UNIVERSIDAD RICARDO PALMA

FACULTAD DE INGENIERÍA

PROGRAMA DE TITULACIÓN POR TESIS

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

EVALUACIÓN DE MODELOS DE REDES NEURONALES CONVOLUCIONALES APLICADO A RADIOGRAFÍAS DE TÓRAX, PARA APOYAR AL PROCESO DE DIAGNÓSTICO DE NEUMONÍA ASOCIADA AL COVID-19

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE

INGENIERO ELECTRÓNICO

PRESENTADA POR:

Bach. CAYA PÉREZ, JHAN CARLOS

Asesor: Dr. Ing. HUAMANÍ NAVARRETE, PEDRO FREDDY

LIMA-PERÚ

2020

DEDICATORIA

A mis padres por brindarme el apoyo necesario e incondicional en mi etapa universitaria, además de enseñarme a nunca rendirme ante los problemas. A mi prima Vanessa que me apoya desde el cielo. A mis queridas Bomi y Suji, dado que sin el apoyo de ellas no hubiera sido capaz de continuar y afrontar el día a día.

AGRADECIMIENTO

A Mel por brindarme su tiempo, conocimientos y consejos necesarios hacia mi persona. A mi asesor el Dr. Pedro Huamaní Navarrete por guiarme a aprender sobre inteligencia artificial y por la orientación brindada en la elaboración de esta tesis.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En marzo del presente año la OMS declaró como pandemia mundial al COVID-19. Esta enfermedad, causada por el nuevo coronavirus, afecta principalmente al sistema respiratorio ocasionando enfermedades graves como la neumonía. Esta requiere para su diagnóstico la revisión y análisis de radiografías de tórax, las cuales permiten evaluar el estado de los pulmones. Sin embargo, ante este contexto de pandemia, el tiempo empleado para detectar la enfermedad dificulta ofrecer un tratamiento temprano y oportuno al paciente. Por ello, este proyecto de tesis propuso la evaluación de tres modelos de redes neuronales convolucionales aplicado a radiografías de tórax, para apoyar al proceso de diagnóstico de neumonía asociada al COVID-19, por medio de la clasificación de imágenes. Esto permite contribuir en la reducción del tiempo que toma la labor de detección de la enfermedad en radiografías de tórax, además de definir cuál de los tres modelos es el más apto para dicha labor. Los modelos utilizados en este proyecto de tesis son un modelo de implementación propia, ResNet50 e InceptionV3. Para la implementación de estos dos últimos se requirió aplicar transfer learning. Además, se aplicó data augmentation para conocer su utilidad e influencia en el proceso de entrenamiento de cada uno de los tres modelos. Se utilizó un dataset compuesto por imágenes de radiografías de tórax de casos positivos a COVID-19 y casos normales para el entrenamiento y validación de los tres modelos. Finalmente, basado en los resultados obtenidos, el modelo más efectivo de los tres evaluados fue InceptionV3 con un 0.9886 de exactitud cuando se entrenó con data augmentation y 0.9848 sin data augmentation.

Palabras claves: COVID-19, red neuronal convolucional, radiografías de tórax, transfer learning, data augmentation.

ABSTRACT

In March of this year, WHO declared COVID-19 a global pandemic. This disease, caused by the new coronavirus, mainly affects the respiratory system causing serious illnesses such as pneumonia. This disease, caused by the new coronavirus, mainly affects the respiratory system, causing serious diseases such as pneumonia. However, in the context of this pandemic, the time taken to detect the disease makes it difficult to offer early and timely treatment to the patient. For this reason, this thesis project proposed the evaluation of three models of convolutional neuronal networks applied to chest radiography, to support the process of diagnosis of pneumonia associated to COVID-19, by means of image classification. This allows to contribute in the reduction of the time that takes the detection of the disease in thorax radiographies, besides defining which of the three models is the most suitable for such work. The models used in this thesis project are a proprietary model, ResNet50 and InceptionV3. For the implementation of the latter two it was required to apply transfer learning. In addition, data augmentation was applied to determine its usefulness and influence on the training process of each of the three models. A dataset composed of images from chest X-rays of COVID-19 positive cases and normal cases was used for training and validation of the three models. Finally, based on the results obtained, the most effective model of the three evaluated was InceptionV3 with 0.9886 accuracy when trained with data augmentation and 0.9848 without data augmentation.

Keywords: COVID-19, Convolutional Neural Network, chest X-ray, transfer learning, data augmentation.

INTRODUCCIÓN

A finales del 2019 el mundo se enteraba de la existencia del COVID-19, la cual fue declarada a mediados de marzo como pandemia mundial por parte de la OMS. Esto motivo en muchos países a establecer protocolos para el cuidado preventivo y detección esta enfermedad, así como controlar la cantidad de infectados. Esta enfermedad afecta principalmente al sistema respiratorio, llegando a desarrollar en algunos casos neumonía. El presente proyecto de tesis evalúa tres modelos de redes neuronales convolucionales aplicado a radiografías de tórax, para apoyar al proceso de diagnóstico de neumonía asociada al COVID-19, por medio de la clasificación de las imágenes permitiendo de este modo contribuir con la reducción del tiempo que toma la labor de detección de la enfermedad en radiografías de tórax, además de indicar cuál de los tres modelos es el más apto para esta labor. Los modelos evaluados fueron uno de implementación propia, ResNet50 e InceptionV3. El interés que llevo a desarrollar este proyecto de tesis, fue para la problemática existente del tiempo empleado para la detección de la enfermedad por medio de la revisión y análisis de radiografías de tórax. Dado que, ante este contexto de pandemia, se requiere brindar un tratamiento temprano y oportuno al paciente para evitar mayores complicaciones en su salud.

La investigación es cuantitativa de tipo experimental, además empleó un método empírico para su desarrollo. Su enfoque es cuantitativo, porque tuvo un proceso sistemático, rígido y ordenado para comprobar la relación causa-efecto entre las variables de investigación que vienen a ser los modelos de redes neuronales convolucionales y el diagnóstico de neumonía asociada al COVID-19. Asimismo, es de tipo experimental porque manipuló los modelos de redes neuronales convolucionales para ayudar en el proceso de diagnóstico de la enfermedad, esto fue comprobado por medio de resultados obtenidos en las pruebas experimentales; y su método es empírico porque se basó en la experiencia adquirida por las pruebas realizadas, utilizando un dataset conformado por imágenes de radiografías de tórax.

Para el desarrollo de esta investigación se implementaron y entrenaron cada uno de los modelos de forma independiente, utilizando Keras con TensorFlow en Python empleando una laptop convencional. Finalmente, la comparación del desempeño de los modelos propuestos para la clasificación, en las radiografías de tórax, se obtuvo por medio de los resultados obtenidos en las pruebas realizadas a cada uno de ellos.

En el capítulo 1, se aborda el planteamiento y delimitación del problema, así como también se definen los objetivos necesarios para desarrollo del proyecto de tesis. Asimismo, se explican la importancia, justificación y limitación del mismo.

En el capítulo 2, se elabora el marco teórico para sustentar teóricamente lo planteado en el capítulo 1. Se realiza la revisión y análisis de investigaciones relacionadas con el tema de investigación planteado, se definen y detallan las bases teóricas para comprender los conceptos de las variables de investigación. Finalmente, se desarrolla el diseño de investigación que posee el proyecto de tesis.

En el capítulo 3, se generó del dataset con las imágenes de radiografías de tórax adquiridas, para posteriormente preprocesar dichas imágenes. Finalmente se implementan y entrenan cada uno de los modelos de redes neuronales convolucionales, siendo necesaria la aplicación de transfer learning para la implementación de los modelos ResNet50 e InceptionV3, además se evaluó la influencia que tiene la aplicación de data augmentation en el proceso de entrenamiento en cada uno de los modelos implementados.

En el capítulo 4, se realizan las pruebas de predicción para comprobar el funcionamiento de los modelos de redes neuronales convolucionales entrenados. Además, se presentan las gráficas del proceso de entrenamiento de cada modelo para mostrar la utilidad e influencia de la aplicación e inaplicación de data augmentation. Asimismo, se presentan las tablas de las métricas de medición y las matrices de confusión de cada modelo. Finalmente, se muestran las tablas comparativas basadas en las tablas de las métricas de medición y matrices de confusión mencionadas además de una tabla referente al presupuesto utilizado para este proyecto de tesis.

CAPÍTULO I: PLANTEAMIENTO Y DELIMITACIÓN DEL PROBLEMA

1.1. Formulación del problema

La COVID-19, es una infección respiratoria causada por nuevo coronavirus, siendo catalogada como pandemia por parte de la OMS en marzo del presente año. Esta enfermedad provoca síntomas desde tos seca y fiebre, hasta complicaciones más graves como la insuficiencia respiratoria y neumonía. Para el caso particular de la neumonía se requiere realizar una radiografía de tórax, la cual proporciona información sobre el estado de los pulmones con el fin de obtener su diagnóstico adecuado. De este modo, poder indicar el tratamiento médico a seguir para la recuperación de la salud del paciente.

Sin embargo, en este contexto de pandemia el tiempo empleado para detectar la neumonía por medio de la radiografía de tórax, termina siendo un tiempo valioso debido a que esta enfermedad requiere recibir un tratamiento oportuno en el menor tiempo posible. Por ello, es importante brindar un apoyo al proceso de diagnóstico para optimizar y agilizar la detección de la enfermedad en radiografías de tórax. Para así poder asignar en el menor tiempo posible el tratamiento adecuado para el diagnóstico de neumonía obtenido y evitar mayores complicaciones en la salud de los pacientes con COVID-19.

A continuación, se presenta la formulación del problema:

1.1.1. Problema General

¿Cómo evaluar los modelos de redes neuronales convolucionales aplicado a radiografías de tórax, para apoyar al proceso de diagnóstico de neumonía asociada al COVID-19?

1.1.2. Problemas Específicos

a. ¿Cómo implementar y entrenar un modelo particular de red neuronal convolucional, utilizando Keras con TensorFlow en Python y desarrollado en una laptop convencional?
b. ¿Cómo implementar y entrenar el modelo de red neuronal convolucional ResNet50, utilizando Keras con TensorFlow en Python y desarrollado en una laptop convencional?
c. ¿Cómo implementar y entrenar el modelo de red neuronal convolucional InceptionV3, utilizando Keras con TensorFlow en Python y desarrollado en una laptop convencional?

d. ¿Cómo comparar el desempeño de los tres modelos de redes neuronales convolucionales propuestos para la clasificación en las radiografías de tórax de neumonía asociada al COVID-19?

1.2. Objetivos

1.2.1. Objetivo General

Evaluar los modelos de redes neuronales convolucionales aplicado a radiografías de tórax, para apoyar al proceso de diagnóstico de neumonía asociada al COVID-19.

1.2.2. Objetivos Específicos

a. Implementar y entrenar un modelo particular de red neuronal convolucional, utilizando Keras con Tensorflow en Python y desarrollado en una laptop convencional.
b. Implementar y entrenar el modelo de red neuronal convolucional ResNet50, utilizando Keras con Tensorflow en Python y desarrollado en una laptop convencional.
c. Implementar y entrenar el modelo de red neuronal convolucional InceptionV3, utilizando Keras con Tensorflow en Python y desarrollado en una laptop convencional.
d. Comparar el desempeño de los tres modelos de redes neuronales convolucionales propuestos para la clasificación en las radiografías de tórax de neumonía asociada al COVID-19.

1.3. Importancia y justificación

1.3.1. Importancia

En el presente proyecto se propone un apoyo al proceso de diagnóstico de neumonía asociada al COVID-19 aplicando Deep Learning en radiografías de tórax, a través de la evaluación del desempeño de tres modelos de redes neuronales convolucionales. Con el fin de conocer cuál de los tres modelos es el más óptimo para detectar un posible caso de neumonía asociada al COVID-19. Asimismo, este proyecto de tesis es importante porque emplea redes neuronales convolucionales en la clasificación de imágenes de radiografías de tórax para agilizar el proceso de diagnóstico.

1.3.2. Justificación

Ante la coyuntura actual del COVID-19, se requiere un accionar médico rápido y oportuno para diagnosticar la neumonía provocada por el nuevo coronavirus. Por ello, es necesario reducir el tiempo del proceso de diagnóstico de esta enfermedad para que el tratamiento proporcionado pueda ser más efectivo. Razón por la cual, este proyecto de investigación propone la evaluación de tres modelos de redes neuronales convolucionales, en la aplicación de clasificación de imágenes, utilizando radiografías de tórax, para apoyar al proceso de diagnóstico. De esta manera, señalar cuál de los tres modelos es el más indicado para detectar la neumonía asociada al COVID-19 por medio de radiografías de tórax, así apoyar al proceso de diagnóstico, de la enfermedad y reducir el tiempo reducir el tiempo que ocupa este proceso.

1.4. Limitaciones

La siguiente investigación consta de las siguientes limitaciones:

- El tamaño del dataset con el que se cuenta es limitado, dado que se logró obtener 662 imágenes adecuadas de radiografía de tórax por cada clase (COVID-19 y Normal). Dando un total de 1324 imágenes de radiografías de tórax. Las cuales fueron adquiridas por medio de bases de datos extraídas de plataforma webs.
- Los formatos de las imágenes de radiografías de tórax obtenidas se encuentran en los formatos PNG y JPG. Esto implica que existe pérdida de información dado que el formato adecuado para almacenamiento de imágenes médicas es DICOM (Digital Imaging and Comunication On Medicine).
- Para al uso de la librería TensorFlow, se recomienda usar un hardware como la Tarjeta GPU NVIDIA® con capacidad de procesamiento CUDA® 10.1 o versiones posteriores y cuDNN 7.5 o versiones superiores (TensorFlow, 2020). Sin embargo, este proyecto de tesis se limita a utilizar la CPU de una laptop convencional que posee una tarjeta de video NVIDIA GeForce GT 540M, para realizar la implementación y entrenamiento de las redes neuronales convolucionales.
- Asimismo, es primordial aclarar que el empleo de las redes neuronales convolucionales en radiografías de tórax, solo brindará una clasificación entre casos positivos a COVID-19 y no COVID-19. Esto permitirá realizar un apoyo en el proceso de diagnóstico de neumonía asociada al COVID19. No obstante, el diagnóstico clínico de la enfermedad únicamente puede ser proporcionado por un médico especialista.

CAPÍTULO II: MARCO TEÓRICO

2.1. Marco Histórico

En el pasado para poder realizar una adecuada lectura de las radiografías de tórax, solo se contaba con la experiencia del médico especialista como aval, que luego de una inspección minuciosa de cada una de las placas radiográficas podía confirmar si el paciente padecía de alguna enfermedad pulmonar. Por otra parte, con el paso de los años la tecnología se ha convertido en una herramienta fundamental para la medicina, dado que proporciona un apoyo en la mejora de evaluaciones y diagnósticos de enfermedades. Así como, por ejemplo, tenemos el sistema de diagnóstico asistido por ordenadores (CAD, por sus siglas en inglés), el cual de acuerdo con Rodríguez (2017), realiza una caracterización de la información, con el fin de detectar patrones en los datos (radiografías, tomografías, entre otros), estableciendo la importancia relativa de las variables y finalmente proporcionando un diagnóstico secundario al del medio especialista. Asimismo, el desarrollo de la inteligencia artificial aplicada en la medicina provee un apoyo aún mayor para el proceso de diagnóstico de enfermedades. Esto es un indicador que muestra la importancia que tiene y tendrá la tecnología en la medicina en los próximos años, buscando como objetivo conjunto la precisión y optimización del diagnóstico de enfermedades, reduciendo en el proceso los posibles errores observacionales humanos que se puedan suscitar por parte del médico.

2.2. Antecedentes de la investigación.

Sethi et al. (2020), en el artículo: Deep Learning based Diagnosis Recommendation for COVID-19 using Chest X-Rays Images.

Proponen un sistema de recomendación basado en Deep Learning, el cual ayuda cuando la cantidad de pacientes es muy alta y la experiencia radiológica requerida es baja. En este estudio, se investigaron cuatro arquitecturas diferentes de red neuronal convolucional profunda (Xception, MobileNet, ResNet50 e InceptionV3) usando imágenes de rayos X de tórax obtenidas de varios repositorios de la plataforma de GitHub, mientras que para la implementación de las diferentes arquitecturas se utilizó el editor Jupyter Notebook. El objetivo de esta investigación es proporcionar una recomendación del diagnóstico en pacientes de COVID-19. Los modelos de redes neuronales convolucionales son pre-entrenados en la base de datos de ImageNet, es decir, se les proporciona pesos pre-entrenados que ayudan a transferir la información o data previa sobre el conjunto de datos

que se está investigando. Los resultados muestran que, de los cuatro modelos estudiados, MobileNet es el mejor. Asimismo, se demuestra que las arquitecturas basadas en redes neuronales convolucionales tienen el potencial para el correcto diagnóstico de la COVID-19. Además, la técnica de aprendizaje llamada transfer learning y el ajuste fino de los modelos juegan papeles importantes en la mejora de precisión. Finalmente, en trabajos futuros se sugiere incluir el desarrollo de nuevas arquitecturas basadas en redes neuronales convolucionales para la detección de COVID-19, así como de otras enfermedades en el ámbito médico.

La comparativa entre resultados obtenidos por las cuatro arquitecturas en la clasificación de imágenes de rayos X de tórax se presenta en la tabla 1.

Tabla N°1: Comparación de resultados de las cuatro arquitecturas

Method	**Evaluation Metrics**								
	TP	**TN**	**FP**	**FN**	**Accuracy**	**Specificity**	**Precision**	**Recall / Sensitivity**	**F1 Score**
InceptionV3	73	1461	12	17	0.981	0.992	0.859	0.811	0.834
ResNet50	86	1203	4	270	0.825	0.997	0.956	0.242	0.386
MobileNet	79	1462	11	11	0.986	0.993	0.878	0.878	0.878
Xception	56	1466	7	34	0.974	0.995	0.889	0.622	0.732

Fuente: (Sethi, Mehrotra, & Sethi, 2020)

Colula y Ángel (2019), en la tesis: Reconocimiento de patrones en imágenes médicas por medio de redes neuronales convolucionales.

Presentan tres modelos redes neuronales convolucionales pre-entrenados e implementados en Python utilizando las librerías de TensorFlow y Keras, para la implementación de estos modelos se usó la técnica de aprendizaje Transfer Learning. Además, el objetivo de esta investigación es resolver los problemas de clasificación o detección de patrones asociados a enfermedades en diferentes imágenes médicas. En esta investigación las imágenes médicas utilizadas fueron resonancias magnéticas del cerebro, tomografías de tórax y radiografías de tórax. Para el primer caso, se utilizó el modelo pre entrenado InceptionV3 con una base de datos basada en 13 pacientes con tumores cerebrales del Instituto Tecnológico de Montreal en el 2010. El modelo de red neuronal convolucional fue entrenado con el fin de identificar tumores cerebrales, los resultados muestran que el modelo InceptionV3 permite clasificar imágenes con tejido tumoral. En el segundo caso también se utilizó el modelo InceptionV3 con una base de datos que contienen imágenes de radiografías correspondientes a 5 tipos de tuberculosis con más de

1000 muestras por tipo. se realizaron diferentes evaluaciones, la conclusión que se alcanza es que el modelo de red neuronal convolucional se comporta de manera más efectiva cuando la cantidad de muestras por clases esta balanceada y si la clasificación es del tipo binaria. En el tercer y último caso se presenta el modelo CheXNet, para poder detectar la neumonía en las imágenes de radiografías del tórax. Para ello, se utilizó la base de datos proporcionada por la Sociedad Radiológica de América del Norte (RSNA). Obteniéndose como resultado una mejora marginal al momento de clasificar las imágenes de rayos X con/sin neumonía. Además de ello, los autores proporcionan ejemplos de la implementación de una red neuronal convolucional para la comprensión de los modelos de redes neuronales que utilizaron. Finalmente recomiendan que para trabajos futuros se utilice el PCA (Análisis de componentes principales) para obtener una descomposición de las imágenes de rayos X para formar imágenes de pseudo color previo a la capa de entrada de la CNN para detectar la columna.

Narin et al. (2020), en el artículo: Automatic detection of coronavirus disease (covid-19) using x-ray images and deep convolutional neural networks.
Presentan un estudio referente a cinco modelos de redes neuronales convolucionales pre-entrenados (ResNet50, ResNet101, ResNet152, InceptionV3 e Inception-ResNetV2), los cuales se han propuesto para la detección de pacientes infectados con neumonía por coronavirus usando imágenes de radiografías de tórax. Estas imágenes fueron obtenidas de tres diferentes bases de datos. En primer lugar, se utilizó la base de datos de Cohen ubicada en el repositorio GitHub, de esta se extrajeron imágenes referentes a casos de SDRA (dificultad de respiración aguda), COVID-19, MERS (síndrome respiratorio de Oriente Medio) y SARS (síndrome respiratorio agudo severo). En segundo lugar, se extrajeron 2800 imágenes de casos normales de la base de datos "ChestX-ray8" perteneciente a Wang et al. En tercer y último lugar, se extrajeron 2772 imágenes de neumonía bacteriana y 1493 de neumonía viral de la base de datos “Chest X-Ray Images (Pneumonia)”, la cual se encuentra en la plataforma Kaggle. Una vez extraídas las imágenes de todas las bases de datos se decidió crear tres dataset: (COVID-19/normal, COVID-19/Neumonía Viral y COVID-19/Neumonía Bacteriana). Estas bases de datos son presentadas en la tabla N°2 a continuación.

Tabla N°2: Número de imágenes por clase para cada conjunto de datos

	Neumonía Bacteriana	COVID-19	Normal	Neumonía Viral
Dataset 1		341	2800	
Dataset 2		341		1493
Dataset 3	2772	341		

Fuente: (Narin, Kaya, & Pamuk, 2020)

Estas tres bases de datos son utilizadas en tres casos de clasificación binaria, además se cuenta con cuatro clases (COVID-19, normal (sano), neumonía viral y la neumonía bacteriana). Referente a la implementación de los modelos trabajados, se decidió aplicar Transfer Learning, esta se ejecutó agregando 3 capas nuevas a cada uno de los cinco modelos, estas capas fueron una pooling Average para reducir el tamaño de las imágenes y aumentar la profundidad de las mismas, seguido de una capa Dense con 1024 neuronas y activación ReLU, es con el fin de que todas las neuronas se conecten con las 1024 de dicha capa, y una capa de salida con activación Softmax, la cual permite clasificar de dos a más clases. De este modo, se logró obtener una red neuronal convolucional adecuada para realizar la clasificación deseada. Los resultados obtenidos referente al rendimiento de los modelos de redes neuronales convolucionales implementados indican que el modelo que proporciona un alto rendimiento en clasificación realizada es ResNet50 con un 96,1% de precisión utilizando el Dataset-1, 99,5% de precisión utilizando el Dataset-2 y 99,7% de precisión utilizando el Dataset-3. Finalmente, se recomienda para estudios posteriores comprobar el rendimiento de la clasificación de diferentes modelos de CNN aumentando el número de imágenes de rayos X de tórax de COVID-19 en el conjunto de datos.

Ozturk et al. (2020), en el artículo: Automated detection of COVID-19 cases using deep neural networks with X-ray images.

Presentan un nuevo modelo de red neuronal convolucional para la detección automática del COVID-19 mediante imágenes de rayos X de tórax. La base de datos que se utilizó está compuesta por otras bases de datos, obtenidas de plataformas como GitHub y Kaggle, además utilizó una cantidad de datos balanceada con la intención de optimizar la clasificación de las imágenes. El modelo propuesto se desarrolla para proporcionar diagnósticos precisos en la clasificación binaria (COVID vs. No-Findings) y en la clasificación multiclase (COVID vs. No-Findings vs. Pneumonia). El modelo produjo

posee una precisión de clasificación del 98,08% para las clases binarias y del 87,02% para los casos de clases múltiples. Su diseño está basado en el modelo DarkNet, el cual cumple la función de clasificador para el sistema de detección de objetos en tiempo real " you only look once" (YOLO). Para adaptar el modelo DarkNet a las necesidades requeridas se le agregaron 17 capas convolucionales e introdujeron diferentes números de filtros en cada capa como de 8, 16 y 32. Este nuevo modelo fue llamado DarkCovidNet. Los autores explican que, para poseer una mejor comprensión de este nuevo modelo, es útil entender los fundamentos de la Darknet-19, el cual consiste en 19 capas convolucionales y cinco capas de max pooling, dando como resultado una estructura como la que se presenta a continuación:

C1-M1-C2-M2-C3-C4-C5-M3-C6-C7-C8-M4-C9-C10-C11-C12-C13-M5-C14-C15-C16-C17-C18-C19

Finalmente, consideran que una limitación de su estudio es el uso de un número limitado de imágenes de rayos X de COVID-19. Dado que pretenden que su modelo sea mucho más robusto y preciso para utilizarlo en hospitales de su localidad.

Sharma et al. (2020), en el artículo: Feature Extraction and Classification of Chest X-Ray Images Using CNN to Detect Pneumonia.

Proponen dos arquitecturas de redes neuronales convolucionales (CNN), una con capa de Dropout (capa de abandono) y otra sin capa Dropout, ambas CNN se encuentran estructuradas por una capa de convolución, un max pooling y una capa de clasificación diseñadas desde cero para detectar la neumonía a partir de imágenes de rayos X de tórax. Como solución para evitar el sobreajuste en la CNN sin capa Dropout, se utilizaron técnicas de aumento de datos. Respecto al diseño de la arquitectura de la CNN, esta consiste en una serie de capas de convolución y de max pooling, las cuales actúan como un extractor de características que se divide en dos partes. La primera parte, consiste en dos capas de convolución con 32-32 unidades cada una, junto con una capa de agrupación máxima de tamaño 3×3 y un activador ReLU. Mientras que la otra también tiene dos capas de convolución, pero con 64 y 128 unidades respectivamente junto con una capa de agrupación máxima de tamaño 2×2 y un activador ReLU. Ambas con la activación Sigmoud en la capa de salida. Respecto al resultado obtenido en los experimentos realizados para evaluar el rendimiento de las arquitecturas propuestas y el efecto de los datos, nos da a conocer que la CNN con mayor rendimiento es la que posee la capa Dropout, la cual fue entrenada con data augmentation. Como trabajos futuros se plantea

utilizar diferentes optimizadores y otras técnicas de aumento de datos en un intento de mejorar aún más la precisión de clasificación de las arquitecturas propuestas de CNN con aumento de datos. También se planea utilizar early stopping (parada temprana) y batch normalization (normalización por lotes) en lugar de la capa de abandono como técnica de regulación básica para ver su efecto en evitar el sobreajuste.

Artola Moreno (2019), en la tesis: Clasificación de imágenes usando redes neuronales convolucionales en Python.
Presenta un estudio sobre el funcionamiento de las redes neuronales convolucionales y su aplicación en clasificación de imágenes, también explica el concepto y la estructura de una red neuronal convolucional, detallando el funcionamiento de cada una de sus capas. Luego define los conceptos de diferentes modelos de redes neuronales convolucionales profundas como son LeNet, AlexNet, VGG (VGG-16 y VGG-19), GoogleNet, ResNet y DenseNet, de los cuales para la parte experimental utiliza ResNet y CancerNet, este último no es un modelo oficial pero su modelo se asemeja en funcionamiento al VGG. Además, para la implementación de ambos modelos se aplica data augmentation. Posteriormente, desarrolla un análisis sobre los lenguajes de programación más destacados y por qué se declina por Python utilizando librerías como Tensorflow y Keras. En la parte experimental utilizó una base de datos formada por otras como es el de IDC (Invasive Ductal Carcinoma), unas extraídas de la plataforma Kaggle y otra base de datos propia. Esta base de datos se encuentra divida en dos conjuntos (entrenamiento y test) por la regla de 80-20, y consta de imágenes de cánceres de mama, melanoma y lesiones de hueso o musculo. Finalmente, referente a los resultados, para la comprobación del correcto funcionamiento de las redes neuronales convolucionales se utilizó el conjunto de datos de huesos o músculos debido a que no se encontró suficiente información de los otros dos, obteniendo una precisión del 92%, mientras que en el resto de conjuntos menor al 90%. Como recomendación a investigaciones futuras el autor recomienda seguir la línea de desarrollo de Deep Learning, especialmente en el ámbito médico.

