UNIVERSIDAD RICARDO PALMA

FACULTAD DE INGENIERÍA

PROGRAMA DE TITULACIÓN POR TESIS

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

DISEÑO Y APLICACIÓN DE REDES NEURONALES CONVOLUCIONALES PARA EL DIAGNÓSTICO DE ENFERMEDADES PULMONARES, A PARTIR DE RADIOGRAFÍAS DE TÓRAX FRONTAL

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO ELECTRÓNICO

PRESENTADA POR

Bach. MALDONADO LEZAMA, LISSET FERNANDA

Bach. MOREANO ROJAS, EDER OMAR

Asesor: Dr. Ing. PEDRO FREDDY HUAMANÍ NAVARRETE

LIMA – PERÚ

2021

DEDICATORIA

Dedico esta tesis a mis padres por haberme forjado con su ejemplo y determinación al igual que ser un ejemplo a seguir en mi vida; a mi hermana por darme los ánimos necesarios para poder lograr las metas trazadas.

Eder O. Moreano Rojas.

Dedico esta tesis a mi familia, en especial a mi mamita Lidia y a mi papito Alvino por brindarme su apoyo incondicional en mi vida universitaria, por enseñarme que mientras quiera y me concentre en un objetivo todo es posible, este logro les pertenece. Con mucho amor de mi para ustedes.

Lisset F. Maldonado Lezama

AGRADECIMIENTO

Agradecemos a nuestra casa de estudios la Universidad Ricardo Palma, a la Escuela Profesional de Electrónica y a todos los docentes que nos acompañaron en nuestra formación profesional en especial a nuestro asesor el Ingeniero Pedro Freddy Huamaní Navarrete.

Y un especial agradecimiento a nuestras familias, por darnos su incondicional apoyo a lo largo de nuestras vidas.

Eder Moreano y Lisset Maldonado

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El 11 de marzo de 2020 el COVID-19 fue declarado Pandemia por la OMS. Esta enfermedad infecciosa causada por el SARS-CoV-2 afecta al sistema respiratorio al igual que la Tuberculosis, este último según el informe de la OMS en el 2020 declara que los casos de TBC se incrementaron a nivel mundial a raíz de la pandemia del COVID-19.

Por ello, se ha visto afectado el reconocimiento de dichas enfermedades puesto que ambas presentan síntomas similares. Este proyecto propone el diseño y aplicación de Redes Neuronales Convolucionales para el diagnóstico de enfermedades como el Covid-19 y Tuberculosis, a partir de radiografías de tórax frontal. Es así que, dos de los modelos de redes neuronales fueron InceptionV3 y VGG16, mientras que el tercero fue denominado Particular, y todos ellos fueron desarrollados en la interfaz de JupyterLab, mediante el lenguaje de programación Phyton, con las librerías de Tensorflow y Keras. En el caso de la red neuronal particular, estuvo conformada por 10 capas, la cual fue entrenada por 2535 imágenes de radiografías de tórax frontal (compuesta por Covid-19, Tuberculosis y Sanos), asimismo para el entrenamiento se usó la CPU y GPU del computador. Estas mismas imágenes se usaron para el entrenamiento de los modelos de Redes neuronales artificiales INCEPTION Y VGG16, para poder comparar y obtener la eficacia de la red particular, A las redes neuronales artificiales se les aplicó Transfer Learning para poder implementarlas y entrenarlas.

Las métricas de evaluación para cada red fueron “Accuracy”, “Recall”, “F1-Score” y “General Precisión” para conocer la precisión y sensibilidad de la red particular a comparación de las otras mencionadas. Finalmente, la red neuronal convolucional particular obtuvo una precisión de 92.70% entrenada con la CPU, y 94.28% con la GPU, ambos valores totalmente aceptables para una red entrenada con 3 clases (Covid-19, Tuberculosis y Sanos).

Palabras Clave: Covid-19, Tuberculosis, InceptionV3, VGG16, radiografías de tórax frontal, Transfer Learning.

ABSTRACT

On March 11, 2020, COVID-19 was declared a Pandemic by the WHO. This infectious disease caused by SARS-CoV-2 affects the respiratory system like Tuberculosis, the latter according to the WHO report in 2020 states that TB cases increased worldwide as a result of the COVID pandemic -19.

For this reason, the recognition of these diseases has been affected since both present similar symptoms. This project proposes the design and application of Convolutional Neural Networks for the diagnosis of diseases such as Covid-19 and Tuberculosis, from frontal chest radiographs. Thus, two of the neural network models were InceptionV3 and VGG16, while the third was called Particular, and all of them were developed in the JupyterLab interface, using the Pyton programming language, with the Tensorflow and Keras libraries. In the case of the particular neural network, it was made up of 10 layers, which was trained by 2535 images of frontal chest X-rays (composed of Covid-19, Tuberculosis and Sanos), also for training the CPU and GPU of the computer. These same images were used for the training of the INCEPTION and VGG16 artificial neural network models, in order to compare and obtain the efficiency of the particular network. Transfer Learning was applied to the artificial neural networks in order to implement and train them.

The evaluation metrics for each network were “Accuracy”, “Recall”, “F1-Score” and “General Precision” to know the precision and sensitivity of the particular network compared to the others mentioned. Finally, the particular convolutional neural network obtained an accuracy of 92.70% trained with the CPU, and 94.28% with the GPU, both values totally acceptable for a network trained with 3 classes (Covid-19, Tuberculosis and Healthy).

Keywords: Covid-19, Tuberculosis, InceptionV3, VGG16, Frontal chest X-rays, Transfer Learning

INTRODUCCIÓN

En la actualidad, el sector salud del país se encuentra congestionado por la pandemia debido al COVID-19, siendo dificultoso acceder a una atención con un diagnóstico temprano. Por otro lado, en las enfermedades pulmonares como la Tuberculosis y Covid-19, los pacientes presentan síntomas iniciales similares. Es por ello que para el presente proyecto de tesis se diseñaron redes neuronales convolucionales, para el diagnóstico de enfermedades pulmonares, a partir de radiografías de tórax frontal, las cuales fueron desarrolladas en la interfaz JupyterLab, con las librerías de Tensorflow y Keras, usando el lenguaje de programación Python.

Asimismo, se diseñó el modelo de red neuronal convolucional particular la cual para poder ser evaluada se comparó con otros dos modelos de redes ya existentes las cuales son InceptionV3 y VGG16. Es por esto que este proyecto de tesis es de tipo experimental ya que se implementaron y diseñaron los modelos de redes neuronales convolucionales para el proceso de diagnóstico temprano de las enfermedades. Esto se demostró mediante los resultados obtenidos en las pruebas de validación; el método del proyecto es empírico ya que se basa en las experiencias obtenidas mediante el proceso de validación.

Por lo tanto, para este proyecto de tesis se estructuró de la siguiente manera:

En el capítulo 1 se plantea la formulación del problema, problemas específicos y problema general, así como también el objetivo general y los objetivos específicos, igualmente se menciona la importancia, justificación del proyecto, alcance y limitaciones de este.

En el capítulo 2 se conoce el marco teórico del presente proyecto, donde se exponen las investigaciones relacionadas con el tema y las bases teóricas del presente proyecto, indicando las métricas de evaluación para las redes neuronales convolucionales.

En el capítulo 3 se detalla el tratamiento y preprocesamiento de la base de datos, el diseño e implementación y entrenamiento de la red neuronal particular, InceptionV3 y la VGG16. Aquí se explican los algoritmos usados en cada uno de los pasos indicados los cuales llevarán luego a poder evaluar los modelos de redes.

En el Capítulo 4 se muestran las pruebas de validación realizada con cada uno de los modelos de redes neuronales convolucionales explicadas anteriormente, así como

también los resultados obtenidos de las métricas de evaluación propuestas inicialmente para poder identificar el desempeño de estas mismas. Así como también se presenta el cuadro comparativo de estas tres redes para la determinación de cual presentó una mejor precisión.

Finalmente, se tienen las conclusiones donde se describen los resultados obtenidos en el capítulo 4 y donde se verifica el cumplimiento de los objetivos planteados, así como también se presentan recomendaciones encontradas en el trabajo del presente proyecto.

CAPÍTULO I: PLANTEAMIENTO Y DELIMITACIÓN DEL PROBLEMA

1.1. Formulación del problema

Las enfermedades pulmonares como la Tuberculosis y COVID-19 presentan síntomas similares las cuales son: tos, fiebre, escalofríos, etc. Para ello al paciente le solicitan exámenes médicos, entre ellos una radiografía para poder diagnosticar que tipo de enfermedad tiene y así poder darle el adecuado tratamiento a tiempo. Sin embargo, actualmente con la pandemia del COVID-19 estas dos enfermedades pueden ser confundidas y puede darse un tratamiento incorrecto, lo cual puede llevar al deceso del paciente. Es por esto por lo que se busca tener un diagnóstico rápido y preciso respecto al tipo de enfermedad que pueda tener el paciente en sus pulmones, y con ello poder agilizar el tratamiento que recibirá dependiendo de la enfermedad que se le diagnostique. Esto mejoraría la recuperación del paciente ya que se actuaría con más anticipación y evitaría complicaciones de estas enfermedades.

1.1.1. Problema General

¿Como diseñar y aplicar una red neuronal convolucional para el diagnóstico de enfermedades pulmonares a partir de imágenes digitalizadas de radiografía de tórax frontal?

1.1.2. Problemas Específicos

a) ¿Como diseñar y entrenar una red neuronal convolucional particular, usando Python, Keras y Tensorflow, para el diagnóstico de dos enfermedades pulmonares, Tuberculosis y Covid-19, a partir de imágenes digitalizadas de radiografía de tórax frontal?

b) ¿Como entrenar dos modelos de redes neuronales convolucionales, Inceptionv3 y VGG16 aplicando transfer Learning, para el diagnóstico de enfermedades pulmonares a partir de radiografías de tórax frontal?

c) ¿Cómo evaluar las redes neuronales convolucionales aplicadas al diagnóstico de enfermedades pulmonares a partir de imágenes digitalizadas de radiografía de tórax frontal?

1.2. Objetivos

1.2.1. Objetivo General

Diseñar y aplicar una red neuronal convolucional para el diagnóstico de enfermedades pulmonares, a partir de imágenes digitalizadas de radiografías de tórax frontal.

1.2.2. Objetivos Específicos

a) Diseñar y entrenar una red neuronal convolucional particular, usando Python, Keras y Tensorflow, para el diagnóstico de dos enfermedades pulmonares, Tuberculosis y Covid-19, a partir de imágenes digitalizadas de radiografía de tórax frontal.

b) Entrenar dos modelos de redes neuronales convolucionales, Inception v3 y VGG16 aplicando Transfer Learning, para el diagnóstico de dos enfermedades pulmonares, Tuberculosis y Covid-19, a partir de imágenes digitalizadas de radiografías de tórax frontal.

c) Evaluar las tres redes neuronales convolucionales para el diagnóstico de dos enfermedades pulmonares, Tuberculosis y Covid-19, a partir de imágenes digitalizadas de radiografías de tórax frontal.

1.3. Importancia y justificación

1.3.1. Importancia

La importancia del presente proyecto es realizar un correcto descarte de dos enfermedades pulmonares como lo son la Tuberculosis y la Covid-19, aplicando redes neuronales convolucionales en las imágenes digitalizadas de radiografías de tórax frontal; de esta manera será posible agilizar el inicio del tratamiento debido a un correcto diagnóstico de dos enfermedades respiratorias.

1.3.2. Justificación

Actualmente debido a la pandemia ocasionada por la Covid-19, las instituciones de salud tanto públicas como privadas se encuentran colapsando por el número diario de personas que ingresan por esta enfermedad, ocasionando que el personal médico no se de abasto para realizar los

diagnósticos y tratamientos oportunos al paciente. Es por ello que este proyecto de investigación propone el diseño y aplicación de una red neuronal convolucional para el diagnóstico de Tuberculosis y Covid-19, la cual se cuenta con una base de datos de radiografías de tórax frontal, y será comparada con otros modelos establecidos para la comprobación del diseño propuesto.

1.4. Alcances y limitaciones

1.4.1. Alcances

El alcance de este proyecto es presentar una red neuronal convolucional particular que detecte enfermedades respiratorias como Covid-19 y Tuberculosis usando imágenes de radiografía de tórax frontal en pacientes que presentan dichos síntomas, así mismo no se pretende reemplazar o desplazar el diagnóstico brindado por el médico especialista.

1.4.2. Limitaciones

Esta investigación posee las siguientes limitaciones:

- El tamaño de la base de datos respecto a Covid-19, Tuberculosis y Personas Sanas se encuentra limitado a 2535 imágenes (845 para cada caso) ya que es la cantidad de radiografías de tórax frontal que se determinó para este proyecto, luego de haber sido revisadas y tratadas, para luego mediante un histograma se eligieron las más adecuadas. Estas imágenes fueron obtenidas de la base de datos de la IEEE Xplorer.
- Debido al alto uso del CPU por parte del entrenamiento de la red neuronal, se tiene limitada la cantidad de iteraciones/épocas para evitar que el entrenamiento demore demasiado tiempo.
- Es importante definir que la red neuronal particular, solo será entrenada con imágenes digitalizadas de radiografía de tórax frontal de personas Adultas con síntomas relacionados a la Covid-19 o Tuberculosis.

CAPÍTULO II: MARCO TEÓRICO

2.1. Marco Histórico

En el pasado para el análisis y diagnóstico de enfermedades pulmonares mediante el uso de la radiografía, solo lo podía dar un médico especialista con experiencia en estos casos, en la cual se basaba en una inspección a detalle de cada una de las partes de la radiografía y así confirmar el tipo de enfermedad que presentaba el paciente. Con el pasar de los años y la llegada de la era digital la medicina y tecnología evolucionaron, y con ello los diagnósticos y las radiografías. Con el uso masivo del open source y redes neuronales convolucionales se comenzaron a realizar varios proyectos dirigidos a distintos ámbitos de la medicina, para este caso el diagnóstico de enfermedades. Con el uso y desarrollo de redes neuronales convolucionales se ha dado un mayor apoyo en el diagnóstico de enfermedades. Por ello se busca una precisión y optimización en el diagnóstico de enfermedades y evitar posibles errores.

2.2. Investigaciones relacionadas con el Tema.

2.2.1. Antecedentes nacionales

Caya Pérez, J. (2020) en su investigación titulada “Evaluación de modelos de redes neuronales convolucionales aplicado a radiografías de tórax, para apoyar al proceso de diagnóstico de neumonía asociada al Covid-19”.

En esta tesis compara 3 tipos de redes neuronales, Resnet50, Inceptionv3 y su propia red neuronal en binario, para la detección de pacientes con COVID o pacientes sanos. Nos presenta estos tres tipos de redes y las modifica para obtener una mejor detección en las radiografías de Tórax, donde en su primera red creada le asigna 10 capas con 2 salidas previamente entrenadas con su data set. Así mismo modifica la red Resnet50 e InceptionV3 quitando la capa de predicción include_top=false. Nos indica que la red con mejor desempeño fue la InceptionV3 ya que en sus pruebas le da una exactitud de 0.9886. Nos recomienda tener una mayor dataset para poder entrenar mejor la red neuronal y dar mejores resultados, así como también poder agregar más capas convolucionales.

2.2.2. Antecedentes internacionales

Chan, L (2017) en su artículo "TX-CNN: Detecting tuberculosis in chest X-ray images using convolutional neural network".

Indican el uso de dos arquitecturas de red neuronal ALEXNET y GOOGLENET para poder acelerar la detección de tuberculosis en Lima, Perú. En el uso de sus redes neuronales agregan una capa softmax para obtener la puntuación de cada categoría el cual cada puntaje va de 0 a 1 y representa la probabilidad de clasificar correctamente la manifestación. A su vez utilizaron una estructura de una red estándar de siete capas las cuales constan de cinco capas convolucionales y dos completamente conectadas, Implementaron Dropout y Relu para abordar el problema del sobreajuste y convergencia. Para el caso del Googlenet lo implementaron usando la arquitectura caffe y lo entrenaron utilizando un descenso gradual estocástico (SGD). También experimentando encontraron que para la red Alexnet con una base de aprendizaje de 0.01, un impulso de 0.9 y caída de peso de 0.0005, produce el mejor resultado en un tiempo razonable en cambio para googlenet se usó una taza de aprendizaje con base 0.001 un impulso de 0.9 y una disminución del peso de 0.0002. Indican que respecto a sus resultados el uso de la red Alexnet tuvo una precisión de 85.68% y Googlenet un 91.72%.

Guangyu (2021) en su artículo "Classification of COVID-19 chest X-Ray and CT images using a type of dynamic CNN modification method".

propone un tipo de método de modificación dinámica de CNN para la clasificación de dos conjuntos de datos de imágenes COVID-19 CXR y un conjunto de datos de imágenes CT. El método propuesto establece conexiones entre diferentes capas de la arquitectura CNN original a través de bloques de convolución puntual, que logran combinaciones dinámicas de diferentes capas. Se emplearon y compararon con el método propuesto de seis algoritmos de aprendizaje profundo ampliamente utilizados, así como dos modelos publicados recientemente diseñados específicamente para la detección de COVID-19. Los resultados se analizan mediante la exactitud de la prueba, la sensibilidad, la precisión, la prueba de robustez y los mapas de activación de clase. La arquitectura CNN modificada demostró un

rendimiento de clasificación satisfactorio en su estudio comparativo, que muestra su potencial para ser aplicado en entornos clínicos para el diagnóstico asistido por computadora de casos positivos de COVID-19. Como se menciona en la Sección 2, empleamos VGG16, Inceptionv3, ResNet18_v1, DenseNet121, MobileNetv3_small y SqueezeNet1.0 para la detección de COVID-19 en este documento. Las versiones del modelo están determinadas por la compensación del desempeño de la clasificación y los costos computacionales. Por ejemplo, en nuestros experimentos, ResNet18 y ResNet50 alcanzaron una precisión similar en la clasificación. En este caso, elegimos ResNet18 en lugar de ResNet50 debido a su mayor eficiencia de parámetros y menor costo computacional. Además, los modelos se seleccionaron en función del rendimiento de clasificación informado en la literatura relacionada. Durante los experimentos, los modelos anteriores se establecieron para ser el modo pre entrenado. Por lo tanto, poseen los pesos y los hiper parámetros ajustados en ImageNet. Cómo se investigan 5 clases en este documento, la capa de salida en los modelos anteriores se revisa para tener 5 nodos.

