FACULTAD DE INGENIERÍA

DEPARTAMENTO DE COMPUTACIÓN E INFORMÁTICA

CARRERA DE INGENIERÍA EN INFORMATICA

CLASIFICACIÓN DE RADIOGRAFÍAS MEDIANTE REDES NEURONALES CONVOLUCIONALES Y METAHEURÍSTICAS PARA MEJORAR LA DETECCIÓN DE ENFERMEDADES PULMONARES

Proyecto de Título para optar al Título de Ingeniero Informático

Mención en Gestión de la Información

Francisco Javier Montenegro Sepúlveda

Profesor Guía: Dr. Franklin Johnson Parejas

San Felipe, Chile

2023

Algunas personas temen que la inteligencia artificial nos hará sentir inferiores, pero entonces, cualquiera en su sano juicio debería tener un complejo de inferioridad cada vez que mira una flor.

Alan Curtis Kay (1940)

Agradecimientos

Al escribir estas palabras me encuentro en las etapas finales de un largo proceso, que ha definido tanto a mi persona como el camino que continuaré recorriendo en la vida: el rumbo de la informática. Este ha requerido de incontables horas de esfuerzo en búsqueda del perfeccionamiento, y nada de ello hubiese sido posible sin el constante apoyo recibido durante estos cinco años, más aún considerando todo lo ocurrido durante este lapso, factores tales como la pandemia global de COVID-19, la cual se hizo presente en el segundo año de mi carrera, y que prácticamente cambió el mundo y la forma en que se desarrollaron estos años.

Comienzo con cada uno de los profesores y profesionales de la Universidad de Playa Ancha que me han guiado en este camino, enseñándome de ámbitos relacionados a la planificación de proyectos, a la economía y estadística, a la vida laboral y ética, al idioma, y por supuesto, relacionados a la informática, desde el diseño visual hasta la codificación y evaluación de técnicas de inteligencia artificial y bases de datos. Agradezco además a mi profesor guía, Dr. Franklin Johnson, por sus consejos y recomendaciones en estos últimos dos semestres.

De la Universidad, agradezco también, y con gran cariño, a mis compañeros. Por su calurosa acogida y aprecio. Enormemente, a Roberto Cavieres, por cada una de las veces que él ha estado ahí para apoyarme, en especial en cada uno de los trabajos de alta calidad que hemos desarrollado juntos. También a Yanko Delgado, delegado, por su enorme coordinación e iniciativas por nosotros. No hubiese sido capaz de llegar tan lejos de no ser por tal ambiente de amistad.

Finalmente, agradezco a mi familia, por el apoyo incondicional que me entregaron al escoger esta carrera y este camino, sin dudar de mis capacidades. En especial a mi madre, Ingrid, por su comprensión y cariño, a mi tío Tristán, por el incontable soporte que nos ha dado, y a mis hermanos Carlos y Eduardo, también informáticos, y probablemente mis mayores fuentes de inspiración en la infancia.

Resumen

Han pasado más de cuatro años desde que inició la pandemia global de COVID-19, causada por el virus SARS-CoV-2. El efecto que ha tenido en el mundo, incluyendo a Chile, con más de cinco millones de casos y sesenta mil muertes registradas, de acuerdo a la Organización Mundial de la Salud (2023), ha llevado a la exploración y reutilización de distintas tecnologías para la identificación eficiente y eficaz de enfermedades y afecciones pulmonares, entre las cuales se encuentran el uso de pruebas de reacción en cadena de la polimerasa (PCR), pruebas de antígeno, pruebas de anticuerpos, tomografías computarizadas y radiografías de tórax, cada una contando con distintas disponibilidades, costos, requisitos y precisión a la hora de identificar la presencia de estas enfermedades o los efectos que hayan tenido en el cuerpo humano.

Con la presentación de este trabajo de investigación y desarrollo, se propondrán, estudiarán y evaluarán soluciones para resolver el problema de reconocimiento de enfermedades pulmonares con radiografías, mediante la lectura de bases de datos y el uso de técnicas avanzadas de inteligencia artificial, tales como lo son el uso de redes neuronales convolucionales, las cuales consisten en arquitecturas que aprenden a partir de datos para identificar patrones y reconocer clases, y el uso de algoritmos metaheurísticos novedosos inspirados en el comportamiento de animales, tales como el utilizado Forrajeo de Mantarrayas, que buscarán modificar los hiperparámetros de estas redes para obtener resultados con mayores valores de precisión.

Palabras Clave: Enfermedad pulmonar, radiografía, clasificación, red neuronal convolucional, metaheurística.

Abstract

More than three years have passed since the start of the global pandemic of COVID-19, caused by the SARS-CoV-2 virus. The effect it has had on the world, including Chile, with more than five million cases and sixty thousand registered deaths, according to the World Health Organization (2023), has led to the exploration and reuse of different technologies for the efficient and effective identification of lung diseases and conditions, including the use of polymerase chain reaction (PCR) tests, antigen or antibody tests, CT scans, and chest X-rays, each with varying availability, cost, requirements, and accuracy when identifying the presence of these diseases or the effects they've had on the human body.

By introducing this research and development work, solutions to solve the problem of recognition of lung diseases with X-rays will be proposed, studied and evaluated, by reading databases and using advanced artificial intelligence techniques, such as the use of convolutional neuronal networks, which consist of architectures that learn from data to identify patterns and recognize classes, and the use of novel metaheuristic algorithms, inspired by the behavior of groups of animals, such as the used Manta Ray Foraging, that will seek to modify the hyperparameters of these networks to obtain results with higher precision values.

Keywords: Lung disease, X-ray, classification, convolutional neural network, metaheuristic.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

1 PRESENTACIÓN DEL PROYECTO

1.1 Introducción al problema

Múltiples campos de estudio y desarrollo han hecho utilización del aprendizaje automático, subapartado de la inteligencia artificial que se centra en desarrollar sistemas que aprenden a identificar y clasificar elementos, o mejoran el rendimiento en función de los datos que consumen (Oracle, 2023), junto a distintas técnicas de optimización de procesos, en búsqueda de mejores resultados en cuanto a tiempo de ejecución y precisión obtenida. Entre estas técnicas se menciona el uso de metaheurísticas: métodos para resolver un problema computacional de manera automatizada, buscando facilitar la creación o generación de combinaciones que eventualmente son transformadas a configuraciones de estas redes y arquitecturas. Los resultados de la aplicación de tales tecnologías se traducen en mayores niveles de precisión a la hora de determinar características, como lo puede ser el reconocimiento de imágenes, y la optimización en cuanto a tiempos y recursos requeridos en estos procesos.

Es de entender que estas soluciones buscan tanto imitar las capacidades humanas a la hora de reconocer patrones, como facilitar la identificación, predicción o proyección de clases, valores y datos realizada por humanos, reduciendo a su vez los porcentajes de errores y ofreciendo una base de soporte.

En el campo de la medicina han existido múltiples aplicaciones de tales tecnologías, entre las cuales se encuentran la reconstrucción de mecanismos de propagación de enfermedades, pruebas de hipótesis a través de modelos estadísticos, proyecciones de síntomas en pacientes a través de simulaciones con aprendizaje automático, y el desarrollo de diagnósticos mediante técnicas de análisis de imágenes, que facilitan el proceso de identificación de síntomas y patrones en estas, y en ciertos casos, dan una clasificación con mayor precisión que la realizada manualmente por expertos (May, 2021).

Ejemplos de esta última integración en la medicina, son el uso de algoritmos de aprendizaje automático y redes neuronales convolucionales, las cuales consisten en arquitecturas que aprenden a partir de datos a identificar patrones, reconocer objetos, clases y categorías, para identificar de manera correcta condiciones y enfermedades en radiografías, tomografías computarizadas y distintos tipos de escaneos o procedimientos de carácter médico.

El propósito de este proyecto se enfoca en la optimización e incorporación de tales tecnologías bajo el contexto de la actual situación global. Habiendo transcurrido más de cuatro años desde que inició la pandemia global de COVID-19, causada por el virus respiratorio SARS-CoV-2, se ha presenciado su efecto a través de los millones de casos registrados a lo largo del tiempo, parte de estos resultando en la muerte, también, de millones de personas. Solamente en Chile, de acuerdo a cifras recogidas por la Organización Mundial de la Salud (2023), se han reportado más de 5.288.481 casos y 61.590 muertes; cifras que aumentan a 768.237.788 casos totales y 6.951.677 muertes a nivel global, poniendo en perspectiva la gravedad, sólo de esta enfermedad en específico. A su vez, se ha generado mayor conciencia respecto a la detección y prevención de una multitud de virus, enfermedades e infecciones respiratorias de carácter similar, entre las cuales se mencionan gripes, influenzas y neumonías, generando e intensificando campañas preventivas de vacunación e higiene durante los últimos años. Sin embargo, esto no remueve totalmente la posibilidad de infecciones y contagios, llevando el enfoque a la siguiente etapa: los procedimientos de diagnóstico.

Han sido varios los avances tecnológicos y aplicaciones utilizadas para facilitar el diagnóstico de enfermedades respiratorias, tales como el uso de pruebas de reacción en cadena de la polimerasa (PCR), pruebas de antígeno y anticuerpos, tomografías computarizadas y radiografías de tórax, cada una contando con distintas disponibilidades, costos, requisitos y precisión a la hora de identificar estas enfermedades o los efectos que hayan tenido en el cuerpo humano.

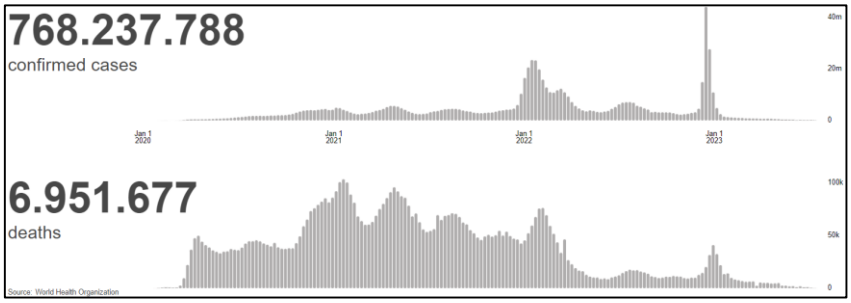


Figura 1: Representación visual de casos y muertes semanales confirmadas a nivel mundial, producidas por COVID-19 entre 2020 y 2023; Fuente: Organización Mundial de la Salud (2023).

Vacunación Campaña Influenza 2022				
Chile	Dosis únicas	1as dosis	2as dosis	Total
	6.874.358	289.685	111.433	7.275.476
Arica y Parinacota	93.329	3.623	1.214	98.166
Tarapacá	129.241	7.103	2.855	139.199
Antofagasta	240.030	11.794	3.777	255.601
Atacama	108.414	4.323	1.743	114.480
Coquimbo	292.448	12.334	4.064	308.846
Valparaíso	693.724	28.244	10.360	732.328
Metropolitana	2.474.282	126.186	47.636	2.648.104
Lib. Bdo O´Higgins	423.070	15.141	6.004	444.215
Maule	531.186	17.430	7.947	556.563
Ñuble	240.288	5.725	2.959	248.972
Biobío	615.550	21.873	8.742	646.165
Araucanía	394.325	13.371	5.974	413.670
Los Ríos	177.044	5.701	1.780	184.525
Los Lagos	340.034	13.131	5.133	358.298
Aysén	40.716	1.015	497	42.228
Magallanes	80.631	2.688	746	84.065
Sin Reporte de Residencia	46	3	2	51

Tabla 1: Campaña de Vacunación contra la Influenza año 2022 en Chile, dividida por regiones; Fuente: Registro Nacional de Inmunizaciones (2022).

A través de este proyecto se hará énfasis en la utilización de las técnicas de aprendizaje automático mencionadas anteriormente para potenciar y optimizar la lectura y clasificación de radiografías de pacientes, considerando la utilización de una base de datos la cual será procesada a través de codificaciones en el lenguaje Python. Para este proyecto se planteará el valor que representará esta solución, junto a las distintas consideraciones éticas, legales y de costo de la información y procesos manejados, para finalmente plantear soluciones que se espera mejoren resultados en cuanto a precisión de clasificación, tiempo de ejecución de los programas, o ambos, con mayor enfoque en el uso de técnicas de metaheurísticas y redes neuronales convolucionales.

1.2 Formulación del problema

Para entender la situación que desea ser enfrentada a través de este proyecto de título, se necesita hacer un acercamiento más profundo a los procedimientos mencionados en el punto anterior, y cómo estos se vincularían. Es de entender que existirán distintas fases de problemas o tareas a resolver en este proyecto de título, las cuales serán estudiadas, confeccionadas y evaluadas por separado. Una vez se encuentren funcionales, se implementarán en nuevas fases junto a aspectos añadidos o modificados, para así verificar la efectividad de ciertos métodos minimizando la mayor cantidad posible de ruido o factores externos que pudiesen alterar la percepción de los resultados obtenidos.

- La primera fase o tarea será entender cómo funciona la detección de enfermedades pulmonares mediante radiografías de tórax y qué factores implican la presencia o no de una condición en esta zona.

- La segunda fase o tarea será estudiar y aplicar conocimientos asociados al uso de redes neuronales convolucionales para clasificar bases de datos de imágenes en distintas categorías, y junto con esto, entender cómo se benefician de la utilización de técnicas metaheurísticas en sus procesos.

- La tercera fase o tarea será confeccionar algoritmos que permitan el funcionamiento de las ideas planteadas. Existirán múltiples formas de realizar este proceso, y a través de la investigación de este proyecto, se escogerá un modo específico de resolución que será estudiado, optimizado y evaluado finalmente.

- Una vez se obtengan resultados de aquellos algoritmos, se evaluarán para obtener una respuesta clara respecto a su efectividad, exista o no.

A continuación, se profundizará en las ideas mencionadas con anterioridad, junto a aspectos que justifican el valor y utilidad del trabajo propuesto y desarrollado.

1.2.1 Detección de enfermedades pulmonares mediante radiografías

Durante la introducción se mencionaron distintas técnicas de diagnóstico de enfermedades respiratorias. Entre estas, está la utilización de radiografías, las cuales consisten en imágenes obtenidas a través de la exposición de un receptor (el paciente) a una fuente de radiación, siendo esta, en la mayoría de los casos, rayos X. Aquellas obtenidas del área del tórax suelen realizarse cuando los doctores sospechan de desordenes y enfermedades asociadas al corazón o los pulmones, debido a que otorgan información valiosa sobre anormalidades (Dezube, 2021). Ejemplos de estas, para el caso del tórax, incluyen infecciones, colapso pulmonar parcial o total, líquidos y fluidos acumulados y depósitos de calcio, los cuales pueden llevar a enfermedades y lesiones tales como neumotórax o neumonía en uno o ambos pulmones.

Sin embargo, leer radiografías es una tarea difícil y desafiante, debido a la existencia de ruido y artefactos técnicos, descritos como rasgos que enmascaran, ocultan o imitan características clínicas (Willis et al., 2003). Pueden ser causados por limitaciones de hardware, software o el operador radiólogo, y causar casos de interpretaciones incorrectas, como por ejemplo, falsos positivos.

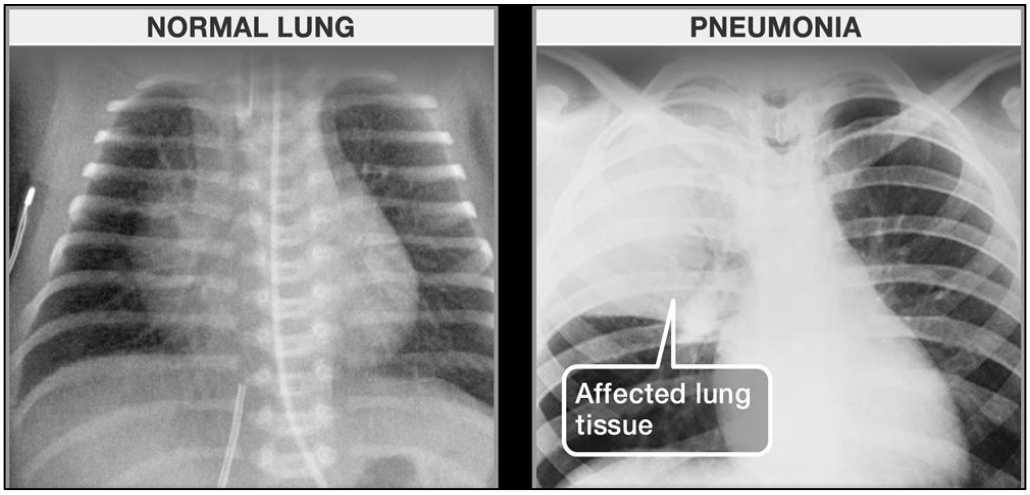


Figura 2: Comparación entre pulmones sanos y afectados con neumonía. Estos últimos presentan sombras blancas causadas por fluidos; Fuente: AboutKidsHealth (2023).

1.2.2 Clasificación de imágenes mediante CNNs y metaheurísticas

Retomando este concepto tratado al inicio del informe, las redes neuronales convolucionales (o CNNs) son un subconjunto del machine learning y el núcleo de los algoritmos de aprendizaje profundo (IBM, 2023). Definidas como arquitecturas compuestas por capas de nodos, incluyendo capas de entrada, ocultas y de salida, buscan imitar las redes de neuronas del cerebro humano, a través de la conexión entre nodos y la activación y desactivación de estos para enviar mensajes entre capas, mediante conceptos como pesos y umbrales (fórmulas y valores matemáticos) que permiten ajustar la transmisión de datos. Existen varios tipos de redes neuronales, tales como las redes neuronales recurrentes utilizadas para el reconocimiento de voz y el procesamiento de lenguaje natural, y las redes neuronales convolucionales, las cuales se asocian a tareas de clasificación de imágenes y visión artificial. Aprovechan además principios del álgebra lineal tales como la operación sobre matrices, para identificar patrones en una imagen (líneas, manchas, luces y sombras) y aplicar funciones que filtran o pre procesan los datos para mejorar resultados. Esto las lleva a exigir un uso intensivo de recursos informáticos para entrenar los modelos que producen, junto a un considerable uso de tiempo para estos procesos.

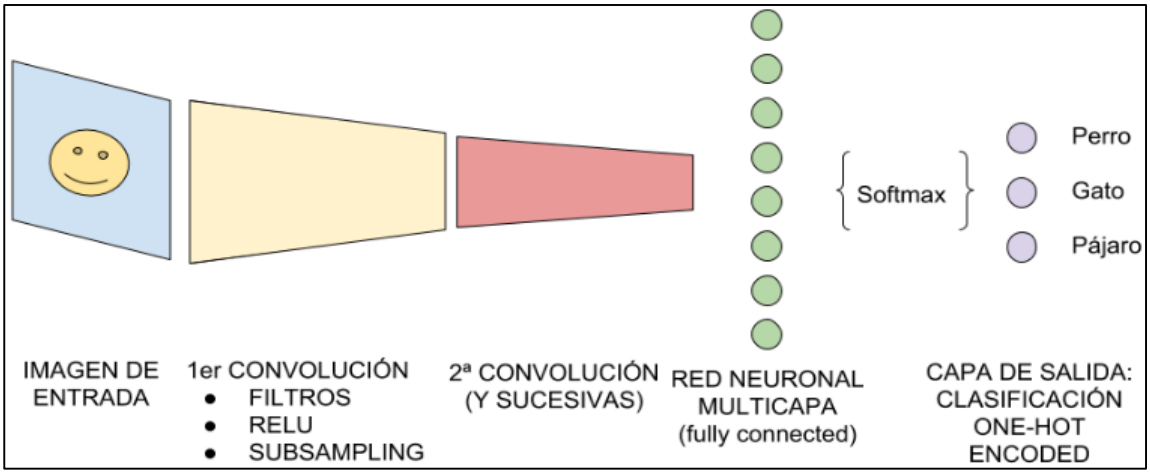


Figura 3: Diagrama de una red neuronal convolucional, indicando capas de entrada, ocultas y salida, junto con funciones de procesamiento; Fuente: Aprende Machine Learning (2018).

1.2.3 Descripción del problema a tratar

Comprendiendo, a través de los puntos anteriores, el porqué y cómo se utilizan los procesos de radiografías de tórax para identificar y diagnosticar síntomas de enfermedades pulmonares, y además, el funcionamiento y utilidad de las redes neuronales convolucionales para clasificar imágenes y asistir la visión artificial, se puede profundizar en la idea mencionada durante la introducción: juntar ambos campos de investigación para dar paso a la lectura y clasificación de radiografías mediante redes neuronales convolucionales, y así, generar código que pueda asistir este proceso y optimizar los resultados, ayudando a asistir a profesionales con la labor de clasificación de radiografías, permitiendo identificar de manera más eficiente enfermedades pulmonares y los efectos que estas causan. Junto a esto, se espera no sólo realizar el trabajo de clasificación mediante redes neuronales convolucionales, sino que además se buscará aplicar técnicas asociadas a metaheurísticas (métodos para resolver problemas computacionales) que permitan optimizar los resultados obtenidos por el sistema.

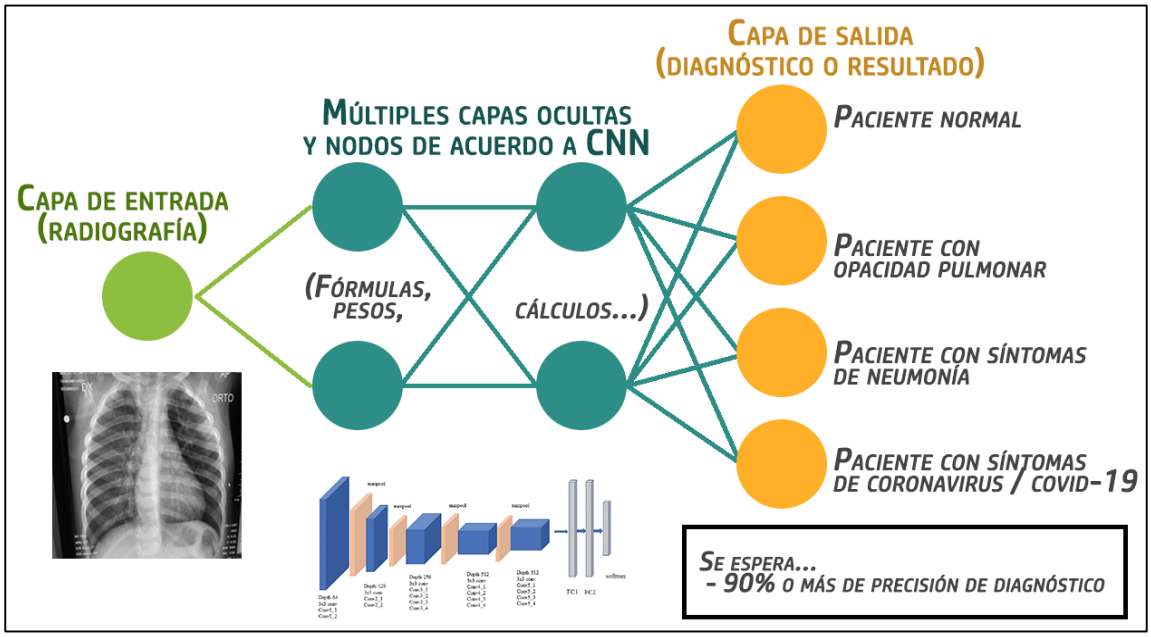


Figura 4: Ejemplo de red neuronal. Se ha simplificado el número de capas y nodos para facilitar el entendimiento de esta; Fuente: Elaboración Propia (2023).

