mejor rendimiento, los resultados que presentó fueron de todas formas destacables.

Se demostró que la elección de los valores óptimos para los hiperparámetros batch-size y learning-rate es un aspecto crucial en el entrenamiento de los modelos. El primero, tuvo directa influencia en el valor de pérdida calculado en el entrenamiento, puesto que se comprobó que la cantidad de datos utilizados en el cálculo puede alterar significativamente el resultado. Por otra parte, el segundo jugó un importante rol en guíar que la modificación de kernels y pesos fuese a favor de dos importantes aspectos: la correcta extracción de características relevantes y la minimización del valor de pérdida.

Con el objetivo de reforzar el rendimiento, la técnica de aumento de datos no presentó buenos resultados. Esto se puede explicar por la constante variación en el conjunto de entrenamiento, ya que los datos son diferentes en cada época. A pesar de que las transformaciones aplicadas sobre las imágenes fueron sutiles para preservar las características naturales de las patologías, los modelos no lograron aprender de la información inconstante. Por el contrario, la técnica fine-tuning demostró su efectividad. Su aplicación contribuyó con importantes mejoras en las métricas y, en consecuencia, en un aumento en la cantidad de imágenes clasificadas correctamente. El efecto positivo fue más notorio en las bases de datos Tumores Cerebrales y Covid-19, mientras que para la base de datos Enfermedades de la Retina se presentaron resultados mixtos. Sin embargo, las funciones auxiliares Early-Stopping, ReduceLROnPlateau y ModelCheckpoint ayudaron a sortear en gran parte estos inconvenientes, colaborando en una buena generalización de los modelos y evitando fenómenos de sobreajuste.

Sobre los errores cometidos en la clasificación, en la base de datos Tumores Cerebrales ocurrieron

mayoritariamente y de forma recíproca en las clases meningioma y glioma. Las formas y tamaños similares que estos tipos de tumores presentan, y el hecho de que su posicionamiento en el cerebro es altamente variable, podría explicar estos errores. Por otra parte, si bien el tumor pituitario puede variar su tamaño, su ubicación es prácticamente inalterable, características que el algoritmo puede aprender a identificar con mayor facilidad. Para Covid-19, los errores ocurrieron principalmente entre las clases normal y covid. Esto se puede explicar teniendo en cuenta lo estudiado en la sección de la patología, donde se indica que la radiografía de tórax no siempre es la técnica más adecuada para diagnosticar la enfermedad en etapas iniciales, lo cual podría ser el caso en imágenes de la clase covid, asemejándose a las de la clase normal por la poca o nula presencia de anomalías. Finalmente, para Enfermedades de la Retina, el hecho de que la cantidad de patologías presentes en las imágenes de retinas enfermas es elevada puede significar demasiada información para ser interpretada por el modelo como solo una clase. Ejemplos para imágenes clasificadas correctamente e incorrectamente son presentados en las Figuras 6.1, 6.2 y 6.3 para las bases de datos Tumores Cerebrales, Covid-19 y Enfermedades de la Retina, respectivamente. En cada imagen presente en las figuras, se indica la clase real, la clase predicha y la probabilidad de pertenencia de la clase predicha a la real.

Los modelos obtenidos permitieron alcanzar puntajes altos en las métricas de evaluación, valores que son comparables a los presentados en los trabajos previos que tenían objetivos similares.

Por último, los tutoriales, al haber sido construidos con el apoyo de información fundamentada, garantizan un correcto aprendizaje en la implementación de redes neuronales convolucionales para la clasificación de imágenes médicas.

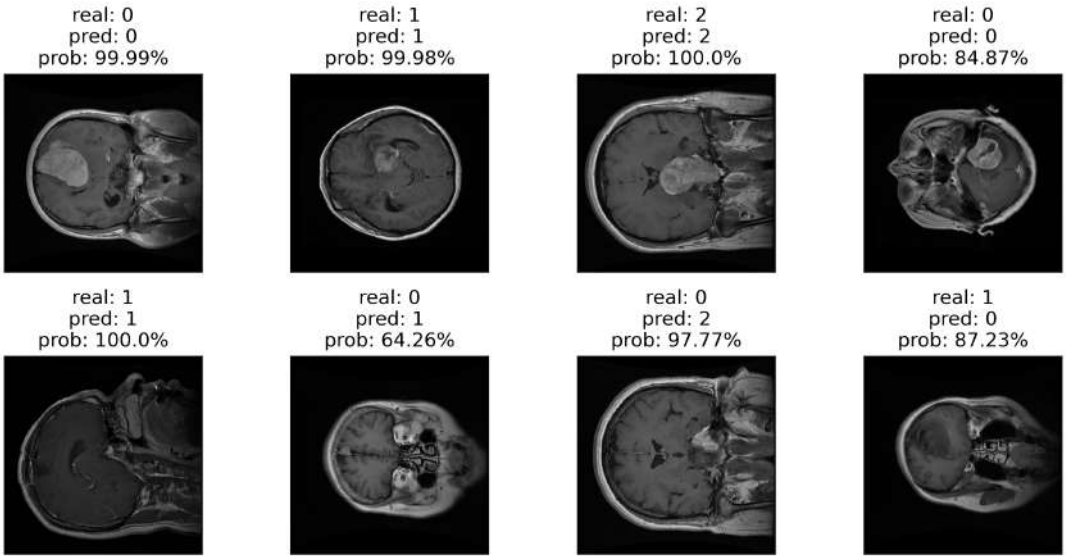


Fig. 6.1: Predicciones para Tumores Cerebrales. En cada imagen se indica la clase real, la clase predicha y la probabilidad de pertenencia de la clase predicha a la clase real. 0: meningioma; 1: glioma; 2: pituitario.

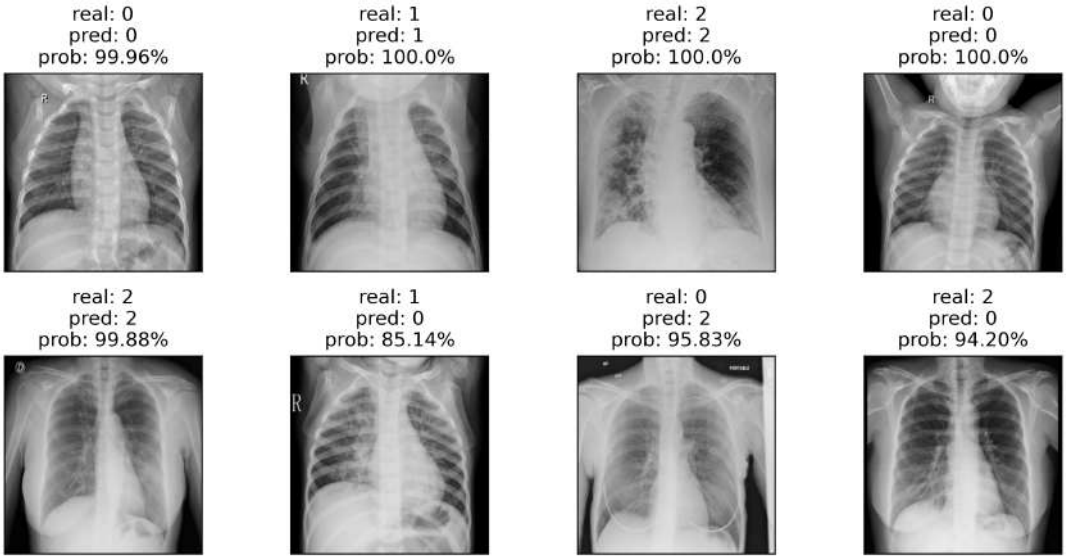


Fig. 6.2: Predicciones para Covid-19. En cada imagen se indica la clase real, la clase predicha y la probabilidad de pertenencia de la clase predicha a la clase real. 0: normal, 1: neumonía; 2: covid.

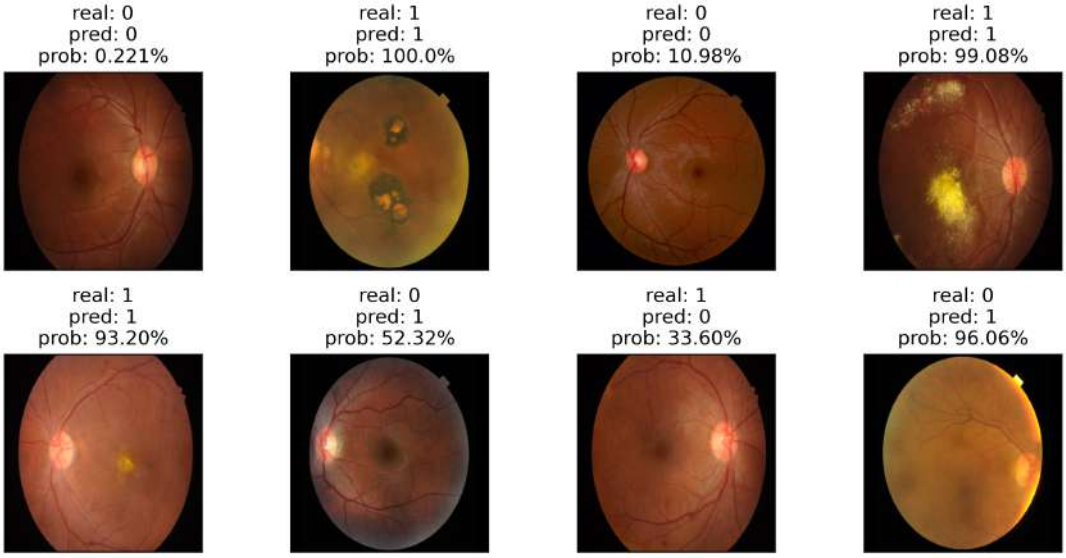


Fig. 6.3: Predicciones para Enfermedades de la Retina. En cada imagen se indica la clase real, la clase predicha y la probabilidad de pertenencia de la clase predicha a la clase real. Para esta base de datos, si la probabilidad es mayor al 50 % la imagen es clasificada como clase 1, mientras que si la probabilidad es menor al 50 % la imagen es clasificada como clase 0. Mientras el valor de probabilidad es más cercano a 100 %, mayor seguridad al clasificar como clase 1. Mientras el valor de probabilidad es más cercano a 0 %, mayor seguridad al clasificar como clase 0. 0: retina sana; 1: retina enferma.

6.2 Conclusiones

El desarrollo del trabajo permitió explorar los componentes fundamentales de uno de los algoritmos base del Deep Learning: la Red Neuronal Convolucional. Su uso es conveniente al trabajar en tareas de clasificación que involucren imágenes, pues la extracción de características relevantes se realiza de forma automática, siendo las imágenes con sus respectivas etiquetas la única información requerida. La aplicación del algoritmo a las bases de datos de imágenes médicas seleccionadas, demostró su competencia y eficiencia, obteniendo resultados destacables al trabajar con grandes cantidades de datos.

Los resultados obtenidos a lo largo del estudio permitieron la construcción de tutoriales que, al haber sido desarrollados en formato Notebook con Google Colab, presentan la ventaja de ser ejecutados por todo usuario. Esto, debido a que la plataforma cuenta con la integración de todas las bibliotecas necesarias. Además, los requerimientos en cuanto a hardware y software para su funcionamiento son mínimos. Cada tutorial, uno para cada base de datos utilizada, cubre los pasos fundamentales para la aplicación de los algoritmos en la práctica, y su utilidad es potenciada al ser complementada con el estudio presentado en el informe.