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

Presentaron tres modelos de redes neuronales convolucionales pre-entrenadas e implementadas en Python haciendo el uso de las librerías de Keras y Tensorflow, para poder implementar estos tres tipos de modelos de redes neuronales usaron la técnica de Transfer Learning. El objetivo de la investigación es poder resolver los problemas de detección o clasificación de los patrones asociados a las enfermedades en distintas imágenes médicas. En la investigación, las imágenes utilizadas fueron las resonancias magnéticas del cerebro humano, tomografías y radiografías del tórax de personas adultas. Se usó primero el modelo pre entrenado InceptionV3 el cual se le incluyó en una base de datos de 13 pacientes con tumores cerebrales del Instituto Tecnológico de Montreal en el 2010. Este modelo de red fue entrenado para poder identificar estos tumores cerebrales, los resultados confirman que el modelo InceptionV3 pudo clasificar imágenes

de tejido tumoral. Para el siguiente caso este mismo modelo de red fue entrenada con imágenes de radiografías de 5 tipos de tuberculosis con más de 200mil muestras. Luego de la evaluación se logró alcanzar un mejor comportamiento de la red y esto se debió a la cantidad inmensa de muestras por clases. Para el último caso se usó el modelo de red CheXNet, para poder detectar la neumonía en las imágenes de radiografías del tórax para esto se usó la base de datos proporcionada por la Sociedad Radiológica de América del Norte (RSNA). Como resultado se obtuvo una mejora en la clasificación de imágenes de radiografías con o sin neumonía. Además, los autores proporcionaron ejemplos de implementación de una red neuronal convolucional para la comprensión de los modelos de redes que se usaron. Los autores recomiendan el uso del PCA (Análisis de componentes principales) para poder tener una mejor descomposición de las imágenes previo a la capa de entrada y así tener mejores resultados de la red neuronal.

2.3. Estructura teórica y científica que sustenta el estudio

2.3.1. Red Neuronal Convolucional

En inglés Convolutional Neural Networks, según Eduardo (2017, p. 23) define que una red neuronal es un procesador paralelo masivamente distribuido que tiene una facilidad natural para el almacenamiento de conocimiento obtenido de la experiencia para luego hacerlo utilizable, el cual se parece al cerebro en dos aspectos:

- El conocimiento es obtenido por la red a través de un proceso de aprendizaje.
- Las conexiones entre las neuronas, conocidas como pesos sinápticos son utilizadas para almacenar dicho conocimiento.

Según Mathworks (2020) al igual que otras redes neuronales, una CNN está compuesta por una capa de entrada, una capa de salida y muchas capas intermedias ocultas. Estas capas realizan operaciones que alteran los datos con el objetivo de aprender características específicas de dichos datos. Las 3 capas más frecuentes son: convolución, activación o ReLU, y pooling.

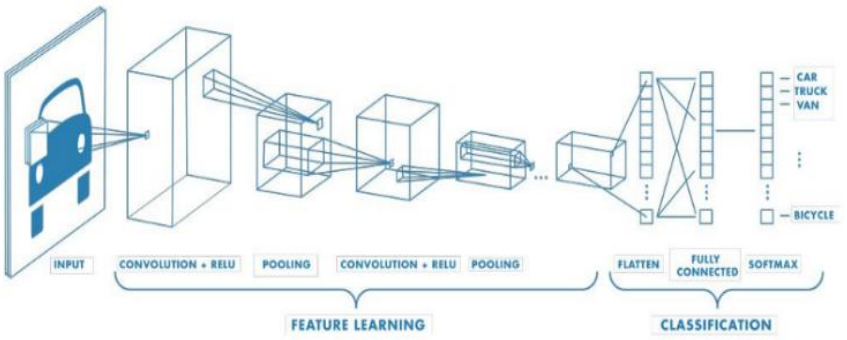


Figura Nº 1: Arquitectura de una red neuronal convolucional.
Fuente: Mathworks (2020).

Capa de Entrada:

Aquí se asigna el tamaño de las dimensiones de los parámetros de la entrada (ancho, alto y profundidad). Las imágenes a color constan de 3 canales (RGB) donde 3 es la profundidad y para el caso de imágenes a escala de grises sería profundidad 1.

Capa Pooling:

Es una capa de agrupación que proporciona una operación de reducción de resolución típica, la cual reduce la dimensionalidad en el plano de los mapas de características para introducir una invariancia de traducción a pequeños cambios y distorsiones, y disminuir el número de parámetros que puede aprender posteriormente. Según Yamashita (2018) señala que no hay ningún parámetro que se pueda aprender en ninguna de las capas de agrupación, mientras que el tamaño del filtro, la zancada y el relleno son hiper parámetros en las operaciones de agrupación, similares a las operaciones de convolución.

Capa Convolución:

Según Yamashita (2018) una capa de convolución es un componente fundamental de la arquitectura de la CNN que realiza extracción de características, que normalmente consiste en una combinación de operaciones lineales y no lineales, es decir, operación de convolución y función de activación. En ella se aplica una pequeña matriz de números

llamada kernel, a través de la entrada que es una matriz de números, llamada tensor. Un producto de elementos entre cada elemento
del núcleo y el tensor de entrada se calcula en cada ubicación del tensor y se suma para obtener el valor de salida en la posición correspondiente del tensor de salida, llamado mapa de características.

Capa Dropout:

Según Alzubaidi (2021) es una técnica ampliamente utilizada para la generalización. Durante cada época de entrenamiento, las neuronas se dejan caer al azar. Al hacer esto, el poder de selección de características se distribuye equitativamente entre todo el grupo de neuronas, además de obligar al modelo a aprender diferentes características independientes. Durante el proceso de entrenamiento, la neurona caída no será una parte de la propagación hacia atrás o hacia adelante. Por el contrario, la red a gran escala se utiliza para realizar la predicción durante el proceso de prueba.

Capa Flatten:

Esta capa convierte los elementos de un matriz a un solo plano, esto quiere decir que todas toma toda la data de su entrada y la aplana lo que conlleva a que pase de multidimensional a unidimensional, esto permite que el procesador tenga menos carga computacional.

INCEPTION V3:

Según Gomez (2019). Es un modelo de red neuronal la cual fue introducida en el año 2015 con el nombre de GoogleNet. Este modelo de red contiene 48 capas y estas mismas pueden clasificar hasta 1000 clases distintas. Tiene como un tamaño máximo de entrada de 299 x 299, además de contar con cuatro módulos de ramas en paralelo las cuales se conforman por: una convolución de 1x1 seguida de dos convoluciones de 3x3 una convolución de 1x1, otra convolución de 3x3, un pooling, otra convolución de 1x1 y al final una convolución de 1x1. La salida de este módulo termina siendo una concatenación de cuatro ramas. siendo la salida del módulo base la concatenación de las cuatro ramas.

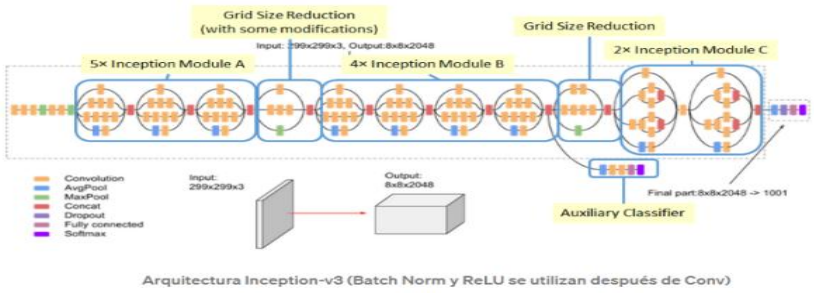


Figura Nº 2: Arquitectura de InceptionV3.

Fuente: Google Cloud (2021).

VGG16:

Visual Geometry Group (VGG) inventó el VGG-16 que tiene 16 capas convolucionales y 3 completamente conectadas, lo cual lleva consigo la parte tradicional de ReLu (AlexNet). VGG-16 apila muchas más capas en AlexNet y usa filtros con un tamaño menor (2x2 y 3x3). Esta red consta de 138mil parámetros y ocupa 500MB de almacenamiento.

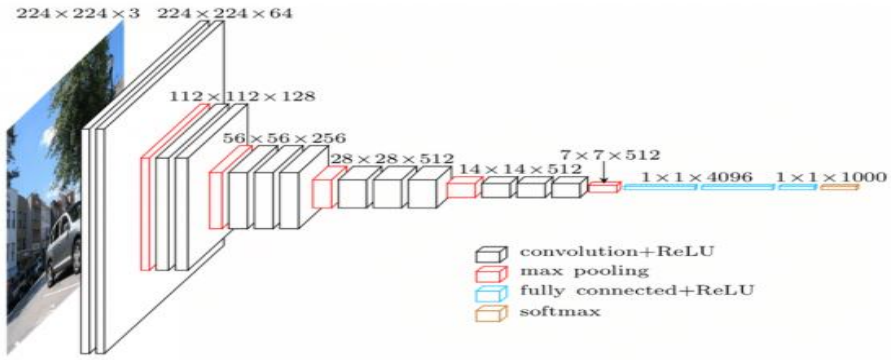


Figura Nº 3: Arquitectura de VGG16.

Fuente: Jose, Manolis (2021).

Métricas de Evaluación:

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 tres columnas, que representan las posibilidades que se pueden obtener. La cual se puede observar en la Figura N º4.

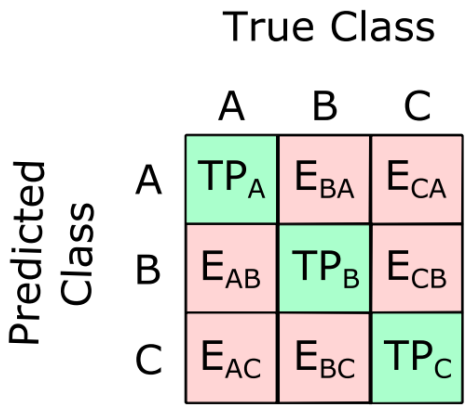


Figura Nº 4: Matriz de confusión para multiclases.
Fuente: Google Cloud (2021).

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.

Según Mohajon (2020) estas variables que nos entrega la matriz de confusión pueden ser usadas en ecuaciones para obtener las siguientes métricas como por ejemplo el Acurracy, precision, recall, specifity, f1-core. las cuales se detallan a continuación:

Accuracy: Hace referencia a la exactitud, respecto a lo cerca que se encuentra el resultado de ofrecer una medición de valor verdadero. Esta se puede definir con la siguiente ecuación:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

Precisión: Nos permite medir la calidad del algoritmo en la tarea de clasificación y saber los verdaderos positivos entre los demás resultados positivos obtenidos. Esto se consigue mediante la siguiente ecuación:

$$Precision = \frac{TP}{TP+FP} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2)$$

Recall: Nos proporciona el total de casos positivos que son correctamente predichos como positivos segun el algoritmo. Esta métrica se obtiene de la ecuación:

$$Recall = \frac{TP}{TP+FN} \quad \text{.....(3)}$$

Specificity: Nos proporciona el total de casos negativos que son correctamente predichos como negativos. Para obtener este parámetro se usa la siguiente fórmula:

$$Specificity = \frac{TN}{TN+FP} \quad \text{.....(4)}$$

F1-score: Es el resultado de la combinación entre Precisión y Recall para la obtención de un solo parámetro que represente a los dos. Para obtener este parámetro se usa la siguiente fórmula:

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad \text{.....(5)}$$

2.3.2. Radiografía de Tórax Frontal:

Mayo Clinic (2021) define que la radiografía de tórax produce imágenes del corazón, los pulmones, los vasos sanguíneos, las vías respiratorias y los huesos del tórax y de la columna vertebral. Los rayos X de tórax también pueden revelar la presencia de líquidos dentro de los pulmones o alrededor de ellos, o la presencia de aire rodeando a los pulmones del afectado. Una radiografía de tórax revela muchos detalles dentro del cuerpo, entre ellos, el estado de los pulmones, problemas pulmonares relacionados con el corazón, el tamaño y el contorno del corazón, vasos sanguíneos, calcificación, fracturas, cambios postoperatorios, marcapasos, desfibrilador o catéter.

2.4. Definición de términos básicos

- Python: Es un lenguaje de programación interpretado, orientado a objetos de alto nivel y con semántica dinámica. Su sintaxis hace énfasis en la legibilidad del código, lo que facilita su depuración y, por tanto, favorece la

productividad. Además, ofrece la potencia y la flexibilidad de los lenguajes compilados con una curva de aprendizaje sencilla. (LUCA AI Powered Decisions, 2021)

- Tensorflow: Es una plataforma de código abierto de extremo a extremo para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que les permite a los investigadores innovar con el aprendizaje automático y, a los desarrolladores, compilar e implementar con facilidad aplicaciones con tecnología de AA. (Tensor Show,2021).
- Keras: Es una interfaz de programación nivel alto el cual pertenece a TensorFlow desde el 2017, para poder construir la infraestructura del aprendizaje profundo reduciendo el tamaño del código en el algoritmo. (TensorFlow, 2020).
- Transfer learning: Es una técnica de aprendizaje la cual es empleada en Deep Learning y utilizada mayormente en modelos de redes neuronales pre entrenadas o en redes neuronales profundas. Esta técnica modifica la arquitectura de la misma red neuronal, agregando nuevas capas, las cuales se estrenarán, mientras que el resto de las capas serán "paralizadas" para mantener el mismo valor de su "peso" obtenido por el preentrenamiento. El objetivo de esto es obtener una red mucho más eficiente con respecto a su clasificación de imágenes, así como mejorar y disminuir el esfuerzo del cpu en el entrenamiento (POCHO COSTA, 2019).
- Covid 19: Según la Organización Mundial de la Salud (2020), el COVID-19 es la enfermedad causada por el nuevo coronavirus conocido como SARS-CoV-2. La OMS tuvo noticia por primera vez de la existencia de este nuevo virus el 31 de diciembre de 2019, al ser informada de un grupo de casos de «neumonía vírica» que se habían declarado en Wuhan (República Popular China)
- Tuberculosis: Según la OMS (2020) la tuberculosis es causada por Mycobacterium tuberculosis, una bacteria que casi siempre afecta a los pulmones. Se trata de una enfermedad curable y prevenible.
  La infección se transmite de persona a persona a través del aire. Cuando un enfermo de tuberculosis pulmonar tose, estornuda o escupe, expulsa bacilos

tuberculosos al aire. Basta con que una persona inhale unos pocos de estos bacilos para quedar infectada.

- JupyterLab: es un entorno de desarrollo interactivo basado en la web para blocs de notas, código y datos de Jupyter. Permite configurar y organicar la interfaz de usuario para admitir una amplia gama de flujos de trabajo en ciencia de datos, computación científica y aprendizaje automático. (JupyterLab,2020)

2.5. Diseño de la Investigación

2.5.1. Variables de investigación

Variable independiente: Red neuronal Convolucional
Variable dependiente: Diagnóstico de enfermedades pulmonares e independiente.

2.5.2. Tipo y Método de investigación

El tipo de investigación es Aplicada y Tecnológica.
El método de investigación es del tipo empírico y experimental. Esto se debe a que 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 está orientada a la elección del investigador. Estas son imágenes de radiografías de tórax frontal que fueron adquiridas por medio de dos bases de datos; la primera de la base de datos de IEEE DATASETS llamada "TUBERCULOSIS (TB) CHEST X-RAY DATABASE" y la segunda, del autor Tawsifur en la plataforma Kaggle (2021).

2.5.3. Técnicas e Instrumentos de recolección de datos

El instrumento utilizado para la recolección de datos es la base obtenida de Tuberculosis (TB) chest X-ray database IEEE DataPort (2020) para el caso de Tuberculosis y del usuario Tawsifur Rahman en su su dataset Chest X-ray Covid-19 de la plataforma Kaggle(2021) para el caso de Covid-19. La

técnica de revisión de las imágenes digitalizadas de las bases de datos mencionadas anteriormente.

2.5.4. Procedimiento para la recolección de datos

Para el procedimiento de recolección de datos se procedió de la siguiente manera:

-Se buscaron las imágenes en las bases de datos de IEEE DataPort y Kaggle.

-Se procedió a revisar las bases de datos citadas anteriormente que contengan radiografías hechas a personas adultas y de ambos sexos.

-Se revisó si las bases de datos son de una fuente confiable.

-El tipo de imágenes que se procedió a buscar fueron radiografías de tórax frontal.

-Mediante un algoritmo se sacó el promedio de píxeles de todas las imágenes de la base de datos inicial.

-Se realizó un Histograma con el promedio de píxeles de cada imagen de la base de datos.

-Se define un rango para el promedio de píxeles de las imágenes para poder así quedarse con solo un porcentaje de la base de datos que son las imágenes óptimas para nuestro entrenamiento de la red neuronal.

CAPÍTULO III: DESARROLLO PROYECTO

En este capítulo se muestra el diseño y desarrollo del proyecto en el cual se presenta el diagrama de flujo general, el tratamiento y evaluación de la base de datos, el preprocesamiento de las imágenes, la arquitectura de red para cada modelo, la implementación y entrenamiento de los 3 modelos de redes (Particular, InceptionV3 y VGG16).

3.1. Diagrama de Flujo

En la Figura N° 5, se muestra el diagrama de Flujo general del proyecto de tesis planteado. Donde, se presentan dos etapas fundamentales, la cual para el caso de la primera etapa lo conforma el tratamiento y evaluación de imágenes, para el caso de la etapa 2 está conformada por los modelos de redes neuronales mencionadas en este proyecto de tesis, las cuales fueron usadas para la detección de enfermedades respiratorias en pacientes con Tuberculosis y Covid-19, a partir de las radiografías de tórax frontal.

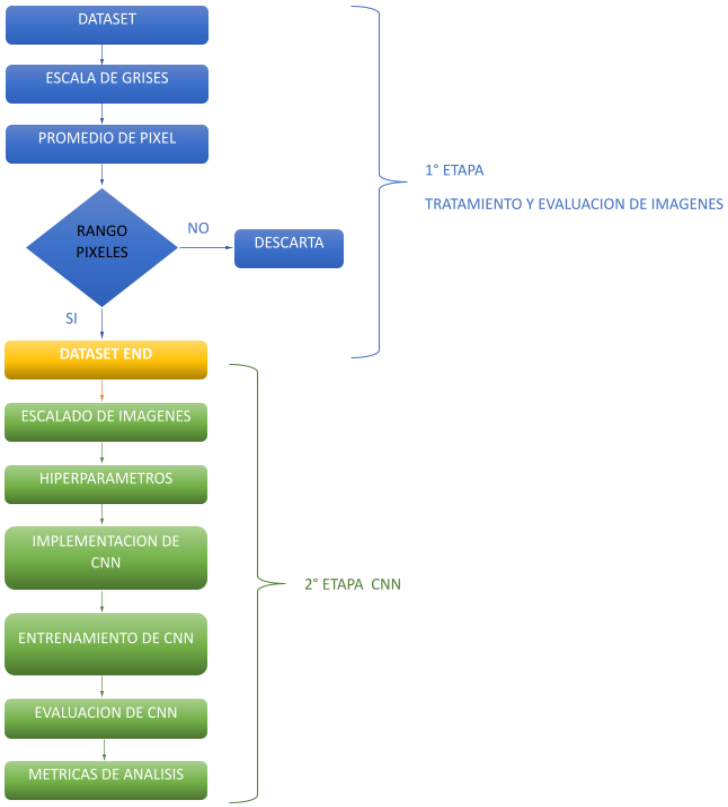


Figura Nº 5: Diagrama de Flujo de la Red Neuronal Convolucional.