1.2.4 Objetivos generales y específicos del proyecto

El objetivo general de este proyecto será la creación y optimización de una red neuronal convolucional para facilitar la clasificación de radiografías asociadas a enfermedades pulmonares, a través del uso de un algoritmo híbrido basado en técnicas de metaheurística y machine learning.

Los objetivos específicos de este proyecto se pueden especificar como:

- Aplicar conceptos asociados al aprendizaje automático (machine learning) e inteligencia artificial (IA) a través de codificación práctica y reproducible (que puedan obtenerse resultados consistentes).
- Estudio, revisión bibliográfica y aplicación de redes neuronales convolucionales y técnicas metaheurísticas populares o de interés en el mundo del aprendizaje automático y la inteligencia artificial.
- Estudio y revisión bibliográfica de autores que también hayan explorado la clasificación de radiografías de enfermedades pulmonares a través de CNNs y metaheurísticas.
- Desarrollo y pruebas de los algoritmos realizados para el proyecto, en sus distintas variantes (potenciados o no con distintas técnicas).
- Registrar cambios en la predicción del algoritmo para ciertas clases o categorías de la base de datos.
- Optimizar la selección de hiperparámetros, que serán modificados mediante técnicas de metaheurísticas.
- Análisis de los resultados obtenidos a través de la arquitectura creada, y comparación de estos con los resultados de otros autores, para identificar cambios en la precisión (accuracy score) y el tiempo de ejecución requerido para generar los modelos de la red neuronal.
- Confección de propuestas, reflexiones y observaciones en base a los resultados obtenidos.
- Proponer mejoras y cambios en base a las observaciones obtenidas.

1.3 Justificación del problema y estudio

Antes de comenzar la implementación de este trabajo, debe considerarse el valor y la utilidad que representaría como trabajo desarrollado y finalizado.

Aspectos de valor y utilidad	Descripción y comentario
¿Quién o quiénes se beneficiarían de una solución como esta?	Personal médico y centros que utilicen técnicas con radiografías.
¿Cómo se beneficiaría o beneficiarían ?	Obteniendo una base de apoyo para sus funciones, labores y revisiones.
¿Qué representa este trabajo para la informática?	La aplicación, estudio y comparación de técnicas con CNNs y metaheurísticas.
¿Qué representa este trabajo para el proceso de tesis de carrera?	La aplicación del conocimiento adquirido durante los últimos cinco años de manera práctica y evaluativa, con resultados.

Tabla 2: Aspectos de valor y utilidad asociados a la propuesta;
Fuente: Elaboración Propia (2023).

Junto con los aspectos mencionados anteriormente, existen dos puntos importantes que aún no han sido tratados, y que se definirán a mayor detalle.

- La viabilidad legal, comercial y ética (o social) de esta solución, en caso de integrarse en un proyecto de alcance comercial o social, como lo podría ser su uso en hospitales y clínicas.
- La viabilidad temporal y técnica de esta solución, respecto a su eficiencia y eficacia en comparación a otros métodos de detección de afecciones pulmonares que puedan existir a día de hoy.

A continuación, se estudiará a profundidad tales viabilidades mencionadas.

1.3.1 Viabilidad legal, comercial y ética: privacidad y consentimiento

La fabricación de modelos de aprendizaje automático requiere de una gran cantidad de datos, de manera que se pueda asegurar su correcto funcionamiento y a su vez la generación de un modelo capaz de entregar resultados útiles. Sin embargo, las bases utilizadas no siempre son creadas con el consentimiento de las fuentes, o los datos recopilados por estas no son protegidos con los estándares esperados. Casos emblemáticos asociados a estas problemáticas incluyen a Burke v. Clearview AI, Inc., demanda colectiva realizada hacia esta última compañía de reconocimiento facial, por la utilización de tres billones de imágenes de personas para entrenar algoritmos de identificación, sin obtener su consentimiento y poniendo en riesgo su privacidad (Wu, 2022).

Otro caso de interés, es lo sucedido con la empresa iRobot Corporation, fabricante de tecnología asociada a robótica que incluye traperos y aspiradoras automatizadas; estas últimas con capacidades de grabar audio y video para reconocer obstáculos y terreno, y a su vez, favorecer la tarea de entrenar algoritmos que identifiquen elementos. Sin embargo, se vieron envueltos en controversia al filtrarse grabaciones con contenido privado y delicado, obtenido por modelos de desarrollo exclusivos a trabajadores y coleccionistas asociados a iRobot que firmaron cláusulas para permitir la grabación de sus hogares, pero no esperaron que este contenido terminara finalmente en internet (Guo, 2022).

Por último, se encuentra el actual debate asociado a la utilización de inteligencia artificial generativa para crear arte, dibujos e ilustraciones, la cual ha sido ampliamente criticada por su utilización de bases de datos de piezas, cuadros, retratos, cómics y renderizados 3D recopilados de internet sin el consentimiento explícito de sus autores, en lo que se considera beneficiarse del trabajo de otras personas sin dar crédito o pagos (Marr, 2023). Este debate se extiende a otras ramas artísticas como la música y la escritura, lo que evidencia la importancia de estudiar la fuente de los datos trabajados y el manejo de estos.

Con casos como estos y similares perjudicando la percepción y utilización de técnicas de aprendizaje automático, es importante entender lo que este proyecto requerirá y utilizará, para así aclarar cuestiones éticas y legales en caso de querer implementar una solución como esta de manera comercial o a mayor escala.

- La utilización de una base de datos de radiografías. Su clasificación como material médico lleva a plantear si estas radiografías fueron conseguidas con el consentimiento explícito de los pacientes. En el proyecto realizado se utilizará una base de datos que recopila radiografías de distintas fuentes, cada una de estas sin características que permitan asociarlas a un individuo en específico, al carecer de metadatos tales como nombres, direcciones, fechas de atenciones, entre otros. El único metadato con el que cuentan es el tipo de paciente al que representan (en específico, si este presenta una enfermedad respiratoria o no), utilizado para determinar la precisión de modelos entrenados con esta base. Aún así, carecer del consentimiento de los pacientes involucrados en estas radiografías significa que debe tenerse mayor consideración a la hora de hacer uso de estas. A su vez, restringe el uso de esta base de datos a propósitos educativos y no comerciales, por lo que de poner en uso un modelo generado con esta en instituciones públicas o privadas podría llevar a problemas legales.

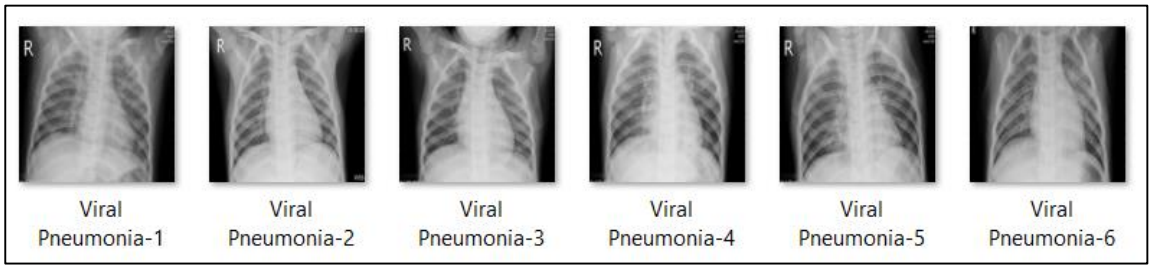


Figura 5: Extracto de las carpetas de la base de datos, mostrando la carencia de factores que identifiquen a los pacientes de estas radiografías; Fuente: Elaboración Propia (2023).

La utilización de una CNN: una arquitectura que se utilizará más adelante es VGG-19, entrenada con más de un millón de imágenes de la base de datos ImageNet (Simonyan y Zisserman, 2015). VGG-19 ha sido publicada bajo una licencia de Atribución 4.0 Internacional, lo que permite que se comparta y utilice en medios comerciales y no comerciales. Sin embargo, también presenta dilemas éticos asociados a ImageNet: esta base de datos no posee los derechos de autor de sus propias imágenes. En 2021, bajo el riesgo de la utilización inapropiada de sistemas de reconocimiento facial, se vieron en la obligación de difuminar y censurar los rostros de las personas en su base de datos (Shrestha, 2021).

Por lo tanto, es de entender que una de las mayores dificultades legales y éticas es justamente la obtención de las bases de datos. Sin embargo, como este proyecto es de carácter educativo, carece de mayores obstáculos para utilizar los recursos mencionados, en comparación a los que se presentarían en caso de querer recopilar radiografías y confeccionar una red neuronal convolucional manualmente y desde cero. Quedaría a disposición de la entidad asociada el reentrenamiento de este modelo con datos nuevos de pacientes, junto con las implicaciones de utilizar radiografías con o sin consentimiento explícito.

Figura 6: Extracto de imágenes contenidas en ImageNet y utilizadas para entrenar la CNN VGG-19; Fuente: V7 Labs (2022).

1.3.2 Viabilidad en cuanto a efectividad de la solución

Entendiendo que se desea facilitar procesos mediante soluciones efectivas y novedosas, es de importancia entender qué tan viable sería una solución como esta en la vida real. Bajo el caso de poseer cero implicaciones legales y éticas asociadas a las bases de datos de radiografías y redes neuronales convolucionales utilizadas, tema tratado en el punto anterior, surgen nuevas incógnitas: ¿implicará un mayor costo económico? ¿Requerirá mayor personal?

Para resolverlas, a continuación se realizará un recuadro comparativo con distintos métodos de detección y diagnóstico de SARS-CoV-2, el virus que causa el COVID-19, y uno de los que se plantea estudiar en este proyecto.

Tipo de examen	Descripción	Ventajas	Desventajas
Pruebas de PCR	Se detecta el ADN o ARN de un patógeno (organismo que causa una enfermedad) en una muestra de sangre, saliva, moco o tejido, utilizando una máquina especial y una enzima que ayuda a la identificación o no del virus.	Método preciso y confiable para identificar enfermedades infecciosas antes de que aparezcan síntomas de estas. Equipamiento económico y disponible en laboratorios.	Puede requerirse de una extracción de sangre o el uso de un hisopado nasal, los cuales pueden causar incomodidad al paciente involucrado. Junto con esto, los resultados pueden llegar a demorar hasta 3 días, y puede presentarse una multitud de problemas asociados a reacciones de componentes.
Pruebas de Antígenos	Utiliza un método de flujo lateral, con tiras reactivas a la presencia de proteínas superficiales como las del SARS-CoV-2. Si el resultado es positivo se observa una banda de color, similar a las pruebas de embarazo.	Pruebas extremadamente rápidas y realizables en casa, con resultados en 15 a 30 minutos. Bajo costo de compra.	Menor probabilidad de detección del virus que las pruebas de PCR, en especial cuando no hay síntomas. Un resultado negativo aislado en una prueba de antígeno no permite descartar la infección.

Tomografía Computarizada (CT Scan)	Toma de imágenes de varias secciones del pecho creando un modelo 3D, buscando áreas opacas o patrones que permitan identificar los síntomas asociados a SARS-CoV-2.	Más precisa que una radiografía, prueba de PCR o de antígeno. Mayor nivel de detalle en las imágenes generadas debido a su naturaleza 3D.	Carencia de disponibilidad en hospitales pequeños o rurales. Mayor gasto de recursos energéticos y conocimiento requerido para su operación. Requiere desinfectar equipamiento.
Radiografía (X-Ray)	Generación de imágenes en 2D de las estructuras internas del cuerpo a través de rayos X, permitiendo observar áreas opacas, patrones y síntomas asociados al SARS-CoV-2.	Más rápidas de realizar que las tomografías computarizadas. Mayor accesibilidad y menor costo de realización y equipamiento.	La precisión es usualmente menor que una tomografía computarizada. Además, presenta nuevamente el problema de requerir médicos o capacitados para realizarse. Requiere desinfectar equipamiento.

Tabla 3: Comparación entre distintos tipos de pruebas y exámenes para diagnosticar SARS-CoV-2; Fuente: Elaboración propia a través de datos del Centro para el Control y la Prevención de Enfermedades (2023), IntegraMédica (2023), Iryna, B. (2022) y Universidad de Chile (2020).

Como se pudo apreciar a través de la tabla, tan sólo con cuatro métodos distintos de detección existe una cantidad innumerable de factores a tener en cuenta al utilizarlos. Escogiendo las radiografías como tema de este proyecto, se espera potenciar la efectividad de este método y a la vez resolver las desventajas que presente su uso, mediante la utilización de diagnósticos automáticos con CNNs y potenciando su uso a través de las técnicas de metaheurística que se estudiarán. Así y en un futuro se podrá reevaluar si esta técnica presentaría superioridad en ventajas percibidas.

Además, es importante destacar que estas aplicaciones no se restringen al ámbito de detección de enfermedades respiratorias: lo aprendido en este proyecto podría aplicarse en otros ámbitos relacionados a la clasificación de imágenes con redes neuronales convolucionales, como lo pueden ser la identificación de tumores cerebrales, lesiones cancerígenas, entre otros.

2 ESTADO DEL ARTE

2.1 Metodologías y tipo de investigación a utilizar

Este proyecto, en cuanto a metodologías de desarrollo de software, se asimila a las metodologías de prototipado y desarrollo incremental, en base a lo realizado durante ambos semestres del presente año 2023.

- Se codificaron distintos tipos de prototipos en cuanto a código y funcionalidades (carga de datos, lectura de bases de datos, funcionamiento de bio-metaheurísticas e implementación de una CNN) de manera separada y en distintas oportunidades, permitiendo que se pudiese formar un ciclo de feedback o retroalimentación que ayudase a entender y aterrizar los requerimientos que este trabajo conlleva, en forma de prueba y error durante los meses de mayo y junio.
- Posterior a esto, se hizo avance en la aplicación y estudio de distintas técnicas y optimización de procesos durante el segundo semestre, mediante el estudio y codificación de estas, evaluadas posteriormente, y dependiendo de su rendimiento, descartadas o potenciadas con nuevas versiones de código e implementaciones.
- Además, el desarrollo mencionado anteriormente encaja con la construcción progresiva e incremental del producto final (en este caso, el modelo que permitirá clasificar radiografías automáticamente), comenzando por módulos pequeños de importación y lectura que dan paso a las arquitecturas más pesadas de aprendizaje automático.

Cabe destacar que debido a la naturaleza de este proyecto, el principal tipo de investigación realizada es cuantitativa, ya que con las comparativas mencionadas de precisión y tiempo de ejecución, se espera hacer énfasis en relaciones numéricas de causa y efecto para buscar soluciones óptimas, mediante un análisis estadístico de los resultados de estas y su observación experimental.

2.2 Recursos y arquitecturas a utilizar

Parte de estos fueron mencionados parcialmente en el punto 1.3.1 para tratar específicamente temas éticos y legales. El primer recurso utilizado para este proyecto será la base de datos de radiografías de COVID-19, por Rahman et al. (2022). Este set de datos, confeccionado por un equipo de investigadores de distintas universidades de Catar, Bangladés, Pakistán y Malasia, y disponible de manera pública a través de la comunidad de aprendizaje automático y ciencia de datos Kaggle, se caracteriza por incluir un total de 21.165 muestras o radiografías de pacientes, completamente anonimizadas. Estas se dividen en cuatro categorías, permitiendo su utilización en problemas multi-clase (donde una red neuronal debe clasificar un elemento en una de varias opciones): 3.616 radiografías de pacientes con COVID-19, 6.012 radiografías de pacientes con Opacidad Pulmonar, 10.192 radiografías de pacientes en condición normal y 1.345 radiografías de pacientes con Neumonía.

Este set, además, cuenta con capas de máscara que permiten recortar solamente las áreas de los pulmones en cada radiografía, evitando que los resultados se distorsionen producto del ruido en las demás áreas del tórax. A esto se le conoce como trabajo de segmentación, y se revisará su efectividad durante la creación de los distintos módulos asociados a la red neuronal convolucional deseada.

Es de importancia mencionar que los mismos autores de esta base de datos proponen versiones mejoradas, tales como COVID-QU-Ex Dataset (Tahir et al., 2022) debido a su mayor número de radiografías y clases, pero se ha preferido utilizar la base de Rahman et al. debido a la mayor documentación y utilización que posee. Finalmente, esta base de datos será trabajada en tres versiones durante el desarrollo y evaluación de los algoritmos, para facilitar la prueba de distintas modalidades bajo restricciones de tiempos, debido al costo computacional y temporal mencionado previamente en el punto 1.2.2. De esta forma, se podrá codificar y evaluar con mayor rapidez y bajo distintos casos.

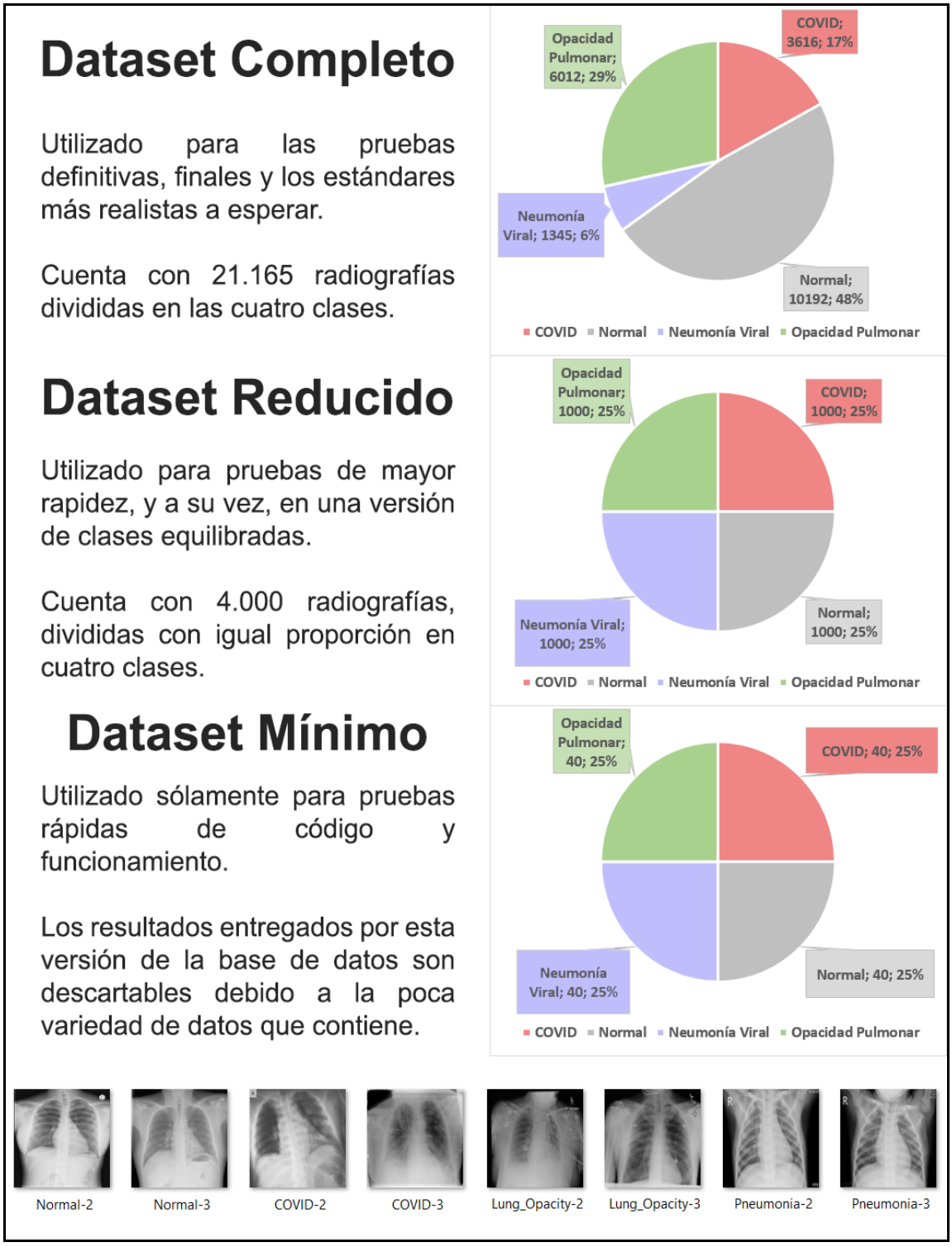


Figura 7: Muestra de las tres versiones de la base de datos que se utilizará, junto a una pequeña selección de radiografías; Fuente: Elaboración Propia (2023).

Como arquitectura principal se utilizará la red neuronal convolucional VGG-19 (Simonyan y Zisserman, 2015). Esta arquitectura es una variante del modelo VGG (Visual Geometry Group) basada en 19 capas de distintos tipos, y la cual fue entrenada mediante la base de datos ImageNet como se mencionó anteriormente. Entre las características a destacar de esta, se encuentran:

- Recibe imágenes a color o en blanco y negro de tamaño fijo (224 pixeles de ancho y alto), las cuales son procesadas como matrices. Sin embargo, permite hasta un tamaño mínimo de 32 pixeles de ancho y alto. Durante el desarrollo de este ejercicio se probará principalmente con su resolución máxima para evitar la pérdida de datos y rasgos, y se utilizarán tamaños menores de imagen para pruebas rápidas bajo distintas condiciones de entrenamiento.

- Las imágenes utilizadas como entrenamiento pasan a través de distintas capas convolucionales (extraen y comprimen datos a través de funciones para revelar características visuales), capas de pooling (reducen el sobreajuste u overfitting producido al trabajar con conjuntos de datos poco generales), y capas completamente conectadas que aplican transformaciones lineales a lo que reciben. De esta forma y finalmente, se obtiene una clasificación final obtenida en base a funciones que reparten valores numéricos entre las clases disponibles por el ejercicio. Un ejemplo de esto es la función softmax, utilizada comúnmente en problemas de clasificación múltiple como el estudiado debido a su capacidad de asignar probabilidades hacia cada clase estudiada en el problema.

VGG-19 sobresale clasificando clases como animales, personas, utensilios y adornos, sin embargo, para utilizarla en un trabajo de radiografías esta debe ser entrenada nuevamente, permitiéndole utilizar los conocimientos adquiridos de patrones y características para un problema con clases completamente nuevas.