2.3. Bases Teóricas relacionadas con el tema

En esta sección se describen las bases teóricas relacionadas con las variables de investigación para la comprensión del proyecto de tesis.

2.3.1. Redes Neuronales Convolucionales

Para poder comprender el concepto de una red neuronal convolucional, es necesario previamente definir el concepto de una red neuronal artificial.

De acuerdo con Salas (2004), una red neuronal artificial es un modelo computacional inspirado en el sistema nervioso del ser humano, es decir, que su funcionamiento y estructura están basados en una neurona biológica. Esto se puede observar en la figura N°1, donde se compara la estructura de una neurona artificial con la estructura de una neurona biológica. Además, al conjunto de neuronas artificiales se le conoce con el nombre de capa, siendo la suma e interconexión de estas el resultado conocido como una red neuronal artificial. La cual consiste en una capa de entrada, una o varias capas ocultas y una capa de salida. Tal como se muestra en la figura N°2.

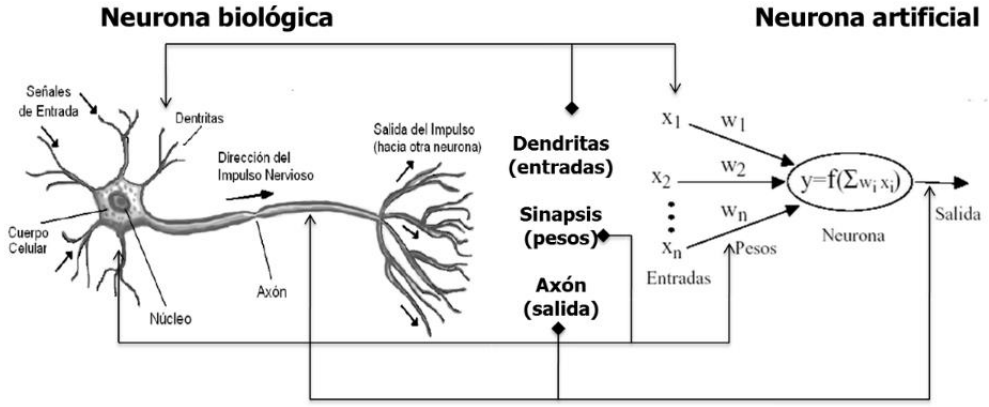


Figura N°1: Comparación entre una neurona biológica y una neurona artificial
Fuente: Lao, Y. O et al. (2017)

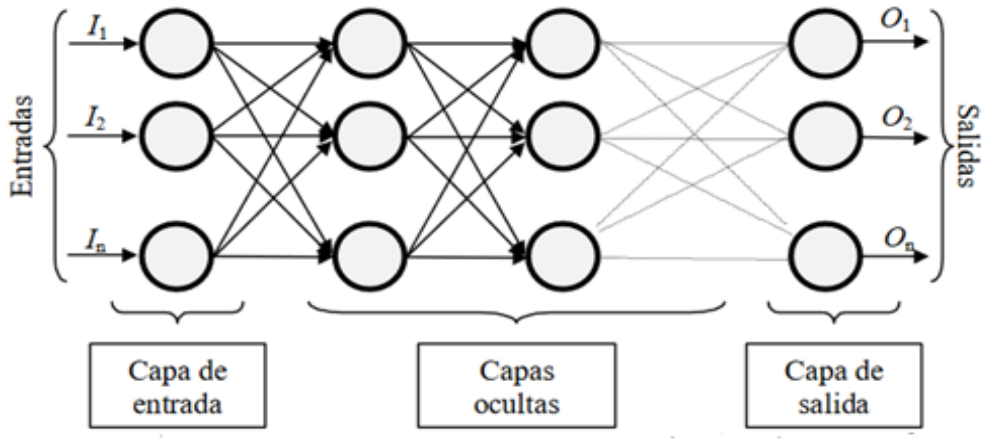


Figura N°2: Esquema de una red neuronal artificial
Fuente: Matich (2001)

Por lo tanto, el concepto de una red neuronal convolucional se puede definir de la siguiente manera. De acuerdo con Medrano (2019), una red neuronal convolucional (*convolutional neural network - CNN*), es un tipo de red neuronal artificial profunda, es decir, posee una gran cantidad de capas ocultas y es de aprendizaje supervisado. Las CNN nacen por la necesidad que se tiene de procesar imágenes de manera efectiva y eficiente, modificando el tamaño de las dimensiones de datos ingresados permitiendo reducir estas dimensiones sin perder la calidad de la imagen, utilizando diferentes capas para este proceso. Entre sus aplicaciones más conocidas se encuentran la clasificación de imágenes, reconocimiento de objetos, reconocimiento de voz y reconocimiento de patrones de lenguaje natural. La arquitectura que presenta una CNN se visualiza en la figura N°3.

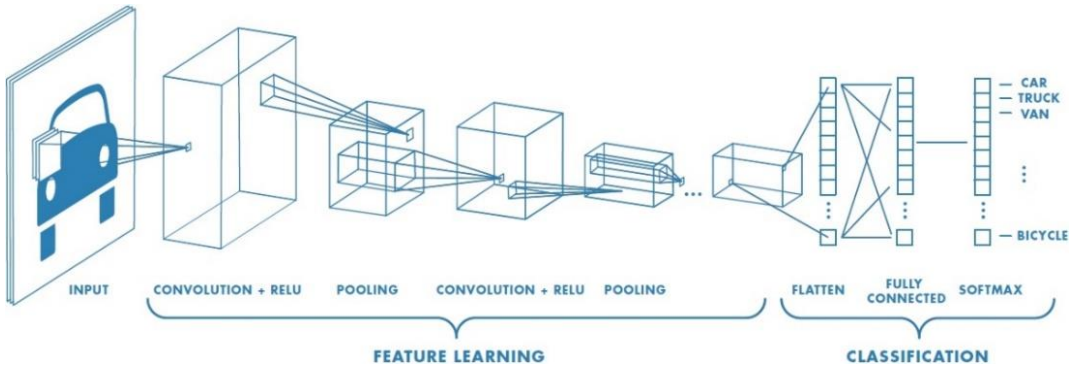


Figura N°3: Arquitectura de una red neuronal convolucional
Fuente: Mathworks (2020)

Como se observa en la figura N°3, la arquitectura de una CNN cuenta con múltiples capas, las cuales son: de entrada, de convolución, Pooling, Flatten, Full Connected y de salida, así como las activaciones ReLU y Softmax. Además de la capa Dropout, la cual fue mencionada por Sharma et al. (2020) en su artículo de investigación. A continuación, se definen las capas antes mencionadas para un mejor entendimiento del funcionamiento de una red neuronal convolucional.

2.3.1.1. Capa de entrada

Esta capa permite asignar el tamaño de las dimensiones de los datos de entrada (ancho, altura y profundidad). Las imágenes en escala a gris poseen una profundidad igual a 1, mientras que las de color poseen una igual a 3 dado sus tres canales (RGB). Esto último se observa en la figura N°4.

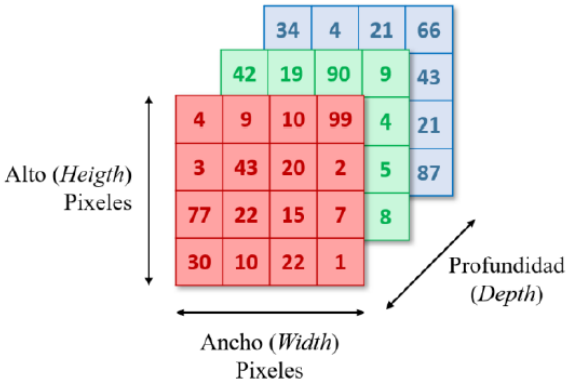


Figura N°4: Capa de entrada con tres canales (RGB)
Fuente: Medrano (2019)

2.3.1.2. Capa de Convolución

Esta capa ejecuta la operación de convolución, la cual consiste en realizar producto punto entre los valores de la capa de entrada y los pesos que posee los filtros (conjunto de kernels) que son definidos dentro de esta capa. El resultado de ello, se le conoce como feature maps, el cual posee diferentes dimensiones que el dato ingresado. Este proceso se realizará tantas veces como la cantidad de "pasos" que fue definido al momento de implementar la red. El proceso de convolución se puede observar en la figura N°5.

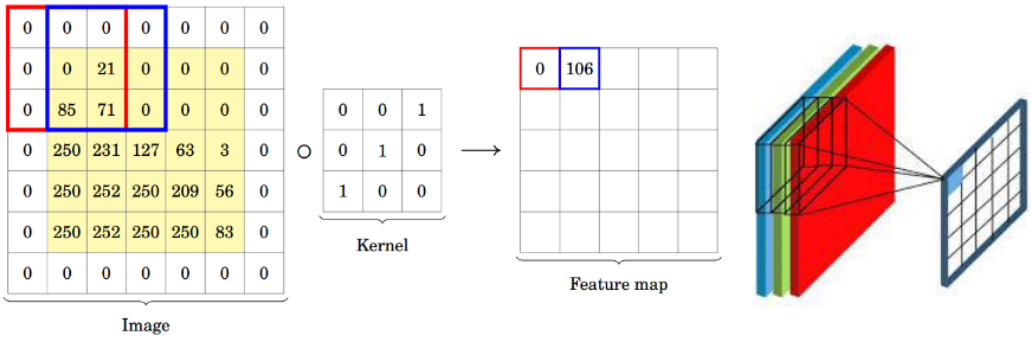


Figura N°5: Operación de convolución
Fuente: Erroz (2019)

2.3.1.3. Capa Pooling

Esta capa realiza una operación para reducir las dimensiones del dato recibido, así procesar una imagen más pequeña y con esto proporcionar un menor esfuerzo computacional a la máquina. Esta capa también posee una operación denominada "max pooling", la cual proporciona la salida máxima dentro de una vecindad rectangular. Esto se puede observar en la figura N°7.

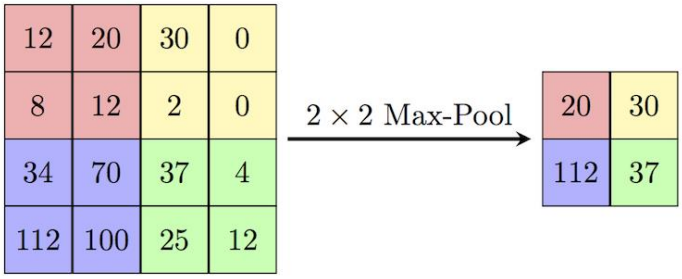


Figura N°6: Operación Max Pooling
Fuente: Jauregui (2020)

2.3.1.4. Capa Dropout

Esta capa está diseñada para prevenir y reducir el problema de sobreajuste, conocido también como overfitting, el cual suele suscitarse en el proceso de entrenamiento de la red. Su función radica en que durante el entrenamiento desactiva una cantidad determinada de neuronas en cada "paso" durante una "epoca" evitando que la red neuronal aprenda un camino en específico para realizar el proceso de clasificación.

2.3.1.5. Capa Flatten

Esta capa realiza un aplanamiento, es decir, transforma el dato multidimensional a uno unidimensional, la cual contiene toda la información del dato recibido. Esto ayuda a reducir el esfuerzo computacional realizado por la máquina.

2.3.1.6. Capa Dense o Full Connected

Esta capa tiene la particularidad de poseer una conexión completa con la capa anterior, es decir, todas sus neuronas se encuentran conectadas con las neuronas de la capa anterior. La cantidad de activaciones que posee se calcula por medio de una multiplicación matricial.

2.3.1.7. Activación ReLU (Rectified Linear Unit)

Esta función de activación se encarga de recibir los valores resultantes de la capa de convolución o la capa Dense, reemplazando los valores negativos por cero, con el propósito de agregar no linealidad al modelo, eliminando la relación proporcional entre la salida y la entrada. Se encuentra denotada por la siguiente función, donde "z" es el valor del dato recibido:

$$R(z) = \max(0, z) \quad (1)$$

Dando como resultado lo observado en la figura N°6.

Figura N°7: Ejecución de la función ReLU
Fuente: Medrano (2019)

2.3.1.8. Capa de salida y activación Softmax

Esta última capa realiza el proceso de clasificación, utilizando la función de activación Softmax. Esta función permite asignar un valor probabilístico entre 0 y 1 a cada clase, indicando que tan probable es que la imagen o dato ingresado pertenezca a algunas de las clases en específico. Además, la suma de todas las probabilidades de cada salida es igual a 1. Esta función Softmax se encuentra denotada por la siguiente función, donde "z" es el valor de la salida de las capas ocultas y "k" es el número de clases en el modelo implementado.

$$F(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K}(e^{z_k})} \quad (2)$$

Por lo tanto, se puede concluir que el diseño de la arquitectura de una CNN es un tema complejo, dado que requiere definir la cantidad de capas, el tipo de capas y la cantidad de conexiones entre ellas, para obtener la funcionabilidad deseada. Por ello, como solución se han creado los modelos de CNN, los cuales poseen una arquitectura definida, ofreciendo una efectividad y precisión alta en aplicaciones como reconocimiento de patrones y clasificación de imágenes. Esto se debe a que se trata de modelos pre-entrenados, además existen métodos como el data augmentation y Transfer Learning que permiten mejorar la etapa de entrenamiento de la red. De los modelos existentes se seleccionaron dos para el desarrollo del proyecto de tesis, los cuales se presentan a continuación.

2.3.1.9. ResNet50

Según Gómez-Ríos et al. (2019) ResNet es un modelo de red neuronal convolucional propuesto en el 2016 por He et al. El cual, no posee una profundidad de capas fija y es dependiente del número de módulos que se utilicen, además

permite un tamaño de entrada máximo de 224 x 224. Su arquitectura está compuesta por un bloque base llamado “residual block”, la cantidad de capas que posee depende del número asignado conjuntamente con su nombre como son los casos de ResNet152, ResNet101 y ResNet50. Este último cuenta con 50 capas y su bloque base está compuesto por tres convoluciones secuenciales una convolución de 1x1, una convolución de 3x3 y una convolución de 1x1. Asimismo, una conexión que une a la entrada de la primera convolución con la salida de la tercera convolución. En la figura N°8 se puede observar el bloque base del modelo ResNet50.

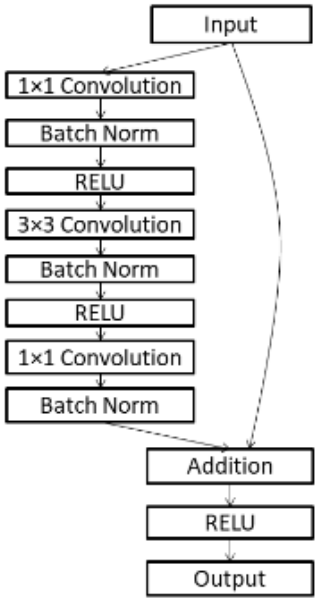


Figura N°8: Residual Block de ResNet50
Fuente: Sultana et al. (2018)

2.3.1.10. InceptionV3

Según Gómez-Ríos et al. (2019). Es un modelo de red neuronal convolucional introducido en el 2015 con el nombre GoogleNet por Szegedy et al. Contiene un total de 48 capas y puede clasificar hasta 1000 clases diferentes; Permite un tamaño de entrada máxima de 299 x 299, además su módulo base consta de cuatro ramas en paralelo, este módulo base contiene una convolución de 1x1 seguida de dos convoluciones de 3x3, una convolución de 1x1, otra convolución 3x3, un pooling, una convolución 1x1 y por último una convolución 1x1, siendo la salida del módulo base la concatenación de las cuatro ramas. InceptionV3 consta de un total de 10 módulos bases y estos son modificados ligeramente mientras la red se hace más

profundidad para evitar un mayor esfuerzo computacional. En la figura N°9 se muestra el modelo base de InceptionV3.

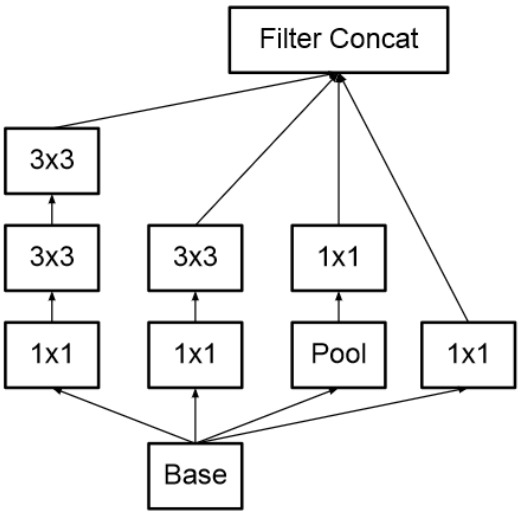


Figura N°9: Módulo base de InceptionV3
Fuente: Szegedy et al. (2016)

2.3.1.11. Métricas de la clasificación:

Para conocer la capacidad de los modelos de CNN en realizar la tarea de clasificación binaria, existen herramientas como la matriz de confusión, que permite visualizar el desempeño del modelo implementado, la matriz está compuesta por cuatro columnas, que representan las posibilidades que se pueden obtener. En la figura N°10 se puede observar la matriz de confusión con sus cuatro posibilidades.

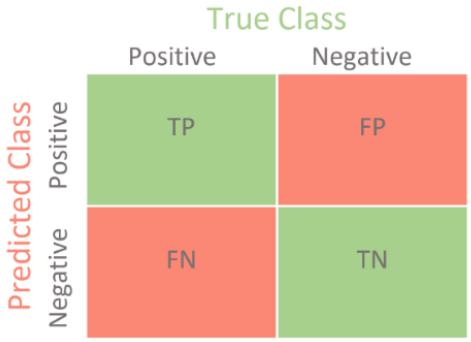


Figura N°10: Matriz de confusión para una clasificación binaria
Fuente: Mohajon (2020)

- Verdadero Positivo (TP: True Positive): Ocurre cuando el dato ha sido predicho de manera correcta como positivo.
- Verdadero Negativo (TN: True Negative): Ocurre cuando el dato ha sido predicho de manera correcta como negativo.

- Falso Positivo (FP: False Positive): Ocurre cuando el dato ha sido predicho como positivo, pero en realidad es un caso negativo.
- Falso Negativo (FN: False Negative): Ocurre cuando el dato ha sido predicho como negativo, pero en realidad es un caso positivo.

Asimismo, existen métricas que proporcionan una mayor información sobre el desempeño la red neuronal. Para ello, se emplean ecuaciones, las cuales usan los valores que proporciona la matriz de confusión. Además, estas métricas permiten obtener una comparación más precisa entre diferentes modelos de redes neuronales convolucionales.

Estas métricas se definen a continuación.

- Accuracy: Hace referencia la exactitud, respecto a lo cerca que se encuentra el resultado de ofrecer una medición de valor verdadero. Se obtiene por medio de la siguiente ecuación:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

- Precision: Permite medir la calidad del algoritmo implementado en la tarea de clasificación, dándote a conocer los verdaderos positivos entre todos los resultados positivos obtenidos. Se obtiene por medio de la siguiente ecuación:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

- Recall: Permite conocer qué proporción del total de casos positivos son correctamente predichos como positivos por el algoritmo. Esta métrica se obtiene de la ecuación:

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

- Specificity: Permite conocer qué proporción del total de casos negativos son correctamente predichos como negativos por el algoritmo. Para obtener este parámetro se realiza la siguiente ecuación:

$$Specificity = \frac{TN}{TN + FP} \tag{6}$$

- F1-score: Combina la Precision con Recall para obtener un solo parámetro que los represente a los dos, se obtiene realizando una media armónica entre las dos métricas, tal como se observa en la siguiente ecuación.

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (7)$$

2.3.2. Diagnóstico de Neumonía

Para determinar el diagnóstico de neumonía, generalmente se requiere de una radiografía de tórax, previo de una exploración física donde el médico evalúa los síntomas del paciente como son tos, fiebre y dificultad para respirar. Sumado a ello, se realizan otras pruebas como el hemograma para determinar cuál es el tipo de agente que está provocando la enfermedad y el tipo de severidad de la misma. En caso del diagnóstico de neumonía asociada al COVID-19, se sigue todo el proceso antes mencionado, pero con el conocimiento de que el coronavirus es el agente que ocasiona la neumonía.

A continuación, se define que es la neumonía, para comprender mejor el cómo afecta al cuerpo humano y el porqué de la importancia para su rápido tratamiento.

2.3.2.1. Neumonía

Es una infección que inflama los sacos aéreos de uno o ambos pulmones. Los sacos aéreos se pueden llenar de líquido o pus (material purulento), lo que provoca tos con flema o pus, fiebre, escalofríos y dificultad para respirar, su falta o tardío tratamiento puede llevar a la muerte. Entre los agentes que provocan esta enfermedad están diversos microrganismos, como bacterias, virus y hongos (Mayo Clinic, 2018).

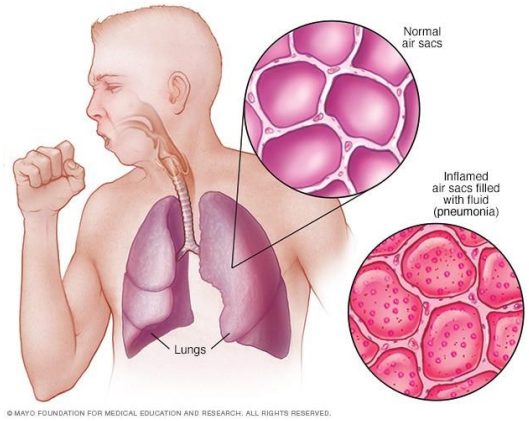


Figura N°11: Afección a pulmones por la neumonía
Fuente: (Mayo Clinic, 2018)

Como se mencionó antes, el médico especialista requiere de radiografía de tórax para poder diagnosticar la neumonía, así como el daño causado por parte de esta a los pulmones. Por ello, se explica a continuación su concepto y el cómo esta prueba ayuda a detectar enfermedades pulmonares como la neumonía.

2.3.2.2. Radiografía de tórax

Es una prueba rápida e indolora que genera imágenes de las estructuras internas del tórax, en especial de los huesos. Los haces de rayos X pasan a través del cuerpo y se absorben en diferentes cantidades según la densidad del material a través del cual pasan. Los materiales densos, como huesos y metales, aparecen de color blanco, mientras que el aire en los pulmones aparece de color negro. Además, la grasa y los músculos aparecen como sombras de color gris. (Mayo Clinic, 2020). Esto se puede visualizar en la figura N°12 donde se muestra distintos tipos de neumonía (Bacteriana, Viral y COVID-19) y además un caso normal.

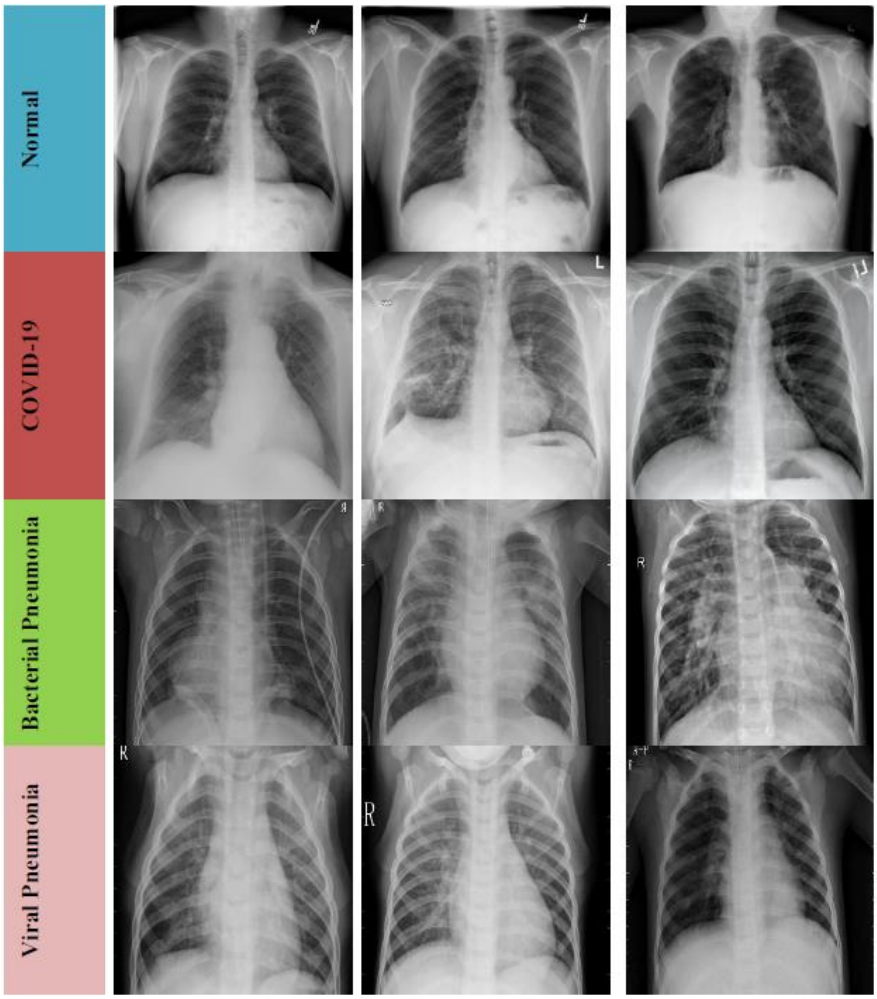


Figura N°12: Radiografías de tórax de cuatro casos diferentes
Fuente: Narin et al. (2020)

2.4. Definición de términos básicos

A continuación, se presentan los términos básicos para una mejor comprensión del proyecto de tesis realizado.

2.4.1. Aprendizaje supervisado

Matich (2001) menciona que el aprendizaje supervisado es aquel aprendizaje que es controlado por un agente externo (supervisor, maestro), este determina la respuesta que genera el algoritmo a partir de los datos de entrada (p. 19).

2.4.2. Data balanceada

El termino data balanceada, hace referencia a que cantidad de datos con la cual se cuenta, debe poseer una misma cantidad para cada una de sus categorías o clases con las cuales se desea realizar el proceso de clasificación. Cabe mencionar que el tener una data balanceada permite mejorar significativamente el rendimiento de los algoritmos utilizados.

2.4.3. Python

Es un lenguaje de programación interpretado, es decir, no requiere compilar el código fuente para compilarlo, está orientado a objetos de alto nivel y posee una semántica dinámica. Su sintaxis hace énfasis en la legibilidad del código, lo que facilita su depuración, por ende, favorece la productividad. (Alvarez, 2003)

2.4.4. TensorFlow

Ertam y Aydın (2017) señala que TensorFlow es una librería de código abierto desarrollada por Google para la computación numérica. Actualmente es utilizada también para el desarrollo de redes neuronales artificiales. Además, provee una interfaz para expresar los algoritmos de aprendizaje de la máquina como también de una aplicación para ejecutar estos algoritmos (p. 755).

2.4.5. Keras

Es una interfaz de programación de aplicaciones (API, por sus siglas en inglés) de alto nivel perteneciente a TensorFlow desde el año 2017, para construir y entrenar modelos de aprendizaje profundo reduciendo el tamaño del código en la implementación del algoritmo (TensorFlow, 2020).