Fuente: Elaboración propia.

3.2. Tratamiento y evaluación de las imágenes de las bases de datos.

Para el tratamiento y evaluación de las imágenes se trabajó con 3 bases de datos, (Tuberculosis, Covid-19 y Sano). Para esta sección se muestra el trabajo realizado para una de las bases de datos, en este caso Tuberculosis, dado que se utilizó la misma metodología para las demás clases (Sano y Covid-19).

3.2.1. Tratamiento de las imágenes de la Base de Datos

Para el tratamiento de imágenes, primero se estableció convertir todas las imágenes en formato de escala de grises, ya que con esto se tuvo características similares entre las imágenes de cada clase. Para ello se usó el algoritmo que se muestra en la Figura Nº 6, el cual permitió llamar todas las imágenes de la dirección de "Imput_dir" para convertirlo en escala de grises y luego guardarlo en la dirección establecida para "Out_dir()". Para lograr hacer la conversión de grises se usó el comando "I.convert('L')", el cual permitió pasar las imágenes RGB a escala de grises cuando se le da la condición de 'L'.

```
In [11]: from PIL import Image
         import os
          #Se convertira en escala de grises.
         Input_dir = 'C:/Users/Lisset/Desktop/AQUI/Base total de datos/BASE RECOLECTADA/Tuberculosis/'
         Out_dir = 'C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS/'
         a=os.listdir(Input_dir)
         for i in a:
             print(i)
             I = Image.open(Input_dir+i)
             L = I.convert('L')
             L.save(Out_dir+i)
```

Figura Nº 6: Algoritmo de la conversión de escala de grises.
Fuente: Captura de pantalla de la plataforma JupyterLab.

Teniendo las imágenes de la base de datos en escalas grises se procedió a calcular el promedio de píxeles de cada imagen para poder definir qué imágenes tienen una buena o mala calidad. Esto se puede observar en la Figura Nº 7 en la cual el algoritmo mostrado permite de manera global mostrar el promedio de píxeles de las imágenes, esto se hizo para cada uno de los casos (Tuberculosis, Covid-19 y Sano)

```
In [16]: import cv2
         import os
         import glob
         import numpy
         img_dir = "C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS/" # ubicacion del grupo de imagenes a trabajar
         data_path = os.path.join(img_dir,'*png') # Concatena todos los archivos que tengan la extension .png
         files = glob.glob(data_path) # Llama de forma independiente al grupo de imagenes que pertenecer al data_path
         #data = []
         for f1 in files:
             img = cv2.imread(f1) # lee cada imagen de entrada
             #data.append(img)
             avg_color_per_row = numpy.average(img, axis=0) # Es un promedio de los pixeles de cada imagen de entrada
             avg_color = numpy.average(numpy.average(avg_color_per_row, axis=0)) #Un promedio del promedio de estas entradas
             print(avg_color,f1)# Se vizualizaa el promedio de pixeles que le corresponde a cada imagen

93.02720642089844 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1).png
129.9228286743164 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (10).png
93.22840118408203 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (100).png
123.24304580688477 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1000).png
129.41646575927734 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1001).png
121.13271713256836 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1002).png
108.43621444702148 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1003).png
135.9182243347168 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1004).png
133.25376892089844 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1005).png
134.54939651489258 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1006).png
144.33415985107422 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1007).png
141.82989883422852 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1008).png
111.81512069702148 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1009).png
86.61870574951172 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (101).png
140.27645874023438 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1010).png
161.56848907470703 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1011).png
141.35161972045898 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1012).png
99.30137252807617 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1013).png
131.42414474487305 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1014).png
133.02110290527344 C:/Users/Lisset/Desktop/TESIS 2021/TUBERCULOSIS\Tuberculosis (1015).png
```

Figura Nº 7: Algoritmo para el cálculo del promedio de píxeles por imágenes.

Fuente: Captura de pantalla de la plataforma JupyterLab.

3.2.2. Evaluación y selección de las imágenes de la Base de Datos

Los datos brindados por el programa fueron trasladados a un archivo de Excel, donde se procedió a realizar un histograma para saber dónde se ubicaba el mejor promedio de píxeles. En la Figura Nº 8 se muestra el histograma luego de haber retirado las imágenes que presentaban un mejor promedio de píxeles, las cuales fueron usadas para el entrenamiento de los modelos de redes neuronales. Por lo cual las imágenes que se encontraron en los rangos mostrados en el histograma pasaron a ser eliminadas ya que no presentan una calidad óptima para el entrenamiento.

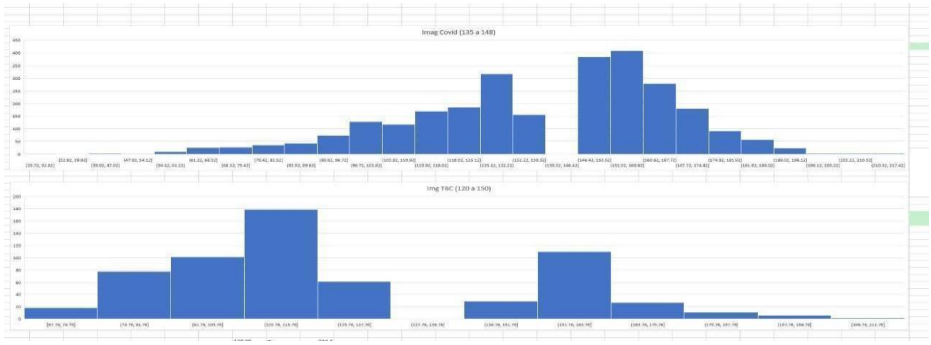


Figura Nº 8: Histograma de las imágenes descartadas por promedio de píxeles.

Fuente: Captura de pantalla de la hoja de cálculo de Excel.

Se decidió escoger varias imágenes con buena calidad, por la cual se obtuvo el rango de promedio de píxeles. En el Anexo N°1 se muestran las imágenes con promedio de píxeles de 80, 120, 150, de las cuales se concluyó que las imágenes en el rango de 120 a 140 son imágenes que presentan mayor calidad para el entrenamiento de las redes.

Para la eliminación de las imágenes, se usó el Excel y Command prompt debido a que las imágenes a descartar no son correlativas. Para este caso se procedió a usar la función "=TRANSPONER" en el archivo de Excel, el cual nos permitió pasar los datos de la columna a una celda en la cual los datos se mostraban concatenados separados por coma. A continuación, en la Figura N° 9 se muestra el uso de la función "TRANSPONER".

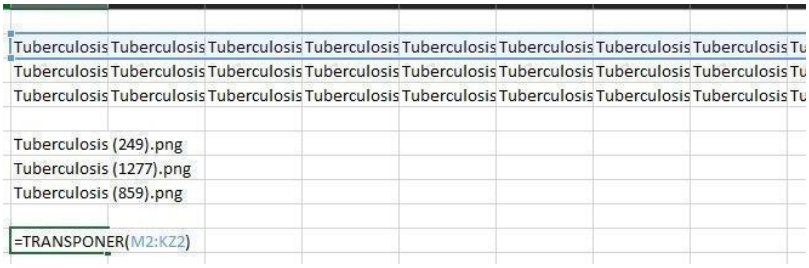


Figura Nº 9: Ordenamiento del Promedio de píxeles con el nombre de imagen correspondiente.

Fuente: Captura de pantalla de la hoja de cálculo de Excel.

Estos datos concatenados fueron ingresados al Command prompt para realizar la eliminación de forma masiva en la carpeta donde se ubicaron estas imágenes. Para el caso de Tuberculosis se obtuvieron inicialmente 1 465 imágenes de radiografías de tórax frontal, pero de acuerdo al rango de píxeles establecido, se establece que la base de datos final, es de 845 imágenes, siendo el restante 620 imágenes, las cuales se eliminaron de la base de datos inicial. Este procedimiento se replicó para el caso de Covid-19 y Sano. En la Figura N° 10 se muestra el uso del "Command Prompt" para la eliminación de las imágenes con promedio de píxeles fuera del rango estimado.

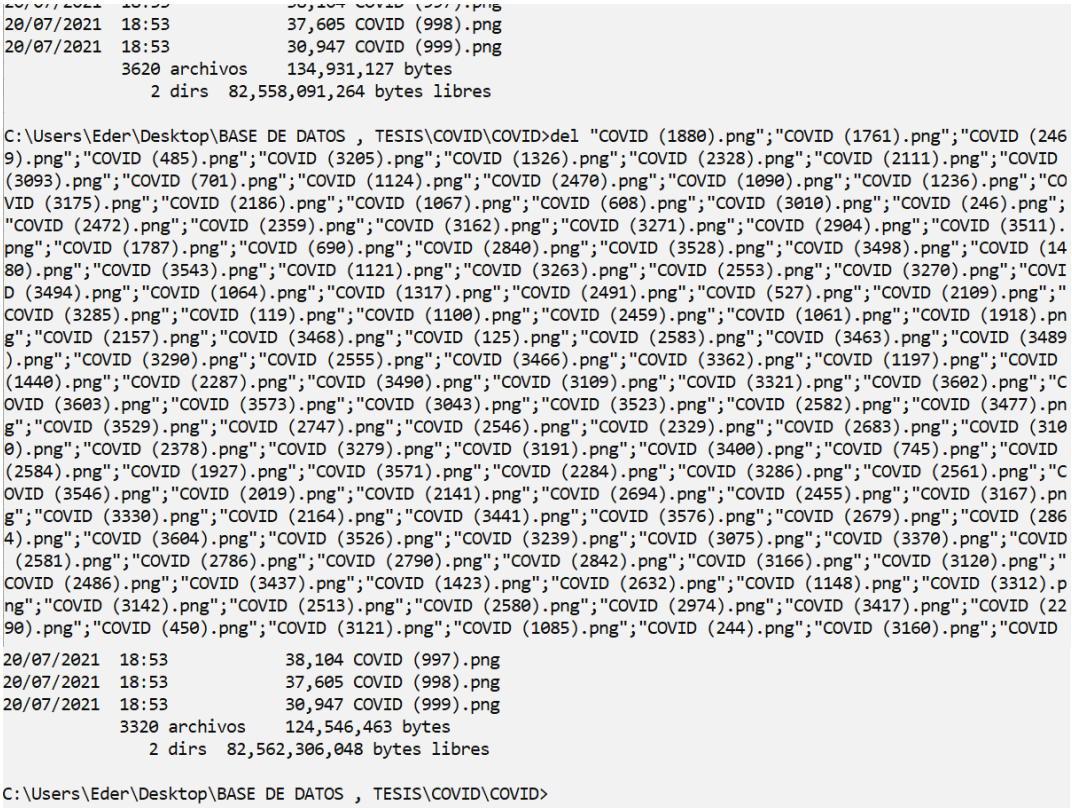


Figura Nº 10: Eliminación de imágenes no óptimas para la base de datos.

Fuente: Captura de pantalla del Command Prompt.

Luego de finalizar todo este proceso se obtuvo un total de 2 535 imágenes para Covid-19, Tuberculosis y Personas Sanas. Donde el 80% (676 imágenes para cada clase) de cada clase se puso en la carpeta Train y el 20% en la carpeta Val (169 imágenes de validación para cada clase), y un 20% restante de val y train se copió en una carpeta llamada testeo para su posterior uso en el capítulo 4.

3.3. Preprocesamiento de las imágenes

Luego de haber obtenido las imágenes correctas de la base de datos final, se procedió a definir los hiper parámetros que se usaron para la etapa del entrenamiento de los modelos de redes, las cuales fueron “épocas” y “batch size”. Para el caso de “épocas”, esto define la cantidad de veces que se va a iterar los modelos de redes para su entrenamiento y con respecto al “batch size”, este nos indica la cantidad de imágenes que ingresaron en cada “paso”. El “paso” indica el número de veces que la información fue procesada dentro de cada época, cuando se realizó el entrenamiento de las redes neuronales.

En la siguiente Figura N° 11 se muestran los hiper parámetros y las variables "train_data_dir" y "validation data dir" las cuales se asocian a las rutas donde se ubican las imágenes de Train y Val.

```
[2]: #ALMACENAMIENTO DE DIRECTORIOS
     train_data_dir = 'E:/TESIS 2021/Tesis 3 clases/dataset/train'
     validation_data_dir = 'E:/TESIS 2021/Tesis 3 clases/dataset/val'

[4]: #HIPERPARÁMETROS
     epocas=120
     batch_size = 64
```

Figura Nº 11: Hiperparametros para la Red Neuronal Particular.

Fuente: Captura de pantalla de la plataforma JupyterLab.

Para el preprocesamiento de imágenes se necesitó importar la librería de Tensorflow el cual fue instalado en su versión 2.1(para el uso del GPU) ya que en la API de Keras (versión 2.4.3) venía con la versión 2.5, para su uso se necesitó tener instalado CUDA 11.3 de NVIDIA y cuDNN 8.2. Para implementar los modelos de redes neuronales se necesitó de Keras ya que este incluye librerías para la implementación y entrenamiento de estas mismas, las cuales fueron de mucha ayuda para el desarrollo de la presente tesis. De todas las librerías que contiene Keras se optó por usar la ImageDataGenerator para realizar el preprocesamiento de imágenes, ya que esta permite dividir y generar el proceso de entrenamiento en batchs(bloques), además de que realiza data augmentation. Data augmentation es la técnica usada cuando se cuenta con una base de datos no tan amplia, ya que nos permite modificar las imágenes y aumentar la cantidad de estas mismas aplicando pequeñas modificaciones (zoom, horizontal flip, shear range). En la Figura N° 12 se observa el uso de la librería ImageDataGenerator y dentro de esta definimos los siguientes parámetros:

- Rescale =1. /255, este parámetro indica que las imágenes que están en el directorio train_datagen se reescala , ya que el rango de píxeles es de 0 a 255 y esto lo vuelve de 0 a 1, y pasa de tener una profundidad de 3 a 1, a esto se le conoce como normalización.
- Shear_range , esto genera imágenes con inclinación para que la red aprenda a trabajar con imágenes que no están bien encuadradas.

- Zoom_range, este parámetro realiza zoom a imágenes aleatorias, esto permite que la red aprenda a trabajar con imágenes que no estén completas.
- Horizontal_flip, este parámetro invierte la imagen de manera horizontal, para que la red pueda aprender y distinguir imágenes con distinta direccionalidad.

```
#PREPROCESAMIENTO DE IMÁGENES
#SIN DATA AUGMENTATON

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True
)

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

Figura Nº 12: Algoritmo de preprocesamiento de imágenes.

Fuente: Captura de pantalla de la plataforma JupyterLab.

Luego de indicar los parámetros, estos fueron usados para definir las variables "train_generator" y "val_generator", las cuales contienen las imágenes preprocesadas con los parámetros indicados previamente. Dentro de los cuales se declara un "target_size", el cual permite redimensionar el tamaño de las imágenes a una especificada, el cual para la red particular y VGG16 es de 224x224 y para el modelo InceptionV3. En la Figura N° 13 se observa el algoritmo que se usó para el entrenamiento y validación de las imágenes del modelo particular.

```
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 2028 images belonging to 3 classes.
Found 507 images belonging to 3 classes.
```

Figura Nº 13: Algoritmo de parámetros para el entrenamiento y validación de la red neuronal.

Fuente: Captura de pantalla de la plataforma JupyterLab.

3.4. Implementación y Entrenamiento de la Red Neuronal particular

En esta sección se desarrolló la implementación y entrenamiento de la red neuronal particular utilizando las librerías de Tensorflow, Keras y el lenguaje de programación Python.

3.4.1. Implementación

La red neuronal particular cuenta con 10 capas descritas y su estructura se presenta en la Figura N° 14.

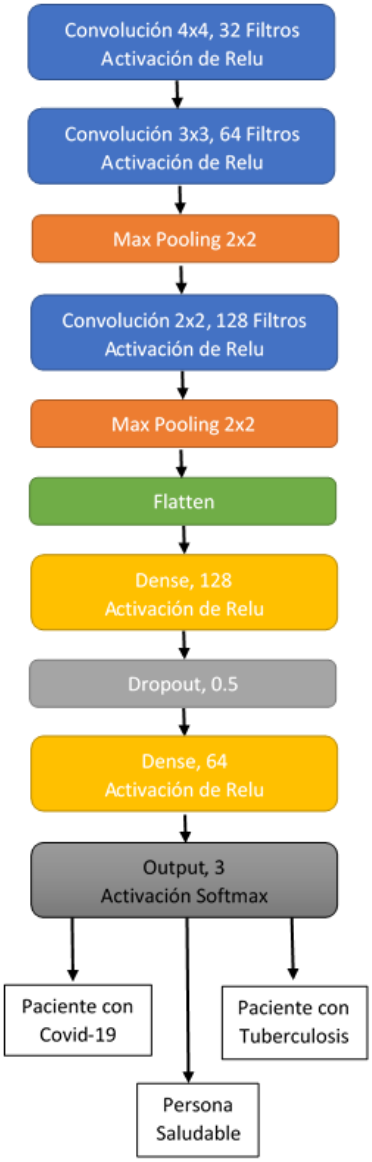
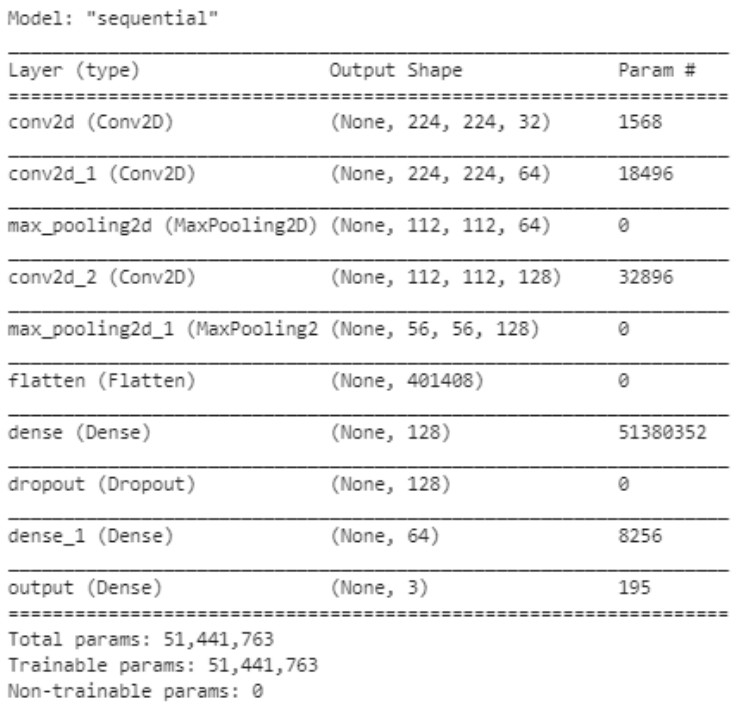


Figura Nº 14: Arquitectura de la Red neuronal Particular.

Fuente: Elaboración propia.

El diseño presentado de la red neuronal particular está basado en el método de prueba y error, los cuales fueron analizados para corregir, agregar o quitar ciertas capas de la red logrando implementar una red particular óptima para la detección de Tuberculosis, Covid-19 y personas sanas. Para el diseño de esta red se tomó como referencia la red implementada por Colula y Ángel (2019), el cual permitió implementar una red neuronal convolucional utilizando TensorFlow y Keras, así mismo se consideró lo realizado por Narin et al. (2020) el cual menciona la aplicación de filtros para lograr afinar la red respecto a la extracción de características de las imágenes, también al uso de la activación softmax para la capa de salida la cual permite clasificar las 3 clases descritas y así minimizar las dimensiones en el proceso de la red. Para poder reducir la posibilidad de sufrir un sobre ajuste se tomó en consideración lo realizado por Sharma et al. (2020) el cual hace uso de la capa Dropout para el proceso de entrenamiento de los modelos de redes neuronales convolucionales. Todo esto se puede observar en la Figura N° 15.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 32)      1568
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 64)      18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 128)     32896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 401408)            0
_________________________________________________________________
dense (Dense)                (None, 128)               51380352
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
output (Dense)               (None, 3)                 195
=================================================================
Total params: 51,441,763
Trainable params: 51,441,763
Non-trainable params: 0
```

Figura Nº 15: Implementación de la Red Neuronal Convolucional Particular.

