UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

ESCUELA DE POSTGRADO

MAESTRÍA EN CIENCIAS MENCIÓN INFORMÁTICA

TESIS

OPTIMIZACIÓN DE LA ARQUITECTURA DE UNA RED NEURONAL CONVOLUCIONAL PARA EL RECONOCIMIENTO DE COVID-19 EN IMÁGENES DE RAYOS-X DE TÓRAX UTILIZANDO EL ALGORITMO METAHEURISTICO BAT

PARA OPTAR AL GRADO ACADÉMICO DE MAESTRO EN CIENCIAS MENCIÓN INFORMÁTICA

AUTOR:

Br. Iván César Medrano Valencia

ASESOR:

Mgt. Javier Arturo Rozas Huacho

CÓDIGO ORCID:

0000-0003-3080-1530

CUSCO - PERU

2023

INFORME DE ORIGINALIDAD

(Aprobado por Resolución Nro.CU-303-2020-UNSAAC)

El que suscribe, **Asesor** del trabajo de investigación/tesis titulada: OPTIMIZACIÓN DE LA ARQUITECTURA DE UNA RED NEURONAL CONVOLUCIONAL PARA EL RECONOCIMIENTO DE COVID-19 EN IMAGENES DE RAYOS -X DE TÓRAX UTILIZANDO EL ALGORITMO METAHEURÍSTICO BAT

presentado por: IVAN CESAR MEDRANO VALENCIA con DNI Nro.: 23881501 presentado por: ..... con DNI Nro.: ..... para optar el título profesional/grado académico de MAESTRO EN CIENCIAS MENCIÓN INFORMATICA

Informo que el trabajo de investigación ha sido sometido a revisión por 2 veces, mediante el Software Antiplagio, conforme al Art. 6° del ***Reglamento para Uso de Sistema Antiplagio de la UNSAAC*** y de la evaluación de originalidad se tiene un porcentaje de 6 %.

Evaluación y acciones del reporte de coincidencia para trabajos de investigación conducentes a grado académico o título profesional, tesis

Porcentaje	Evaluación y Acciones	Marque con una (X)
Del 1 al 10%	No se considera plagio.	X
Del 11 al 30 %	Devolver al usuario para las correcciones.	
Mayor a 31%	El responsable de la revisión del documento emite un informe al inmediato jerárquico, quien a su vez eleva el informe a la autoridad académica para que tome las acciones correspondientes. Sin perjuicio de las sanciones administrativas que correspondan de acuerdo a Ley.	

Por tanto, en mi condición de asesor, firmo el presente informe en señal de conformidad y **adjunto** la primera página del reporte del Sistema Antiplagio.

Cusco, 08 de FEBRERO de 2024

.....
Firma

Post firma Javier Arturo Rozas Huacho

Nro. de DNI 23847232

ORCID del Asesor 0000-0003-3080-1530

Se adjunta:

1. Reporte generado por el Sistema Antiplagio.
2. Enlace del Reporte Generado por el Sistema Antiplagio: **oid:** 27259:295607415

turnitin **Identificación de reporte de similitud**: oid:27259:295607415

NOMBRE DEL TRABAJO
Tesis_maestria_dictaminado_Ivan_Medrano.docx

AUTOR
Iván César Medrano Valencia

RECUENTO DE PALABRAS
14310 Words

RECUENTO DE CARACTERES
77476 Characters

RECUENTO DE PÁGINAS
70 Pages

TAMAÑO DEL ARCHIVO
5.1MB

FECHA DE ENTREGA
Dec 14, 2023 9:01 AM GMT-5

FECHA DEL INFORME
Dec 14, 2023 9:03 AM GMT-5

6% de similitud general
El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para cada base c

- 6% Base de datos de Internet
- 3% Base de datos de publicaciones
- Base de datos de Crossref
- Base de datos de contenido publicado de Crossr

Excluir del Reporte de Similitud

- Base de datos de trabajos entregados
- Material bibliográfico
- Material citado
- Material citado
- Coincidencia baja (menos de 20 palabras)

ÍNDICE GENERAL

ÍNDICE GENERAL i
LISTA DE TABLAS iv
LISTA DE FIGURAS v
RESUMEN vii
ABSTRACT viii
INTRODUCCIÓN ix
1. PLANTEAMIENTO DEL PROBLEMA 1
1.1 Situación Problemática 1
1.2 Formulación del problema 2
1.2.1 Problema general 2
1.2.2 Problemas específicos 2
1.3 Justificación de la investigación. 3
1.4 Objetivos de la investigación. 4
2. MARCO TEÓRICO CONCEPTUAL 5
2.1 Bases Teóricas. 5
2.1.1 Neurona Artificial 5
2.1.2 Red Neuronal Artificial. 7
2.1.3 Entrenamiento de una red neuronal artificial 8
2.1.4 Evaluación de modelos predictivos 9
2.1.5 Optimización. 12
2.1.6 Optimización combinatoria. 13
2.1.7 Algoritmos Metaheurísticos 13
2.2 Marco Conceptual 16
2.2.1 El COVID-19. 16
2.2.2 Clasificación de imágenes con Redes Neuronales Convolucionales 18

2.2.3 Hiperparámetros de una red neuronal convolucional 29
2.2.4 Optimización de Hiperparámetros 29
2.2.5 Soluciones existentes para la optimización de hiperparámetros. 32
2.2.6 Metaheurística BAT. 37
2.2.7 Antecedentes empíricos de la investigación 42
3. METODOLOGÍA. 46
3.1 Ámbito de estudio. 46
3.2 Tipo y nivel de investigación. 46
3.3 Unidad de análisis. 46
3.4 Etapas de la investigación. 46
4. RESULTADOS Y DISCUSIÓN 47
4.1 El conjunto de datos de COVID-19. 47
4.2 Preprocesamiento. 48
4.3 Arquitectura de la red neuronal convolucional. 48
4.4 Experimentos 49
4.4.1 Selección de los hiperparámetros a optimizar. 49
4.4.2 Parámetros de la red neuronal convolucional. 50
4.4.3 Implementación de la metaheurística BAT. 51
4.4.4 Hardware y software 56
4.4.5 Hiperparámetros óptimos. 56
4.4.6 Arquitectura óptima de la CNN 56
4.4.7 Resultados del modelo óptimo. 57
4.4.8 Implementación de Optimización Bayesiana de los hiperparámetros de la CNN. 60
4.4.9 Discusión de resultados. 61
5. CONCLUSIONES Y RECOMENDACIONES 64
CONCLUSIONES 64

RECOMENDACIONES 66
BIBLIOGRAFIA..... 67
ANEXOS..... 70

LISTA DE TABLAS

Tabla 2.1 Tipos de niveles de redes neuronales artificiales y sus definiciones 8

Tabla 2.2 Definición de exploración y explotación. 15

Tabla 2.3 Lista de funciones de activación(Mirza Rahim Baig, 2020) 26

Tabla 2.4 Hiperparámetros relacionados con la estructura de una red neuronal 30

Tabla 2.5 Hiperparámetros relacionados con el entrenamiento de una red neuronal 31

Tabla 4.1 Distribución de los datos 47

Tabla 4.2 Arquitectura de la red neuronal convolucional base 48

Tabla 4.3. Resumen de hiperparámetros optimizados en antecedentes 49

Tabla 4.4. Rango de valores de los hiperparámetros a optimizar 51

Tabla 4.5 Valores óptimos de los hiperparámetros encontrados por BAT 56

Tabla 4.6 Detalle de la matriz de confusión 59

Tabla 4.7. Resultados de la Optimización Bayesiana de la CNN 60

Tabla 4.8. Valores de los hiperparámetros con optimización bayesiana 61

Tabla 4.9. Exactitud (accuracy) obtenida en la investigación comparada con otros resultados. 61

Tabla 4.10. Tiempo de ejecución de algoritmos de optimización 62

LISTA DE FIGURAS

Figura 2.1 Neurona Biológica 5

Figura 2.2 Neurona Artificial 6

Figura 2.3 Conexión de tres nodos o neuronas artificiales 7

Figura 2.4 Arquitectura de una Red Neuronal Artificial 8

Figura 2.5 Matriz de confusión (Josh Patterson, 2017) 10

Figura 2.6 Diferentes tipos de algoritmos metaheurísticos 15

Figura 2.7 Rayos X de tórax, opacidades marcadas con flechas que confirman COVID-19 18

Figura 2.8 Representación digital de una imagen 19

Figura 2.9 Dimensiones de una imagen digital 19

Figura 2.10 Ejemplo de mapa de características de borde vertical 21

Figura 2.11 Ejemplo de dos matrices 22

Figura 2.12 Secuencia de la operación de convolución 23

Figura 2.13 Ejemplo de salto (stride) 24

Figura 2.14 Ejemplo de relleno 24

Figura 2.15 Estructura de una CNN simple (Mirza Rahim Baig, 2020) 26

Figura 2.16 Max pooling y Average pooling 27

Figura 2.17 Arquitectura de una CNN con agrupación máxima 28

Figura 2.18 Enfoques para la selección automatizada de hiperparámetros 32

Figura 2.19 Búsqueda en cuadrícula de dos hiperparámetros (Tanay, 2021) 33

Figura 2.20 Comparación de la búsqueda en cuadrícula (izquierda) con la búsqueda aleatoria (derecha) 34

Figura 2.21 Estimación inicial del modelo sustituto (Frazier, 2018) 36

Figura 2.22 Función sustituta coincide con la función real 37

Figura 2.23 Trayectoria de un murciélago buscando su presa 38

Figura 2.24 Algoritmo metaheurístico BAT (Yang, 2010) 41
Figura 4.1 Distribución de imágenes de entrenamiento 47
Figura 4.2. Arquitectura óptima de la CNN 57
Figura 4.3 Resultados de pérdida del modelo óptimo 57
Figura 4.4 Exactitud de entrenamiento y validación 58
Figura 4.5 Matriz de confusión del modelo óptimo 59

RESUMEN

La enfermedad del COVID-19 aún está afectando nuestra sociedad, por lo que un diagnóstico preciso y rápido es muy importante para prestar una atención eficaz a las personas afectadas. Esto se puede lograr aplicando técnicas computacionales inteligentes como las Redes Neuronales Convolucionales (CNN) a imágenes de rayos X de tórax. Existen numerosos trabajos que utilizan modelos de aprendizaje profundo para diagnosticar la gravedad de los afectados con COVID-19 con resultados bastante interesantes. La exactitud con la que las CNN pueden clasificar imágenes, depende en gran medida de los valores de los hiperparámetros con los que fueron entrenados. Encontrar esos valores óptimos es una tarea desafiante ya que representa un problema de optimización combinatoria muy complejo. En esta investigación se propone utilizar el algoritmo metaheurístico BAT para encontrar los hiperparámetros óptimos de una CNN para detectar COVID-19 en imágenes de rayos X de tórax. Con tal propósito se ha utilizado un conjunto de datos con 284 imágenes clasificadas en “Covid” y “Normal”. Los hiperparámetros encontrados por la metaheurística BAT permiten que el modelo optimizado obtenga una exactitud de 98% para datos de validación, lo que demuestra que la metaheurística BAT es eficiente para realizar optimización de hiperparámetros en Redes Neuronales Convolucionales.

Palabras clave: Metaheurística BAT, Optimización de hiperparámetros, Redes Neuronales Convolucionales, COVID-19.

ABSTRACT

The COVID-19 disease is still affecting our society, so an accurate and rapid diagnosis is very important to provide effective care to affected people. This can be achieved by applying intelligent computational techniques such as Convolutional Neural Networks (CNN) to chest X-ray images. There are numerous works that use deep learning models to diagnose the severity of those affected by COVID-19 with quite interesting results. The accuracy with which CNNs can classify images depends largely on the values of the hyperparameters with which they were trained. Finding those optimal values is a challenging task since it represents a very complex combinatorial optimization problem. This research proposes to use the BAT metaheuristic algorithm to find the optimal hyperparameters of a CNN to detect COVID-19 in chest X-ray images. For this purpose, a data set with 284 images classified into “Covid” and “Normal” has been used. The hyperparameters found by the BAT metaheuristic allow the optimized model to obtain an accuracy of 98% for validation data, which demonstrates that the BAT metaheuristic is efficient for performing hyperparameter optimization in Convolutional Neural Networks.

Keywords: BAT Metaheuristics, Hyperparameter Optimization, Convolutional Neural Networks, COVID-19.

INTRODUCCIÓN

Las redes neuronales convolucionales (CNN) han logrado en los últimos años resultados que antes se consideraban puramente del ámbito humano (Zhou Kevin, 2020). Son ampliamente utilizadas y efectivas para tareas de visión artificial, como el reconocimiento de acciones humanas, la localización de objetos, la detección de peatones, y reconocimiento facial (Lopez Pinaya, 2020). Sin embargo, las redes neuronales convolucionales no se limitan solo a tareas de visión artificial, las CNN han demostrado ser efectivas para el lenguaje natural y el procesamiento del habla, logrando excelentes resultados en clasificación de oraciones, modelado de oraciones y reconocimiento de voz (Lopez Pinaya, 2020). Actualmente, se utilizan ampliamente diversas herramientas para el diseño de una red neuronal convolucional, como Matlab, Keras, Tensorflow, Pytorch, etc. Varios modelos de redes convolucionales pre entrenadas han demostrado tener buena precisión, las más populares son LeNet-5, AlexNet y VGG entre otras. Sin embargo, aunque los modelos son diferentes, todas las CNN se componen de capas convolucionales, capas de agrupación y capas completamente conectadas (Ragav Venkatesan, 2018).

El rendimiento de una red neuronal convolucional depende en gran medida de los valores de sus hiperparámetros (Graupe, 2013). Las CNN se componen de muchos hiperparámetros, como el número de capas convolucionales, el número de filtros y sus respectivos tamaños, etc. Elegir los valores correctos para los hiperparámetros siempre ha sido una tarea muy importante y difícil porque depende del conocimiento y la experiencia del investigador. En la práctica, normalmente la selección de hiperparámetros se realiza manualmente, sin embargo, computacionalmente es muy costosa porque se deben evaluar diferentes conjuntos de valores de hiperparámetros para verificar cuál de ellos tuvo mejores resultados, por lo que implica un consumo razonable de hardware y tiempo (Abdelrahman Ezzeldin Nagib, 2022).

Existen muchos algoritmos de optimización de hiperparámetros que han demostrado ser competitivos y, en algunos casos, han superado a los expertos humanos. Algunos algoritmos son la búsqueda en cuadrícula, la búsqueda aleatoria, la optimización bayesiana y los algoritmos que equilibran la explotación y exploración del espacio de búsqueda (Lorenzo, 2017).

Por otro lado, los algoritmos metaheurísticos son algoritmos aproximados de búsqueda y optimización de propósito general. Recientemente, muchos algoritmos metaheurísticos se están aplicando con éxito para resolver problemas intratables o difíciles de resolver con métodos clásicos en diversas áreas como finanzas, medicina, ingeniería, etc. La ventaja de utilizar estos algoritmos para resolver problemas complejos es que obtienen las mejores u óptimas soluciones incluso para problemas de gran tamaño en razonables períodos de tiempo (Tansel Dokeroglu, 2019).

En la actualidad se han desarrollado muchos algoritmos metaheurísticos de optimización modernos basados en inteligencia de enjambre inspirados en la naturaleza (Yang, 2010). Uno de ellos es el algoritmo de murciélagos (BAT), que se basa en la función de búsqueda de alimento de los micro murciélagos (Yang, 2010).

La metaheurística BAT es interesante porque combina las principales ventajas de los algoritmos de optimización de enjambre de partículas (PSO), la búsqueda de armonía (HS) y el recocido simulado (SA), en condiciones apropiadas. Emplea no sólo una técnica de sintonización de frecuencia para aumentar la diversidad de soluciones en la población, sino también el mecanismo automático para equilibrar la exploración y la explotación durante el proceso de búsqueda, variando entre las tasas de emisión de pulsos y la sonoridad de los murciélagos.

Por otro lado, el COVID-19 ha infectado a 257 millones de personas desde su aparición en diciembre de 2019 y ha provocado la muerte de 5.15 millones de personas. El estudio inicial sobre la propagación de COVID-19 informó que el virus es altamente infeccioso y se propaga principalmente debido a gotitas al toser o estornudar. Para su diagnóstico solamente se contaba con las denominadas RT-PCR que implica un método invasivo para recolectar muestras, que generalmente se toman frotando la nariz o la garganta del paciente. Las desventajas de las pruebas RT-PCR incluyen un alto número de resultados falsos, baja calidad de la muestra y retraso en la obtención de los resultados. Algunos pacientes con infecciones graves desarrollan insuficiencia respiratoria y necesitan cuidados intensivos dentro de las 48 horas posteriores a la infección por el virus COVID-19. Para estos pacientes con infecciones graves, se debe utilizar ventilación mecánica inmediatamente para cuidados intensivos. Esto implica que existe una gran necesidad de detectar con precisión el virus y utilizar herramientas de diagnóstico rápidas y confiables en lugar de solo RT-PCR. Como resultado, la tomografía computarizada y los rayos X son métodos

alternativos para diagnosticar pacientes con neumonía por COVID-19 mediante la detección de las características radiográficas típicas.

En esta investigación se propone utilizar el algoritmo metaheurístico BAT para determinar los hiperparámetros óptimos de una red neuronal convolucional con el propósito de clasificar con mayor precisión imágenes de rayos X de tórax para detectar el COVID-19. Se construirá una red neuronal convolucional desde cero (*from scracht*) utilizando la librería *tensorflow* cuya arquitectura inicial se diseña en base a los artículos de Deepa S. (2022), Talha (2021) y Muhammad Talha Nafees (2021). Después la CNN se somete a la búsqueda del conjunto de hiperparámetros óptimos con la metaheurística BAT utilizando un conjunto de datos bastante reducido debido al costo computacional requerido.

1. PLANTEAMIENTO DEL PROBLEMA

1.1 Situación Problemática

El COVID-19 se descubrió por primera vez en Wuhan, China, en diciembre de 2019. |vieron afectados por esta enfermedad a los pocos meses de su primer caso en China (F. Wu, 2020).

Con el aumento constante de la tasa de infección, la detección rápida se convirtió en una necesidad urgente que ayude con un diagnóstico rápido y automático a través de imágenes de rayos X de tórax.

Las pruebas de imagen tienen un papel importante en la detección y manejo de estos pacientes y se han utilizado para apoyar el diagnóstico, determinar la gravedad de la enfermedad, guiar el tratamiento y valorar la respuesta terapéutica (E. Martínez Chamorro, 2021).

Recientemente, las Redes Neurales Convolucionales (CNN) presentaron resultados optimistas cuando se utilizan para clasificar imágenes radiológicas. Las CNN se utilizan en muchas aplicaciones de la vida real, incluida la clasificación de imágenes.

Las Redes Neuronales Convolucionales (CNN) son un modelo de aprendizaje profundo ampliamente utilizadas para clasificación de datos usando muchas capas de neuronas para extraer la jerarquía de características de imágenes, videos, etc. (Yann Lecun, 1998).

El objetivo de los modelos de aprendizaje profundo como las CNN, es que con la mayor precisión puedan clasificar imágenes después de su entrenamiento. Pero esta precisión puede verse afectada al variar los parámetros de entrenamiento tales como: tasa de aprendizaje, número de épocas de entrenamiento, tamaño del lote, optimizador utilizado, tamaño del filtro, el tipo de función de activación, número de capas, número de neuronas por capa, etc. a los que se les denominan hiperparámetros. El impacto de los hiperparámetros en la precisión se descubre más empíricamente que teóricamente y su compleja interacción mutua sigue siendo desconocida (Dmytro Mishkina, 2016).