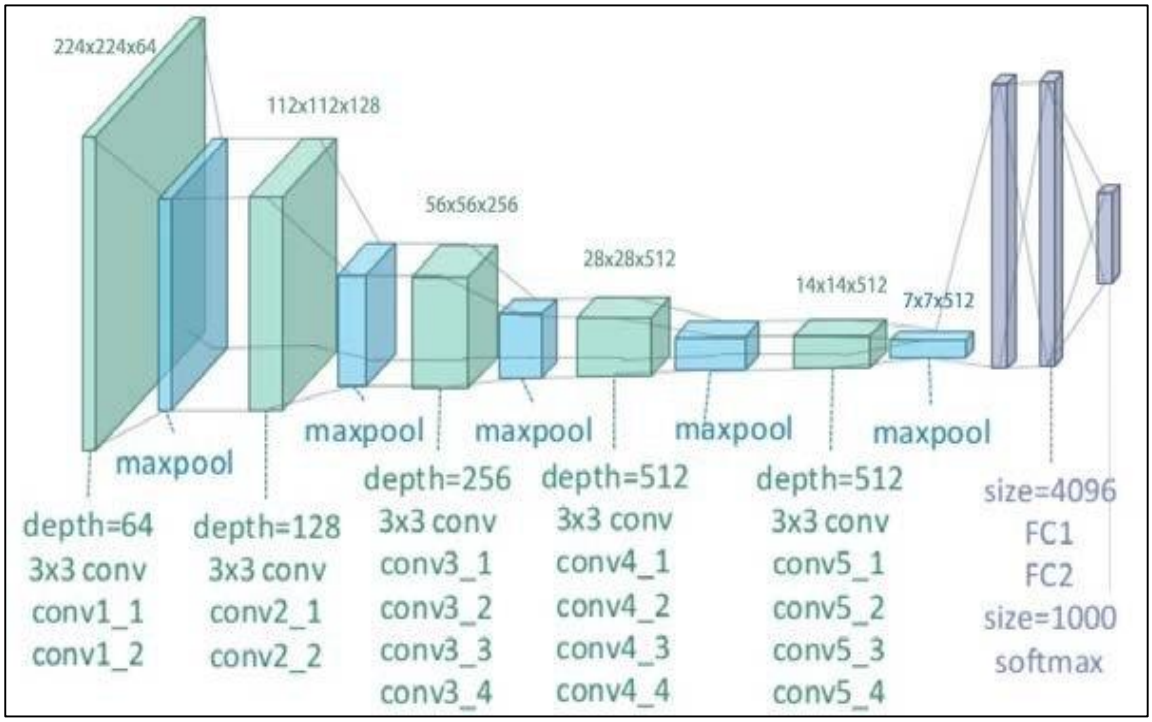


Figura 8: Arquitectura de VGG-19, con énfasis en los tipos de capas, profundidad de nodos y funciones matemáticas asociadas; Fuente: Yang, C. (2018).

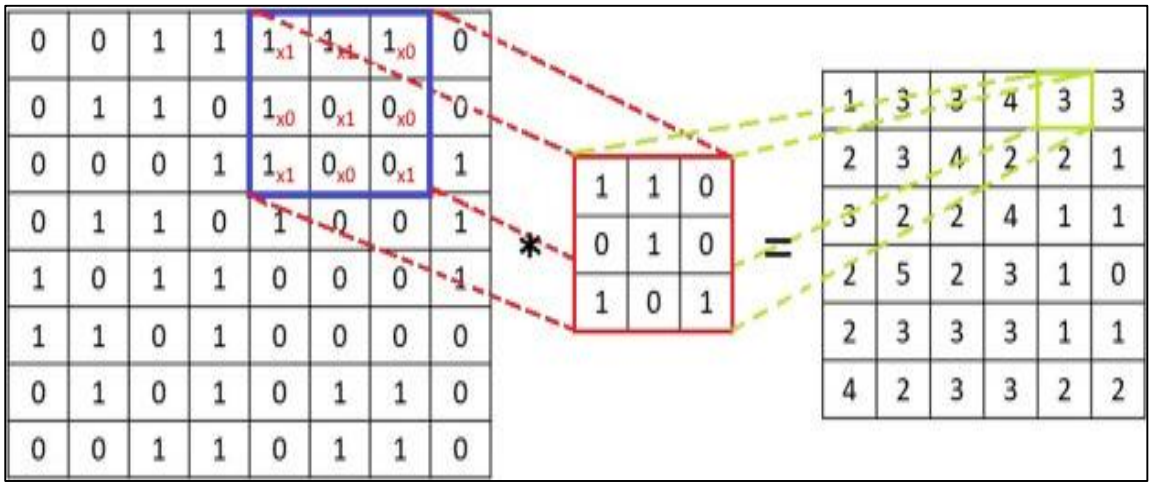


Figura 9: Muestra del proceso de extracción de características en una capa convolucional; Fuente: Singh, Meitei, Majumder (2020).

2.3 Herramientas y plataformas a utilizar

Son distintas y variadas las herramientas y comunidades disponibles para trabajar los conceptos de aprendizaje automático. Lo que se decidió utilizar es:

- Lenguaje de programación Python, ampliamente utilizado en aplicaciones web, desarrollo de software, ciencia de datos y aprendizaje automático (Amazon Web Services, 2023). Desarrollado en 1991, este es el lenguaje de preferencia por expertos a la hora de trabajar el aprendizaje automático debido a la legibilidad de su código, interpretación en tiempo de ejecución, multi-plataforma mediante la utilización de su intérprete correspondiente, y las librerías comúnmente asociadas a este (Frías, 2020):

    - NumPy, utilizada para computación científica al proporcionar estructuras de datos, vectores y matrices multidimensionales, junto con funciones matemáticas de alto nivel y eficiencia en ejecución.
    - Pandas, diseñada para la manipulación y análisis de datos, que proporciona estructuras tales como dataframes, y procesamiento de datos como limpieza, normalización, visualización e inspección.
    - Matplotlib, biblioteca de trazado 2-D de Python que produce figuras e ilustraciones en una gran variedad de formatos y entornos, tales como histogramas, gráficos de barra, dispersión, entre otros.

Figura 10: Logotipos asociados a Python y sus librerías;
Fuente: Python.org, NumPy.org, Pandas.pydata.org y Matplotlib.org. (2023).

Junto con esto, entre las librerías específicas al trabajo de aprendizaje automático que se desea realizar como parte de este proyecto, se menciona:

- OpenCV y cv2, librerías asociadas al problema de visión artificial y procesado de imágenes para identificar patrones y características. De carácter de código abierto, otorga una infraestructura que acelera trabajos de percepción automatizada, lectura de datos y algoritmos optimizados.
- Scikit-learn, herramientas para la predicción y análisis de datos, construida en base a NumPy, SciPy y matplotlib, y optimizada para problemas de clasificación, regresión o clustering, entre los que se pueden encontrar la detección de mensajes indeseados (spam), reconocimiento facial, respuestas de fármacos y drogas, o inclusive segmentación de clientes.
- Tensorflow, que facilita la creación y entrenamiento de modelos de aprendizaje automático a través del uso eficiente de cálculos con CPUs (unidades de procesamiento computacional), GPUs (unidades de procesamiento gráfico) e inclusive TPUs (unidades de procesamiento de tensores), las cuales se adaptan específicamente a este software, favoreciendo los tiempos de ejecución e inclusive los resultados obtenidos.
- Seaborn, librería de visualización basada en matplotlib que proporciona interfaces de alto nivel para la generación de gráficos llamativos e informativos, con interfaces de alto nivel y facilidad de uso.

Figura 11: Logotipos asociados a las librerías mencionadas;
Fuente: Opencv.org, Tensorflow.org, Scikit-learn.org y Seaborn.pydata.org. (2023).

En cuanto a la utilización de plataformas y entornos de programación, se investigaron distintas posibilidades y se escogió finalmente una que será utilizada de manera oficial para el desarrollo de este proyecto, todas asociadas a interfaces de cuadernillos y celdas como se verá más adelante.

La primera es Jupyter Notebook, aplicación de escritorio que permite la programación en python de manera local. Sin embargo, presentó dificultades e incompatibilidades entre librerías con su suite de código abierto Anaconda, debido a la cantidad de dependencias requeridas para este trabajo. Además, el uso intensivo de recursos por parte de una red neuronal convolucional dificulta su utilización de manera local, al carecer los recursos tecnológicos necesarios para entrenar múltiples redes de estos tipos en tiempos razonables, como aquellos menores a media hora o una hora por cada prueba realizada.

Figura 12: Logotipo de Jupyter Notebook y muestra de la interfaz asociada;
Fuente: Jupyter.org y elaboración propia (2023).

Otra alternativa que se evaluó fue Google Colab, con una interfaz similar a Jupyter Notebook que permite codificar en la nube y realizar fácilmente importaciones de archivos desde Google Drive. No presentó problemas de incompatibilidades o librerías al utilizarse, sin embargo, fue incapaz de correr algoritmos de entrenamiento debido a que al acceder a la base de datos desde Drive, la comunicación entre ambas plataformas ralentizaba considerablemente el tiempo de ejecución, estancando el programa. Junto con esto, en su momento contaba con opciones extremadamente limitadas respecto a aceleradores GPU o TPU, los cuales no siempre aparecían disponibles o requerían una suscripción monetaria para ser utilizados. Sin estas herramientas, el tiempo de entrenamiento para cada prueba hubiese aumentado considerablemente.

Figura 13: Logotipo de colab y muestra de la interfaz asociada;
Fuente: Colab.google y elaboración propia (2023).

Después de investigar las dos posibilidades anteriores, se escogió la interfaz de Kaggle: habiendo mencionado previamente la comunidad a la que pertenece en el punto 2.2, esta interfaz combina funciones de las dos para ofrecer un entorno de programación amigable, junto con nuevas funcionalidades: entre estas se encuentra la creación de copias de respaldo e historial de versiones, importación del set de datos desde la misma comunidad, y la utilización de aceleradores de GPU o TPU que permiten reducir los tiempos de ejecución como se mencionó con anterioridad. Para estas últimas características se requiere la verificación gratis de una cuenta en Kaggle, y cuenta con una cuota de 30 horas de uso semanal gratis para GPU y 20 para TPU, permitiéndole ser viable a la hora de experimentar. Durante el desarrollo de los algoritmos mencionados no presentó problemas en cuanto a importación de librerías, como fue el caso de Jupyter Notebook, o ralentizaciones como las de Google Colab.

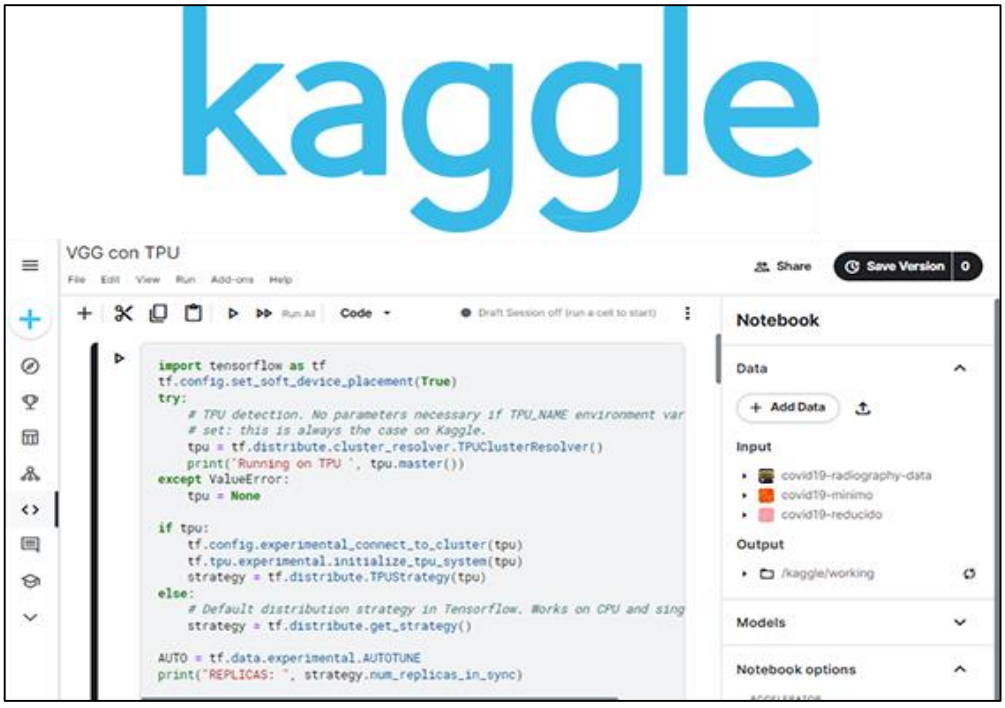


Figura 14: Logotipo de kaggle, y captura de la interfaz de cuadernillo y celdas;
Fuente: Kaggle.com y elaboración propia (2023).

2.4 Metaheurísticas y métodos de optimización

De acuerdo a Sancho (2021), en la inteligencia artificial (IA) se emplea el término heurístico en un sentido muy genérico, para aplicarlo a aquellos aspectos que tienen que ver con el empleo de conocimiento en la realización dinámica de tareas. Se habla de heurística para referirse a una técnica, método o procedimiento inteligente de realizar una tarea que no es producto de un riguroso análisis formal, sino de conocimiento experto sobre esta, con buen rendimiento. Eventualmente, se forman y obtienen técnicas y recursos computacionales específicos para la resolución de problemas y construcción de algoritmos, que llevan a denominarse metaheurísticas: mediante el sufijo meta que significa “más allá”, se hace referencia a mejorar procedimientos heurísticos eficientemente.

El término metaheurística aparece por primera vez en 1986 a través de un artículo seminal sobre búsqueda tabú, método de optimización matemática para seleccionar mejores elementos mediante conceptos tales como memoria a distintos plazos, espacios de búsqueda, candidatos de soluciones, ajuste, entre otros, estos estudiados por el Director Científico Fred W. Glover.

Entre las distintas aplicaciones que se puede dar a esta tecnología y en base a la información recopilada por Torres-Jiménez y Pavón (2014), se encuentran:

- La programación dinámica de una terminal de contenedores en un puerto, para poder facilitar el movimiento de un entorno altamente dinámico y con múltiples objetivos, como la distribución de grúas y terminales.
- Distribuir de manera eficiente e inteligente cursos de un currículo escolar, un problema clásico que es subdividido en sub problemas: la asignación, distribución y planificación de estos cursos para cada alumno y profesor.

Las metaheurísticas pueden ser analizadas de acuerdo a una serie de clasificaciones distintas de acuerdo al uso que se espera darles, e inclusive la combinación de estos distintos tipos como se ejemplificará a continuación.

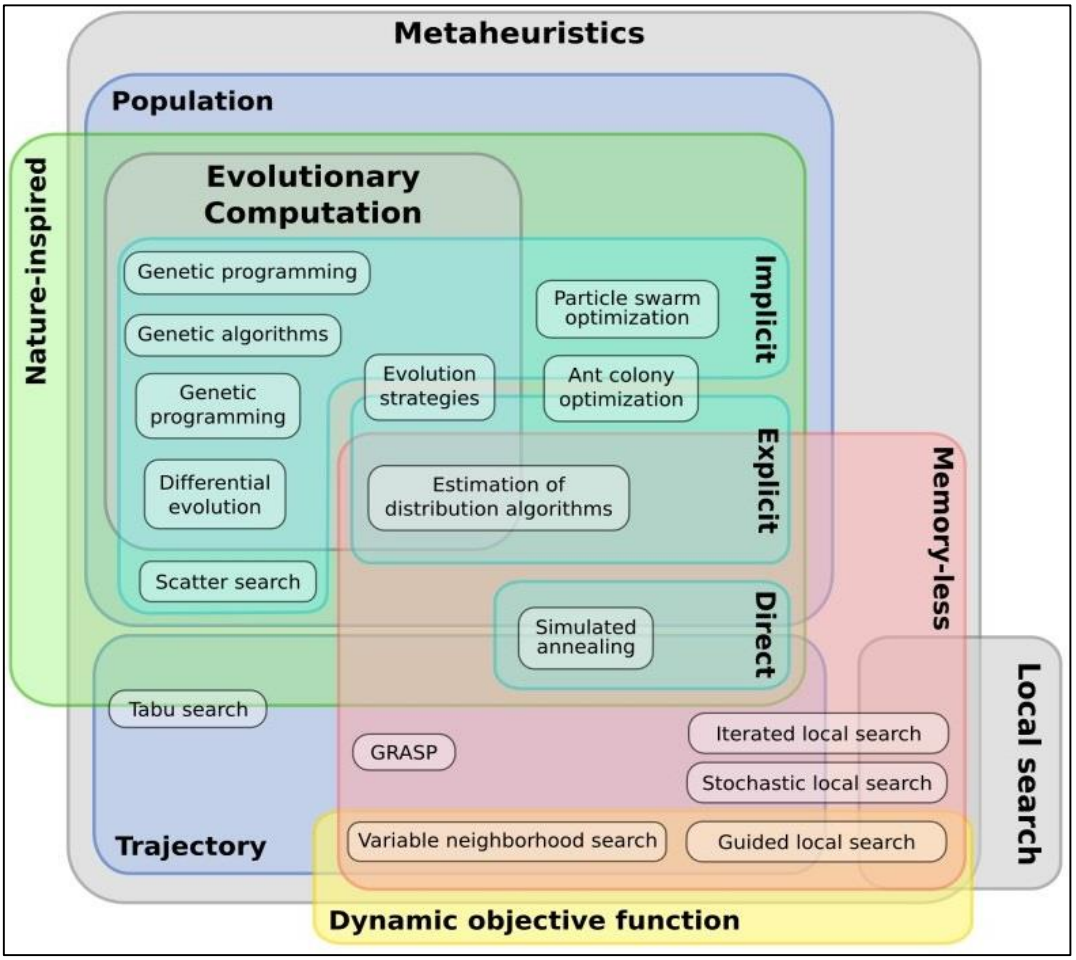


Figura 15: Diagrama de Euler para las distintas clasificaciones de metaheurísticas; Fuente: Sadigh et al. (2012).

- Metaheurísticas de relajación, procedimientos de resolución de problemas que utilizan modificaciones de un modelo original para facilitar su resolución a través de comparativas entre distintas versiones.
- Metaheurísticas constructivas, que buscan obtener una solución a través del análisis y selección paulatina de los componentes que la forman.
- Metaheurísticas de búsqueda, que utilizan transformaciones o movimientos para recorrer el espacio de soluciones alternativas y explotar las estructuras de entornos que puedan estar asociadas.
- Metaheurísticas evolutivas, que están enfocadas a los procedimientos basados en conjuntos de soluciones que evolucionan sobre este espacio.

Es de relevancia mencionar que parte de estos algoritmos tiene su origen en los comportamientos observados en la naturaleza por diversos grupos de insectos y animales. A este tipo de algoritmo se le puede conocer de distintas formas: metaheurística bioinspirada o bio-metaheurística. Junto con esto, la utilización de tales algoritmos no es rígida: pueden utilizarse para optimizar y/o modificar una gran variedad de características de acuerdo al problema que desea resolverse.

- En el trabajo de Gaspar et al. (2021), se hace un enfoque a la optimización de hiperparámetros mediante metaheurísticas debido a su importancia en el rendimiento de una red neuronal convolucional. Se busca minimizar el valor de pérdida y maximizar el valor de precisión del algoritmo a través de la alteración de valores como el número de capas y unidades utilizadas. Esto cae dentro de un primer caso conocido como Parameter Tuning.
- Otro caso de Parameter Tuning es el realizado por Johnson et al. (2020), en el que se propone un nuevo algoritmo genético para optimizar una arquitectura de CNN para un problema de clasificación de imágenes, mediante la modificación de hiperparámetros como la profundidad de la red neuronal, el número de filtros de cada capa y el tamaño del kernel.
- En el artículo de Abdelsamee et al. (2022), se utilizan técnicas de metaheurísticas híbridas para seleccionar, con mayor precisión, características de un set de radiografías analizado con CNNs y con hiperparámetros fijos. Este caso cae dentro de un segundo término conocido como Feature Selection, o selección de características.

Como se ha mencionado, se desea la utilización de metaheurísticas para optimizar el rendimiento del modelo de clasificación, por lo que resultará útil estudiar distintos tipos de metaheurísticas que puedan ser de ayuda para esto, tomando especial interés en aquellas dentro de la categoría de bioinspiradas, y escogiendo una para implementar finalmente, en base a la disponibilidad, popularidad y novedad que corresponda.

Metaheurística	Descripción de la metaheurística
Ant Colony Optimization (ACO)	Traducida como Optimización Basada en Colonias de Hormigas. Se inspira en el comportamiento estructurado de las colonias de hormigas, donde individuos muy simples de una colonia se comunican entre sí por medio de una sustancia química denominada feromona, para marcar caminos y recorridos realizados por estos, que finalmente llevan a identificar las rutas más adecuadas entre un nido y recursos.
Fish Swarm Algorithm (FSA)	Traducida como Algoritmo de Enjambre de Peces. Imita el movimiento de peces acuáticos en un medio líquido. Se escoge un objetivo de manera aleatoria hacia el cual se avanza de manera iterativa mediante pequeños pasos que afectan la optimización.
Genetic Bee Colony (GBC)	Traducida como Colonia Genética de Abejas. Imita el comportamiento de colonias de abejas que dividen tareas de acuerdo a roles de abejas obreras (identifican fuentes de comida), abejas espectadoras (escogen la mejor fuente entre las identificadas) y abejas exploradoras (asignan fórmulas matemáticas para designar trabajos de búsqueda aleatorios a obreras).
Manta Ray Foraging Optimization (MRFO)	Traducida como Optimización Basada en Forrajeo de Mantarrayas. Imita la búsqueda de alimento realizada por las mantarrayas: búsqueda en cadena por comida, búsqueda en forma de ciclón, y búsqueda en forma de voltereta, utilizadas en la vida real para identificar las fuentes más densas de plancton.
Particle Swarm Optimization (PSO)	Traducida como Optimización Basada en Enjambre de Partículas. Se inspira en el comportamiento de grupos de aves (parvadas) recorriendo áreas de búsquedas con distintas velocidades, y el trabajo en conjunto que realizan para comunicar los hallazgos en cada iteración del recorrido o búsqueda que realizan, ajustando parámetros.
Whale Optimization Algorithm (WOA)	Traducida como Algoritmo de Optimización de Ballenas. Este algoritmo es similar al comportamiento de caza de las ballenas jorobadas, e implica tres pasos principales: rodear a las presas, realizar un ataque de red de burbujas para aturdirlas, y seguir buscando más presas incluso durante la caza de acuerdo a la ubicación de otras ballenas.

Tabla 4: Descripción y clasificación para distintos tipos de metaheurísticas de relevancia;

Fuente: Elaboración propia a través de información recopilada de los siguientes autores:

Suárez, O. (2013), Game et al. (2020), Devika, G. y Gowda, A. (2022) y Abdullahi et al. (2023).