Fuente: Captura de pantalla de la plataforma JupyterLab.

Este modelo consta con 10 capas implementadas que va desde el preprocesamiento hasta la salida multiclase, las cuales se detallan a continuación:

- La primera capa de la red es una convolución la cual contiene 32 filtros kernels con un tamaño de 4x4, un input shape de 224x224 que es el tamaño de la imagen que ingresó y la función de activación que se usó fue ReLU. El funcionamiento de esta primera capa es agrupar píxeles cercanos de la imagen entrante y realiza un producto escalar, luego se aplica la activación ReLU para convertir los negativos en ceros y con ello se obtuvieron 32 matrices de salida el cual da un total de 1 605 632 neuronas para la primera capa.
- Para la segunda capa de la red, se aplicó la convolución con 64 filtros kernels, con un tamaño de 3x3 y se usó la función de activación ReLU. El funcionamiento de esta capa es similar a la anterior, lo que cambia aquí es la cantidad de matrices de salidas que se obtendrán ya que esta será de 64 dando así un total de 3 211 264 neuronas para esta segunda capa.
- Para la tercera capa se aplicó un Max Pooling con un tamaño de 2x2 esto permite reducir la altura y ancho dando una reducción en las dimensiones de la imagen, la cual se redimensionará a 112x112, lo cual reducirá la cantidad de neuronas en esta tercera capa a 802 816.
- Para la cuarta capa se aplicó la última convolución con 128 filtros con kernels con un tamaño de 2x2 y con la activación ReLU. Con esto se obtuvieron 128 matrices de salidas con un total de 1 605 632
- Para la quinta capa se aplicó otra capa Max pooling, con un tamaño de 2x2 y con esto reduciendo las dimensiones a un valor de 56x56 y así obteniendo un total de 401 408 neuronas para la sexta capa.
- Para la sexta capa se aplicó Flatten lo cual su función es “aplanar” las dimensiones, lo cual permite pasar de trabajar con resultados de tres dimensiones a una dimensión y conserva la misma cantidad de neuronas de la capa anterior.

- En la séptima capa se agregó Dense de 128 neuronas, lo cual su función es unir todas las neuronas de la capa anterior con esta capa. Esta capa lleva la activación ReLU como las anteriores.
- En la octava capa se agrega Dropout = 0.5, esto indica que desactiva el 50% de todas las neuronas recibidas con el fin de que la red convolucional no "memorice" y que realice un correcto aprendizaje.
- En la novena capa se agregó una capa "Dense" de 64 neuronas cuya finalidad fue conectar todas las neuronas de capas anteriores con esta misma, la cual tiene función de activación ReLU.
- Para la última capa se finalizó agregando una capa de salida el cual contiene la activación Softmax, esta asigna a cada una de las tres clases un valor probabilístico. La salida con un mayor valor será el resultado de la evaluación del modelo. Esta evaluación indicará si esta imagen se trata de "Positivo a Tuberculosis", "Positivo a COVID-19" o "Persona Sana"; esta función Softmax es la más ideal para este proyecto de tesis porque permite una clasificación para multiclases.

Luego de haber implementado la arquitectura del modelo particular de la red, esta se ejecutó usando la función model.compile(). Esta permite compilar la red por optimizadores,funciones, métricas, etc. Para este caso se usó la función de optimización "Loss" de tipo categorical ('Categorical_crossentropy'), el optimizador a usar es Adam y para las métricas de evaluación se usó "Accuracy"

3.4.2. Entrenamiento con CPU

Luego de realizar toda la implementación del modelo particular de la red neuronal convolucional, se pasó al entrenamiento de la misma, para esto se realizaron los siguientes pasos.

- Se definió los hiperparámetros ya mencionados en el apartado 3.3 donde se indica la cantidad de épocas = 120 y el batch_size = 64 y los pasos por cada época fue definido por la cantidad de imágenes de entrenamiento (2028) entre el batch_size (64) lo cual sería 31 pasos por cada época.

- Se ingresó los pasos de validación (validation_steps) para luego que finalice cada época utilice un conjunto de validación para poder observar si el aprendizaje de la red es correcto. Esta cantidad de pasos se obtiene dividiendo el val_sample (507) entre el bacth_size (64) dando un valor aproximado de 8.
- Finalmente se ejecutó el entrenamiento con los datos ya mencionados anteriormente junto con la función "fit" la cual es la encargada del entrenamiento de la red. Luego de finalizado el entrenamiento se declara para que se guarde el entrenamiento con el nombre de "cnnp120CPU.h5" y su peso con el nombre de "pesos120CPU.h5"

Luego de haber realizado el entrenamiento se observó que cada época tuvo una duración de 130 segundos aproximadamente, dando un total aproximado de 4 horas y 30 minutos. En la Figura Nº 16 se puede observar el tiempo total de la duración del entrenamiento de la red particular.

```
#Entrenamiento de la red neuronal

print("Inicio del entrenamiento")
cnn_p=cnn.fit(
    train_generator,
    epochs=epocas,
    steps_per_epoch=train_samples//batch_size,
    validation_data=val_generator,
    validation_steps=val_samples//batch_size)
cnn.save('E:/TESIS 2021/Tesis 3 clases/modelos/cnnp120CPU.h5')
cnn.save_weights('E:/TESIS 2021/Tesis 3 clases/modelos/pesos120CPU.h5')
```

```
Inicio del entrenamiento
Epoch 1/120
31/31 [==============================] - 130s 4s/step - loss: 0.9737 - accuracy: 0.5188 - val_loss: 0.7035 - val_accuracy: 0.7545
Epoch 2/120
31/31 [==============================] - 130s 4s/step - loss: 0.7362 - accuracy: 0.6767 - val_loss: 0.5280 - val_accuracy: 0.8237
Epoch 3/120
31/31 [==============================] - 131s 4s/step - loss: 0.6366 - accuracy: 0.7082 - val_loss: 0.4533 - val_accuracy: 0.8795
Epoch 4/120
31/31 [==============================] - 131s 4s/step - loss: 0.5496 - accuracy: 0.7637 - val_loss: 0.3776 - val_accuracy: 0.8884
Epoch 5/120
31/31 [==============================] - 130s 4s/step - loss: 0.4815 - accuracy: 0.7933 - val_loss: 0.3881 - val_accuracy: 0.8638
Epoch 6/120
31/31 [==============================] - 124s 4s/step - loss: 0.4452 - accuracy: 0.8131 - val_loss: 0.3455 - val_accuracy: 0.8884
Epoch 7/120
31/31 [==============================] - 123s 4s/step - loss: 0.3972 - accuracy: 0.8315 - val_loss: 0.3253 - val_accuracy: 0.8817
Epoch 115/120
31/31 [==============================] - 173s 6s/step - loss: 0.0978 - accuracy: 0.9633 - val_loss: 0.2565 - val_accuracy: 0.9420
Epoch 116/120
31/31 [==============================] - 196s 6s/step - loss: 0.1312 - accuracy: 0.9562 - val_loss: 0.1984 - val_accuracy: 0.9420
Epoch 117/120
31/31 [==============================] - 173s 6s/step - loss: 0.1276 - accuracy: 0.9542 - val_loss: 0.2613 - val_accuracy: 0.9375
Epoch 118/120
31/31 [==============================] - 181s 6s/step - loss: 0.0980 - accuracy: 0.9638 - val_loss: 0.2775 - val_accuracy: 0.9286
Epoch 119/120
31/31 [==============================] - 178s 6s/step - loss: 0.1141 - accuracy: 0.9572 - val_loss: 0.1856 - val_accuracy: 0.9375
Epoch 120/120
31/31 [==============================] - 166s 5s/step - loss: 0.1120 - accuracy: 0.9572 - val_loss: 0.2795 - val_accuracy: 0.9241
```

Figura Nº 16: Entrenamiento de la Red neuronal particular.

Fuente: Captura de pantalla de la plataforma JupyterLab.

Luego de haber obtenido estos resultados se graficó los datos obtenidos por el “Loss” y “Accurracy” de la data de entrenamiento. Esto se observá en la Figura Nº 17.

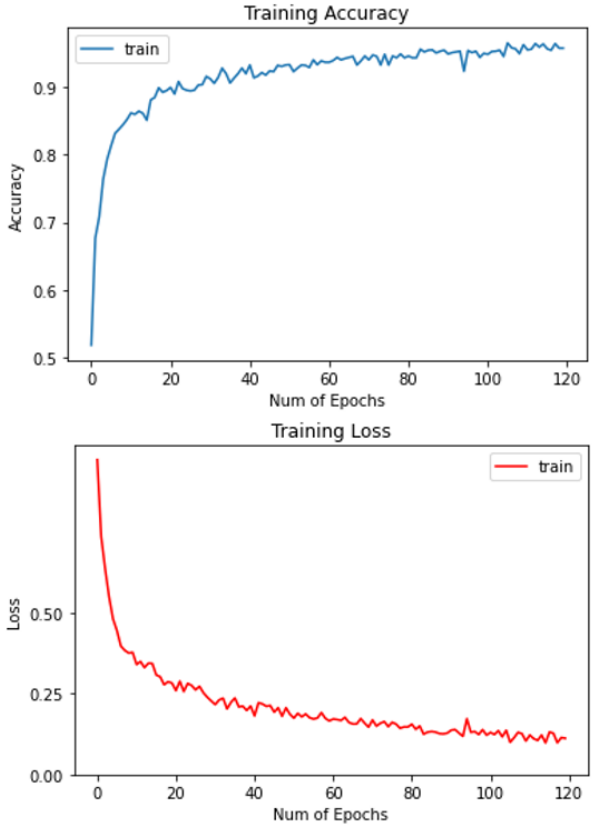


Figura Nº 17: Gráfica de Accuracy y Loss del entrenamiento de la red particular

Fuente: Captura de pantalla de la plataforma JupyterLab.

3.4.3. Entrenamiento con GPU

Para el entrenamiento con la GPU, se procedió a instalar la librería tensorflow-gpu para poder usar la GPU como medio de entrenamiento y se validó con los comandos mostrados en la Figura Nº 18.

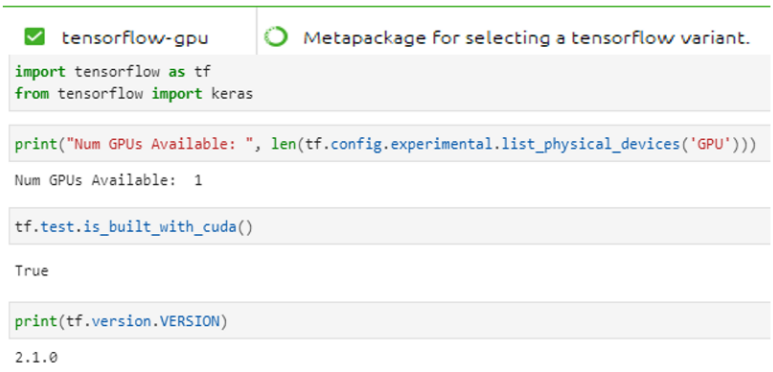


Figura Nº 18: Validación de GPU en Python

Fuente: Captura de pantalla de la plataforma JupyterLab.

Luego de haber realizado estos pasos se procedió a definir los mismos parámetros de entrenamiento de la red con CPU, pero para este caso se procedió a guardar el modelo y pesos con los respectivos nombres de "cnnp120.h5" y "pesos120.h5". Luego de haberse realizado el entrenamiento como se logra apreciar en la Figura Nº 19 se observa que para este caso con GPU cada época tuvo una duración de 24 segundos, dando así una duración total de entrenamiento de 1 hora con 20 minutos.

Figura Nº 19: Entrenamiento de la red neuronal Particular usando GPU

Fuente: Captura de pantalla de la plataforma JupyterLab.

Luego de haber obtenido estos resultados se graficó los datos obtenidos por el "Loss" y "Accurracy" de la data de entrenamiento. Esto se observa en la Figura Nº 20.

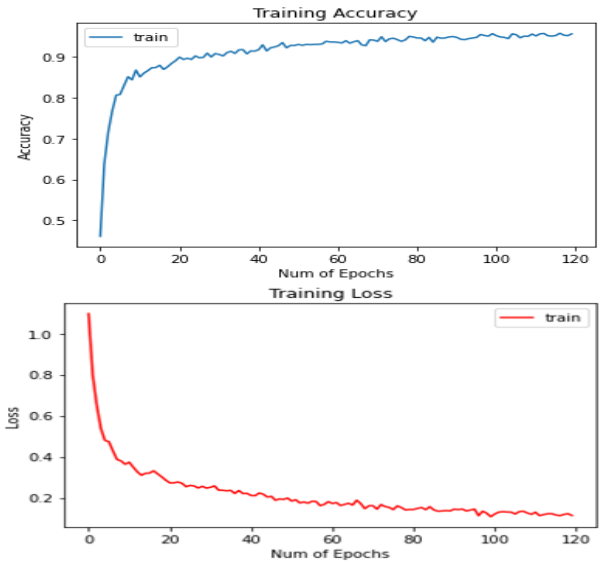


Figura Nº 20: Gráfica de Accuracy y Loss del entrenamiento de la red particular

Fuente: Captura de pantalla de la plataforma JupyterLab.

3.5. Implementación y Entrenamiento del modelo InceptionV3

Para este subcapítulo se desarrolla la implementación y entrenamiento del modelo de red InceptionV3 utilizando Python, Keras y Tensorflow.

3.5.1. Implementación

Según Gómez-Ríos et al (2019), este modelo de red tiene 48 capas en su estructura, de las cuales todas ya se encuentran pre entrenadas. Este modelo está diseñado para admitir hasta 1000 clases y la máxima dimensión de entrada que permite es de 299 x 299 con una profundidad de 3. Según la web de Keras (Keras, 2021), esta cuenta con 23 851 784 parámetros entrenados de los cuales de estos hay 34 432 que no han sido entrenados. Para su implementación se necesitó definir los parámetros de entrada de las imágenes para que estas tengan una dimensión de 299x299x3 lo máximo soportado por este modelo de red. Al ser este modelo una red pre entrenada con una base de datos de ImageNet se procedió a eliminar la capa de predicción de 1000 salidas con el siguiente comando "include_top=false", esto es necesario ya que se ingresarán las imágenes de la base de datos seleccionadas previamente. En la Figura N

°21 se muestra el llamado del modelo de red InceptionV3 quitando la última capa mencionada anteriormente.

```
#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()
```

```
Model: "inception_v3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 299, 299, 3) 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 149, 149, 32) 864         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 149, 149, 32) 96          conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 149, 149, 32) 0           batch_normalization[0][0]
```

Figura Nº 21: Implementación del modelo InceptionV3

Fuente: Captura de pantalla del editor Jupyter Lab.

Este modelo cuenta con 48 capas, por esto es considerado una red neuronal muy profunda. Para este caso se decide aplicar la técnica de aprendizaje Transfer Learning y con esto poder aprovechar el pre entrenamiento que posee gracias a la base de datos de ImageNet, la cual se realiza cuando se declaran los argumentos de implementación del modelo de esta red.

Para el trabajo de esta tesis se aplicó Transfer Learning por lo cual se añadió al último unas 6 capas más, quitando la capa de predicción de 1000 clases. Para esto se tomó en cuenta lo hecho por Narin et al. (2020) el cual adicionó 3 capas mas a la arquitectura, las cuales fueron una capa Pooling, una capa “Dense” de 1024 neuronas y por último una capa “Softmax”, lo cual no dio resultados tan eficientes, por lo cual para este trabajo de investigación se decidió añadir 6 capas adicionales ya que inicialmente añadiendo las 3 capas que añadió Narin no se tenía resultados óptimos en el entrenamiento. Las capas añadidas para este caso fueron 3 capas “Dense” una capa “Flatten”, una capa de “Dropout” y por último una capa de predicción de 3 salidas. A continuación, se detallan cada una de las 6

capas añadidas luego de aplicar la técnica de aprendizaje Transfer Learning. Todo esto se observa en la Figura N °22

- Primero se agregó una capa “Flatten” para poder llevar las dimensiones de la data recibida a una sola dimensión con las mismas neuronas que su capa anterior.
- Para la segunda, se agregó una capa “Dense” que contiene 128 neuronas la cual cuenta con activación ReLU para poder transformar los negativos en ceros, y al igual que el caso anterior conserva las mismas neuronas que la capa anterior.
- Se agregó una capa Dense que contiene 64 neuronas la cual cuenta con activación ReLU para poder transformar los negativos en ceros, y al igual que el caso anterior conserva las mismas neuronas que la capa anterior.
- Se agregó una capa “Dense” que contiene 32 neuronas la cual cuenta con activación ReLU para poder transformar los negativos en ceros, y al igual que el caso anterior conserva las mismas neuronas que la capa anterior.
- Se agregó una capa “Dropout” de 0.5 lo cual hace que el 50% de las neuronas se inhabiliten con el objetivo de que la red no memorice si no que aprenda.
- Para la última capa se finalizó agregando una capa de salida el cual contiene la activación “Softmax”, esta asigna a cada una de las tres clases un valor probabilístico. Esta evaluación indicará si esta imagen se trata de “Positivo a Tuberculosis”, “Positivo a COVID-19” o “Persona Sana”; esta función “Softmax” es la más ideal para este proyecto de tesis porque permite una clasificación para multiclases.