Los objetivos propuestos fueron cumplidos satisfactoriamente, permitiendo demostrar que el potencial que las herramientas de inteligencia artificial poseen en el apoyo de diagnóstico médico es notable, y seguirá en aumento a medida que se desarrollen nuevas tecnologías y se creen bases de datos más grandes y completas. El conocer cómo estos instrumentos funcionan y cómo llevar a cabo su implementación, es importante para el desarrollo de nuevos modelos, dada la alta cantidad de patologías que demandan el uso de la tecnología en favor de su diagnóstico y evaluación, lo que va, finalmente, en directo beneficio de la salud de todas y todos.

6.3 Trabajo Futuro

Como trabajo futuro sería interesante evaluar los modelos obtenidos en datos nacionales y ver qué tan bien se da el rendimiento y, en caso de presentar resultados positivos, desarrollar un sistema para implementarlos en el diagnóstico clínico.

En lo que respecta al área de clasificación de imágenes, se podría probar y evaluar otros hiperparámetros como distintos optimizadores o funciones de pérdida, ya que esto puede tener un efecto positivo en el rendimiento. Además, adentrarse en la clasificación para imágenes en tres dimensiones

es relevante, pues son muchas las utilidades que este tipo de imágenes tienen. Sin embargo, al ser archivos de un tamaño considerablemente mayor, los requerimientos y costos computacionales para la implementación de algoritmos de Deep Learning también aumentan.

Finalmente, se podría expandir el desarrollo de tutoriales a otras áreas del Machine Learning y Deep Learning que involucren imágenes médicas, otorgando enriquecimiento en la formación, complementando lo ya aprendido. Tópicos como segmentación y detección de objetos son también relevantes en el diagnóstico clínico, y el desarrollo de estas herramientas es posible dada la disponibilidad de materiales de libre acceso para su implementación.

Bibliografía

[1] H.-P. Chan, R. K. Samala, L. M. Hadjiiski, and C. Zhou, *Deep Learning in Medical Image Analysis*. Cham: Springer International Publishing, 2020, pp. 3–21. [Online]. Available: https://doi.org/10.1007/978-3-030-33128-3_1

[2] I. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, 05 2021.

[3] I. Kononenko, "Machine learning for medical diagnosis: history, state of the art and perspective," *Artificial Intelligence in Medicine*, vol. 23, no. 1, pp. 89–109, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S093336570100077X

[4] Y. Mintz and R. Brodie, "Introduction to artificial intelligence in medicine," *Minimally Invasive Therapy & Allied Technologies*, vol. 28, no. 2, pp. 73–81, 2019, pMID: 30810430. [Online]. Available: https://doi.org/10.1080/13645706.2019.1575882

[5] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aaa8415

[6] L. Saba, M. Biswas, V. Kuppili, E. Cuadrado Godia, H. S. Suri, D. R. Edla, T. Omerzu, J. R. Laird, N. N. Khanna, S. Mavrogeni, A. Protogerou, P. P. Sfikakis, V. Viswanathan, G. D. Kitas, A. Nicolaides, A. Gupta, and J. S. Suri, "The present and future of deep learning in radiology," *European Journal of Radiology*, vol. 114, pp. 14–24, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0720048X19300919

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[9] R. C. Deo, "Machine learning in medicine," *Circulation*, vol. 132, no. 20, pp. 1920–1930, 2015. [Online]. Available: https://www.ahajournals.org/doi/abs/10.1161/CIRCULATIONAHA.115.001593

[10] J. Mueller and L. Massaron, *Data Science Programming All-in-One For Dummies*. Wiley, 2019. [Online]. Available: https://books.google.cl/books?id=C97BDwAAQBAJ

[11] S. Uddin, A. Khan, M. Hossain, and M. A. Moni, "Comparing different supervised machine learning algorithms for disease prediction," *BMC Medical Informatics and Decision Making*, vol. 19, 12 2019.

[12] B. J. Erickson, P. Korfiatis, Z. Akkus, and T. L. Kline, "Machine learning for medical imaging," *RadioGraphics*, vol. 37, no. 2, pp. 505–515, 2017, pMID: 28212054. [Online]. Available: https://doi.org/10.1148/rg.2017160130

[13] C. Y. K. J. L. S.-H. Cho Gyeongcheol, Yim Jinyeong, "Review of machine learning algorithms for diagnosing mental illness," *Psychiatry Investig*, vol. 16, no. 4, pp. 262–269, 2019. [Online]. Available: http://www.psychiatryinvestigation.org/journal/view.php?number=1012

[14] Z. Zhang, "A gentle introduction to artificial neural networks," *Annals of Translational Medicine*, vol. 4, pp. 370–370, 10 2016.

[15] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," *Journal of Pharmaceutical and Biomedical Analysis*, vol. 22, no. 5, pp. 717–727, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0731708599002721

[16] R. Yamashita, M. Nishio, R. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, 06 2018.

[17] Phung and Rhee, "A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets," *Applied Sciences*, vol. 9, p. 4500, 10 2019.

[18] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361841517301135

[19] M. A. Mazurowski, M. Buda, A. Saha, and M. R. Bashir, "Deep learning in radiology: An overview of the concepts and a survey of the state of the art with focus on mri," *Journal of Magnetic Resonance Imaging*, vol. 49, no. 4, pp. 939–954, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jmri.26534

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[21] M. Elgendy, *Deep Learning for Vision Systems*. Manning Publications, 2020. [Online]. Available: https://books.google.cl/books?id=6gkLzAEACAAJ

[22] J. Kukačka, V. Golkov, and D. Cremers, "Regularization for deep learning: A taxonomy," 2018. [Online]. Available: https://openreview.net/forum?id=SkHkeixAW

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[24] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[25] C. Shorten and T. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, 07 2019.

[26] J. Nalepa, M. Marcinkiewicz, and M. Kawulok, "Data augmentation for brain-tumor segmentation: A review," *Frontiers in Computational Neuroscience*, vol. 13, 12 2019.

[27] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556*, 09 2014.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015. [Online]. Available: http://arxiv.org/abs/1512.00567

[31] V. Komanapalli, N. Sivakumaran, and S. Hampannavar, *Advances in Automation, Signal Processing, Instrumentation, and Control: Select Proceedings of i-CASIC 2020*, ser. Lecture Notes in Electrical Engineering. Springer Nature Singapore, 2021. [Online]. Available: https://books.google.cl/books?id=lqkhEAAAQBAJ

[32] L. Ali, F. Alnajjar, H. Jassmi, M. Gochoo, W. Khan, and M. Serhani, "Performance evaluation of deep cnn-based crack detection and localization techniques for concrete structures," *Sensors*, vol. 21, p. 1688, 03 2021.

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[34] M. Mahdianpari, B. Salehi, M. Rezaee, F. Mohammadimanesh, and Y. Zhang, "Very deep convolutional neural networks for complex land cover mapping using multispectral remote sensing imagery," *Remote Sensing*, vol. 10, p. 1119, 07 2018.

[35] T. Araújo, G. Aresta, E. Castro, J. Rouco, P. Aguiar, C. Eloy, A. Polónia, and A. Campilho, "Classification of breast cancer histology images using convolutional neural networks," *PLOS ONE*, vol. 12, p. e0177544, 06 2017.

[36] Y. Wang, E. J. Choi, Y. Choi, H. Zhang, G. Y. Jin, and S.-B. Ko, "Breast cancer classification in automated breast ultrasound using multiview convolutional neural network with transfer learning," *Ultrasound in Medicine Biology*, vol. 46, no. 5, pp. 1119–1132, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030156292030003X

[37] B. Ehteshami Bejnordi, M. Veta, P. Johannes van Diest, B. van Ginneken, N. Karssemeijer, G. Litjens, J. A. W. M. van der Laak, , and the CAMELYON16 Consortium, "Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer," *JAMA*, vol. 318, no. 22, pp. 2199–2210, 12 2017. [Online]. Available: https://doi.org/10.1001/jama.2017.14585

[38] T. Tran, C. Vununu, S. Atoev, S.-H. Lee, and K.-R. Kwon, "Leukemia blood cell image classification using convolutional neural network," *International journal of computer theory and engineering*, vol. 10, pp. 54–58, 04 2018.

[39] C. Boo-Kyeong, M. Nuwan, C. Heung-Kook, S. Jae-Hong, K. Cho-Hee, P. Hyeon-Gyun, B. Subrata, and P. Deekshitha, "Convolutional neural network-based mr image analysis for alzheimer's disease classification," *Current Medical Imaging*, 2020.

[40] A. Çinar and M. Yildirim, "Detection of tumors on brain mri images using the hybrid convolutional neural network architecture," *Medical Hypotheses*, vol. 139, p. 109684, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306987720301717

[41] N. A. Butowski, "Epidemiology and diagnosis of brain tumors," *CONTINUUM: Lifelong Learning in Neurology*, vol. 21, no. 2, pp. 301–313, 2015.

[42] R. Buerki, C. Horbinski, T. J. Kruser, P. Horowitz, C. James, and R. Lukas, "An overview of meningiomas," *Future Oncology*, vol. 14, 08 2018.

[43] M. Weller, W. Wick, K. Aldape, M. Brada, M. Berger, S. M. Pfister, R. Nishikawa, M. Rosenthal, P. Y. Wen, R. Stupp *et al.*, "Glioma," *Nature reviews Disease primers*, vol. 1, no. 1, pp. 1–18, 2015.

[44] M. Lake, L. Krook, and S. Cruz, "Pituitary adenomas: An overview," *American family physician*, vol. 88, pp. 319–27, 09 2013.

[45] G. Katti, S. Ara, and D. Shireen, "Magnetic resonance imaging (MRI) - a review," *Intl J Dental Clin*, vol. 3, 03 2011.

[46] V. Grover, J. Tognarelli, M. Crossey, I. Cox, S. Taylor-Robinson, and M. McPhail, "Magnetic resonance imaging: Principles and techniques: Lessons for clinicians," *Journal of Clinical and Experimental Hepatology*, vol. 5, 08 2015.

[47] J. Cheng, "brain tumor dataset," Apr 2017. [Online]. Available: https://figshare.com/articles/dataset/brain_tumor_dataset/1512427/5

[48] T. P. Velavan and C. G. Meyer, "The covid-19 epidemic," *Tropical Medicine & International Health*, vol. 25, no. 3, pp. 278–280, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/tmi.13383

[49] S. Weston and M. B. Frieman, "Covid-19: Knowns, unknowns, and questions," *mSphere*, vol. 5, no. 2, pp. e00 203–20, 2020. [Online]. Available: https://journals.asm.org/doi/abs/10.1128/mSphere.00203-20

[50] A. Kerpel, S. Apter, N. Nissan, E. Houri-Levi, M. Klug, S. Amit, E. Konen, and E. Marom, "Diagnostic and prognostic value of chest radiographs for covid-19 at presentation," *The western journal of emergency medicine*, 08 2020.

[51] L. Farias, E. Fonseca, D. Strabelli, B. Loureiro, Y. Neves, T. Potrich Rodrigues, R. Chate, C. Nomura, M. Sawamura, and G. Cerri, "Imaging findings in covid-19 pneumonia," *Clinics (Sao Paulo, Brazil)*, vol. 75, p. e2027, 06 2020.

[52] J. Zhou, T. Hui, C. H. Tan, H. Khoo, B. Young, D. Lye, Y. Lee, and G. Kaw, "Chest radiography in coronavirus disease 2019 (covid-19): Correlation with clinical course," *Annals of the Academy of Medicine, Singapore*, vol. 49, pp. 456–461, 07 2020.

[53] M. E. H. Chowdhury, T. Rahman, A. Khandakar, R. Mazhar, M. A. Kadir, Z. B. Mahbub, K. R. Islam, M. S. Khan, A. Iqbal, N. A. Emadi, M. B. I. Reaz, and M. T. Islam, "Can ai help in screening viral and covid-19 pneumonia?" *IEEE Access*, vol. 8, pp. 132 665–132 676, 2020.