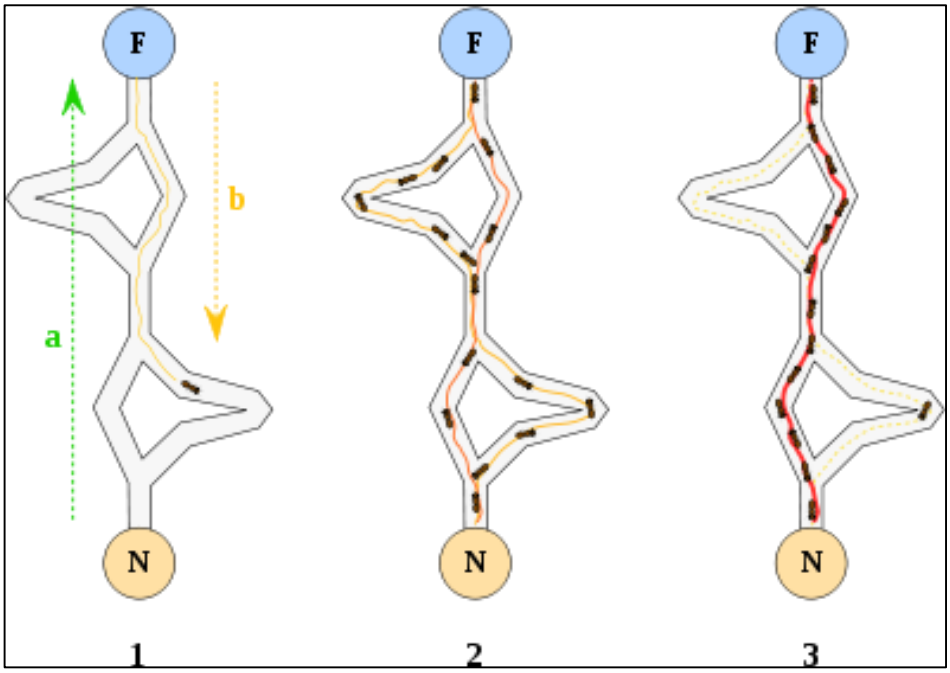


Figura 16: El camino más corto encontrado en una ruta específica según ACO; Fuente: Dréo, J. (2006).

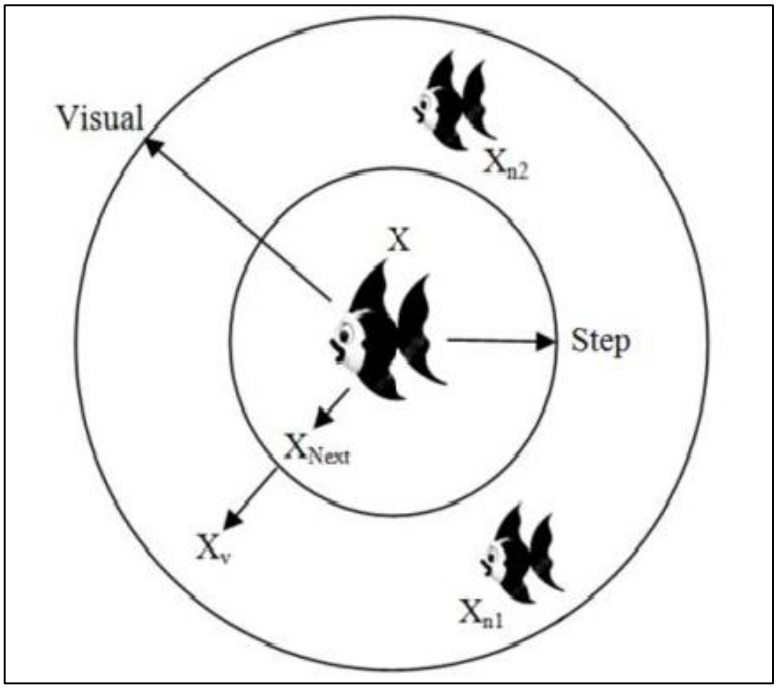


Figura 17: Ilustración del funcionamiento de FSA, destacando el espacio de exploración de un pez y los elementos que podrá encontrar; Fuente: Azizi, R. (2014).

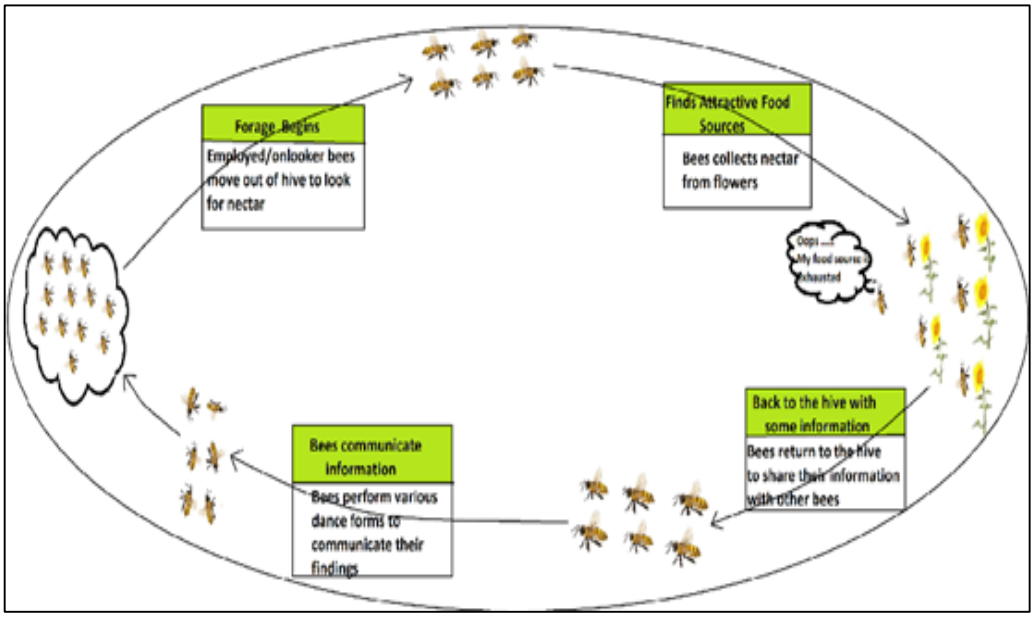


Figura 18: Ilustración del funcionamiento de GBC, destacando los procesos de búsqueda y transmisión de información; Fuente: Bansal et al. (2013).

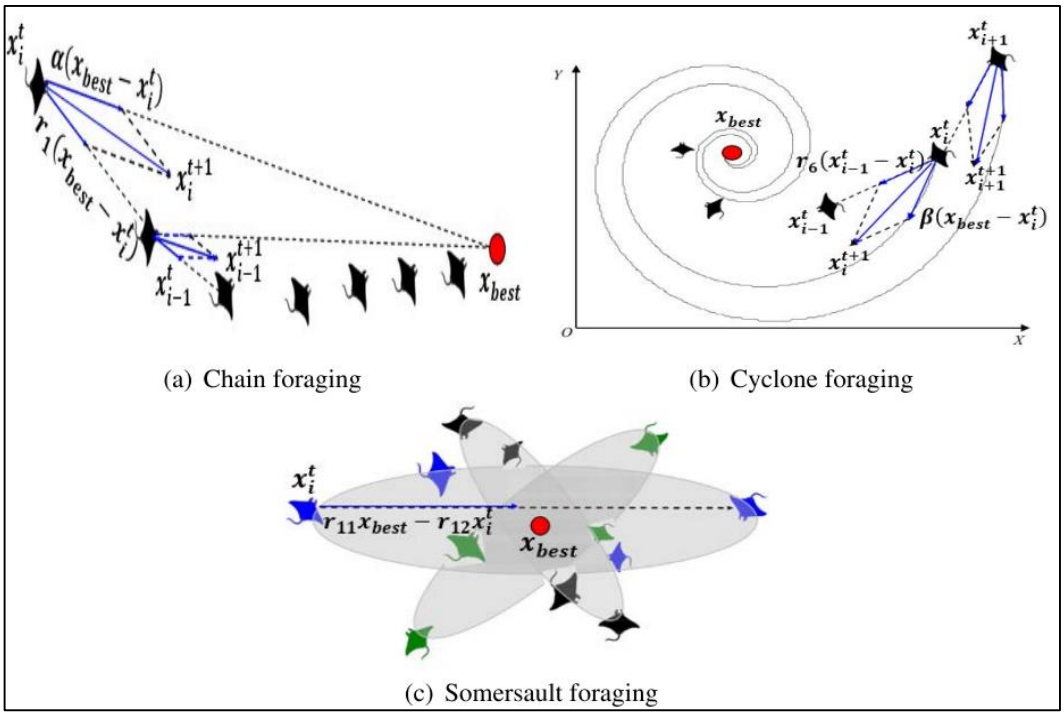


Figura 19: Las distintas formas de caza de la mantarraya de acuerdo a la situación de su ambiente y entorno; Fuente: Daqaq et al. (2013).

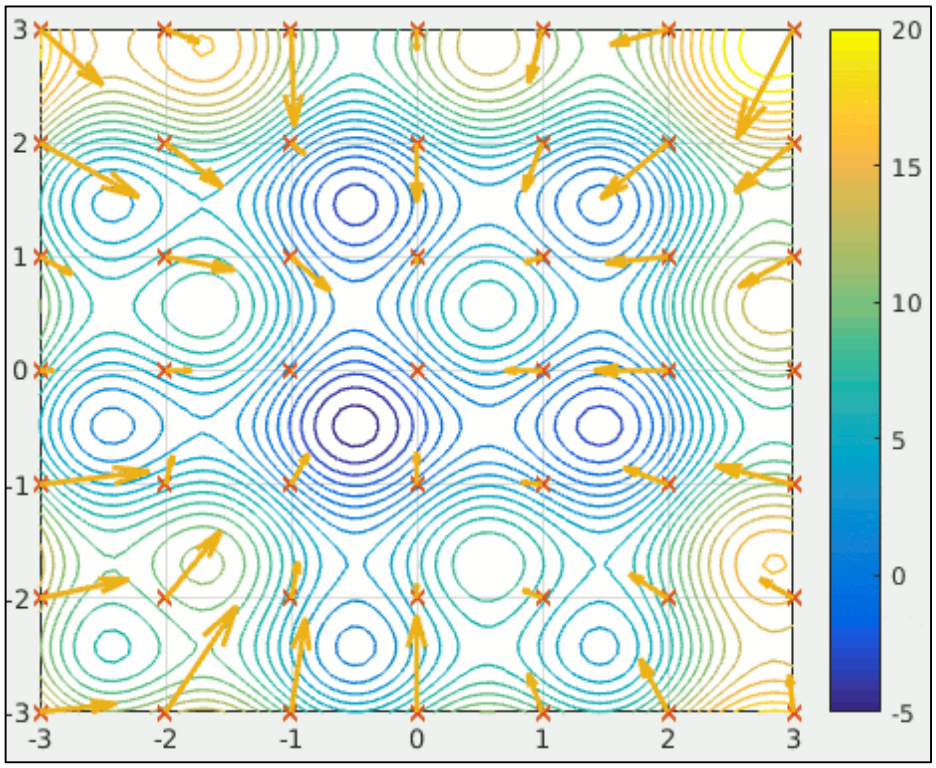


Figura 20: Un enjambre de partículas buscando el mínimo global de una función;
Fuente: Ephramac (2017).

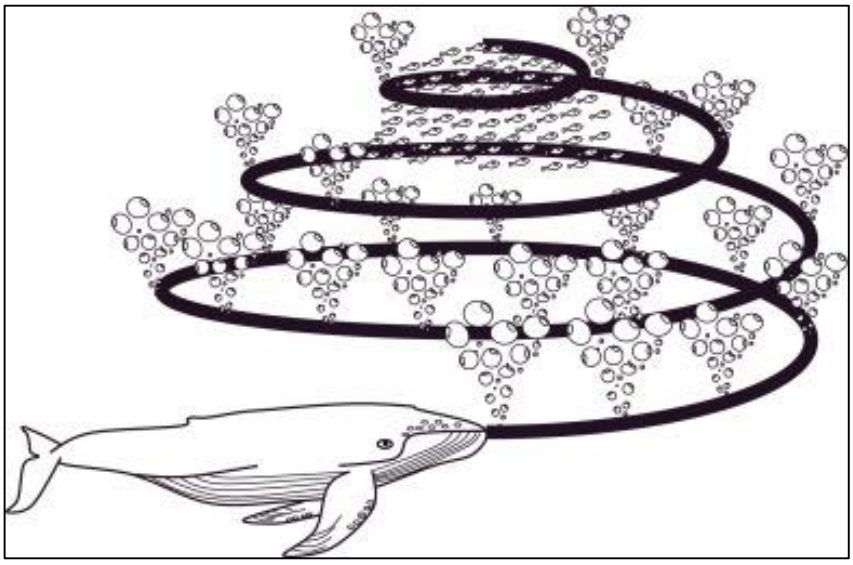


Figura 21: Ilustración del ataque de red de burbujas de una ballena;
Fuente: Mirjalili, S. y Lewis, A. (2016).

2.5 Técnicas de tratamiento de datos: aumentación de imágenes

Durante el punto 2.2 se habló del término trabajo de segmentación, el cual se asocia a la clasificación de imágenes mediante distintas etiquetas, para obtener una versión recortada de esta que omite detalles innecesarios y/o no asociados al elemento en cuestión. Un ejemplo de segmentación de imágenes es el siguiente, en el que un algoritmo es capaz de reconocer elementos concretos.

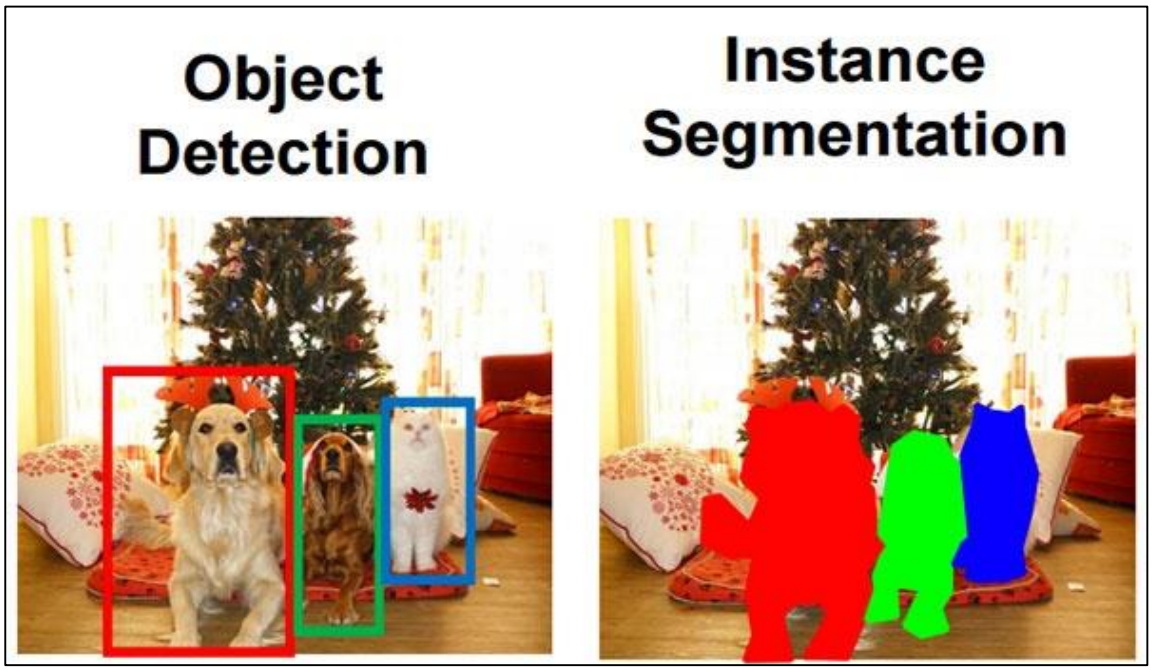


Figura 22: Ilustración de segmentación de imagen;
Fuente: Sharma, P. (2023).

Entre los aspectos que pueden determinar los algoritmos de visión artificial actualmente, se encuentran objetos, formas de estos, la dirección en la que se mueven, e inclusive las acciones que realizan. Sin embargo, en el mismo punto anterior se mencionó que la base de datos a trabajar ya cuenta con máscaras disponibles, es decir, ya se ha realizado un trabajo de segmentación de datos.

Entendiendo esto, es de interés preguntarse qué otro método o técnica puede aplicarse para trabajar estas imágenes antes o durante el proceso de la CNN, lo que lleva al término aumentación de imágenes, que se explicará a continuación.

La aumentación de datos puede definirse de dos formas, compatibles una con la otra y aplicables a la misma vez dependiendo de la necesidad presentada. Se describe, por Sudhakar (2017), como una técnica que es utilizada para expandir artificialmente una base de datos. Esto es de utilidad cuando se trabaja una base con muestras de poca variedad o cantidad, lo que puede llevar al fenómeno de sobreajuste de la red: esta sólo es capaz de aprender las características de un conjunto reducido de elementos y se vuelve incapaz de generalizar lo suficiente para identificar y clasificar correctamente un conjunto nunca antes visto.

Se describe, por la librería de TensorFlow, como una técnica que aumenta la diversidad de un conjunto de entrenamiento mediante la aplicación de transformaciones aleatorias (pero realistas), como la rotación de imágenes. Con ambas descripciones se puede entender que esta técnica podrá crear elementos nuevos para la base de datos, o modificar los existentes para aumentar la variedad de características, en búsqueda de un aprendizaje generalizado. Esta técnica de manejo de datos es de relevancia para el presente proyecto debido a que puede considerarse una forma de Parameter Tuning para muestras existentes, optimizándose con el uso de metaheurísticas como las estudiadas.

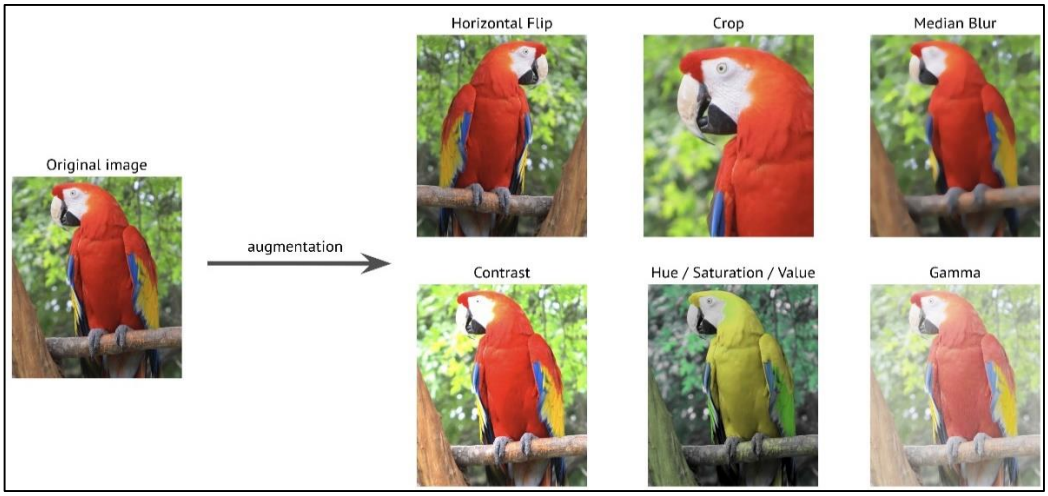
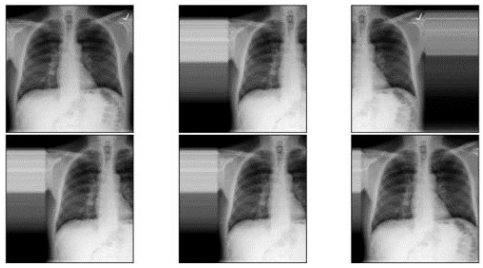
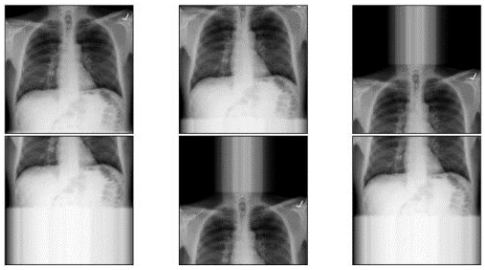
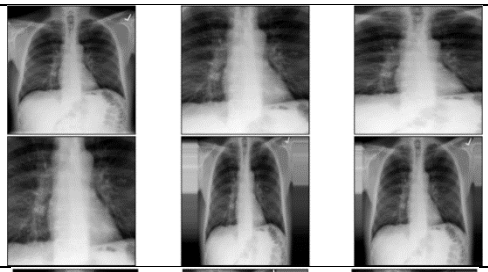
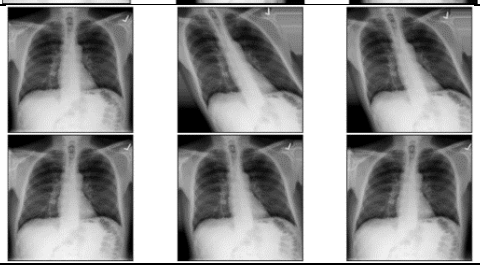


Figura 23: Ilustración de aumentación de imagen;
Fuente: Albumentations (2023).

A través del módulo keras de Tensorflow, se puede acceder a la clase ImageDataGenerator, la cual permite modificar una serie de características de manera aleatoria. Se destaca que las siguientes demostraciones fueron realizadas con valores elevados, y que en el caso práctico, se utilizarán menores escalas de cambio para evitar destruir características importantes.

Argumento o característica	**Descripción y efecto**	**Representación visual respecto a una radiografía**
Rango de Movimiento al Ancho ***(Width Shift Range)***	La imagen es desplazada en un rango horizontal mediante porcentajes.	
Rango de Movimiento al Alto ***(Height Shift Range)***	La imagen es desplazada en un rango vertical mediante porcentajes.	
Rango de Zoom ***(Zoom Range)***	Se realiza un acercamiento a la imagen estudiada.	
Rango de Inclinación **(Shear Range)**	Se estira la imagen en distintos ángulos para aumentar su variedad.	