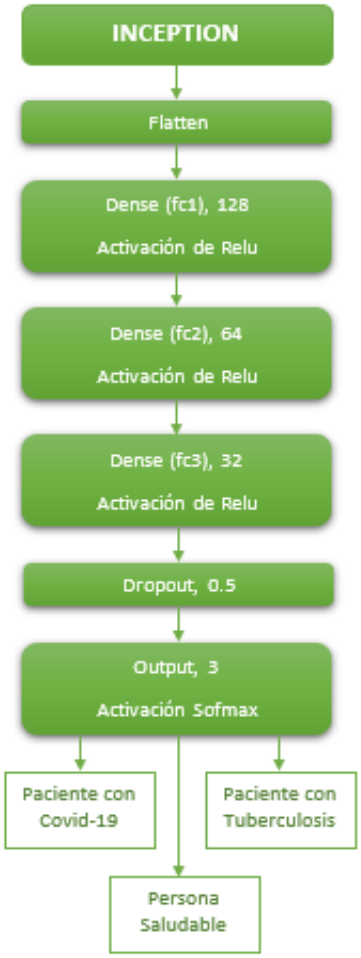


Figura Nº 22: Arquitectura del modelo de Red INCEPTION V3.

Fuente: Elaboración propia.

Luego de declarar las variables y asignar los valores correspondientes se compiló el modelo de red INEPTIONV3 el cual se observa en la Figura N° 23. Para este caso se está usando los mismos parámetros mencionados en el subcapítulo 3.4 en la cual se detalla el tipo de optimizador y las métricas a usar para el entrenamiento del modelo de red neuronal que para este caso es el InceptionV3.

```
•[22]: #Transfer Learning del modelo 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(3, activation='softmax', name='predictions')(ic)
InceptionB = Model(image_input, predictions)

# freeze all the layers except the dense layers
for layer in pre_trained_inception.layers[:-6]:
    layer.trainable = False

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

```
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_1 (Dropout)          (None, 32)           0           fc3[0][0]
__________________________________________________________________________________________________
predictions (Dense)          (None, 3)            99          dropout_1[0][0]
==================================================================================================
Total params: 38,590,563
Trainable params: 16,787,971
```

Figura Nº 23: Aplicación de Transfer Learning al modelo de Red InceptionV3

Fuente: Captura de pantalla del editor Jupyter Lab.

3.5.2. Entrenamiento con CPU

Para el proceso de entrenamiento de IncenptionV3 se usó los mismos parámetros mencionados para el entrenamiento de la red particular en el subcapítulo 3.4.2, se usó la mismas “épocas”, “steps_per_epoch”, “validation_steps” y “batch_size”. Para este caso el modelo fue guardado con el nombre de “inceptionv3.h5” esto se muestra en la Figura N° 24. Aquí se puede observar que cada época tuvo una duración entre 133 a 140 segundos, lo cual hace que tenga un tiempo de demora aproximado de 5 horas.

```
#Entrenamiento de la red neuronal

print("Inicio del entrenamiento")
Inception_N2=InceptionB.fit(
    train_generator,
    epochs=epocas,
    steps_per_epoch=train_samples//batch_size,
    validation_data=val_generator,
    validation_steps=val_samples//batch_size)
InceptionB.save('E:/TESIS 2021/Tesis 3 clases/modelos/CPUinceptionv3.h5')
```

```
Inicio del entrenamiento
Epoch 1/120
31/31 [==============================] - 133s 4s/step - loss: 3.2121 - accuracy: 0.6955 - val_loss: 0.3338 - val_accuracy: 0.8661
Epoch 2/120
31/31 [==============================] - 127s 4s/step - loss: 0.4853 - accuracy: 0.7953 - val_loss: 0.3160 - val_accuracy: 0.9330
Epoch 3/120
31/31 [==============================] - 136s 4s/step - loss: 0.3920 - accuracy: 0.8131 - val_loss: 0.2604 - val_accuracy: 0.9330
Epoch 118/120
31/31 [==============================] - 140s 5s/step - loss: 0.0442 - accuracy: 0.9832 - val_loss: 0.7781 - val_accuracy: 0.9442
Epoch 119/120
31/31 [==============================] - 138s 4s/step - loss: 0.0367 - accuracy: 0.9863 - val_loss: 0.4050 - val_accuracy: 0.9464
Epoch 120/120
31/31 [==============================] - 139s 4s/step - loss: 0.0418 - accuracy: 0.9869 - val_loss: 0.2626 - val_accuracy: 0.9375
```

Figura Nº 24: Entrenamiento del modelo InceptionV3

Fuente: Captura de pantalla del editor Jupyter Lab.

Luego de haber obtenido estos resultados se graficó los datos obtenidos por el Loss y Accurracy de la data de entrenamiento. Esto lo observamos en la Figura Nº 25.

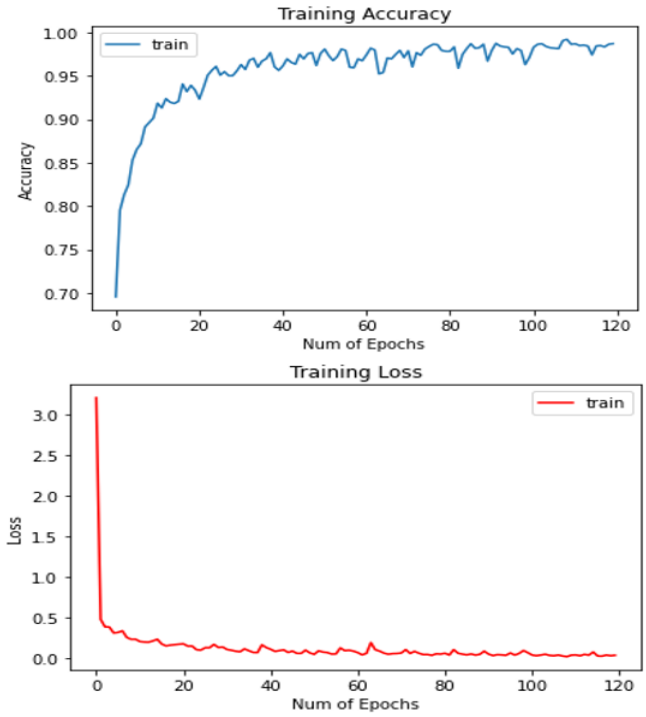


Figura Nº 25: Gráfica de Accuracy y Loss del entrenamiento del modelo InceptionV3 usando CPU

Fuente: Captura de pantalla del editor Jupyter Lab.

3.5.3. Entrenamiento con GPU

Para el entrenamiento con la GPU, se procedió a instalar la librería tensorflow-gpu la cual permitió usar la GPU como medio de entrenamiento y se validó con los comandos ya explicados para el caso de red particular y visualizado en la figura N°21.

Luego de haber realizado estos pasos se procedió a definir los mismos parámetros de entrenamiento de la red con CPU, pero para este caso se procedió a guardar el modelo con el nombre de “inceptionv3.h5”. Luego de haberse realizado el entrenamiento como se logra apreciar en la Figura Nº 26 se observa que para este caso con GPU cada época tuvo una duración de 36 segundos, dando así una duración total de entrenamiento de 1 hora con 12 minutos.

```
#Entrenamiento de la red neuronal

print("Inicio del entrenamiento")
Inception_N2=InceptionB.fit(
    train_generator,
    epochs=epocas,
    steps_per_epoch=train_samples//batch_size,
    validation_data=val_generator,
    validation_steps=val_samples//batch_size)
InceptionB.save('E:/TESIS 2021/Tesis 3 clases/modelos/inceptionv3.h5')
```

```
Inicio del entrenamiento
Epoch 2/120
31/31 [==============================] - 36s 1s/step - loss: 0.6284 - accuracy: 0.7495 - val_loss: 0.7668 - val_accuracy: 0.7143
Epoch 3/120
31/31 [==============================] - 36s 1s/step - loss: 0.4267 - accuracy: 0.8050 - val_loss: 1.0157 - val_accuracy: 0.6897
Epoch 4/120
31/31 [==============================] - 36s 1s/step - loss: 0.3520 - accuracy: 0.8391 - val_loss: 0.6177 - val_accuracy: 0.7812
Epoch 118/120
31/31 [==============================] - 36s 1s/step - loss: 0.0429 - accuracy: 0.9888 - val_loss: 2.1362 - val_accuracy: 0.7098
Epoch 119/120
31/31 [==============================] - 36s 1s/step - loss: 0.0668 - accuracy: 0.9786 - val_loss: 0.6625 - val_accuracy: 0.8326
Epoch 120/120
31/31 [==============================] - 36s 1s/step - loss: 0.0673 - accuracy: 0.9807 - val_loss: 1.1523 - val_accuracy: 0.7098
```

Figura Nº 26: Entrenamiento del modelo de red IncentpionV3

Fuente: Captura de pantalla del editor Jupyter Lab.

Luego de haber obtenido estos resultados se graficó los datos obtenidos por el "Loss" y "Accuracy" de la data de entrenamiento. Esto lo observamos en la Figura Nº 27.

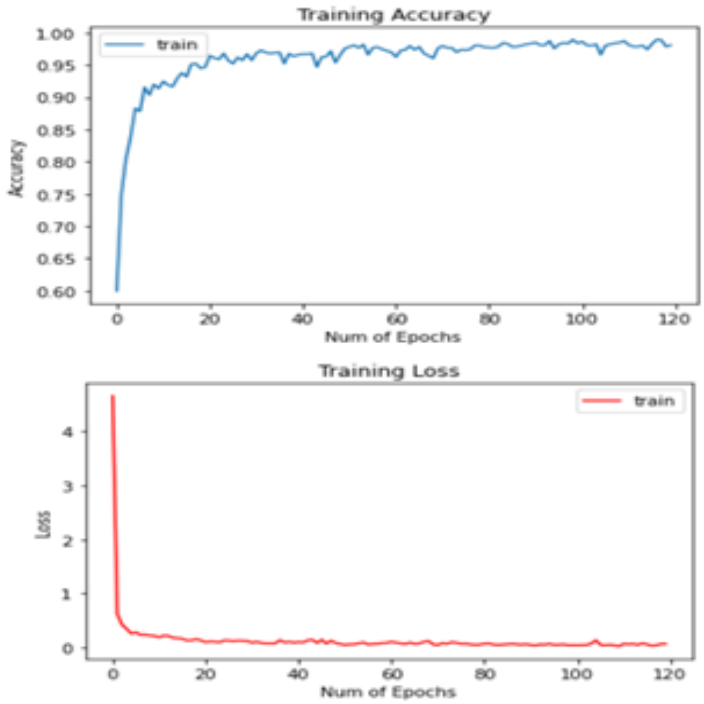


Figura Nº 27: Gráfica de Accuracy y Loss del entrenamiento del modelo InceptionV3 usando GPU

Fuente: Captura de pantalla del editor Jupyter Lab.

3.6. Implementación y Entrenamiento del modelo VGG16

Para esta sección se desarrolla la implementación y entrenamiento del modelo de red VGG16 utilizando las librerías de Tensorflow, Keras y el lenguaje de programación Python dentro de la interfaz Jupyter Lab.

3.6.1. Implementación

Según Yusuf, K. (2020), este modelo de red tiene 16 capas en su estructura, de las cuales todas ya se encuentran pre entrenadas. Este modelo está diseñado para admitir hasta 1000 clases y la máxima dimensión de entrada que permite es de 224x224 con una profundidad de 3. Según la web de Keras (Keras, 2021), esta cuenta con 138 357 544 parámetros entrenados. Para su implementación se usó los parámetros de entrada de las imágenes para que estas tengan una dimensión de 224x224x3 lo máximo soportado por este modelo de red. Al ser este modelo una red pre entrenada con una base de datos de ImageNet se procedió a eliminar la capa de predicción de 1000 salidas con el siguiente comando "include_top=false", esto es necesario ya que se ingresarán las imágenes de la base de datos seleccionadas previamente. En la Figura N ° 28 se muestra el llamado del modelo de red VGG16 quitando la última capa mencionada anteriormente.

```
#modelo base de VGG16

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

pre_trained_VGG = VGG16(input_tensor=image_input,weights='imagenet',
                        include_top=False)

pre_trained_VGG.summary()
```

```
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
```

Figura Nº 28: Implementación del modelo VGG16

Fuente: Captura de pantalla del editor Jupyter Lab.

Luego de llamar al modelo de red VGG16, se procedió a aplicar Transfer Learning el cual consistió en adicionar 6 capas más y quitar la capa de

predicción de 1000 clases. Para esto se tomó en cuenta lo hecho por Narin et al. (2020) el cual adicionó 3 capas más a la arquitectura, las cuales fueron una capa “Pooling”, una capa “Dense” de 1024 neuronas y por último una capa “Softmax”, lo cual no les dio resultados tan eficientes, por lo cual para este trabajo de investigación se decidió añadir 6 capas adicionales ya que inicialmente añadiendo las 3 capas que añadió Narin no se tenía resultados óptimos en el entrenamiento. Las capas añadidas para este caso fueron 3 capas “Dense”, una capa “Flatten”, una capa de “Dropout” y por último una capa de predicción de 3 salidas. A continuación, se detallan cada una de las 6 capas añadidas luego de aplicar la técnica de aprendizaje Transfer Learning.

- Primero se agregó una capa “Flatten” para poder llevar las dimensiones de la data recibida a una sola dimensión con las mismas neuronas que su capa anterior.
- Para la segunda, se agregó una capa “Dense” que contiene 128 neuronas la cual cuenta con activación ReLU para poder transformar los negativos en ceros, y al igual que el caso anterior conserva las mismas neuronas que la capa anterior.
- Se agregó una capa “Dense” que contiene 64 neuronas la cual cuenta con activación ReLU para poder transformar los negativos en ceros, y al igual que el caso anterior conserva las mismas neuronas que la capa anterior.
- Se agregó una capa “Dense” que contiene 32 neuronas la cual cuenta con activación ReLU para poder transformar los negativos en ceros, y al igual que el caso anterior conserva las mismas neuronas que la capa anterior.
- Se agregó una capa “Dropout” de 0.5 lo cual hace que el 50% de las neuronas se inhabiliten con el objetivo de que la red no memorice si no que aprenda.
- Para la última capa se finalizó agregando una capa de salida el cual contiene la activación Softmax, esta asigna a cada una de las tres clases un valor probabilístico. Esta evaluación indicará si esta imagen

se trata de “Positivo a Tuberculosis”, “Positivo a COVID-19” o “Persona Sana”; esta función “Softmax” es la más ideal para este proyecto de tesis porque permite una clasificación para multiclases.

En la Figura N° 29 se observa la arquitectura del modelo de red VGG16.

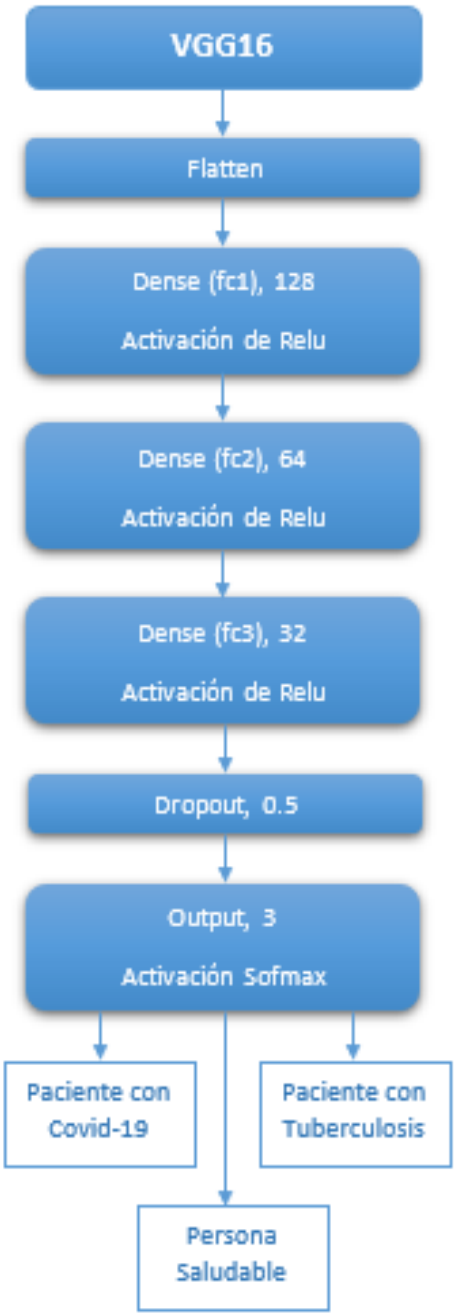


Figura Nº 29: Arquitectura del modelo de Red VGG16.

Fuente: Elaboración propia.

Luego de declarar las variables y asignar los valores correspondientes se compilo el modelo de red VGG16 el cual se puede observar en la Figura N° 30. Para este caso se está usando los mismos parámetros mencionados en el subcapítulo 3.4 en la cual se detalla el tipo de optimizador y las métricas a usar para el entrenamiento del modelo de red neuronal que para este caso es la VGG16.

```
#Implementación del modelo VGG16 utilizando Transfer Learning
#Modelo modificado de VGG16

last_layer = pre_trained_VGG.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(3, activation='softmax', name='predictions')(ic)
VGGB = Model(image_input, predictions)


# freeze all the layers except the dense layers

for layer in pre_VGG.layers[:-6]:
    layer.trainable = False

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

```
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 128)               3211392
_________________________________________________________________
fc2 (Dense)                  (None, 64)                8256
_________________________________________________________________
fc3 (Dense)                  (None, 32)                2080
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
predictions (Dense)          (None, 3)                 99
=================================================================
```

Figura Nº 30: Aplicación de Transfer Learning al modelo de Red VGG16.

Fuente: Captura de pantalla del editor Jupyter Lab.