2.4.6. COVID-19

Según la Organización Mundial de la Salud (2019), es una enfermedad infecciosa causada por el nuevo coronavirus, la notificación de su descubrimiento fue recibida al conocerse el estallido del brote de este virus en Wuhan (China), a finales de diciembre del 2019.

2.4.7. Transfer Learning

Es una técnica de aprendizaje empleada en Deep Learning, utilizada comúnmente en modelos de redes neuronales pre-entrenados o en redes neuronales demasiado profundas; consiste en modificar la arquitectura de la red neuronal agregando nuevas capas, las cuales serán las únicas en ser entrenadas, mientras que el resto de capas serán "congeladas" para mantener el valor del "peso" obtenido por el pre-entrenamiento. Esta técnica tiene como objetivo, obtener una red neuronal mucho más eficiente en el aspecto de clasificación de imágenes, así como proporcionar un menor esfuerzo computacional en el proceso de entrenamiento (POCHO COSTA, 2019).

2.4.8. Data Augmentation

De acuerdo con Gómez-Rios et al. (2019), esta técnica de aprendizaje que consiste en aumentar la cantidad de datos de forma artificial, como es por ejemplo girar la imagen, realizar un zoom, cambiar el brillo, reflejar la imagen, entre otros. Obteniéndose un conjunto mayor de imágenes para las etapas de entrenamiento y validación de la red neuronal. (p. 1173).

2.5. Diseño de la Investigación

2.5.1. Variables de investigación

Para el desarrollo de este proyecto de tesis se establecieron las siguientes variables:

Variable independiente:

Modelos de redes neuronales convolucionales

Variable dependiente:

Diagnóstico de neumonía asociada al COVID-19

2.5.2. Tipo y Método de investigación

La investigación es cuantitativa de tipo experimental y se empleó el método empírico para el desarrollo de la misma.

Es de enfoque cuantitativo, porque la investigación tuvo un proceso sistemático, rígido y ordenado; que va desde plantear y delimitar el problema de investigación hasta analizar los resultados obtenidos por medio de comprobación numérica, para comprobar la relación causa-efecto entre las variables de la investigación.

Esta investigación consistió en la evaluación del desempeño en la clasificación de radiografías de tórax, por medio de la experimentación de tres modelos de redes neuronales convolucionales; con el fin de detectar las radiografías de tórax positivas a la neumonía asociada al COVID-19. Por consiguiente, el diseño de esta investigación es de tipo experimental. Respecto a ello, Hernández Sampieri (2014) define al diseño experimental, como aquel que manipula intencionalmente la variable independiente, con el fin de conocer su influencia sobre la variable dependiente comprobando dicha influencia por medio del análisis de los resultados obtenidos (p. 129). Por lo tanto, se infiere que el conjunto de modelos de redes neuronales convolucionales es la variable manipulada, la cual servirá de apoyo al proceso de diagnóstico de neumonía asociada al COVID-19.

El método de investigación es empírico porque se basa en la experiencia adquirida por medio de las pruebas de experimentación realizadas por métodos cuantitativos y la recolección de información a través de las variables.

Por último, las muestras utilizadas son de tipo no probabilístico, porque la selección esta orientada a la elección del investigador. Estas son imágenes de radiografías de tórax que fueron adquiridas por medio de dos bases de datos; la primera de la Sociedad Italiana de Radiología Médica e Intervencionista (SIRM) a través de su página web. y la segunda, de una base de datos llamada “Chest X-ray (Covid-19 & Pneumonia)”, la cual se encuentra en la plataforma Kaggle.

2.5.3. Técnicas e Instrumentos de recolección de datos

La recolección de datos en la presente investigación fue en base a fuentes secundarias externas, es decir, los datos fueron recolectados por otros autores o entidades. Por lo cual, esta investigación no posee técnica ni instrumento de recolección de datos ya que no se ha hecho uso de fuentes primarias. Cabe recalcar, que las fuentes secundarias externas utilizadas fueron obtenidas de las plataformas web Kaggle y SIRM, de las cuales, se descargaron las bases de datos que sirvieron para generar el dataset del proyecto de tesis.

2.5.4. Procedimiento para la recolección de datos

Para el procedimiento de recolección se emplearon imágenes de radiografías de tórax, las cuales fueron obtenidas de las bases de datos pertenecientes a la Sociedad Italiana de Radiología Médica e Intervencionista (2020) y al usuario Prashant Patel en la plataforma Kaggle (2020). La razón por la cual se escogió esta base de datos se debió a que es una recolección de otras bases de datos importantes, como son las de Josep Paul Cohen y Paul Mooney, las cuales fueron utilizadas por los autores que fueron citados como antecedentes de investigación para este proyecto de tesis. El motivo por el cual se decidió emplear bases de datos externas, se debió a lo complicado que representa la obtención de estas imágenes médicas por cuenta propia.

CAPÍTULO III: DESARROLLO DEL PROYECTO

En este capítulo se muestra el desarrollo del proyecto de tesis, para desarrollar y validar los objetivos definidos en el capítulo 1. En la figura N°13, se muestra a continuación el diagrama de bloques general del desarrollo del proyecto de tesis, el cual está dividido en siete bloques:

En el primer bloque, se desarrolló la adquisición de datos y generación del dataset, para la adquisición de imágenes de radiografías de tórax se utilizaron las bases de datos de la página web del SIRM y del usuario Patel en la plataforma Kaggle, y para la generación se utilizaron algoritmos con el lenguaje de programación Python por medio del editor de desarrollo Jupyter Notebook.

En el segundo bloque, se desarrolló la etapa del preprocesamiento de imágenes por medio de un algoritmo utilizando Keras con TensorFlow en Python, para definir las características con los cuales contaran las imágenes al momento de la implementación y entrenamiento de los tres modelos de CNN. Además, se consideró la inclusión de la técnica de aprendizaje data augmentation al momento de realizar el preprocesamiento con el fin de conocer la utilidad de esta técnica en el proceso de entrenamiento.

Del tercer al quinto bloque, se desarrollaron la implementación y entrenamiento de cada uno de los modelos de CNN de forma independiente. Se requirió de técnicas de aprendizaje para la implementación de los modelos ResNet50 e InceptionV3 al ser modelos muy profundos. Por otro lado, se realizaron dos pruebas por proceso de entrenamiento de cada modelo implementado para analizar la utilidad de data augmentation en dicho proceso.

En el sexto bloque se realizaron pruebas de predicción de los tres modelos de CNN, cada uno de ellos cuenta con dos casos (con data augmentation y sin data augmentation), dando un total de 6 pruebas de predicción realizadas.

Finalmente, en el séptimo y último bloque, se realizó la comparación de los resultados obtenidos por medio de las matrices de confusión y las métricas de medición de cada modelo de CNN implementados y entrenados previamente.

Las pruebas de predicción y los resultados, indicados en los bloques 6 y 7, respectivamente, se encuentran en el capítulo 4.

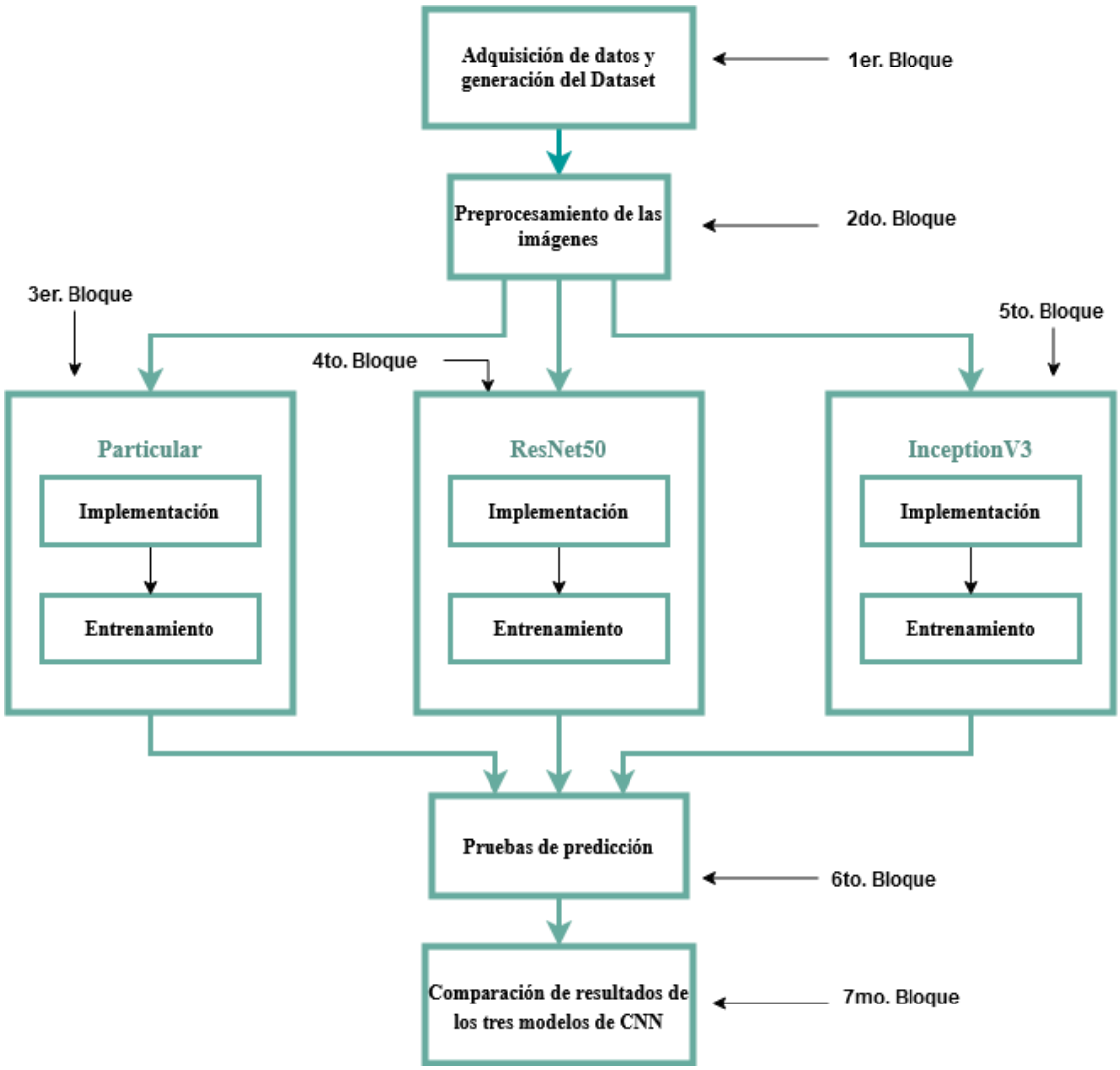


Figura N°13: Diagrama de bloques general
Fuente: Elaboración propia (2020)

3.1. Adquisición de datos y generación del Dataset

En este proyecto de tesis se utilizaron dos bases de datos diferentes para la adquisición de imágenes de radiografías de tórax de casos con COVID-19 y casos normales. Estas bases de datos pertenecen a la Sociedad Italiana de Radiología Médica e Intervencionista (SIRM) y al usuario Prashant Patel, quien tiene su base de datos publicada en la plataforma Kaggle. Siendo ambas bases de datos de acceso gratuito. La decisión de escoger la base de datos de Patel correspondió a que se encuentra conformada por las bases de datos de Cohen, perteneciente a la plataforma GitHub (2020) y Mooney en la plataforma Kaggle (2018), las cuales fueron utilizadas por autores como Sethi et al. (2020), Ozturk et al. (2020) y Narin et al. (2020) para la realización de sus proyectos de investigación dado que las consideran fuentes confiables de información.

La implementación del algoritmo para generar el dataset personal, así como los algoritmos posteriores requirió la instalación de la interfaz gráfica de usuario (GUI) llamada Anaconda Navigator, dado que permite la creación de entornos personalizados, en los cuales se puede ejecutar distintos editores de desarrollo tal como son Jupyter Notebook, Spyder y Visual Studio Code, como también instalar diferentes versiones del lenguaje de programación Python con sus librerías respectivas. La elección de Jupyter Notebook como editor de desarrollo se debe a la experiencia personal adquirida en el empleo de editores de desarrollo con Python, además del empleo que le da Sethi et al. (2020) para el desarrollo de su proyecto de investigación.

El algoritmo implementado para la generación del dataset es llamado “Create Dataset”, el cual tuvo el siguiente desarrollo; en primer lugar, se decidió importar las librerías pandas, os y shutil, las cuales sirven para realizar operaciones con archivos y carpetas, luego de ello se inicia con la selección y extracción de imágenes de radiografías de tórax positivas a COVID-19, de la base de datos del SIRM (véase Figura N°14). Para ello, se importó el archivo “COVID-19.metadata.xlsx”, el cual por medio del comando “pd.read_excel” es leído por el algoritmo y por medio del comando print(df.shape) muestra que este archivo contiene un total de 219 imágenes divididas en 4 columnas. La importación de librerías y uso del metadata se visualiza en la figura N°15.

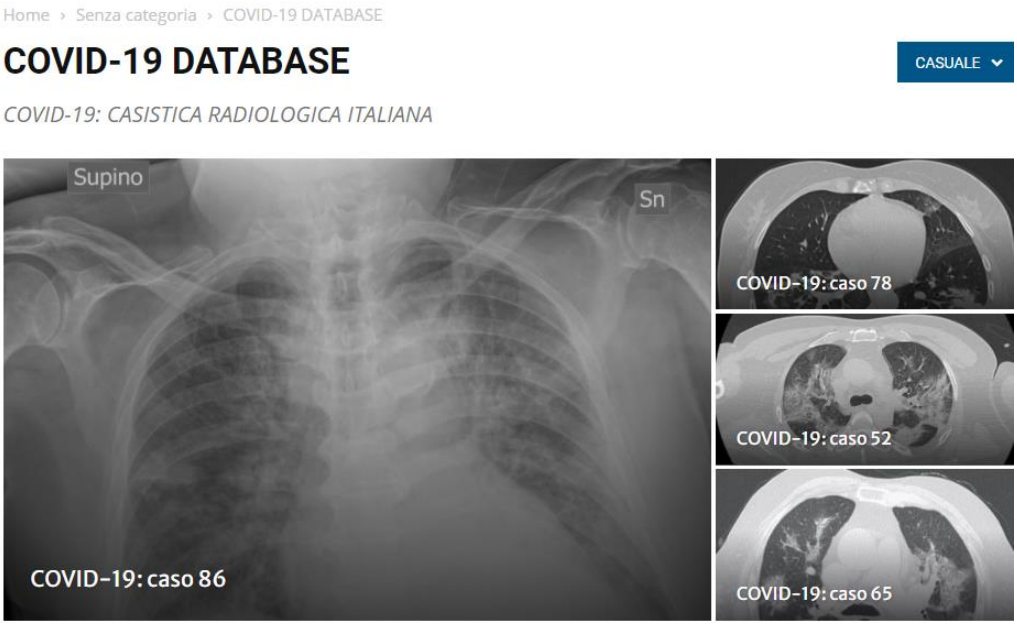


Figura N°14: Base de datos del SIRM
Fuente: Elaboración propia (2020)

```
import pandas as pd
import os
import shutil
```

```
FILE_PATH = "D:/TESIS/COVID-19 Radiography Database/COVID-19.metadata.xlsx"
IMAGES_PATH = "D:/TESIS/COVID-19 Radiography Database/COVID-19"
```

```
df = pd.read_excel(FILE_PATH)
print(df.shape)
```

```
(219, 4)
```

Figura N°15: Importación de librerías y uso del metadata
Fuente: Elaboración propia (2020)

Para poder visualizar la distribución de la base datos del SIRM se utilizó el comando df.tail(), además se copió todo el contenido de la base de datos del SIRM a una carpeta llamada COVID-19 que se encuentra dentro de la carpeta "Dataset", utilizando el comando "shutil.copytree". La distribución de la base de datos del SIRM se muestra en la figura N°16.

```
df.tail()
```

	FILE NAME	FORMAT	SIZE	URL
214	COVID-19(215)	PNG	1024*1024	https://www.sirm.org/2020/03/28/covid-19-caso-64/
215	COVID-19(216)	PNG	1024*1024	https://www.sirm.org/2020/03/28/covid-19-caso-65/
216	COVID-19(217)	PNG	1024*1024	https://www.sirm.org/2020/03/28/covid-19-caso-65/
217	COVID-19(218)	PNG	1024*1024	https://www.sirm.org/2020/03/28/covid-19-caso-66/
218	COVID-19(219)	PNG	1024*1024	https://www.sirm.org/2020/03/28/covid-19-caso-66/

```
TARGET_DIR = "D:/TESIS/Dataset/COVID-19"
```

```
shutil.copytree(IMAGES_PATH,TARGET_DIR)
```

```
'D:/TESIS/Dataset/COVID-19'
```

Figura N°16: Distribución de la base de datos del SIRM
Fuente: Elaboración propia (2020)

Dado que la base de datos del SIRM contiene tanto imágenes de radiografías de tórax como imágenes de tomografías, tal como se observó en la figura N°14, se decidió solo mantener en el dataset las imágenes correspondientes a radiografías de tórax, dado que estas fueron el tipo de muestras elegidas para el desarrollo del proyecto de tesis. Por lo cual, se pasó a tener en el dataset un total de 86 imágenes médicas de las 219 iniciales.

Luego de ello, se copió de forma manual las imágenes de radiografías de tórax asociadas al COVID-19 de la base de datos de Patel, siendo un total de 576 imágenes.

Estas imágenes agregadas a las 86 antes obtenidas, dio un total de 662 imágenes para la carpeta llamada “COVID-19” correspondiente al dataset personal.

En segundo lugar, se utilizó de nuevo la base de datos de Patel y se copió de forma aleatoria 662 imágenes de radiografías de casos normales hacia otra carpeta. Para ello, se utilizaron los comandos “random.shuffle” y “shutil.copy2”; el primero selecciona una imagen de forma aleatoria, mientras que el segundo copia la imagen seleccionada, manteniendo su información, hacia la carpeta destino, la cual es la carpeta llamada “Normal”, la cual almacena las imágenes de radiografías de tórax de casos normales.

Se decidió almacenar una misma cantidad de imágenes en cada carpeta (COVID-19 y Normal) para poseer una data balanceada, con el fin de mejorar los resultados luego del entrenamiento realizado a los modelos de redes neuronales convolucionales.

En la figura N°17 se presenta el algoritmo encargado de obtener las 662 imágenes de radiografías de tórax para la carpeta “Normal” usando los comandos antes mencionados.

```
#se recolectaron las imágenes
#de las demas bases de datos
#la carpeta covid-19 contiene 662 imágenes
```

```
#imagenes radiografícas de tórax normales
import random
KAGGLE_FILE_PATH = "D:/TESIS/Data/train/NORMAL"
TARGET_NORMAL_DIR = "D:/TESIS/Dataset/Normal"
```

```
image_names = os.listdir(KAGGLE_FILE_PATH)
random.shuffle(image_names)

#Se escoge la misma cantidad de imagenes que los casos positivos
for i in range(662):
    image_name = image_names[i]
    image_path = os.path.join(KAGGLE_FILE_PATH,image_name)
    target_path = os.path.join(TARGET_NORMAL_DIR, image_name)
    shutil.copy2(image_path, target_path)
```

Figura N°17: Algoritmo para copiar aleatoriamente las imágenes de radiografías de tórax
Fuente: Elaboración propia (2020)

En tercer y último lugar, se dividió la cantidad de imágenes del dataset en tres conjuntos (Train, Val y Test). La cantidad de imágenes en cada conjunto se basó en el criterio de equilibrio de aprendizaje que utiliza Artola (2020), el cual consiste en dividir el total de datos en un 80% para el conjunto de entrenamiento y 20% para el de test. Del mismo modo, se dividió la cantidad de imágenes del conjunto de entrenamiento, manteniendo un

80%, mientras que el 20% fue asignado para el conjunto de validación (Val). Esto con el fin de evitar deficiencias en el rendimiento de la CNN como son underfitting y overfitting; el primero se da por poseer una cantidad no muy grande de datos, provocando que la CNN sea incapaz generalizar lo aprendido. El segundo ocurre al realizar el entrenamiento con imágenes demasiadas específicas sobre las clases que se desean analizar dando como resultado que la red no pueda ser capaz de distinguir adecuadamente.

La cantidad de imágenes en cada conjunto quedo distribuida de la siguiente manera: El conjunto de entrenamiento cuenta con 848 imágenes, el conjunto de test 264 imágenes y el conjunto de validación 212 imágenes. El algoritmo implementado para realizar dicha distribución requirió de la importación de las librerías shutil y random; la primera permite copiar y mover archivos de un directorio a otro y la segunda da un orden aleatorio para la acción que se esté ejecutando. Por lo cual, primero se copió de forma aleatoria 132 imágenes de cada clase para formar el conjunto de test, luego se realizó el mismo proceso con 424 imágenes por clase para obtener el conjunto de entrenamiento y finalmente las imágenes restantes pasaron a pertenecer al conjunto de validación. En la figura N°18 se puede observar el algoritmo para la distribución aleatoria de imágenes de radiografías de tórax de casos normales, mientras que en la figura N°19 se observa el mismo algoritmo, pero para los casos de COVID-19.

```
import random
NORMAL_FILE_PATH = "D:/TESIS/Dataset/Normal"
TARGET_TEST_DIR = "D:/TESIS/Dataset/Test/Normal"
TARGET_TRAIN_DIR = "D:/TESIS/Dataset/Train/Normal"
```

```
image_names1 = os.listdir(NORMAL_FILE_PATH)
random.shuffle(image_names1)

for i in range(132):
    image_name1 = image_names1[i]
    image_path1 = os.path.join(NORMAL_FILE_PATH,image_name1)
    target_path1 = os.path.join(TARGET_TEST_DIR, image_name1)
    shutil.move(image_path1, target_path1)
```

```
image_names2 = os.listdir(NORMAL_FILE_PATH)
random.shuffle(image_names2)

for i in range(424):
    image_name2 = image_names2[i]
    image_path2 = os.path.join(NORMAL_FILE_PATH,image_name2)
    target_path2 = os.path.join(TARGET_TRAIN_DIR, image_name2
    shutil.move(image_path2, target_path2)
```

Figura N°18: Algoritmo para la distribución aleatoria de las imágenes de radiografías de casos normales
Fuente: Elaboración propia (2020)

```
import random
COVID_FILE_PATH = "D:/TESIS/Dataset/COVID-19"
TARGET_TESTC_DIR = "D:/TESIS/Dataset/Test/COVID-19"
TARGET_TRAINC_DIR = "D:/TESIS/Dataset/Train/COVID-19"
```

```
image_names3 = os.listdir(COVID_FILE_PATH)
random.shuffle(image_names3)

for i in range(132):
    image_name3 = image_names3[i]
    image_path3 = os.path.join(COVID_FILE_PATH,image_name3)
    target_path3 = os.path.join(TARGET_TESTC_DIR, image_name3)
    shutil.move(image_path3, target_path3)
```

```
image_names4 = os.listdir(COVID_FILE_PATH)
random.shuffle(image_names4)

for i in range(424):
    image_name4 = image_names4[i]
    image_path4 = os.path.join(COVID_FILE_PATH,image_name4)
    target_path4 = os.path.join(TARGET_TRAINC_DIR, image_name4)
    shutil.move(image_path4, target_path4)
```

Figura N°19: Algoritmo para la distribución aleatoria de las imágenes de radiografías de casos de COVID-19
Fuente: Elaboración propia (2020)

Quedando finalmente distribuidos en tres carpetas, cada una conteniendo 2 subcarpetas que corresponden a las clases de COVID-19 y Normal. Esto se muestra en la figura N°20.

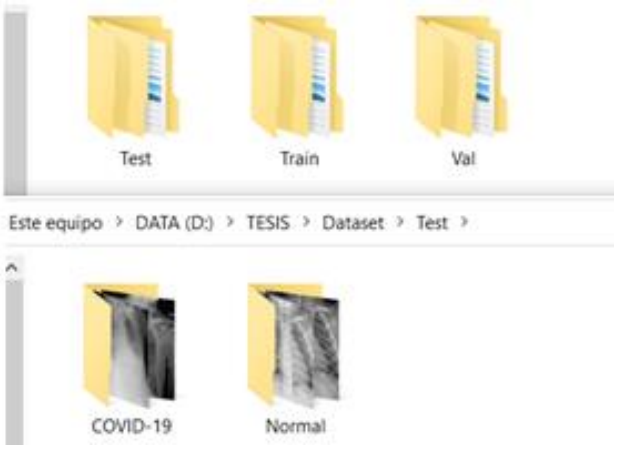


Figura N°20: Contenido del dataset y sus carpetas pertinentes
Fuente: Elaboración propia (2020)

3.2. Preprocesamiento de las imágenes

Previamente al preprocesamiento de las imágenes, se declararon las variables donde se almacenaron los directorios de imágenes de los conjuntos entrenamiento y validación, así como los hiperparámetros que se utilizaron en la etapa de entrenamiento de los modelos de CNN, los cuales son “epocas” y “batch_size”. Por un lado, “epocas” hace referencia

al número de veces que se va itinerar sobre los conjuntos de entrenamiento y validación durante el proceso de entrenamiento y, por otro lado, "batch_size" es la cantidad de imágenes que se le dará a la computadora para procesar en cada uno de los "pasos". El término "pasos", hace referencia al número de veces que se va procesar la información en cada una de las "epocas".

En la figura N°21 se muestran las variables de almacenamiento y los hiperparámetros definidos para de los modelos de CNN.

```
#ALMACENAMIENTO DE DIRECTORIOS
train_data_dir = 'D:/TESIS/Dataset/Train'
validation_data_dir = 'D:/TESIS/Dataset/Val'

#HIPERPARÁMETROS
epocas=40
batch_size = 64
```

Figura N°21: Variables de almacenamiento e hiperparámetros de los modelos de CNN
Fuente: Elaboración propia (2020)

Para realizar el preprocesamiento de imágenes se requirió importar la librería TensorFlow, esta para su uso necesitó la instalación de CUDA 10.1 de NVIDIA (Véase Anexo N°2) y cuDNN 7.5, la versión de Tensorflow instalada fue la 2.3.1, la cual contiene la API de Keras en la versión 2.4.3. Dado que desde la versión 2.0 de TensorFlow la API Keras ya viene incluida. Keras contiene librerías para la implementación de modelos de CNN, preprocesamiento de imágenes, entrenamiento de redes neuronales entre otros, que fueron útiles para el desarrollo de este proyecto de tesis. Entre estas librerías se decidió importar ImageDataGenerator, la cual realizó el preprocesamiento de las imágenes por medio de argumentos declarados en su algoritmo. Además, esta librería permite realizar técnicas de aprendizaje como es data augmentation, la cual fue utilizada para evaluar su utilidad en la mejora del proceso de entrenamiento por medio de dos pruebas (con data augmentation y sin data augmentation). Esta técnica también fue utilizada por Artola (2019) y Sharma et al. (2020), debido a que requerían ampliar la cantidad de imágenes de sus bases de datos porque las consideraban no muy amplias.

La declaración de la función ImageDataGenerator() con sus argumentos para preprocesar las imágenes, así como la aplicación de data augmentation se detallan a continuación:

- Se utilizó la librería ImageDataGenerator para crear un generador para los conjuntos de imágenes de entrenamiento y validación, los cuales son llamados "train_datagen" y "val_datagen", respectivamente.