[54] T. Rahman, A. Khandakar, Y. Qiblawey, A. Tahir, S. Kiranyaz, S. B. Abul Kashem, M. T. Islam, S. Al Maadeed, S. M. Zughaier, M. S. Khan, and M. E. Chowdhury, "Exploring the effect of image enhancement techniques on covid-19 detection using chest x-ray images," *Computers in Biology and Medicine*, vol. 132, p. 104319, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S001048252100113X

[55] T. MacGillivray, E. Trucco, J. Cameron, B. Dhillon, J. Houston, and E. Beek, "Retinal imaging as a source of biomarkers for diagnosis, characterisation and prognosis of chronic illness or long-term conditions." *The British journal of radiology*, vol. 87, p. 20130832, 06 2014.

[56] R. Tauscher, S. Simon, and N. Volpe, "Retinal disease in the neurology clinic," *Current Opinion in Neurology*, vol. Publish Ahead of Print, 12 2020.

[57] H.-G. Le and A. Shakoor, "Diabetic and retinal vascular eye disease," *Medical Clinics of North America*, vol. 105, 04 2021.

[58] S. Pachade, P. Porwal, D. Thulkar, M. Kokare, G. Deshmukh, V. Sahasrabuddhe, L. Giancardo, G. Quellec, and F. Mériaudeau, "Retinal fundus multi-disease image dataset (rfmid): A dataset for multi-disease detection research," *Data*, vol. 6, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2306-5729/6/2/14

[59] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

A Transformaciones Aplicadas e Imágenes Resultantes en el Aumento de Datos

Tabla A.1: Transformaciones aplicadas a las imágenes en el aumento de datos. Los valores delimitan el rango considerado por el algoritmo al aplicar transformaciones de carácter aleatorio.

Transformación	Tumores Cerebrales	Covid-19	Enfermedades de la Retina
Rotaton	<15º	<10º	<15º
Width Shift	0.1	0.1	0.1
Height Shift	0.1	0.1	0.1
Shear	0.05	0.05	0.05
Zoom	0.1	0.1	0.1
Horizontal Flip	True	True	True
Vertical Flip	True	False	True

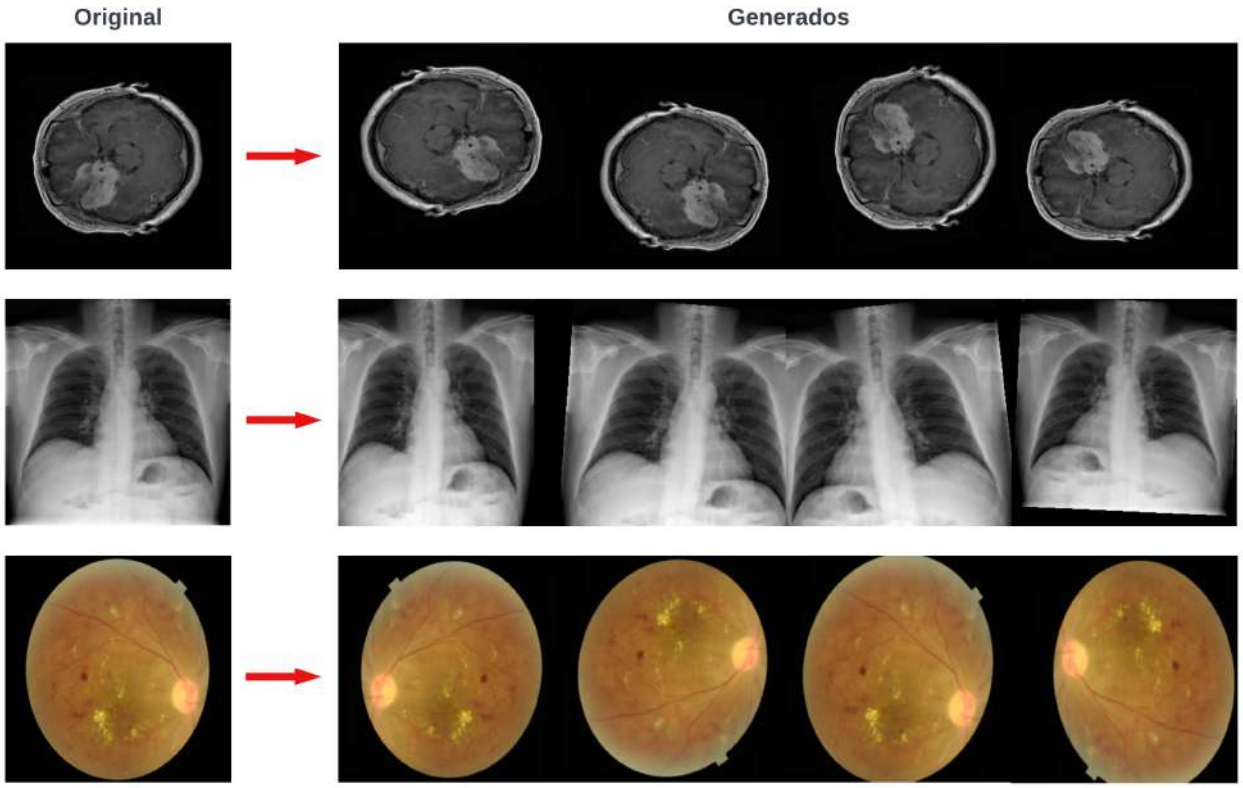


Fig. A.1: Ejemplos de imágenes obtenidas en el aumento de datos. La imagen izquierda corresponde a la imagen original, mientras que las imágenes de la derecha son las obtenidas luego del proceso.

B Anexo B - Extensión de Tablas

Tabla B.1: Extensión de resultados para el efecto del batch-size sobre el rendimiento de los modelos en la base de datos Tumores Cerebrales. Las métricas presentadas corresponden al promedio entre la totalidad de clases.

Modelo	Batch Size	Recall	Precision	F1-Score	Accuracy	Correcto	Incorrecto	Total
VGG-16	32	0.912	0.886	0.897	0.904	415	44	459
VGG-16	64	0.900	0.881	0.890	0.898	412	47	459
VGG-16	**128**	**0.922**	**0.893**	**0.906**	**0.911**	**418**	**41**	**459**
VGG-16	256	0.883	0.862	0.870	0.882	405	54	459
ResNet-50	32	0.916	0.892	0.902	0.917	421	38	459
ResNet-50	64	0.931	0.897	0.912	0.922	423	36	459
ResNet-50	128	0.929	0.903	0.914	0.926	425	34	459
ResNet-50	**256**	**0.931**	**0.904**	**0.915**	**0.928**	**426**	**33**	**459**
Inception-V3	**32**	**0.897**	**0.889**	**0.892**	**0.902**	**414**	**45**	**459**
Inception-V3	64	0.911	0.882	0.894	0.898	412	47	459
Inception-V3	128	0.896	0.870	0.882	0.887	407	52	459
Inception-V3	256	0.899	0.879	0.888	0.895	411	48	459

Tabla B.2: Extensión de resultados para el efecto del batch-size sobre el rendimiento de los modelos en la base de datos Covid-19. Las métricas presentadas corresponden al promedio entre la totalidad de clases.

Modelo	Batch Size	Recall	Precision	F1-Score	Accuracy	Correcto	Incorrecto	Total
VGG-16	32	0.960	0.970	0.965	0.974	2.952	79	3.031
VGG-16	**64**	**0.980**	**0.980**	**0.970**	**0.984**	**2.982**	**49**	**3.031**
VGG-16	128	0.967	0.962	0.965	0.973	2.948	83	3.031
VGG-16	256	0.974	0.980	0.976	0.982	2.976	55	3.031
ResNet-50	32	0.957	0.970	0.964	0.973	2.948	83	3.031
ResNet-50	64	0.973	0.966	0.970	0.978	2.964	67	3.031
ResNet-50	**128**	**0.976**	**0.974**	**0.975**	**0.982**	**2.975**	**56**	**3.031**
ResNet-50	256	0.967	0.970	0.969	0.982	2.955	76	3.031
Inception-V3	32	0.960	0.950	0.955	0.964	2.922	109	3.031
Inception-V3	64	0.955	0.962	0.959	0.968	2.933	98	3.031
Inception-V3	128	0.964	0.951	0.958	0.966	2.927	104	3.031
Inception-V3	**256**	**0.962**	**0.967**	**0.965**	**0.972**	**2.946**	**85**	**3.031**

Tabla B.3: Extensión de resultados para el efecto del batch-size sobre el rendimiento de los modelos en la base de datos Enfermedades de la Retina. Las métricas presentadas corresponden al promedio entre la totalidad de clases.

Modelo	Batch Size	Recall	Precision	F1-Score	Accuracy	Correcto	Incorrecto	Total
VGG-16	32	0.857	0.844	0.850	0.898	575	65	640
VGG-16	64	0.872	0.829	0.848	0.892	571	69	640
VGG-16	128	0.883	0.829	0.851	0.892	571	69	640
VGG-16	**256**	**0.891**	**0.888**	**0.890**	**0.927**	**593**	**47**	**640**
ResNet-50	32	0.796	0.866	0.824	0.894	572	68	640
ResNet-50	**64**	**0.877**	**0.858**	**0.867**	**0.909**	**582**	**58**	**640**
ResNet-50	128	0.883	0.843	0.860	0.902	577	63	640
ResNet-50	256	0.857	0.863	0.860	0.908	581	59	640
Inception-V3	32	0.868	0.794	0.819	0.864	553	87	640
Inception-V3	64	0.801	0.861	0.826	0.894	572	68	640
Inception-V3	**128**	**0.866**	**0.876**	**0.871**	**0.916**	**586**	**54**	**640**
Inception-V3	256	0.845	0.854	0.849	0.902	577	63	540

Tabla B.4: Extensión de resultados para el efecto del learning-rate sobre el rendimiento de los modelos en cada base de datos. Las métricas presentadas corresponden al promedio entre la totalidad de clases.

Base de Datos	Modelo	LR	Recall	Precision	F1-Score	Accuracy	Correcto	Incorrecto	Total
TC	ResNet-50	1e-1	0.333	0.185	0.238	0.556	255	204	459
TC	ResNet-50	1e-2	0.901	0.884	0.889	0.906	416	43	459
TC	**ResNet-50**	**1e-3**	**0.931**	**0.904**	**0.915**	**0.928**	**426**	**33**	**459**
TC	ResNet-50	1e-4	0.896	0.849	0.867	0.882	405	54	459
TC	ResNet-50	1e-5	0.908	0.894	0.899	0.911	418	41	459
C19	VGG-16	1e-1	0.341	0.539	0.284	0.673	2.021	1.010	3.031
C19	VGG-16	1e-2	0.946	0.971	0.958	0.966	2.929	102	3.031
C19	**VGG-16**	**1e-3**	**0.980**	**0.980**	**0.980**	**0.984**	**2.982**	**49**	**3.031**
C19	VGG-16	1e-4	0.979	0.965	0.972	0.979	2.966	65	3.031
C19	VGG-16	1e-5	0.981	0.971	0.976	0.980	2.970	61	3.031
ER	VGG-16	1e-1	0.500	0.396	0.442	0.791	506	134	640
ER	VGG-16	1e-2	0.751	0.881	0.793	0.884	566	74	640
ER	**VGG-16**	**1e-3**	**0.891**	**0.888**	**0.890**	**0.927**	**593**	**47**	**640**
ER	VGG-16	1e-4	0.877	0.841	0.857	0.900	576	64	640
ER	VGG-16	1e-5	0.869	0.823	0.842	0.888	568	72	640

Donde TC: Tumores cerebrales; C19: Covid-19; ER: Enfermedades de la retina; LR: learning-rate.

Tabla B.5: Extensión de resultados para el efecto del aumento de datos sobre el rendimiento de los modelos en cada base de datos. Se exponen las métricas obtenidas por clase y por promedio.