3.6.2. Entrenamiento con CPU

Para el proceso de entrenamiento de la VGG16 se utilizó los mismos parámetros mencionado para el entrenamiento de la red particular en el subcapítulo 3.4.2, se usó la mismas “épocas”, “steps_per_epoch”, “validation_steps” y “batch_size”. Para este caso el modelo fue guardado con el nombre de “CPUVGG16.h5” esto se muestra en la Figura N° 31. Aquí se puede observar que cada época tuvo una duración entre 285 a 434 segundos teniendo un promedio de 359.5 segundos por época, lo cual hace que tenga un tiempo de demora aproximado de 12 horas.

```
#Entrenamiento de la red neuronal

print("Inicio del entrenamiento")
VGG_N2=VGGB.fit(
    train_generator,
    epochs=epocas,
    steps_per_epoch=train_samples//batch_size,
    validation_data=val_generator,
    validation_steps=val_samples//batch_size)
VGGB.save('E:/TESIS 2021/Tesis 3 clases/modelos/CPUVGG16.h5')
```

```
Inicio del entrenamiento
Epoch 1/120
31/31 [==============================] - 285s 9s/step - loss: 1.1264 - accuracy: 0.3391 - val_loss: 1.0986 - val_accuracy: 0.3393
Epoch 2/120
31/31 [==============================] - 276s 9s/step - loss: 1.1329 - accuracy: 0.3315 - val_loss: 1.0991 - val_accuracy: 0.3237
Epoch 3/120
31/31 [==============================] - 287s 9s/step - loss: 1.1010 - accuracy: 0.3386 - val_loss: 1.0985 - val_accuracy: 0.3281
Epoch 118/120
31/31 [==============================] - 421s 14s/step - loss: 0.1453 - accuracy: 0.9623 - val_loss: 0.3452 - val_accuracy: 0.9308
Epoch 119/120
31/31 [==============================] - 434s 14s/step - loss: 0.1175 - accuracy: 0.9644 - val_loss: 0.7113 - val_accuracy: 0.9219
Epoch 120/120
31/31 [==============================] - 412s 13s/step - loss: 0.0511 - accuracy: 0.9883 - val_loss: 0.2344 - val_accuracy: 0.9621
```

Figura Nº 31: Entrenamiento del modelo de Red VGG16 usando CPU

Fuente: Captura de pantalla del editor Jupyter Lab.

Luego de haber obtenido estos resultados se graficó los datos obtenidos por el "Loss" y "Accuracy" de la data de entrenamiento. Esto se observa en la Figura Nº 32.

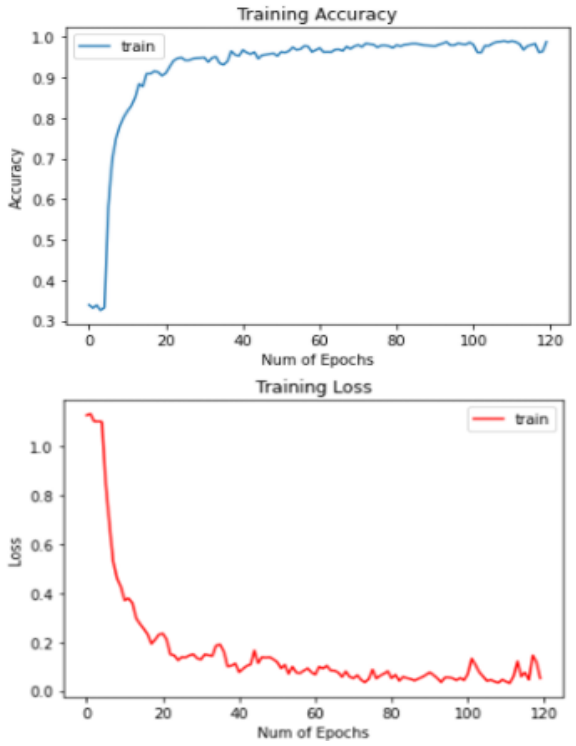


Figura Nº 32: Gráfica de Accuracy y Loss del entrenamiento del modelo VGG16 usando CPU

Fuente: Captura de pantalla del editor Jupyter Lab.

3.6.3. Entrenamiento con GPU

Para el entrenamiento con la GPU, se procedió a instalar la librería tensorflow-gpu la cual permite usar la GPU como medio de entrenamiento y se valida con los comandos ya explicados para el caso de red particular y visualizado en la Figura N° 33.

Luego de haber realizado estos pasos se procedió a definir los mismos parámetros de entrenamiento de la red con CPU, pero para este caso se procedió a guardar el modelo con el nombre de "VGG16.h5". Luego de haberse realizado el entrenamiento como se logra apreciar en la figura N.º 33 se observa que para este caso con GPU cada época tuvo una duración de 23 segundos, dando así una duración total de entrenamiento de 46 minutos.

```
#Entrenamiento de la red neuronal

print("Inicio del entrenamiento")
VGG_N2=VGG8.fit(
    train_generator,
    epochs=epocas,
    steps_per_epoch=train_samples//batch_size,
    validation_data=val_generator,
    validation_steps=val_samples//batch_size)
VGG8.save('E:/TESIS 2021/Tesis 3 clases/modelos/VGG16.h5')
```

```
31/31 [==============================] - 23s 757ms/step - loss: 1.1052 - accuracy: 0.3371 - val_loss: 1.0973 - val_accuracy: 0.3438
Epoch 3/120
31/31 [==============================] - 24s 767ms/step - loss: 1.0986 - accuracy: 0.3376 - val_loss: 1.0965 - val_accuracy: 0.5045
Epoch 4/120
31/31 [==============================] - 23s 750ms/step - loss: 1.0296 - accuracy: 0.5229 - val_loss: 0.6908 - val_accuracy: 0.6451
Epoch 5/120
31/31 [==============================] - 23s 747ms/step - loss: 0.6903 - accuracy: 0.6721 - val_loss: 0.5856 - val_accuracy: 0.7924
Epoch 118/120
31/31 [==============================] - 23s 745ms/step - loss: 0.0261 - accuracy: 0.9903 - val_loss: 0.1410 - val_accuracy: 0.9621
Epoch 119/120
31/31 [==============================] - 23s 745ms/step - loss: 0.0258 - accuracy: 0.9898 - val_loss: 0.8541 - val_accuracy: 0.9263
Epoch 120/120
31/31 [==============================] - 23s 745ms/step - loss: 0.0678 - accuracy: 0.9786 - val_loss: 0.6269 - val_accuracy: 0.9464
```

Figura Nº 33: Entrenamiento del modelo de Red VGG16 usando GPU

Fuente: Captura de pantalla del editor Jupyter Lab.

Luego de obtener estos resultados se graficó los datos obtenidos por el Loss y Accuracy de la data de entrenamiento. Esto se observa en la Figura Nº 34.

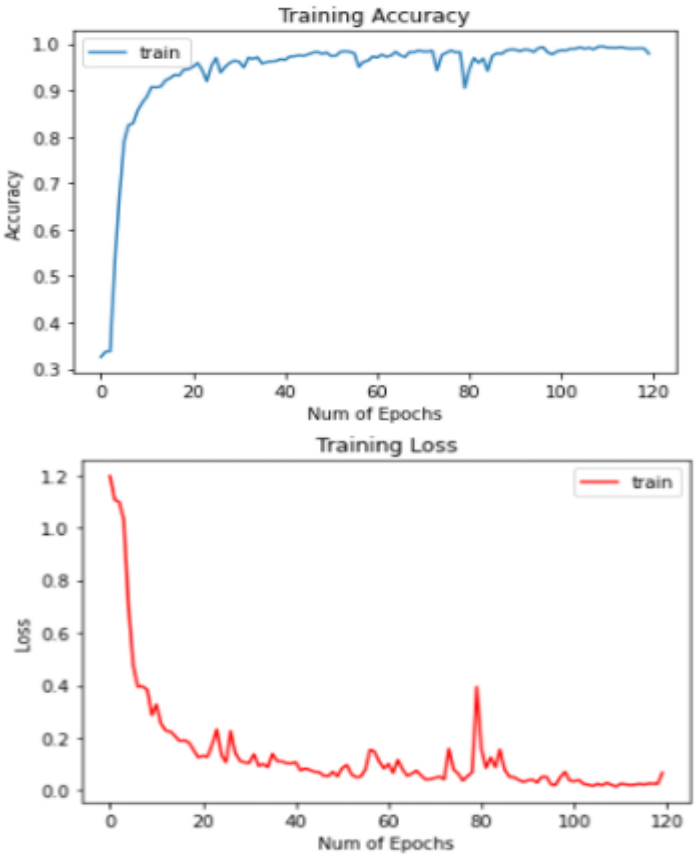


Figura Nº 34: Gráfica de Accuracy y Loss del entrenamiento del modelo VGG16 usando GPU

Fuente: Captura de pantalla del editor Jupyter Lab.

CAPÍTULO IV: PRUEBAS Y RESULTADOS

En este capítulo se detallarán las pruebas de validación y resultados obtenidos de la Red Neuronal convolucional Particular, así como los dos modelos INCEPTIONV3 y VGG16 desarrollados en el capítulo 3. Para las pruebas de validación se trabajó con la base de datos ubicada en la carpeta "Testeo". Las pruebas de validación ayudaron a determinar el correcto funcionamiento de la red, obteniendo como resultados el poder diferenciar en la radiografía si el paciente tiene Tuberculosis, Covid-19 o es una Persona Sana. A continuación, se presentan los resultados obtenidos de los entrenamientos; primero se detalla el proceso de validación de las redes neuronales, en las cuales se muestra mediante gráficas que usan como parámetros la función de optimización "Loss" y la métrica "accuracy". Para el segundo, se detalla las pruebas de validación realizada tomando una imagen aleatoria de la carpeta "Testeo". Para el tercer caso se muestra la matriz de confusión donde se puede observar los falsos negativos, falsos positivos, verdadero positivos y verdaderos negativos. Y por último se muestra la tabla de resultados con las métricas establecidas inicialmente en el capítulo 2 de la presente investigación.

4.1. Pruebas y resultados de la red neuronal Particular

4.1.1. Proceso de Validación

El proceso de validación del modelo de red particular fue establecido con 120 "épocas", para medir la validación de este modelo, se consideró el "Accuracy" y el "Loss", estos parámetros se consideraron en el entrenamiento del modelo de la red particular con CPU y GPU.

A continuación, se presentan las Figuras N° 35 y N° 36, en donde se observa, que el modelo de red particular usando la CPU y GPU respectivamente tiene pequeños sobreajustes a lo largo del proceso de "Validation Loss".

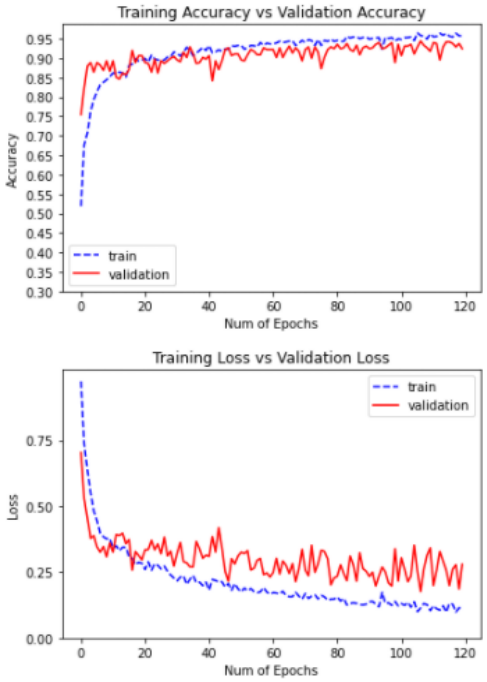


Figura Nº 35: Gráfica de Validación de Accuracy y Loss de la red particular usando CPU.

Fuente: Captura de pantalla del editor Jupyter Lab.

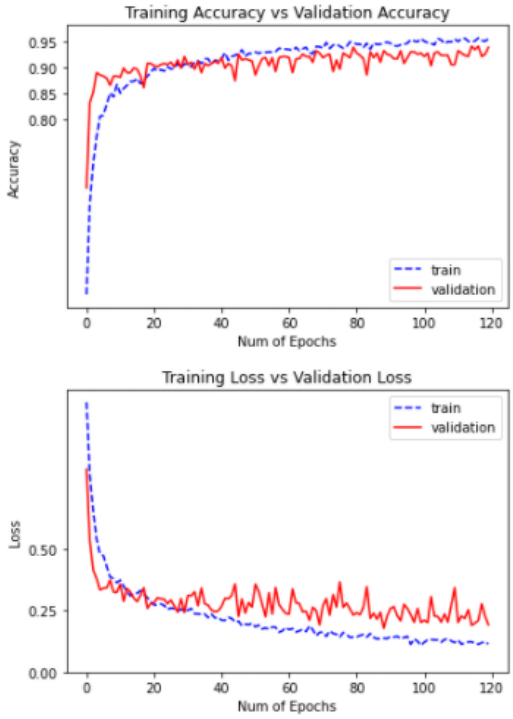


Figura Nº 36: Gráfica de Validación de Accuracy y Loss de la red particular usando GPU.

Fuente: Captura de pantalla del editor Jupyter Lab.

4.1.2. Pruebas de Validación

Las pruebas de validación se realizaron para verificar el adecuado funcionamiento del modelo de red Particular, mediante el algoritmo presentado en el Anexo N°2 y N°3. Para ello, el algoritmo utilizó las imágenes de la carpeta "Testeo". En la Figura N°37 se muestran las pruebas de validación realizada a partir de imágenes de radiografía de tórax frontal para cada clase, además se muestra la comparación del modelo de red Particular usando la CPU y la GPU para las pruebas de validación.

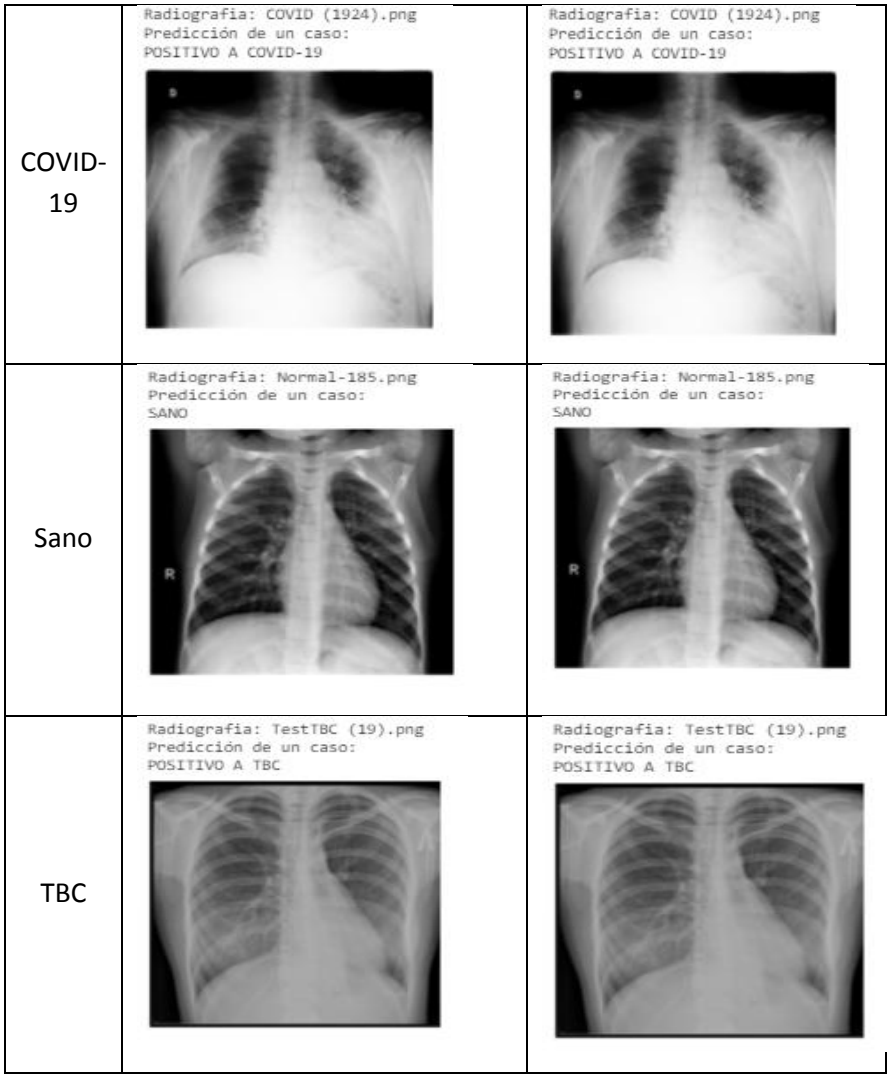


Figura Nº 37: Pruebas de Validación del modelo particular usando CPU vs GPU respectivamente

Fuente: Elaboración propia (2021)

4.1.3. Matriz de Confusión y Resultados.

En esta sección se presenta la matriz de confusión, Matriz de observación y Métricas de evaluación del modelo de Red Particular, las cuales fueron calculadas a partir de las ecuaciones explicadas en la sección 2.3.9.

En la Figura N°38 se observa la matriz de confusión obtenida al finalizar el proceso de validación de las 3 clases haciendo uso de la CPU y GPU. En esta matriz obtuvimos los verdaderos positivos y brindó datos para el cálculo de los verdaderos negativos, falso positivos y negativos, para cada una de las tres clases.

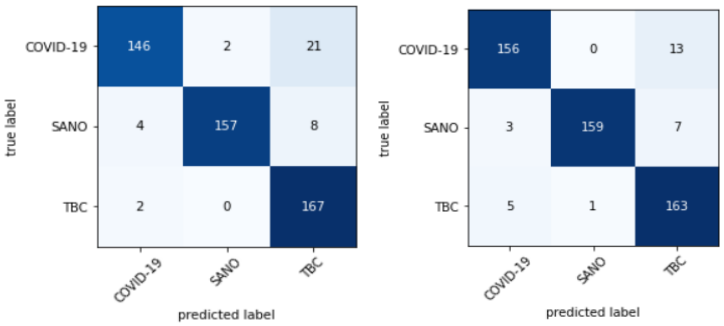


Figura Nº 38: Matriz de Confusión de la red particular usando CPU y GPU respectivamente

Fuente: Elaboración propia (2021)

Las tablas N°1 y N°2, son las matrices de observación del modelo de la Red Particular usando la CPU y GPU respectivamente. Dichos valores fueron determinados a través de la matriz de confusión.

Tabla N° 1: Matriz de Observación de la Red Particular usando CPU.

	TP	TN	FP	FN
Covid-19	146	332	6	23
Sano	157	336	2	12
Tuberculosis	167	309	29	2

Fuente: Elaboración propia (2021)

Tabla N° 2: Matriz de Observación de la Red Particular usando GPU

	TP	TN	FP	FN
Covid-19	156	330	8	13
Sano	159	337	1	10
Tuberculosis	163	318	20	6

Fuente: Elaboración propia (2021)

Los valores mostrados en las tablas N°3 y N°4 se obtuvieron a partir de los datos de la Matriz de observación, los cuales permiten calcular el "Accuracy", "Precision", "Recall", "F1 Score" y "General Precision". Dichos parámetros son explicados en el capítulo 2.3.9, el cual ayudó conocer la precisión del modelo de Red Particular para cada clase después del entrenamiento usando CPU y GPU.

Tabla N° 3: Métricas de medición de la red Particular usando CPU

Indicadores	Accuracy	Precision	Recall	Specifity	F1-Score	General Precision	General Precision (%)
Covid-19	0.9428	0.9605	0.8639	0.9822	0.9097	0.9270	92.70%
Sano	0.9724	0.9874	0.9290	0.9941	0.9573		
Tuberculosis	0.9389	0.8520	0.9882	0.9142	0.9151		

Fuente: Elaboración propia (2021)

Tabla N° 4: Métricas de medición de la red Particular usando GPU.