Cuando enfrentamos la tarea de entrenar una CNN con un conjunto de datos determinado, no solo es necesario probar un modelo, sino ajustar sus hiperparámetros para mejorar la precisión original. Pero el número de todas las combinaciones de hiperparámetros crecerá exponencialmente con el aumento del tamaño de la CNN.

El problema radica en que, hacer una búsqueda a través de todas esas combinaciones es solo posible para modelos pequeños de CNN y ciclos de entrenamiento cortos; pero para redes más complejas la búsqueda de hiperparámetros para optimizar la arquitectura de una red neuronal convolucional es un problema de optimización combinatoria y es necesario considerar estrategias de búsqueda inteligentes debido a la gran cantidad de valores posibles para cada hiperparámetro, que si se considera la estrategia de búsqueda manual, es posible que se pase por alto algunas buenas combinaciones.

1.2 Formulación del problema

1.2.1 Problema general

¿En qué medida el algoritmo metaheurístico BAT permite optimizar la arquitectura de una red neuronal convolucional para detectar COVID-19 en imágenes de rayos X de tórax?

1.2.2 Problemas específicos

1. ¿Cuáles son los hiperparámetros a optimizar en una red neuronal convolucional para detectar COVID-19 en imágenes de rayos X de tórax?
2. ¿Cuál es la función de aptitud que se utilizará para optimizar la arquitectura de una red neuronal convolucional para detectar COVID-19 en imágenes de rayos X de tórax, aplicando el metaheurístico BAT
3. ¿Cuál es la eficiencia en términos de tiempo de ejecución del algoritmo metaheurístico BAT al optimizar una red neuronal convolucional para detectar COVID-19 en imágenes de rayos X de tórax?

1.3 Justificación de la investigación.

En marzo de 2020, la Organización Mundial de la Salud declaró la pandemia mundial por el virus COVID-19. La razón de su crecimiento exponencial es que se propaga a través del tacto, el aliento o la tos de una persona infectada con COVID-19. La tos de un paciente con COVID-19 emite gotitas, que pueden entrar en la boca o la nariz de una persona sana y provocar la misma infección (F. Wu, 2020).

Hay muchas características clínicas de COVID-19 que varían de un caso a otro, desde un estado asintomático hasta el síndrome de dificultad respiratoria aguda y la disfunción multiorgánica, pero hay muchas características clínicas comunes como tos, dolor de garganta, dolor de cabeza, fiebre (no en todos casos), mialgia y dificultad para respirar, en algunos casos, la infección puede progresar al final de la primera semana a neumonía, luego insuficiencia respiratoria y muerte (Singhal, 2020).

Desafortunadamente, es muy difícil diferenciar la infección por COVID-19 de todos los tipos de infecciones virales respiratorias como influenza, parainfluenza, virus sincitial respiratorio, coronavirus no COVID-19, etc. (Singhal, 2020). La detección efectiva de pacientes con COVID-19 es el paso más crítico e importante para dar atención oportuna a los pacientes. En este contexto, la radiografía de tórax es uno de los enfoques de detección más importantes debido a que la radiografía de los pacientes muestra anomalías en las imágenes que se utilizan como características de la infección (Shuai Wang, 2020).

Por lo que es necesario contar con herramientas más precisas de diagnóstico que apoyen al personal médico utilizando imágenes de tórax de rayos X. Una de esas herramientas nos proporciona la inteligencia artificial, muy utilizada en estos tiempos para resolver diversos problemas de la vida cotidiana y permite a través del aprendizaje profundo con las Redes Neuronales Convolucionales (CNN) clasificar imágenes, por tanto, los hiperparámetros de estos modelos deben de configurarse de forma inteligente para obtener la máxima precisión posible. En consecuencia, se necesita un algoritmo para hacer una búsqueda eficiente de buenos valores de hiperparámetros.

Recientemente, muchos algoritmos metaheurísticos se están aplicando con éxito para resolver problemas intratables o difíciles de resolver con métodos clásicos en

diversas áreas como finanzas, medicina, ingeniería, etc. (Angel Gaspar, 2021). La ventaja de usar estos algoritmos para resolver problemas complejos es que obtienen las mejores u óptimas soluciones incluso para problemas muy grandes en pequeñas cantidades de tiempo (Tansel Dokeroglu, 2019).

La búsqueda de los mejores hiperparámetros de entrenamiento de una CNN es un problema de optimización combinatoria, en consecuencia, es posible aplicar algoritmos metaheurísticos para su solución tal como el algoritmo BAT basado en el comportamiento de ecolocalización de los murciélagos (Yang, 2010).

Es por eso que en esta investigación se propone utilizar el algoritmo BAT para optimizar la arquitectura de una CNN para detectar COVID-19 en imágenes de rayos X de tórax.

1.4 Objetivos de la investigación.

a) Objetivo general

Determinar en qué medida el algoritmo metaheurístico BAT permite optimizar una red neuronal convolucional para detectar COVID-19 en imágenes de rayos X de tórax.

b) Objetivos específicos

1. Determinar cuáles son los hiperparámetros a optimizar en una Red Neuronal Convolucional para detectar COVID-19 en imágenes de rayos X de tórax.
2. Determinar la función de aptitud que se utilizará para optimizar la arquitectura de una red neuronal convolucional para detectar COVID-19 en imágenes de rayos X de tórax, aplicando el metaheurístico BAT
3. Determinar cuál es la eficiencia en términos de tiempos de ejecución del algoritmo metaheurístico BAT al optimizar una Red Neuronal Convolucional para detectar COVID-19 en imágenes de rayos X de tórax.

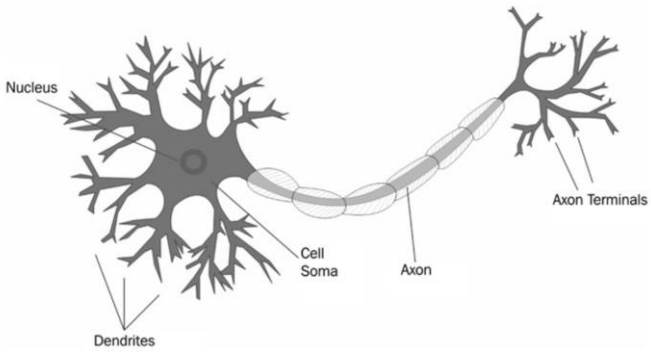
2. MARCO TEÓRICO CONCEPTUAL

2.1 Bases Teóricas.

2.1.1 Neurona Artificial

Las neuronas artificiales están inspiradas en las neuronas biológicas que están formadas por las dendritas, el soma y el axón, ver Figura 2.1. Las dendritas son las encargadas de captar los impulsos nerviosos emitidos por otras neuronas. Posteriormente, estos impulsos son procesados en el soma y transmitidos a través del axón, que emite un impulso nervioso hacia las neuronas vecinas (Graupe, 2013).

Figura 2.1 Neurona Biológica

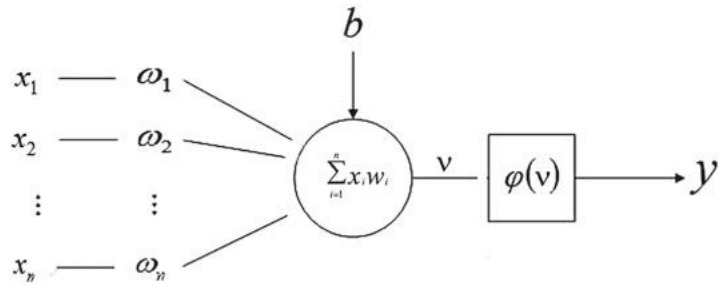


De Graupe (2013)

Por lo general, una neurona recibe información de miles de otras neuronas y, a su vez, envía información a miles de otras neuronas. Entonces, el procesamiento de la información es esencialmente paralelo (Angel Gaspar, 2021)

A partir de los aspectos funcionales de la neurona biológica en las neuronas artificiales, la suma de las entradas multiplicada por sus pesos asociados determina el impulso nervioso que recibe la neurona. Este valor se procesa dentro de la celda empleando una función de activación no lineal y un sesgo que devuelve un valor que se envía como salida de la neurona. Este proceso se puede ver en la Figura 2.2.

Figura 2.2 Neurona Artificial



De Graupe (2013)

La Figura 2.2 también muestra el flujo de la señal de forma explícita. Donde x_1, x_2 hasta x_n son las señales de entrada. w_1, w_2 hasta w_n son los pesos de las señales correspondientes, y b es el sesgo, donde el sesgo o umbral se puede ver como un número que indica a partir de qué valor la neurona artificial o nodo produce una salida significativa. La señal de entrada se multiplica por el peso antes de llegar al nodo. Una vez que las señales ponderadas se recopilan en el nodo, estos valores se suman para obtener la suma ponderada y el resultado más el sesgo se pasa a través de una función de activación no lineal. Por lo tanto, se puede expresar como en las ecuaciones (2.1), (2.2) y (2.3).

$$v = \sum_{i=1}^{n} x_i w_i + b \tag{2.1}$$

$$v = (w_1 \times x_1) + (w_2 \times x_1) + \ldots + (w_n \times x_n) + b \tag{2.2}$$

$$y = \varphi(v) \tag{2.3}$$

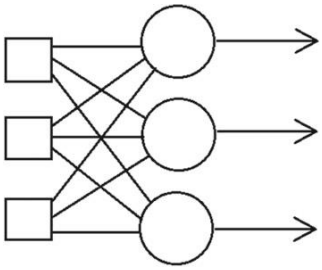
Hay varias funciones de activación no lineales, sin embargo, la más utilizada para una tarea de clasificación binaria es la función sigmoidea, que se muestra en la ecuación (2.4).

$$\varphi(x) = \frac{1}{1+e^{-x}} \tag{2.4}$$

Para una tarea de salida múltiple, por ejemplo, clasificar varias clases, puede agregar fácilmente neuronas artificiales con sus respectivos pesos y conexiones, vea la Figura 2.3. Se omitieron el sesgo y los valores por simplicidad ilustrativa. En la Figura 2.3, los cuadrados simulan señales de

entrada que no deben confundirse con neuronas, y los círculos simulan neuronas o nodos artificiales. Cabe señalar que la Figura. 2.2 y la Figura 2.3 se consideran arquitecturas de redes neuronales de una sola capa. Para modelos multiclase, es común utilizar la función de activación softmax (ecuación 2.5), donde los valores de salida pueden interpretarse como una probabilidad de pertenecer a una clasificación.

Figura 2.3 Conexión de tres nodos o neuronas artificiales



De Graupe (2013)

$$\varphi(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \tag{2.5}$$

donde *x* es un vector y *j* indexa los nodos de salida y *j = 1,2, …, k*

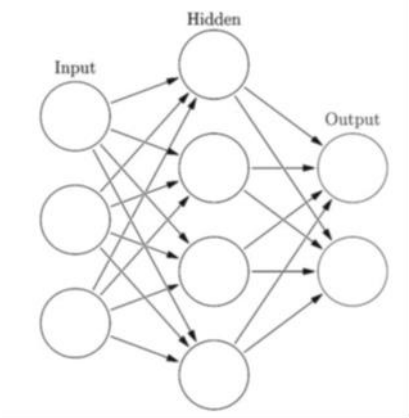
2.1.2 Red Neuronal Artificial.

Una sola neurona artificial, también conocida como red neuronal artificial de una sola capa o perceptrón, tiene una baja capacidad de procesamiento y su aplicación en problemas reales es muy limitada debido a su separación lineal para una tarea de clasificación, aunque utiliza una función de activación no lineal. Las redes neuronales artificiales o también conocidas como perceptrones multilineales agregan una capa llamada “oculta” entre la capa de entrada y la capa de salida como se puede ver en la Figura 2.4 (Graupe, 2013).

Es muy importante saber identificar la distribución de neuronas dentro de una red neuronal artificial, por lo tanto, si un conjunto de neuronas artificiales recibe simultáneamente la misma información, se le llama “capa”. La Tabla 2.1 muestra los tipos de niveles de las redes neuronales artificiales y sus

definiciones. Teóricamente, es posible aplicar diferentes tipos de funciones de activación para diferentes capas o neuronas artificiales, sin embargo, en la práctica, es común aplicar la misma función de activación para todas las capas ocultas (Angel Gaspar, 2021).

Figura 2.4 Arquitectura de una Red Neuronal Artificial



De Graupe (2013)

Tabla 2.1 Tipos de niveles de redes neuronales artificiales y sus definiciones

Nivel	**Definición**
Entrada	Es el conjunto de neuronas que recibe información directamente de fuentes externas en la red
Oculta	Corresponde a un conjunto de neuronas internas que no tienen contacto directo con el exterior de la red neuronal artificial. Generalmente, el número de niveles ocultos va de 1 a un número alto. Por lo general, las neuronas en cada nivel oculto comparten el mismo tipo de información
Salida	Es el conjunto de neuronas que trasladan al exterior la información que la red ha procesado previamente

De Graupe (2013)

2.1.3 Entrenamiento de una red neuronal artificial

En el caso de la red neuronal artificial, se trata de un algoritmo de aprendizaje supervisado, el entrenamiento consiste en un proceso iterativo de ajuste de los pesos entre las neuronas dentro de la red neuronal artificial, de la misma

manera que un cerebro humano aprende al estrechar las conexiones entre neuronas (Bengio, 2009).

Es decir, la modificación de estos pesos se realiza para que la salida de la red neuronal artificial sea lo más parecida posible a la salida proporcionada por el supervisor o salida deseada. El entrenamiento de la red neuronal artificial se formula como un problema de optimización para encontrar el valor mínimo. Donde la diferencia entre la predicción y la meta establecida por el supervisor se puede medir con una función de pérdida. Es fácil ver que cuanto menor sea el valor de la función de pérdida, mayor será la precisión de la red neuronal artificial. Algunos de los optimizadores más utilizados en la práctica para esta tarea son los basados en el gradiente descendente (Bengio, 2009).

2.1.4 Evaluación de modelos predictivos

La evaluación de modelos es el proceso de comprender qué tan bien dan la clasificación correcta y luego medir el valor de la predicción en un contexto determinado. A veces, solo nos importa la frecuencia con la que un modelo obtiene una predicción correcta; otras veces, es importante que el modelo obtenga cierto tipo de predicción correcta con más frecuencia que las demás. La herramienta básica para evaluar modelos es la matriz de confusión (Josh Patterson, 2017)

2.1.4.1 Matriz de Confusión

La matriz de confusión (Figura 2.5), es una tabla de filas y columnas que representa las predicciones y los resultados reales (etiquetas) para un clasificador. Usamos esta tabla para comprender mejor qué tan bien se está desempeñando el modelo o clasificador en función de dar la respuesta correcta en el momento adecuado

Figura 2.5 Matriz de confusión (Josh Patterson, 2017)

	Predicted Positive	Predicted Negative	
Actual Positive	TP *True Positive*	FN *False Negative*	Sensitivity $\frac{TP}{(TP+FN)}$
Actual Negative	FP *False Positive*	TN *True Negative*	Specificity $\frac{TN}{(TN+FP)}$
	Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$

De Josh Patterson (2017)

Se miden estas respuestas contando el número de los siguientes:

- Verdaderos positivos
  - Predicción positiva
  - La etiqueta fue positiva
- Falsos positivos
  - Predicción positiva
  - La etiqueta era negativa
- Verdaderos negativos
  - Predicción negativa
  - La etiqueta era negativa
- Falsos negativos
  - Predicción negativa
  - La etiqueta fue positiva

Se pueden realizar diferentes evaluaciones del modelo basado en combinaciones de los cuatro recuentos antes mencionados en la matriz de confusión, tal como se detalla a continuación.

- Exactitud (Accuracy)

- Precisión
- Recuperación (Recall)
- Puntaje F1

Exactitud (Accuracy): La exactitud es el grado de cercanía de las mediciones de una cantidad al valor real de esa cantidad (Josh Patterson, 2017).

$$\textbf{Exactitud} = (TP + TN) / (TP + FP + FN + TN)$$

La exactitud puede ser engañosa en la calidad del modelo cuando el desequilibrio de clases es alto. Si simplemente clasificamos todo como la clase más grande, nuestro modelo obtendrá automáticamente un gran número de conjeturas correctas y nos proporcionará una puntuación de alta exactitud, pero una indicación de valor engañosa basada en una aplicación real del modelo (por ejemplo, nunca predecirá la clase más pequeña o evento raro)

Precisión: El grado en que las mediciones repetidas en las mismas condiciones nos dan los mismos resultados se llama precisión en el contexto de la ciencia y la estadística. La precisión también se conoce como el valor de predicción positivo.

$$\textbf{Precisión} = TP / (TP + FP)$$

Una medición puede ser exacta pero no precisa, no exacta pero todavía precisa, ni exacta ni precisa, o exacta y precisa a la vez. Consideramos que una medida es válida si es exacta y precisa (Josh Patterson, 2017).

Recuperación (Recall): La verdadera tasa positiva mide la frecuencia con la que clasificamos un registro de entrada como clase positiva y es la clasificación correcta. Esto también se llama sensibilidad o recuerdo. La sensibilidad cuantifica qué tan bien el modelo evita los falsos negativos (Josh Patterson, 2017)

$$\textbf{Sensibilidad} = TP / (TP + FN)$$

Puntaje F1: En la clasificación binaria, consideramos que la puntuación F1 (o puntuación F, medida F) es una medida de la precisión de un modelo.

La puntuación F1 es la media armónica de las medidas de precisión y recuperación (descritas anteriormente) en una sola puntuación, como se define aquí:

F1 = 2TP / (2TP + FP + FN)

Vemos puntajes para F1 entre 0.0 y 1.0, donde 0.0 es el peor puntaje y 1.0 es el mejor puntaje que nos gustaría ver. La puntuación F1 se usa normalmente en la recuperación de información para ver qué tan bien un modelo recupera resultados relevantes. En el aprendizaje automático, vemos que la puntuación F1 se usa como una puntuación general sobre el rendimiento de nuestro modelo (Josh Patterson, 2017).

2.1.5 Optimización.

Muchos problemas implican encontrar la mejor manera de realizar alguna tarea. A menudo esto implica encontrar el valor máximo o mínimo de alguna función: el tiempo mínimo para realizar un determinado viaje, el coste mínimo para realizar una tarea, la potencia máxima que puede generar un dispositivo, etc. Muchos de estos problemas se pueden resolver encontrando la función apropiada y luego usando técnicas de cálculo para encontrar el valor máximo o mínimo requerido (Chong Edwin, 2013).

Generalmente, un problema de este tipo tendrá la siguiente forma matemática: Encontrar el valor más grande (o más pequeño) de $f(x)$ cuándo $a \leq x \leq b$. A veces a o b son infinitos, pero frecuentemente el mundo real impone alguna restricción sobre los valores que x puede tener (Chong Edwin, 2013).

Este problema difiere en dos formas de los problemas locales de máximo y mínimo que se encuentran al graficar funciones: sólo nos interesa la función entre a y b, y se quiere saber el valor mayor o menor que toma $f(x)$ no simplemente valores que son los más grandes o los más pequeños en un intervalo pequeño. Es decir, no se busca un máximo o mínimo local sino un máximo o mínimo global, a veces también llamado máximo o mínimo absoluto (Chong Edwin, 2013).