- Dentro de “train_datagen” se declararon los siguientes argumentos:
  - “rescale=1. / 255”, el cual reescala el rango de pixeles que va 0 a 255 a un rango de 0 a 1, a este proceso se le conoce como normalización.
  - “rotation_range”, genera las imágenes rotadas en el ángulo sexagesimal indicado.
  - “width_shift_range”, genera las imágenes aumentando su ancho de la imagen.
  - “height_shift_range”, genera las imágenes cambiando la altura de la imagen inicialmente ingresada.
  - “shear_range”, genera las imágenes inclinadas para que la CNN aprenda a trabajar con imágenes descuadradas.
  - “zoom_range”, realiza un zoom a las imágenes dependiendo en que parte de la imagen se encuentre la información requerida.
  - “horizontal_flip”, invierte la imagen de forma horizontal con el fin de que la CNN pueda distinguir direccionalidad.
  - “fill_mode”, es declarado como ‘nearest’, por ser su valor por defecto para llenar los puntos o pixeles fuera del límite de la imagen.
- Dentro de “val_datagen” solo se declaró “rescale=1. / 255”, porque el objetivo fue darle las imágenes del conjunto de validación sin cambios a la CNN.

Estos argumentos declarados permiten la aplicación de data augmentation en el proceso de entrenamiento. Para el entrenamiento sin aplicar data augmentation solo se requirió declarar el argumento “rescale” que se encarga de normalizar los datos. En la tabla N°3, se observa los argumentos declarados para la aplicación de data augmentation como también para su inaplicación.

Tabla N°3: Valores de los argumentos declarados

Argumentos Declarados	Valores	
	Con Data Augmentation	Sin Data Augmentation
rescale	1.0 / 255	1.0 / 255
rotation_range	40	
width_shift_range	0.2	
height_shift_range	0.2	
shear_range	0.2	
zoom_range	0.2	
horizontal_flip	True	
fill_mode	nearest	

Fuente: Elaboración propia (2020)

Una vez definidos los generadores “train_datagen” y “val_datagen”, fueron utilizados para definir las variables train_generator y val_generator, respectivamente, estas

variables almacenan las imágenes ya preprocesadas, además requieren la declaración de argumentos para posteriormente ser utilizadas en la implementación y entrenamiento de los modelos de CNN. Los argumentos declarados fueron los siguientes:

- Se declaró “target_size”, para redimensionar el tamaño de las imágenes (largo y ancho), siendo 224x244 para los modelos Particular y ResNet50, y 299x299 para el modelo InceptionV3.
- Se llamó al parámetro “batch_size” que es igual a 64.
- Se declaró el tipo de clase con el argumento “class_mode”, para que este sea “categorical” en los tres modelos de redes neuronales convolucionales

La diferencia que existe entre las dimensiones declaradas por el argumento “target_size” de los modelos de CNN, se debe a que inicialmente todos iban a poseer las mismas dimensiones (224x224), las cuales son las dimensiones máximas que admite el modelo ResNet50. Sin embargo, al momento de realizar las pruebas de predicción se obtuvo errores con el modelo InceptionV3. Por lo cual, se decidió trabajar ese modelo con sus dimensiones máximas admisibles (299x299), obteniendo mejores resultados con el cambio realizado. En la figura N°22 se observa el algoritmo de preprocesamiento de imágenes del modelo Particular sin data augmentation.

Cabe mencionar que el preprocesamiento de imágenes es suma importancia porque permite mejorar el proceso de entrenamiento de los modelos de CNN.

```
train_datagen = ImageDataGenerator(rescale=1. / 255)

val_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical')

val_generator = val_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical')
```

```
Found 848 images belonging to 2 classes.
Found 212 images belonging to 2 classes.
```

Figura N°22: Algoritmo para preprocesamiento de las imágenes del modelo Particular sin data augmentation
Fuente: Elaboración propia (2020)

3.3.Modelo de Red Neuronal Convolucional Particular

En esta sección se desarrolló la implementación y entrenamiento del modelo Particular de red neuronal convolucional utilizando el API de Keras con la librería Tensorflow en el lenguaje de programación Python.

3.3.1. Implementación

La arquitectura del modelo Particular, constó de 10 capas, las cuales se presentan en la figura N°23. Asimismo, en la figura N°24 se representa el resumen del modelo implementado.

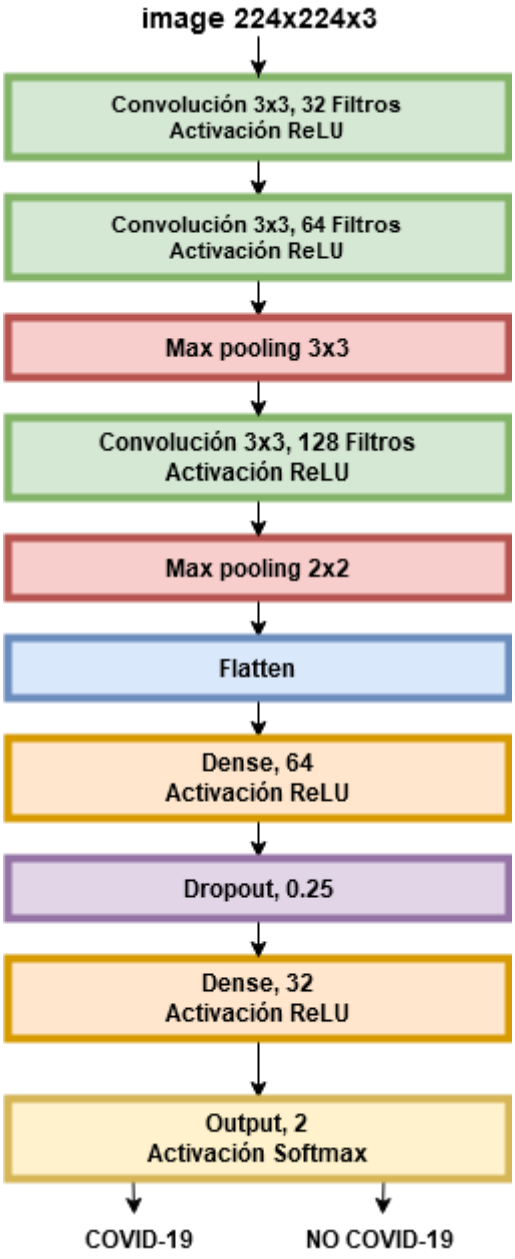


Figura N°23: Arquitectura del modelo Particular
Fuente: Elaboración Propia (2020)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 32)      896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 64)      18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 74, 74, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 175232)            0
_________________________________________________________________
dense (Dense)                (None, 64)                11214912
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
output (Dense)               (None, 2)                 66
=================================================================
Total params: 11,310,306
Trainable params: 11,310,306
Non-trainable params: 0
```

Figura N°24: Implementación del modelo de Particular detallado.
Fuente: Elaboración Propia (2020)

Este diseño de la arquitectura del modelo Particular se obtuvo aplicando el método heurístico de prueba y error. Para ello, se tomó como base el ejemplo proporcionado por Coluga y Ángel (2019), sobre como implementar una red neuronal convolucional utilizando Keras y TensorFlow, además se tomó en cuenta lo realizado por Ozturk et al. (2020), lo cual fue aplicar diferentes cantidades de filtros con el fin de mejorar la extracción de características de la imagen, reduciendo el tamaño de las dimensiones en el proceso; así como también la elección de la activación Softmax en la capa de salida para el proceso de clasificación binaria. Asimismo, se consideró la evaluación realizada por parte de Sharma et al. (2020) sobre uso de la capa Dropout para reducir la posibilidad de sufrir un sobreajuste, conocido también como overfitting, en el proceso de entrenamiento de los modelos de redes neuronales convolucionales evaluados.

Como se mencionó al inicio, este modelo Particular es una red neuronal convolucional que cuenta con 10 capas implementadas, las cuales son detalladas a continuación, para una mejor comprensión sobre la función que realiza cada una de ellas.

- Se implementó la primera capa de convolución, de 32 filtros con kernels que poseen un tamaño de 3x3, y utiliza una activación ReLU para convertir a ceros

los valores negativos resultantes. El proceso de esta capa consiste en tomar grupos cercanos de pixeles en la imagen de entrada y realizar producto escalar con el kernel. Obteniendo 32 matrices de salida, dando un total de 1.605,632 neuronas en esta capa.

- Se implementó una segunda capa de convolución, de 64 filtros, con kernels de 3x3 de tamaño, de igual modo que la anterior capa, esta contiene una activación ReLU, esta vez lo que ingresa son 32 matrices de 224x224x1. Esta capa procede igual que la anterior, obteniendo 64 matrices de salida de dimensiones 224x224x32 cada una y dando un total de 3.211,264 neuronas para esta capa.
- Se aplicó una capa Max pooling de tamaño 3x3, es decir, en vez de tomar pixel por pixel, se tomó de 3 en 3 tanto en altura como en ancho dando como resultado una reducción de dimensiones equivalente a 74x74, reduciendo también la cantidad de neuronas en esta capa a 356,807.
- Se implementó una tercera y última capa de convolución de 128 filtros con kernels que poseen un tamaño de 3x3 y activación ReLU. Esta vez se obtienen 128 matrices de 74x74x64 como dimensiones, dando un total de 700,928 neuronas para esta capa.
- Se aplicó otra capa Max pooling, pero de tamaño 2x2, dando como resultado la reducción de dimensiones a 37x37 pixeles, obteniendo un total de 175,232 neuronas en esta capa.
- Se agregó la capa Flatten para “aplanar” las dimensiones, es decir se pasa de trabajar en tridimensional a ser unidimensional, tiene la misma cantidad de neuronas que la capa anterior, es decir, 175,232.
- Se agregó una capa Dense de 64 neuronas que es una capa tradicional, con activación ReLU, esta capa se encarga de conectar todas las neuronas de la capa pasada con las neuronas de su capa.
- Se agregó una capa Dropout de 0.25, para evitar el overfitting, esto implica desactivar el 25% de la cantidad total de neuronas recibidas para que la red neuronal convolucional no realice un aprendizaje errado al “memorizar” una determinada cantidad de rutas.
- Se agregó una capa Dense de 32 neuronas que es una capa tradicional con activación ReLU, reduciendo aún más la cantidad de neuronas conectadas una a la otra.

- Se finalizó agregando la capa de Salida, la cual posee una activación Softmax, esta activación asigna a cada una de las dos clases un valor probabilístico. La salida con el mayor valor probabilístico será la predicción realizada por el modelo. Esta predicción indicará si la imagen evaluada se trata de un caso "Positivo a COVID-19" o es un caso "NO COVID"; la activación Softmax es ideal para este proyecto de tesis, dado que permite realizar una clasificación de 2 a más clases.

Una vez implementada la arquitectura del modelo Particular, se compiló la red utilizando la función model.compile(). Esta función permite la compilación por medio de argumentos como la función de optimización, el optimizador a usar y la métrica de evaluación. En este caso la función de optimización será "Loss" de tipo "categorical_crossentropy", con un optimizador como "Adam" y se asignó "accuracy" como métrica de evaluación.

3.3.2. Entrenamiento

Luego de la implementación del modelo particular de red neuronal convolucional, se pasó a la etapa de entrenamiento. Para ello, se realizaron los siguientes pasos:

- Se definió el valor del hiperparámetro "steps_per_epoch", también conocido como "pasos". Este se obtuvo dividiendo la cantidad de muestras del conjunto de entrenamiento (848 imágenes) con el parámetro batch_size (64), dando un valor aproximado de 13.
- Se definió el valor del hiperparámetro "validation_steps", también conocido como "pasos de validación", este parámetro indica que al final de cada "epoca" se va correr una cantidad de "pasos de validación", utilizando el conjunto de validación, con el fin de visualizar si el aprendizaje del algoritmo implementado está siendo el correcto. La cantidad de "pasos de validación" se obtuvo dividiendo la cantidad de muestras del conjunto de validación (212 imágenes) entre el hiperparámetro "batch_size", dando un valor aproximado de 3.
- Se declaró la función "fit", el cual realiza el proceso de entrenamiento. Para ello, se requiere declarar como argumentos de la función los hiperparámetros que se definieron previamente.
- Finalmente, se ejecutó el entrenamiento del modelo Particular, el cual se realizó dos veces; la primera, con data augmentation y la segunda, sin data augmentation con el fin de conocer la utilidad e influencia de esta técnica de aprendizaje en el

proceso de entrenamiento. Se almacenaron las pruebas de entrenamiento realizado con los nombres de “cnnp.h5” y “pesos.h5” para el entrenamiento sin data augmentation y “cnnpda.h5” y “pesosda.h5” para el entrenamiento con data augmentation.

El tiempo que tomó el proceso de entrenamiento con data augmentation fue un total de 56 minutos con 38 segundos, mientras que sin data augmentation fue de 50 minutos con 10 segundos. Las pruebas de predicción realizadas para corroborar lo aprendido por el modelo Particular, así como, las gráficas de los procesos de entrenamiento se muestran en el capítulo 4.

En las figuras N°25 y N°26 se observan los procesos de entrenamiento con data augmentation y sin data augmentation , respectivamente.

```
Inicio del entrenamiento
Epoch 1/40
13/13 [==============================] - 87s 7s/step - loss: 0.2128 - accuracy: 0.9209
acy: 0.9844
Epoch 2/40
13/13 [==============================] - 99s 8s/step - loss: 0.1508 - accuracy: 0.9337
acy: 0.9948
Epoch 3/40
13/13 [==============================] - 85s 7s/step - loss: 0.1943 - accuracy: 0.9324
acy: 0.9896
Epoch 4/40
13/13 [==============================] - 87s 7s/step - loss: 0.1552 - accuracy: 0.9477
acy: 0.9896
Epoch 5/40
13/13 [==============================] - 85s 7s/step - loss: 0.1784 - accuracy: 0.9399
acy: 0.9844
Epoch 6/40
13/13 [==============================] - 83s 6s/step - loss: 0.1432 - accuracy: 0.9579
acy: 0.9896
Epoch 7/40
13/13 [==============================] - 82s 6s/step - loss: 0.1681 - accuracy: 0.9426
acy: 0.9635
Epoch 8/40
13/13 [==============================] - 81s 6s/step - loss: 0.1408 - accuracy: 0.9515
acy: 0.9896
Epoch 9/40
13/13 [==============================] - 80s 6s/step - loss: 0.1423 - accuracy: 0.9515
acy: 0.9844
Epoch 10/40
13/13 [==============================] - 82s 6s/step - loss: 0.1315 - accuracy: 0.9554
acy: 0.9792
Epoch 11/40
13/13 [==============================] - 81s 6s/step - loss: 0.1299 - accuracy: 0.9592
acy: 0.9844
Epoch 12/40
13/13 [==============================] - 86s 7s/step - loss: 0.1377 - accuracy: 0.9507
acy: 0.9948
Epoch 13/40
13/13 [==============================] - 82s 6s/step - loss: 0.1478 - accuracy: 0.9490
acy: 0.9792
Epoch 14/40
13/13 [==============================] - 81s 6s/step - loss: 0.1591 - accuracy: 0.9311
acy: 0.9062
Epoch 15/40
```

Figura N°25: Proceso de entrenamiento con data augmentation del modelo Particular
Fuente: Elaboración Propia (2020)

```
Inicio del entrenamiento
Epoch 1/40
13/13 [==============================] - 113s 9s/step - loss: 1.3115 - accuracy: 0.6288
racy: 0.9479
Epoch 2/40
13/13 [==============================] - 93s 7s/step - loss: 0.1969 - accuracy: 0.9235
acy: 0.9688
Epoch 3/40
13/13 [==============================] - 91s 7s/step - loss: 0.1239 - accuracy: 0.9515
acy: 0.9896
Epoch 4/40
13/13 [==============================] - 92s 7s/step - loss: 0.0609 - accuracy: 0.9847
acy: 0.9896
Epoch 5/40
13/13 [==============================] - 91s 7s/step - loss: 0.0404 - accuracy: 0.9872
acy: 0.9792
Epoch 6/40
13/13 [==============================] - 93s 7s/step - loss: 0.0388 - accuracy: 0.9898
acy: 0.9740
Epoch 7/40
13/13 [==============================] - 108s 8s/step - loss: 0.0223 - accuracy: 0.9923
racy: 0.9844
Epoch 8/40
13/13 [==============================] - 88s 7s/step - loss: 0.0140 - accuracy: 0.9974
acy: 0.9896
Epoch 9/40
13/13 [==============================] - 91s 7s/step - loss: 0.0310 - accuracy: 0.9898
acy: 0.9583
Epoch 10/40
13/13 [==============================] - 93s 7s/step - loss: 0.0567 - accuracy: 0.9860
acy: 0.9688
Epoch 11/40
13/13 [==============================] - 94s 7s/step - loss: 0.0141 - accuracy: 0.9949
acy: 0.9740
Epoch 12/40
13/13 [==============================] - 91s 7s/step - loss: 0.0222 - accuracy: 0.9911
acy: 0.9792
Epoch 13/40
13/13 [==============================] - 90s 7s/step - loss: 0.0187 - accuracy: 0.9949
acy: 0.9896
Epoch 14/40
13/13 [==============================] - 101s 8s/step - loss: 0.0055 - accuracy: 0.9987
racy: 0.9740
Epoch 15/40
```

Figura N°26: Proceso de entrenamiento sin data augmentation del modelo Particular
Fuente: Elaboración propia (2020)

3.4. Modelo de Red Neuronal Convolucional ResNet50

En esta sección se desarrolló la implementación y entrenamiento del modelo ResNet50 utilizando el API de Keras con la librería Tensorflow en el lenguaje de programación Python.

3.4.1. Implementación

De acuerdo con Gómez-Ríos et al. (2019), ResNet50 es un modelo de red neuronal convolucional que cuenta con 50 capas, las cuales se encuentran pre-entrenadas. Este modelo admite una dimensión de entrada máxima de 224x224 pixeles con una profundidad igual a 3 y puede realizar una predicción de hasta 1,000 clases. Además, según la web de Keras (Keras, 2020), el modelo ResNet50 cuenta con un total de 25.636,712 parámetros para entrenamiento, de los cuales 53,120 son no entrenables o

fijos. Igualmente, Keras tiene disponible la implementación del algoritmo para declarar el modelo ResNet50 con sus respectivos argumentos. Estos argumentos fueron declarados de forma que ResNet50 sea una red pre-entrenada con la base de datos ImageNet, admita unas dimensiones de entrada de 224x244 pixeles y elimine la capa predicción de 1000 salida con el argumento "include_top=False". Esto se puede visualizar en la figura N°27, previamente se debió llamar al modelo ResNet50 importando la librería ResNet50 con keras.applications.

```
image_input = Input(shape=(224, 224, 3))

pre_trained_resnet50 = ResNet50(input_tensor=image_input,weights='imagenet',
                   include_top=False)

pre_trained_resnet50.summary()
```

```
conv5_block3_2_conv (Conv2D)    (None, 7, 7, 512)    2359808     conv5_block3_1_relu[0][0]
__________________________________________________________________________________________________
conv5_block3_2_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block3_2_conv[0][0]
__________________________________________________________________________________________________
conv5_block3_2_relu (Activation (None, 7, 7, 512)    0           conv5_block3_2_bn[0][0]
__________________________________________________________________________________________________
conv5_block3_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624     conv5_block3_2_relu[0][0]
__________________________________________________________________________________________________
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block3_3_conv[0][0]
__________________________________________________________________________________________________
conv5_block3_add (Add)          (None, 7, 7, 2048)   0           conv5_block2_out[0][0]
                                                                 conv5_block3_3_bn[0][0]
__________________________________________________________________________________________________
conv5_block3_out (Activation)   (None, 7, 7, 2048)   0           conv5_block3_add[0][0]
==================================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
```

Figura N°27: Implementación del modelo ResNet50 con declaración de sus argumentos
Fuente: Elaboración Propia (2020)

Sin embargo, la arquitectura del modelo ResNet50 (Véase Anexo N°3), sigue siendo demasiada profunda como para entrenar todas sus capas y pesos. Por ello, se aplicó la técnica de aprendizaje llamada transfer learning, la cual consistió en modificar la arquitectura del modelo ResNet50 agregando nuevas capas para que pueda realizar una predicción de solo dos clases, además de aprovechar el preentrenamiento de sus pesos con la base de datos ImageNet que se realizó al momento de importar el modelo con sus argumentos declarados, así reducir el esfuerzo computacional por parte de la laptop convencional utilizada.

En total se agregaron 6 capas al modelo ResNet50 para que pueda ser capaz discernir entre las clases existentes y proporcionar una respuesta adecuada al momento de realizar una prueba de predicción. La técnica de aprendizaje transfer learning fue utilizada por Sethi et al. (2020) y Narin et al. (2020), para poder entrenar los diferentes modelos de CNN utilizados en sus trabajos de investigación.

Cada una de las capas agregadas se detallan a continuación, para comprender el funcionamiento del modelo ResNet50 cuando se aplicó transfer learning, además se explica el porqué de la decisión del número capas agregadas.

- Se agregó una capa Flatten para "aplanar" las dimensiones de los datos recibidos, con lo cual los datos pasan a tener una sola dimensión. Esta capa posee la misma cantidad de neuronas que la capa anterior.
- Se agregó una capa Dense de 128 neuronas con activación ReLU, sus neuronas se conectan en su totalidad con las neuronas de la capa Flatten, mientras que la función ReLU convierte los resultados negativos en ceros.
- Se agregó una capa Dense de 64 neuronas con activación ReLU, reduce la cantidad de capas conectadas respecto a la capa anterior y con la función ReLU pasa los valores negativos a ceros.
- Se agregó una capa Dense de 32 neuronas con activación ReLU, reduce la cantidad de capas conectadas respecto a la capa anterior y con la función ReLU pasa los valores negativos a ceros.
- Se agregó una capa Dropout de 0.5, para evitar el overfitting, esto implica desactivar el 50% de la cantidad total de neuronas recibidas con el objetivo de reducir las opciones de que la red neuronal convolucional aprenda erradamente.
- Se agregó una última capa con la función de activación Softmax, para asignar un valor probabilístico a cada salida, esto con el objetivo de que la predicción realizada sea dada por el mayor valor probabilístico resultante. Esta predicción puede indicar si se trata de un caso "Positivo a COVID-19" o de un caso "NO COVID".

La decisión de agregar 6 capas se debió a que esta adición está basada en la que presenta Narin et al. (2020) en su trabajo de investigación, cuando aplicó transfer learning a los modelos de CNN, esta adición significo el agregado de 3 nuevas capas a la arquitectura de cada uno de los modelos, estas capas agregadas fueron una capa Pooling, una capa Dense de 1024 neuronas con activación ReLU y una capa de salida con activación Softmax. Sin embargo, al replicar este mismo agregado de capas se obtuvo resultados muy deficientes. Por lo cual, se decidió modificarlo por una estructura de 6 nuevas capas: una capa Flatten, 3 capas Dense de 1024, 256 y 32 neuronas, una capa Dropout de 0.5 y la capa de salida con dos neuronas y activación Softmax. Esta estructura fue obtenida luego de aplicar el método heurístico de prueba

y error, estas pruebas fueron corroboradas por medio de las gráficas del proceso de entrenamiento, quedando finalmente la estructura que proporciono un mayor valor en la métrica de medición “accuracy” para el entrenamiento del modelo.

Para este modelo también se usó la función model.compile(), siendo sus argumentos de compilación declarados la función de optimización “Loss” de tipo “categorical_crossentropy”, con un optimizador como “Adam”, debido a que entre los 3 optimizadores probados (Adam, Adadelta y SGD), “Adam” fue el optimizador que mejor rendimiento proporcionó para este modelo, y finalmente como métrica de evaluación se asignó “accuracy”.

La implementación por medio de Keras con TensorFlow utilizando Python de las nuevas capas para el modelo ResNet50, así como su compilación asignando los argumentos de optimización y medición se muestran en la figura N°28.

```
last_layer = pre_trained_resnet50.layers[-1].output

r= Flatten(name='flatten')(last_layer)
r = Dense(1024, activation='relu', name='fc1')(r)
r = Dense(256, activation='relu', name='fc2')(r)
r = Dense(32, activation='relu', name='fc3')(r)
r = Dropout(0.5)(r)
predictions = Dense(2, activation='softmax', name='predictions')(r)
resnet50m = Model(image_input, predictions)
# Congela todas las capas excepto las 6 ultimas agregadas
for layer in pre_trained_resnet50.layers[:-6]:
    layer.trainable = False

#lr = 0.001 #learning_rate
#optimizer='Adam'
#optimizer=optimizers.Adam(lr=lr)
resnet50m.compile(loss='categorical_crossentropy',
                  optimizer='Adam',
                  metrics=['accuracy'])
resnet50m.summary()
```

conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	conv5_block3_add[0][0]
flatten (Flatten)	(None, 100352)	0	conv5_block3_out[0][0]
fc1 (Dense)	(None, 128)	12845184	flatten[0][0]
fc2 (Dense)	(None, 64)	8256	fc1[0][0]
fc3 (Dense)	(None, 32)	2080	fc2[0][0]
dropout_1 (Dropout)	(None, 32)	0	fc3[0][0]
predictions (Dense)	(None, 2)	66	dropout_1[0][0]

```
Total params: 36,443,298
Trainable params: 13,911,330
Non-trainable params: 22,531,968
```

Figura N°28: Implementación del modelo ResNet50 aplicando transfer learning
Fuente: Elaboración Propia (2020)

Mientras que, en la figura N°29 se muestra la arquitectura del modelo ResNet50 luego de la aplicación de transfer learning.

Cabe resaltar que el modelo ResNet50 se representa en la figura como un bloque único, para un mejor entendimiento de las capas agregadas.

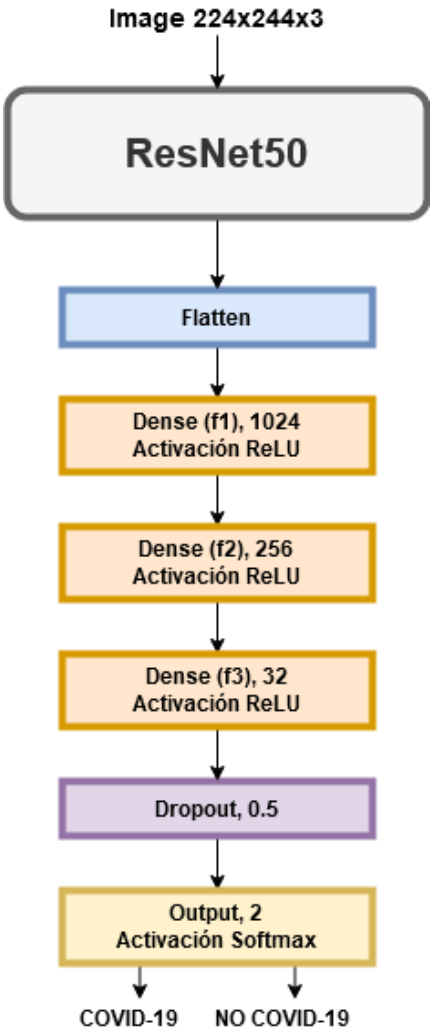


Figura N°29: Arquitectura del modelo ResNet50 luego de aplicar transfer learning
Fuente: Elaboración Propia (2020)

3.4.2. Entrenamiento

El proceso de entrenamiento del modelo ResNet50 utilizó los mismos hiperparámetros que el modelo Particular, es decir, posee los mismos números de “epocas”, “steps_per_epoch”, “validation_steps” y “batch_size”, además de las mismas dimensiones de entrada.

Para su entrenamiento, de igual forma que para el modelo Particular, se realizaron dos pruebas con el fin de evaluar la utilidad e influencia de la técnica de aprendizaje data

augmentation. Se utilizó la función “fit” con la declaración de sus hiperparámetros pertinentes. Los modelos entrenados con data augmentation y sin data augmentation se guardaron con los nombres de “resnet50da.h5” y “resnet50.h5”, respectivamente. Respecto al tiempo que tomó el proceso de entrenamiento en cada prueba; se indica que cuando se aplicó data augmentation, el tiempo total fue 73 minutos con 23 segundos, mientras que, cuando no se aplicó data augmentation el tiempo fue de 70 minutos con 45 segundos. Las pruebas de predicción realizadas para corroborar lo aprendido por el modelo ResNet50, así como, las gráficas de los procesos de entrenamiento se muestran en el capítulo 4.

Las figuras N°30 y N°31, muestran los procesos de entrenamiento con data augmentation y sin data augmentation , respectivamente.