Tumores Cerebrales													
Clase	P-a	P-b	R-a	R-b	F-a	F-b	Acc-a	Acc-b	C-a	C-b	I-a	I-b	T
0	0.862	0.875	0.855	0.778	0.858	0.823	0.928	0.906	100	91	17	26	117
1	0.998	0.948	0.937	0.933	0.962	0.941	0.928	0.906	239	238	16	17	255
2	0.861	0.837	1	1	0.926	0.911	0.928	0.906	87	87	0	0	87
P/T	0.904	0.887	0.931	0.904	0.915	0.892	0.928	0.906	426	416	33	43	459
Covid-19													
Clase	P-a	P-b	R-a	R-b	F-a	F-b	Acc-a	Acc-b	C-a	C-b	I-a	I-b	T
0	0.989	0.922	0.989	0.994	0.989	0.956	0.984	0.939	1.999	2.009	23	13	2.022
1	0.978	0.987	0.978	0.964	0.978	0.922	0.984	0.939	266	235	6	37	272
2	0.973	0.982	0.973	0.818	0.966	0.893	0.984	0.939	717	603	20	134	737
P/T	0.980	0.964	0.980	0.925	0.970	0.924	0.984	0.939	2.982	2.847	49	184	3.031
Enfermedades de la Retina													
Clase	P-a	P-b	R-a	R-b	F-a	F-b	Acc-a	Acc-b	C-a	C-b	I-a	I-b	T
0	0.822	0.757	0.828	0.858	0.825	0.804	0.927	0.913	111	115	23	19	134
1	0.954	0.961	0.953	0.927	0.954	0.927	0.927	0.913	482	469	24	37	506
P/T	0.888	0.859	0.891	0.893	0.890	0.866	0.927	0.913	593	584	47	56	640

Donde P: precision; R: recall; F: F1-score; Acc: accuracy; C: imágenes clasificadas correctamente; I: imágenes clasificadas incorrectamente; T: cantidad total de imágenes; P/T: promedio o total; a: antes del aumento de datos; b: posterior al aumento de datos.

Tabla B.6: Extensión de resultados para el efecto del fine-tuning sobre el rendimiento de los modelos en cada base de datos. Se exponen las métricas obtenidas por cada clase y por promedio.

Tumores Cerebrales													
Clase	P-a	P-b	R-a	R-b	F-a	F-b	Acc-a	Acc-b	C-a	C-b	I-a	I-b	T
0	0.862	0.926	0.855	0.855	0.858	0.889	0.928	0.946	100	100	17	17	117
1	0.998	0.972	0.937	0.969	0.962	0.971	0.928	0.946	239	247	16	8	255
2	0.861	0.897	1	1	0.926	0.946	0.928	0.946	87	87	0	0	87
P/T	0.904	0.932	0.931	0.941	0.915	0.935	0.928	0.946	426	434	33	25	459
Covid-19													
Clase	P-a	P-b	R-a	R-b	F-a	F-b	Acc-a	Acc-b	C-a	C-b	I-a	I-b	T
0	0.989	0.990	0.989	0.994	0.989	0.992	0.984	0.988	1.999	2.010	23	12	2.022
1	0.978	0.982	0.978	0.978	0.978	0.980	0.984	0.988	266	266	6	6	272
2	0.973	0.985	0.973	0.974	0.973	0.980	0.984	0.988	717	718	20	19	737
P/T	0.980	0.986	0.980	0.982	0.980	0.984	0.984	0.988	2.982	2.994	49	37	3.031
Enfermedades de la Retina													
Clase	P-a	P-b	R-a	R-b	F-a	F-b	Acc-a	Acc-b	C-a	C-b	I-a	I-b	T
0	0.822	0.733	0.828	0.881	0.825	0.800	0.927	0.908	111	118	23	16	134
1	0.954	0.967	0.953	0.915	0.954	0.940	0.927	0.908	482	463	24	43	506
P/T	0.888	0.850	0.891	0.898	0.890	0.870	0.927	0.908	593	581	47	59	540

Donde P: precision; R: recall; F: F1-score; Acc: accuracy; C: imágenes clasificadas correctamente; I: imágenes clasificadas incorrectamente; T: cantidad total de imágenes; P/T: promedio o total; a: antes del aumento de datos; b: posterior al aumento de datos.

TUTORIAL PARA LA CLASIFICACIÓN DE COVID-19

ÍNDICE DE CONTENIDOS

---

1. INTRODUCCIÓN.

En este tutorial se cubren las etapas necesarias para la clasificación de imágenes médicas utilizando Redes Neuronales Convolucionales (CNN). Para ello, se utilizará una base de datos pública de imágenes para la patología Covid-19 y el modelo VGG-16.

En primer lugar se hace una descripción de las bibliotecas necesarias para la ejecución del código. Luego, se hace una descripción de la base de datos a utilizar y se ejemplifican algunas imágenes disponibles en ella. Luego, se explica cómo realizar la carga de imágenes a la plataforma y generar los distintos conjuntos de datos. Además, se cubre la configuración de los hiperparámetros principales, es decir, batch-size, learning-rate, optimizador y función de pérdida, junto a la implementación de funciones auxiliares llamadas callbacks. Por último, se instruye en cómo llevar a cabo el entrenamiento del modelo, su evaluación de rendimiento y cómo mejorarlo a través de la técnica fine-tuning. Por último, se revisa la obtención de métricas y evaluación final.

Dado que se trabaja con una gran cantidad de imágenes y los modelos de redes neuronales requieren de un alto costo computacional, se hará uso de la GPU integrada en Colab. Para su activación, seguir los siguientes pasos:

Editar -> Configuración del cuaderno -> Acelerador por hardware: GPU -> Guardar.

---

2. LISTADO E IMPORTACIÓN DE BIBLIOTECAS.

Las bibliotecas implementadas en este tutorial están integradas en la plataforma Colab. Por lo mismo, no requieren su instalación, pero sí deben ser importadas.

A continuación se detallan las bibliotecas a utilizar y con qué propósito.

- **Tensorflow** y **Keras** para:
  - Carga de imágenes
  - Generación de conjuntos de datos
  - Generación de modelos
  - Capas
  - Callbacks
- **Numpy** y **Pandas** para:
  - Trabajo sobre datos y generación de DataFrames.
- **Matplotlib** y **Seaborn** para:
  - Generación y visualización de figuras.
- **Scikit-learn** para:
  - Obtención de métricas (reporte de clasificación)
  - Obtención de matriz de confusión
- **ZipFile** y **OS** para:
  - Descomprensión de base de datos (.zip)
  - Navegación en directorios

Al ejecutar la siguiente celda, todas las bibliotecas mencionadas serán importadas.

```
[ ]
    # Tensorflow y Keras:
    import tensorflow as tf

    # para trabajo sobre imágenes:
    from tensorflow.keras.preprocessing.image import load_img, img_to_array, ImageDataGenerator

    # para modificación de arquitectura del modelo:
    from tensorflow.keras.models import Model
    from tensorflow.keras.layers import Dense, Flatten, Dropout, GlobalAveragePooling2D

    # Callbacks:
    from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

    # Modelo VGG-16 y función para preprocesar las imágenes:
    from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input

    # Reporte de clasificación y mátrices de confusión
    from sklearn.metrics import classification_report, confusion_matrix

    # Generación y visualización de figuras:
    import matplotlib.pyplot as plt
    import seaborn as sns

    # Trabajo sobre datos:
    import numpy as np
    import pandas as pd

    # Otras:
    import os # navegación por directorios
    import zipfile # extracción de archivo con imágenes
    from google.colab import drive # montar Google Drive
    from datetime import datetime # registrar tiempos de ejecución
```

---

3. DESCRIPCIÓN DE BASE DE DATOS.

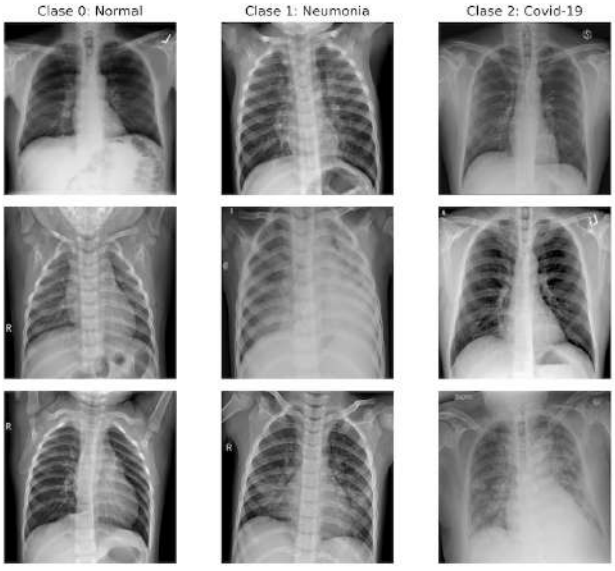
La base de datos a utilizar es sobre la patología Covid-19. Cuenta con un total de 15.153 imágenes de radiografía de tórax separadas en clases de la siguiente forma:

- Normal, con 10.192 imágenes.
- Neumonía, con 1.345 imágenes.
- Covid, con 3.616 imágenes.

Las imágenes poseen dimensión 299x299 píxeles, su formato es .PNG y están en escala de grises.

Las imágenes y archivos necesarios para la ejecucicón de este Notebook, se encuentran disponibles en el siguiente enlace: COVID-19

A continuación se muestran tres imágenes por clase.



Imágenes obtenidas de [1]-[2].

---

4. CARGA DE IMÁGENES

La carga de imágenes se puede realizar directamente a Google Colab, o se puede montar la cuenta de Google Drive.

Para la primera opción, seleccionar "Archivos" en el panel de la izquierda y luego "Subir al almacenamiento de sesión".

Para la segunda opción, que es considerada en este caso, seleccionar "Montar Drive" o ejecutar la siguiente celda. Se solicitará la autenticación en la cuenta de Drive.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

Una vez Drive haya sido vinculado, navegamos por el panel de la izquierda y buscamos el archivo "images.zip" que contiene las imágenes y, con el click derecho, seleccionamos "copiar ruta".

Luego, ejecutamos la siguiente celda para descomprimir el archivo en la plataforma, haciendo uso de la biblioteca Zipfile.

```
[ ] # seleccionamos la ruta y leemos el archivo
    unzip = zipfile.ZipFile('/content/drive/MyDrive/Tutoriales/COVID-19/images.zip', 'r')
    # descomprimimos las imágenes en el directorio deseado. En este caso: '/content/data'.
    unzip.extractall('/content/data')
    unzip.close()
```

Nuevamente en el panel de la izquierda, comprobamos que las imágenes están en el directorio '/content/data/images/'.

---

5. GENERACIÓN DE CONJUNTOS DE DATOS.

Ahora procedemos a la generación de los conjuntos de datos correspondientes a entrenamiento, validación y prueba.

En este caso, obtendremos la clase a la que corresponde cada imagen a partir del nombre del archivo.

Al revisar los archivos descomprimidos, podemos ver la forma en que cada uno está nombrado:

- La clase Normal comienza con "Normal" y tendrá la etiqueta "0".
- La clase Neumonía comienza con "Viral" y tendrá la etiqueta "1".
- La clase Covid comienza con "COVID" y tendrá la etiqueta "2".