2.1.6 Optimización combinatoria.

Los problemas de optimización en los que las variables de decisión son enteras, es decir, donde el espacio de soluciones está formado por ordenaciones o subconjuntos de números naturales, reciben el nombre de problemas de optimización combinatoria. En este caso, se trata de hallar el mejor valor de entre un número finito o numerable de soluciones viables. Sin embargo, la enumeración de este conjunto resulta prácticamente imposible, aún para problemas de tamaño moderado (Yepes Piqueras, 2002).

Las raíces históricas de la optimización combinatoria subyacen en ciertos problemas económicos: la planificación y gestión de operaciones y el uso eficiente de los recursos. Pronto comenzaron a modelizarse de esta manera aplicaciones más técnicas, y hoy vemos problemas de optimización discreta en diversas áreas: informática, gestión logística (rutas, almacenaje), telecomunicaciones, ingeniería, etc., así como para tareas variadas como el diseño de campañas de marketing, la planificación de inversiones, la división de áreas en distritos políticos, la secuenciación de genes, la clasificación de plantas y animales, el diseño de nuevas moléculas, el trazado de redes de comunicaciones, el posicionamiento de satélites, la determinación del tamaño de vehículos y las rutas de medios de transporte, la asignación de trabajadores a tareas, la construcción de códigos seguros, el diseño de circuitos electrónicos, etc. (Yepes Piqueras, 2002).

2.1.7 Algoritmos Metaheurísticos.

La optimización es un campo muy importante en todas las disciplinas ya que en cualquier disciplina es necesario minimizar costes, tiempo, distancia, riesgos, etc. en algunos casos es necesario maximizar la calidad, los beneficios, la eficiencia, etc. Esto depende del propósito y área para la cual se realiza la optimización. Por tanto, matemáticamente hablando, la optimización se puede resumir buscando el valor mínimo o máximo de una función objetivo. Existen métodos clásicos que se utilizan para optimizar funciones. Sin embargo, estos métodos se vuelven muy complejos a medida que aumenta el número de dimensiones de la función e incluso en algunos casos ineficientes. Por tanto, el uso de algoritmos metaheurísticos es la principal alternativa para

resolver esta clase de problemas ya que permite obtener una solución óptima o “casi” óptima entre un conjunto muy amplio de posibles soluciones con un tiempo y consumo de hardware razonables. El problema de optimización se puede formular como en la Ec. (2.6), donde se quiere encontrar el valor óptimo *x* que minimice o maximice la función objetivo *f(x)* (Bastien Chopard, 2018).

$$Minimizar / \max imizar,\ considerando \quad x \in X,\ f(x),\ x = (x_1, ..., x_d) \in ¡^{\ d} \tag{2.6}$$

donde *x* representa el vector de variables de decisión *d* representa su número y *X* representa el espacio de búsqueda. En la mayoría de los algoritmos, el espacio de búsqueda está en un rango definido por los límites inferiores *(li)* o los límites superiores *(ui)* de cada una de las variables de decisión *d*, ver Ecuación (2.7).

$$X = \{x \in R^d \mid l_i \le x_i \le u_i, i = 1, ..., d \mid\} \tag{2.7}$$

La relación entre la maximización y minimización de la función objetivo se puede ver en la Ecuación (2.8).

$$\max f(x) \Leftrightarrow \min -1 * f(x) \tag{2.8}$$

Por lo tanto, los algoritmos metaheurísticos son una familia de algoritmos aproximados de propósito general. Suelen ser procedimientos iterativos que guían una heurística de búsqueda subordinada, combinando inteligentemente diferentes conceptos para explorar y explotar adecuadamente el espacio de búsqueda. Algunas ventajas y desventajas son las siguientes:

Ventajas

- Algoritmos de propósito general
- Gran éxito en la práctica
- Fácilmente desplegable
- Fácilmente paralelizable

Desventajas

- Son algoritmos aproximados, no exactos
- Son no deterministas (probabilistas)

Los factores que pueden resultar interesantes para el uso de algoritmos metaheurísticos son los siguientes:

- Cuando no existe un método exacto de resolución, o requiere mucho tiempo de cómputo y memoria (ineficiente).
- Cuando no se necesita la solución óptima y basta con una de buena calidad.

El rendimiento de un algoritmo metaheurístico está fuertemente relacionado con sus operadores. Los operadores de los algoritmos metaheurísticos deben tener un buen equilibrio entre exploración y explotación para llegar a soluciones óptimas en un número limitado de iteraciones o para que converja lo antes posible en el óptimo global. La Tabla 2.2 muestra la definición de exploración y explotación (Bastien Chopard, 2018).

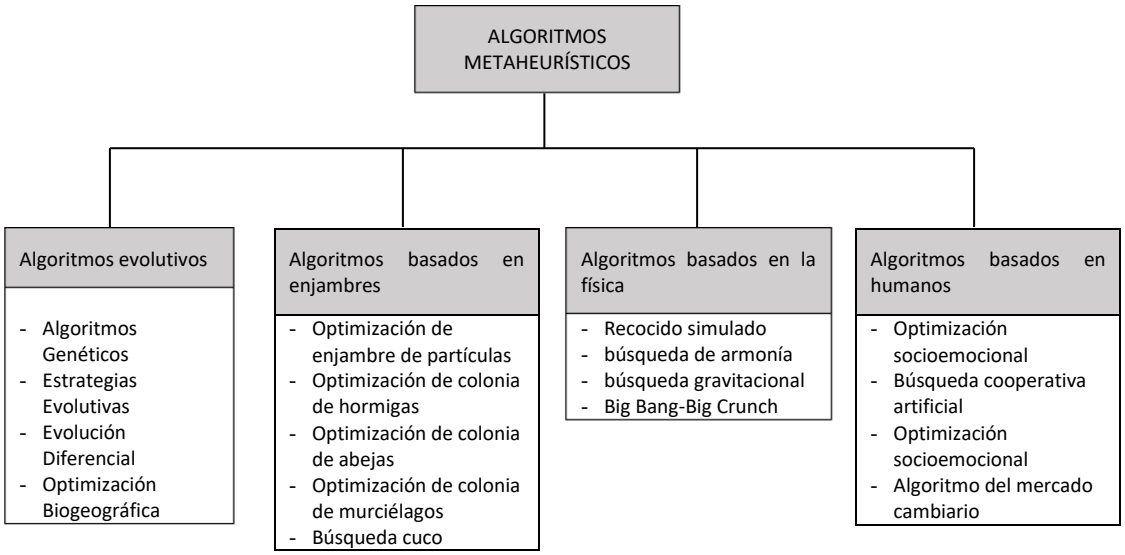
Tabla 2.2 Definición de exploración y explotación.

Exploración	**Explotación**
La exploración es la estrategia de búsqueda de nuevas soluciones.	La explotación es el proceso de mejorar (intensificar) las mejores soluciones encontradas durante el proceso de exploración, para encontrar soluciones de mayor calidad.

De Bastien Chopard (2018)

La Figura 2.6 muestra los diferentes tipos de algoritmos metaheurísticos que incluyen cuatro categorías principales.

Figura 2.6 Diferentes tipos de algoritmos metaheurísticos



De Tansel Dokeroglu (2019).

2.2 Marco Conceptual

2.2.1 El COVID-19.

En E. Martínez Chamorro (2021), el COVID-19 (enfermedad por coronavirus 2019) es una enfermedad infecciosa causada por una cepa de coronavirus denominada SARS-CoV-2 (síndrome respiratorio agudo severo coronavirus 2). Los primeros casos se vieron en Wuhan, China, a fines de diciembre de 2019 y desde allí se ha extendido rápidamente a prácticamente todo el mundo. Fue reconocida oficialmente como pandemia por la Organización Mundial de la Salud (OMS) el 11 de marzo de 2020.

La rápida expansión internacional del virus ha provocado un enorme impacto social, económico y sanitario y ha obligado a la adopción de medidas extraordinarias de confinamiento social para frenar su propagación y a la reestructuración sanitaria para evitar su colapso. La infección se transmite predominantemente a través del contacto con gotitas de secreciones del tracto respiratorio superior de las personas infectadas[2]. Las gotas contaminadas depositadas en los objetos pueden facilitar la transmisión del virus[3]. Otras vías de transmisión como la orofecal, sexual, sanguínea o vertical no están, en la actualidad, claras[4].

La infección ocurre generalmente dentro de los 14 días posteriores a la exposición y en la mayoría de los casos a los 4-5 días[5]. Aunque puede ocurrir a cualquier edad, es más frecuente en adultos varones de mediana edad y ancianos.

2.2.1.1 Características clínicas

Las características clínicas de COVID-19 son variadas y van desde un estado asintomático hasta un síndrome de dificultad respiratoria aguda y una disfunción multiorgánica. Las características clínicas comunes incluyen fiebre (no en todos), tos, dolor de garganta, dolor de cabeza, fatiga, dolor de cabeza, mialgia y dificultad para respirar. También se ha descrito conjuntivitis. Por tanto, son indistinguibles de otras infecciones respiratorias. En un subconjunto de pacientes, al final de la primera semana la enfermedad puede progresar a neumonía, insuficiencia respiratoria y muerte (Singhal, 2020).

2.2.1.2 Diagnóstico.

La prueba estándar para detectar SARS-CoV-2 es la reacción en cadena de la polimerasa con transcriptasa inversa (RT-PCR) obtenida habitualmente de muestra nasofaríngea o de secreciones respiratorias. La RT-PCR se cree que es altamente específica, pero la sensibilidad puede oscilar del 60-70%19 al 95-97%20, por lo que los falsos negativos son un problema clínico real, especialmente en las fases precoces. La sensibilidad varía según el tiempo transcurrido desde la exposición al SARS-CoV-2, con una tasa de falsos negativos del 100% el primer día después de la exposición, que disminuye al 38% el día de inicio de los síntomas y al 20% el tercer día de sintomatología, su nivel más bajo (E. Martínez Chamorro, 2021).

Las pruebas de imagen tienen un papel importante en la detección y manejo de estos pacientes y se han utilizado para apoyar el diagnóstico, determinar la gravedad de la enfermedad, guiar el tratamiento y valorar la respuesta terapéutica.

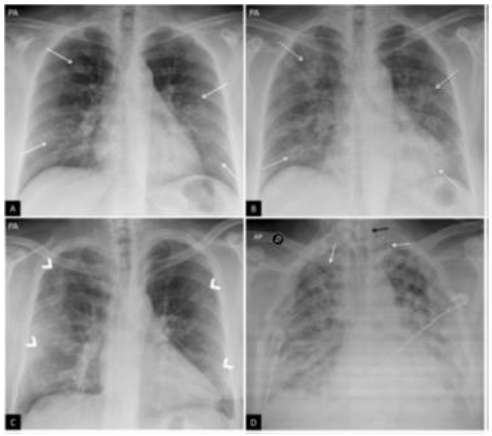
2.2.1.3 Diagnóstico de COVID-19 en imágenes de radiografía de tórax.

Los hallazgos de la radiografía de tórax en pacientes con sospecha de COVID-19 se han dividido en las siguientes categorías para facilitar el diagnóstico (E. Martínez Chamorro, 2021):

- Radiografía de tórax normal. No es infrecuente que la radiografía de tórax sea normal al principio de la enfermedad, por lo que una radiografía normal no excluye la infección.
- Hallazgos típicos o aquellos que se han asociado comúnmente en la literatura científica a COVID-19 (Figura. 2.7). Incluyen el patrón reticular, las opacidades en vidrio deslustrado y las consolidaciones, con morfología redondeada y una distribución multifocal parcheada o confluente. El diagnóstico diferencial incluye la neumonía organizada, la toxicidad farmacológica y otras causas de daño pulmonar agudo.
- Hallazgos atípicos o aquellos poco frecuentes o no descritos en neumonía COVID-19. Incluyen la consolidación lobar, el nódulo o la

masa pulmonar, el patrón miliar, la cavitación y el derrame pleural, descrito solo en el 3% de los pacientes y más típico de la enfermedad avanzada.

Figura 2.7 Rayos X de tórax, opacidades marcadas con flechas que confirman COVID-19



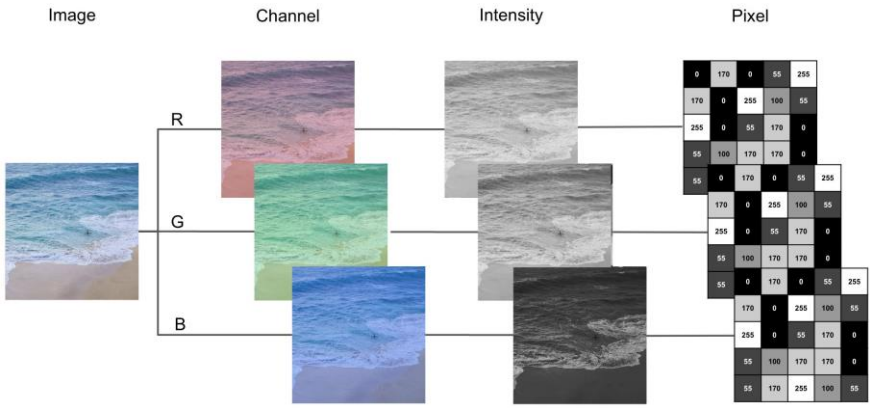
De E. Martínez Chamorro (2021)

2.2.2 Clasificación de imágenes con Redes Neuronales Convolucionales

2.2.2.1 Imágenes digitales

Los humanos pueden ver a través de sus ojos transformando la luz en señales eléctricas que luego son procesadas por el cerebro. Pero las computadoras no tienen ojos físicos para capturar la luz. Sólo pueden procesar información en formas digitales compuestas de bits (0 o 1). Entonces, para poder “ver”, las computadoras requieren una versión digitalizada de una imagen. Una imagen digital está formada por una matriz bidimensional de píxeles. Para una imagen en escala de grises, cada uno de estos píxeles puede tomar un valor entre 0 y 255 que representa su intensidad o nivel de gris. Una imagen digital puede estar compuesta por un canal para una imagen en blanco y negro o tres canales (rojo, azul y verde) para una imagen en color (Mirza Rahim Baig, 2020).

Figura 2.8 Representación digital de una imagen



De Mirza Rahim Baig (2020)

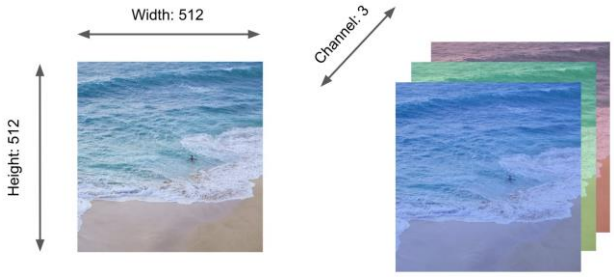
.

Una imagen digital se caracteriza por sus dimensiones (alto, ancho y canal):

- Alto: Cuántos píxeles hay en el eje vertical.
- Ancho: Cuántos píxeles hay en el eje horizontal.
- Canal: Cuántos canales hay. Si solo hay un canal, una imagen estará en escala de grises. Si hay tres canales, la imagen se coloreará.

La siguiente imagen digital tiene dimensiones (512, 512, 3).

Figura 2.9 Dimensiones de una imagen digital



De Mirza Rahim Baig (2020)

2.2.2.2 Procesamiento de imágenes

En Mirza Rahim Baig (2020) se menciona que las computadoras pueden usar la información de las imágenes digitales para encontrar patrones que usarán para clasificar una imagen o localizar objetos en ella. Entonces, para obtener información útil o procesable de una imagen, una computadora

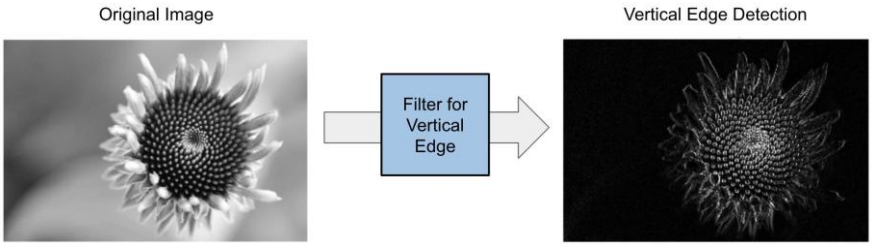
debe resolver una imagen en un patrón reconocible o conocido. Como ocurre con cualquier algoritmo de aprendizaje automático, la visión por computadora necesita algunas características para poder aprender patrones.

Además, a diferencia de los datos estructurados, donde cada característica está bien definida de antemano y almacenada en columnas separadas, las imágenes no siguen ningún patrón específico. Es imposible decir, por ejemplo, que la tercera línea siempre contendrá el ojo de un animal o que la esquina inferior izquierda siempre representará un objeto rojo de forma redonda. Las imágenes pueden ser de cualquier cosa y no siguen ninguna estructura. Por eso se consideran datos no estructurados.

Sin embargo, las imágenes contienen características. Contienen diferentes formas (líneas, círculos, rectángulos, etc.), colores (rojo, azul, naranja, amarillo, etc.) y características específicas relacionadas con diferentes tipos de objetos (pelo, rueda, hojas, etc.). Nuestros ojos y cerebro pueden analizar e interpretar fácilmente todas estas características e identificar objetos en imágenes. Por tanto, necesitamos simular el mismo proceso analítico para las computadoras. Aquí es donde entran en juego los filtros de imagen (también llamados *kernels*).

Los filtros de imagen son pequeñas matrices especializadas en detectar un patrón definido. Por ejemplo, podemos tener un filtro para detectar sólo líneas verticales y otro sólo para líneas horizontales. Los sistemas de visión por computadora ejecutan dichos filtros en cada parte de la imagen y generan una nueva imagen con los patrones detectados resaltados. Este tipo de imágenes generadas se denominan mapas de características. En la siguiente figura se muestra un ejemplo de un mapa de características donde se utiliza un filtro de detección de bordes:

Figura 2.10 Ejemplo de mapa de características de borde vertical



de Mirza Rahim Baig (2020)

2.2.2.3 Aumento de datos (data augmentation)

El aumento de datos es una técnica para aumentar artificialmente el conjunto de entrenamiento mediante la creación de copias modificadas de un conjunto de datos utilizando datos existentes (Mirza Rahim Baig, 2020). Incluye realizar cambios menores en el conjunto de datos o utilizar el aprendizaje profundo para generar nuevos datos. El aumento de datos se utiliza cuando:

- Es necesario evitar el sobreajuste del modelo.
- El conjunto de entrenamiento inicial es demasiado pequeño.
- Se requiere mejorar la precisión del modelo.
- Se quiere reducir el costo operativo de etiquetar y limpiar el conjunto de datos sin procesar.

Algunas técnicas para aumentar datos en imágenes son:

Transformaciones geométricas: voltear recortar, rotar, estirar y ampliar imágenes aleatoriamente. Se debe tener cuidado al aplicar múltiples transformaciones en las mismas imágenes, ya que esto puede reducir el rendimiento del modelo.

Transformaciones del espacio de color: cambiar aleatoriamente los canales de color RGB, el contraste y el brillo.

Filtros kernel: cambiar aleatoriamente la nitidez o el desenfoque de la imagen.

Borrado aleatorio: elimina alguna parte de la imagen inicial.

Mezclar imágenes: fusionar y mezclar múltiples imágenes.

2.2.2.4 Operaciones de convolución.

Como se mencionó anteriormente, la visión por computadora se basa en la aplicación de filtros a una imagen para reconocer diferentes patrones o características y generar mapas de características. Una operación de convolución se compone de dos etapas (Mirza Rahim Baig, 2020):

- Un producto por elementos de dos matrices
- Una suma de los elementos de una matriz.