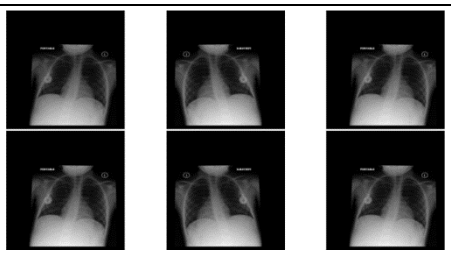
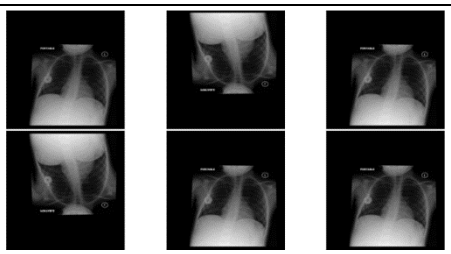
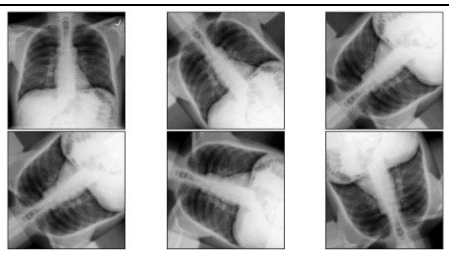
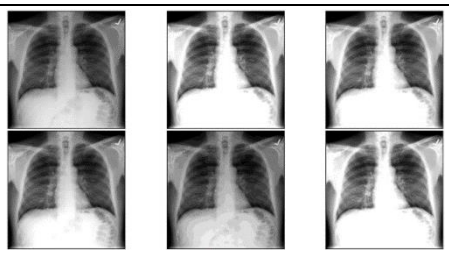
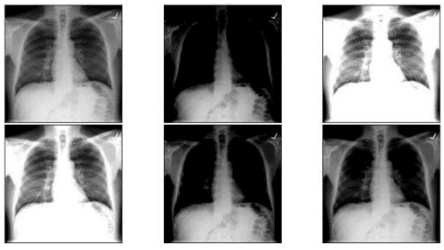
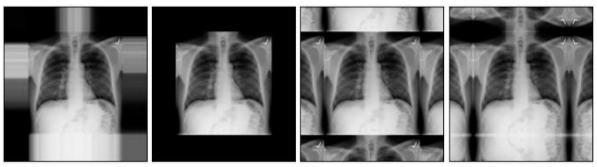
Volteo Horizontal **(Horizontal Flip)**	La imagen se voltea de manera horizontal.	
Volteo Vertical **(Vertical Flip)**	La imagen se voltea de manera vertical.	
Rango de Rotación **(Rotation Range)**	Se rota la imagen con el ángulo deseado.	
Rango de Brillo **(Brightness Range)**	Se aplica un cambio, oscureciendo o aclarando la imagen.	
Rango de Cambio de Canal **(Channel Shift Range)**	Similar a brillo, pero con énfasis en conservar el contraste.	
Modo de Llenado **(Fill Mode)**	Cómo se rellenan nuevos bordes. Se utilizó de manera fija el primer ejemplo.	

Tabla 5: Descripción y demostración de la aumentación de imagen; Fuente: Elaboración propia a través de información recopilada a través del sitio oficial de Tensorflow (2023).

2.6 Soluciones y trabajos de otros autores

Como se mencionó anteriormente en el punto 1.2.4, se deberá realizar un estudio y revisión bibliográfica respecto a diversos autores que hayan abordado la problemática de clasificación de imágenes asociada a radiografías o muestras clínicas, para lo cual se ha recolectado tres fuentes principales y comparativas. Con este punto, se buscará abordar los siguientes puntos:

- ¿Qué bases de datos han sido utilizadas?
- ¿Qué problemas a solucionar conllevan estas bases de datos?
- ¿Qué tipo de red o arquitectura se utilizó para solucionar estos problemas?
- ¿Qué tipo de metaheurística se utilizó en este problema de investigación?
- ¿Qué uso se le dio a esta metaheurística (ej: Feature Selection)?

La precisión de estas respuestas generará, a su vez, nuevas reflexiones.

- ¿Qué soluciones son las más populares?
- ¿Cuáles métodos de metaheurísticas se repiten más?
- ¿Cuáles no han sido explorados lo suficiente?
- ¿Qué porcentajes y aumentos presenciaron en sus implementaciones?

Figura 24: Referencias obtenidas en un estudio externo sobre la utilización de metaheurísticas; Fuente: Riaz, Bashir y Younas (2022).

2.6.1 Metaheuristic Optimization Through Deep Learning Classification of COVID-19 in Chest X-Ray Images (Abdelsamee et al., 2022)

Artículo que comienza debatiendo la deficiencia de pruebas tradicionales y kits de testeos utilizados comúnmente en hospitales, tales como los mencionados en el punto 1.3.2 a través de un recuadro comparativo. Procede a postular la necesidad de una forma de detectar COVID-19 de manera rápida y confiable a través de imágenes de radiografías. Entre los aspectos más importantes a destacar del acercamiento realizado por Abdelsamee et al. se encuentran:

- Propuesta de un modelo metaheurístico híbrido que combina PSO y DTO (Dipper Throated Optimization), junto con el uso de algoritmos de segmentación y aumentación de imágenes y la arquitectura VGG-16 para extracción de características, para así obtener resultados de alta precisión.
- El método de aplicación de metaheurística utilizado por esta propuesta fue Feature Selection, buscando conservar sólo características relevantes.
- Este artículo fue realizado bajo un problema de clasificación binaria: su interés fue específicamente generar un modelo que permitiese detectar la presencia o no de coronavirus en una radiografía, lo que lo convierte en un problema distinto al presentado en este proyecto (cuatro clases distintas, como se mencionó en el punto 2.2 al hablar de categorías).
- Utilizaron la misma base de datos preparada por Rahman et al. (2022) que se utilizará en este proyecto, sin embargo, su versión sólo utilizó radiografías de pacientes normales y pacientes con COVID-19 para el entrenamiento. Junto con esto, el número de muestras totales que poseían al momento de realizar el estudio era de 1.468 radiografías de COVID-19, y 1.382 radiografías de pacientes en condición normal. Al ser una cantidad reducida para trabajar, prosiguieron con un trabajo de aumentación de datos, clonando, copiando, rotando, contrastando y editando las radiografías para finalmente trabajar con 8.550 en total, y así fortalecer los resultados obtenidos por su modelo.

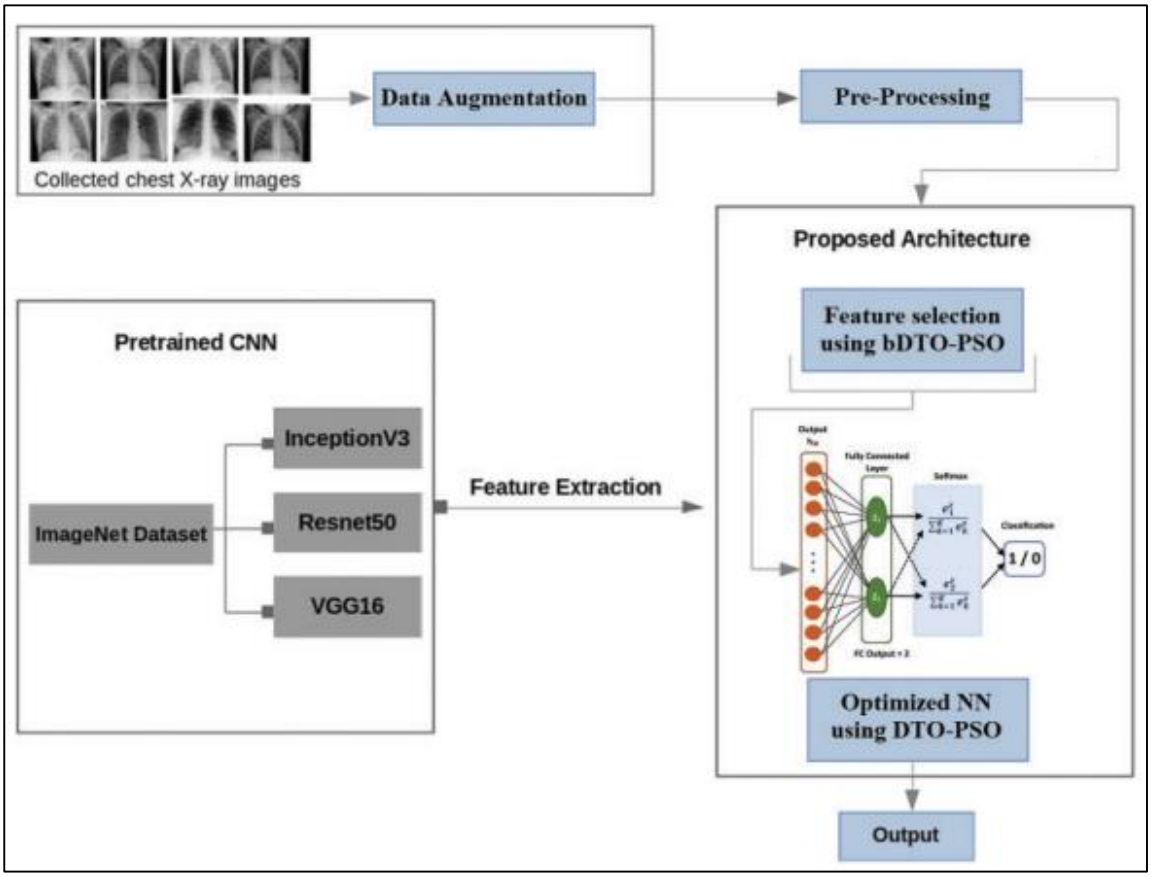


Figura 25: Arquitectura propuesta por los creadores del artículo anteriormente mencionado; Fuente: Abdelsamee et al. (2022).

- Realizaron, finalmente, un trabajo comparativo entre lo postulado y lo obtenido con otras técnicas o por otros autores, considerando características como error promedio y precisión en detección. Sin embargo, no hicieron énfasis en la duración o tiempo requerido para ejecutar estos algoritmos y generar los modelos.
- Destacan la superioridad y eficiencia de su modelo al comparar el valor final de precisión obtenido por su propuesta (0.998824912) en comparación a la utilización de WOA Binario (0.970564837), Optimizador de Lobo Gris (GWO) (0.97668557), Algoritmo Genético (GA) (0.986964618) y PSO Binario (0.979060555), Además, sugieren de seguir la experimentación con bases de datos de mayor complejidad.

2.6.2 Metaheuristics based COVID-19 detection using medical images: A review (Riaz, Bashir y Younas, 2022)

Artículo que explora la utilización de aprendizaje automático para detectar COVID-19 a través de radiografías, pero a su vez, también mediante tomografías computarizadas y otros medios. Se destaca que este artículo no propone ni codifica una arquitectura: es una revisión bibliográfica que compara y estudia a múltiples autores que han abarcado estos temas. Realiza diversos análisis estadísticos, asociados a tipos de procesamientos de imagen, datasets con mayor uso y popularidad (citaciones), diferencias entre trabajos de clasificación binaria y multiclase, referencias a CNNs y metaheurísticas bioinspiradas, y resultados obtenidos por los autores de los artículos que se estudian.

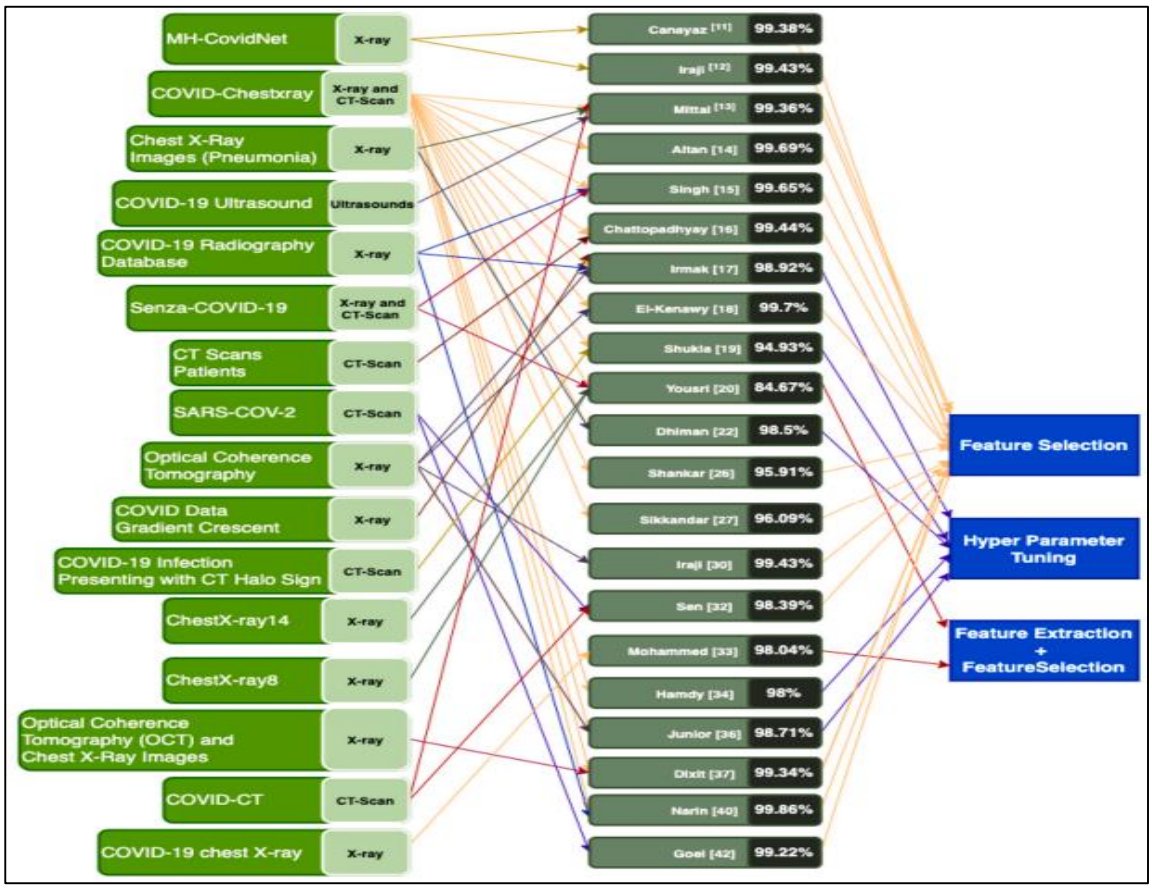


Figura 26: Vínculo entre datasets, número de citaciones, resultados obtenidos con estos por parte de distintos autores, y usos finales; Fuente: Riaz, Bashir y Younas (2022).

CNN architecture	CNN architecture variants	Reference (Use case is COVID-19 detection)	Accuracy %
VGG [54]	VGG19	[6,32]	99.38, 98.71
	VGG16	[23,27]	96.09, 94.17
ResNet [55]	ResNet18	[11]	99.44
	ResNet50	[13,32,36]	99.7, 94.17, 99.86
	ResNet101	[17,36]	98.5, 99.86
EfficientNet [59]	EfficientNet-B0	[9]	99.69
	EfficientNet-B1	[16]	97.62
AlexNet [60]	AlexNet	[8,43]	99.5, 99.36
	AlexNet Modifier	[42]	98.55
CheXNet [56]	CheXNet	[18]	97.94
DenseNet [57]	DenseNet201	[18]	97.94
LeNet [61]	LeNet-5	[35]	99.11
GoogLeNet [62]	GoogLeNet	[14]	98.38
Inception [58]	Inception	[63]	99.8
	InceptionV3	[27,32]	94.17, 98.71
Customised layers	Customised layers	[37,64]	99.99, 97.78
Xception [65]	Xception	[27]	94.17

Tabla 6: Accuracy (precisión) reportada por distintas arquitecturas de redes neuronales convolucionales en distintas referencias; Fuente: Riaz, Bashir y Younas (2022).

Ref.	Nature Inspired Algorithm	Use case	Accuracy %	Precision %	Recall %	F-Score %
[6]	MH-CovidNet	Feature Selection	99.38	100	–	99.07
[43]	PSO-Guided-WOA		99.5	100	100	–
[63]	FO-MPA		**99.8**	–	–	99
[8]	KIGSA-C		99.36	100	100	99.37
[9]	CSSA		99.69	99.62	99.44	99.53
[10]	HSGO		99.65	99.66	99.65	99.65
[11]	CGRO		99.31	99	100	98
[13]	ASSOA		99.7	–	–	–
[7]	BDE		99.43	99.16	99.57	–
[68]	OGA		92.6	92.5	92.5	–
[38]	WOA		99.22	97.78	99.78	98.79
[78]	MODE		93	91	90.5	90
[22]	SSA		95.91	95.97	95.1	95.29
[23]	GA		96.09	93.85	91.54	90
[40]	GA		**100**	100	100	100
[69]	GA		96	76	64	–
[26]	BD		99.43	99.57	99.16	–
[27]	Two-Tier-FS -Coalition game		94.17	87	98	–

Tabla 7: Casos de uso de algoritmos y metaheurísticas bioinspiradas junto con el valor de precisión que estas representan (extracto); Fuente: Riaz, Bashir y Younas (2022).

2.6.3 Accurate detection of COVID-19 using deep features based on X-Ray images and feature selection methods (Narin, A., 2021)

Artículo realizado y publicado durante los primeros años de la pandemia, buscando generar sistemas de detección y diagnóstico automático de COVID-19 mediante las técnicas mencionadas en este informe previamente. Se destaca su acercamiento mediante tres tipos de redes neuronales convolucionales (ResNet50, ResNet101 e InceptionResNetV2), y la utilización de un problema de tres clases (pacientes en condición normal, con coronavirus y con neumonía).

- La utilización de una combinación entre distintas bases de datos; se menciona la recopilación de 219 radiografías de pacientes con condición de coronavirus, 1341 individuos en condición normal y 1345 pacientes con condición de neumonía. Este problema realiza la utilización de metaheurísticas con Feature Selection (selección de características)

Utilizan, además, algoritmos bioinspirados como lo son PSO y ACO, y el apoyo de SVM (máquinas de vectores de soporte) para obtener clasificaciones de alto puntaje (cercanos a 99.83%) en la base de datos trabajada por ellos. Además, se menciona en este artículo la diferencia en tiempos de ejecución en cuanto a segundos, de manera separada para cada algoritmo y después combinada.

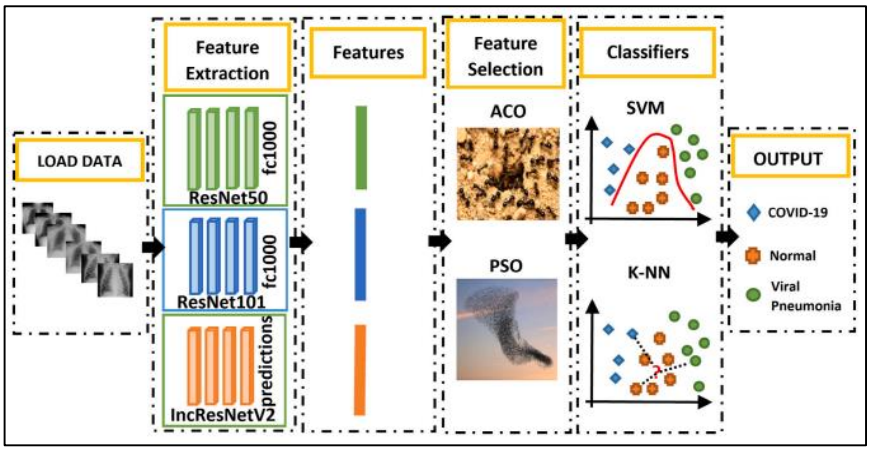


Figura 27: Arquitectura propuesta mediante las distintas CNNs y metaheurísticas bioinspiradas; Fuente: Narin, A. (2021).

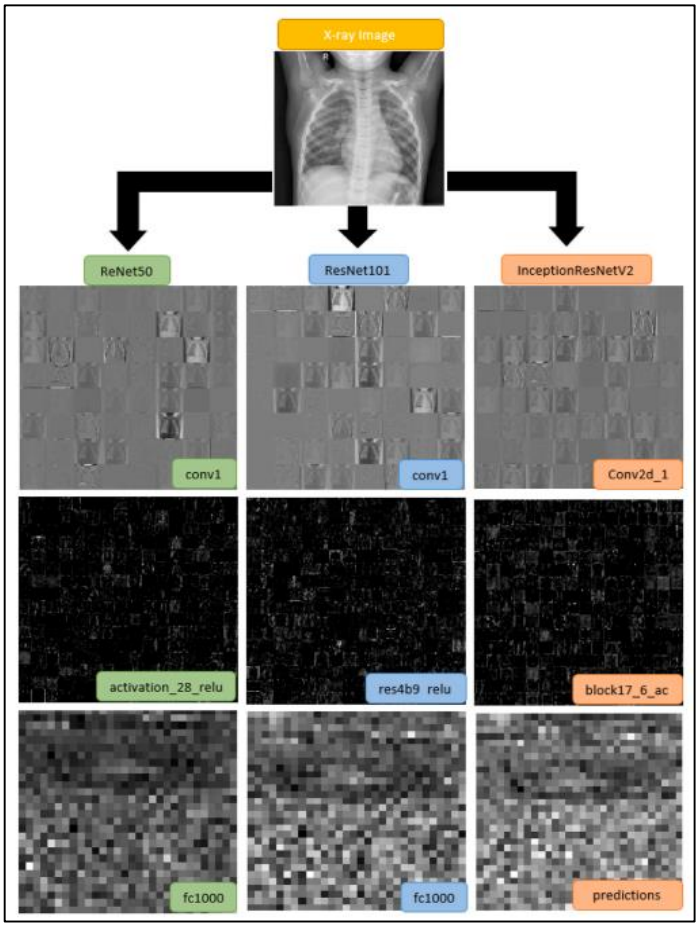


Figura 28: Datos de salida de los tres diferentes modelos propuestos en este artículo; Fuente: Narin, A. (2021).

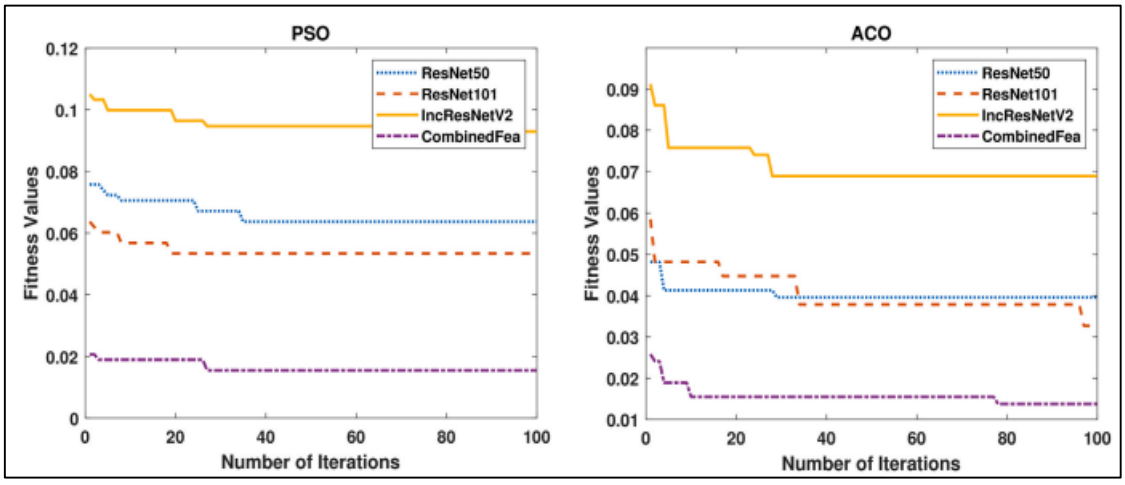


Figura 29: Valores de funciones fitness (candidatas a solucionar un problema) para PSO y ACO en el artículo de referencia; Fuente: Narin, A. (2021).