Haciendo uso de esta información, agregamos los nombres de los archivos a una lista y su respectiva clase a otra. Luego, construimos un DataFrame con esta información haciendo uso de la biblioteca Pandas.

```
[ ] # creamos una lista llamada "filenames" que contiene los nombres de cada archivo
    filenames = [x for x in os.listdir('/content/data/images') if not x.startswith('.')]

    # creamos lista llamada "labels" con las etiquetas o clases de cada imagen
    labels = []
    for filename in filenames:
      if filename.startswith('Normal'): labels.append('0')  # si comienza con "Normal" -> Clase 0
      elif filename.startswith('Viral'): labels.append('1') # si comienza con "Viral" -> Clase 1
      else: labels.append('2')                              # si comienza con "COVID" -> Clase 2

    # generamos un DataFrame con esta información
    df = pd.DataFrame({'filename': filenames, 'label': labels})
```

De esta forma, obtenemos la información de cada imagen con su respectiva clase en un DataFrame. Ahora, para efectos de reproducibilidad, cargaremos y utilizaremos el mismo DataFrame que fue generado en el estudio previo haciendo uso de la biblioteca Pandas y que se encuentra disponible en el enlace de la *sección 3*.

```
[ ] # cargamos el archivo
    df = pd.read_csv('/content/drive/MyDrive/Tutoriales/COVID-19/covid_dataframe.csv')

    # convertimos las clases a tipo string (requerido más adelante)
    df.label = [str(x) for x in df.label]
```

Luego, a partir del DataFrame obtenido, generamos nuevos DataFrames con la información por cada conjunto.

- Para entrenamiento consideramos el 60% de los datos.
- Para validación consideramos el 20% de los datos.
- Para prueba consideramos el 20% de los datos.

Y comprobamos que la cantidad total de imágenes coincida con la suma de imágenes en cada conjunto.

```
[ ]  # conjunto de entrenamiento con el 60% de los datos
     df_train = df[0: int(len(df)*0.6)]

     # conjunto de validación con el 20% de los datos
     df_val = df[int(len(df)*0.6): int(len(df)*0.8)]

     # conjunto de validación con el 20% de los datos
     df_test = df[int(len(df)*0.8)::]

     # comprobamos que la cantidad de imágenes en la unión de los conjuntos
     # coincida con el total de imágenes: 15153.
     print("Total de imágenes: ", len(df_train) + len(df_val) + len(df_test))
```

```
Total de imágenes:  15153
```

Con el objetivo de cargar las imágenes en grupos para no tener problemas relacionados a límites de memoria, utilizaremos funciones "generadoras" de imágenes de la biblioteca Keras.

Para ello, utilizamos las siguientes funciones:

- ImageDataGenerator, que aplica el preprocesamiento a las imágenes junto a la función *preprocess_input*. Este preprocesamiento es idéntico al que fue aplicado al momento de entrenar el modelo VGG-16 en su construcción original. Los canales RGB son pasados a BGR y los valores de píxeles son centrados en cero. Además, esta función permite realizar aumento de datos. Algunas operaciones disponibles son: rotación, traslación en ejes horizontal y vertical, flips horizontal y vertical, zoom, entra otras.
- Flow_from_dataframe, que carga las imágenes a partir de los DataFrames construidos y el directorio donde se encuentran. Los siguientes parámetros son sus principales:
    - **dataframe**: el DataFrame respectivo al conjunto.
    - **directory**: ruta con las imágenes.
    - **class_mode**: tipo de clasificación.
    - **batch_size**: cantidad de imágenes por grupo.
    - **target_size**: dimensión de imágenes resultantes.

Primero, creamos un generador de datos por conjunto que se encargará de aplicar el preprocesamiento a las imágenes, haciendo uso de *ImageDataGenerator* y *preprocess_input*.

Además, para el conjunto de entrenamiento, en el caso de que deseemos aplicar aumento de datos, indicamos en el generador para entrenamiento (*train_datagen* en la siguiente celda) las transformaciones, considerando los siguientes parámetros y, por ejemplo, los siguientes valores:

- **rotation_range:** rotación máxima, 10º.
- **width_shift_range:** traslación en eje horizontal, 0.1.
- **height_shift_range:** traslación en eje vertical, 0.1.
- **shear_range:** deformación (similar a esfuerzo cortante), 0.05.
- **zoom_range:** acercamiento, 0.1.
- **horizontal_flip:** reflejo horizontal, "True".
- **vertical_flip:** reflejo vertical, "False".
- **fill_mode:** relleno de píxeles vacíos producto de las operaciones, 'constant'.
- **c_val:** valor para 'constant' en fill_mode. Seleccionamos 0, que indica valores de píxeles iguales a cero.
- **seed:** semilla para efectos de reproducibilidad, indicamos "27" o cualquier otro número.

Sin embargo, según el estudio realizado, el aumento de datos no otorgó beneficios en el rendimiento de la clasificación, por lo que en esta ocasión será omitido.

Es importante notar que el aumento de datos, en caso de ser aplicado, debe ser solo al conjunto de entrenamiento. Por este motivo, se debe tener cuidado al construir los conjuntos generadores de datos.

A continuación configuramos el preprocesamiento de datos para cada conjunto, haciendo uso de *ImageDataGenerator*.

```
[ ]  train_datagen = ImageDataGenerator(preprocessing_function = preprocess_input)
     val_datagen = ImageDataGenerator(preprocessing_function = preprocess_input)
     test_datagen = ImageDataGenerator(preprocessing_function = preprocess_input)
```

Luego, a partir del DataFrame respectivo a cada conjunto, creamos los generadores para entrenamiento, validación y prueba haciendo uso de la función *flow_from_dataframe* y las variables construidas en el paso anterior.

- Generador para entrenamiento: Indicamos el dataframe, la ruta a las imágenes, la columna con los nombres de archivo, la columna con las clases, el tipo de clasificación, la dimensión de destino de las imágenes, el tamaño de batch (en este caso 64), y la aleatoriedad al considerar los datos.

```
[ ]  # construimos el generador para el conjunto de entrenamiento.

     train_generator = train_datagen.flow_from_dataframe(dataframe = df_train, # DataFrame para entrenamiento
                                                         directory = '/content/data/images/', # ruta a imágenes
                                                         x_col = 'filename', # columna con el nombre de archivos
                                                         y_col = 'label', # columna con etiquetas o clases
                                                         class_mode = 'categorical', # tipo de clasificación
                                                         target_size = (224, 224), # dimensión de imagen resultante
                                                         batch_size = 64, # tamaño de batch
                                                         shuffle = True, # considera imágenes de forma aleatoria
                                                         seed = 27) # semilla para efectos de reproducibilidad
```

```
Found 9091 validated image filenames belonging to 3 classes.
```

El output obtenido nos indica que las imágenes consideradas por el generador de entrenamiento han sido encontradas satisfactoriamente.

Ahora, procedemos de forma análoga para el conjunto de validación, pero cambiando el dataframe.

```
[ ]  # construimos el generador para el conjunto de validación
     val_generator = val_datagen.flow_from_dataframe(dataframe = df_val,
                                                     directory = '/content/data/images/',
                                                     x_col = 'filename',
                                                     y_col = 'label',
                                                     target_size = (224, 224),
                                                     class_mode = 'categorical',
                                                     batch_size = 64,
                                                     shuffle = True,
                                                     seed = 27)
```

```
Found 3031 validated image filenames belonging to 3 classes.
```

Nuevamente, obtenemos confirmación de la construcción satisfactoria del generador.

Por último, realizamos el proceso una vez más para el generador de prueba. Esta vez configuramos *shuffle* como False, para obtener siempre el mismo orden de imágenes en el conjunto de prueba.

```
[ ]  # construimos el generador para el conjunto de prueba
     test_generator = test_datagen.flow_from_dataframe(dataframe = df_test,
                                                       directory = '/content/data/images/',
                                                       x_col = 'filename',
                                                       y_col = 'label',
                                                       target_size = (224, 224),
                                                       class_mode = 'categorical',
                                                       batch_size = 64,
                                                       shuffle = False)
```

```
Found 3031 validated image filenames belonging to 3 classes.
```

Y comprobamos, una vez más, la correcta construcción del generador para el conjunto de prueba.

---

6. CONFIGURACIÓN DE CALLBACKS.

A continuación, se presentan tres funciones auxiliares que permiten un entrenamiento óptimo del modelo:

- **Early-Stopping**: Permite detener el entrenamiento en caso de verse perjudicado por fenómenos de sobreajuste o si es que no hay mejoras en el rendimiento. Sus principales parámetros son:
  - **monitor:** métrica a monitorear. Seleccionamos "val_loss", valor de pérdida para el conjunto de validación.
  - **min_delta:** mejora mínima requerida para continuar el entrenamiento. Seleccionamos un valor decimal, en este caso "0.0001" ya que necesitamos que el valor mejore, aunque esta mejora sea mínima.
  - **patience:** "paciencia", cantidad de épocas considerada antes de detener el modelo en caso de que no existan mejoras en la métrica monitoreada. Seleccionamos 10.
  - **restore_best_weights:** permite restaurar los mejores pesos en caso de que el entrenamiento sea perjudicado. Seleccionamos "True".

Con la configuración indicada, el callback actuará de la siguiente manera: si en el entrenamiento del modelo, por 10 épocas no hay una mejora mínima de 0.0001 en la pérdida del conjunto de validación, el entrenamiento se detendrá y los mejores pesos serán restaurados.

```
[ ]  # Configuración de callback "Early-Stopping"

     # asignamos el callback a la variable "es"
     es = tf.keras.callbacks.EarlyStopping(
         monitor = "val_loss", # métrica monitoreada: pérdida del conjunto de validación
         min_delta = 0.0001,   # mejora mínima requerida: 0.0001
         patience = 10,        # paciencia: 10 (epochs)
         restore_best_weights = True) # restaura los pesos óptimos
```

- **ReduceLROnPlateau:** Permite reducir el learning-rate si el modelo no está aprendiendo correctamente. Funciona de forma análoga a Early-Stopping. Sus principales parámetros son mencionados a continuación.
  - **monitor:** métrica a monitorear. Seleccionamos "val_loss".
  - **min_delta:** mejora mínima requerida para mantener el learning-rate. Seleccionamos "0.001". Esta vez el valor mínimo requerido es mayor que en Early-Stopping. De esta forma nos aseguramos que el callback actúe.
  - **patience:** "paciencia", cantidad de épocas considerada antes de disminuir el learning-rate en caso de que no existan mejoras en la métrica monitoreada. Seleccionamos 5. Nuevamente, considerando este valor menor que en Early-Stopping, nos aseguramos que el callback actúe.
  - **factor:** factor de reducción de learning-rate. Seleccionamos 0.1.
  - **min_lr:** mínimo learning-rate posible. Seleccionamos 1e-7.

Con esta configuración, el callback ReduceLROnPLateau actuará de la siguiente manera: si en 5 épocas no hay una mejora de 0.001 puntos en la pérdida del conjunto de validación, el learning-rate será reducido en un factor de 0.1.

```
[ ]  # Configuración de callback "ReduceLROnPlateau"

     reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
         monitor = "val_loss", # métrica monitoreada
         factor = 0.1, # factor de decrecimiento de learning-rate
         patience = 5, # factor de paciencia
         min_delta = 0.001, # mejora mínima requerida
         min_lr = 1e-7) # mínimo learning-rate posible
```

- **ModelCheckpoint:** Permite guardar el entrenamiento del modelo en caso de verse interrumpido. Sus principales parámetros son mencionados a continuación.
  - **filepath:** ruta donde el checkpoint será almacenado.
  - **monitor:** métrica monitoreada. Seleccionamos "val_loss".
  - **save_best_only:** guarda el entrenamiento cada vez que hay mejoras en la métrica monitoreada. En este caso lo configuramos en "True"
  - **save_weights_only:** permite guardar solo los pesos en vez del modelo completo. En este caso lo configuramos en "False" y guardaremos el modelo completo.