```
Inicio del entrenamiento
Epoch 1/40
13/13 [==============================] - 89s 7s/step - loss: 1.5449 - accuracy: 0.5901
acy: 0.5000
Epoch 2/40
13/13 [==============================] - 84s 6s/step - loss: 0.5071 - accuracy: 0.7921
acy: 0.4948
Epoch 3/40
13/13 [==============================] - 90s 7s/step - loss: 0.4544 - accuracy: 0.7883
acy: 0.5052
Epoch 4/40
13/13 [==============================] - 88s 7s/step - loss: 0.3651 - accuracy: 0.8457
acy: 0.5208
Epoch 5/40
13/13 [==============================] - 87s 7s/step - loss: 0.3396 - accuracy: 0.8584
acy: 0.7969
Epoch 6/40
13/13 [==============================] - 86s 7s/step - loss: 0.4249 - accuracy: 0.8087
acy: 0.6406
Epoch 7/40
13/13 [==============================] - 84s 6s/step - loss: 0.3895 - accuracy: 0.8367
acy: 0.8125
Epoch 8/40
13/13 [==============================] - 86s 7s/step - loss: 0.3298 - accuracy: 0.8763
acy: 0.7760
Epoch 9/40
13/13 [==============================] - 89s 7s/step - loss: 0.3129 - accuracy: 0.8712
acy: 0.8438
Epoch 10/40
13/13 [==============================] - 85s 7s/step - loss: 0.3541 - accuracy: 0.8559
acy: 0.8385
Epoch 11/40
13/13 [==============================] - 84s 6s/step - loss: 0.3133 - accuracy: 0.8839
acy: 0.8698
Epoch 12/40
13/13 [==============================] - 83s 6s/step - loss: 0.2912 - accuracy: 0.8839
acy: 0.8125
Epoch 13/40
13/13 [==============================] - 83s 6s/step - loss: 0.3256 - accuracy: 0.8648
acy: 0.8229
Epoch 14/40
13/13 [==============================] - 79s 6s/step - loss: 0.3091 - accuracy: 0.8852
acy: 0.8646
```

Figura N°30: Proceso de entrenamiento con data augmentation del modelo ResNet50
Fuente: Elaboración Propia (2020)

```
Inicio del entrenamiento
Epoch 1/40
13/13 [==============================] - 70s 5s/step - loss: 2.2996 - accuracy: 0.5523 -
acy: 0.5000
Epoch 2/40
13/13 [==============================] - 65s 5s/step - loss: 0.4363 - accuracy: 0.8176 -
acy: 0.4896
Epoch 3/40
13/13 [==============================] - 67s 5s/step - loss: 0.2200 - accuracy: 0.9235 -
acy: 0.4844
Epoch 4/40
13/13 [==============================] - 64s 5s/step - loss: 0.1701 - accuracy: 0.9324 -
acy: 0.5052
Epoch 5/40
13/13 [==============================] - 66s 5s/step - loss: 0.1764 - accuracy: 0.9375 -
acy: 0.5156
Epoch 6/40
13/13 [==============================] - 68s 5s/step - loss: 0.1801 - accuracy: 0.9235 -
acy: 0.4896
Epoch 7/40
13/13 [==============================] - 67s 5s/step - loss: 0.1615 - accuracy: 0.9503 -
acy: 0.5052
Epoch 8/40
13/13 [==============================] - 66s 5s/step - loss: 0.1895 - accuracy: 0.9247 -
acy: 0.4948
Epoch 9/40
13/13 [==============================] - 67s 5s/step - loss: 0.1256 - accuracy: 0.9554 -
acy: 0.5156
Epoch 10/40
13/13 [==============================] - 67s 5s/step - loss: 0.1150 - accuracy: 0.9643 -
acy: 0.4948
Epoch 11/40
13/13 [==============================] - 69s 5s/step - loss: 0.1487 - accuracy: 0.9554 -
acy: 0.5156
Epoch 12/40
13/13 [==============================] - 74s 6s/step - loss: 0.1198 - accuracy: 0.9630 -
acy: 0.6510
Epoch 13/40
13/13 [==============================] - 71s 5s/step - loss: 0.0936 - accuracy: 0.9694 -
acy: 0.5990
Epoch 14/40
13/13 [==============================] - 71s 5s/step - loss: 0.1077 - accuracy: 0.9630 -
acy: 0.5729
```

Figura N°31: Proceso de entrenamiento sin data augmentation del modelo ResNet50
Fuente: Elaboración Propia (2020)

3.5. Modelo de Red Neuronal Convolucional InceptionV3

En esta sección se desarrolló la implementación y entrenamiento del modelo InceptionV3 utilizando el API de Keras con la librería Tensorflow en el lenguaje de programación Python.

3.5.1. Implementación

De acuerdo con Gómez-Ríos et al (2019), InceptionV3 es un modelo de red neuronal convolucional que cuenta con 48 capas, las cuales se encuentran pre-entrenadas. Este modelo puede admitir una dimensión de entrada de hasta 299x299 pixeles con 3 de profundidad y al igual que el modelo ResNet50 puede realizar una clasificación de hasta 1,000 clases. Según Keras cuenta con 23.851,784 parámetros de los cuales 34,432 son no entrenables. Dispone de un algoritmo de implementación con sus

respectivos argumentos dentro de Keras. Para la implementación del modelo InceptionV3 se decidió declarar los argumentos para que admita unas dimensiones de entrada de 299x299x3, además sea una red pre-entrenada en la base de datos ImageNet al igual que el modelo ResNet50 y elimine la capa de predicción de 1000 salidas al declarar el argumento "include_top=False".

En la figura N°32, se muestra el algoritmo utilizado para declarar el modelo InceptionV3 utilizando el lenguaje de programación Python, previamente se debió llamar el modelo InceptionV3 importando a la librería InceptionV3 con keras.applications.

```
#modelo base de Inception V3

image_input = Input(shape=(299, 299, 3))

pre_trained_inception = InceptionV3(input_tensor=image_input,weights='imagenet',
                       include_top=False)

pre_trained_inception.summary()
```

```
activation_367 (Activation)     (None, 8, 8, 320)    0           batch_normalization_367[0][0]

mixed9_1 (Concatenate)          (None, 8, 8, 768)    0           activation_369[0][0]
                                                                 activation_370[0][0]

concatenate_7 (Concatenate)     (None, 8, 8, 768)    0           activation_373[0][0]
                                                                 activation_374[0][0]

activation_375 (Activation)     (None, 8, 8, 192)    0           batch_normalization_375[0][0]

mixed10 (Concatenate)           (None, 8, 8, 2048)   0           activation_367[0][0]
                                                                 mixed9_1[0][0]
                                                                 concatenate_7[0][0]
                                                                 activation_375[0][0]
==================================================================================================
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
```

Figura N°32: Implementación del modelo InceptionV3 con la declaración de sus argumentos.
Fuente: Elaboración Propia (2020)

El modelo InceptionV3 como se mencionó antes posee 48 capas, siendo al igual que el modelo ResNet50 es una red neuronal convolucional demasiada profunda para poder entrenar a todas sus capas y asignar los pesos a cada una de ellas. Por lo que, también decidió aplicar la técnica de aprendizaje transfer learning y aprovechar el pre-entrenamiento que posee con la base de datos ImageNet, la cual fue realizada cuando se declararon los argumentos al implementar el modelo.

La aplicación de transfer learning en el modelo InceptionV3, significó la adición de 6 nuevas capas, las cuales fueron añadidas de forma secuencial. Esta adición se basó en lo realizado con el modelo ResNet50, siendo la base de todo, la aplicación de transfer learning por parte de Narin et al. (2020), a los modelos implementados en su trabajo de investigación. Se realizó al igual que en el modelo ResNet50, la aplicación del

método heurístico de prueba y error, hasta obtener los valores pertinentes para cada capa que permita obtener una medida de "accuracy" aceptable en el proceso de entrenamiento, dado que las capas que agregó Narin tampoco proporcionaron los resultados positivos en el modelo InceptionV3.

Las 6 capas agregadas con la aplicación de transfer learning consistió en una capa Flatten, tres capas Dense de 128, 64 y 32 neuronas, respectivamente, una capa Dropout de 0.5 y una capa de salida de dos neuronas con activación Softmax.

Se explica a continuación cada una de las 6 nuevas capas incorporadas a la arquitectura del modelo InceptionV3 luego de aplicar la técnica de aprendizaje transfer learning.

- Se agregó la capa Flatten para "aplanar" los datos recibidos, con lo cual la red neuronal convolucional entrenará con datos en una sola dimensión. Posee la misma cantidad de neuronas que la capa anterior.
- Se agregó una capa Dense de 128 neuronas con activación ReLU, sus neuronas se conectan en su totalidad con las neuronas de la capa Flatten, mientras que la función ReLU convierte los resultados negativos en ceros.
- Se agregó una capa Dense de 64 neuronas con activación ReLU, reduce la cantidad de capas conectadas respecto a la capa anterior y con la función ReLU pasa los valores negativos a ceros.
- Se agregó una capa Dense de 32 neuronas con activación ReLU, reduce la cantidad de capas conectadas respecto a la capa anterior y con la función ReLU pasa los valores negativos a ceros.
- Se agregó una capa Dropout de 0.5, para evitar el overfitting, esto implica desactivar el 50% de la cantidad total de neuronas recibidas para que la red neuronal convolucional no aprenda una forma o determinadas formas, obtenido un aprendizaje errado.

- Se agregó una última capa con la función de activación Softmax, para asignar un valor probabilístico a cada salida, esto con el objetivo de que la predicción realizada sea dada por el mayor valor probabilístico resultante. Esta predicción puede indicar si se trata de un caso "Positivo a COVID-19" o de un caso "NO COVID".

Finalmente, para compilar este modelo se decidió utilizar los mismos argumentos que para los modelos anteriores, es decir, la función de optimización será "Loss" de tipo "categorical_crossentropy", además para mejorar el proceso de entrenamiento se definió como optimizador "Adam" y la métrica de evaluación utilizada fue "accuracy".

En la figura N°33, se presenta la implementación por medio de Keras con TensorFlow utilizando Python de las nuevas capas para el modelo InceptionV3, así como su compilación con sus argumentos de optimización y medición asignados.

```
#Implementación del modelo InceptionV3 utilizando Transfer Learning
#Modelo modificado de InceptionV3

last_layer = pre_trained_inception.layers[-1].output

ic= Flatten(name='flatten')(last_layer)
ic = Dense(128, activation='relu', name='fc1')(ic)
ic = Dense(64, activation='relu', name='fc2')(ic)
ic = Dense(32, activation='relu', name='fc3')(ic)
ic = Dropout(0.5)(ic)
predictions = Dense(2, activation='softmax', name='predictions')(ic)
InceptionB = Model(image_input, predictions)
# Congela todas las capas excepto las 6 ultimas agregadas
for layer in pre_trained_inception.layers[:-6]:
    layer.trainable = False

InceptionB.compile(loss='categorical_crossentropy',
                   optimizer='adam',metrics=['accuracy'])
InceptionB.summary()
```

```
                                                    concatenate_7[0][0]
                                                    activation_375[0][0]

flatten (Flatten)            (None, 131072)         0           mixed10[0][0]

fc1 (Dense)                  (None, 128)            16777344    flatten[0][0]

fc2 (Dense)                  (None, 64)             8256        fc1[0][0]

fc3 (Dense)                  (None, 32)             2080        fc2[0][0]

dropout_5 (Dropout)          (None, 32)             0           fc3[0][0]

predictions (Dense)          (None, 2)              66          dropout_5[0][0]
==================================================================================================
Total params: 38,590,530
Trainable params: 16,787,938
Non-trainable params: 21,802,592
```

Figura N°33: Implementación del modelo InceptionV3 aplicando transfer learning
Fuente: Elaboración Propia (2020)

Mientras que, en la figura N°34 se muestra la arquitectura del modelo InceptionV3 luego de la aplicación de transfer learning.

Cabe resaltar que el modelo InceptionV3 se representa en la figura como un bloque único, para un mejor entendimiento de las capas agregadas.

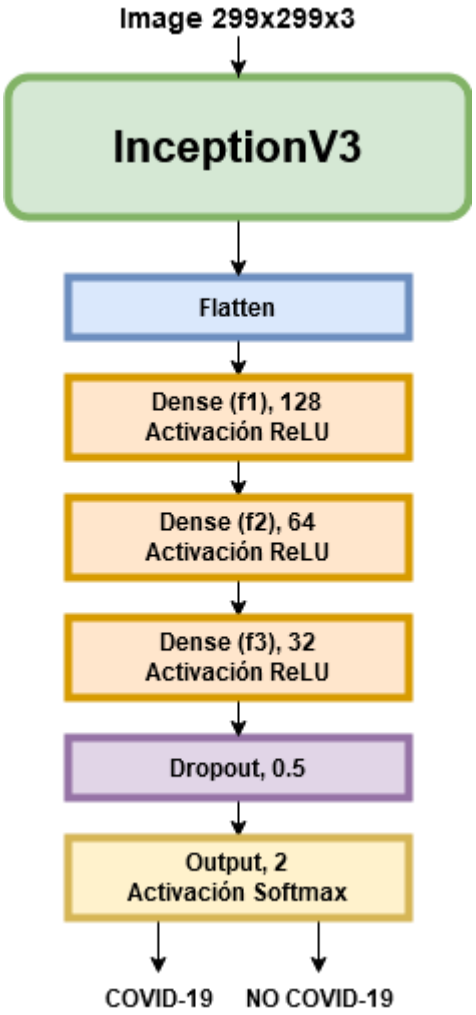


Figura N°34: Arquitectura del modelo InceptionV3 luego de aplicar transfer learning
Fuente: Elaboración Propia (2020)

3.5.2. Entrenamiento

El entrenamiento del modelo de InceptionV3 utilizó los mismos hiperparámetros que los modelos anteriores, las dimensiones de entrada para este modelo fueron de 299x299 pixeles con 3 de profundidad. Su entrenamiento, también contó con dos pruebas para evaluar la utilidad e influencia de data augmentation. Para ello, se utilizó la función “fit” y se declararon sus hiperparámetros antes mencionados.

Finalmente, se ejecutó el entrenamiento del modelo InceptionV3, realizando los procesos de entrenamiento con data augmentation y sin data augmentation, estos fueron almacenados con los nombres “inceptionv3da.h5” y “inceptionv3.h5”, respectivamente.

Respecto al tiempo que tomo el proceso de entrenamiento en cada prueba, se puede indicar que en la prueba cuando se aplicó data augmentation el tiempo total fue de 67 minutos con 17 segundos, mientras que en la prueba donde no se aplicó data

augmentation el tiempo fue de 69 minutos con 50 segundos. Las pruebas de predicción realizadas para corroborar el aprendizaje del modelo InceptionV3, así como, las gráficas de los procesos de entrenamiento se muestran en el capítulo 4.

En las figuras N°35 y N°36, se muestran los procesos de entrenamiento con data augmentation y sin data augmentation, respectivamente.

```
Inicio del entrenamiento
Epoch 1/40
13/13 [==============================] - 88s 7s/step - loss: 3.5414 - accuracy: 0.6952 -
acy: 0.9167
Epoch 2/40
13/13 [==============================] - 90s 7s/step - loss: 0.8633 - accuracy: 0.8571 -
acy: 0.9740
Epoch 3/40
13/13 [==============================] - 89s 7s/step - loss: 0.3538 - accuracy: 0.8929 -
acy: 0.9948
Epoch 4/40
13/13 [==============================] - 90s 7s/step - loss: 0.2129 - accuracy: 0.8980 -
acy: 0.9688
Epoch 5/40
13/13 [==============================] - 85s 7s/step - loss: 0.1809 - accuracy: 0.9043 -
acy: 0.9896
Epoch 6/40
13/13 [==============================] - 85s 7s/step - loss: 0.1908 - accuracy: 0.9362 -
acy: 0.9948
Epoch 7/40
13/13 [==============================] - 91s 7s/step - loss: 0.1863 - accuracy: 0.9069 -
acy: 0.9896
Epoch 8/40
13/13 [==============================] - 121s 9s/step - loss: 0.2817 - accuracy: 0.9107
racy: 0.9896
Epoch 9/40
13/13 [==============================] - 83s 6s/step - loss: 0.1480 - accuracy: 0.9311 -
acy: 0.9844
Epoch 10/40
13/13 [==============================] - 91s 7s/step - loss: 0.1272 - accuracy: 0.9363 -
acy: 0.9896
Epoch 11/40
13/13 [==============================] - 91s 7s/step - loss: 0.1168 - accuracy: 0.9643 -
acy: 0.9896
Epoch 12/40
13/13 [==============================] - 91s 7s/step - loss: 0.1149 - accuracy: 0.9554 -
acy: 0.9948
Epoch 13/40
13/13 [==============================] - 87s 7s/step - loss: 0.0921 - accuracy: 0.9656 -
acy: 0.9948
Epoch 14/40
13/13 [==============================] - 92s 7s/step - loss: 0.0907 - accuracy: 0.9668 -
acy: 0.9948
Epoch 15/40
```

Figura N°35: Proceso de entrenamiento con data augmentation del modelo InceptionV3
Fuente: Elaboración Propia (2020)