3 DISEÑOS PROPUESTOS DURANTE EL PRIMER SEMESTRE

Durante la elaboración del proyecto de título en el primer semestre del presente año 2023, se esperó que en este proyecto se desarrollara una arquitectura eficaz, útil, que contase con los siguientes aspectos.

- La carga, lectura y procesado de la base de datos de radiografías mencionada, utilizando el proceso de segmentación de radiografías para considerar sólo el área de los pulmones y tórax, descartando aquellos datos correspondientes a zonas no asociadas como lo pueden ser hombros, cuello, brazos o estómago.

- La partición de esta base de datos en conjuntos de entrenamiento (que se utilizan para enseñar características al algoritmo), validación (que se utilizan para comprobar un correcto aprendizaje durante el entrenamiento, calculando el rendimiento y reajustando parámetros) y prueba (utilizado para evaluar el rendimiento del modelo obtenido frente a datos nuevos). Estos tres se repartirían con tamaños a considerar entre los siguientes:

    - 80% de entrenamiento, 10% de validación, 10% de prueba.
    - 70% de entrenamiento, 15% de validación, 15% de prueba.
    - 60% de entrenamiento, 20% de validación, 20% de prueba.
    - Otro tipo de variaciones específicamente para validación y prueba.

- Posterior a la partición, realizar la carga de la red neuronal convolucional pre-entrenada, en este caso, VGG-19, y las funciones asociadas a esta de acuerdo a lo mostrado con anterioridad.

Se propuso, además, repetir los siguientes pasos varias veces para probar distintas configuraciones y valores de hiperparámetros, buscando obtener la versión más óptima de qué hiperparámetros modificar y a qué escala.

- Ejecución de algoritmos y metaheurísticas bioinspiradas para ajustarlos.
- Ejecución del modelo mediante los parámetros establecidos, junto con el entrenamiento de este (con una duración de ejecución de media hora).

Buscando que una vez se terminase este proceso, se pudiese realizar una visualización y evaluación de los resultados obtenidos. Entre esto, se destacaron:

- Gráficos para evidenciar el aumento la caída de precisión y errores.
- Ilustraciones y matrices de valores para observar resultados.
- Creación de tablas comparativas entre lo obtenido personalmente.
- Creación de tablas comparativas entre lo propio y de otros autores.

Como objetivos a desarrollar finalizando el primer semestre, se destacan los siguientes problemas e incógnitas planteadas.

- Asegurar la reproducibilidad del algoritmo: que bajo las mismas semillas o configuraciones de procesamiento, pudiesen obtenerse los mismos resultados siempre, y así, evitar cualquier tipo de ruido o interferencia que pudiese llevar a obtener conclusiones erróneas respecto a qué beneficia y qué no beneficia al algoritmo.
- Estudiar y evaluar qué uso y aplicación de los dos estudiados (Selección de características o Ajuste de Hiperparámetros).
- En el segundo caso, qué tipo de hiperparámetros debía ser modificado, ya fuesen los mencionados anteriormente, u otros aparte asociados como tal a la estructura del modelo.
- Independiente del caso, qué tipo de técnica metaheurística utilizar para el ajuste de estos, ya fuese una de las seis mencionadas con anterioridad, o una completamente distinta.
- Cómo aplicar las técnicas de ajuste de hiperparámetros al algoritmo, por ejemplo, utilizar ciclos repetitivos para entrenar, probar y reajustar.

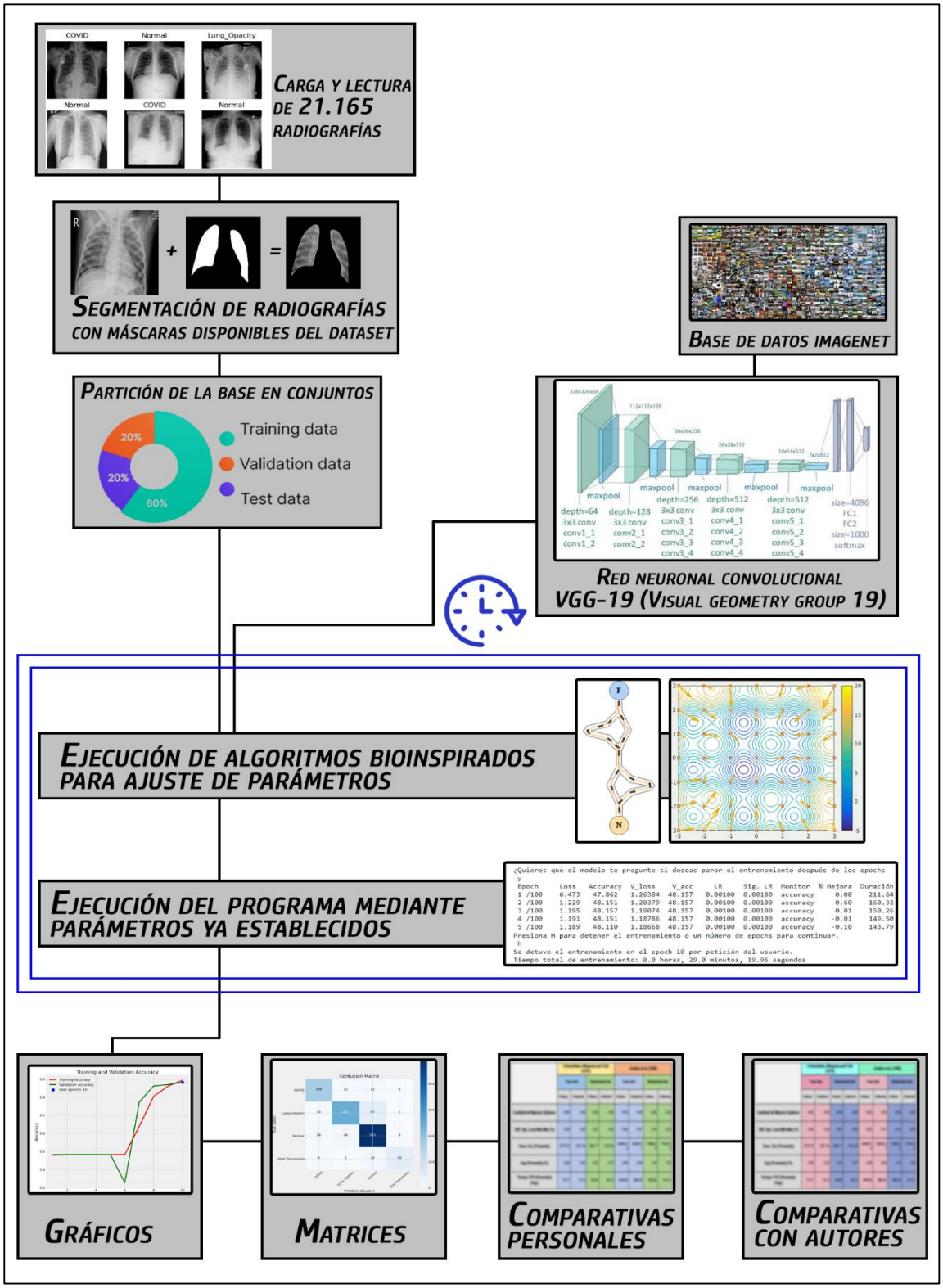


Figura 30: Arquitectura propuesta al cierre del primer semestre y primera parte del proceso de proyecto de título; Fuente: Elaboración propia (2023).

4 DESCRIPCIÓN DE CONCEPTOS Y GRÁFICOS A EVALUAR

Para continuar a la segunda fase del desarrollo y evaluación, de la propuesta de red neuronal convolucional con utilización de metaheurísticas, es necesario entender qué métricas y técnicas se utilizarán para observar el funcionamiento del algoritmo, analizar su correcto funcionamiento, predicciones y resultados.

En cuanto a terminología asociada a modelos, se considerará lo siguiente:

Término	Descripción
Conjunto de Entrenamiento (Training)	Recordando lo mencionado en el punto tercero, es generalmente la partición más grande de la base de datos, utilizada para entrenar un modelo o red neuronal, obteniendo valores que permitan a este aprender características y rasgos.
Conjunto de Validación (Validation)	La partición utilizada para mejorar el rendimiento del modelo, reajustándolo después de cada ciclo (Epoch).
Conjunto de Prueba (Test)	La partición utilizada para comprobar el rendimiento de un modelo una vez ha terminado de generarse. Contiene datos que este nunca ha visto antes, y se utiliza para observar su capacidad de predecir y clasificar con casos nuevos.
Loss (Pérdida)	Medida utilizada para observar cómo se ajusta un modelo a datos, ya sean de entrenamiento, validación o prueba, en específico tomando en cuenta los errores cometidos.
Accuracy (Precisión)	Medida utilizada para observar cómo se ajusta un modelo a datos, en específico tomando en cuenta los aciertos.
Learning Rate (Ratio de aprendizaje)	Hiperparámetro que ajusta la rapidez con la que un modelo se adapta a un problema; por ejemplo, la cantidad de actualizaciones a los pesos (características aprendidas).

Monitor	En la estructura utilizada, monitor será un valor que se referirá al tipo de dato con el que se ajustará el ratio de aprendizaje durante el entrenamiento. Este podrá ser la pérdida de validación o la precisión total, dependiendo de los resultados que se estén obteniendo en el momento.
Tamaño de lote (Batch Size)	Número de instancias (elementos) en cada lote de entrenamiento. Es un Hiperparámetro que controla el número de muestras que se trabajaran antes de que se actualicen los parámetros internos del modelo al final de cada ciclo (Epoch).
Epoch	Iteración completa de entrenamiento de un modelo de aprendizaje automático en un conjunto de datos (cMax, 2023). El modelo recibe distintos ejemplos y ajusta sus parámetros, pesos y funciones en base a los errores obtenidos.

Tabla 8: Terminología a considerar respecto a los modelos estudiados;

Fuente: Elaboración propia (2023).

En cuanto a gráficos, se considerarán los siguientes términos y visualizaciones:

Término	Descripción
Underfitting (Infra ajuste)	Fenómeno que se produce cuando un modelo no captura la complejidad de los datos, y por lo tanto, no aprende ni se ajusta adecuadamente a estos; es demasiado simple.
Fitting (Ajuste)	Ajuste ideal y esperado del modelo. Este logra capturar la complejidad de los datos y aprender cómo clasificarlos de acuerdo a sus características.
Overfitting (Sobre ajuste)	Fenómeno que se produce cuando un modelo se ajusta de sobremanera a los datos de entrenamiento, lo que produce un aprendizaje específico y no generalizado de características.

Tabla 9: Terminología a considerar respecto a gráficos de utilidad;

Fuente: Elaboración propia (2023).

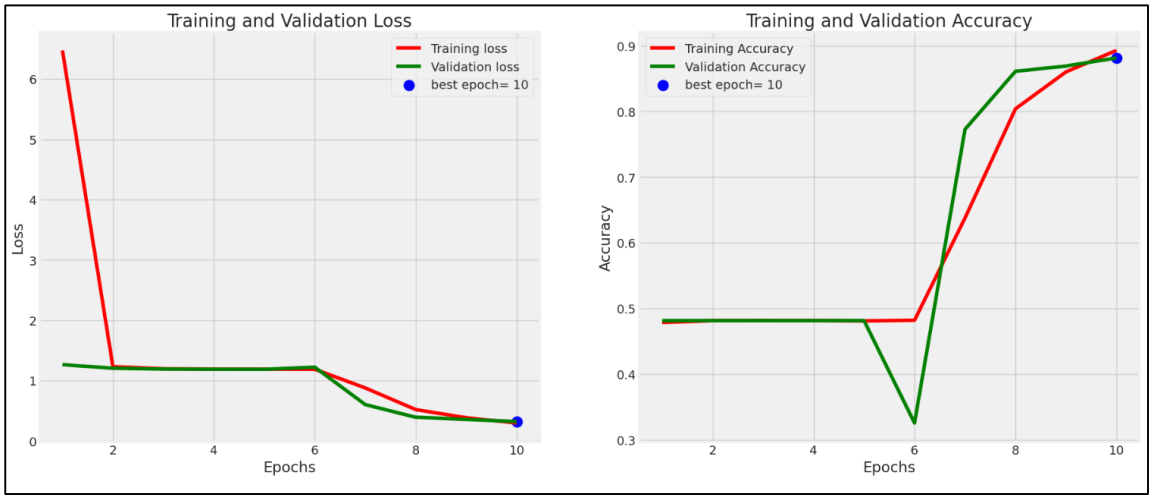


Figura 31: Curvas de error y precisión a lo largo de los distintos epochs, mostrando los peores y mejores puntos de un modelo; Fuente: Elaboración propia (2023).

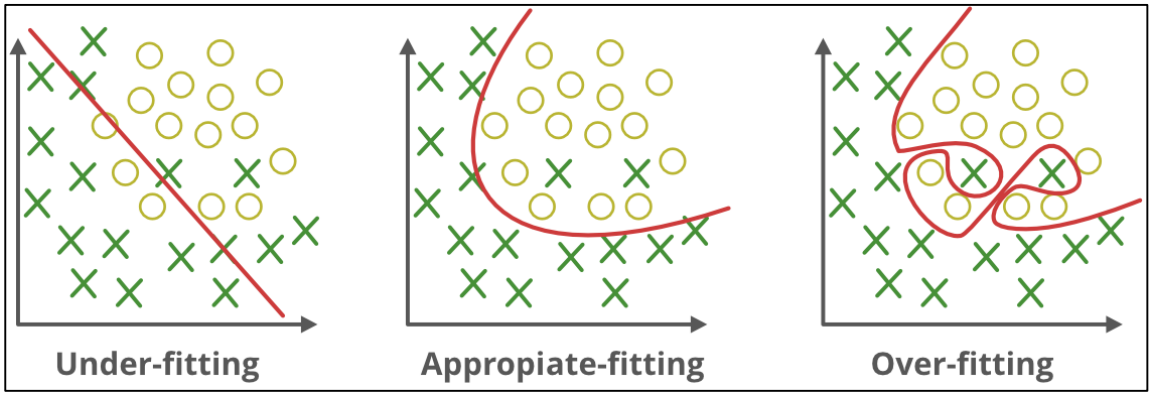


Figura 32: Ejemplificación de los tres casos mencionados: underfitting, fitting y overfitting; Fuente: Geeksforgeeks. (2023).

Finalmente, en cuanto a resultados, se considerará la siguiente visualización, junto al significado de los valores que la componen.

Término	Descripción
Matriz de confusión	Recuadro que describe el rendimiento de un modelo utilizando los datos de prueba. Permite ver con facilidad dónde ocurren las confusiones (errores) del modelo.

Verdadero Positivo (VP)	Tomando de ejemplo un problema de clasificación binaria, en donde un algoritmo debe escoger entre cero y uno: A un elemento de clase uno (verdadero), el modelo lo ha predicho también como uno (verdadero). Es un caso ideal.
Verdadero Negativo (VN)	A un elemento de clase cero (falso), el modelo lo ha predicho también como cero (falso). Es un caso ideal.
Falso Positivo (FP)	A un elemento de clase cero (falso), el modelo lo ha predicho erróneamente como uno (verdadero). Es un caso a evitar.
Falso Negativo (FN)	A un elemento de clase uno (verdadero), el modelo lo ha predicho erróneamente como cero (falso). Caso a evitar.
Precisión (Precision)	Definido como una fracción de los resultados que son relevantes, entregada por la siguiente fórmula. $$\boldsymbol{Precisión} = \frac{\boldsymbol{VP}}{\boldsymbol{VP} + \boldsymbol{FP}}$$
Accuracy (Exactitud)	El porcentaje total de elementos clasificados correctamente. Para obtenerlo, se considera una línea diagonal hacia la derecha, destacando solamente los verdaderos positivos y verdaderos negativos. $$\boldsymbol{Exactitud} = \frac{\boldsymbol{VP} + \boldsymbol{VN}}{\boldsymbol{VP} + \boldsymbol{FP} + \boldsymbol{FN} + \boldsymbol{VN}}$$
Recall (Sensibilidad)	El número de elementos identificados correctamente como positivos, del total de positivos verdaderos. $$\boldsymbol{Sensibilidad} = \frac{\boldsymbol{VP}}{\boldsymbol{VP} + \boldsymbol{FN}}$$

f1-score (Puntuación F1)	Valor final de precisión en una base de datos. Definido como un valor que permite obtener la habilidad predictiva del modelo examinando su rendimiento con cada clase de manera individual, este se obtiene con: $$\boldsymbol{Puntuación\ F1} = \frac{\boldsymbol{VP}}{\boldsymbol{VP} + \frac{\mathbf{1}}{\mathbf{2}}(\boldsymbol{FP} + \boldsymbol{FN})}$$

Tabla 10: Terminología a considerar respecto a evaluaciones;
Fuente: Elaboración propia (2023).

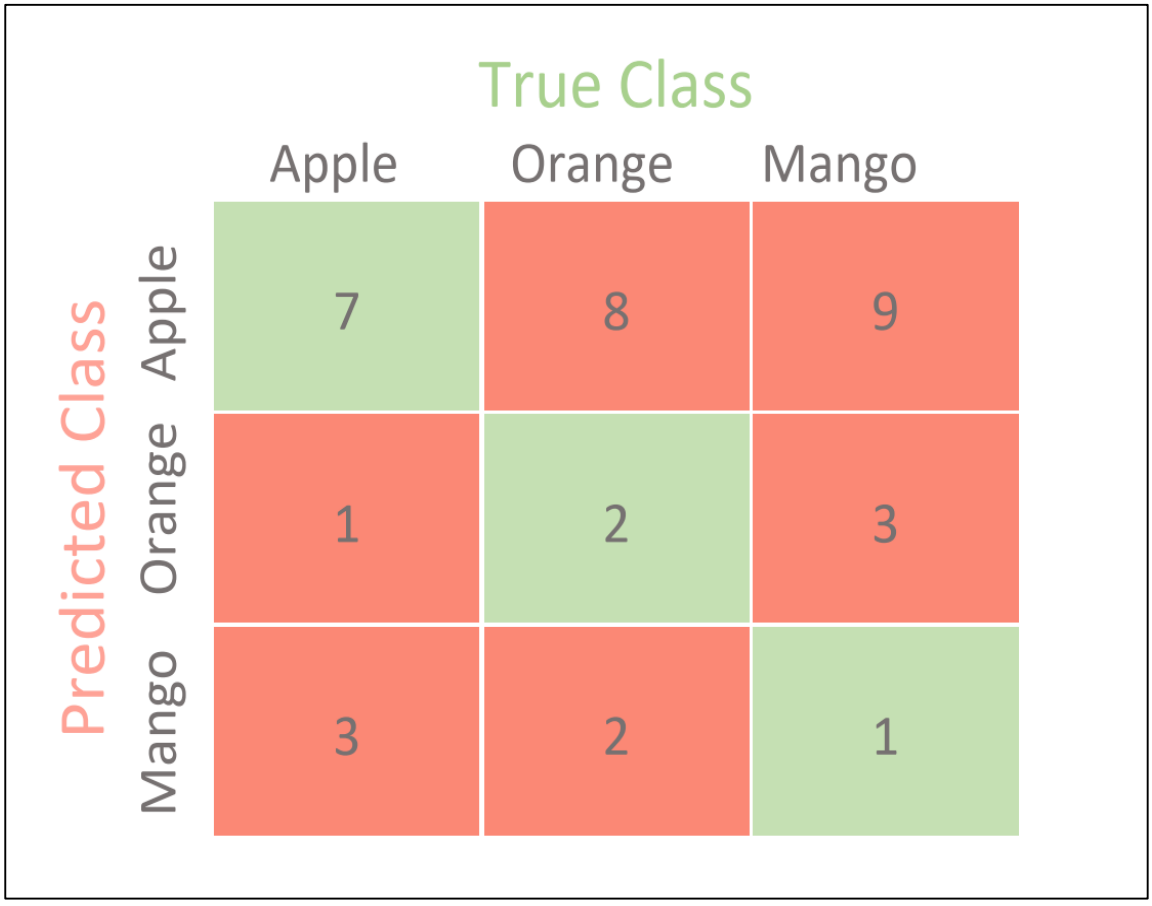


Figura 33: Ejemplo de matriz de confusión, en donde los cuadros rojos representan predicciones erróneas; Fuente: Mohajon (2020).

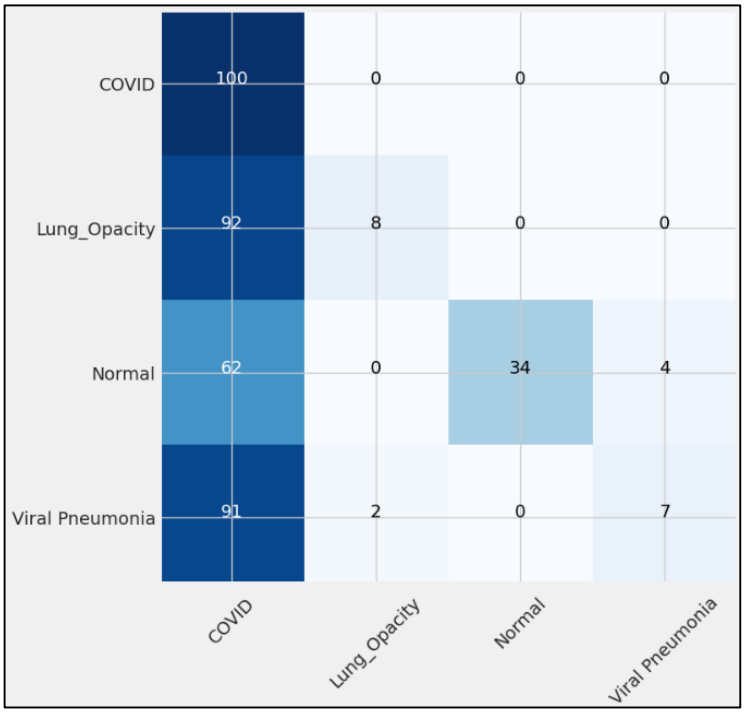


Figura 34: Ejemplo de matriz de confusión, en donde un modelo específico no es capaz de clasificar correctamente tres categorías; Fuente: Elaboración propia (2023).