Con esta configuración el callback ModelCheckpoint actuará de la siguiente manera: si al final de cada época existió una mejora en la pérdida del conjunto de validación, el modelo será guardado como checkpoint en la ruta especificada.

```
[ ]  # Configuración de callback "ModelCheckpoint"

     mc = tf.keras.callbacks.ModelCheckpoint(
         filepath = '/content/drive/MyDrive/Tutoriales/COVID-19/checkpoint.ckpt', # ruta donde será guardado el checkpoint
         monitor = "val_loss", # métrica monitoreada: pérdida del conjunto de validación
         save_best_only = True, # guarda solamente los mejores pesos
         save_weights_only = False, # guarda el modelo completo
         save_freq = "epoch") # guarda cada época si existe mejora en la métrica monitoreada
```

---

7. CARGA DE MODELO PRE-ENTRENADO Y MODIFICACIÓN PARA LA TAREA ESPECÍFICA DE CLASIFICACIÓN.

El modelo a utilizar en este caso es VGG-16.

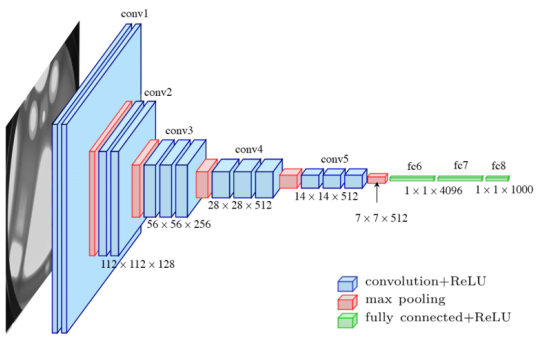


Imagen obtenida de [3].

Para aplicar transferencia de aprendizaje, que consiste en la utilización de pesos que originalmente fueron obtenidos en una tarea de clasificación distinta a la que se desea resolver, tendremos en consideración lo siguiente.

Implementamos el modelo VGG-16 con los siguientes parámetros principales:

- **include_top:** indica si incluir o no las capas finales para la clasificación original. En este caso lo configuramos en "False".
- **weights:** permite cargar pesos pre-entrenados, ya sea los de ImageNet u otros obtenidos en otros trabajos. Seleccionamos "imagenet".
- **input_shape:** indica la dimensión de imagen de entrada del modelo. Para poder utilizar los pesos de imagenet, seleccionamos la dimensión original: (224, 224, 3) -> (alto, ancho, número de canales)

```
[ ]  # cargamos el modelo VGG-16
     model = VGG16(include_top = False,  # no considera las capas de clasificación original
                   weights = 'imagenet', # pesos pre-entrenados en ImageNet
                   input_shape = (224, 224, 3)) # dimensión de imagen de entrada
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_orderi
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

Una vez que el modelo haya sido descargado y asignado a la variable *model*, alteramos la configuración de sus capas, inhibiendo el entrenamiento de los pesos y así mantendremos los de imagenet.

```
[ ]  # accedemos a cada capa del modelo:
     for layer in model.layers:
       layer.trainable = False # inhibimos el entrenamiento
```

Ahora, agregamos capas al final del modelo. Estas capas permitirán realizar la clasificación en tres clases. Siguiendo la arquitectura original de VGG-16, añadimos lo siguiente:

- Agregamos una operación de aplanamiento.
- Agregamos una capa densa con 4096 nodos y función de activación "ReLU".
- Agregamos Dropout con factor 50%.
- Agregamos una capa densa y Dropout igual a los anteriores.
- Luego, generamos la salida de nuestro modelo. Esta última capa es también una capa densa con una cantidad de nodos igual a la cantidad de clases disponibles. En esta última capa, la función de activación dependerá del tipo de clasificación. Como en este caso se desea obtener una clasificación multiclase, seleccionamos la función "softmax".
- Finalmente, unimos el modelo VGG-16 con las nuevas capas generadas.

```
[ ]  # agregamos una operación de aplanamiento
     # que va unida al final del modelo pre-entrenado
     flatten1 = Flatten()(model.layers[-1].output)

     # agregamos una capa densa con 4096 nodos, y función de activación "ReLU".
     dense1 = Dense(4096, activation='relu')(flatten1)

     # agregamos Dropout con un factor del 50%.
     dropout1 = Dropout(0.50)(dense1)

    # agregamos Dropout con un factor del 50%.
    dropout1 = Dropout(0.50)(dense1)

    # agregamos una nueva capa densa y dropout con las mismas configuraciones
    # que las anteriores.
    dense2 = Dense(4096, activation = 'relu')(dropout1)
    dropout2 = Dropout(0.50)(dense2)

    # generamos la salida de la red, que es una capa densa con tres nodos y
    # función de activación "softmax".
    output =  Dense(3, activation = 'softmax')(dropout2)

    # unimos el modelo VGG-16 pre-entrenado con las nuevas capas generadas
    # para ello indicamos como input el modelo original, y como output
    # las capas generadas.
    model = Model(inputs = model.inputs, outputs = output)
```

Ejecutando la siguiente celda, podemos inspeccionar en detalle la arquitectura del modelo generado y observar la cantidad de parámetros que entrega cada capa.

```
[ ] model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 4096)              102764544

 dropout (Dropout)           (None, 4096)              0

 dense_1 (Dense)             (None, 4096)              16781312

 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 3)                 12291

=================================================================
Total params: 134,272,835
Trainable params: 119,558,147
Non-trainable params: 14,714,688
_________________________________________________________________
```

8. COMPILACIÓN Y ENTRENAMIENTO DEL MODELO.

Cuando el modelo a utilizar ya esté definido, procedemos a su compilación. Para compilar el modelo, debemos indicar dos hiperparámetros: optimizador y función de pérdida.

El optimizador a utilizar es *Adam*, mientras que la función de pérdida es *Categorical Cross-entropy*, debido a que es una clasificación multiclase.

- El optimizador se encarga de la actualización de los pesos y kernels de convolución presentes en la red. Este hiperparámetro incluye el valor de *learning-rate*, que configuramos en *1e-3*, ya que el estudio realizado indica que es el óptimo.
- La función de pérdida por otra parte, permite calcular el error y guiar al optimizador en la actualización de pesos en la dirección correcta.
- Por último, configuramos *accuracy* como métrica de monitoreo.

```
[ ] # configuramos los hiperparámetros
    Adam = tf.keras.optimizers.Adam(learning_rate = 1e-3) # optimizador y learning-rate

    # compilamos el modelo
    model.compile(optimizer = Adam, # optimizador
                  loss = 'categorical_crossentropy', # función de pérdida
                  metrics = ['accuracy']) # métrica de monitoreo
```

Ahora, procedemos al entrenamiento del modelo. Para ello, hacemos uso del atributo *fit*, que recibe como parámetros principales lo siguiente:

- **x:** conjunto de entrenamiento. En este caso es "train_generator", el generador para el conjunto de entrenamiento obtenido en la sección 5.
- **epochs:** número máximo de épocas por las que el modelo será entrenado. Seleccionamos 100. En este hiperparámetro, se prioriza la acción de Early-Stopping.
- **validation_data:** conjunto de validación. En este caso es "val_generator", el generador para el conjunto de validación obtenido en la *sección 5*.
- **callbacks:** funciones auxiliares callbacks definidas en la *sección 6*. Recibe una lista con los callbacks.
- **steps_per_epoch:** permite determinar cuándo termina una época. Este parámetro generalmente no es considerado, pero en caso de considerar el aumento de datos, debemos asignarlo. El valor debe ser igual a (# datos entrenamiento / batch-size), en este caso, el valor es 143.

Nota: El batch-size no debe ser especificado, ya que fue definido en la construcción de los generadores de imágenes en la *sección 5*.

Por último, es útil obtener un registro del tiempo de ejecución del entrenamiento. Para ello, hacemos uso de la biblioteca *datetime*.

```
[ ] # obtenemos el tiempo de inicio
    inicio = datetime.now()

    # asignamos el entrenamiento a una variable para almacenar información que utilizaremos más adelante
    # y entrenamos el modelo con el atributo fit.
    history = model.fit(x = train_generator, # generador del conjunto de entrenamiento
                        epochs = 100, # epochs
                        validation_data = val_generator, # conjunto de validación
                        callbacks = [es, mc, reduce_lr]) # callbacks

    # obtenemos el tiempo al terminar el entrenamiento
    fin = datetime.now()
    # calculamos el tiempo de ejecución y lo desplegamos en pantalla
    tiempo_de_ejecucion = fin - inicio
    print("tiempo de ejecución: ", tiempo_de_ejecucion)
```

```
Epoch 1/100
143/143 [==============================] - ETA: 0s - loss: 8.0817 - accuracy: 0.8858INFO:tensorflow:Assets written to: /content/drive/MyDrive/Tutoriales/COVID-19/checkpoint.ckpt/assets
143/143 [==============================] - 93s 553ms/step - loss: 8.0817 - accuracy: 0.8858 - val_loss: 0.2275 - val_accuracy: 0.9588 - lr: 0.0010
Epoch 2/100
143/143 [==============================] - ETA: 0s - loss: 0.3940 - accuracy: 0.9452INFO:tensorflow:Assets written to: /content/drive/MyDrive/Tutoriales/COVID-19/checkpoint.ckpt/assets
143/143 [==============================] - 79s 550ms/step - loss: 0.3940 - accuracy: 0.9452 - val_loss: 0.2155 - val_accuracy: 0.9528 - lr: 0.0010
Epoch 3/100
143/143 [==============================] - 75s 521ms/step - loss: 0.5538 - accuracy: 0.9473 - val_loss: 0.2407 - val_accuracy: 0.9594 - lr: 0.0010
Epoch 4/100
143/143 [==============================] - 71s 493ms/step - loss: 0.3838 - accuracy: 0.9577 - val_loss: 0.2840 - val_accuracy: 0.9597 - lr: 0.0010
Epoch 5/100
143/143 [==============================] - 71s 493ms/step - loss: 0.3491 - accuracy: 0.9615 - val_loss: 0.3588 - val_accuracy: 0.9604 - lr: 0.0010
Epoch 6/100
143/143 [==============================] - 71s 495ms/step - loss: 0.4072 - accuracy: 0.9606 - val_loss: 0.2685 - val_accuracy: 0.9703 - lr: 0.0010
Epoch 7/100
143/143 [==============================] - 71s 494ms/step - loss: 0.3761 - accuracy: 0.9672 - val_loss: 0.7553 - val_accuracy: 0.9644 - lr: 0.0010
Epoch 8/100
143/143 [==============================] - ETA: 0s - loss: 0.2030 - accuracy: 0.9813INFO:tensorflow:Assets written to: /content/drive/MyDrive/Tutoriales/COVID-19/checkpoint.ckpt/assets
143/143 [==============================] - 80s 558ms/step - loss: 0.2030 - accuracy: 0.9813 - val_loss: 0.2035 - val_accuracy: 0.9739 - lr: 1.0000e-04
Epoch 9/100
143/143 [==============================] - ETA: 0s - loss: 0.0945 - accuracy: 0.9887INFO:tensorflow:Assets written to: /content/drive/MyDrive/Tutoriales/COVID-19/checkpoint.ckpt/assets
143/143 [==============================] - 84s 586ms/step - loss: 0.0945 - accuracy: 0.9887 - val_loss: 0.1679 - val_accuracy: 0.9769 - lr: 1.0000e-04
Epoch 10/100
143/143 [==============================] - ETA: 0s - loss: 0.0783 - accuracy: 0.9888INFO:tensorflow:Assets written to: /content/drive/MyDrive/Tutoriales/COVID-19/checkpoint.ckpt/assets
143/143 [==============================] - 86s 605ms/step - loss: 0.0783 - accuracy: 0.9888 - val_loss: 0.1443 - val_accuracy: 0.9766 - lr: 1.0000e-04
```

La pantalla permite visualizar el proceso de entrenamiento.

- Se indica el epoch actual y el tiempo que tarda en cada uno.
- Se pueden ver las métricas *loss* y *accuracy* para el conjunto de entrenamiento y *val_loss* y *val_accuracy* para el conjunto de validación.
- Se puede apreciar el valor de learning-rate considerado en cada época.
- Respecto a la acción de los callbacks:
    - ModelCheckpoint indica que el modelo ha sido guardado en la ruta seleccionada.
    - ReduceLROnPlateau actuó en la época número 8 (5 épocas luego de la última mejora registrada en *val_loss*), reduciendo el learning-rate de 1e-3 a 1e-4. Luego volvió a actuar en las épocas 16, 32 y 37.
    - Por último, luego de 10 épocas sin mejoras en *val_loss*, Early-Stopping detuvo el entrenamiento.

Una forma de monitorear el entrenamiento, es poniendo atención a las métricas descritas. Ambas pérdidas deben disminuir cada cierto tiempo, y ambas exactitudes (accuracy) deben aumentar.

9. FINE-TUNING.

Una vez que hayamos entrenado el modelo, podemos aplicar la técnica de *Fine-tuning*. Esta técnica consiste en habilitar el entrenamiento en algunas capas (o la totalidad de ellas) que habían sido "congeladas" para mantener los pesos. El objetivo es *afinar* estos pesos re-entrenando el modelo por una cantidad pequeña de épocas y utilizando un learning-rate reducido.

Para ello, habilitamos el entrenamiento de las capas de la segunda mitad del modelo, ejecutando la siguiente celda.