```
Inicio del entrenamiento
Epoch 1/40
13/13 [==============================] - 87s 7s/step - loss: 2.3340 - accuracy: 0.8444
acy: 0.9948
Epoch 2/40
13/13 [==============================] - 80s 6s/step - loss: 0.7843 - accuracy: 0.9617
acy: 0.9896
Epoch 3/40
13/13 [==============================] - 75s 6s/step - loss: 0.6006 - accuracy: 0.9732
acy: 0.9948
Epoch 4/40
13/13 [==============================] - 75s 6s/step - loss: 0.2538 - accuracy: 0.9847
acy: 0.9948
Epoch 5/40
13/13 [==============================] - 74s 6s/step - loss: 0.0737 - accuracy: 0.9923
acy: 0.9948
Epoch 6/40
13/13 [==============================] - 81s 6s/step - loss: 0.0687 - accuracy: 0.9911
acy: 0.9948
Epoch 7/40
13/13 [==============================] - 84s 6s/step - loss: 0.1434 - accuracy: 0.9872
acy: 0.9948
Epoch 8/40
13/13 [==============================] - 86s 7s/step - loss: 0.0212 - accuracy: 0.9936
acy: 0.9948
Epoch 9/40
13/13 [==============================] - 86s 7s/step - loss: 0.1413 - accuracy: 0.9892
acy: 0.9948
Epoch 10/40
13/13 [==============================] - 86s 7s/step - loss: 0.0347 - accuracy: 0.9936
acy: 0.9948
Epoch 11/40
13/13 [==============================] - 79s 6s/step - loss: 0.0272 - accuracy: 0.9949
acy: 0.9948
Epoch 12/40
13/13 [==============================] - 80s 6s/step - loss: 0.0018 - accuracy: 1.0000
acy: 0.9948
Epoch 13/40
13/13 [==============================] - 79s 6s/step - loss: 0.0112 - accuracy: 0.9974
acy: 0.9948
Epoch 14/40
13/13 [==============================] - 81s 6s/step - loss: 0.0011 - accuracy: 1.0000
acy: 0.9948
Epoch 15/40
```

Figura N°36: Proceso de entrenamiento sin data augmentation del modelo InceptionV3
Fuente: Elaboración Propia (2020)

CAPÍTULO IV: PRUEBAS Y RESULTADOS

En este capítulo se presentan las pruebas de predicción y resultados obtenidos de cada uno de los tres modelos de redes neuronales convolucionales desarrollados en el capítulo 3. Para ello, se utilizó el conjunto de imágenes pertenecientes a la carpeta llamada "Test". Las pruebas de predicción sirvieron para comprobar el funcionamiento de los modelos entrenados por medio del desempeño que estos tienen al momento de diferenciar adecuadamente entre las clases, dando a conocer cuando se trata de un caso "POSITIVO A COVID-19" o un caso "NO COVID-19". Asimismo, los resultados son mostrados en tres secciones; en el primero se muestran los procesos de entrenamiento, los cuales son visualizados por medio de gráficas que utilizaron como parámetros de medición, la función de optimización "Loss" y la métrica de medición "accuracy". En el segundo, se muestran las tablas comparativas, las cuales contienen información relacionada con las métricas de evaluación, y en el tercero y último, se muestran las matrices de confusión, en las cuales se visualizan numéricamente la cantidad de aciertos y fallos que tuvieron cada uno de los modelos de CNN evaluados.

Cabe mencionar que se realizaron las pruebas de predicción y la obtención de resultados para los procesos de entrenamiento con data augmentation y sin data augmentation, con el fin de corroborar la influencia y utilidad que tiene esta técnica de aprendizaje en el desempeño de los tres modelos implementados en el proyecto de tesis.

4.1. Pruebas y resultados del modelo Particular

En la sección 4.1.1. se presentan las pruebas de predicción para cada caso del modelo Particular, mientras que en la sección 4.1.2. se presentan las gráficas del proceso de entrenamiento y los resultados obtenidos por medio de las métricas de evaluación y matrices de confusión.

4.1.1. Pruebas de predicción

Se realizaron las pruebas de predicción para verificar el adecuado funcionamiento del modelo Particular, por medio de un algoritmo (Véase Anexo N°5). Este algoritmo utilizó las imágenes almacenadas en la carpeta "Test" para realizar dichas pruebas.

En la figura N°37 se muestra la prueba de predicción realizada a una imagen de radiografía de tórax correspondiente a un caso positivo a neumonía asociada COVID-19, además se realizó una comparativa entre las predicciones obtenidas cuando se aplicó data augmentation y cuando no fue aplicada al proceso de entrenamiento.

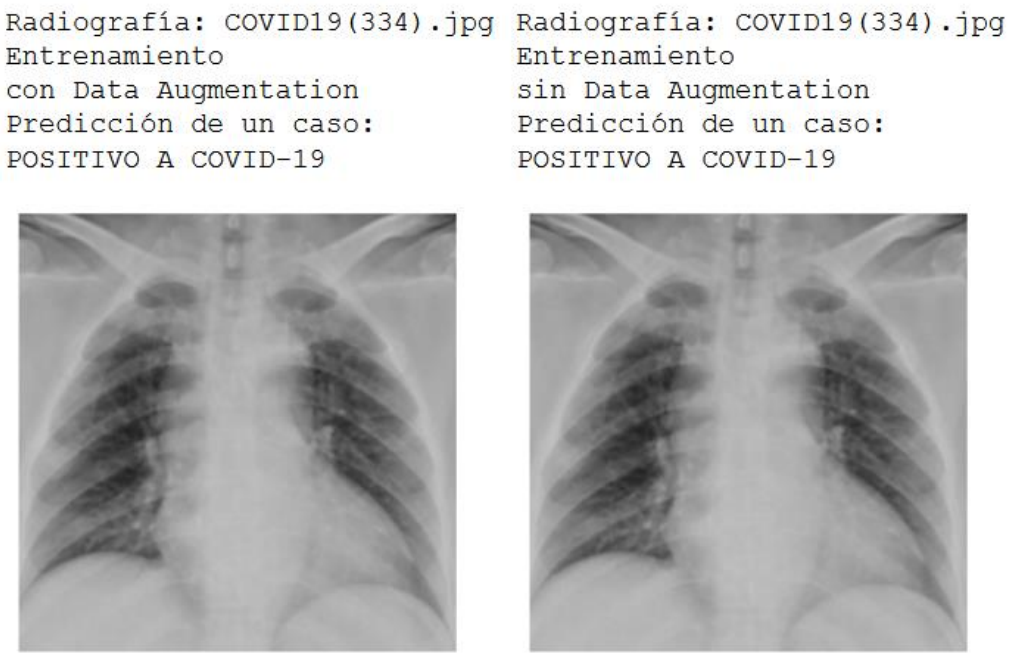


Figura N°37: Prueba de predicción de un caso positivo a COVID-19
Fuente: Elaboración Propia (2020)

En la figura N°38 se muestra la prueba de predicción realizada a una imagen de radiografía de tórax correspondiente a un caso normal, siendo llamada para las pruebas de predicción como caso “NO COVID-19”, además se presenta una comparativa entre las predicciones cuando se aplicó data augmentation y cuando no se aplicó data augmentation al proceso de entrenamiento.

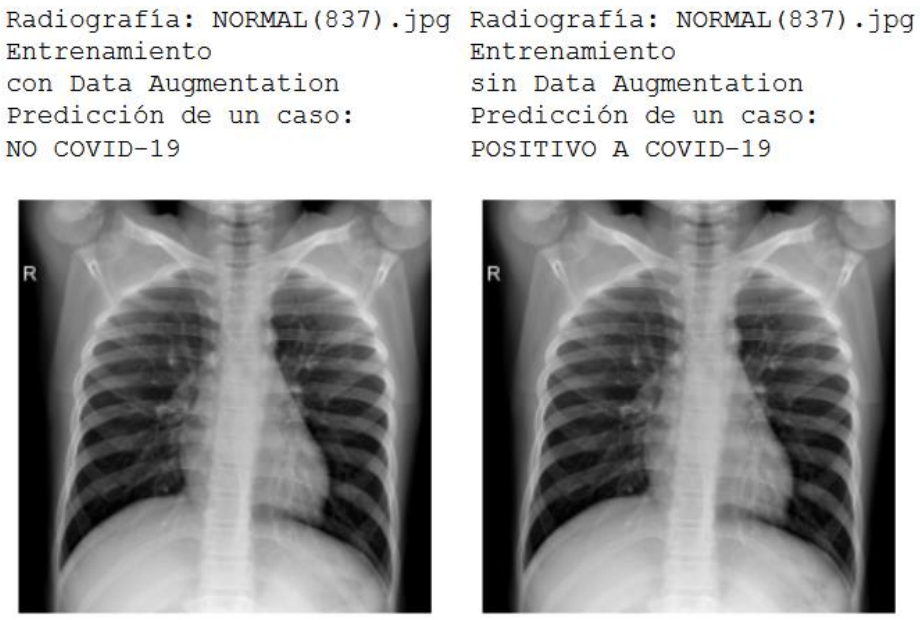


Figura N°38: Prueba de predicción de un caso NO COVID-19
Fuente: Elaboración propia (2020)

4.1.2. Resultados

Los resultados que se muestran a continuación tienen relación con el proceso de entrenamiento y el desempeño del modelo Particular

4.1.2.1. Proceso del entrenamiento

El proceso de entrenamiento con data augmentation y sin data augmentation contó con un total de 40 “epocas”. Los parámetros de medición utilizados para medir el proceso de entrenamiento fueron la métrica “accuracy” y función “Loss”.

En las figuras N°39 y N°40 se presentan las gráficas basadas en la métrica de “accuracy”. Mientras que las figuras N°41 y N°42 se muestran las gráficas basadas en la función “Loss”. Se infiere de las gráficas que el entrenamiento con data augmentation sufrió overfitting entre las “epocas” 10 y 15.

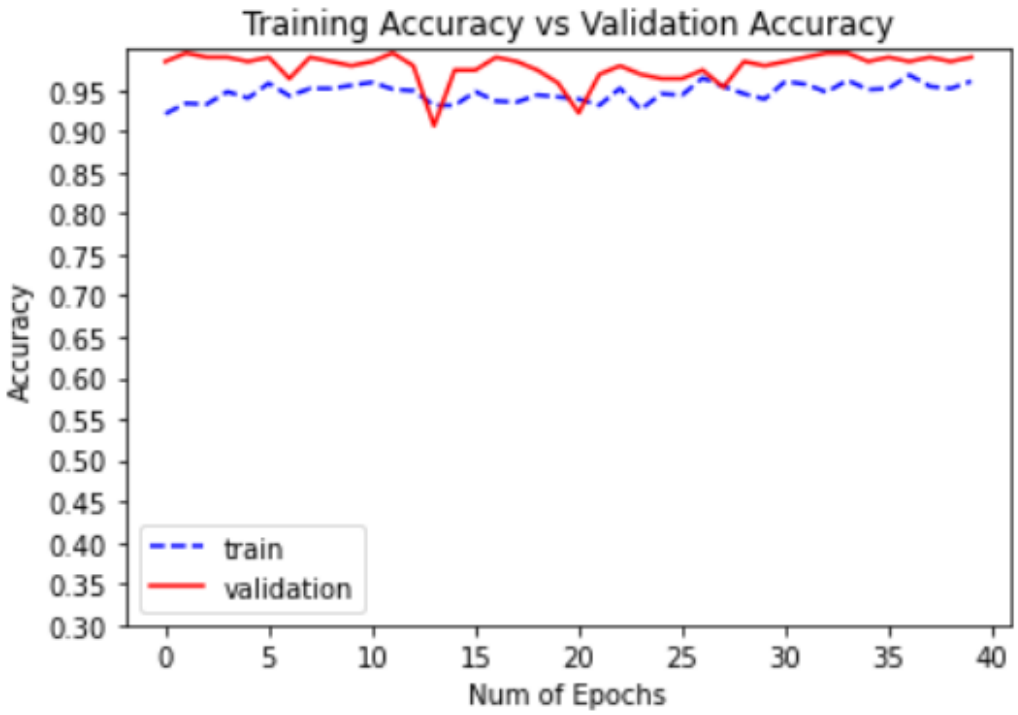


Figura N°39: Gráfica de Accuracy con data augmentation
Fuente: Elaboración propia (2020)

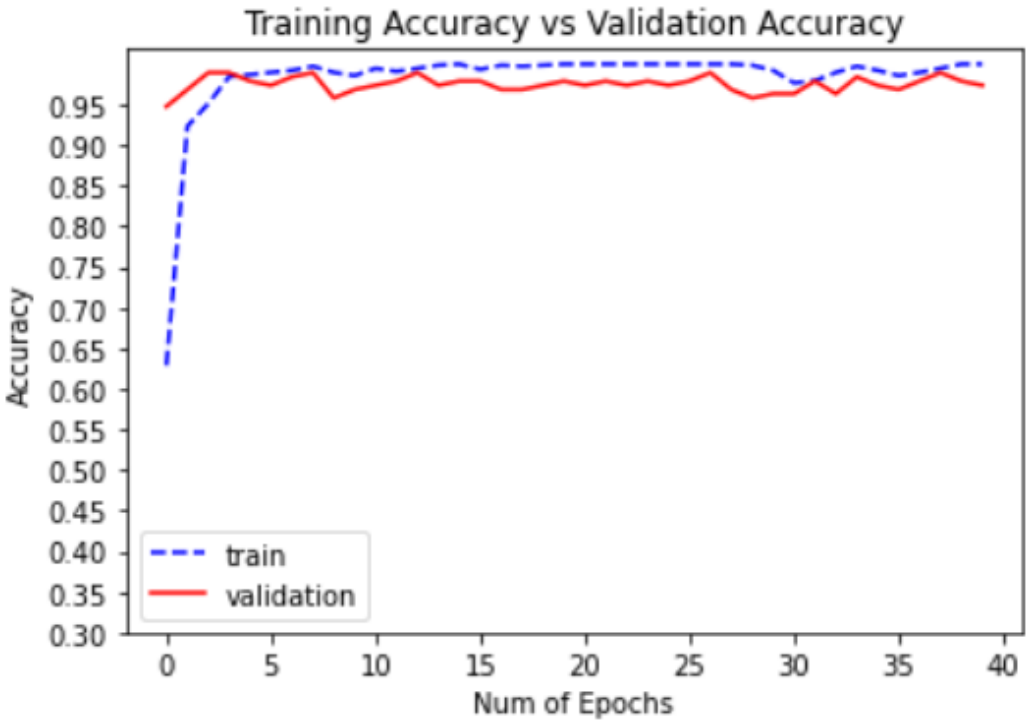


Figura N°40: Gráfica de Accuracy sin data augmentation
Fuente: Elaboración propia (2020)

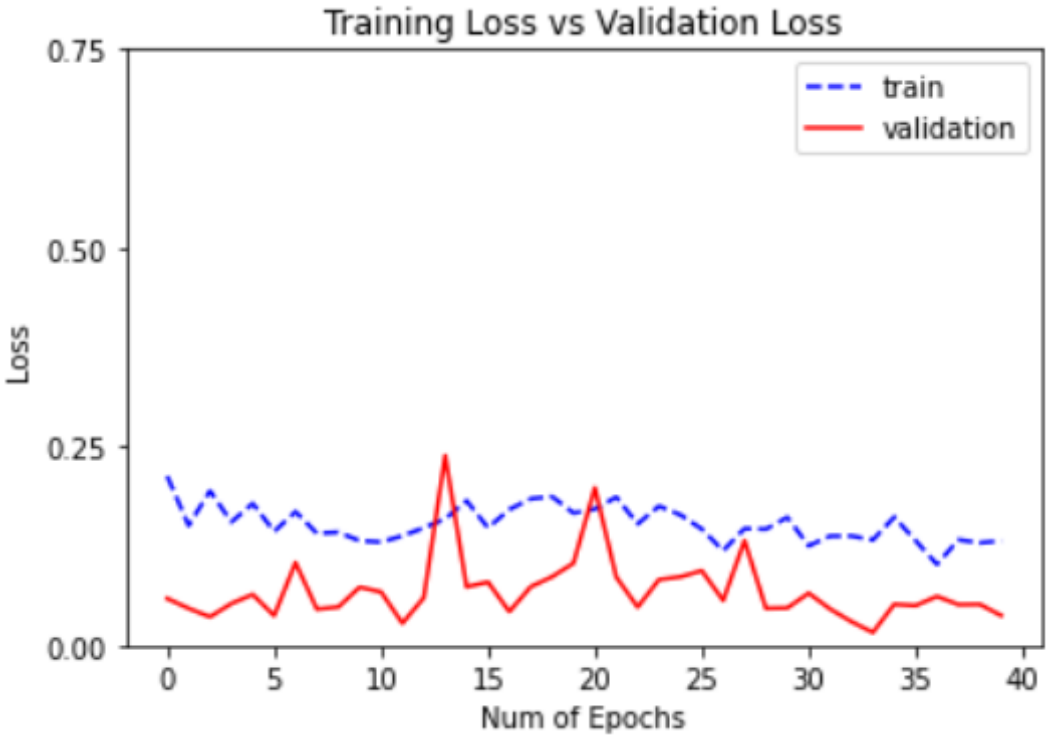


Figura N°41: Gráfica de la función Loss con data augmentation
Fuente: Elaboración propia (2020)

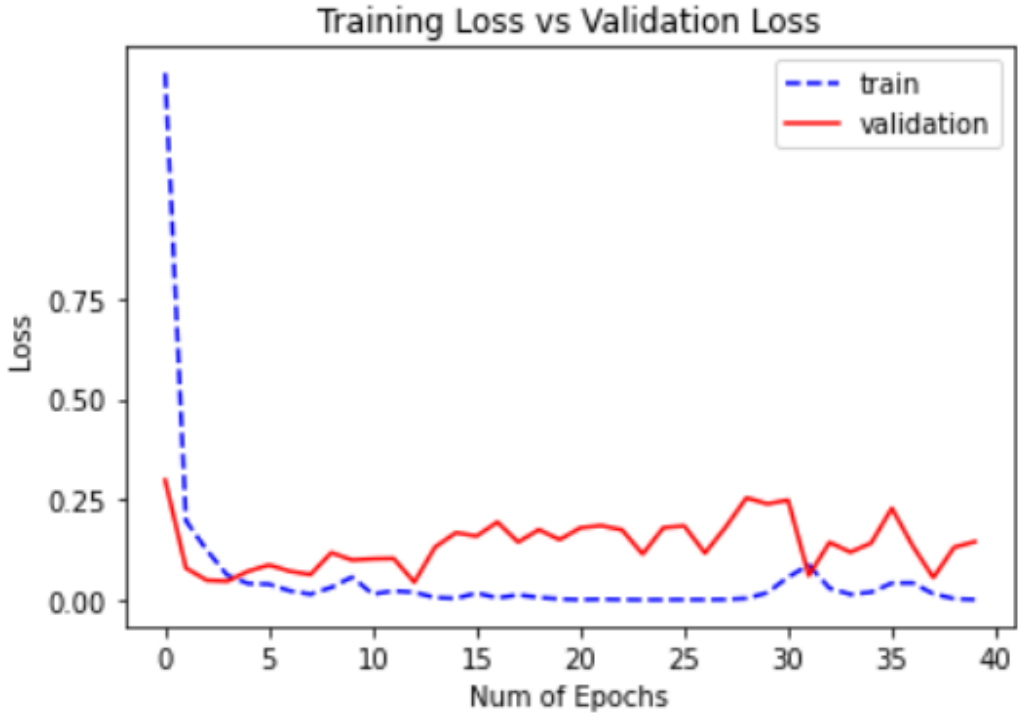


Figura N°42: Gráfica de la función Loss sin data augmentation
Fuente: Elaboración propia (2020)

4.1.2.2. Tabla de resultados:

Para conocer el desempeño del modelo Particular se utilizaron métricas de evaluación. Estas fueron accuracy, precisión, recall y F1-score.

En la Tabla N°4, se muestra una comparación entre los valores de las métricas de medición obtenidas, cuando el modelo fue entrenado con data augmentation y sin data augmentation.

Tabla N°4: Tabla de métricas de medición del modelo Particular

	Precision		**Recall**		**F1-score**		**Support**
	C/ D. A.	**S/ D. A.**	**C/ D. A.**	**S/ D. A.**	**C/ D. A.**	**S/ D. A.**	
COVID-19	0.9921	0.9424	0.9545	0.9924	0.9730	0.9668	132
NO COVID-19	0.9562	0.9920	0.9924	0.9394	0.9740	0.9650	132
Accuracy					0.9735	0.9659	264
macro avg	0.9742	0.9672	0.9735	0.9659	0.9735	0.9659	264
weighted avg	0.9742	0.9672	0.9735	0.9659	0.9735	0.9659	264

Fuente: Elaboración propia (2020)

De la tabla N°4, se puede extraer que el valor de la métrica “specificity” para el modelo cuando se aplicó data augmentation fue igual a 0.9924, mientras que cuando no se le aplicó data augmentation fue igual a 0.9394. Esto indica que el modelo entrenado con data augmentation es más preciso para proporcionar falsos positivos.

Además, referente a la métrica “accuracy”, la cual indica la exactitud que posee el modelo, se puede inferir que cuando el modelo fue entrenado con data augmentation se obtuvo un valor igual a 0.9735, mientras que cuando fue entrenado sin data augmentation fue igual a 0.9659.

4.1.2.3. Matriz de confusión:

El desempeño del modelo Particular se puede analizar por medio de una matriz de confusión, esta muestra las cantidades de casos referentes a verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.
En las figuras N°43 y N°44 se muestran las matrices de confusión para cuando se aplicó data augmentation y cuando no fue aplicado, respectivamente.

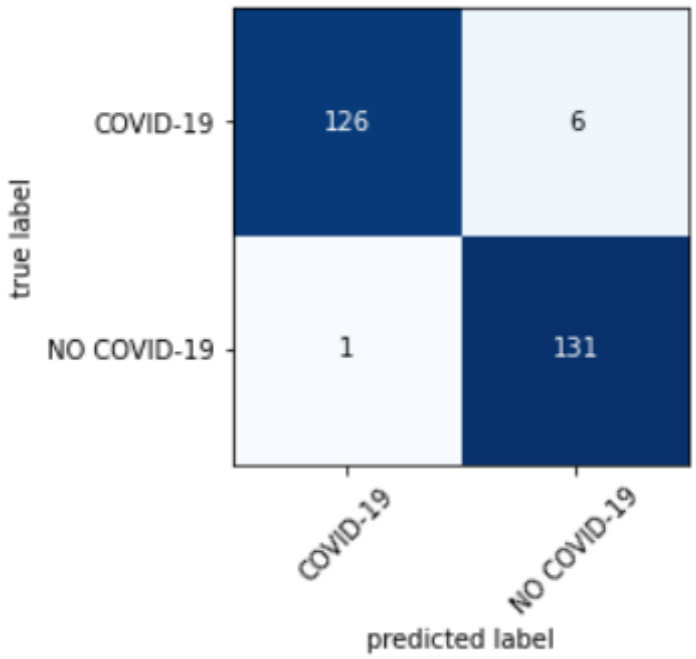


Figura N°43: Matriz de confusión del modelo Particular aplicando data augmentation
Fuente: Elaboración propia (2020)

De la matriz se puede extraer los siguientes datos:

Verdaderos Positivos = 126 Falsos Positivos = 1

Verdaderos Negativos = 131 Falsos Negativos = 6

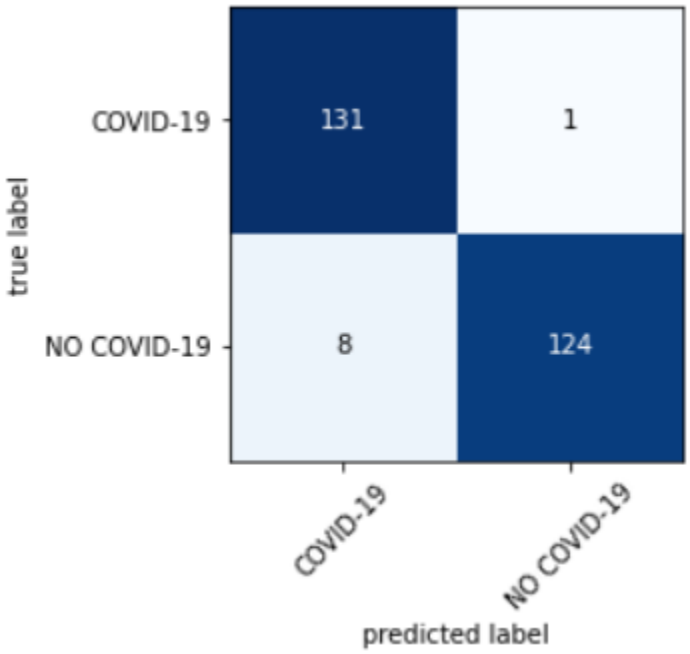


Figura N°44: Matriz de confusión del modelo Particular sin aplicar data augmentation
Fuente: Elaboración propia (2020)

De la matriz se puede extraer los siguientes datos:

Verdaderos Positivos = 131 Falsos Positivos = 8

Verdaderos Negativos = 124 Falsos Negativos = 1

4.2. Pruebas y resultados del modelo ResNet50

En la sección 4.2.1. se presentan las pruebas de predicción para cada caso del modelo ResNet50, mientras que en la sección 4.2.2. se presentan las gráficas del proceso de entrenamiento y los resultados obtenidos por medio de las métricas de evaluación y matrices de confusión.

4.2.1. Pruebas de predicción

Se realizaron las pruebas de predicción para verificar el adecuado funcionamiento del modelo ResNet50, por medio de un algoritmo (Véase Anexo N°5). Este algoritmo utilizó las imágenes almacenadas en la carpeta “Test” para realizar dichas pruebas.

En la figura N°45 se muestra la prueba de predicción realizada a una imagen de radiografía de tórax correspondiente es un caso positivo a COVID-19, además se realizó una comparativa entre las predicciones obtenidas cuando se aplicó data augmentation y cuando no se aplicó data augmentation al proceso de entrenamiento.

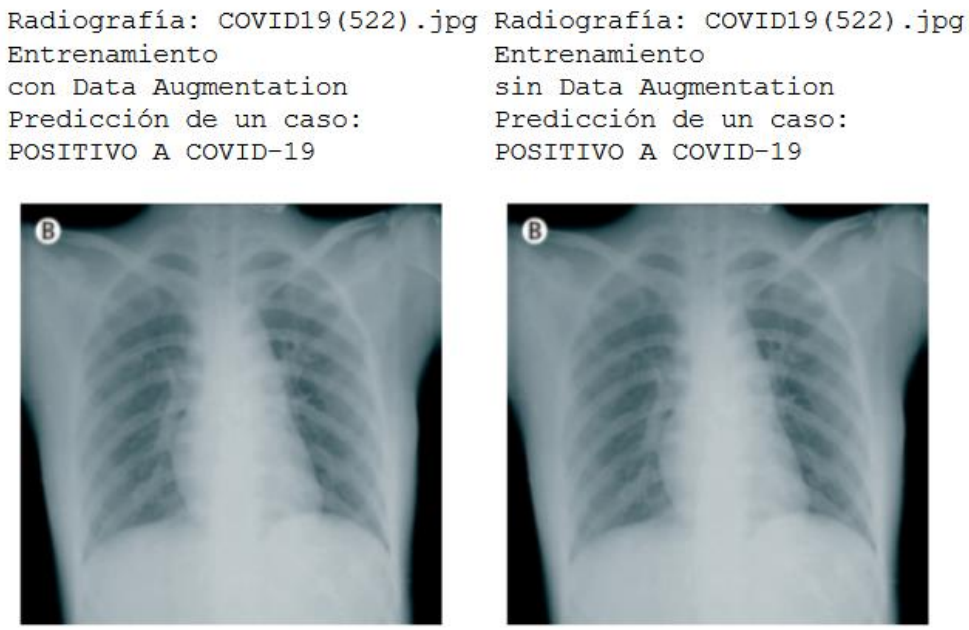


Figura N°45: Prueba de predicción de un caso positivo a COVID-19
Fuente: Elaboración Propia (2020)

En la figura N°46 se muestra la prueba de predicción realizada a una imagen de radiografía de tórax correspondiente a un caso normal, siendo llamada para las pruebas de predicción como caso “NO COVID-19”, además se presenta una comparativa entre las predicciones cuando se aplicó data augmentation y cuando no se aplicó data augmentation al proceso de entrenamiento.

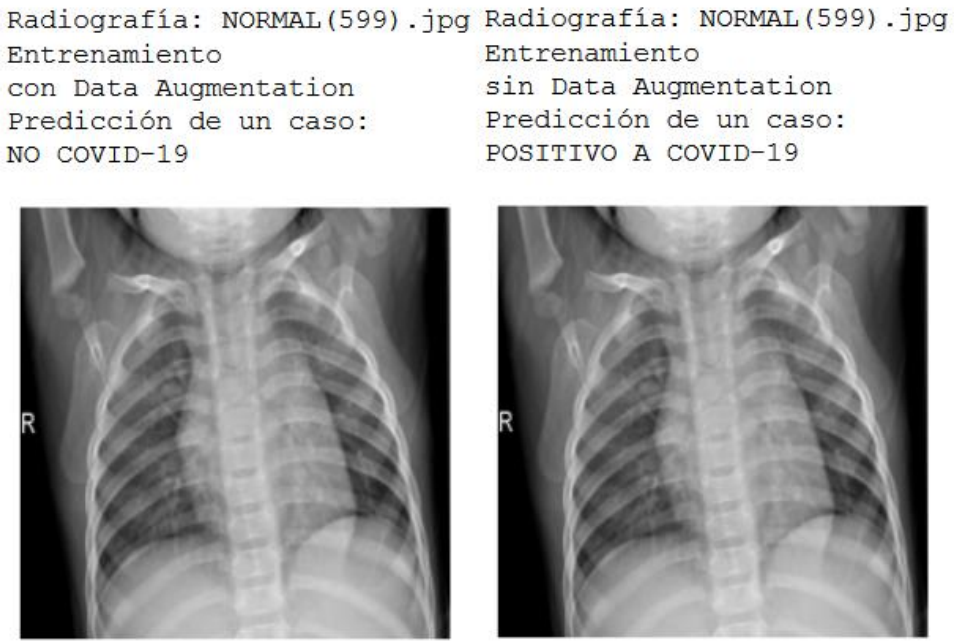


Figura N°46: Prueba de predicción de un caso NO COVID-19
Fuente: Elaboración propia (2020)

4.2.2. Resultados

Los resultados que se muestran a continuación tienen relación con el proceso de entrenamiento y el desempeño del modelo ResNet50.

4.2.2.1. Proceso del entrenamiento

El proceso de entrenamiento con data augmentation y sin data augmentation contó con un total de 40 “epocas”. Los parámetros de medición utilizados para medir el proceso de entrenamiento fueron la métrica “accuracy” y función “Loss”.

En las figuras N°47 y N°48 se presentan las gráficas basadas en la métrica de medición “accuracy”. Mientras que las figuras N°49 y N°50 se muestran las gráficas basadas en la función “Loss”. Se infiere de las gráficas que el entrenamiento tuvo un mejor progreso cuando fue entrenado con data augmentation, pero también sufrió de overfitting en algunos tramos del proceso.

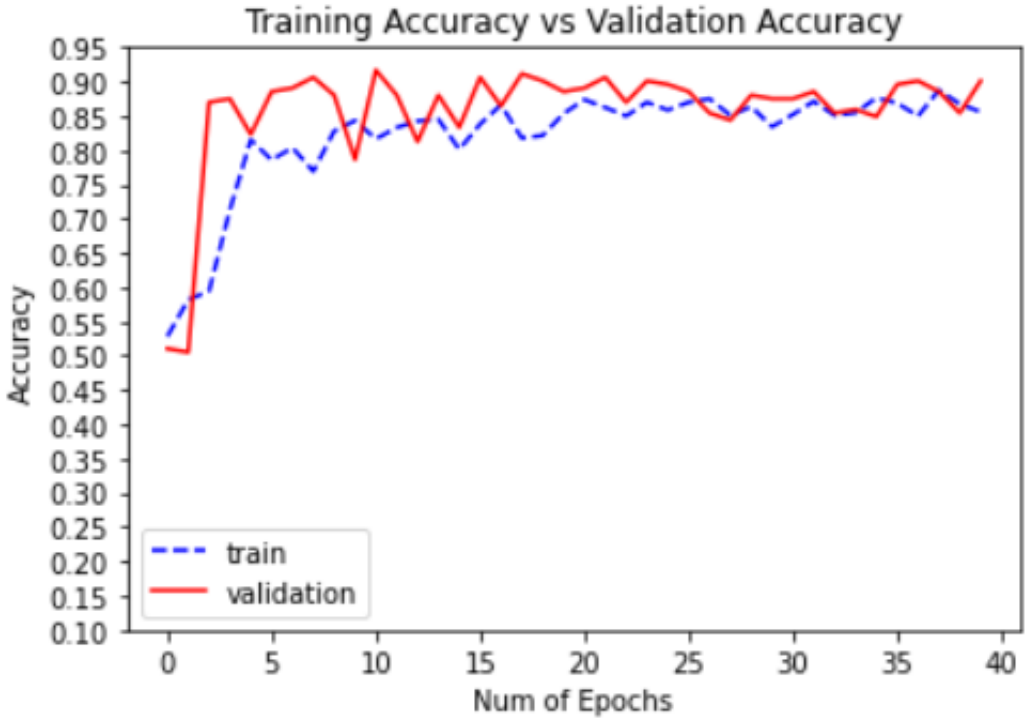


Figura N°47: Gráfica de Accuracy con data augmentation
Fuente: Elaboración propia (2020)

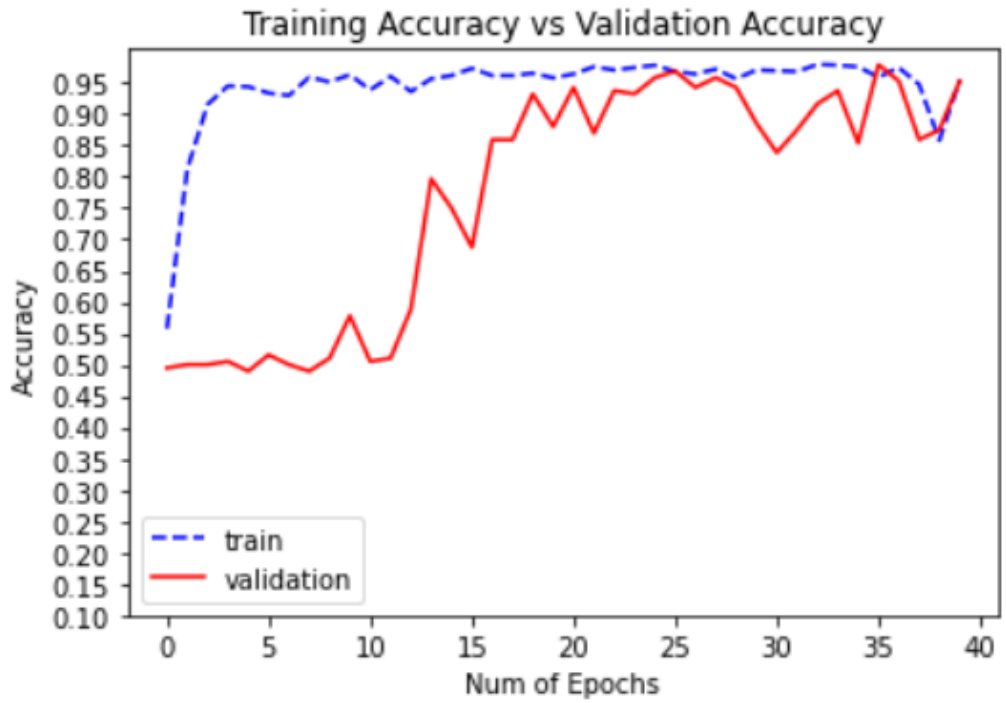


Figura N°48: Gráfica de Accuracy sin data augmentation
Fuente: Elaboración propia (2020)

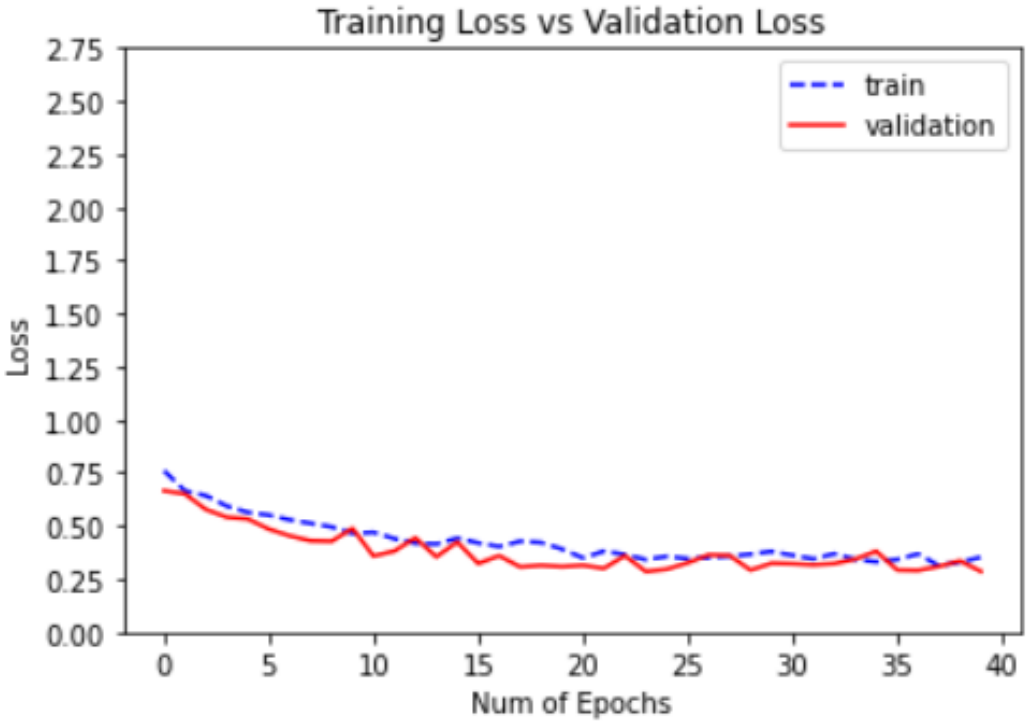


Figura N°49: Gráfica de la función Loss con data augmentation
Fuente: Elaboración propia (2020

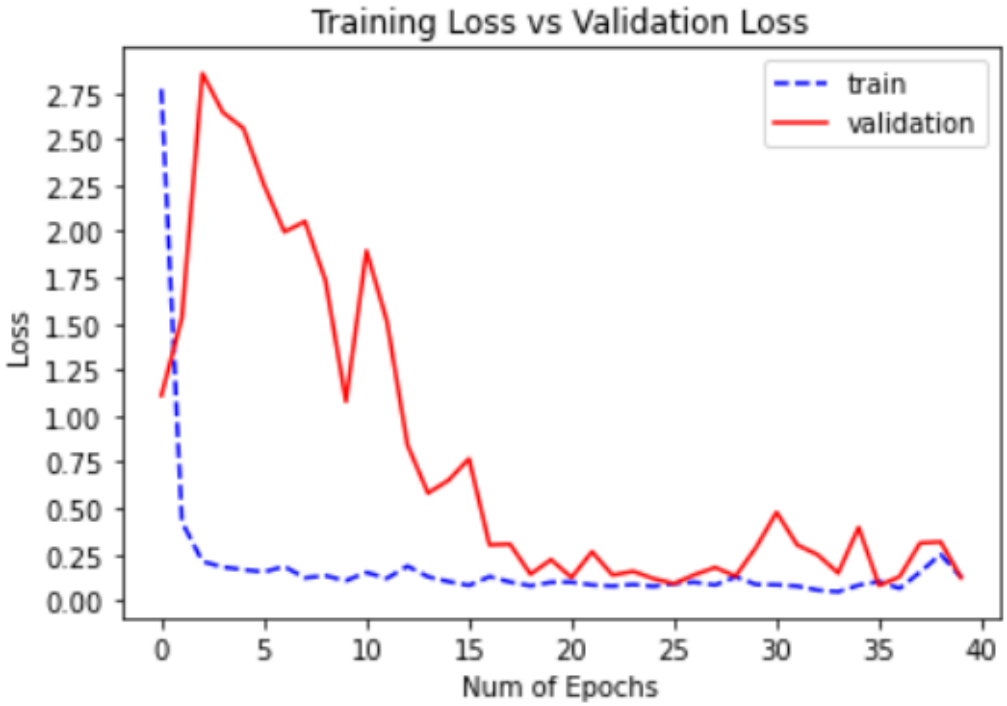


Figura N°50: Gráfica de la función Loss sin data augmentation
Fuente: Elaboración propia (2020)

4.2.2.2. Tabla de resultados:

Para conocer el desempeño del modelo ResNet50, se utilizaron métricas de evaluación. Estas fueron accuracy, precisión, recall y F1-score.

En la Tabla N°5, se muestra una comparación entre los valores de las métricas de medición obtenidas, cuando el modelo fue entrenado con data augmentation y sin data augmentation.

Tabla N°5: Tabla de métricas de medición del modelo ResNet50

	Precision		**Recall**		**F1-score**		**Support**
	C/ D. A.	**S/ D. A.**	**C/ D. A.**	**S/ D. A.**	**C/ D. A.**	**S/ D. A.**	
COVID-19	0.9762	0.9771	0.9318	0.9697	0.9535	0.9734	132
NO COVID-19	0.9348	0.9699	0.9773	0.9773	0.9556	0.9736	132
Accuracy					0.9545	0.9735	264
macro avg	0.9555	0.9735	0.9545	0.9735	0.9545	0.9735	264
weighted avg	0.9555	0.9735	0.9545	0.9735	0.9545	0.9735	264

Fuente: Elaboración propia (2020)

De la tabla N°5, se puede extraer que el valor de la métrica “specificity” para ambos casos fue la misma, siendo este igual a 0.9773. Además, basados en los valores respecto a la métrica “accuracy”, se puede inferir que cuando el modelo fue entrenado sin aplicar data augmentation se obtuvo un valor igual a 0.9735, mientras

que cuando se aplicó data augmentation se obtuvo un valor igual a 0.9545. Esto indica que el entrenamiento fue mejor cuando no se aplicó data augmentation. Sin embargo, en la etapa de predicción el entrenamiento sin data augmentation tuvo más errores que el entrenamiento con data augmentation. Esto puede deberse a que el entrenamiento sin data augmentation sufrió de overfitting, lo cual puede ser corroborado en las gráficas del proceso de entrenamiento.

4.2.2.3. Matriz de confusión:

El desempeño del modelo ResNet50 se puede analizar por medio de una matriz de confusión, esta muestra las cantidades de casos referentes a verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

En las figuras N°51 y N°52 se muestran las matrices de confusión para cuando se aplicó data augmentation y cuando no fue aplicado, respectivamente.

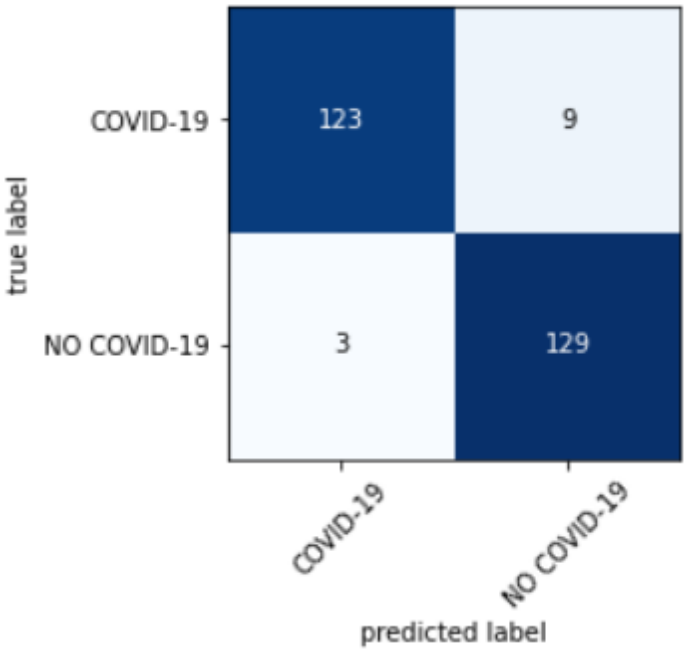


Figura N°51: Matriz de confusión del modelo ResNet50 aplicando data augmentation
Fuente: Elaboración propia (2020)

De la matriz se puede extraer los siguientes datos:

Verdaderos Positivos = 123 Falsos Positivos = 3

Verdaderos Negativos = 129 Falsos Negativos = 9

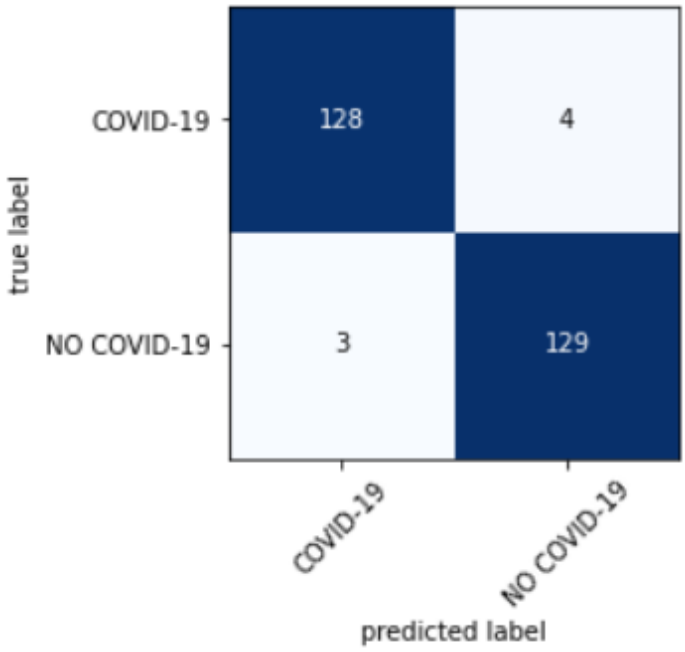


Figura N°52: Matriz de confusión del modelo ResNet50 sin aplicar data augmentation
Fuente: Elaboración propia (2020)

De la matriz se puede extraer los siguientes datos:

Verdaderos Positivos = 128 Falsos Positivos = 3

Verdaderos Negativos = 129 Falsos Negativos = 4

4.3. Pruebas y resultados del modelo InceptionV3

En la sección 4.3.1. se presentan las pruebas de predicción para cada caso del modelo InceptionV3, mientras que en la sección 4.3.2. se presentan las gráficas del proceso de entrenamiento y los resultados obtenidos por medio de las métricas de evaluación y matrices de confusión.

4.3.1. Pruebas de predicción

Se realizaron las pruebas de predicción para verificar el adecuado funcionamiento del modelo InceptionV3, por medio de un algoritmo (Véase Anexo N°5). Este algoritmo utilizó las imágenes almacenadas en la carpeta “Test” para realizar dichas pruebas.

En la figura N°53 se muestra la prueba de predicción realizada a una imagen de radiografía de tórax correspondiente es un caso positivo a COVID-19, además se realizó una comparativa entre las predicciones obtenidas cuando se aplicó data augmentation y cuando no se aplicó data augmentation al proceso de entrenamiento.

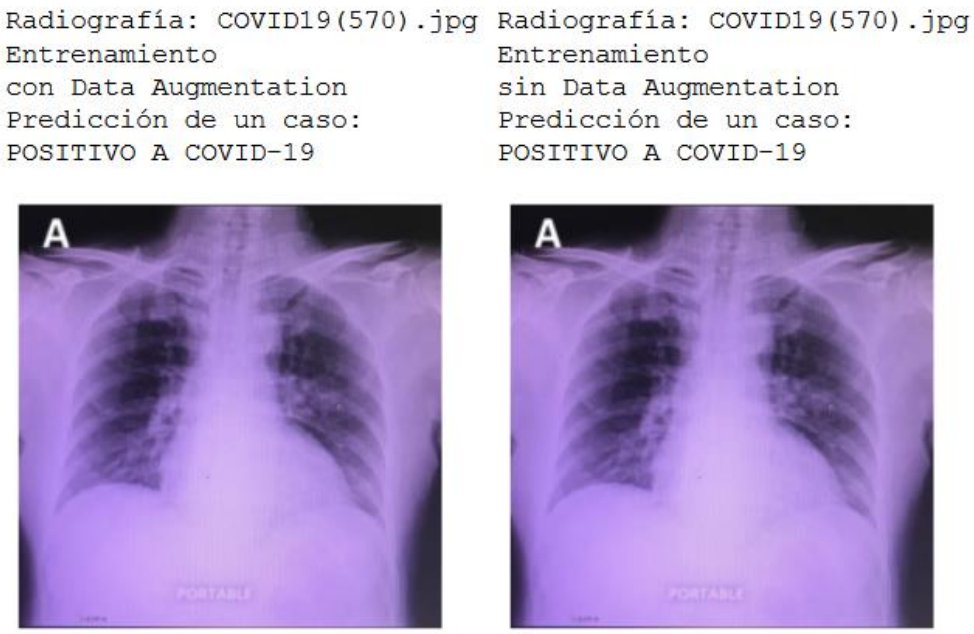
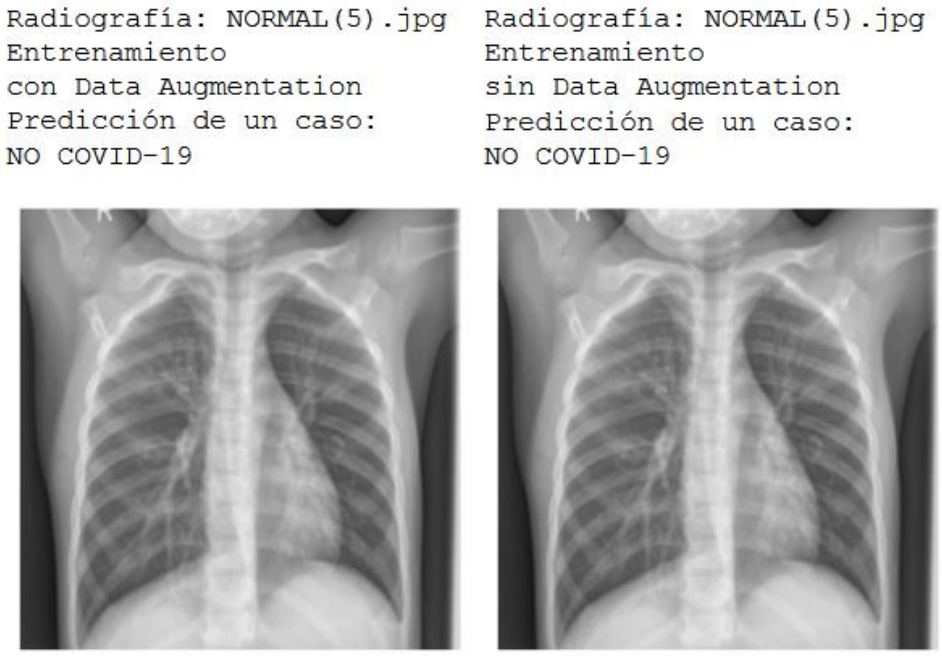


Figura N°53: Prueba de predicción de un caso positivo a COVID-19
Fuente: Elaboración Propia (2020)