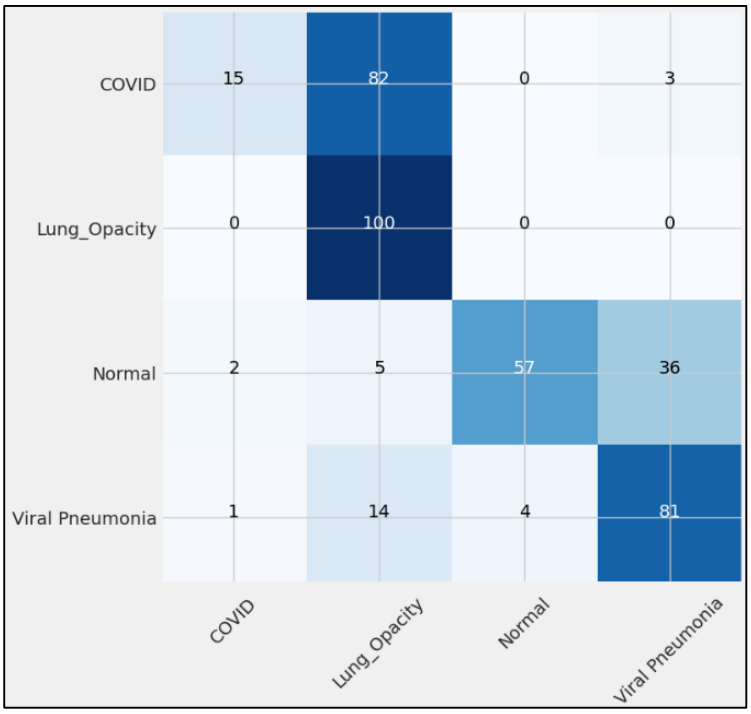


Figura 35: Ejemplo de matriz de confusión, en donde un modelo específico obtiene un rendimiento relativamente mediocre para dos clases; Fuente: Elaboración propia (2023).

	precision	recall	f1-score
COVID	0.29	1.00	0.45
Lung_Opacity	0.80	0.08	0.15
Normal	1.00	0.34	0.51
Viral Pneumonia	0.64	0.07	0.13
accuracy			0.37
macro avg	0.68	0.37	0.31
weighted avg	0.68	0.37	0.31

	precision	recall	f1-score
COVID	0.54	0.78	0.64
Lung_Opacity	0.46	0.77	0.57
Normal	0.96	0.53	0.68
Viral Pneumonia	0.76	0.25	0.38
accuracy			0.58
macro avg	0.68	0.58	0.57
weighted avg	0.68	0.58	0.57

	precision	recall	f1-score
COVID	0.83	0.15	0.25
Lung_Opacity	0.50	1.00	0.66
Normal	0.93	0.57	0.71
Viral Pneumonia	0.68	0.81	0.74
accuracy			0.63
macro avg	0.74	0.63	0.59
weighted avg	0.74	0.63	0.59

	precision	recall	f1-score
COVID	0.91	0.94	0.93
Lung_Opacity	0.90	0.86	0.88
Normal	0.91	0.94	0.93
Viral Pneumonia	0.96	0.87	0.91
accuracy			0.91
macro avg	0.92	0.90	0.91
weighted avg	0.91	0.91	0.91

Figura 36: Distintos valores obtenidos a través del reporte de clasificación asociado a la matriz de confusión, ordenados de peor a mejor; Fuente: Elaboración propia (2023).

Finalmente, y para cerrar este punto, se revisará que a través de la inclusión de técnicas de segmentación, aumentación y utilización de metaheurísticas, se pueda dar uno de los siguientes cuatro casos, en comparación a una prueba sin ajuste alguno de hiperparámetros mediante técnicas de metaheurística.

Mejor caso posible **Disminuye el tiempo de ejecución** **Aumenta el valor de precisión**	**Primer caso aceptable** **Disminuye el tiempo de ejecución** **Disminuye el valor de precisión**
Segundo caso aceptable **Aumenta el tiempo de ejecución** **Aumenta el valor de precisión**	**Peor caso posible** **Aumenta el tiempo de ejecución** **Disminuye el valor de precisión**

Tabla 11: Terminología a considerar respecto a casos;
Fuente: Elaboración propia (2023).

5 ESPECIFICACIONES TÉCNICAS Y DE USO DE ACELERADORES

Como se mencionó en el punto 2.3, se utilizará la plataforma de conocida como Kaggle para realizar las pruebas y los modelos deseados. Esta plataforma cuenta con la opción de utilizar aceleradores de GPU, asociados al procesamiento de imagen y redes neuronales, y TPU, asociados además a unidades de tensores. Por lo tanto, es necesario especificar los componentes y diferencias que existen entre estas dos alternativas y la utilización de Kaggle sin seleccionar GPU o TPU.

Componente	Sin acelerador	GPU P100	TPU VM v3-8
Arquitectura	x86_64	x86_64	x86_64
Modo de CPU	32-bit, 64-bit	32-bit, 64-bit	32-bit, 64-bit
CPU(s)	4	4	96
Lista de CPU(s) en línea	0-3	0-3	0-95
Hilos en cada núcleo	2	2	2
Núcleos en cada zócalo	2	2	24
Zócalos	1	1	2
Nodos de acceso a memoria no uniforme	1	1	2
Nombre de modelo	Intel (R) Xeon (R) CPU @ 2.20GHz	Intel (R) Xeon (R) CPU @ 2.00GHz	Intel (R) Xeon (R) CPU @ 2.00GHz
Stepping (revisión)	0	3	3
MHz del CPU	2199.998	2000.178	2000.172
BogoMIPS (medida sobre la velocidad del sistema)	4399.99	4000.35	4000.34
Caché L1d	32K	32K	1.5 MiB
Caché L1i	32K	32K	1.5 MiB
Caché L2	256K	1024K	48 MiB
Caché L3	56320K	39424K	77 MiB

CPUs para el nodo 0 de acceso a memoria no uniforme	0-3	0-3	0-23, 48-71
CPUs para el nodo 1 de acceso a memoria no uniforme	No utiliza	No utiliza	24-47, 72-95
Requerimiento de cola de espera	No utiliza	No utiliza	Sí
Tiempo transcurrido en la cola de espera (peor caso posible)	No utiliza	No utiliza	50-70 minutos
Tiempo transcurrido en la cola de espera (mejor caso posible)	No utiliza	No utiliza	15-20 minutos
Peores disponibilidades	No utiliza	No utiliza	En el horario diurno (UTC-03:00)
Mejores disponibilidades	No utiliza	No utiliza	En el horario nocturno (UTC-03:00)
¿Requiere configuración manual para su uso?	No utiliza	No utiliza	Configurar e importar funciones
Incompatibilidades	No presenta	No presenta	Capas de procesamiento de Keras
Límite de uso semanal	No restringe	30 horas	20 horas
Rapidez de ejecución	Más lento	Estándar	Más rápido

Tabla 12: Especificaciones técnicas de Kaggle y sus aceleradores;

Fuente: Elaboración propia en base a la información disponible de Kaggle (2023).

6 DESARROLLO Y CODIFICACIÓN DE LA PROPUESTA FINAL

6.1 Programación y evaluación de la base del modelo

Para desarrollar la propuesta de este informe, se comenzó con la codificación como tal de una red neuronal convolucional básica mediante la plataforma Kaggle. Se realizan los siguientes pasos, cuyos códigos podrán ser visualizados en la sección Apéndice (13) finalizando el contenido.

- Importar los módulos requeridos para las funciones y mencionados con anterioridad; entre estos se vuelve a destacar librerías como NumPy, Pandas, Matplotlib, OpenCV, TensorFlow, Scikit-learn y Seaborn.
- Junto con esto, se importan librerías asociadas a la lectura de archivos de sistema, para poder cargar las bases de datos, y fechas, para poder hacer registro de la duración de algoritmos.
- Se codifican las funciones responsables de hacer lectura de archivos, generar dataframes con estos, y finalmente dividirlos en conjuntos de entrenamiento, validación y prueba, finalmente creando el modelo y su estructura como tal. Al ser una versión básica, no se consideró ningún paso asociado a lo siguiente.
    - Ajustar la reproducibilidad del modelo.
    - Segmentación de imágenes y datos.
    - Ajuste de hiperparámetros y aplicación de metaheurística.

  Por lo tanto, estos aspectos son trabajados continuando la primera revisión del funcionamiento del algoritmo. Se destaca la decisión de escoger la división 80%, 10%, 10% planteada en el punto 3 del informe, junto con la utilización, por defecto, de un tamaño de lote de 16.
- En base al modelo propuesto por Hafez (2023), se generan funciones de llamada de vuelta (Callback) que permiten reajustar distintos valores durante el entrenamiento y a su vez entregar información valiosa durante este. De la función de llamada de vuelta, se destaca lo siguiente:

- El registro del inicio del tiempo en la función de inicio del entrenamiento (on_train_begin).
- El cálculo de la duración del entrenamiento en la función de fin del entrenamiento (on_train_end).
- La obtención del valor de precisión y pérdida al final de cada lote calculado del algoritmo (on_train_batch_end).
- El reinicio del tiempo en cada epoch, permitiendo obtener la duración individual de estos (on_epoch_begin).
- Obtención de valores de ratio de aprendizaje, precisión y pérdida de entrenamiento, y comparación de estos valores, para reajustar el learning rate de acuerdo a los resultados y un límite (threshold) establecido previamente. Esta misma función se encargara finalmente de comprobar si se ha llegado al final del entrenamiento, y consultar al usuario si este desea continuar con una cantidad de epochs o desea finalizar este proceso (on_epoch_end).

- Se codifican funciones asociadas al muestreo y visualización de datos y resultados trabajados y obtenidos por las pruebas. Se destacan:
  - Gráficos de línea para visualizar la precisión y pérdida en entrenamiento y validación, y gráficos de matriz de confusión.

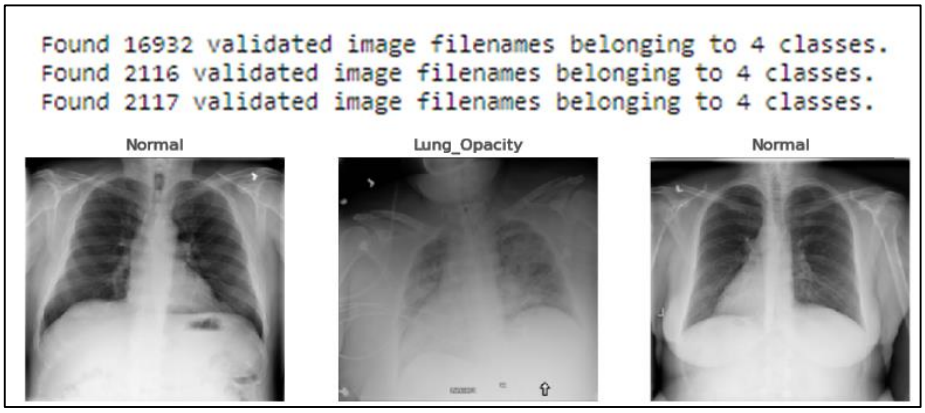


Figura 37: Mensaje de confirmación entregado por la plataforma al leer la base de datos y clasificar de acuerdo a clases y conjuntos; Fuente: Elaboración propia (2023).

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 512)               20024384
 dense (Dense)               (None, 4)                 2052
=================================================================
Total params: 20,026,436
Trainable params: 20,026,436
Non-trainable params: 0
_________________________________________________________________
```

Epoch	Loss	Accuracy	V_loss	V_acc	LR	Sig. LR	Monitor	% Mejora	Duración
1 /100	3.014	57.123	0.80121	69.093	0.00100	0.00100	accuracy	0.00	213.27
2 /100	0.679	74.020	0.61263	77.079	0.00100	0.00100	accuracy	29.58	142.61
3 /100	0.528	80.079	0.52879	80.057	0.00100	0.00100	accuracy	8.19	142.57
4 /100	0.455	83.020	0.45022	84.121	0.00100	0.00100	accuracy	3.67	142.65
5 /100	0.398	85.371	0.47965	83.885	0.00100	0.00100	accuracy	2.83	142.93
6 /100	0.326	88.111	0.34128	88.752	0.00100	0.00100	accuracy	3.21	142.50
7 /100	0.279	89.919	0.29399	89.981	0.00100	0.00100	accuracy	2.05	142.29
8 /100	0.248	91.064	0.31870	89.319	0.00100	0.00100	val_loss	-8.41	142.02
9 /100	0.227	91.962	0.26006	90.974	0.00100	0.00100	val_loss	11.54	142.20
10 /100	0.210	92.588	0.24623	90.832	0.00100	0.00100	val_loss	5.32	142.10

Figura 38: Estructura generada del modelo, y datos de entrenamiento (con un total de veinticinco minutos y nueve segundos mediante GPU P100); Fuente: Elaboración propia (2023).

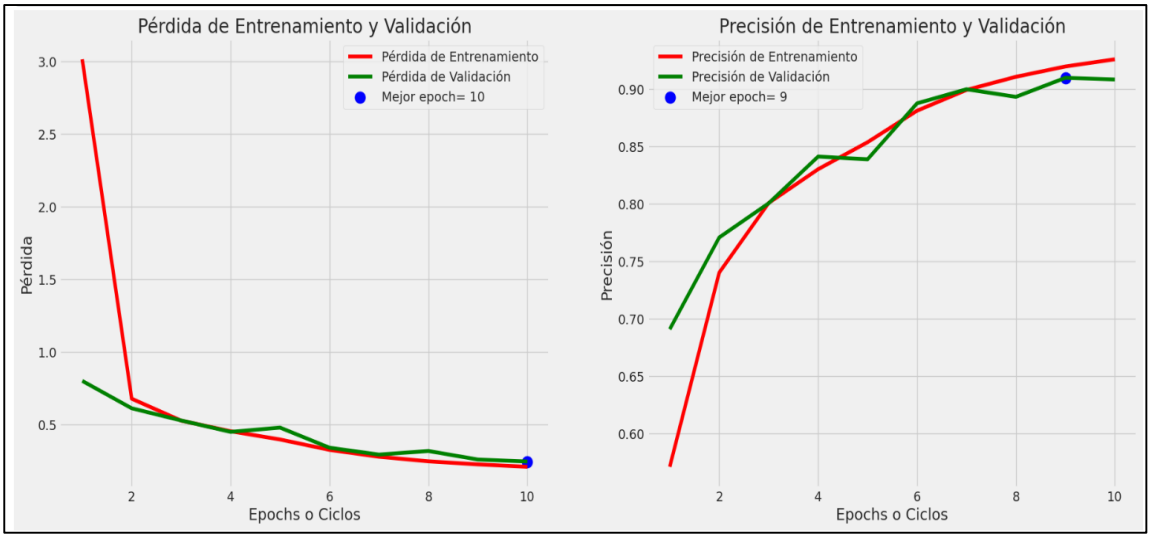


Figura 39: Gráfico de líneas para detallar los cambios de precisión y pérdida; Fuente: Elaboración propia (2023).