```
[ ]  # obtenemos la capa media del modelo
     n_layers = int(np.ceil(len(model.layers)/2))

     # activamos el entrenamiento de la segunda mitad de capas
     # del modelo, manteniendo la primera mitad inhibida
     for layer in model.layers[n_layers::]:
       layer.trainable = True

     # comprobamos que se haya habilitado el entrenamiento
     for layer in model.layers: print(layer.trainable)
```

```
False
False
False
False
False
False
False
False
False
False
False
False
False
True
True
True
True
True
True
True
True
True
True
True
True
```

Podemos observar que la primera mitad de capas tienen inhibido su entrenamiento, mientras que las de la segunda mitad lo tienen activo. Ahora, es necesario compilar nuevamente el modelo. Esta vez lo hacemos con un learning-rate reducido de 1e-5. Mantenemos el optimizador, la función de pérdida y la métrica de monitoreo.

```
[ ]  # compilamos una vez más el modelo, con learning-rate reducido
     optimizer = tf.keras.optimizers.Adam(learning_rate = 1e-5)
     model.compile(optimizer = optimizer,
                   loss = 'categorical_crossentropy',
                   metrics =['accuracy'])
```

Y reanudamos el entrenamiento del modelo. Esta vez por solo 5 épocas más, y manteniendo el resto de parámetros a excepción del callback ModelCheckpoint. Este callback debe ser ignorado ya que, de lo contrario, la configuración para el entrenamiento de las capas será restablecida al cargar el checkpoint.

```
[ ]  # entrenamos nuevamente el modelo, por 5 épocas
     inicio = datetime.now()
     history_ft = model.fit(x = train_generator,
                            epochs = 5,
                            validation_data = val_generator,
                            callbacks = [es, reduce_lr])
     fin = datetime.now()
     tiempo_de_ejecucion = fin-inicio
     print("Tiempo de ejecución: ", tiempo_de_ejecucion)
```

```
Epoch 1/5
143/143 [==============================] - 77s 519ms/step - loss: 0.0280 - accuracy: 0.9935 - val_loss: 0.1204 - val_accuracy: 0.9779 - lr: 1.0000e-05
Epoch 2/5
143/143 [==============================] - 76s 530ms/step - loss: 0.0289 - accuracy: 0.9942 - val_loss: 0.0869 - val_accuracy: 0.9809 - lr: 1.0000e-05
Epoch 3/5
143/143 [==============================] - 77s 539ms/step - loss: 0.0185 - accuracy: 0.9945 - val_loss: 0.0911 - val_accuracy: 0.9835 - lr: 1.0000e-05
Epoch 4/5
143/143 [==============================] - 79s 554ms/step - loss: 0.0215 - accuracy: 0.9954 - val_loss: 0.1167 - val_accuracy: 0.9822 - lr: 1.0000e-05
Epoch 5/5
143/143 [==============================] - 80s 556ms/step - loss: 0.0098 - accuracy: 0.9971 - val_loss: 0.1189 - val_accuracy: 0.9772 - lr: 1.0000e-05
Tiempo de ejecución:  0:06:32.087453
```

Luego del fine-tuning, se observa un aumento en la métrica *val_accuracy*. Además, debido a la disminución del valor de *val_loss*, podemos confirmar que el modelo ha mejorado luego de la aplicación de esta técnica.

---

10. PREDICCIÓN SOBRE EL CONJUNTO DE PRUEBA

Una vez que el modelo haya sido entrenado, hacemos la predicción sobre el conjunto de prueba. Para ello:

- Aplicamos, en primer lugar un reinicio del conjunto de prueba, en caso de que hayamos configurado su parámetro *shuffle* como *True*.
- Obtenemos la predicción con el atributo *predict* y lo asignamos a la variable *prediction*.

```
[ ]  # reiniciamos el conjunto de prueba
     test_generator.reset()

     # predecimos sobre el conjunto de prueba
     prediction = model.predict(test_generator)
```

La variable *prediction* es un arreglo con las probabilidades obtenidas para cada clase, en cada imagen. Este arreglo es de tres columnas, una por cada clase, donde en cada una de ellas se indica la probabilidad de pertenencia. Además, la cantidad de filas es igual a la cantidad de imágenes del conjunto de prueba, en este caso 3.031.
Para que estas probabilidades sean interpretables como una clase, obtenemos el índice donde la probabilidad sea mayor. Este índice, que toma valores 0, 1 y 2 en este caso, representa la clase predicha. Para esto, hacemos uso de la biblioteca numpy y la función argmax, y asignamos el resultado en una nueva variable.

```
[ ]  prediction_ = np.argmax(prediction, axis = -1) # axis = -1 indica filas
```

---

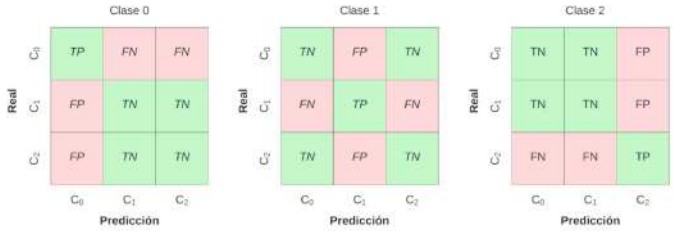
11. DESPLIEGUE DE MÉTRICAS Y EVALUACIÓN DE RENDIMIENTO

Para cada una de las métricas mencionadas, se tiene que:

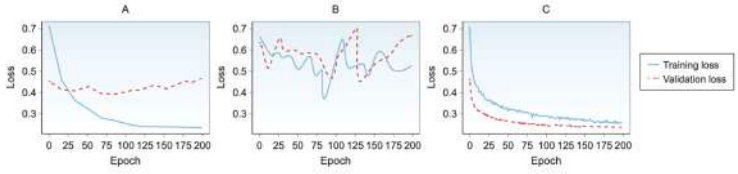
- **TP (true positives o verdaderos positivos):** clases identificadas como positivas cuando la etiqueta real también es positiva.
- **TN (true negatives o verdaderos negativos):** clases identificadas como negativas cuando la etiqueta real también es negativa.
- **FP (false positives o falsos positivos):** clases identificadas como positivas cuando la etiqueta real es negativa.
- **FN (false negatives o falsos negativos):** clases identificadas como negativas cuando la etiqueta real es positiva.

Métricas:

- **Recall:** indica la proporción de clases correctamente identificadas como positivas sobre el total de clases que efectivamente son positivas. $Recall = \frac{TP}{TP+FN}$
- **Precision:** indica la proporción de clases que son correctamente identificadas como positivas sobre el total de clases identificadas como positivas. $Precision = \frac{TP}{TP+FP}$
- **F1-Score:** combina recall y precision en una métrica. $F1 - Score = 2 \cdot \frac{precision \cdot recall}{precision+recall}$
- **Accuracy:** indica el porcentaje de clasificaciones correctas entre la totalidad de datos. $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- **Mátriz de confusión:** expone las imágenes con sus clases reales en filas y las predicciones del modelo en columnas. Dependiendo de la clase observada, los valores para TP, TN, FP y FN pueden ser obtenidos según se muestra a continuación.



- **Curvas de aprendizaje**: permiten evaluar el entrenamiento del modelo a partir de gráficos que reflejan el comportamiento de valores de pérdida y accuracy en conjuntos de validación y entrenamiento.



Donde *A* indica sobreajuste u overfitting, *B* indica conjuntos de datos poco representativos y *C* indica un buen entrenamiento. Imagen obtenida de [4].

Para la obtención de estas métricas haremos uso de la variable *prediction_* obtenida en el paso anterior y las clases reales de las imágenes del conjunto de prueba.

REPORTE DE CLASIFICACIÓN

Primero, obtenemos el reporte de clasificación. Para ello, hacemos uso de la biblioteca Scikit-learn y la función *classification_report*, que recibe como parámetros las clases reales y las clases predichas.

```
[ ] # obtenemos las etiquetas reales
    test_labels = test_generator.classes

    # obtenemos el reporte de clasificación
    report = classification_report(test_labels, prediction_)

    # y lo desplegamos en pantalla
    print(report)
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      2022
           1       0.99      0.94      0.96       272
           2       0.99      0.95      0.97       737

    accuracy                           0.98      3031
   macro avg       0.98      0.96      0.97      3031
weighted avg       0.98      0.98      0.98      3031
```

Podemos observar el valor de las métricas *precision*, *recall* y *f1-score* conseguidas en cada clase, además de accuracy de forma general. La variable *support* indica la cantidad de imágenes pertenecientes a cada clase que forman parte del conjunto de prueba.

- Para precision, las clases con mejor desempeño fueron neumonía y covid, con un 99% cada una. A estos le sigue la clase normal con un 98%. Para la clase normal, esto indica que de todas las imágenes que el modelo clasificó como normal, el 98% efectivamente lo era. De forma similar para la clase neumonía, de todas las imágenes que el modelo clasificó como neumonía, el 99% efectivamente lo era. Por último, para la clase Covid, sucede lo mismo que con neumonía.
- Para recall, el mejor desempeño se dio en las clase normal y covid, con un 100% y 95%, respectivamente. La clase neumonía obtuvo un 94%. Esto se interpreta de la siguiente manera: para la clase normal, de todas las imágenes que son efectivamente de clase normal, el modelo clasificó al 100% como tal. Para la clase neumonía, de todas las clases que son efectivamente de la clase neuminía, el modelo identificó al 94% de ellas. Para la clase covid, de todas las imágenes que efectivamente llevan la etiqueta covid, el modelo clasificó al 95% correctamente.
- Para F1-score, los mejores resultados los obtuvo la clase normal con 99%, seguido de covid con 97% y finalmente neumonía con 96%. Esta métrica permite una evaluación considerando las métricas recall y precision.
- La métrica Accuracy obtenida, fue del 98%. Esto quiere decir que, del total de imágenes disponibles en el conjunto de prueba, el modelo clasificó correctamente al 98% de ellas.

MATRIZ DE CONFUSIÓN

Ahora, obtenemos la matriz de confusión haciendo uso de la biblioteca *Scikit-learn* y la función *confusion_matrix*, que recibe como parámetro las clases reales y las clases predichas.

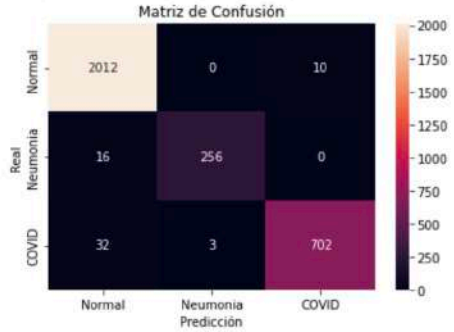
```
[ ] # obtenemos matriz de confusión
    cm = confusion_matrix(test_labels, prediction_)
```

Para su visualización, generamos una figura con Matplotlib y Seaborn.

- En primer lugar, generamos una lista con las etiquetas en palabras. Este paso no es necesario, pero facilita la lectura de la matriz.
- Luego generamos una figura con *Matplotlib*, *Seaborn* y la función *heatmap*