En la figura N°54 se muestra la prueba de predicción realizada a una imagen de radiografía de tórax correspondiente a un caso normal, siendo llamada para las pruebas de predicción como caso “NO COVID-19”, además se presenta una comparativa entre las predicciones cuando se aplicó data augmentation y cuando no se aplicó data augmentation al proceso de entrenamiento.



.

Figura N°54: Prueba de predicción de un caso NO COVID-19
Fuente: Elaboración propia (2020)

4.3.2. Resultados

Los resultados que se muestran a continuación tienen relación con el proceso de entrenamiento y el desempeño del modelo InceptionV3.

4.3.2.1. Proceso del entrenamiento

El proceso de entrenamiento con data augmentation y sin data augmentation contó con un total de 40 “epocas”. Los parámetros de medición utilizados para medir el proceso de entrenamiento fueron la métrica “accuracy” y función “Loss”.

En las figuras N°55 y N°56 se presentan las gráficas basadas en la métrica de medición “accuracy”. Mientras que las figuras N°57 y N°58 se muestran las gráficas basadas en la función “Loss”. Se infiere que el entrenamiento tuvo un mejor progreso cuando fue entrenado con data augmentation, mientras que el entrenamiento sin data augmentation sufrió de overfitting.

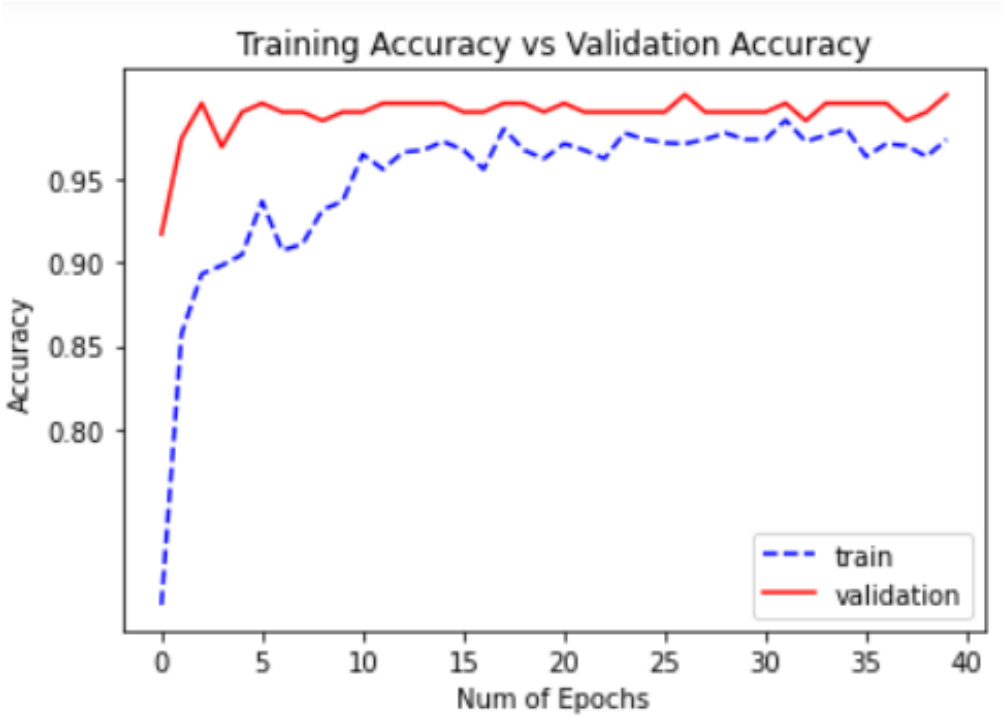


Figura N°55: Gráfica de Accuracy con data augmentation
Fuente: Elaboración propia (2020)

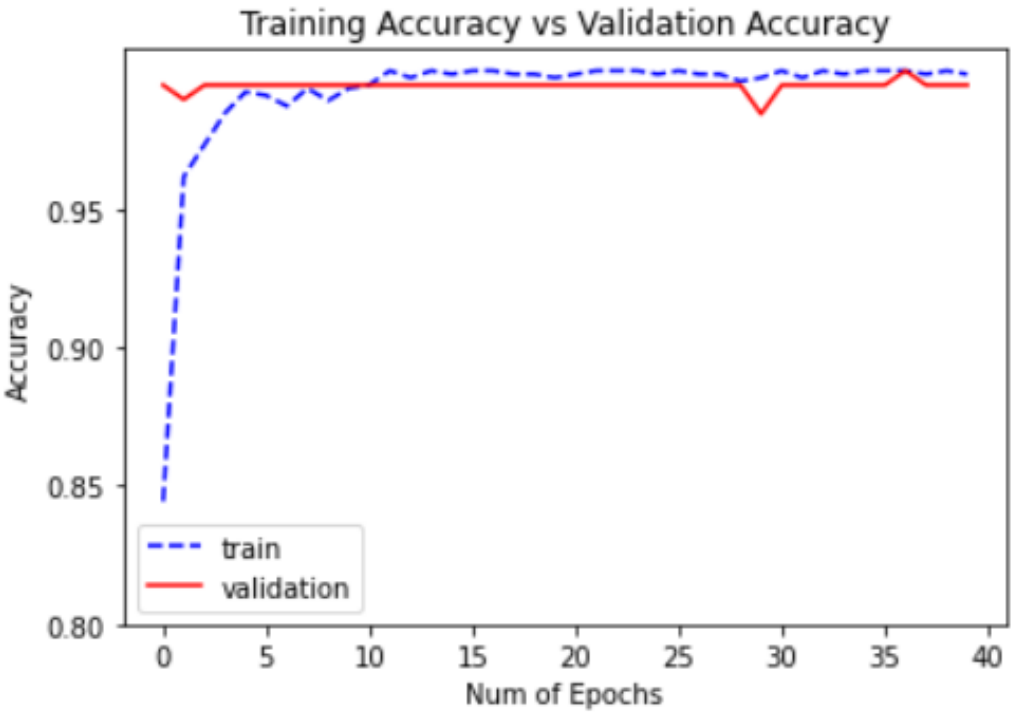


Figura N°56: Gráfica de Accuracy sin data augmentation
Fuente: Elaboración propia (2020)

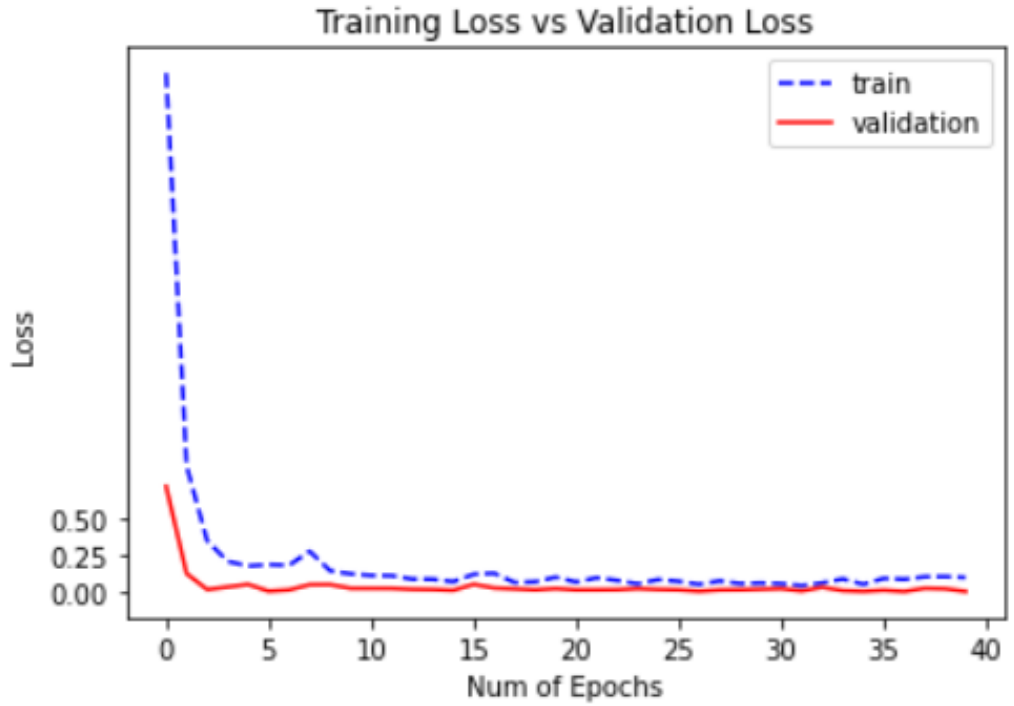


Figura N°57: Gráfica de la función Loss con data augmentation
Fuente: Elaboración propia (2020)

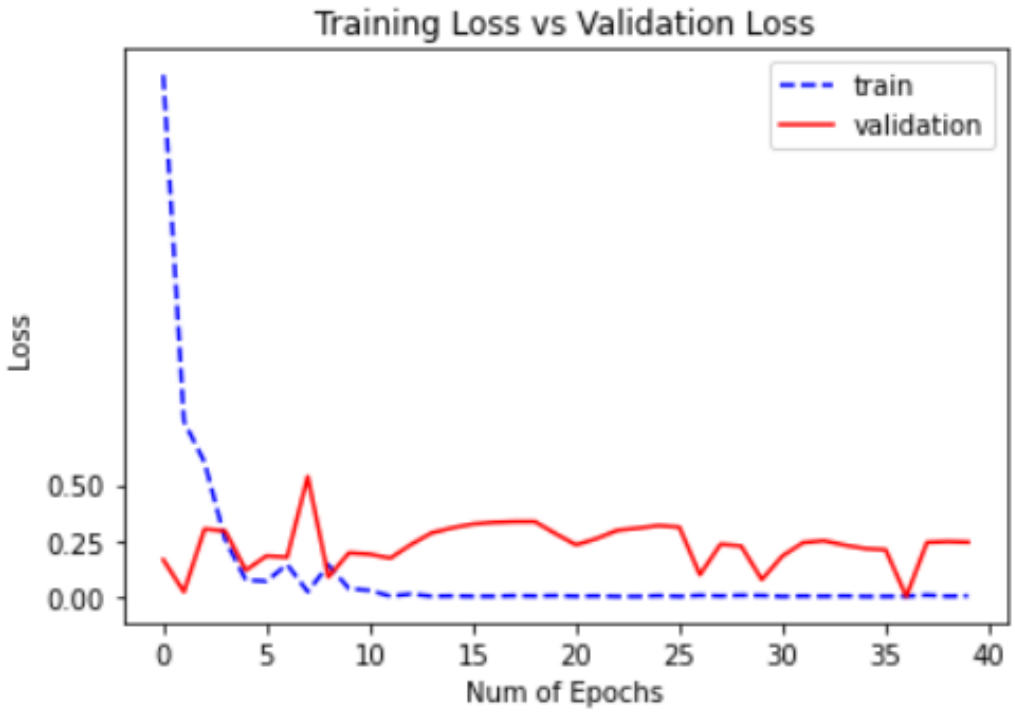


Figura N°58: Gráfica de la función Loss sin data augmentation
Fuente: Elaboración propia (2020)

4.3.2.2. Tabla de resultados:

Para conocer el desempeño del modelo InceptionV3 se utilizaron métricas de evaluación. Estas fueron accuracy, precisión, recall y F1-score.

En la Tabla N°6, se muestra una comparación entre los valores de las métricas de medición obtenidas, cuando el modelo fue entrenado con data augmentation y sin data augmentation.

Tabla N°6: Tabla de métricas de medición del modelo Inceptionv3

	Precision		**Recall**		**F1-score**		**Support**
	C/ D. A.	**S/ D. A.**	**C/ D. A.**	**S/ D. A.**	**C/ D. A.**	**S/ D. A.**	
COVID-19	0.9924	0.9776	0.9848	0.9924	0.9886	0.9850	132
NO COVID-19	0.9850	0.9923	0.9924	0.9773	0.9887	0.9847	132
Accuracy					0.9886	0.9848	264
macro avg	0.9887	0.9850	0.9886	0.9848	0.9886	0.9848	264
weighted avg	0.9887	0.9850	0.9886	0.9848	0.9886	0.9848	264

Fuente: Elaboración propia (2020)

De la tabla N°6, se puede extraer que la métrica “specificity” para cuando el modelo fue entrenado aplicando data augmentation se obtuvo un valor de 0.9924, mientras que cuando no se aplicó data augmentation se obtuvo un valor de 0.9773. Esto implica que el modelo entrenado con data augmentation posee una mayor precisión

para proporcionar falsos positivos. Además, basado en los valores obtenidos respecto a la métrica “accuracy” se puede extraer que el modelo entrenado con data augmentation es mucho más preciso que el entrenado sin data augmentation, siendo los valores de estos 0.9886 y 0.9848, respectivamente.

4.3.2.3. Matriz de confusión:

El desempeño del modelo InceptionV3 se puede analizar por medio de una matriz de confusión, esta muestra las cantidades de casos referentes a verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.
En las figuras N°59 y N°60 se muestran las matrices de confusión para cuando se aplicó data augmentation y cuando no fue aplicado, respectivamente.

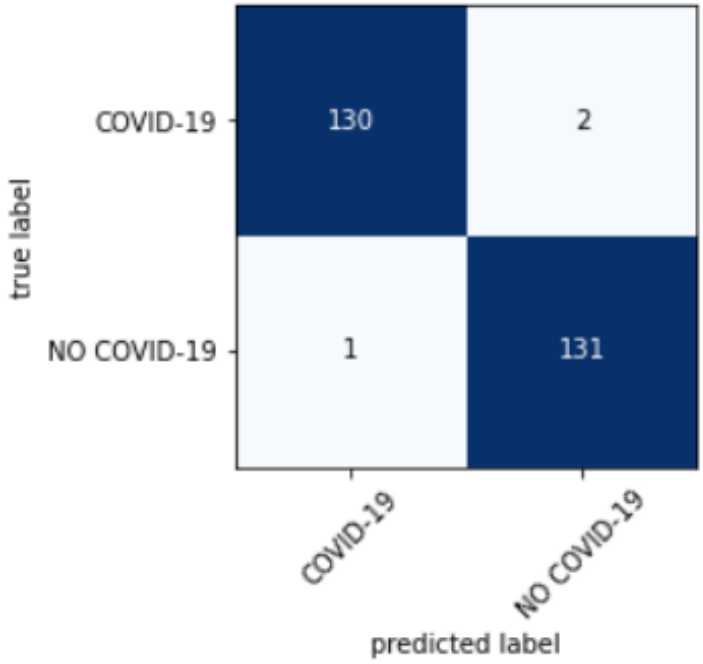


Figura N°59: Matriz de confusión del modelo InceptionV3 aplicando data augmentation
Fuente: Elaboración propia (2020)

De la matriz se puede extraer los siguientes datos:

Verdaderos Positivos = 130 Falsos Positivos = 1

Verdaderos Negativos = 131 Falsos Negativos = 2

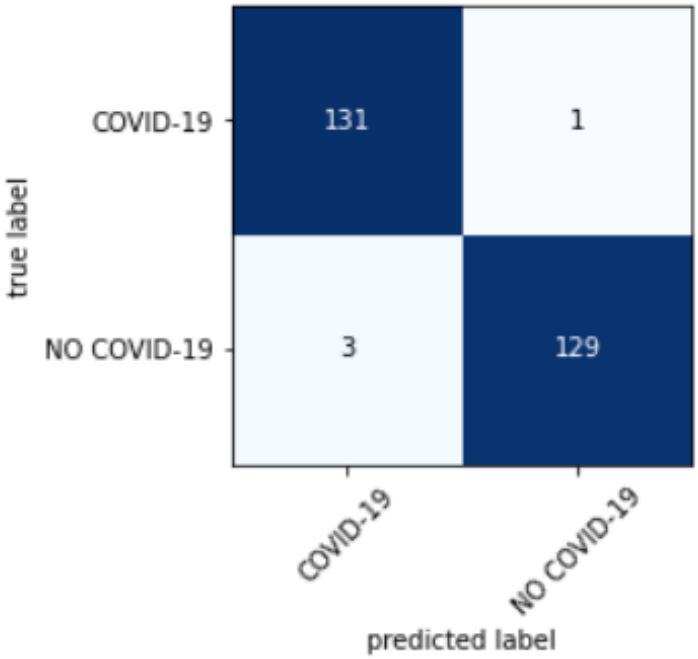


Figura N°60: Matriz de confusión del modelo InceptionV3 sin aplicar data augmentation
Fuente: Elaboración propia (2020)

De la matriz se puede extraer los siguientes datos:

Verdaderos Positivos = 131 Falsos Positivos = 3

Verdaderos Negativos = 129 Falsos Negativos = 1

4.4. Comparación de los modelos de Redes Neuronales Convolucionales

En las siguientes tablas se presentan la comparación entre los tres modelos de redes neuronales convolucionales basados en los resultados obtenidos por las métricas de medición y las matrices de confusión. Estas tablas permiten evaluar el desempeño de los modelos para conocer cuál de los tres es el más apto.

En las tablas N°7 y N°8, se representan los resultados obtenidos cuando se aplicó data augmentation y cuando no se aplicó, respectivamente.

Tabla N°7: Comparación de los tres modelos con data augmentation

Modelos	**Métricas de evaluación**								
	TP	**TN**	**FP**	**FN**	**Accuracy**	**Specificity**	**Precision**	**Recall**	**F1-score**
Particular	126	131	1	6	0.9735	0.9924	0.9742	0.9735	0.9735
ResNet50	123	129	3	9	0.9545	0.9773	0.9555	0.9545	0.9545
InceptionV3	130	131	1	2	0.9886	0.9924	0.9887	0.9886	0.9886

Fuente: Elaboración propia (2020)

Tabla N°8: Comparación de los tres modelos sin data augmentation

Modelos	Métricas de evaluación								
	TP	**TN**	**FP**	**FN**	**Accuracy**	**Specificity**	**Precision**	**Recall**	**F1-score**
Particular	131	124	8	1	0.9659	0.9394	0.9672	0.9659	0.9659
ResNet50	128	129	3	4	0.9735	0.9773	0.9735	0.9735	0.9735
InceptionV3	131	129	3	1	0.9848	0.9773	0.9850	0.9848	0.9848

Fuente: Elaboración propia (2020)

4.5. Presupuesto

Se presenta en la tabla N°9 el presupuesto utilizado para el desarrollo del proyecto de tesis, detallando los gastos realizados .

Tabla N°9: Presupuesto del proyecto de tesis

Dispositivos	**Precio (S/.)**
Unidad de estado sólido KINGSTON de 240GB	350
Útiles de escritorio	10
TOTAL	**360**

Fuente: Elaboración propia (2020)

CONCLUSIONES

1. Según los resultados que se muestran en la tabla N°4 correspondientes al modelo Particular, se concluye que su mejor desempeño fue cuando se aplicó data augmentation, con exactitud igual a 0.9735 y mientras que sin la aplicación de data augmentation fue igual a 0.9659. Además, según las figuras N°39 y N°41, correspondientes al proceso de entrenamiento, se puede indicar que este modelo sufrió de overfitting cuando se aplicó data augmentation.

2. De acuerdo a la tabla N°5, el modelo ResNet50 obtuvo una exactitud de 0.9545 cuando se aplicó data augmentation y de 0.9735 sin data augmentation. Se infiere de esto, que ResNet50 no requiere de la aplicación de data augmentation para mejorar su desempeño. Sin embargo, según las figuras N°48 y N°50 el modelo sufrió de overfitting cuando no se aplicó de data augmentation, además, en las pruebas de predicción realizadas, el modelo entrenado sin data augmentation tuvo un menor acierto que el entrenado con data augmentation.

3. El modelo InceptionV3 según indica la tabla N°6 obtuvo una exactitud de 0.9886, luego de aplicar data augmentation y 0.9848 sin aplicar data augmentation. Por consiguiente, el modelo InceptionV3 tuvo un buen desempeño para ambos casos, siendo el mejor cuando se aplicó data augmentation. Además, en la figura N°58 se observó que el entrenamiento sin data augmentation sufrió de overfitting.

4. Teniendo en cuenta la información proporcionada por las tablas N°7 y N°8, se puede concluir que el modelo más apto de los tres evaluados, para realizar la clasificación de imágenes de radiografías de tórax, fue InceptionV3, tanto para cuando se aplicó data augmentation como para la inaplicación del mismo. Además, el modelo ResNet50 fue el que peor desempeño tuvo luego de aplicar data augmentation, mientras que sin aplicar data augmentation el de peor desempeño fue el modelo Particular.

RECOMENDACIONES

1. Se recomienda aumentar el tamaño del dataset, esto permitirá poder obtener mejores resultados respecto al entrenamiento de una red neuronal convolucional, dado que mientras mayor sea la cantidad de datos, mejor podrá la red discernir entre las clases definidas.

2. Se recomienda utilizar un GPU (Unidad de procesamiento gráfico) o un TPU (Unidad de procesamiento tensorial), con el fin de reducir el tiempo de entrenamiento de la red neuronal convolucional, así como también reducir el esfuerzo computacional de la PC o Laptop que se esté utilizando.

3. Se sugiere utilizar otros editores de desarrollo u otro software, tal es el caso de Google Colab, Matlab y PyCharm, los cuales permiten realizar la implementación y entrenamiento de forma más visual y cómodo para el usuario debido a la interfaz que manejan haciéndolos más adecuados para el empleo de las redes neuronales artificiales.

REFERENCIAS BIBLIOGRÁFICAS

Alvarez, M. A. (19 de noviembre de 2003). *Qué es Python*. Recuperado el 15 de septiembre de 2020, de Desarrollo Web: https://desarrolloweb.com/articulos/1325.php

Artola Moreno, Á. (2019). Clasificación de imágenes usando redes neuronales convolucionales en Python. *Trabajo Fin de Grado.* Universidad de Sevilla, Sevilla.

Cohen, J. P. (2020). *COVID-19 Image Data Collection: Prospective Predictions Are the Future*. Recuperado el 2020 de Setiembre de 21, de GitHub: https://github.com/ieee8023/covid-chestxray-dataset

Colula, S., & Ángel, L. (2019). Reconocimiento de patrones en imágenes médicas por medio de redes neuronales convolucionales. *(Master's thesis).* Benemérita Universidad Autónoma de Puebla, Puebla.

Erroz Arroyo, D. (2019). Visualizando neuronas en redes neuronales convolucionales. *(Trabajo de fin de grado).* Universidad Pública de Navarra, Pamplona.

Ertam, F., & Aydın, G. (2017). Data classification with deep learning using Tensorflow. *International Conference on Computer Science and Engineering (UBMK)*, 755-758.

Gómez-Ríos, A., Tabik, S., Luengo, J., & Herrera, F. (2019). Redes Neuronales Convolucionales para Una Clasificacion Precisa de Imágenes de Corales. *In XVIII Conferencia de la Asociación Española para la Inteligencia Artificial*, 1171-1176.

Hernández Sampieri, R., Fernández Collado, C., & Pilar Baptista Lucio, M. (2014). *Metodología de la investigación.* México: McGraw-Hill.

Jauregui, A. F. (29 de Marzo de 2020). *Qué son y cómo crear una red neuronal convolucional con Keras*. Recuperado el 18 de Octubre de 2020, de Ander Fernández Jauregui: https://anderfernandez.com/blog/que-es-una-red-neuronal-convolucional-y-como-crearlaen-keras/

Keras. (2020). *Keras documentation: Keras Applications*. Recuperado el 26 de Octubre de 2020, de Keras.io: https://keras.io/api/applications/

Lao, Y. O., Rivas-Méndez, A., Pérez-Pravia, M. C., & Marrero-Delgado, F. (2017). Procedimiento para el pronóstico de la demanda mediante redesneuronales artificiales. *Ciencias Holguín, 23*(1), 1-18.

Mathworks. (2020). *Mathworks*. Recuperado el 12 de octubre de 2020, de Redes Neuronales Convolucionales: https://la.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Matich, D. J. (2001). Redes Neuronales: Conceptos básicos y aplicaciones. *Catedra*. Universidad Tecnológica Nacional, Mexico.

Mayo Clinic. (13 de Marzo de 2018). *Neumonía*. Recuperado el 15 de octubre de 2020, de Mayo Clinic: https://www.mayoclinic.org/es-es/diseases-conditions/pneumonia/symptoms-causes/syc-20354204

Mayo Clinic. (5 de Mayo de 2020). *Radiografía*. Recuperado el 15 de octubre de 2020, de Mayo Clinic: https://www.mayoclinic.org/es-es/tests-procedures/x-ray/about/pac-20395303

Medrano Roldán, A. K. (Junio de 2019). Red neuronal convolucional en un ambiente pseudo-distribuido para la clasificación de radiografías de pacientes con neumonía. Licenciatura en Ingeniería en Sistemas Computacionales. *Licenciatura en Ingeniería en Sistemas Computacionales.* Universidad Autónoma de Ciudad Juarez, Juarez.

Mohajon, J. (28 de Mayo de 2020). *Confusion Matrix for Your Multi-Class Machine Learning Model*. Recuperado el 10 de octubre de 2020, de towards data science: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

Mooney, P. (2018). *Chest X-Ray Images (Pneumonia)*. Recuperado el 15 de Setiembre de 2020, de Kaggle: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

Narin, A., Kaya, C., & Pamuk, Z. (2020). Automatic detection of coronavirus disease (covid-19) using x-ray images and deep convolutional neural networks. *arXiv preprint arXiv:2003.10849*, 1-31.

Organización Mundial de la Salud. (2019). *¿Qué es la COVID-19?* Recuperado el 12 de septiembre de 2020, de Organización Mundial de la Salud: https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019/advice-for-public/q-a-coronaviruses

Ozturk, T., Talo, M., Yildirim, E. A., Baloglu, U. B., Yildirim, O., & Acharya, U. R. (2020). Automated detection of COVID-19 cases using deep neural networks with X-ray images. *Computers in Biology and Medicine, 121*(103792), 1-11.

Patel, P. (2020). *Chest X-ray (Covid-19 & Pneumonia)*. Recuperado el 27 de octubre de 2020, de kaggle.com: https://www.kaggle.com/prashant268/chest-xray-covid19-pneumonia

POCHO COSTA. (2019). *Transfer Learning ¿qué es y para que sirve?* Recuperado el 23 de septiembre de 2020, de www.pochocosta.com: https://pochocosta.com/podcast/transfer-learning-que-es-y-para-que-sirve/

Rodríguez, A. K. (6 de Marzo de 2017). *Sistemas de diagnóstico asistido por ordenadores*. Recuperado el 14 de septiembre de 2020, de quiurevista: https://quiurevista.com/sistemas-diagnostico-asistido-ordenadores/

Salas, R. (2004). Redes neuronales artificiales. *Universidad de Valparaıso. Departamento de Computación, 1*, 1-7.

Sethi, R., Mehrotra, M., & Sethi, D. (July de 2020). Deep Learning based Diagnosis Recommendation for COVID-19 using Chest X-Rays Images. (IEEE, Ed.) *Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, 1-4.

Sharma, H., Jain, J. S., Bansal, P., & Gupta, S. (January de 2020). Feature Extraction and Classification of Chest X-Ray Images Using CNN to Detect Pneumonia. (IEEE, Ed.) *10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 227-231.

SIRM. (2020). *COVID-19 DATABASE*. Recuperado el 2020 de Octubre de 27, de Sirm.org: https://www.sirm.org/category/senza-categoria/covid-19/

Sultana, N. N., Mandal, B., & Puhan, N. B. (2018). Deep residual network with regularised fisher framework for detection of melanoma. *IET Computer Vision, 12*(8), 1096-1104.

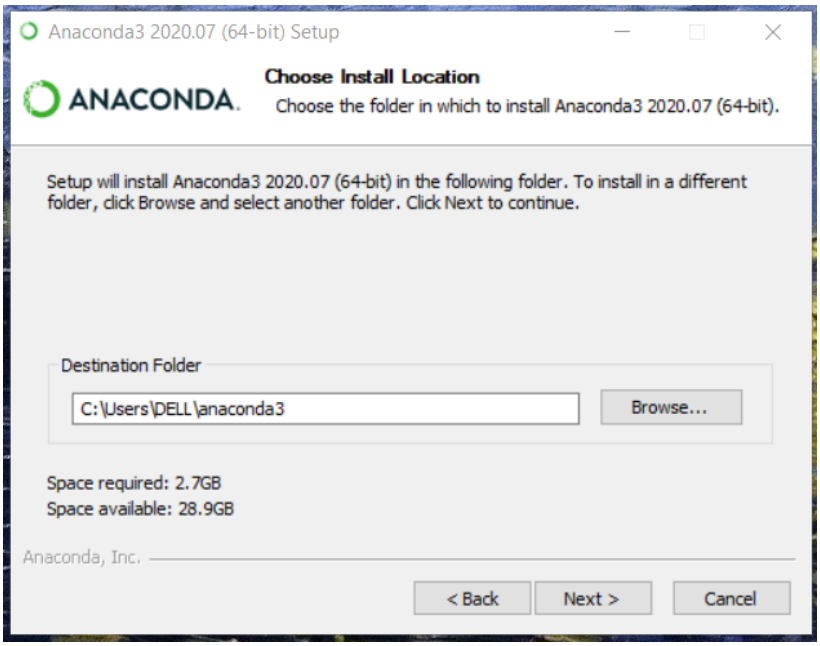
Szegedy, C., Vanhoucke, V., Ioffe, S. S., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818-2826.

TensorFlow. (26 de Mayo de 2020). *Compatibilidad con GPU*. Recuperado el 2 de octubre de 2020, de TensorFlow: https://www.tensorflow.org/install/gpu?hl=es-419

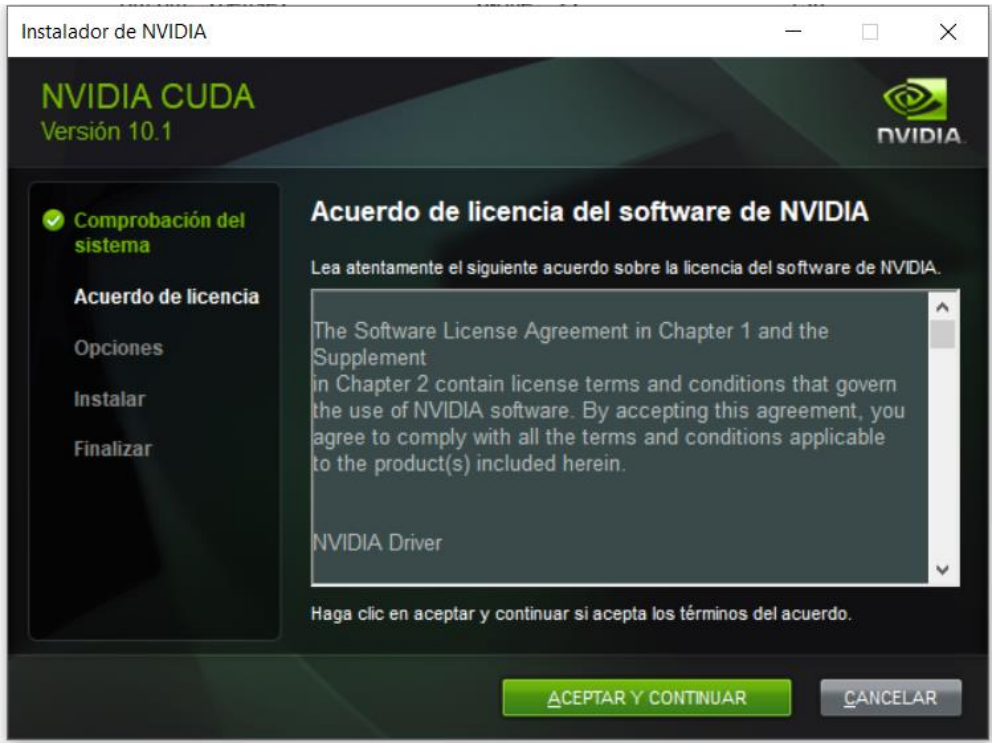
TensorFlow. (18 de Agosto de 2020). *Keras*. Recuperado el 14 de octubre de 2020, de TensorFlow: https://www.tensorflow.org/guide/keras?hl=es

ANEXOS

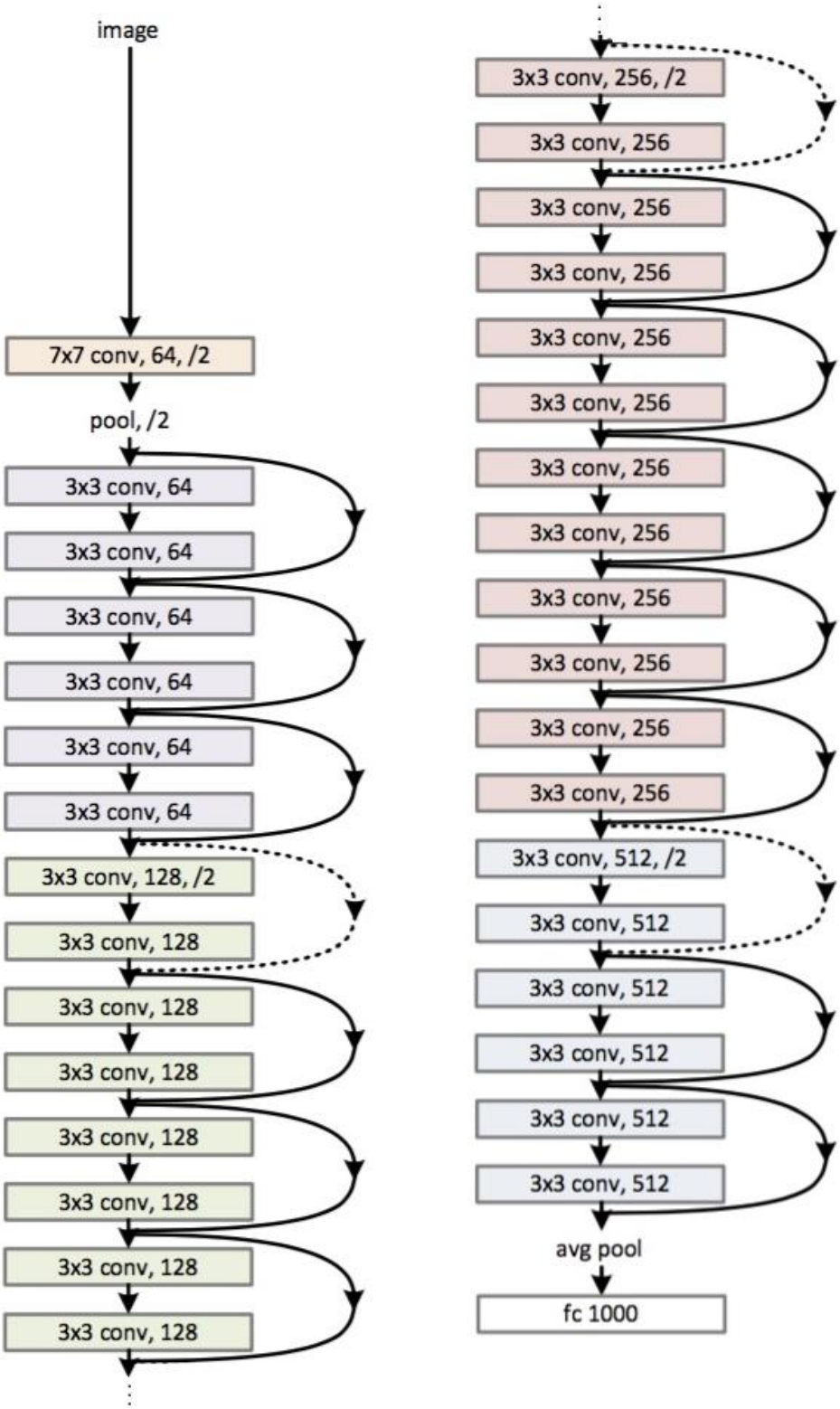
ANEXO N°1: INSTALACIÓN DEL INTERFAZ GRÁFICA DE USUARIO ANACONDA NAVIGATOR



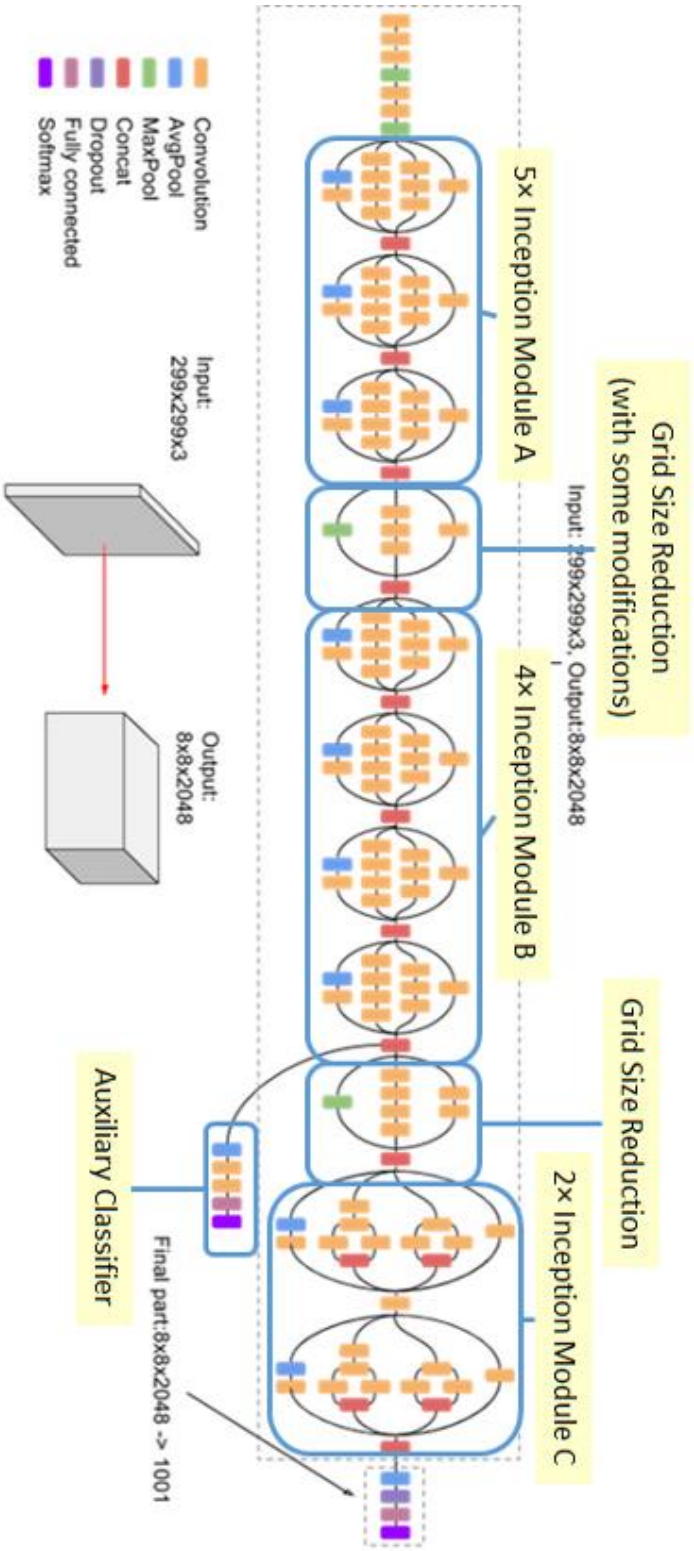
ANEXO N°2: INSTALACIÓN DE NVIDIA CUDA VERSION 10.1



ANEXO N°3: DIAGRAMA DE LA ARQUITECTURA DEL MODELO RESNET50



ANEXO N°4: DIAGRAMA DE LA ARQUITECTURA DEL MODELO INCEPTION-V3



ANEXO N°5: ALGORITMO DE PREDICCIÓN PARA UNA SOLA IMAGEN