Veamos un ejemplo de cómo convolucionar dos matrices, A y B:

Figura 2.11 Ejemplo de dos matrices

Matrix A

5	10	15
10	20	30
100	150	200

Matrix B

1	0	-1
2	0	-2
1	0	-1

de Mirza Rahim Baig (2020)

Primero, se realiza una multiplicación por elementos con las matrices A y B. Como resultado se obtiene otra matriz, C, con los siguientes valores:

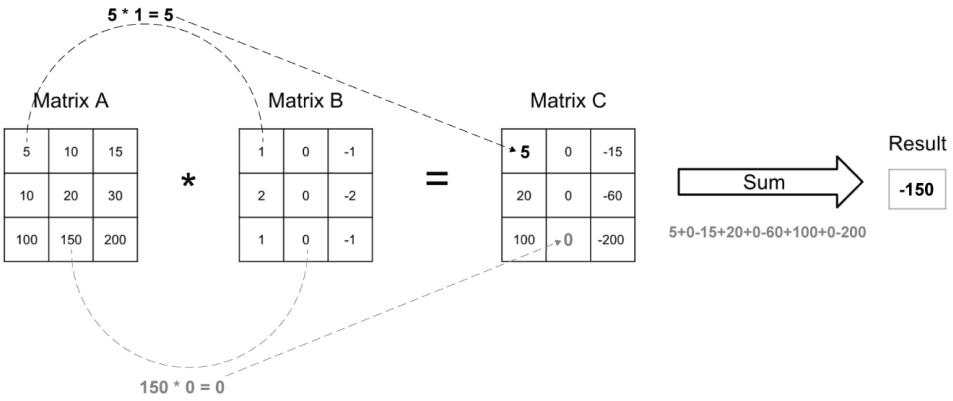
- 1.ª fila, 1.ª columna: 5 × 1 = 5
- 1.ª fila, 2.ª columna: 10 × 0 = 0
- 1.ª fila, 3.ª columna: 15 × (-1) = -15
- 2.ª fila, 1.ª columna: 10 × 2 = 20
- 2.ª fila, 2.ª columna: 20 × 0 = 0
- 2.ª fila, 3.ª columna: 30 × (-2) = -60
- 3.ª fila, 1.ª columna: 100 × 1 = 100
- 3.ª fila, 2.ª columna: 150 × 0 = 0
- 3.ª fila, 3.ª columna: 200 × (-1) = -200

Finalmente, solo queda realizar una suma de todos los elementos de la matriz C, lo que dará lo siguiente:

5+0-15+20+0-60+100+0-200 = -150

El resultado final de toda la operación de convolución en las matrices A y B es 150, como se muestra en el siguiente diagrama:

Figura 2.12 Secuencia de la operación de convolución



de Mirza Rahim Baig (2020)

En este ejemplo, Matrix B es en realidad un filtro (o núcleo) llamado *Sobel* que se utiliza para detectar líneas verticales (también existe una variante para líneas horizontales). La matriz A será una porción de una imagen con las mismas dimensiones que el filtro (esto es obligatorio para realizar la multiplicación por elementos).

Para una CNN, los filtros son en realidad parámetros que se aprenderán (es decir, se definirán) durante el proceso de entrenamiento. Así, los valores de cada filtro que se utilizarán los establecerá la propia CNN.

2.2.2.5 Salto (*Stride)*

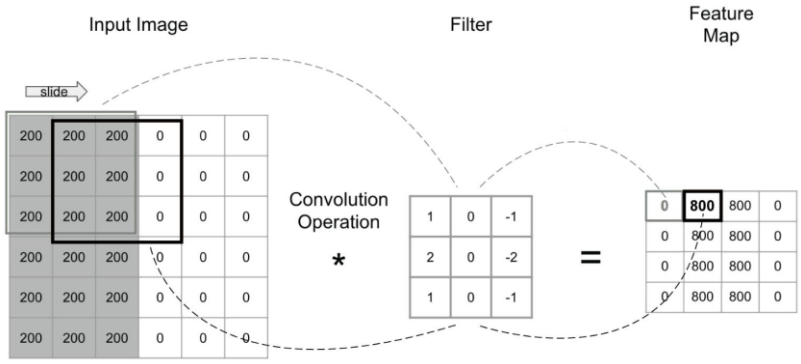
Una operación de convolución utiliza un filtro de un tamaño específico, por ejemplo (3, 3), es decir, 3 × 3, y lo aplica a una porción de la imagen de un tamaño similar. Si se tiene una imagen grande, por ejemplo, de tamaño (512, 512), entonces se puede mirar solo una parte muy pequeña de la imagen.

Tomando pequeñas partes de la imagen a la vez, se realiza la misma operación de convolución en todo el espacio de una imagen determinada. Para ello se aplica una técnica llamada deslizamiento. Como su nombre lo indica, deslizar es aplicar el filtro a un área adyacente de la operación de

convolución anterior: simplemente se desliza el filtro y se aplica la convolución (Mirza Rahim Baig, 2020).

Si se comienza desde la esquina superior izquierda de una imagen, se puede deslizar el filtro un píxel a la vez hacia la derecha. Una vez que se llega al borde derecho, se desliza el filtro hacia abajo un píxel. Se repite esta operación de deslizamiento hasta que se haya aplicado la convolución a todo el espacio de la imagen:

Figura 2.13 Ejemplo de salto (stride)



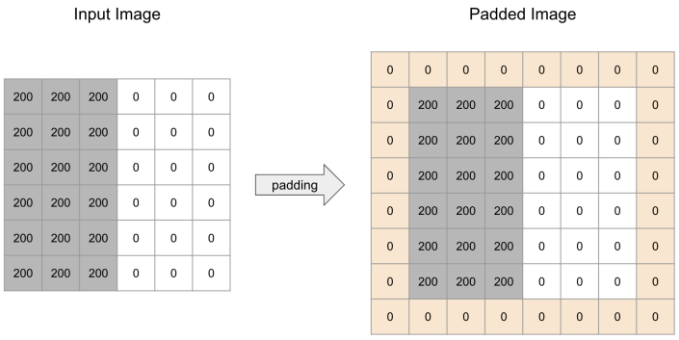
de Mirza Rahim Baig (2020)

En lugar de deslizar sólo 1 píxel, se puede elegir una ventana deslizante más grande, como 2 o 3 píxeles. El parámetro que define el valor de esta ventana deslizante se llama *stride*. Con un valor de *stride* mayor, habrá menos píxeles superpuestos, pero el mapa de características resultante tendrá dimensiones más pequeñas, por lo que perderá un poco de información.

2.2.2.6 Relleno (*Padding*)

Aplicar una convolución a una imagen dará como resultado un mapa de características que tiene dimensiones más pequeñas que la imagen de entrada. Se puede utilizar una técnica llamada relleno para obtener exactamente las mismas dimensiones para el mapa de características que para la imagen de entrada. Consiste en añadir una capa de píxeles con valor 0 al borde (Mirza Rahim Baig, 2020).

Figura 2.14 Ejemplo de relleno



De Mirza Rahim Baig (2020)

2.2.2.7 Redes Neuronales Convolucionales (CNN).

Las CNN son muy similares a las redes neuronales tradicionales, pero en lugar de utilizar capas completamente conectadas, utilizan capas convolucionales. Cada capa de convolución tendrá un número definido de filtros (o núcleos) que aplicarán la operación de convolución con un salto (*stride*) determinado en una imagen de entrada con o sin relleno (*padding*) y puede ir seguida de una función de activación (Mirza Rahim Baig, 2020).

Las CNN se utilizan ampliamente para la clasificación de imágenes, donde la red tendrá que predecir la clase correcta para una entrada determinada. Esto es exactamente lo mismo que los problemas de clasificación de los algoritmos tradicionales de aprendizaje automático. Si el resultado solo puede ser de dos clases diferentes, será una clasificación binaria, como reconocer perros versus gatos. Si el resultado puede ser más de dos clases, será un ejercicio de clasificación de clases múltiples, como reconocer 20 tipos diferentes de frutas (Mirza Rahim Baig, 2020).

Para hacer tales predicciones, la última capa de un modelo CNN debe ser una capa completamente conectada con la función de activación relevante según el tipo de problema de predicción. Se puede utilizar la lista de funciones de activación de la Tabla 2.3. como regla general:

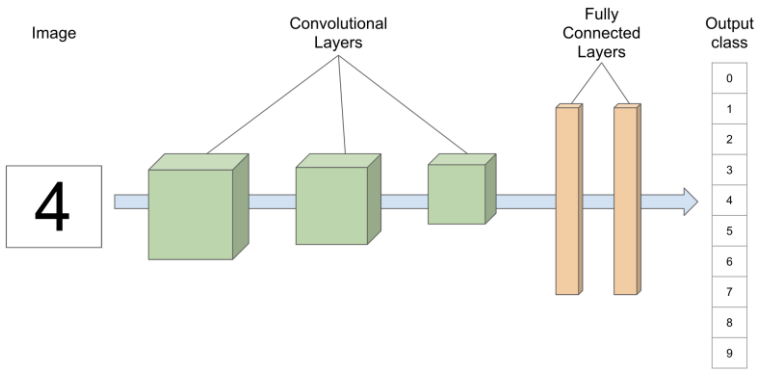
Tabla 2.3 Lista de funciones de activación(Mirza Rahim Baig, 2020)

Tipo de problema	**Función de activación para la última capa**	**Salida**
Clasificación Binaria	Sigmoid	Un número con valores desde 0 hasta 1 correspondiente a la probabilidad de la observación
Clasificación multiclase	Softmax	Múltiples valores numéricos (dependiendo del nro. de clases) desde 0 hasta 1 correspondientes a la probabilidad de cada clase

De Mirza Rahim Baig (2020)

Para obtener una mejor perspectiva de su estructura, así es como se ve un modelo CNN simple:

Figura 2.15 Estructura de una CNN simple (Mirza Rahim Baig, 2020)



De Mirza Rahim Baig (2020)

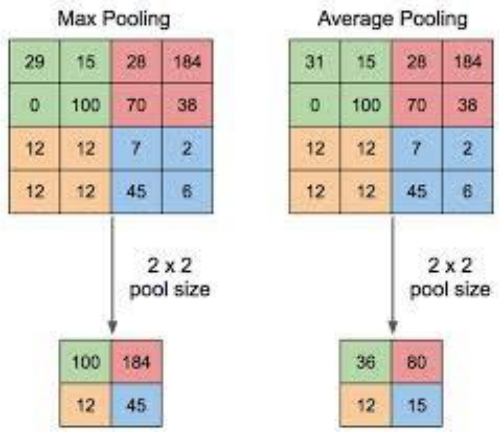
2.2.2.8 Capas de agrupación (*Pooling*)

Las capas de agrupación se utilizan para reducir las dimensiones de los mapas de características de las capas convolucionales. Una de las principales razones es reducir la cantidad de cálculos que se realizan en las redes. Agregar múltiples capas de convolución con diferentes filtros puede tener un impacto significativo en el tiempo de entrenamiento. Además, reducir las dimensiones de los mapas de características puede eliminar parte del ruido en el mapa de características y ayudar a enfocarse solo en el patrón detectado. Es bastante típico agregar una capa de

agrupación después de cada capa convolucional para reducir el tamaño de los mapas de características (Mirza Rahim Baig, 2020).

Una operación de agrupación actúa de manera muy similar a un filtro, pero en lugar de realizar una operación de convolución, utiliza una función de agregación como promedio o máximo (máximo es la función más utilizada en la arquitectura CNN actual). Por ejemplo, la agrupación máxima observará un área específica del mapa de características y encontrará los valores máximos de sus píxeles. Luego, realizará un salto (*stride*) y encontrará el valor máximo entre los píxeles vecinos. Repetirá este proceso hasta procesar la imagen completa:

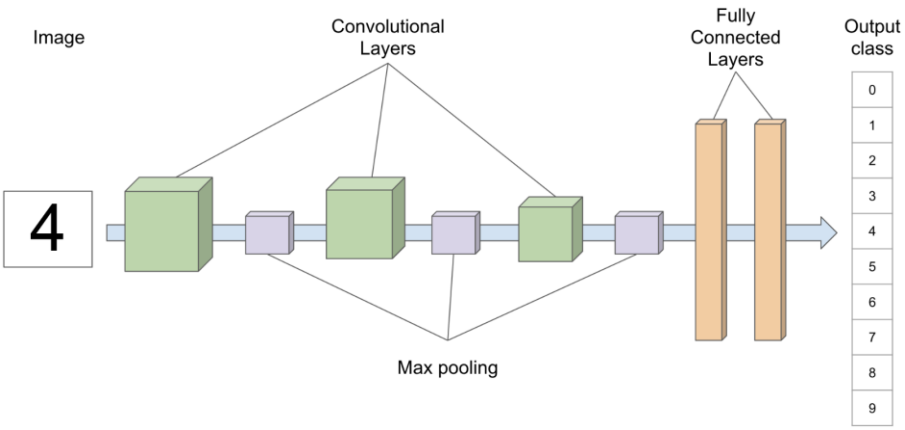
Figura 2.16 Max pooling y Average pooling



De Clement (2020)

Un modelo de CNN con agrupación máxima se ve como en la Figura 2.17.

Figura 2.17 Arquitectura de una CNN con agrupación máxima



De Clement (2020)

2.2.2.9 Capas completamente conectadas

Una capa completamente conectada compuesta por *m* neuronas toma como entrada un vector (Clement, 2020).

$$x = \begin{bmatrix} x_1 \\ \dots \\ x_{n-1} \\ 1 \end{bmatrix} \in \mathbb{R}^n$$

Devuelve un vector de salida

$$y = \begin{bmatrix} y_1 \\ \dots \\ y_m \end{bmatrix} \in \mathbb{R}^m$$

tal que

$$y = Wx \text{ with } W = \begin{bmatrix} w_{1,1} & \dots & w_{1,n} \\ \dots & \dots & \dots \\ w_{m,1} & \dots & w_{m,n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

Cada una de las *m* filas de *W* corresponde a una neurona y el coeficiente $w_{i,j}$ define la conexión entre la entrada x_j y la salida y_i.

El término "1" en el vector x es una forma de incluir un sesgo.

2.2.2.10 Capa de normalización por lotes (*Batch Normalization*)

Durante el entrenamiento de una red neuronal artificial, se modifican los parámetros de las capas. Como consecuencia, la distribución de la entrada de cada capa cambia durante el entrenamiento, este es el "cambio de

covarianza interna". La normalización por lotes, normaliza los datos de entrada de cada capa para resolver este problema. La normalización por lotes también ayuda a los valores de entrada de las funciones de activación a evitar áreas de saturación de estas funciones (Clement, 2020).

2.2.2.11 Capa de aplanamiento (*Flatten Layer*)

Esta es una capa simple que toma un tensor como entrada y lo aplana para generar un vector. Esta capa generalmente se coloca antes de una capa completamente conectada, ya que las capas completamente conectadas toman vectores como entrada.

2.2.3 Hiperparámetros de una red neuronal convolucional

En una red neuronal convolucional durante el proceso de aprendizaje, se evalúan y mejoran parámetros como pesos o sesgos. Sin embargo, un hiperparámetro es una variable cuyo valor se establece antes del entrenamiento, por lo que no se evalúa ni se corrige. En otras palabras, los hiperparámetros son las variables que determinan la estructura de la red (por ejemplo, el número de unidades ocultas) y las variables que determinan cómo se entrena la red (por ejemplo, la tasa de aprendizaje). Por tanto, los hiperparámetros son muy importantes porque de ellos depende en gran medida el rendimiento de la red neuronal artificial o CNN. La Tabla 2.4 muestra, en general, algunos hiperparámetros que están relacionados con la estructura de la red, y la Tabla 2.5 muestra algunos hiperparámetros que están relacionados con el entrenamiento de la red (Graupe, 2013).

2.2.4 Optimización de Hiperparámetros

En un problema de clasificación típico con una red neuronal artificial, los datos se dividen en un conjunto de entrenamiento X_{train} y un conjunto de validación $X_{validation}$. El objetivo es encontrar los parámetros W que minimicen una función de pérdida $L(W_j X_{validation})$ en el conjunto de validación $X_{validation}$. Los parámetros W se obtienen mediante un algoritmo de aprendizaje A definido por sus hiperparámetros λ y entrenado en el conjunto de entrenamiento X_{train}. Entonces $W = A(X_{train}|\lambda)$ (Clement, 2020).

Tabla 2.4 Hiperparámetros relacionados con la estructura de una red neuronal

Hiperparámetro	**Descripción**
Número de capas y unidades ocultas	Las capas ocultas son las capas entre la capa de entrada y la capa de salida.
Tasa de abandono	El abandono es una técnica de regularización para evitar el sobreajuste (aumentar la precisión de la validación) aumentando así el poder de generalización
Inicialización del peso de la red	Idealmente, puede ser mejor usar diferentes esquemas de inicialización de peso de acuerdo con la función de activación utilizada en cada capa. Se usa una distribución mayormente uniforme.
Número de neuronas en las capas ocultas	Cuanto mayor sea el número de neuronas en una capa mejor será la predicción, pero será más complejo el cálculo
Función de activación	Las funciones de activación se usan para introducir la no linealidad en los modelos, lo que permite que los modelos de aprendizaje profundo aprendan los límites de predicción no lineal. Sigmoid se usa en la capa de salida al hacer predicciones binarias. Softmax se usa en la capa de salida al hacer predicciones de varias clases
Dimensiones del Kernel	Los kernels se utilizan para convolucionar las imágenes y obtener los mapas de características
Número de filtros en las capas convolucionales	El éxito de las redes convolucionales radica precisamente en el número de filtros (o kernels) que se usa en cada capa y que permiten extraer diferentes características de la imagen.

De Graupe (2013)

El modelo parametrizado por *W* se entrena en el conjunto de entrenamiento X_{train} pero se evalúa su desempeño en el conjunto de validación $X_{validation}$. La motivación es obtener un modelo con buena capacidad de generalización, es decir, un modelo que funcione bien con datos invisibles. En realidad, es un problema común en el aprendizaje automático que el modelo aprenda demasiado de los datos de entrenamiento: esto se llama sobreajuste. El fenómeno opuesto es el desajuste, que ocurre cuando el modelo no captura la estructura subyacente de los datos de entrenamiento. Por lo general, el sobreajuste ocurre cuando el modelo contiene demasiados parámetros de los que los datos pueden justificar, mientras que el desajuste ocurre cuando el modelo contiene muy pocos parámetros. Un hiperparámetro incorrecto puede provocar un ajuste insuficiente o excesivo (Clement, 2020)

Tabla 2.5 Hiperparámetros relacionados con el entrenamiento de una red neuronal

Hiperparámetro	**Descripción**
Tasa de Aprendizaje	La tasa de aprendizaje define la rapidez con la que una red actualiza sus parámetros. Una tasa de aprendizaje baja ralentiza el proceso de aprendizaje, pero converge sin problemas. Una mayor tasa de aprendizaje acelera el aprendizaje, pero puede no converger
Impulso	El impulso o momento ayuda a conocer la dirección del siguiente paso con el conocimiento de los pasos anteriores. Ayuda a prevenir oscilaciones.
Nro. de Épocas	El número de épocas es el número de veces que todos los datos de entrenamiento se muestran en la red durante el entrenamiento Aumente el número de épocas hasta que la precisión de la validación comience a disminuir incluso cuando la precisión del entrenamiento aumenta (sobreajuste)
Tamaño del lote	El tamaño del lote es la cantidad de submuestras que se le dan a la red después de lo cual ocurre la actualización de parámetros
Optimizador	Mientras se entrena un modelo de aprendizaje profundo, los pesos y sesgos asociados con cada nodo de una capa se actualizan en cada iteración con el objetivo de minimizar la función de pérdida. Este ajuste de pesos está habilitado mediante algoritmos como el descenso de gradiente estocástico, que también se conocen con el nombre de optimizadores.