```
[ ] # indicamos una lista con las etiquetas para cada clase
    clases = ['Normal', 'Neumonia', 'COVID']

    # generamos la figura con matplotlib
    plt.figure('Matriz de Confusión')

    # generamos la figura de la matriz
    sns.heatmap(cm, annot = True, fmt = '', xticklabels = clases, yticklabels = clases)
    plt.title('Matriz de Confusión')
    plt.ylabel('Real')
    plt.xlabel('Predicción')
```

```
Text(0.5, 15.0, 'Predicción')
```



Obtenemos la matriz de confusión con cantidad contable de imágenes. En filas, se aprecian las imágenes con sus etiquetas reales, mientras que en las columnas se indica la predicción del modelo.

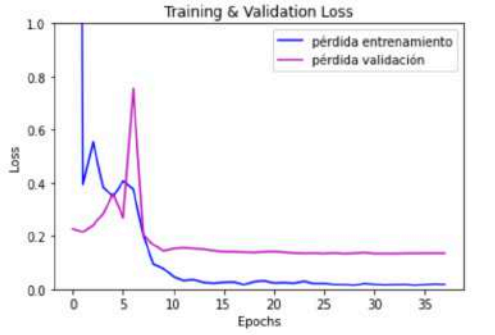
- Para la clase normal, el modelo clasificó 2.012 imágenes de forma correcta de las 2.022 disponibles. Todas las imágenes incorrectamente clasificadas (10) fueron designadas como clase covid.
- Para la clase neumonía, el modelo clasificó 256 imágenes correctamente de un total de 272. Los 16 errores cometidos, fueron clasificados como normal.
- Para la clase covid, el modelo clasificó 702 imágenes correctamente de las 737 disponibles. De las 35 imágenes clasificadas erróneamente, 32 fueron predichas como normal y 3 como Neumonía.

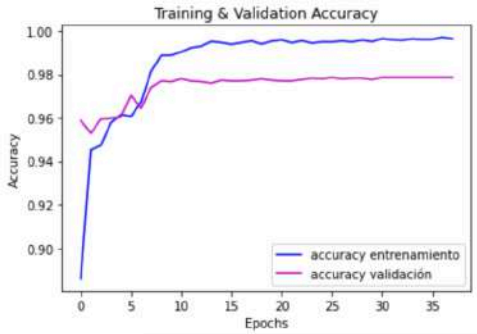
CURVAS DE APRENDIZAJE

Obtendremos dos curvas: una para el valor de pérdida y otra para el valor de accuracy. Para ello, accedemos a la información de la variable *history* generada en el entrenamiento del modelo, y generamos una figura con matplotlib.

```
[ ] # gráfico para valor de pérdida
    plt.figure('Loss')

    # obtenemos y graficamos la pérdida del conjunto de entrenamiento en color azul
    plt.plot(history.history['loss'], label = 'pérdida entrenamiento', color = 'b')

    # obtenemos y graficamos la pérdida del conjunto de validación en color magenta
    plt.plot(history.history['val_loss'], label = 'pérdida validación', color = 'm')

    # ejes, leyenda y título
    plt.xlabel('Epochs') # eje x: épocas
    plt.ylabel('Loss') # eje Y: valor de pérdida (loss)
    plt.legend(loc = 'best') # posicionamos la leyenda automáticamente
    plt.title('Training & Validation Loss') # asignamos un título para la gráfica
    plt.ylim([0, 1])
    # gráfico para valor de accuracy
    plt.figure('Accuracy')

    # obtenemos y graficamos accuracy del conjunto de entrenamiento en color azul
    plt.plot(history.history['accuracy'], label = 'accuracy entrenamiento', color = 'b')

    # obtenemos y graficamos accuracy del conjunto de validación en color magenta
    plt.plot(history.history['val_accuracy'], label = 'accuracy validación', color = 'm')

    # ejes, leyenda y título
    plt.xlabel('Epochs') # eje x: épocas
    plt.ylabel('Accuracy') # eje y: valor de accuracy
    plt.legend(loc = 'best') # posicionamos la leyenda automáticamente
    plt.title('Training & Validation Accuracy') # asignamos un título para la gráfica
```

```
Text(0.5, 1.0, 'Training & Validation Accuracy')
```





Luego de obtener ambas curvas, podemos concluir lo siguiente:

- Para el valor de pérdida, a pesar de presentar problemas de estabilidad en las primeras épocas, en ambos conjuntos se obtuvo el mínimo rápidamente, y se mantuvo bajo con el pasar de las épocas. Esto es un buen resultado, puesto que el objetivo del entrenamiento es disminuir el valor de pérdida.
- Para el valor de accuracy, ambas curvas aumentaron con el paso de las épocas. El accuracy en el conjunto de entrenamiento alcanzó valores más altos, lo cual es un comportamiento común. Sin embargo, los resultados de accuracy en validación se acercaron bastante y coinciden con los obtenidos en el reporte de clasificación luego de aplicar el modelo al conjunto de prueba.

Podemos concluir que el modelo no presentó el fenómeno de sobreajuste y fue entrenado correctamente, obteniendo una buena capacidad de generalización sobre datos nuevos.

Por último, guardamos el modelo obtenido con la función *save* indicando la ruta de destino.

```
[ ] model.save('/content/drive/MyDrive/Tutoriales/COVID-19/modelo_covid19.h5')
```

Además, podemos almacenar la información obtenida en la predicción. Para ello, generamos un nuevo DataFrame con los nombres de archivo, la clase real, la clase predicha, y la probabilidad de pertenencia de la clase predicha a la clase real.

```
[ ] resultados = pd.DataFrame({'filename': test_generator.filenames,
                               'clase_real': test_generator.classes,
                               'clase_predicha': prediction_,
                               'probabilidad': [max(x) for x in prediction]})


    resultados.to_csv('/content/drive/MyDrive/Tutoriales/COVID-19/resultados.csv')
```

Finalmente, podemos visualizar los 5 primeros elementos del DataFrame generado.

```
[ ] resultados.head(5)
```

	filename	clase_real	clase_predicha	probabilidad
0	COVID-1459.png	2	2	1.000000
1	COVID-216.png	2	2	1.000000
2	Normal-1810.png	0	0	0.999868
3	Normal-81.png	0	0	1.000000
4	Normal-2894.png	0	0	1.000000

12. CONCLUSIÓN

En este tutorial se ha implementado el modelo de Red Neuronal Convolucional VGG-16, para la clasificación de imágenes de radiografía de tórax para Covid-19. El objetivo consistió en generar un modelo capaz de clasificar imágenes en tres clases: Normal, Neumonía y Covid.

Se han cubierto los pasos necesarios para la implementación del algoritmo, es decir, la carga y lectura de imágenes, la construcción de los distintos conjuntos de datos, la carga y modificación del modelo pre-entrenado, la aplicación de técnicas para mejorar el rendimiento y, por último, la obtención de métricas y evaluación de resultados.

El modelo obtenido ha logrado un alto nivel en la evaluación sobre el conjunto de prueba. La base de datos utilizada y los conjuntos de imágenes generados, fueron lo suficientemente representativo para tener un entrenamiento óptimo que no presente sobreajuste. Las funciones auxiliares callbacks también han jugado un papel fundamental en el entrenamiento, precisamente en evitar fenómenos de sobreajuste, ya sea deteniendo el modelo de forma anticipada o ejecutando una reducción del learning-rate. Además, se demostró una ganancia en el rendimiento luego de aplicar fine-tuning.

Gracias a este tutorial se ha podido aprender cómo obtener un modelo de inteligencia artificial capaz de clasificar imágenes médicas para la patología Covid-19 con un buen rendimiento sobre datos nuevos.

REFERENCIAS.

[1] M. E. H. Chowdhury, T. Rahman, A. Khandakar, R. Mazhar, M. A. Kadir, Z. B. Mahbub, K. R. Islam, M. S. Khan, A. Iqbal, N. A. Emadi, M. B. I. Reaz, and M. T. Islam, "Can ai help in screening viral and covid-19 pneumonia?" IEEE Access, vol. 8, pp. 132 665–132 676, 2020.

[2] T. Rahman, A. Khandakar, Y. Qiblawey, A. Tahir, S. Kiranyaz, S. B. Abul Kashem, M. T. Islam, S. Al Maadeed, S. M. Zughaier, M. S. Khan, and M. E. Chowdhury, "Exploring the effect of image enhancement techniques on covid-19 detection using chest x-ray images," Computers in Biology and Medicine, vol. 132, p. 104319, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S001048252100113X

[3] V. Komanapalli, N. Sivakumaran, and S. Hampannavar, Advances in Automation, Signal Processing, Instrumentation, and Control: Select Proceedings of i-CASIC 2020, ser. Lecture Notes in Electrical Engineering. Springer Nature Singapore, 2021. [Online]. Available: https://books.google.cl/books?id=lqkhEAAAQBAJ

[4] M. Elgendy, Deep Learning for Vision Systems. Manning Publications, 2020. [Online]. Available: https://books.google.cl/books?id=6gkLzAEACAAJ

[5] M. Abadi et al. "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[6] F. Chollet et al., "Keras," https://keras.io, 2015.

UNIVERSIDAD DE CONCEPCION – FACULTAD DE INGENIERIA
RESUMEN DE MEMORIA DE TITULO

Departamento	**:** Departamento de Ingeniería Eléctrica
Carrera	**:** Ingeniería Civil Biomédica
Nombre del memorista	**:** Nicolás Alfonso Bettancourt Matamala
Título de la memoria	**:** Clasificación de imágenes médicas utilizando redes neuronales
Fecha de la presentación oral	**:** Viernes 26 de Agosto, 2022
Profesor(es) Guía	**:** Pamela Guevara A.
Profesor(es) Revisor(es)	**:** Cecilia Hernández R., Miguel Figueroa T.
Concepto	**:**
Calificación	**:**

Resumen

Modelos de Redes Neuronales Convolucionales de gran prestigio en la literatura científica fueron implementados para tareas de clasificación de imágenes médicas. El objetivo de esta memoria fue generar una base de datos de carácter educativo para el aprendizaje de la implementación y optimización de estos algoritmos. Esto consiste en tutoriales desarrollados en lenguaje de programación Python utilizando herramientas open-source y bases de datos públicas de imágenes médicas. Los modelos estudiados fueron VGG-16, ResNet-50 e Inception-V3, y las bases de datos seleccionadas fueron tres. La primera incluye imágenes de resonancia magnética de tumores cerebrales. La segunda base de datos es sobre Covid-19 e incluye imágenes de radiografía de tórax. La tercera base de datos es una colección de imágenes de fondo de retina. El estudio fue dividido en etapas que cubrieron el efecto que distintas configuraciones poseen sobre el rendimiento de la clasificación, además de la aplicación de técnicas para mejorarlo. Los resultados ayudaron a definir el diseño y construcción de tres tutoriales —uno por cada base de datos—, que comprenden la información fundamental y pasos necesarios para el desarrollo de un modelo de inteligencia artificial capaz de clasificar imágenes médicas.