```
# Prediction
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
from keras.applications.imagenet_utils import preprocess_input,
from keras.applications.imagenet_utils import decode_predictions
from keras.models import load_model

names = ['POSITIVO A COVID-19','NO COVID-19']

modelo= 'D:/TESIS/modelos/cnnpda.h5'
cnn = load_model(modelo)

imaget_path = "D:/TESIS/Dataset/Test/Normal/NORMAL(32).jpg"

imaget=cv2.resize(cv2.imread(imaget_path),
                  (224, 224),#(299, 299) para InceptionV3
                  interpolation = cv2.INTER_AREA)
xt = np.asarray(imaget)
xt=preprocess_input(xt)
xt = np.expand_dims(xt,axis=0)
preds = cnn.predict(xt)

img = os.path.basename(imaget_path)
print("Radiografía:",img)
print("Entrenamiento")
print("sin Data Augmentation")
print("Predicción de un caso:")
print(names[np.argmax(preds)])
plt.imshow(cv2.cvtColor(np.asarray(imaget),cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

ANEXO N°6: ALGORITMO PARA GENERAR LA MATRIZ DE CONFUSIÓN Y LAS MÉTRICAS DE CLASIFICACIÓN.

```
from sklearn.metrics import confusion_matrix, f1_score, roc_curve
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn import metrics
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from mlxtend.plotting import plot_confusion_matrix
from keras.models import load_model
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
names = ['COVID-19','NO COVID-19']

test_data_dir = 'D:/TESIS/Dataset/Test/'

test_datagen = ImageDataGenerator(rescale=1. / 255)

test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(224, 224),#(299, 299) para InceptionV3
    batch_size=1,
    class_mode=None,
    shuffle=False)
savemodel='D:/TESIS/modelos/resnet50.h5'

Model= load_model(savemodel)

pred=Model.predict(test_generator)

y_pred = np.argmax(pred, axis=1)
y_real = test_generator.classes


matc=confusion_matrix(y_real, y_pred)


plot_confusion_matrix(conf_mat=matc, figsize=(4,4),
                      class_names = names, show_normed=False)
plt.tight_layout()

print(metrics.classification_report(y_real,y_pred, digits = 4))
```