Indicadores	Accuracy	Precision	Recall	Specifity	F1-Score	General Precision	General Precision (%)
Covid-19	0.9586	0.9512	0.9231	0.9763	0.9369	0.9428	94.28%
Sano	0.9783	0.9938	0.9408	0.9970	0.9666		
Tuberculosis	0.9487	0.8907	0.9645	0.9408	0.9261		

Fuente: Elaboración propia (2021)

De las Tablas N°3 y N°4 se observa que el desempeño de la Red Particular usando la CPU tiene una Precisión General de 92.70% y con la GPU 94.28%. Estos resultados indican que el entrenamiento de la red fue óptimo cuando se realizó con la GPU.

4.2. Pruebas y resultados del modelo INCEPTION V3

4.2.1. Proceso de Validación

El modelo INCEPTION V3 fue establecido en 120 "épocas". Los parámetros "Accuracy" y "Loss", fueron considerados para medir la validación del modelo de red. A continuación, se muestran las Figuras N° 39 y N° 40, en la cual la Figura N° 39 muestra un mayor sobreajuste entre 50-70 épocas, usando la CPU.

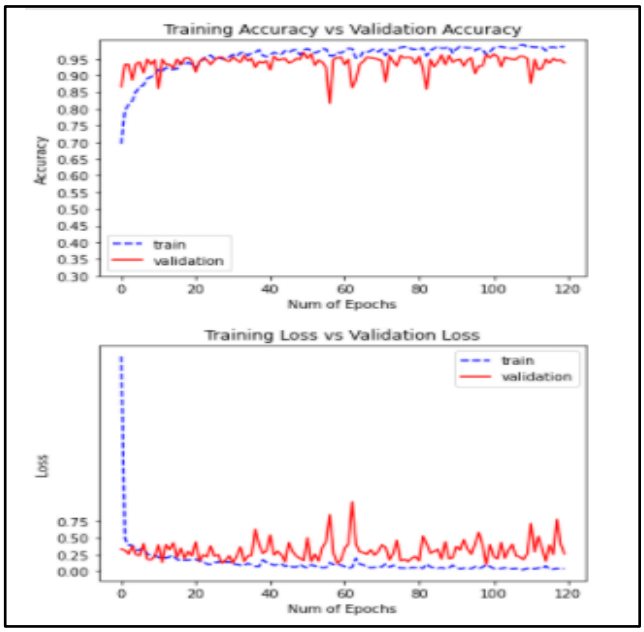


Figura Nº 39: Gráfica de Validación de Accuracy y Loss del modelo InceptionV3 usando CPU.

Fuente: Captura de pantalla del editor Jupyter Lab.

Para el caso de la Figura N° 40 se muestra la validación del entrenamiento del modelo de la red usando GPU, el cual indica que la validación del modelo tuvo demasiados sobreajustes a lo largo del entrenamiento.

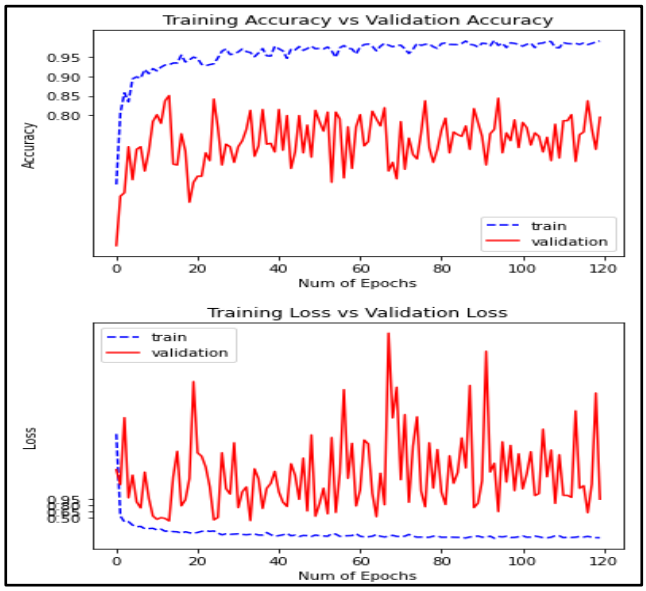


Figura Nº 40: Gráfica de Validación de Accuracy y Loss del modelo InceptionV3 usando GPU.

Fuente: Captura de pantalla del editor Jupyter Lab.

4.2.2. Pruebas de Validación

Se ejecutaron pruebas de validación para verificar el adecuado funcionamiento del modelo INCEPTION V3, mediante el algoritmo presentado en los anexos N°4 y °5. Este algoritmo hizo uso de las imágenes correspondientes a la carpeta “Testeo”, para validar el modelo de red. En la Figura N°41 se muestran las pruebas de validación realizadas a partir de imágenes de radiografías de tórax la cual corresponde a cada uno de los casos vistos (Covid-19, Tuberculosis y Sano), además se muestra la comparación de validación obtenida mediante el entrenamiento con la CPU y la GPU.

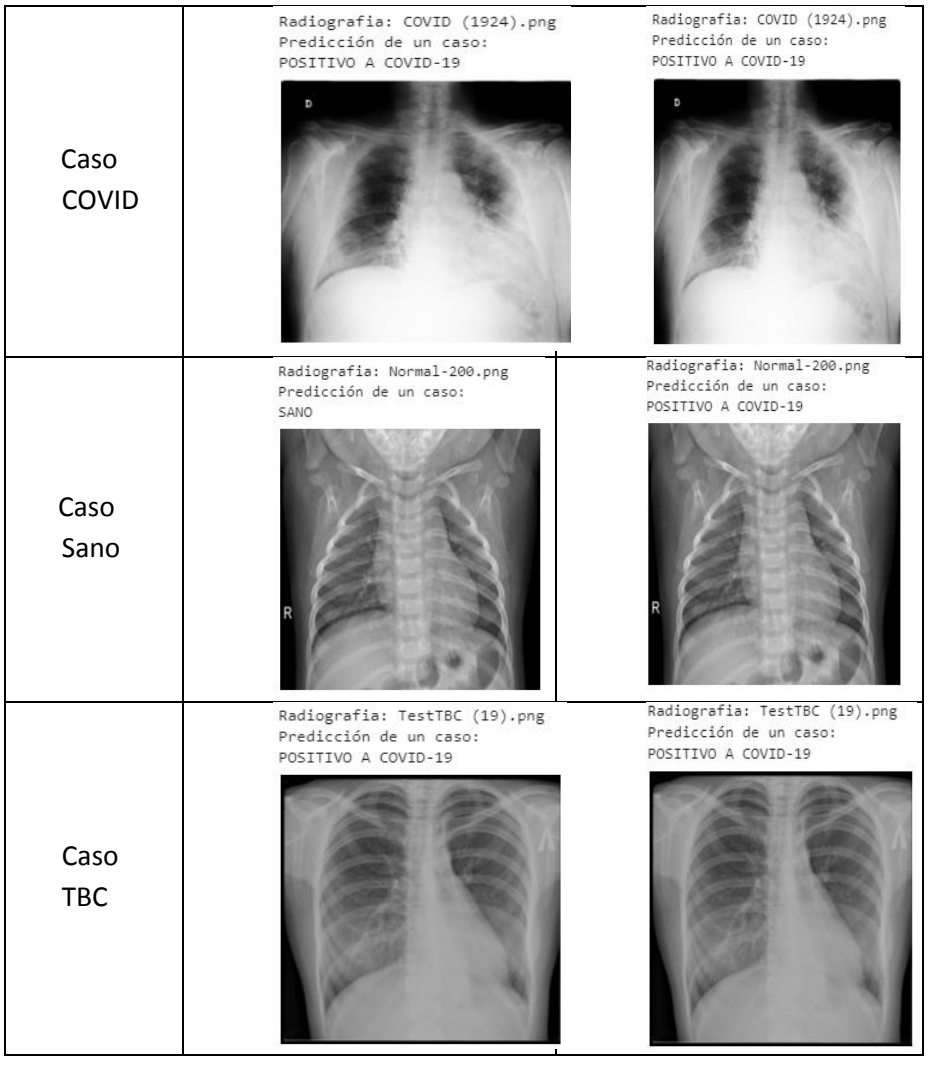


Figura Nº 41: Pruebas de Validación del modelo InceptionV3 usando CPU y GPU respectivamente.

Fuente: Elaboración propia (2021)

4.2.3. Matriz de Confusión y Resultados

El desempeño del modelo INCEPTION V3 es analizado mediante la Matriz de confusión, Matriz de observación y Métricas de evaluación, las cuales fueron calculadas por las ecuaciones expuestas en la sección 2.3.9.

En la Figura N°42 se observa la matriz de confusión obtenida al finalizar el proceso de validación de las 3 clases haciendo uso de la CPU y GPU. Esta matriz muestra los verdaderos positivos y brindó datos para el cálculo de los verdaderos negativos, falso positivos y falsos negativos, para cada una de las tres clases.

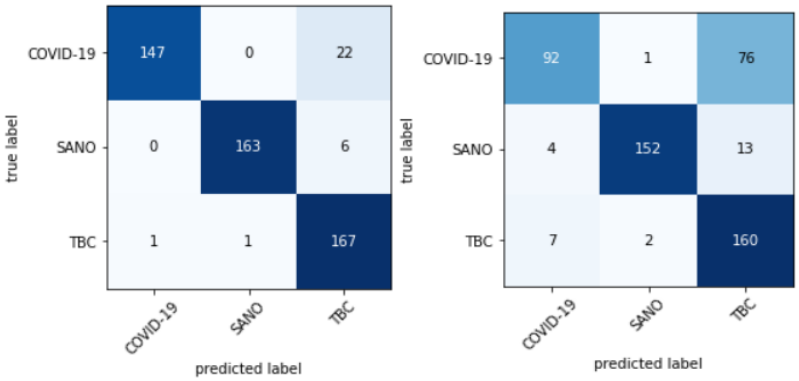


Figura Nº 42: Matriz de Confusión del modelo INCEPTION V3 usando CPU y GPU respectivamente

Fuente: Elaboración propia (2021)

A continuación, se muestran las tablas N°5 y N°6, las cuales son las matrices de observación del modelo de red haciendo uso del CPU y GPU. Dichos valores fueron calculados a partir de la matriz de confusión.

Tabla N° 5: Matriz de Observación del modelo INCEPTION V3 usando CPU.

	TP	**TN**	**FP**	**FN**
Covid-19	147	337	1	22
Sano	163	337	1	6
Tuberculosis	167	310	28	2

Fuente: Elaboración propia (2021)

Tabla N° 6: Matriz de Observación del modelo INCEPTIONV3 usando GPU.

	TP	TN	FP	FN
Covid-19	92	327	11	77
Sano	152	335	3	17
Tuberculosis	160	249	89	9

Fuente: Elaboración propia (2021)

Las tablas N°7 y N°8 se obtuvieron mediante cálculo con los datos de la Matriz de observación, dichas ecuaciones fueron explicadas en la sección 2.3.9, los cuales permiten calcular el "Accuracy", "Precision", "Recall", "F1 Score" y "General Precision". De esta forma pudimos conocer la precisión del modelo de red para 3 clases después del entrenamiento usando CPU y GPU.

Tabla N° 7: Métricas de medición del modelo INCEPTIONV3 usando CPU.

Indicadores	Accuracy	Precision	Recall	Specifity	F1-Score	General Precision	General Precision (%)
Covid-19	0.9546	0.9932	0.8698	0.9970	0.9274	0.9408	94.08%
Sano	0.9862	0.9939	0.9645	0.9970	0.9790		
Tuberculosis	0.9408	0.8564	0.9882	0.9172	0.9176		

Fuente: Elaboración propia (2021)

Tabla N° 8: Métricas de medición del modelo INCEPTIONV3 usando GPU.

Indicadores	Accuracy	Precision	Recall	Specifity	F1-Score	General Precision	General Precision (%)
Covid-19	0.8264	0.8932	0.5444	0.9675	0.6765	0.7968	79.68%
Sano	0.9606	0.9806	0.8994	0.9911	0.9383		
Tuberculosis	0.8067	0.6426	0.9467	0.7367	0.7656		

Fuente: Elaboración propia (2021)

Los resultados obtenidos en las tablas N°7 y N°9 muestran el desempeño de la red InceptionV3 con una Precisión General de 94.08% con CPU y con GPU 79.68% respectivamente. Estos valores indican que el

entrenamiento de la red fue óptimo cuando se realizó con la CPU que con GPU.

4.3. Pruebas y resultados del modelo VGG16

4.3.1. Proceso de Validación

El proceso de validación del modelo VGG16 fue establecido con 120 “épocas”, para medir la validación en la red, se consideró el “Accuracy” y el “Loss”, estos parámetros se consideraron en el entrenamiento de modelo VGG16 con CPU y GPU los cuales se observan en las Figuras N° 43 y N° 44 respectivamente.

Sin embargo, pudimos ver que para el caso donde se utilizó la CPU se encuentra un mayor sobreajuste entre las épocas 60-70 y 100-110, para el uso del GPU el sobreajuste de las épocas se ubica entre 80-90 y 110-120.

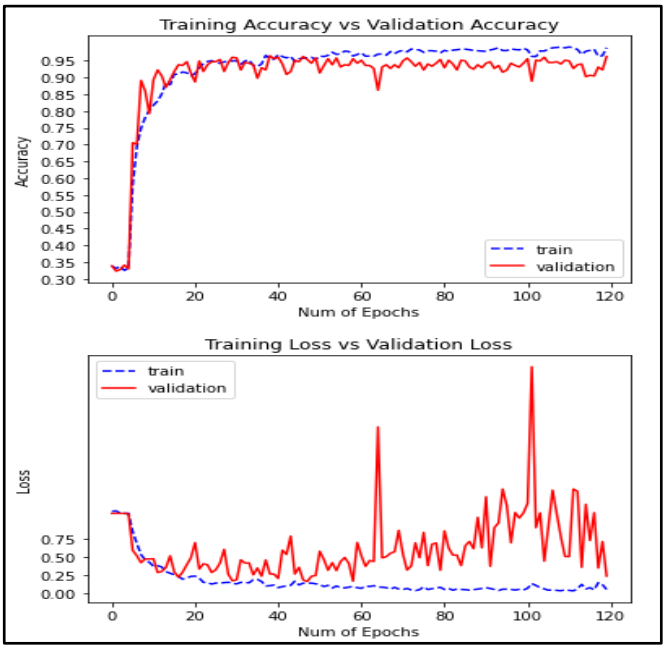


Figura Nº 43: Gráfica de Validación de Accuracy y Loss del modelo VGG16 usando CPU.

Fuente: Captura de pantalla del editor Jupyter Lab.

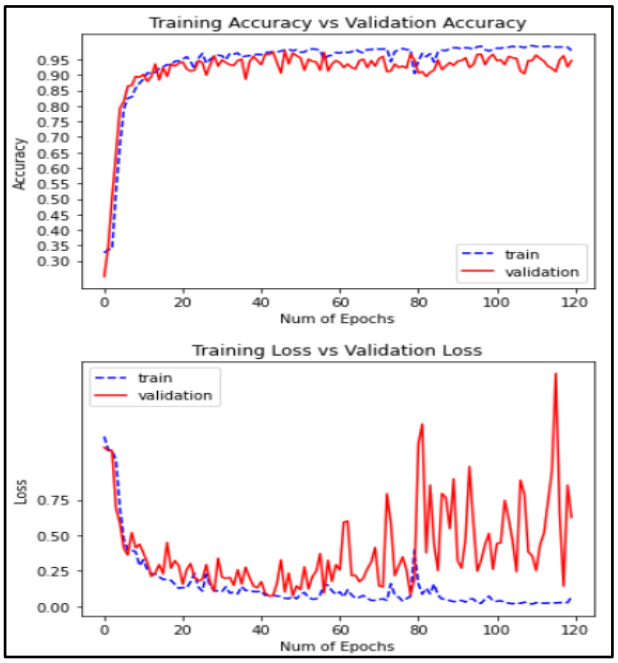


Figura Nº 44: Gráfica de Validación de Accuracy y Loss del modelo VGG16 usando GPU.

Fuente: Captura de pantalla del editor Jupyter Lab.

4.3.2. Pruebas de Validación

Se realizaron pruebas de validación para verificar el adecuado funcionamiento del modelo de red VGG16, mediante el algoritmo presentado en los Anexo N°6 y N°7. Este algoritmo hizo uso de las imágenes correspondientes a la carpeta "Testeo". En la siguiente Figura N°45 se muestran las pruebas de validaciones realizadas mediante una imagen de radiografía de tórax frontal el cual corresponde a cada uno de los casos vistos (Covid-19, Tuberculosis y Sano), además se realiza la comparativa entre la validación obtenida mediante el entrenamiento con la CPU y la GPU.

Caso COVID	Radiografia: COVID (1630).png Predicción de un caso: POSITIVO A COVID-19 	Radiografia: COVID (1630).png Predicción de un caso: POSITIVO A COVID-19 
Caso Sano	Radiografia: Normal-185.png Predicción de un caso: SANO 	Radiografia: Normal-185.png Predicción de un caso: SANO 
Caso TBC	Radiografia: Tuberculosis (480).png Predicción de un caso: POSITIVO A TBC 	Radiografia: Tuberculosis (480).png Predicción de un caso: POSITIVO A TBC 

Figura Nº 45: Pruebas de Validación del modelo VGG16 usando CPU y GPU respectivamente.

Fuente: Elaboración propia (2021)

4.3.3. Matriz de Confusión y Cálculos.

En esta sección se presenta la matriz de confusión, Matriz de observación y Métricas de evaluación del modelo VGG16, las cuales fueron calculadas a partir de las ecuaciones explicadas en la sección 2.3.9.

En la Figura N°46 se observa la matriz de confusión obtenida al finalizar el proceso de validación de las 3 clases haciendo uso de la CPU y GPU. En esa matriz se obtuvo los verdaderos positivos y brindó datos para el

cálculo de los verdaderos negativos, falso positivos y falsos negativos, para cada una de las tres clases.

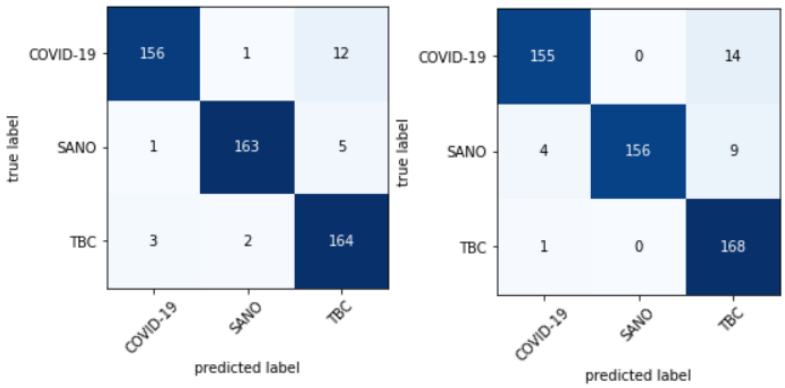


Figura Nº 46: Matriz de Confusión del modelo VGG16 usando CPU y GPU respectivamente.