. *de Graupe (2013)*

El objetivo de la selección de hiperparámetros λ es encontrar cuál minimiza $L(W_j X_{validation})$.

$$\lambda^* = \arg\min_{\lambda} L(W \mid X_{validation}) = \arg\min L(A(X_{train} \mid \lambda) \mid X_{validation})$$

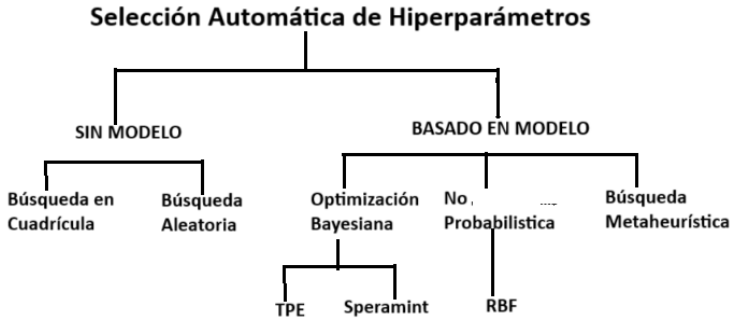
Es importante comprender la diferencia entre los parámetros *W* y los hiperparámetros λ. Los parámetros *W* se modifican durante el entrenamiento mientras que los hiperparámetros son constantes durante el entrenamiento. Entonces, el entrenamiento de una red neuronal artificial es un problema de aprendizaje, mientras que la optimización de hiperparámetros es un problema de meta aprendizaje: el objetivo es aprender a aprender (Clement, 2020).

2.2.5 Soluciones existentes para la optimización de hiperparámetros.

Cuando se trata de optimización de hiperparámetros, todos los enfoques se pueden dividir en *sin modelos* y *basados en modelos*. Las técnicas basadas en modelos construyen un modelo del espacio de hiperparámetros durante su exploración, mientras que los algoritmos de *sin modelo* no utilizan el conocimiento sobre el espacio de solución durante la optimización (Clement, 2020).

La Figura 2.17 muestra diferentes enfoques y sus algoritmos correspondientes: búsqueda en cuadrícula, búsqueda aleatoria, estimador Tree Parzen (denominado TPE), Spearmint, modelo sustituto de funciones de base radial (RBF), Estrategia de evolución de adaptación de la matriz de covarianza (CMA-ES), Optimización por metaheurísticas

Figura 2.18 Enfoques para la selección automatizada de hiperparámetros



de Clement (2020)

2.2.5.1 Algoritmos Sin Modelo

2.2.5.1.1 Búsqueda en Cuadrícula (*Grid Search*)

Quizás el enfoque de fuerza bruta para encontrar el conjunto de hiperparámetros más optimizado sea entrenar el conjunto de datos en cada conjunto posible. Este enfoque, llamado búsqueda en cuadrícula, es la forma más segura de encontrar el mejor conjunto de hiperparámetros, pero también tiene sus desventajas. La Figura 2.18 muestra una cuadrícula que recorre todas las combinaciones posibles de los parámetros 1 y 2 (Tanay, 2021).

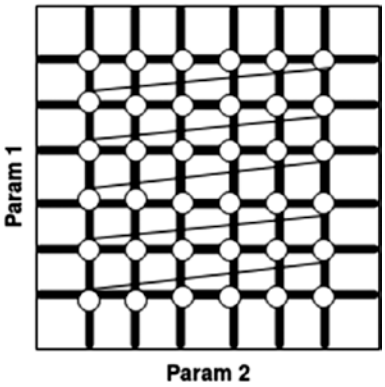


Figura 2.19 Búsqueda en cuadrícula de dos hiperparámetros (Tanay, 2021)

Suponiendo que se tiene diez algoritmos con cinco hiperparámetros cada uno, con cuatro valores posibles para cada hiperparámetro y un conjunto de datos enorme que tarda 1 minuto en entrenarse en un algoritmo en un conjunto de hiperparámetros. Este escenario tardaría alrededor de una semana en descubrir su mejor hiperparámetro. Pero para una menor cantidad de hiperparámetros y un menor tiempo de entrenamiento, se puede optar por la búsqueda en cuadrícula.

El valor de un hiperparámetro puede variar en una distribución continua o discreta. Si el valor es discreto, hay un número finito de valores posibles en un rango. Sin embargo, en una distribución continua, hay infinitos valores posibles,

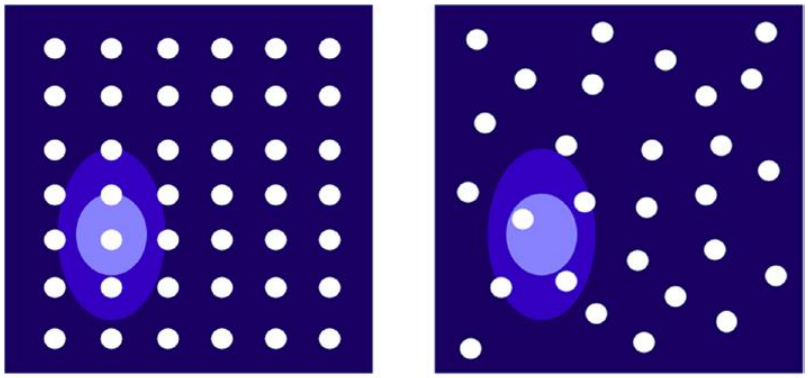
2.2.5.1.2 Búsqueda Aleatoria (*Random Search*)

La búsqueda en cuadrícula finalmente encuentra el conjunto de hiperparámetros casi óptimo, pero su consumo de tiempo y recursos es alto. La búsqueda aleatoria, consume menos tiempo y recursos. Selecciona hiperparámetros al azar, crea un conjunto y entrena el modelo en él. Es posible que este método no encuentre el mejor conjunto, pero hay mayores posibilidades de encontrar un conjunto cercano al mejor, ahorrando una gran cantidad de tiempo (Tanay, 2021).

A diferencia de la búsqueda en cuadrícula, en lugar de dedicar una gran cantidad de tiempo a candidatos poco prometedores, la búsqueda aleatoria salta a hiperparámetros aleatorios y, aunque no aprende de sus resultados anteriores, generalmente ofrece resultados satisfactorios. En la búsqueda aleatoria, se define el número de ensayos, que es el número

de conjuntos de hiperparámetros que se probarán. Se puede ver cómo la búsqueda aleatoria puede ser mejor que la búsqueda en cuadrícula explorando el ejemplo que se muestra en la Figura 2.20.

Figura 2.20 Comparación de la búsqueda en cuadrícula (izquierda) con la búsqueda aleatoria (derecha)



de Tanay (2021)

.

En ambas imágenes de la Figura 2.20, los ejes 'x' e 'y' representan dos hiperparámetros y el fondo representa una precisión cada vez mayor a medida que el color se vuelve más claro. En el caso de la búsqueda en cuadrícula, que se muestra a la izquierda, si comenzamos a buscar desde la esquina superior izquierda, a lo largo de la cuadrícula, la búsqueda tardará una cantidad considerable de tiempo en llegar a la región de mayor precisión. En el caso de la búsqueda aleatoria, que se muestra a la derecha, debido a que busca hiperparámetros aleatoriamente, tiene más posibilidades de alcanzar una mayor precisión antes que con la búsqueda en cuadrícula. Y tan pronto como finalice el número definido de pruebas, selecciona el mejor conjunto de hiperparámetros disponible, con la esperanza de que esté al menos cerca del mejor conjunto (Tanay, 2021).

Los siguientes son dos beneficios principales de utilizar la búsqueda aleatoria en lugar de la búsqueda en cuadrícula.

- El número de ensayos está definido y es independiente del número total de combinaciones.

- Dado que el número de pruebas está definido, incluso si se aumenta el número de parámetros que no contribuyen, la eficiencia temporal del algoritmo no se ve afectada.

2.2.5.2 Algoritmos basados en modelos.

2.2.5.2.1 Optimización bayesiana.

Los enfoques bayesianos, a diferencia de la búsqueda aleatoria o de cuadrícula, realizan un seguimiento de los resultados de evaluaciones anteriores que utilizan para formar un modelo probabilístico que asigna hiperparámetros a una probabilidad de una puntuación en la función objetivo:

$$P = (puntaje \mid hiperparametros)$$

(Ian Dewancker 2015) este modelo se denomina “sustituto” de la función objetivo y se representa como p(y | x). La función sustituta es mucho más fácil de optimizar que la función objetivo y los métodos bayesianos funcionan encontrando el siguiente conjunto de hiperparámetros para evaluar la función objetivo real seleccionando los hiperparámetros que funcionan mejor en la función sustituta. En otras palabras, se siguen los siguientes pasos:

1. Construir un modelo de probabilidad sustituto de la función objetivo.
2. Encontrar los hiperparámetros que funcionan mejor en la función sustituta
3. Aplicar estos hiperparámetros a la verdadera función objetivo.
4. Actualizar el modelo incorporando los nuevos resultados
5. Repetir los pasos 2 a 4 hasta alcanzar el máximo de iteraciones o tiempo

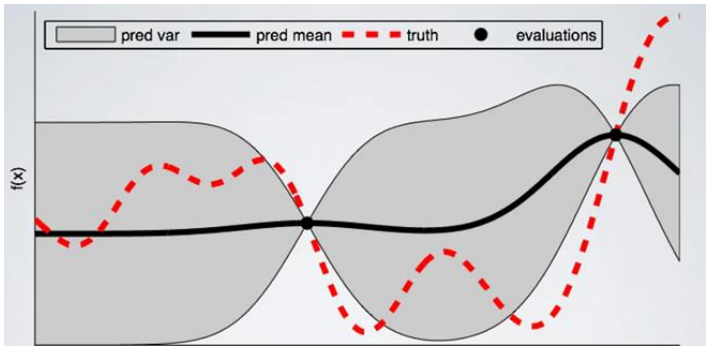
El objetivo del razonamiento bayesiano es volverse "menos equivocado" con más datos, lo que hacen estos enfoques al actualizar continuamente el modelo de probabilidad sustituto después de cada evaluación de la función objetivo.

En un nivel alto, los métodos de optimización bayesianos son eficientes porque eligen los siguientes hiperparámetros de manera informada. La idea básica es: dedicar un poco más de tiempo a seleccionar los siguientes hiperparámetros para realizar menos llamadas a la función objetivo. En la práctica, el tiempo dedicado a seleccionar los siguientes hiperparámetros es intrascendente en comparación con el tiempo dedicado a la función objetivo. Al evaluar los hiperparámetros que parecen más prometedores a partir de resultados anteriores, los métodos bayesianos pueden encontrar mejores configuraciones del modelo que la búsqueda aleatoria en menos iteraciones.

Los métodos basados en modelos bayesianos pueden encontrar mejores hiperparámetros en menos tiempo porque razonan sobre el mejor conjunto de hiperparámetros para evaluar en función de ensayos anteriores.

En las siguientes figuras se puede apreciar lo que ocurre en la optimización bayesiana. La Figura 2.21. muestra una estimación inicial del modelo sustituto (en negro con la incertidumbre asociada en gris) después de dos evaluaciones. Claramente, el modelo sustituto es una mala aproximación de la función objetivo real en rojo.

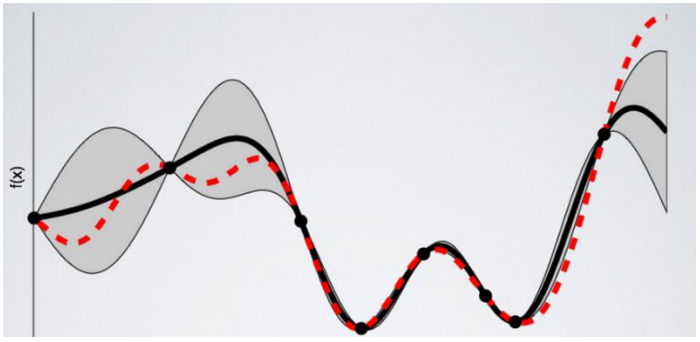
Figura 2.21 Estimación inicial del modelo sustituto (Frazier, 2018)



de Frazier (2018)

En la Figura 2.22 se muestra la función sustituta después de 8 evaluaciones. Ahora el sustituto coincide casi exactamente con la función verdadera. Por lo tanto, si el algoritmo selecciona los hiperparámetros que maximizan el sustituto, probablemente producirán muy buenos resultados en la función de evaluación verdadera.

Figura 2.22 Función sustituta coincide con la función real



de Frazier (2018)

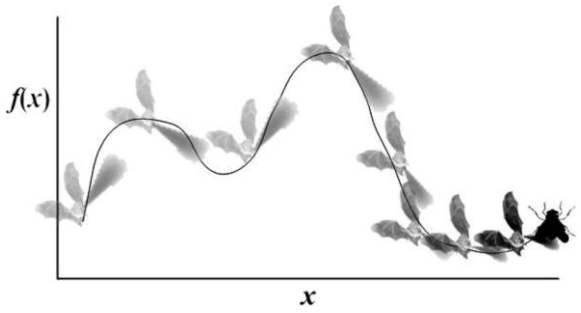
2.2.6 Metaheurística BAT.

2.2.6.1 Origen

En la naturaleza, un murciélago emite naturalmente fuertes pulsos de sonido y escucha los ecos de los objetos circundantes. Los micromurciélagos pueden determinar la táctica de persecución de sus presas dependiendo de los pulsos de sonido enviando señales de frecuencia modulada que van desde 25 KHz a 150 KHz. Cada ráfaga de sonido ultrasónico dura sólo un período corto y los murciélagos pueden enviar entre 10 y 20 ráfagas de sonido por segundo. Sin embargo, la cantidad de pulsos enviados aumenta a aproximadamente 200 pulsos por segundo una vez que el murciélago se acerca a la presa. Aunque un murciélago envía pulsos de carga mientras busca a su presa, el sonido de estos pulsos disminuye a medida que se acerca a su objetivo (Mohammad Shehab, 2022).

La figura 2.23 ilustra la posible trayectoria de un murciélago. Comienza con una posición arbitraria y luego cambia su ubicación dependiendo de los pulsos de sonido emitidos y las tasas de emisión hasta que llega a la ubicación de la presa. Una vez que un murciélago encuentra su presa, el volumen (sonoridad) disminuye y la frecuencia de emisión de pulsos aumenta (Mohammad Shehab, 2022).

Figura 2.23 Trayectoria de un murciélago buscando su presa



de Mohammad Shehab (2022)

2.2.6.2 Algoritmo BAT

El algoritmo BAT fue propuesto por Yang (2010). Se utiliza para imitar la ecolocalización de los murciélagos y se formula como una novedosa técnica de optimización que aplica el comportamiento de un murciélago cuando busca presas y/o evita dificultades al anochecer. En esta sección se introduce la formulación matemática del algoritmo metaheurístico BAT que se inspiró en la vida de los murciélagos en la naturaleza. Vale la pena mencionar que Yang (2010) introdujo seis etapas principales para formular el algoritmo BAT, como se muestra a continuación.

Paso 1: Inicialización de los parámetros del problema y de los murciélagos.

En principio, para evaluar la optimización de la solución x utilizando la función objetivo $f(x)$ de cualquier problema de optimización global, se puede generalizar la siguiente formulación:

$$\min\{f(x) \mid x \in X\}$$

donde $f(x)$ es la función objetivo; $x = \{x_i \mid i = 1,...,d\}$ es el conjunto de variables de decisión; $X = \{X_i \mid i = 1,...,d\}$ es el rango de valores posible para cada variable de decisión, donde $X_i \in [LB_i, UB_i]$, LB_i y UB_i son los límites inferior y superior para la variable de decisión x_i respectivamente, y d es el número de variables de decisión.

Paso 2: Inicializar la población de murciélagos.

La población de murciélagos es una matriz de tamaño $N \times d$ (BM) que contiene N conjuntos de vectores de posición de murciélagos (ver Ecuación (2.9)).

$$BM = \begin{bmatrix} x_1^1 & x_2^1 & \text{L} & x_d^1 \\ x_1^2 & x_2^2 & \text{L} & x_d^2 \\ \text{M} & \text{M} & \text{L} & \text{M} \\ x_1^N & x_2^N & \text{L} & x_d^N \end{bmatrix} \tag{2.9}$$

En este paso, estos vectores de ubicación de murciélagos se generan aleatoriamente de la siguiente manera:

$$x_i^j = LB_i + (UB_i - LB_i) \times \beta(0,1), \quad \forall i = 1,2,...,d \quad , \quad \forall j = 1,2,...,N \tag{2.10}$$

donde $\beta(0,1)$ generan un número aleatorio uniforme entre 0 y 1. Las soluciones generadas se almacenan en la matriz BM en orden ascendente según los valores de su función objetivo, donde $f(x^1) \le f(x^2 \le \text{L} \le f(x^N)$.

En este paso se guarda la mejor ubicación global de murciélagos x^{Gbest}, donde $x^{Gbest} = x^1$.

Paso 3: Movimiento de los murciélagos.

En este paso, cada murciélago x^j vuela con una velocidad v^j afectada por una frecuencia f_j generada aleatoriamente. La nueva posición del murciélago $x^{'j}$ en el espacio de búsqueda se actualiza de la siguiente manera:

$$f_j = f_{\min} + (f_{\min} - f_{\max}) \times \beta(0,1) \tag{2.11}$$

$$v_i^{'j} = v_i^j + (x_i^j - x_i^{Gbest}) \times f_j \tag{2.12}$$

$$x_i^{'j} = x_i^j + v_i^{'j} \tag{2.13}$$

donde $\forall i = 1,2,...,d$ y $\forall j = 1,2,...,N$. La posición de los murciélagos descendientes se actualiza según la ubicación de los murciélagos padres agregada a un valor relativamente pequeño. Este pequeño valor se hereda a medida que la distancia entre los valores en la mejor ubicación global de

murciélagos y el murciélago padre se acerca al murciélago descendiente.

Paso 4: Intensificación de la población actual de murciélagos.

Este paso es la fuente de aleatoriedad controlada en el algoritmo inspirado en murciélagos. Con un rango de probabilidad de frecuencia de pulso r_j, cada nueva ubicación de murciélago se actualiza utilizando una estrategia de búsqueda local con recorrido aleatorio alrededor de las mejores soluciones seleccionadas actualmente. La ubicación histórica del murciélago x^{best} se selecciona inicialmente entre las mejores ubicaciones actuales. A partir de entonces, la nueva ubicación del murciélago $x^{'j}$ se actualiza de la siguiente manera:

$$x_i^{'j} = x_i^{best} + \varepsilon \P_j \quad (2.14)$$

donde $\P_j$ es la sonoridad o volumen medio de todos los murciélagos y ε es un valor aleatorio en el rango [-1,1]. Para resumir: en los pasos 3 y 4, la nueva posición del murciélago $x^{'j}$ se puede calcular como:

$$x^{'j} \leftarrow \begin{cases} x^{best} + \varepsilon \P_j & \beta(0,1) > r_j \\ x^j + v^{'j} & en\ otro\ caso \end{cases} \quad (2.15)$$

Paso 5: Actualización de la población de murciélagos.

En este paso, para cada murciélago en BM, la nueva posición del murciélago reemplaza la posición actual del murciélago según las siguientes condiciones:

1. El valor de la función objetivo de $f(x^{best})$ es mejor que $f(x^{'j})$.
2. El valor de $\beta(0,1)$ es menor que A_J.