Fuente: Elaboración propia (2021)

A continuación, se muestran las tablas N°9 y N°10, las cuales son las matrices de observación del modelo de red VGG16 haciendo uso del CPU y GPU. Dichos valores fueron determinados a través de la matriz de confusión.

Tabla N° 9: Matriz de Observación del modelo VGG16 usando CPU.

	TP	**TN**	**FP**	**FN**
Covid-19	156	334	4	13
Sano	163	335	3	6
Tuberculosis	164	321	17	5

Fuente: Elaboración propia mediante el software Excel (2021)

Tabla N° 10: Matriz de Observación del modelo VGG16 usando GPU.

	TP	**TN**	**FP**	**FN**
Covid-19	155	333	5	14
Sano	156	338	0	13
Tuberculosis	168	315	23	1

Fuente: Elaboración propia mediante el software Excel (2021)

Los valores mostrados en las tablas N°11 N°12 fueron obtenidos a partir de los datos de la Matriz de observación, los cuales permiten calcular el "Accuracy", "Precision", "Recall", "F1 Score" y "General Precision".

Dichos parámetros ayudan a conocer la precisión del modelo de red para 3 clases después del entrenamiento usando CPU y GPU.

Tabla N° 11: Métricas de evaluación del modelo VGG16 usando CPU.

Indicadores	Accuracy	Precision	Recall	Specifity	F1-Score	General Precision	General Precision (%)
Covid-19	0.9665	0.9750	0.9231	0.9882	0.9483	0.9527	95.27%
Sano	0.9822	0.9819	0.9645	0.9911	0.9731		
Tuberculosis	0.9566	0.9061	0.9704	0.9497	0.9371		

Fuente: Elaboración propia mediante el software Excel (2021)

Tabla N° 12: Métricas de medición del modelo VGG16 usando GPU.

Indicadores	Accuracy	Precision	Recall	Specifity	F1-Score	General Precision	General Precision (%)
Covid-19	0.9625	0.9688	0.9172	0.9852	0.9422	0.9448	94.48%
Sano	0.9744	1.0000	0.9231	1.0000	0.9600		
Tuberculosis	0.9527	0.8796	0.9941	0.9320	0.9333		

Fuente: Elaboración propia mediante el software Excel (2021)

De las tablas N°11 y N°12 se puede conocer el desempeño de la red. Se observa que para el caso con CPU la red tiene una Precisión General de 95.27% y con la GPU 94.48%. Estos resultados indican que el entrenamiento de la red fue óptimo cuando se realizó con la CPU que con GPU.

4.4. Comparación de las Redes Neuronales Convolucionales

Luego de haber obtenido los resultados de cada una de los modelos de redes (particular, InceptionV3 y VGG16), se hizo una comparación de estas mismas para poder visualizar el desempeño de cada una y poder saber cuál tuvo un mejor rendimiento. En la tabla N°13 se observa la comparación de los 3 modelos de redes con cada uno de sus métricas de evaluación.

Tabla N° 13: Comparación de los tres modelos a nivel de métricas de evaluación.

		Accuracy	Precision	Recall	Specifity	F1-Score
PARTICULAR	**CPU**	0.9513	0.9333	0.9270	0.9635	0.9273
	GPU	0.9619	0.9452	0.9428	0.9714	0.9432
INCEPTION V3	**CPU**	0.9606	0.9479	0.9408	0.9704	0.9413
	GPU	0.8646	0.8388	0.7968	0.8984	0.7934
VGG16	**CPU**	0.9684	0.9543	0.9527	0.9763	0.9529
	GPU	0.9632	0.9494	0.9448	0.9724	0.9452

Fuente: Elaboración propia mediante el software Excel (2021)

En la tabla N°14 se observa la precisión general de los tres modelos de redes convolucionales, entrenados con CPU y GPU respectivamente.

Tabla N° 14: Precisión General de los tres modelos de redes convolucionales

CNN	**PARTICULAR**		**INCEPTION V3**		**VGG16**	
	CPU	GPU	CPU	GPU	CPU	GPU
General Precision (%)	92.70%	94.28%	94.08%	79.68%	95.27%	94.48%

Fuente: Elaboración propia mediante el software Excel (2021)

4.5. Presupuesto

En la tabla N°15, se presenta el detalle de los gastos realizados para el desarrollo del proyecto de tesis.

Tabla N° 15: Presupuesto del Proyecto de Tesis.

Dispositivos	**Precio (S/.)**
Intel Core i5 9400f	600
Disco solido Gigabyte 512gb, M.2, 2280, NVME PCIe Gen3.0x4	450
Utilidades de escritorio	30
ASUS ROG STRIX GeForce RTX 2070 Overclocked 8G GDDR6	2500
TOTAL	**3580**

Fuente: Elaboración propia mediante el software Excel (2021)

CONCLUSIONES

1. Se logró diseñar una red neuronal particular con 10 capas, implementado en el lenguaje de programación Python con librerías de Tensorflow y Keras, asimismo la etapa de entrenamiento se hizo con CPU y GPU lo cual para el caso de CPU se obtuvo un 92.70% de precisión general y un 94.28% para el caso de la GPU tal como se puede apreciar en las tablas N°3 y N°4.
2. Se logró aplicar Transfer Learning a los dos modelos de redes, INCEPTIONV3 y VGG16, para poder entrenarlas con imágenes digitalizadas de radiografía de tórax frontal, las cuales en su etapa de entrenamiento para el modelo INCEPTIONV3, usando el CPU se obtuvo una precisión general de 94.08% y para el caso de la GPU un 79.68% las cuales se observan en las tablas N°7 y N°8, para el modelo VGG16, haciendo uso del CPU se obtuvo una precisión general de 95.27% y con la GPU 94.48% esta se muestra en las tablas N°11 y N°12.
3. Se logró evaluar los tres modelos de redes neuronales convolucionales de las cuales se determina que el mejor rendimiento y con menor sobreajustes para el entrenamiento fue la red particular, esto lo podemos observar en las tablas N°13 y N°14, así también en las gráficas N°35 y N°36 de la red particular y para el modelo VGG las gráficas N°43 y N°44.

RECOMENDACIONES

1. Se recomienda utilizar otros editores o software compatible con Python que permita obtener una gráfica mientras se aplica el entrenamiento y validación de la red neuronal convolucional, ya que con ello es posible ver si se está aplicando o no un sobreajuste.
2. Para poder aumentar la precisión general de la red es recomendable aumentar la cantidad de imágenes, así mismo tendría un impacto en el tiempo del procesamiento de entrenamiento de la red.

REFERENCIAS BIBLIOGRÁFICAS

Alzubaidi, L. (2021, 31 marzo). *Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data.* https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8

Bravo, C. E. F. (2017, 28 agosto). *Una aproximación práctica a las redes neuronales artificiales*. Universidad Del Valle. https://bibliotecadigital.univalle.edu.co/handle/10893/10330

Caya Pérez, J. C. (2020). *Evaluación de Modelos de Redes Neuronales Convolucionales aplicado a Radiografías de Tórax, para apoyar al proceso de diagnóstico de Neumonía asociada al Covid-19.* Retrieved from https://repositorio.urp.edu.pe/bitstream/handle/URP/3523/ELEC-T030_46733086_T%20%20%20CAYA%20P%C3%89REZ%20JHAN%20CARLOS.pdf?sequence=1&isAllowed=y

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.

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

Google Cloud (2021) *Advanced Guide to Inception v3 on Cloud TPU*.Recuperado el 01 de Julio de 2021: https://cloud.google.com/tpu/docs/inception-v3-advanced?hl=en

Guangyu, J. (2021, 1 julio). *Classification of COVID-19 chest X-Ray and CT images using a type of dynamic CNN modification method*. ScienceDirect. https://www.sciencedirect.com/science/article/abs/pii/S0010482521002195

Keras. (2021). *Keras documentation*: Keras Applications. Recuperado el 01 de Septiembre del 2021, de Keras.io: https://keras.io/api/applications

Jupyter. (10 de Septiembre de 2021). Recuperado el 19 de Septiembre de 2021, de Jupyter Hub: https://jupyter.org/

Liu, C., Cao, Y., Alcantara, M., Liu, B., Brunette, M., Peinado, J., & Curioso, W. (2017). TX-CNN: *Detecting tuberculosis in chest X-ray images using convolutional neural network*. 2017 IEEE International Conference on Image Processing (ICIP), 2314–2318. IEEE.

Luca Ai (2021) *What is Python?*. Retrieved 02 Julio 2021 from https://luca-d3.com/data-speaks/technology-dictionary/python

Manolis Loukadakis, José Cano, Michael O'Boyle (2019) *Accelerating Deep Neural Networks on Low Power Heterogeneous Architectures*. Recuperado el 01 de Julio del 2021 de Semantic Scholar: https://www.semanticscholar.org/paper/Accelerating-Deep-Neural-Networks-on-Low-Power-Loukadakis-Cano/781f37debd4cd16a673bba23886fc2780771e77c#citing-papers

Martínez Chamorro, E., Díez Tascón, A., Ibáñez Sanz, L., Ossaba Vélez, S., & Borruel Nacenta, S. (2021). *Radiologic diagnosis of patients with COVID-19. Diagnóstico radiológico del paciente con COVID-19*. Radiología, 63(1), 56–73. https://doi.org/10.1016/j.rx.2020.11.001

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

Mayo Clinic. Mayoclinic.org. (2021) *Rayos X de tórax* . Retrieved 26 Junio 2021,from https://www.mayoclinic.org/es-es/tests-procedures/chest-x-rays/about/pac-20393494.

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-machinelearning-model-ff9aa3bf7826

Narin, A., Kaya, C., & Pamuk, Z. (2020). *Automatic detection of coronavirus disease (COVID-19) using X-ray images and deep convolutional neural networks*. Pattern Analysis and Applications: PAA, 24(3), 1–14.

Organización Mundial de la Salud. (2020). *Información basíca sobre la COVID-19* Recuperado el 01 de Julio de 2021, de Organización Mundial de la Salud: https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/coronavirus-disease-covid-19

Organización Mundial de la Salud. (2020). *Tuberculosis.* Recuperado el 01 de Julio de 2021, de Organización Mundial de la Salud: https://www.who.int/es/news-room/fact-sheets/detail/tuberculosis

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/

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.

Tharwat A. (2018) *Classification assessment methods Faculty of Computer Science and Engineering, Frankfurt University of Applied Sciences*, 60318 Frankfurt am Main, Germany. Recuperado el 14 de Setiembre del 2021: https://www.researchgate.net/publication/327403649_Classification_assessment_methods_a_detailed_tutorial

Tawsifur Rahman, Amith Khandakar, Muhammad Enamul Hoque Chowdhury. (2020). *Tuberculosis (TB) Chest X-ray Database*. IEEE Dataport. https://dx.doi.org/10.21227/mps8-kb56

TensorFlow. (21 de Octubre de 2020). *Keras*. Recuperado el 02 de Julio de 2021, de TensorFlow: https://www.tensorflow.org/guide/keras?hl=es

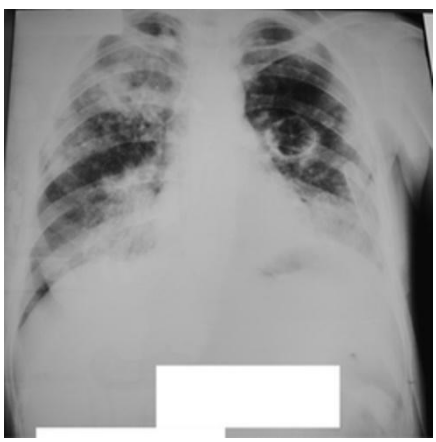
TensorFlow. (2021) *Por qué TensorFlow* , Recuperado el 02 de Julio de 2021, de https://www.tensorflow.org/?hl=es-419

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). *Convolutional neural networks: an overview and application in radiology*. Insights into Imaging, 9(4), 611–629. https://doi.org/10.1007/s13244-018-0639-9

Yusuf, K. (2020) *Ship Detection and Classification using type of Convolution Neural Networks* (Master of Science) http://earsiv.cankaya.edu.tr:8080/xmlui/bitstream/handle/20.500.12416/4898/Thesis.pdf?sequence=1&isAllowed=y

ANEXOS

Anexo 1: Imágenes Eliminadas base de datos.



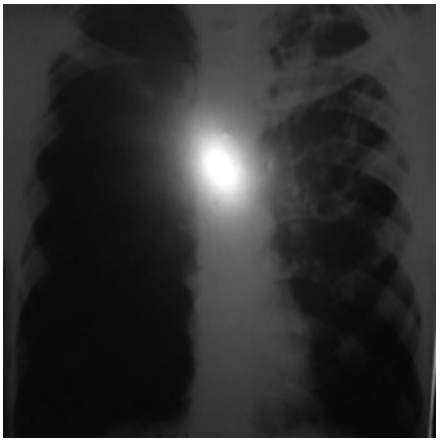


Imagen TBC con promedio de Pixeles 80 y 150

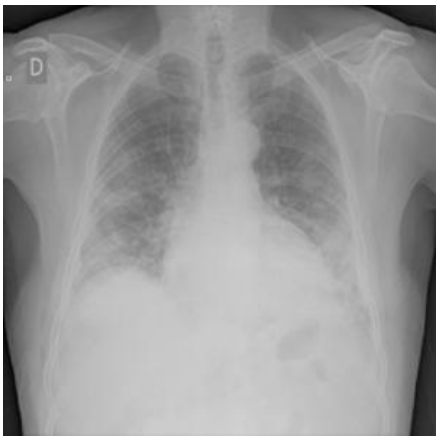


Imagen Covid-19 con promedio de pixeles 80 y 150

Anexo 2: Algoritmo para la Validación de red particular con CPU

```
# Validación

from keras.applications.imagenet_utils import preprocess_input, decode_predictions
from keras.models import load_model

import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline


names = ['POSITIVO A COVID-19','SANO','POSITIVO A TBC']

modelo= 'E:/TESIS 2021/Tesis 3 clases/modelos/cnnp120CPU.h5'
cnn = load_model(modelo)

imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Sano/Normal-185.png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Covid-19/COVID (1924).png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Tuberculosis/TestTBC (19).png"
imaget=cv2.resize(cv2.imread(imaget_path),
                  (224, 224),
                  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("Radiografia:",img)
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()
```

```
Radiografia: Normal-185.png
Predicción de un caso:
SANO
```

Anexo 3: Algoritmo para la Validación de red particular con GPU

```
# Validación GPU

from keras.applications.imagenet_utils import preprocess_input, decode_predictions
from keras.models import load_model

import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline


names = ['POSITIVO A COVID-19','SANO','POSITIVO A TBC']

modelo= 'E:/TESIS 2021/Tesis 3 clases/modelos/cnnp120h.h5'
cnn = load_model(modelo)

imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Sano/Normal-185.png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Covid-19/COVID (1924).png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Tuberculosis/TestTBC (19).png"
imaget=cv2.resize(cv2.imread(imaget_path),
                  (224, 224),
                  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("Radiografia:",img)
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()
```

```
Radiografia: Normal-185.png
Predicción de un caso:
SANO
```

Anexo 4: Algoritmo para la Validación del modelo InceptionV3 con CPU

```
# Validación de Modelo Inception con CPU

.applications.imagenet_utils import preprocess_input, decode_predictions
from keras.models import load_model

import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline


names = ['POSITIVO A COVID-19','SANO','POSITIVO A TBC']

modelo= 'E:/TESIS 2021/Tesis 3 clases/modelos/CPUinceptionv3.h5'
cnn = load_model(modelo)

imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Sano/Normal-200.png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Tuberculosis/TestTBC (19).png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Covid-19/COVID (1924).png"

imaget=cv2.resize(cv2.imread(imaget_path),
                  (299, 299),
                  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("Radiografia:",img)
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()
```

```
Radiografia: Normal-200.png
Predicción de un caso:
SANO
```

Anexo 5: Algoritmo para la Validación del modelo InceptionV3 con GPU

```
# Validación

#GPU
#from keras.applications.imagenet_utils import preprocess_input, decode_predictions
#from tensorflow.keras.models import load_model
#CPU
from keras.applications.imagenet_utils import preprocess_input, decode_predictions
from keras.models import load_model

import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline


names = ['POSITIVO A COVID-19','SANO','POSITIVO A TBC']

modelo= 'E:/TESIS 2021/Tesis 3 clases/modelos/inceptionv3.h5'
cnn = load_model(modelo)

imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Sano/Normal-200.png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Covid-19/COVID (1924).png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Tuberculosis/TestTBC (19).png"
imaget=cv2.resize(cv2.imread(imaget_path),
                  (299, 299),
                  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("Radiografia:",img)
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()
```

```
Radiografia: Normal-200.png
Predicción de un caso:
POSITIVO A COVID-19
```

Anexo 6: Algoritmo para la Validación del modelo VGG16 con CPU

```
# Validación VGG16 CPU

from keras.applications.imagenet_utils import preprocess_input, decode_predictions
from keras.models import load_model

import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline


names = ['POSITIVO A COVID-19','SANO','POSITIVO A TBC']

modelo= 'E:/TESIS 2021/Tesis 3 clases/modelos/CPUVGG16.h5'
cnn = load_model(modelo)

imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Sano/Normal-185.png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Covid-19/COVID (1630).png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Tuberculosis/Tuberculosis (480).png"
imaget=cv2.resize(cv2.imread(imaget_path),
                  (224, 224),
                  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("Radiografia:",img)
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()
```

```
Radiografia: Normal-185.png
Predicción de un caso:
SANO
```

Anexo 7: Algoritmo para la Validación del modelo VGG16 con GPU

```
# Validación VGG16 con GPU

from keras.applications.imagenet_utils import preprocess_input, decode_predictions
from keras.models import load_model

import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline


names = ['POSITIVO A COVID-19','POSITIVO A TBC','SANO']

modelo= 'E:/TESIS 2021/Tesis 3 clases/modelos/VGG16ii.h5'
cnn = load_model(modelo)

#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Sano/Normal-200.png"
imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Covid-19/COVID (1630).png"
#imaget_path = "E:/TESIS 2021/Tesis 3 clases/dataset/test/Tuberculosis/Tuberculosis (480).png"
imaget=cv2.resize(cv2.imread(imaget_path),
                  (224, 224),
                  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("Radiografia:",img)
#print("Entrenamiento 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()
```

```
Radiografia: COVID (1630).png
Predicción de un caso:
POSITIVO A COVID-19
```