El valor de la frecuencia del pulso r_j y el volumen (sonoridad) A_j también se actualizarán de la siguiente manera:

$$r_j = r_j^0(1 - e^{(-\gamma \times itr)}) \quad (2.16)$$

$$A_j = \alpha A_j \quad (2.17)$$

donde itr es el número de generación en el paso de tiempo actual, α y γ son constantes. A medida que el paso de tiempo itr se acerca al infinito, el volumen promedio del murciélago se acerca a cero, mientras que la tasa de emisión de pulsos se acerca a la tasa de emisión de pulsos inicial.

$$A_j^{itr} \to 0, \quad r_j^{itr} \to r_0^j, \quad como\ itr \to \alpha$$

Finalmente, se clasifican los murciélagos y se determina la mejor ubicación actual de los murciélagos x^{Gbest}.

Paso 6: Criterio de parada.

En este paso, el algoritmo inspirado en los murciélagos repite del paso 3 al paso 5 hasta que se cumpla el criterio de terminación. Los criterios de terminación normalmente están relacionados con la restricción de tiempo computacional, el número de generaciones, el número de generaciones detenidas y la calidad de los resultados finales.

El algoritmo de la Figura 2.24 resume la ejecución del algoritmo BAT.

Figura 2.24 Algoritmo metaheurístico BAT (Yang, 2010)

1. *Función objetivo* $f(x), x = (x_1 ... x_d)^T$ *;*
2. *Iniciar la población de murciélagos* $x_i (i = 1, 2, ..., n)$ *y* v_i *;*
3. *Definir la frecuencia del pulso* f_i *en* x_i *;*
4. *Inicializar la frecuencia del pulso* x_i *y la sonoridad* A_i *;*
5. *Mientras (t < # de iteraciones)*
6. *Generar nuevas soluciones estableciendo frecuencia*
7. *y actualizar velocidades y posiciones/soluciones:*
8. *Si* $(rand > r_i)$ *entonces*
9. *Seleccionar la solución;*
10. *Generar una solución local;*
11. *Genera una nueva solución volando aleatoriamente;*
12. *Si* $(rand < A_i)\ and\ (f(x_i) < f(x^{Gbest}))$ *entonces*
13. *Aceptar la solución*
14. *Incrementar* r_i *y reducir* A_i
15. *Encontrar el mejor* x^{Gbest} *;*
16. *Reportar el mejor resultado global*

2.2.7 Antecedentes empíricos de la investigación

2.2.7.1 Optimized Convolutional Neural Networks for Detecting Covid-19 from Chest X-Ray (Deepa S., 2022)

En este trabajo se ha diseñado una arquitectura CNN con hiperparámetros optimizados para clasificar las radiografías de tórax de COVID-19. Los hiperparámetros de la CNN se optimizaron mediante un método híbrido basado en los algoritmos Firefly Algorithm (FA) y Particle Swarm Optimization (PSO) para aumentar el rendimiento del diagnóstico. Este trabajo se centró en optimizar el tamaño del lote, la tasa de aprendizaje, la función de activación y el número de épocas que forman la estructura de la CNN. Se logró una exactitud de 97.34% con el modelo optimizado

2.2.7.2 Hyperparameters Optimization of Deep Convolutional Neural Network for Detecting COVID-19 Using Differential Evolution (Abdelrahman Ezzeldin Nagib, 2022).

En este artículo, se utiliza el algoritmo de Evolución Diferencial para determinar los valores óptimos para los hiperparámetros de las redes neuronales convolucionales para detectar COVID-19 en imágenes de rayos X. El modelo presentado comienza con la preparación de datos, seguido de la optimización de hiperparámetros utilizando el algoritmo metaheurístico de Evolución Diferencial, y finaliza con la fase de aprendizaje y evaluación del desempeño. Los resultados obtenidos del modelo presentado podrían estar logrando una precisión del 100 % para los datos de entrenamiento y del 97.25 % para los datos de pruebas. Los hiperparámetros optimizados fueron: El número de capas convolucionales y completamente conectadas, el número de neuronas en las capas completamente conectadas, el número de épocas de entrenamiento y el tamaño del lote.

2.2.7.3 Hyperparameter Optimization in a Convolutional Neural Network Using Metaheuristic Algorithms (Angel Gaspar, 2021).

En este estudio se comparó el desempeño de la optimización de hiperparámetros en una red neuronal convolucional utilizando los algoritmos metaheurísticos, Colonia de Abejas Artificiales (ABC),

Optimización de Enjambre de partículas (PSO) y Optimización de hormiga león (ALO). El conjunto de datos utilizado fue MNIST. Se realizaron un total de 30 ejecuciones considerando los siguientes índices: promedio, mediana, desviación estándar, mejor valor y el peor valor. Donde el valor de la función objetivo corresponde a la pérdida o error de la red neuronal convolucional. Aunque todos los algoritmos comparados dieron buenos resultados, el rendimiento general de estos algoritmos no es el mismo. La diferencia en el desempeño de los algoritmos metaheurísticos se debe a sus operadores de búsqueda y explotación. El algoritmo que obtuvo mejor resultado para esta tarea fue ALO, debido a lo mencionado anteriormente.

2.2.7.4 Optimal hyperparameter selection of deep learning models for COVID-19 chest X-ray classification (Adeyinka P. Adedigba, 2021).

Este artículo presenta técnicas para diagnosticar COVID-19 a partir de radiografías de tórax (CXR). Utiliza el enfoque de ajuste discriminativo, que asigna dinámicamente diferentes tasas de aprendizaje a cada capa de la red. La tasa de aprendizaje se establece utilizando la política de tasa de aprendizaje cíclica que cambia por iteración. Esta flexibilidad aseguró una rápida convergencia y evitó quedarse estancado en el punto de silla de montar. Además, aborda la alta demanda computacional de los modelos profundos implementando un algoritmo que utiliza el entrenamiento de precisión mixta eficiente en memoria y computacional. A pesar de la disponibilidad de escasos conjuntos de datos, el modelo logró un alto rendimiento y generalización. Obtuvo una precisión de validación del 96,83%, una sensibilidad y una especificidad del 96,26% y 95,54%, respectivamente. Cuando se prueba en un conjunto de datos completamente nuevo, el modelo alcanza una precisión del 97%. Los hiperparámetros que se optimizaron fueron la tasa de aprendizaje y el momento.

2.2.7.5 Hyperparameter Optimization for COVID-19 Chest X-Ray Classification (Ibraheem Hamdi, 2022).

En esta investigación, los autores optimizan los hiperparámetros de una red neuronal convolucional preentrenada DenseNet-121. Para optimizar los hiperparámetros utilizan la herramienta Optuna que es un marco de optimización automática de hiperparámetros que está integrado *en Pytorch Lightning*. Los hiperparámetros considerados son tamaño del lote, tasa de abandono, tasa de aprendizaje y aumento de datos. Se demuestra que es posible desarrollar modelos con una precisión del 83% en clasificación binaria y del 64 % en clases múltiples para detectar infecciones por COVID-19 a partir de radiografías de tórax.

2.2.7.6 Hyperparameter Optimization of Convolutional Neural Networks for Classifying COVID-19 X-ray Images (Podgorelec, 2022).

En esta investigación, se adapta y generaliza el método de clasificación basado en el aprendizaje por transferencia para detectar COVID-19 a partir de imágenes de rayos X y emplea la metaheurística *Grey Wolf Optimizer for Transfer Learning Tuning* para resolver la tarea de configuración de hiperparámetros. Utiliza un conjunto de datos de 1446 imágenes de rayos X, con una precisión general del 84,44%, optimizando los hiperparámetros siguientes: número de neuronas en la capa completamente conectada, la tasa de abandono, la función de optimización y la tasa de aprendizaje.

2.2.7.7 Hyperparameter Optimization for COVID-19 Pneumonia Diagnosis Based on Chest CT (Lacerda, 2021).

El propósito de este trabajo es investigar el uso del algoritmo de optimización Hyperband en el proceso de optimización de una CNN aplicado al diagnóstico de la enfermedad SARS-Cov2 (COVID-19). La prueba se realizó con el marco Optuna y el proceso de optimización tuvo como objetivo optimizar cuatro hiperparámetros: (1) la arquitectura troncal, (2) la cantidad de módulos iniciales de VGG16, (3) la cantidad de neuronas en las capas completamente conectadas y (4) la tasa de aprendizaje. La CNN se entrenó en 2175 imágenes de tomografía computarizada (TC). La

CNN propuesta por el proceso de optimización fue una VGG16 con cinco módulos iniciales, 128 neuronas en las dos capas completamente conectadas y una tasa de aprendizaje de 0,0027. El método propuesto logró una sensibilidad, precisión y exactitud del 97%, 82% y 88%, superando la sensibilidad de las pruebas de reacción en cadena de la polimerasa en tiempo real (RT-PCR) (53–88%) y la precisión de las pruebas. diagnóstico realizado por expertos humanos (72%).

2.2.7.8 Bayesian-based optimized deep learning model to detect COVID-19 patients using chest X-ray image data (Mohamed, 2022).

El propósito de este estudio es clasificar imágenes de radiografías de tórax de COVID-19 l. Se propone un nuevo modelo de red neuronal convolucional (CNN) basado en optimización bayesiana para el reconocimiento de imágenes de rayos X de tórax. El modelo propuesto tiene dos componentes principales. El primero utiliza CNN para extraer y aprender funciones profundas. El segundo componente es un optimizador basado en bayesiano que se utiliza para ajustar los hiperparámetros de CNN de acuerdo con una función objetivo. El conjunto de datos equilibrado utilizado comprende 10.848 imágenes (es decir, 3,616 de COVID-19, 3,616 casos normales y 3,616 de neumonía). Se demuestra que la arquitectura óptima derivada de la búsqueda bayesiana logró una precisión del 96%. Los hiperparámetros que se optimizan son: la tasa de aprendizaje, el momento, profundidad de la capa completamente conectada y la regularización L2.

3. METODOLOGÍA.

3.1 Ámbito de estudio.

El estudio no tiene una acotación geográfica específica por cuanto los datos que se utilizarán provienen de diferentes espacios geográficos del mundo, en consecuencia, los resultados serán válidos en el contexto mundial

3.2 Tipo y nivel de investigación.

Este es un estudio de tipo cuasiexperimental (Pallela S., 2006) debido a que se manipularán variables de manera controlada, en este caso, los hiperparámetros de una red neuronal convolucional para optimizar su arquitectura, utilizando datos que no pueden ser obtenidos al azar para reconocer COVID-19 en imágenes radiológicas de tórax. Y es de nivel aplicado tecnológico porque generará nuevo conocimiento con el propósito de apoyar al diagnóstico de COVID-19.

3.3 Unidad de análisis.

La unidad de análisis son redes neuronales convolucionales en las que se optimizarán sus hiperparámetros para reconocer COVID-19 en imágenes radiológicas.

3.4 Etapas de la investigación.

Con la finalidad de alcanzar los objetivos de este trabajo se realizarán las siguientes actividades:

1. Determinar el conjunto de datos de imágenes de rayos X de tórax a utilizar
2. Diseñar e implementar una red neuronal convolucional para el reconocimiento de COVID-19 en imágenes radiológicas.
3. Aplicar la metaheurística BAT para obtener los hiperparámetros óptimos de la red neuronal convolucional para el reconocimiento de COVID-19 en imágenes radiológicas.
4. Evaluar el desempeño de la red neuronal convolucional con lo hiperparámetros óptimos encontrados por el algoritmo metaheurístico BAT.

4. RESULTADOS Y DISCUSIÓN

4.1 El conjunto de datos de COVID-19.

El conjunto de datos consiste de un total de 284 imágenes. Estas imágenes fueron recopiladas de dos bases de datos diferentes disponibles públicamente: *Kaggle COVID-19 chest X-ray image Dataset* [1] y *Kaggle Covid-19 Chest Radiography Database*[2].

Las 284 imágenes constan de dos clases: “Normal”, y “Covid”. La Tabla 4.1 muestra la distribución de las imágenes entre estas dos clases. En cada clase, las imágenes se dividen con fines de entrenamiento y de validación. Las imágenes de validación representan el 27% de las de entrenamiento. La distribución de datos que se muestra en la Figura 4.1 indica que las imágenes de rayos X disponibles están balanceados

Tabla 4.1 Distribución de los datos

Clases	Train	Valid	Total
Covid-19	112	30	142
Normal	112	30	142
Total	224	60	284

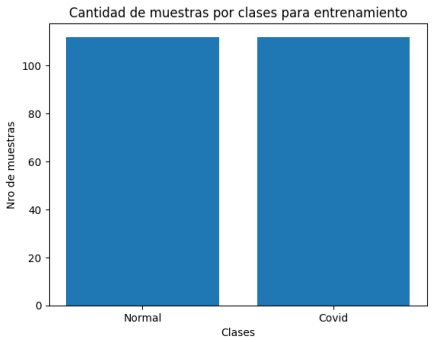


Figura 4.1 Distribución de imágenes de entrenamiento

[1] www.kaggle.com/alifrahman/covid19-chest-xray-imagedataset.
[2] www.kaggle.com/tawsifurrahman/covid19-radiographydatabase

4.2 Preprocesamiento.

Antes de enviar datos al modelo CNN, las imágenes se normalizan para evitar la saturación del modelo y mejorar el proceso de entrenamiento. Con la finalidad de **disminuir el costo computacional**, se redimensionan las imágenes de un tamaño de 1024x1024 a 128x128. Además, las imágenes toman al azar para hacer que los datos sean más versátiles, lo que eventualmente conduce a un entrenamiento genérico y el modelo será más eficiente al momento de generalizar. Asimismo, debido al reducido número de imágenes, se realiza aumento de datos (*data augmentation*) que sigan las mismas características estadísticas de las muestras y así entrenar la red con más datos.

4.3 Arquitectura de la red neuronal convolucional.

Para determinar la arquitectura base de la CNN que se optimizará, se ha consultado los trabajos de Deepa S. (2022). Talha (2021) y Muhammad Talha Nafees (2021). De dichos trabajos se obtiene la arquitectura de la Tabla 4.2 que refleja en gran medida la arquitectura utilizada en los trabajos mencionados. En consecuencia, la arquitectura de la CNN que se optimizará es la que se describe en la Tabla 4.2.

Tabla 4.2 Arquitectura de la red neuronal convolucional base

Nº	**Tipo de capa**	**Representación**
1	Capa convolucional	Conv2D(f1)
2	Capa convolucional	Covn2D(f2)
3	Capa de agrupamiento máx.	MaxPooling(2,2)
4	Capa de abandono	Dropout(d1)
5	Capa convolucional	Conv2D(f3)
6	Capa de agrupamiento máx.	MaxPooling(2,2)
7	Capa de abandono	Dropout(d2)
8	Capa convolucional	Conv2D(f4)
9	Capa de agrupamiento máx.	Maxpooling(2,2)
10	Capa de abandono	Dropout(d2)
11	Capa de aplanamiento	Flatten()
12	Capa densa	Dense(unit1)
13	Capa de abandono	Dropout(0.5)
14	Capa de salida	Dense(1)

4.4 Experimentos

4.4.1 Selección de los hiperparámetros a optimizar.

La calidad y el resultado del entrenamiento de una red neuronal convolucional están condicionados por los diversos valores que definen los hiperparámetros.

Para determinar los hiperparámetros que se optimizarán en esta investigación se toma como referencia aquellos que se consideran en los trabajos de investigación de los antecedentes que se resumen en la Tabla 4.3.

Tabla 4.3. Resumen de hiperparámetros optimizados en antecedentes

Referencia	**Hiperparámetros a optimizar**
Deepa S. (2022)	- Tamaño del lote - Tasa de aprendizaje - Función de activación - Nro. de épocas
Abdelrahman Ezzeldin Nagib (2022)	- Nro. de capas - Nro. de neuronas - Nro. de épocas - Tamaño del lote
Adeyinka P. Adedigba (2021)	- Tasa de aprendizaje - Momentum
Ibraheem Hamdi (2022)	- Tamaño del lote - Tasa de abandono - Tasa de aprendizaje - Aumento de datos
Podgorelec (2022)	- Nro. de neuronas - Tasa de abandono - Función de optimización - Tasa de aprendizaje
Abdelrahman Ezzeldin Nagib (2022)	- Tasa de aprendizaje - Momentum - Nro. de capas - Regularización L2

En todos los trabajos referidos en la Tabla 4.3, se optimizan sólo algunos hiperparámetros que los autores consideran importantes para su investigación, manteniendo constantes los otros hiperparámetros.

Tomando como referencia los datos de la Tabla 4.3, en esta investigación, se analizan los hiperparámetros que consideramos los más importantes y que es posible optimizar mediante el algoritmo metaheurístico BAT. A continuación, se detallan los hiperparámetros seleccionados.

- La tasa de aprendizaje.
- La función de activación.
- El algoritmo de optimización
- El tamaño del filtro (*kernel*).
- El número de filtros en las capas convolucionales.
- La tasa de abandono (*dropout)*.
- El número de neuronas en la capa completamente conectada.

En la tabla 4.4 se muestran los posibles valores de cada hiperparámetro a optimizar.

4.4.2 Parámetros de la red neuronal convolucional.

Para realizar los experimentos se consideran los siguientes datos iniciales:

- **Tamaño y canales de las imágenes de entrada.**
  Las imágenes se han redimensionado a un tamaño de 128 x 128 y tienen 3 canales
- **Tamaño del lote**
  Se ha considerado un lote de tamaño 32 para no saturar la memoria.
- **Número de épocas**
  El número de épocas es de 10

  Plataformas: Anaconda, Google Colaboratory

  Lenguaje de programación Python 3.10.13

  IDE: Visual Studio Code.

4.4.3 Implementación de la metaheurística BAT.

- **Datos del problema**
  - **Función objetivo.**

    La función objetivo $f(x)$ es **minimizar** la pérdida de la red neuronal convolucional utilizando la función *Binary Cross Entropy* debido a que el conjunto de datos tiene solo dos clases, su expresión matemática es la siguiente:

$$-\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{c}[y_i \log(p_i)+(1-y_i)\log(1-p_i)]$$

    donde:

    y : la clase a predecir

    p : probabilidad predicha para la clase

    c : número de clases

    n : número de ejemplos

    Al minimizar la pérdida, implícitamente se maximiza la exactitud.

  - **Rango de valores de los hiperparámetros.**

    Los hiperparámetros que se optimizarán pueden tomar los siguientes valores de acuerdo a la Tabla 4.3

Tabla 4.4. Rango de valores de los hiperparámetros a optimizar

Nº	Hiperparámetro	X	Rango de valores	LB_I	UB_I
1	Nro. de filtros capa conv. 1 (f1)	x_1	[32,64,128]	0	2
2	Nro. de filtros capa conv. 2 (f2)	x_2	[32,64,128]	0	2
3	Nro. de filtros capa conv. 3 (f3)	x_3	[32,64,128]	0	2
4	Nro. de filtros capa conv. 4 (f4)	x_4	[32,64,128]	0	2
5	Tamaño del filtro (k)	x_5	[3,5]	0	1
6	Optimizador (op)	x_6	[Adam, SGD, RMsprop, Adadelta, Adagrad]	0	4
7	Tasa de abandono capa 1 (d1)	x_7	[0.0, 0.2, 0.3, 0.4, 0.5]	0	4
8	Tasa de abandono capa 2 (d2)	x_8	[0.0, 0.2, 0.3, 0.4, 0.5]	0	4
9	Nro. neuronas clasificador (unit1)	x_9	[61,128]	0	1
10	Tasa de aprendizaje (lr)	x_{10}	[0.1, 0.01, 0.001, 0.0001	0	3
11	Función de activación (fa)	x_{11}	[relu, selu, elu]	0	2

Nota: X es variables de decisión, LB_i límite mínimo de valores, UB_i límite máximo de valores

- **Datos de los murciélagos.**

  Los datos con los que se ejecuta el algoritmo metaheurístico BAT para optimizar la arquitectura de la CNN son:

  - Tamaño de la colonia de murciélagos:

    N = 10

  - Número de variables de decisión:

    d = 11

  - Frecuencia mínima:

    $f_{\min} = 0$

  - Frecuencia máxima:

    $f_{\max} = 2$

  - Constante alfa:

    $\alpha = 0.9$

  - Constante gama:

    $\gamma = 0.7$

  - Número de iteraciones: 10

Inicialización de la colonia de murciélagos.

La posición inicial de la colonia de murciélagos (BM) se calcula aleatoriamente mediante la ecuación (2.10):

$$x_i^j = LB_i + (UB_i - LB_i) \times \beta(0,1), \quad \forall i = 1,2,...,d \quad ,\forall j = 1,2,...,N$$

$$BM = \begin{bmatrix} 1 & 2 & \underset{x_3^1}{0} & 2 & L & 2 \\ 0 & 1 & 2 & 1 & L & 0 \\ 1 & 2 & 0 & 1 & L & \underset{x_{11}^3}{1} \\ M & M & M & M & L & M \\ 0 & 1 & 2 & 1 & L & 0 \end{bmatrix}$$

En esta matriz cada fila de variables de decisión representa a un murciélago de la colonia.

El valor $x_3^1 = 0$ significa el valor que tiene índice 0 en el vector de posibles valores de filtros en la tercera capa convolucional (f3) en la Tabla 4.3 el hiperparámetro nro. 3. Gráficamente se tiene:

Nro. de filtros capa conv. 3 (f3) =

0	1	2
32	64	128

Por lo que para calcular la función objetivo el número de filtros en la tercera capa convolucional (f3) será 32 para el murciélago 1.

De la misma manera el valor $x_{11}^3 = 1$, representa el valor que tiene índice 1 en el vector de posibles valores de función de activación (fa) en la Tabla 4.3. el hiperparámetro nro. 11. Gráficamente esto es:

Función de activación (fa) =

0	1	2
relu	selu	elu

De manera que, para calcular la función objetivo se debe utilizar la función de activación ‘selu’ para el murciélago 3

Por consiguiente, los valores iniciales de la función objetivo para cada murciélago de la población son:

$$f(x) = \begin{bmatrix} 0.8 \\ 1.6 \\ 2.1 \\ \text{M} \\ 3.4 \end{bmatrix}$$

El mejor resultado de la función objetivo se almacena en orden ascendente por lo que x^{Gbest} estará en la primera ubicación.

Los valores iniciales de velocidad y frecuencia son:

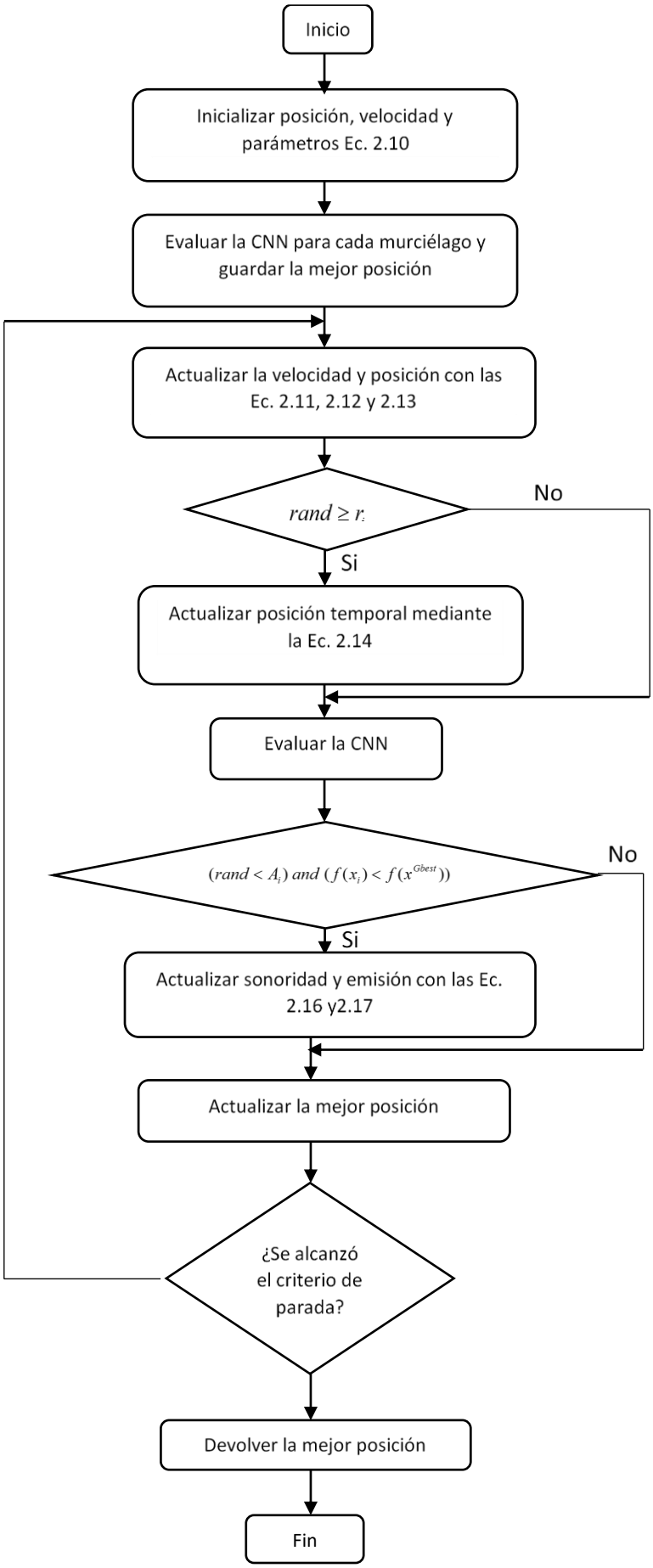
$$v = \begin{bmatrix} 0 & 0 & 0 & 0 & \text{L} & 0 \\ 0 & 0 & 0 & 0 & \text{L} & 0 \\ 0 & 0 & 0 & 0 & \text{L} & 0 \\ \text{M} & \text{M} & \text{M} & \text{M} & \text{L} & \text{M} \\ 0 & 0 & 0 & 0 & \text{L} & 0 \end{bmatrix} \qquad f = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ \text{M} \\ 0.0 \end{bmatrix}$$

El valor inicial de la tasa de pulsaciones (*r*) y la sonoridad (*A*) se calcula mediante las ecuaciones (2.16) y (2.17) respectivamente;

$$r_j = r_j^0(1-e^{(-\gamma \times itr)}) \Rightarrow \quad r = \begin{bmatrix} 0 \\ 1 \\ -1 \\ \text{M} \\ 1 \end{bmatrix}$$

$$A_j = \alpha A_j \Rightarrow A = \begin{bmatrix} 1.0 \\ 1.5 \\ 1.8 \\ \text{M} \\ 1.4 \end{bmatrix}$$

El diagrama de flujo del algoritmo BAT para optimizar la CNN es el siguiente:


Inicio
Inicializar posición, velocidad y parámetros Ec. 2.10
Evaluar la CNN para cada murciélago y guardar la mejor posición
Actualizar la velocidad y posición con las Ec. 2.11, 2.12 y 2.13
rand ≥ r
No
Si
Actualizar posición temporal mediante la Ec. 2.14
Evaluar la CNN
(rand < A_i) and (f(x_i) < f(x^Gbest))
No
Si
Actualizar sonoridad y emisión con las Ec. 2.16 y2.17
Actualizar la mejor posición
¿Se alcanzó el criterio de parada?
Devolver la mejor posición
Fin

4.4.4 Hardware y software

- **Hardware**

  Laptop HP ZBook

  Procesador CPU: Intel Xeon E-2176M, 12 núcleos

  Procesador GPU: Nvidia Quadro P1000, 640 núcleos

  Memoria: 16 GB

- **Software**

  Sistema Operativo: Windows 11.

  Lenguaje de programación: Python versión 3.10.13

  Librerías: Tensorflow GPU, Keras, Sklearn

4.4.5 Hiperparámetros óptimos.

Luego de la ejecución del algoritmo metaheurístico BAT, cuya implementación se puede ver en el Anexo 02, este proporcionó como resultado los hiperparámetros óptimos de la Tabla 4.5.

Tabla 4.5 Valores óptimos de los hiperparámetros encontrados por BAT

Hiperparámetro	**Símbolo**	**Valor óptimo**
Nro. de filtros capa conv. 1	f1	32
Nro. de filtros capa conv. 2	f2	32
Nro. de filtros capa conv. 3	f3	128
Nro de filtros capa conv. 4	f4	128
Tamaño del kernel	k	3
Tasa de abandono capa drop. 1	d1	0.2
Tasa de abandono capa drop. 2	d2	0.2
Tasa de abandono capa drop. 3	d3	0.0
Nro. de neuronas capa densa 1	uni1	64
Función de activación	fa	relu
Optimizador	op	Adam

4.4.6 Arquitectura óptima de la CNN

Con los hiperparámetros optimizados de la Tabla 4.5 se construye la arquitectura óptima de la CNN que se muestra en la Figura 4.2

Figura 4.2. Arquitectura óptima de la CNN

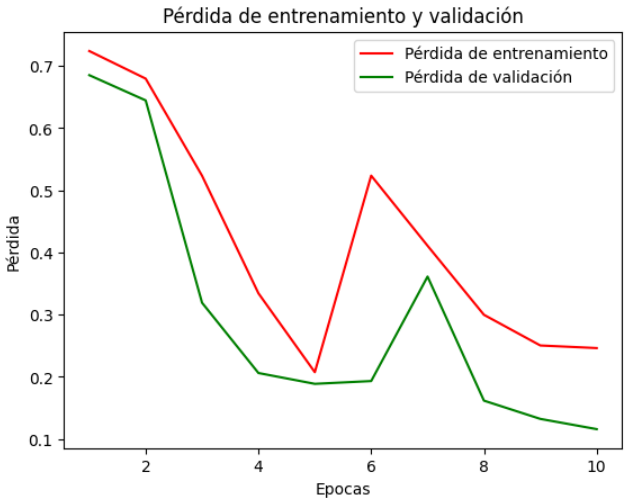
```
model = Sequential()
model.add(Conv2D(32,kernel_size=(3,3),activation=
         'relu',input_shape=(img_fil,img_col,3)))
model.add(Conv2D(32,(k,k),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(128,(k,k),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.0))

model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))
```

4.4.7 Resultados del modelo óptimo.

- **Pérdida de entrenamiento y validación**

  La Figura 4.3 muestra el gráfico de pérdida durante el entrenamiento del modelo óptimo para datos de entrenamiento y validación durante 10 épocas.

Figura 4.3 Resultados de pérdida del modelo óptimo



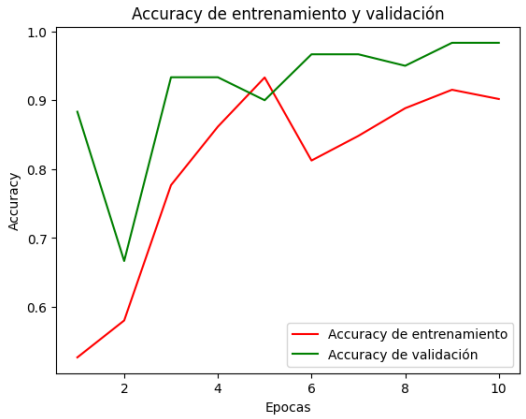
Análisis

Las curvas de la Figura 4.3 indican que el entrenamiento del modelo avanza sin sobreajuste. Se puede notar que en la iteración 10 las curvas muestran todavía una ligera pendiente negativa, y, por lo tanto, el modelo se podría entrenar todavía más. En consecuencia, los hiperparámetros optimizados a través del algoritmo metaheurístico BAT demuestran su validez.

- **Exactitud (accuracy) de entrenamiento y validación.**

En la Figura 4.4 se puede apreciar la exactitud del modelo óptimo para datos de entrenamiento y validación.

Figura 4.4 Exactitud de entrenamiento y validación



Análisis

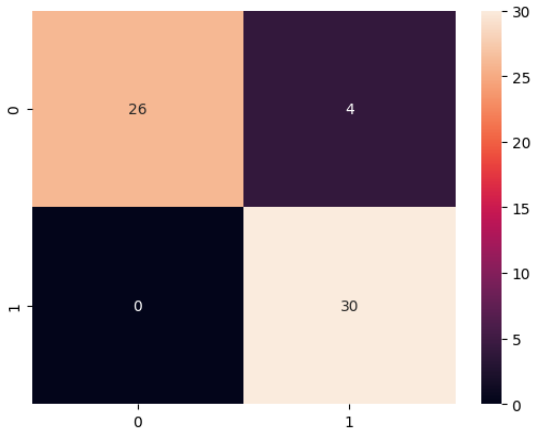
Las curvas de la Figura 4.4. muestran claramente que el modelo óptimo no presenta sobreajuste, además, estas curvas muestran una exactitud del 98.3% para datos de validación. El detalle de las precisiones de cada clase se proporciona en la Tabla 4.5. También es pertinente mencionar que la exactitud para datos de validación comienza en alrededor de 0.8, que es un valor bastante alto debido a un tamaño de lote pequeño de 32 y una tasa de aprendizaje de 0.001.

- **Matriz de confusión**

La matriz de confusión es la columna vertebral de los indicadores de rendimiento de la clasificación, ya sea en clasificadores binarios o clasificadores multiclase. Tiene cuatro parámetros fundamentales, es

decir, verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN). En la Figura 4.6. se muestra la matriz de confusión del modelo estudiado.

Figura 4.5 Matriz de confusión del modelo óptimo



En la matriz de confusión, la etiqueta prevista está en el eje horizontal, mientras que el eje vertical muestra la etiqueta verdadera.

Aquí, para la clase COVID-19 (clase 0), los verdaderos positivos (TP) son 26 casos y se pronostican también 26 verdaderos positivos. Los falsos positivos (FP), son los casos que no son de COVID-19 pronosticados como COVID-19, en este caso es 0. Los verdaderos negativos (TN) son los casos que no son COVID-19 y se pronostican como que no son efectivamente casos de COVID-19, en este caso son en número de 30 y los falsos negativos (FN) es el número de pacientes con COVID-19 pronosticados como que no tiene COVID-19, es decir 4. El detalle completo de la matriz de confusión para cada clase se proporciona en la Tabla 4.6.

Tabla 4.6 Detalle de la matriz de confusión

Clase	**TP**	**TN**	**FP**	**FN**	**Precis.**	**Sensib.**	**Especif.**	**Score F1**	**Acc**
Covid	26	30	0	4	1.00	0.87	1.00	0.93	0.93
Normal	30	26	4	0	0.88	1.00	0.87	0.94	0.93

Análisis

La evaluación del desempeño del modelo optimizado se realiza utilizando las métricas precisión, sensibilidad, especificidad y exactitud. Para todas las clases, estos parámetros se calculan en la Tabla 4.6. El modelo propuesto es muy preciso para la clase COVID-19 con una precisión del 100%, lo que significa que el modelo tiene un rendimiento excelente para los casos de COVID-19. Además, el modelo rara vez clasifica a pacientes normales como positivos para COVID-19. Esto también lo indica la especificidad que es también del 100%. Ambos parámetros dependen de FP, que es 0 para la clase Covid, como se muestra en la Tabla 4.6. Aparte de esto, la sensibilidad del modelo es un poco baja, es decir, 87%, lo que significa que el modelo clasifica el COVID-19 como “No covid” debido al aumento de datos utilizado con fines de entrenamiento.

4.4.8 Implementación de Optimización Bayesiana de los hiperparámetros de la CNN.

Con la finalidad de lograr los objetivos de esta investigación, se ha implementado la optimización bayesiana de hiperparámetros de la red neuronal convolucional (ver Anexo 04) para detectar COVID-19 en imágenes de rayos X de tórax cuyos resultados se pueden apreciar en la Tabla 4.7.

Tabla 4.7. Resultados de la Optimización Bayesiana de la CNN

Iter.	target	filtros	Func_act	kernels	neuronas	optimiz	Tasa_apr	Tasa_drop
1	0.5	72.0	1.441	3.0	83.35	0.587	0.0101	0.1745
2	0.5	65.1	0.793	4.0	90.83	2-741	0.0212	0.4512
3	0.5	34.6	1.341	3.8	99.76	0.561	0.0206	0.4203
4	0.5	127.6	0.157	3.6	127.50	1.707	0.0825	0.2152
5	0.5	127.5	1.278	4.5	64.73	0.407	0.0495	0.3231
6	0.5	32.1	1.345	4.9	127.70	1.542	0.0774	0.3615
7	0.5	32.0	0.748	4.6	64.76	1.730	0.0717	0.4156
8	**0.85**	**127.9**	**0.971**	**4.7**	**126.60**	**3.517**	**0.0300**	**0.2038**
9	0.5	100.8	1.303	3.2	69.57	2.621	0.0833	0.1807

La optimización bayesiana permite obtener una exactitud de 85% en la iteración 8 para los valores de los hiperparámetros correspondientes en la Tabla 4.3 y que se pueden ver en la Tabla 4.8.

Tabla 4.8. Valores de los hiperparámetros con optimización bayesiana

Hiperparámetros	**Valores (optimización bayesiana)**
Número de filtros	128
Función de activación	Selu
Tamaño del kernel	5x5
Número de neuronas	126
Optimizador	Adagrad
Tasa de aprendizaje	0.03006
Tasa de abandono	0.2

4.4.9 Discusión de resultados.

Para determinar en qué medida el algoritmo metaheurístico BAT optimiza la arquitectura de una CNN para detectar COVID-19 en imágenes de rayos X de tórax se construye la Tabla 4.9.

Tabla 4.9. Exactitud (accuracy) obtenida en la investigación comparada con otros resultados.

Referencia	**Algoritmo**	**Exactitud**
Deepa S. (2022)	Metaheurística Firefly – PSO	97.34%
Abdelrahman Ezzeldin Nagib (2022)	Metaheurística Evolución Diferencial	97.25%
Adeyinka P. Adedigba (2021)	Ajuste discriminativo	96.83%
Ibraheem Hamdi (2022)	Optuna	83%
Podgorelec (2022)	Metaheurística Grey Wolf Optimizer for Transfer Learning	84.4%
Abdelrahman Ezzeldin Nagib (2022)	Optimización Bayesiana	96%
Implementado por el autor	Optimización Bayesiana	85%
Implementado por el autor	**Metaheurística BAT**	**98.3%**

Los resultados mostrados en la Tabla 4.9 demuestran que el algoritmo metaheurístico BAT obtiene la mejor exactitud (accuracy) general para clasificar imágenes de rayos X de tórax para detectar COVID-19 en relación a otras técnicas utilizadas para el mismo propósito.

Con respecto a la eficiencia en tiempo de ejecución del algoritmo metaheurístico BAT para optimizar los hiperparámetros de la CNN en la Tabla 4.10 se muestran los tiempos alcanzados por la optimización bayesiana implementada por el autor y el algoritmo metaheurístico BAT. En esta tabla no se consideran los tiempos de ejecución de los algoritmos de optimización de los trabajos de referencia debido a que no se dispone de esa información.

Tabla 4.10. Tiempo de ejecución de algoritmos de optimización

Algoritmo	**Tiempo promedio**	**Exactitud general**
Optimización Bayesiana	00:11:35	85%
Metaheurística BAT	02:43:20	98.3%

Como se puede apreciar en la Tabla 4.10, el tiempo promedio utilizado por el algoritmo metaheurístico BAT es considerablemente superior a la optimización bayesiana debido a que esta metaheurística optimiza los hiperparámetros haciendo una búsqueda local mediante la cual en cada iteración calcula la nueva ubicación de la colonia de murciélagos lo que implica entrenar la CNN con los hiperparámetros que cada vez son más cercanos al óptimo.

Es necesario indicar que este tiempo es obtenido ejecutando el algoritmo sobre una plataforma GPU, de lo contrario el tiempo de ejecución sería muy extenso. El tiempo de ejecución del algoritmo BAT es también afectado por la cantidad de imágenes en el conjunto de datos utilizado, por este motivo en esta investigación se ha considerado solo 284 imágenes sobre los que necesariamente se ha tenido que realizar aumento de datos.

En cuanto a la eficiencia del uso de memoria, ha sido necesario disminuir el tamaño de las imágenes a 128x128 pixeles debido a que con tamaños más

grandes la memoria del equipo utilizado resulta insuficiente. Debido a esto también, el tamaño del lote se ha mantenido en 32.

5. CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. El algoritmo metaheurístico BAT ha permitido optimizar la arquitectura de una red neuronal convolucional para detectar COVID-19 en imágenes de rayos X de tórax, alcanzando una exactitud general de 98.3% para datos de validación como se muestra en la Tabla 4.9, mejorando aquellas obtenidas en los trabajos de investigación de los antecedentes. Incluso ha demostrado ser superior al algoritmo de optimización bayesiana que solo alcanzó un 85% de exactitud.
2. No existe un estudio que indique específicamente qué hiperparámetros considerar para la optimización de la arquitectura de una CNN. En consecuencia, para determinar los hiperparámetros a optimizar en esta investigación, se ha recopilado información en la Tabla 4.3 de los hiperparámetros que se optimizan en las investigaciones de los antecedentes y que sirvieron como base en la selección de aquéllos que optimicen la arquitectura de la CNN de este trabajo. Los hiperparámetros seleccionados se muestran en la Tabla 4.4.
3. El conjunto de datos utilizado contiene 284 imágenes correspondientes a las clases ‘Covid’ y ‘No Covid’. Al tener este conjunto de datos dos clases, la función de aptitud utilizada para la optimización de la arquitectura de la CNN es la función *Binary Cross Entropy* que permite calcular la pérdida o el error de la CNN. Por consiguiente, el algoritmo metaheurístico BAT minimiza el valor de esta función de aptitud.
4. Al ser la optimización de hiperparámetros de una red neuronal convolucional un problema de optimización combinatoria muy complejo, la eficiencia en tiempo del algoritmo metaheurístico BAT para realizar esta tarea depende de factores tales como: la cantidad de datos de entrenamiento, la arquitectura de la CNN, la cantidad de hiperparámetros a optimizar entre otros. En esta investigación se ha optimizado 11 hiperparámetros para una CNN de 14 capas, por lo que el tiempo de cálculo es considerable aun utilizando las GPU.
5. La sensibilidad de modelo mostrada en la Tabla 4.6 resulta un poco baja porque es posible que las imágenes contengan mucho ruido, o el conjunto de datos puede contener imágenes clasificadas como COVID-19 falsamente. Realizando un mejor

preprocesamiento de los datos o utilizando datos de mejor calidad se puede mejorar la sensibilidad.

RECOMENDACIONES

1. En esta investigación se ha optimizado 11 hiperparámetros (pero existen otros que influyen en menor o mayor grado al desempeño de una CNN, por lo que, se recomienda optimizar otros hiperparámetros tales como el tamaño del lote, el número de capas convolucionales y densas y evaluar el rendimiento de la CNN.
2. Explorar otros algoritmos metaheurísticos para optimizar la arquitectura de Redes Neuronales Convolucionales.
3. Debido a que el algoritmo metaheurístico BAT utiliza un tiempo considerable para optimizar una CNN, considerar la paralelización de este algoritmo para mejorar su eficiencia en tiempo.
4. Mejorar la sensibilidad del modelo considerando la conversión de las imágenes a un solo canal y disminuir el ruido existente.

BIBLIOGRAFIA

Abdelrahman Ezzeldin Nagib, M. M. S., Shereen Fathy El-Feky, Ali Khater Mohamed. (2022). Hyperparameters Optimization of Deep Convolutional Neural Network for Detecting COVID-19 Using Differential Evolution. https://doi.org/10.1007/978-3-030-87019-5_18

Adeyinka P. Adedigba, S. A. A., Oluwatomisin E. Aina, Abiodun M. Aibinu. (2021). Optimal hyperparameter selection of deep learning models for COVID-19 chest X-ray classification. *Intelligence-Based Medicine*.

Angel Gaspar, D. O., Erik Cuevas, Daniel Zaldívar, Marco Pérez, Gonzalo Pajares. (2021). Hyperparameter Optimization in a Convolutional Neural Network Using Metaheuristic Algorithms. *Springer Nature Switzerland*.

Bastien Chopard, M. T. (2018). *An Introduction to Metaheuristics for Optimization*. Springer.

Bengio, Y. (2009). *Learning Deep Architectures for AI*. Foundations and Trends in Machine Learning.

Chong Edwin, Z. S. (2013). *An Introuction for Optimization* (Fourth Edition ed.). John Wiley & Sons.

Clement, G. (2020). *Hyperparameter Optimization for Convolutional Neural Networks* KTH Royal Institute of Technology]. Stockholm Sweden.

Deepa S., S. S. (2022). Optimized Convolutional Neural Networks for Detecting Covid-19 from Chest X-Ray. *International Journal of Engineering Trends and Technology*, *Volume 70*, *8*(Issue 12), 210-221.

Dmytro Mishkina, N. S., Jiri Matasa. (2016). Systematic evaluation of CNN advances on the ImageNet. *Neural and Evolutionary Computing*.

E. Martínez Chamorro, A. D. T., L. Ibá˜nez Sanz, S. Ossaba Vélez y S. Borruel Nacenta. (2021). Diagnóstico radiológico del paciente con COVID-19. *Radiología*, *Jan-Feb;63*, 56-73. https://doi.org/10.1016/j.rx.2020.11.001

F. Wu, S. Z., B. Yu, Y.-M. Chen, W. Wang, Z.-G. Song, Y. Hu, Z.-W. Tao, J.- H. Tian, Y.-Y. Pei, et al.,. (2020). A new coronavirus associated with human respiratory disease in China. *Nature*, *579*, 265 -269.

Frazier, P. I. (2018). A tutorial on bayesian optimization.

Graupe, D. (2013). *Principles of Artificial Neural Networks* (A. S. i. C. a. Systems, Ed. 3rd Edition ed., Vol. Vol 7.). World Scientific Publishing Co. Pte. Ltd.

Ibraheem Hamdi, M. R., Mohammad Yaqub. (2022). Hyperparameter Optimization for COVID-19 Chest X-Ray Classification. *arXiv*, *2201*(10885).

Josh Patterson, A. G. (2017). *Deep Learning A Practitioner's Approach* (First Edition ed.). 0'REILLY.

Lacerda, P. B., B.; Albuquerque, C.; Conci, A. (2021). Hyperparameter Optimization for COVID-19 Pneumonia Diagnosis Based on Chest CT. *Sensors 2021*.

Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R.,Mechelli, A.,. (2020). Convolutional neural networks, in Machine Learning.

Lorenzo, P. R., Nalepa, J., Kawulok, M., Ramos, L.S., Pastor, J.R. (2017). Particle swarm optimization for hyper-parameter selection in deep neural networks,. Genetic and Evolutionary Computation Conference,

Mirza Rahim Baig, T. V. J., Nipun Sadvilkar, Mohan Kumar Silaparasetty, and Anthony So. (2020). *The Deep Learning Workshop*.

Mohamed, L. S. E.-S. S. M. (2022). Bayesian-based optimized deep learning model to detect COVID-19 patients using chest X-ray image data. *Computers in Biology and Medicine*.

Mohammad Shehab, e. a. (2022). A Comprehensive Review of Bat Inspired Algorithm: Variants, Applications, and Hybridization. *Archives of Computational Methods in Engineering (2023)*.

Muhammad Talha Nafees, I., Muhammad Rizwan, Maazullah, Muhammad Irfanullah Khan, and Muhammad Farhan. (2021). A Novel Convolutional Neural Network for COVID-19 detection and classification using Chest X-Ray images. *mesRxvid*. https://doi.org/https://doi.org/10.1101/2021.08.11.21261946;

Pallela S., M. F. (2006). *Metodología de la Investigación Cuantitativa* (2da. Edición ed.). FEDUPEL.

Podgorelec, G. V. S. P. V. (2022). Hyper-parameter Optimization of Convolutional Neural Networks for Classifying COVID-19 X-ray Images. *Computer Science and Information Systems*.

Ragav Venkatesan, B. L. (2018). *Convolutional Neural Networks in Visual Computing*. Taylor & Francis Group, LLC.

Shuai Wang, B. K., Jinlu Ma, Xianjun Zeng, Mingming Xiao, Jia Guo, Mengjiao Cai, Jingyi Yang, Yaodong Li, Xiangfei Meng, Bo Xu. (2020). A deep learning algorithm using CT images to screen for Corona Virus Disease (COVID-19). *Nature Publishing Group*.

Singhal, T. (2020). A Review of Coronavirus Disease-2019 (COVID-19). *Indian Journal of Pediatrics, 87*, 281–286.

Talha, M. (2021). A Novel Convolutional Neural Network for COVID-19 detection and classification using Chest X-Ray images. *medRxiv*.

Tanay, A. (2021). *Hyperparameter Optimization in Machine Learning*. https://doi.org/https://doi.org/10.1007/978-1-4842-6579-6

Tansel Dokeroglu, E. S., Tayfun Kucukyilmaz,, Ahmet Cosar. (2019). A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, 137.

Yang, X.-S. (2010). A new metaheuristic bat-inspired algorithm. *Nature Inspired Cooperative Strategies for Optimization*, 65–74.

Yann Lecun, e. a. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceeedings of the IEEE, 86, No. 11*.

Yepes Piqueras, V. (2002). *Optimización heurística económica aplicada a las redes de transporte del tipo VRPTW* Universitat Politècnica de València].

Zhou Kevin, R. D., Fichtinger Gabor. (2020). *Handbook of Medical Image Computing and Computer Assisted Intervention*.

ANEXOS

Anexo 01: Matriz de consistencia

PROBLEMAS	OBJETIVOS	HIPÓTESIS	VARIABLES	INDICADORES
Problema General ¿En qué medida el algoritmo metaheurístico BAT permite optimizar una Red Neuronal Convolucional para detectar COVID-19 en imágenes de rayos X de tórax?	**Objetivo General** Determinar en qué medida el algoritmo metaheurístico BAT permite optimizar una Red Neuronal Convolucional para detectar COVID-19 en imágenes de rayos X de tórax.	No aplica	No aplica	No aplica
Problema específico 1 ¿Qué características debe tener el algoritmo metaheurístico BAT para optimizar una Red Neuronal Convolucional para detectar COVID-19 en imágenes de rayos X de tórax?	**Objetivo específico 1** Determinar qué características debe tener el algoritmo metaheurístico BAT para optimizar una Red Neuronal Convolucional para detectar COVID-19 en imágenes de rayos X de tórax.	No aplica	No aplica	No aplica
Problema específico 2 ¿En qué medida se optimiza la exactitud de una Red Neuronal	**Objetivos Específico 2** Determinar en qué medida se optimiza la exactitud de una Red	No aplica	No aplica	No aplica

Convolucional para detectar COVID-19 en imágenes de rayos X de tórax, aplicando el metaheurístico BAT?	Neuronal Convolucional para detectar COVID-19 en imágenes de rayos X de tórax, aplicando el metaheurístico BAT.			
Problema específico 3 ¿Cuál es la eficiencia en términos de tiempo de ejecución del algoritmo metaheurístico BAT al optimizar una Red Neuronal Convolucional para detectar COVID-19 en imágenes de rayos X de tórax?	**Objetivo específico 3** Determinar cuál es la eficiencia en términos de tiempo de ejecución del algoritmo metaheurístico BAT al optimizar una Red Neuronal Convolucional para detectar COVID-19 en imágenes de rayos X de tórax.			

Anexo 02: Implementación del algoritmo metaheurístico BAT

```
# Inicializa posiciones de la colonia inicial de murcielagos
def posicion_inicial(tam_colonia = 3, valores_min = [0,0,0,0,0,0,0,0,0,0,0,0],
                                      valores_max = [2,2,2,2,1,5,5,5,1,3,2,4],
                                      funcion_objetivo = funcion_objetivo):
    posicion  = np.zeros((tam_colonia, len(valores_min) + 1))
    velocidad  = np.zeros((tam_colonia, len(valores_min)))
    frecuencia = np.zeros((tam_colonia, 1))
    tasa      = np.zeros((tam_colonia, 1))
    sonoridad  = np.zeros((tam_colonia, 1))
    for i in range(0, tam_colonia):
        for j in range(0, len(valores_min)):
             posicion[i,j] = random.uniform(valores_min[j], valores_max[j])
        posicion[i, -1] = funcion_objetivo(posicion[i,0:posicion.shape[1]-1])
        tasa[i, 0]      = int.from_bytes(os.urandom(8), byteorder = "big") /
((1 << 64) - 1)
        sonoridad[i, 0]  = random.uniform(1, 2)
    return posicion, velocidad, frecuencia, tasa, sonoridad
```

```
def algoritmo_bat(tam_colonia = 3, valores_min = [0,0,0,0,0,0,0,0,0,0,0,0],
valores_max = [2,2,2,2,1,5,5,5,1,3,2,4], iteraciones = 50, alfa = 0.9, gama =
0.9, fmin = 0, fmax = 10, funcion_objetivo = funcion_objetivo):
    count = 0
    print('calculando posición inicial...')
    posicion, velocidad, frecuencia, tasa, sonoridad =
      posicion_inicial(tam_colonia = tam_colonia, valores_min = valores_min,
      valores_max = valores_max, funcion_objetivo = funcion_objetivo)
    mejor_ind = np.copy(posicion[posicion[:,-1].argsort()][0,:])
    print('vuelan los murciélagos..')
    while (count <= iteraciones):
        print("Iteración = ", count, " f(x) = ", mejor_ind[-1])
        posicion, velocidad, frecuencia, tasa, sonoridad, mejor_ind =
        acualiza_posicion(posicion, velocidad, frecuencia, tasa, sonoridad,
        mejor_ind, alfa = alfa, gama = gama, fmin = fmin, fmax = fmax, count =
        count, valores_min = valores_min, valores_max = valores_max,
        funcion_objetivo = funcion_objetivo)
        count = count + 1
    print(mejor_ind)
    return mejor_ind
```

```
# actualiza la posición de cada murciélago
def acualiza_posicion(posicion, velocidad, frecuencia, tasa, sonoridad, mejor_ind,
alfa = 0.9, gama = 0.9, fmin = 0, fmax = 10, count = 0, valores_min = [1,2,2,2],
valores_max = [16,8,4,6], funcion_objetivo = funcion_objetivo):
    posicion_temp = np.zeros((posicion.shape[0], posicion.shape[1]))
    for i in range(0, posicion.shape[0]):
        beta = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
        frecuencia[i,0] = fmin + (fmax - fmin)*beta
        for j in range(0, len(valores_max)):
            velocidad[i, j] = velocidad[i,j] + (posicion[i,j] -
            mejor_ind[j])*frecuencia[i,0]
        for k in range(0, len(valores_max)):
            posicion_temp[i,k] = posicion[i,k] + velocidad[i,k]
            if (posicion_temp[i,k] > valores_max[k]):
                posicion_temp[i,k] = valores_max[k]
                velocidad[i,k]      = 0
            elif(posicion_temp[i,k] < valores_min[k]):
                posicion_temp[i,k] = valores_min[k]
                velocidad[i,k]      = 0
                posicion_temp[i,-1] =
            funcion_objetivo(posicion_temp[i,0:len(valores_max)])
        rand = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
        if (rand > tasa[i,0]):
            for L in range(0, len(valores_max)):
                posicion_temp[i,L] = mejor_ind[L] + random.uniform(-1, 1)
                                    *sonoridad.mean()
```

Anexo 03: Función de aptitud que se minimiza.

```
def modeloCNN(variables_values=[0,0,0,0,0,0,0,0,0,0,0,0,0]):
    # Hiperparámetros a optimizar
    f1 = num_filtros[int(variables_values[0])]  # nro filtros
    f2 = num_filtros[int(variables_values[1])] # nro filtros
    f3 = num_filtros[int(variables_values[2])] # nro filtros
    f4 = num_filtros[int(variables_values[3])] # nro filtros
    k  = num_kernels[int(variables_values[4])] # kernel
    d1 = num_tasa_drop[int(variables_values[5])] # tasa dropout
    d2 = num_tasa_drop[int(variables_values[6])] # tasa dropout
    d3 = num_tasa_drop[int(variables_values[7])] # tasa dropout
    uni1 = num_neuronas[int(variables_values[8])]#neuronas capa densa
    lr = num_tasa_aprendizaje[int(variables_values[9])] # tasa de apr.
    fa= num_activacion[int(variables_values[10])] # función de activación
    op= num_optimizador[int(variables_values[11])] # optimizador

    # determinación del optimizador
    if op == 0:
        optimizador = Adam(learning_rate=lr)
    if op ==1:
        optimizador = SGD(learning_rate=lr)
    if op ==2:
        optimizador = RMSprop(learning_rate=lr)
    if op ==3:
        optimizador = Adadelta(learning_rate=lr)
    if op ==4:
        optimizador = Adagrad(learning_rate=lr)
    # arquitectura del modelo
    model = Sequential()
    model.add(Conv2D(f1,kernel_size=(k,k),activation=
                                      fa,input_shape=(img_fil,img_col,3)))
    model.add(Conv2D(f2,(k,k),activation=fa))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(d1))

    model.add(Conv2D(f3,(k,k),activation=fa))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(d2))

    model.add(Conv2D(f4,(3,3),activation=fa))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(d2))

    model.add(Flatten())
    model.add(Dense(uni1,activation=fa))
    model.add(Dropout(0.5))
    model.add(Dense(1,activation='sigmoid'))
```

Anexo 04: Optimización Bayesiana de la CNN

```
image_shape=(128, 128, 3)
fit_with_partial = partial(fit_params, image_shape, data)
# Región acotada del espacio de parámetros
pbounds = {'filtros':(32,128), 'kernels':(3, 5), 'tasa_drop':(0.1, 0.5),
           'neuronas': (64,128),'tasa_aprend': (0.001, 0.1), 'func_act':(0,2),
           'optimiz':(0,4)}

# Configuración del optimizador
optimizer = BayesianOptimization(
    f=fit_with_partial,
    pbounds=pbounds,
    verbose=2,
    random_state=1,
)

# Comenzar la busqueda de hiperparámetros
optimizer.maximize(init_points=3, n_iter=6)

# Imprime los resultados de cada iteración.
for i, res in enumerate(optimizer.res):
    print("Iteration {}: \n\t{}".format(i, res))

# Imprimir el resultado del mejor modelo.
print(optimizer.max)
```