```
29/29 [==============================] - 2s 79ms/step - loss: 0.1885 - accuracy: 0.9246
29/29 [==============================] - 2s 75ms/step - loss: 0.2134 - accuracy: 0.9138
29/29 [==============================] - 11s 384ms/step - loss: 0.1911 - accuracy: 0.9339
Pérdida   de Entrenamiento:  0.188498392701149
Precisión de Entrenamiento:  0.9245689511299133
--------------------
Pérdida   de Validación:     0.21337145566940308
Precisión de Entrenamiento:  0.9137930870056152
--------------------
Pérdida   de Prueba:         0.19107595086097717
Precisión de Prueba:         0.9338687062263489
```

Figura 40: Obtención de valores de precisión y pérdida para cada conjunto; Fuente: Elaboración propia (2023).

	precision	recall	f1-score	support
COVID	0.93	0.96	0.94	362
Lung_Opacity	0.94	0.87	0.91	602
Normal	0.93	0.96	0.94	1019
Viral Pneumonia	0.95	0.93	0.94	134
accuracy			0.93	2117
macro avg	0.94	0.93	0.93	2117
weighted avg	0.93	0.93	0.93	2117

Figura 41: Valores obtenidos del reporte de clasificación con la terminología estudiada anteriormente, indicando un buen nivel de aprendizaje; Fuente: Elaboración propia (2023).

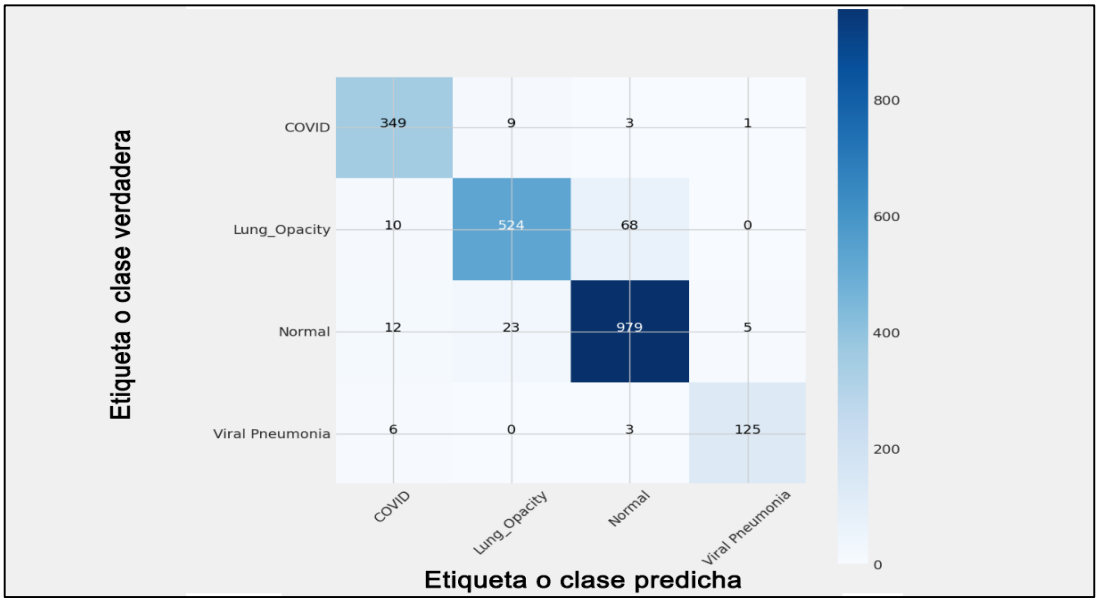


Figura 42: Matriz de confusión obtenida, indicando poca tendencia a confundir clases (esto, mayoritariamente en opacidad pulmonar); Fuente: Elaboración propia (2023).

6.2 Aseguramiento de la reproducibilidad del modelo

Previamente se definió reproducibilidad como la capacidad de obtener los mismos resultados, bajo las mismas semillas o configuraciones de procesamiento de datos, y evitar así cualquier tipo de interferencia que pudiese llevar a obtener conclusiones erróneas de qué beneficia o no al algoritmo. Para lograr este aspecto se adjuntaron los siguientes pasos a la versión anterior.

- La definición de una semilla numérica, la cual será ingresada al algoritmo como una variable o tipo de dato a leer en distintas funciones.
- El ajuste de las siguientes funciones numéricas y de azar en base a la semilla ingresada previamente:
    - Python Random (random.seed)
    - NumPy Random (np.random.seed)
    - Tensorflow Keras Random (tf.keras.utils.set_random_seed)
    - Tensorflow CUDNN (TF_CUDNN_DETERMINISTIC)
- Junto con esto, la instalación de librerías de determinismo de Tensorflow.
    - Tensorflow Determinism
    - Framework Reproducibility
    - Activación de estas (config.experimental.enable_op_determinism)

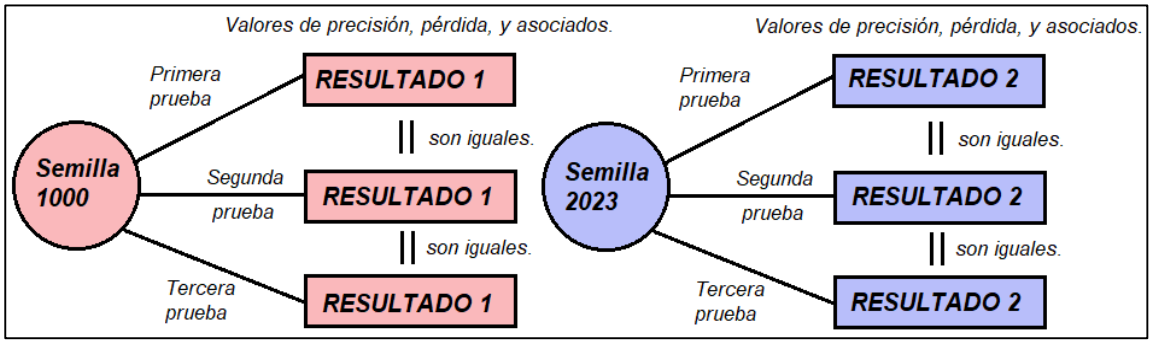


Figura 43: Representación gráfica de la consistencia y reproducibilidad que se espera; Fuente: Elaboración propia (2023).

Como consideración, cada una de las siguientes pruebas fueron realizadas con la activación del acelerador GPU P100, ofrecido por la plataforma de Kaggle.

Primera prueba realizada									
Epoch	**Pérdida**	**Precisión**	**V.Pérd**	**V.Prec**	**R.Aprend.**	**Sig. R.A.**	**Monitor**	**% Mejora**	**Duración**
1	5,540	49,244	1,042	56,947	0,00100	0,00100	accuracy	0,000	**301,940**
2	0,863	60,601	1,003	66,777	0,00100	0,00100	accuracy	23,060	**201,490**
3	0,775	75,012	2,189	80,293	0,00100	0,00100	accuracy	23,780	**202,650**
4	0,491	82,879	5,730	83,790	0,00100	0,00100	accuracy	10,490	**202,010**
5	1,195	85,885	2,301	84,310	0,00100	0,00100	accuracy	3,630	**200,120**
6	0,453	88,040	6,894	87,713	0,00100	0,00100	accuracy	2,510	**203,990**
7	0,748	89,724	30,657	88,422	0,00100	0,00100	accuracy	1,910	**200,340**
8	1,287	90,545	2,630	89,036	0,00100	0,00100	val_loss	-162,170	**202,280**
9	0,264	91,921	2,227	90,312	0,00100	0,00100	val_loss	-122,040	**201,530**
10	0,338	92,777	7,258	88,043	0,00100	0,00500	val_loss	-623,450	**202,830**
Segunda prueba realizada									
Epoch	**Pérdida**	**Precisión**	**V.Pérd**	**V.Prec**	**R.Aprend.**	**Sig. R.A.**	**Monitor**	**% Mejora**	**Duración**
1	5,540	49,244	1,042	56,947	0,00100	0,00100	accuracy	0,000	**264,030**
2	0,863	60,601	1,003	66,777	0,00100	0,00100	accuracy	23,060	**196,510**
3	0,775	75,012	2,189	80,293	0,00100	0,00100	accuracy	23,780	**196,450**
4	0,491	82,879	5,730	83,790	0,00100	0,00100	accuracy	10,490	**195,870**
5	1,195	85,885	2,301	84,310	0,00100	0,00100	accuracy	3,630	**195,720**
6	0,453	88,040	6,894	87,713	0,00100	0,00100	accuracy	2,510	**195,860**
7	0,748	89,724	30,657	88,422	0,00100	0,00100	accuracy	1,910	**195,280**
8	1,287	90,545	2,630	89,036	0,00100	0,00100	val_loss	-162,170	**197,130**
9	0,264	91,921	2,227	90,312	0,00100	0,00100	val_loss	-122,040	**196,420**
10	0,338	92,777	7,258	88,043	0,00100	0,00500	val_loss	-623,450	**196,890**
Promedio de pruebas realizadas									
Epoch	**Pérdida**	**Precisión**	**V.Pérd**	**V.Prec**	**R.Aprend.**	**Sig. R.A.**	**Monitor**	**% Mejora**	**Duración**
1	5,540	49,244	1,042	56,947	0,00100	0,00100	accuracy	0,000	**282,985**
2	0,863	60,601	1,003	66,777	0,00100	0,00100	accuracy	23,060	**199,000**
3	0,775	75,012	2,189	80,293	0,00100	0,00100	accuracy	23,780	**199,550**
4	0,491	82,879	5,730	83,790	0,00100	0,00100	accuracy	10,490	**198,940**
5	1,195	85,885	2,301	84,310	0,00100	0,00100	accuracy	3,630	**197,920**
6	0,453	88,040	6,894	87,713	0,00100	0,00100	accuracy	2,510	**199,925**
7	0,748	89,724	30,657	88,422	0,00100	0,00100	accuracy	1,910	**197,810**
8	1,287	90,545	2,630	89,036	0,00100	0,00100	val_loss	-162,170	**199,705**
9	0,264	91,921	2,227	90,312	0,00100	0,00100	val_loss	-122,040	**198,975**
10	0,338	92,777	7,258	88,043	0,00100	0,00500	val_loss	-623,450	**199,860**

Tabla 13: Reproducibilidad y promedios para dos pruebas de diez epochs;
Fuente: Elaboración propia (2023).

Como se puede observar en el recuadro anterior, con la importación y aplicación de semillas y librerías de determinismo, se ha resuelto el problema de reproducibilidad al menos para estas dos pruebas. Sin embargo, se observa una diferencia en la duración cada una, de manera particular para cada epoch.

A continuación, se llevarán estos datos a un gráfico que permitirá visualizar fácilmente la diferencia presente en la duración de cada epoch anterior.

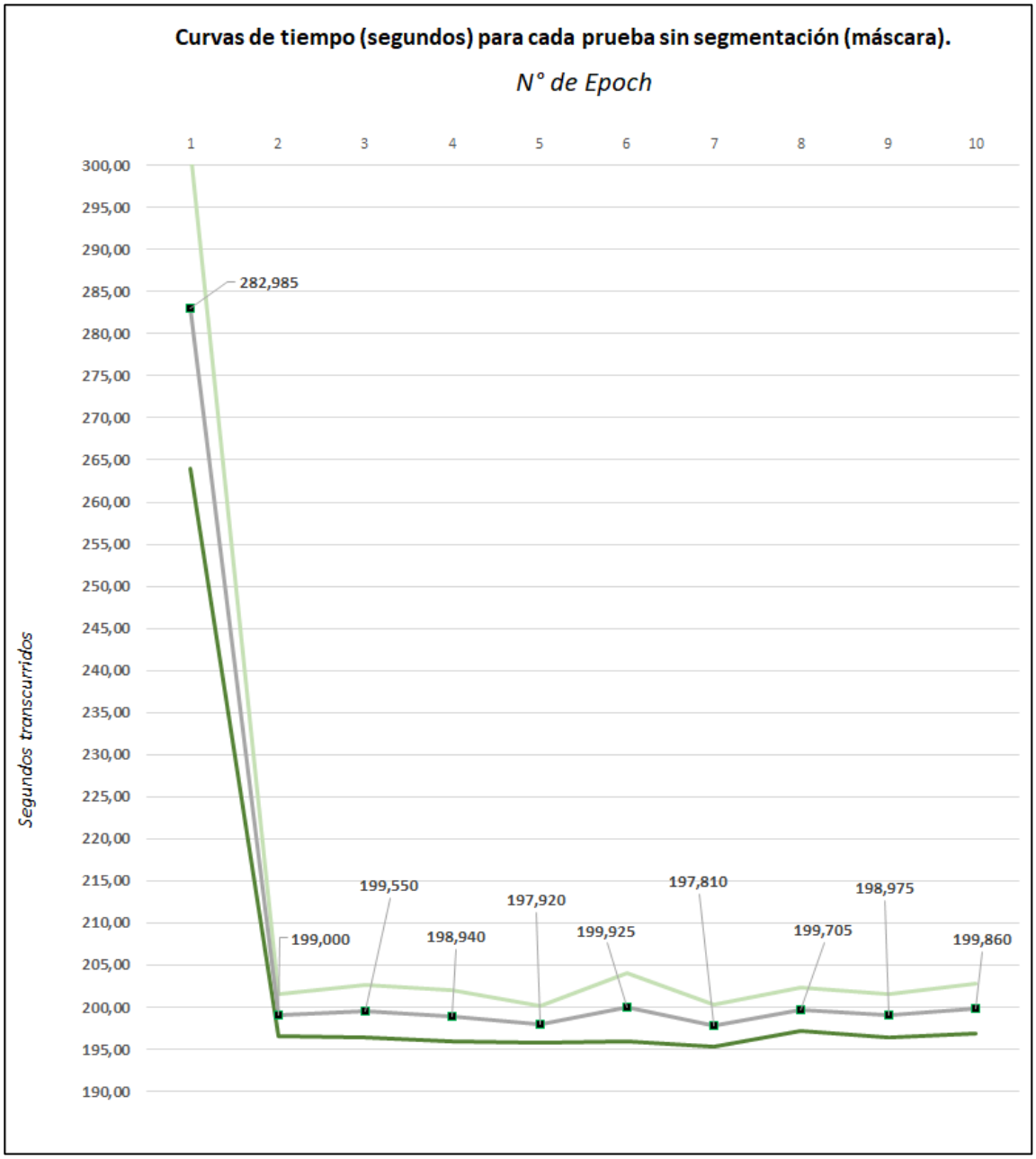


Figura 44: Representación gráfica, de la diferencia de tiempo en dos ejecuciones completas, junto al promedio de estas; Fuente: Elaboración propia (2023).

Para asegurar que esta diferencia sea consistente, es de importancia volver a realizar esta prueba y ver a mayor escala cuál puede ser esta variación de tiempo, para entender si este aspecto deberá ser descartado o no de la evaluación.

Cuatro pruebas consecutivas de solo tres epochs									
Epoch	**Pérdida**	**Precisión**	**V.Pérd**	**V.Prec**	**R.Aprend.**	**Sig. R.A.**	**Monitor**	**% Mejora**	**Duración**
1	5,540	49,244	1,042	56,947	0,00100	0,00100	accuracy	0,000	**296,650**
2	0,863	60,601	1,003	66,777	0,00100	0,00100	accuracy	23,060	**198,310**
3	0,775	75,012	2,189	80,293	0,00100	0,00100	accuracy	23,780	**198,870**
1	5,540	49,244	1,042	56,947	0,00100	0,00100	accuracy	0,000	**202,620**
2	0,863	60,601	1,003	66,777	0,00100	0,00100	accuracy	23,060	**200,960**
3	0,775	75,012	2,189	80,293	0,00100	0,00100	accuracy	23,780	**198,360**
1	5,540	49,244	1,042	56,947	0,00100	0,00100	accuracy	0,000	**204,860**
2	0,863	60,601	1,003	66,777	0,00100	0,00100	accuracy	23,060	**199,820**
3	0,775	75,012	2,189	80,293	0,00100	0,00100	accuracy	23,780	**200,650**
1	5,540	49,244	1,042	56,947	0,00100	0,00100	accuracy	0,000	**301,940**
2	0,863	60,601	1,003	66,777	0,00100	0,00100	accuracy	23,060	**201,490**
3	0,775	75,012	2,189	80,293	0,00100	0,00100	accuracy	23,780	**202,650**
1	5,540	49,244	1,042	56,947	0,00100	0,00100	accuracy	0,000	**251,518**
2	0,863	60,601	1,003	66,777	0,00100	0,00100	accuracy	23,060	**200,145**
3	0,775	75,012	2,189	80,293	0,00100	0,00100	accuracy	23,780	**200,133**

Tabla 14: Reproducibilidad y promedios para cuatro pruebas, cada una de tres epochs, junto al promedio de estas en color gris; Fuente: Elaboración propia (2023).

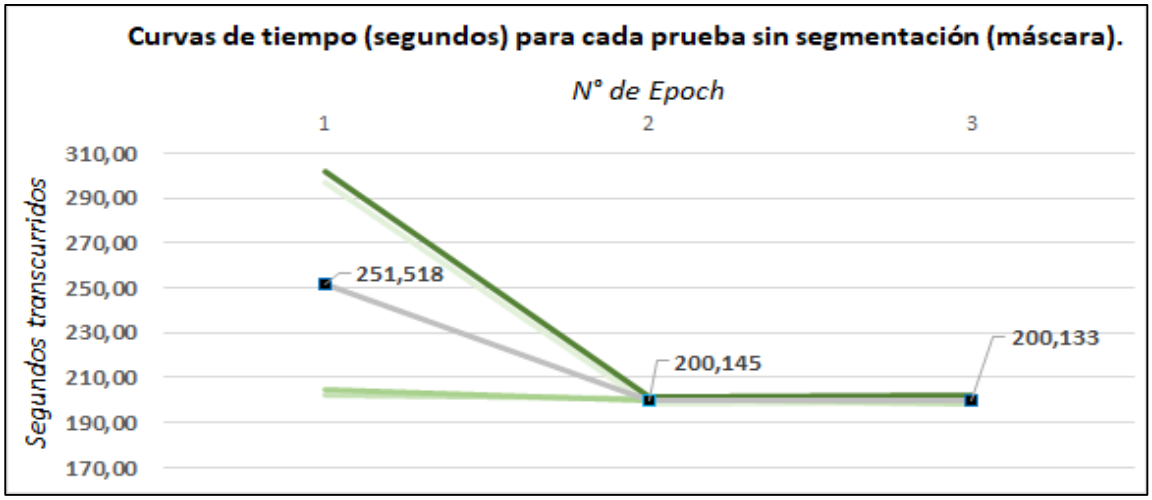


Figura 45: Representación gráfica, de la diferencia de tiempo en cuatro ejecuciones cortas, junto al promedio de estas; Fuente: Elaboración propia (2023).

La única distorsión que existe entre cada prueba es la duración particular de cada epoch. Este aspecto no es corregible al trabajar con las limitaciones impuestas por la plataforma Kaggle y la distribución interna de recursos realizada por esta. Por lo tanto, para considerar cambios a la eficiencia del modelo, se postula no considerar el primer epoch en la duración total, pues es en este que se presenta la mayor discrepancia, para cada uno de los seis casos recién vistos.

6.3 Trabajo de segmentación del modelo

Para este paso, cuyo código también se encuentra disponible en la sección Apéndice (13), se comienza leyendo la base de datos de radiografías de Rahman et al. (2022). Los pasos a seguir por el algoritmo serán los siguientes:

- Importar librerías de lectura de imagen mencionadas previamente.
- Crear directorios para las cuatro clases conocidas de este ejercicio.
- Obtener las rutas a leer de cada imagen y máscara.

El proceso de transformación de imagen puede definirse como:

- Lectura de la imagen y la máscara con CV2, transformándolas a arreglos de Python, fácilmente modificables.
- Se lee cada archivo a modo de blanco y negro, para así poder conservar la profundidad original en bits de estas imágenes (la cual es 8 bits).
- Redimensionar ambos archivos al mismo tamaño. La arquitectura de VGG-19, como se mencionó con anterioridad, sugiere la utilización de imágenes a 224 pixeles de ancho y alto, por lo que esta medida se escoge.
- Aplicar la máscara a través de la función bitwise_and de CV2.
- Se realizan los pasos anteriores en un ciclo para recorrer cada imagen y su máscara, y se comprimen los resultados, permitiendo descargar las imágenes con las máscaras ya aplicadas.

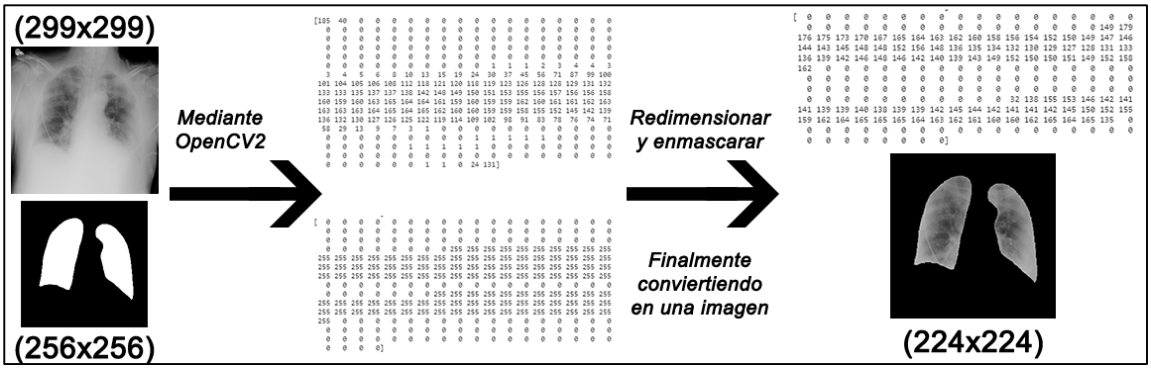


Figura 46: Diagrama que representa el proceso utilizado de segmentación de imagen;
Fuente: Elaboración propia (2023).

Recordando lo mencionado en 2.2, se espera que la segmentación de las máscaras ayude a evitar que los resultados se distorsionen, producto del ruido en las demás áreas del tórax. Esto se traduce en mayor precisión y rapidez de entrenamiento. Sin embargo, en el presente estudio se ha podido comprobar que este modelo y problema de cuatro clases no se beneficia realmente de la utilización de máscaras: el efecto que se ha podido evidenciar es el siguiente:

- La utilización de segmentación con máscaras baja considerablemente la precisión del modelo, tanto en entrenamiento, como validación y prueba.
- Disminuye el tiempo de ejecución, lo que podría llevar a pensar que se ha generado el primer caso aceptable: disminuye el tiempo de ejecución y disminuye el valor de precisión.
- Sin embargo, la diferencia en cuanto a tiempo requerido es mínima y descartable, considerando el efecto negativo que se presencia en la precisión, y en dos clases específicas del ejercicio: pacientes con coronavirus y pacientes con neumonía viral.
- Esto puede deberse a una multitud de razones: por ejemplo, el modelo puede estar basando su aprendizaje en la forma específica de los pulmones, en vez de las manchas y características dentro de estos.

Resultados del aprendizaje - CNN VGG-19 sin utilizar algún tipo de bio/metaheurística						
Semilla Utilizada	Precisión de entrenamiento (%) después de 10 epochs			Duración promediada (segundos) de cada epoch		
	Sin Segmentar	Con Segmentar	Diferencia	Sin Segmentar	Con Segmentar	Diferencia
1	92,777	81,302	11,475	**211,918**	180,483	**31,435**
577	92,275	**77,481**	**14,794**	207,195	184,412	22,783
1000	91,501	81,526	9,975	202,383	180,353	22,03
2023	**94,631**	84,142	10,489	207,494	187,093	20,401
5555	92,062	80,274	11,788	197,004	**179,261**	17,743
Promediado de pruebas	**92,6492**	**80,945**	**11,7042**	**205,1988**	**182,3204**	**22,8784**

Tabla 15: Resultados obtenidos por distintas pruebas, sin y con segmentación (aplicación de máscaras); Fuente: Elaboración propia (2023).

Cuatro pruebas consecutivas de solo tres epoch, esta vez, para las radiografías segmentadas									
Epoch	**Pérdida**	**Precisión**	**V.Pérd**	**V.Prec**	**R.Aprend.**	**Sig. R.A.**	**Monitor**	**% Mejora**	**Duración**
1	1,990	61,658	1,806	68,809	0,00100	0,00100	accuracy	0,000	**307,780**
2	0,811	72,868	2,921	73,913	0,00100	0,00100	accuracy	18,180	**182,680**
3	1,454	75,053	7,978	67,911	0,00100	0,00100	accuracy	3,000	**182,850**
1	1,990	61,658	1,806	68,809	0,00100	0,00100	accuracy	0,000	**189,070**
2	0,811	72,868	2,921	73,913	0,00100	0,00100	accuracy	18,180	**183,100**
3	1,454	75,053	7,978	67,911	0,00100	0,00100	accuracy	3,000	**184,250**
1	1,990	61,658	1,806	68,809	0,00100	0,00100	accuracy	0,000	**188,000**
2	0,811	72,868	2,921	73,913	0,00100	0,00100	accuracy	18,180	**181,760**
3	1,454	75,053	7,978	67,911	0,00100	0,00100	accuracy	3,000	**181,470**
1	1,990	61,658	1,806	68,809	0,00100	0,00100	accuracy	0,000	**188,760**
2	0,811	72,868	2,921	73,913	0,00100	0,00100	accuracy	18,180	**179,500**
3	1,454	75,053	7,978	67,911	0,00100	0,00100	accuracy	3,000	**180,200**
1	1,990	61,658	1,806	68,809	0,00100	0,00100	accuracy	0,000	**218,403**
2	0,811	72,868	2,921	73,913	0,00100	0,00100	accuracy	18,180	**181,760**
3	1,454	75,053	7,978	67,911	0,00100	0,00100	accuracy	3,000	**182,193**

Tabla 16: Reproducibilidad para los casos de radiografías segmentadas. Se aprecian menores precisiones y menores tiempos de ejecución; Fuente: Elaboración propia (2023).

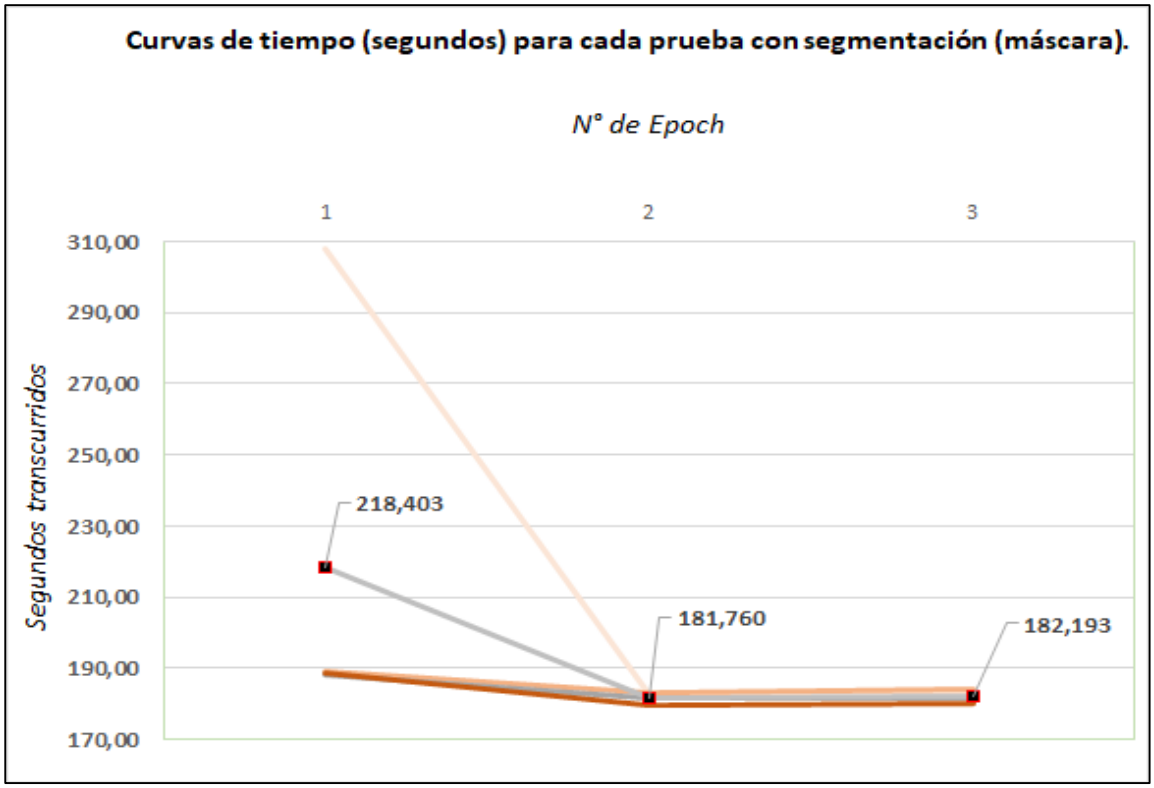


Figura 47: Representación gráfica, de la diferencia de tiempo de cuatro ejecuciones y su promedio, para pruebas con segmentación; Fuente: Elaboración propia (2023).

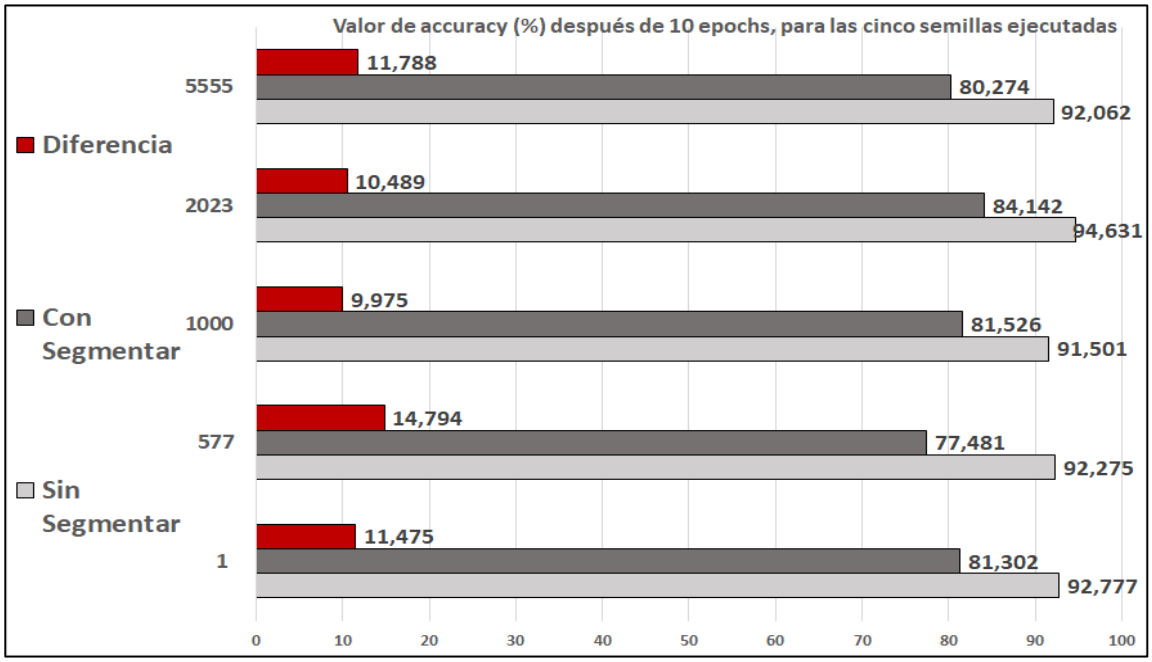


Figura 48: Comparación de la precisión para cada una de las pruebas, y la diferencia particular de cada una de estas; Fuente: Elaboración propia (2023).

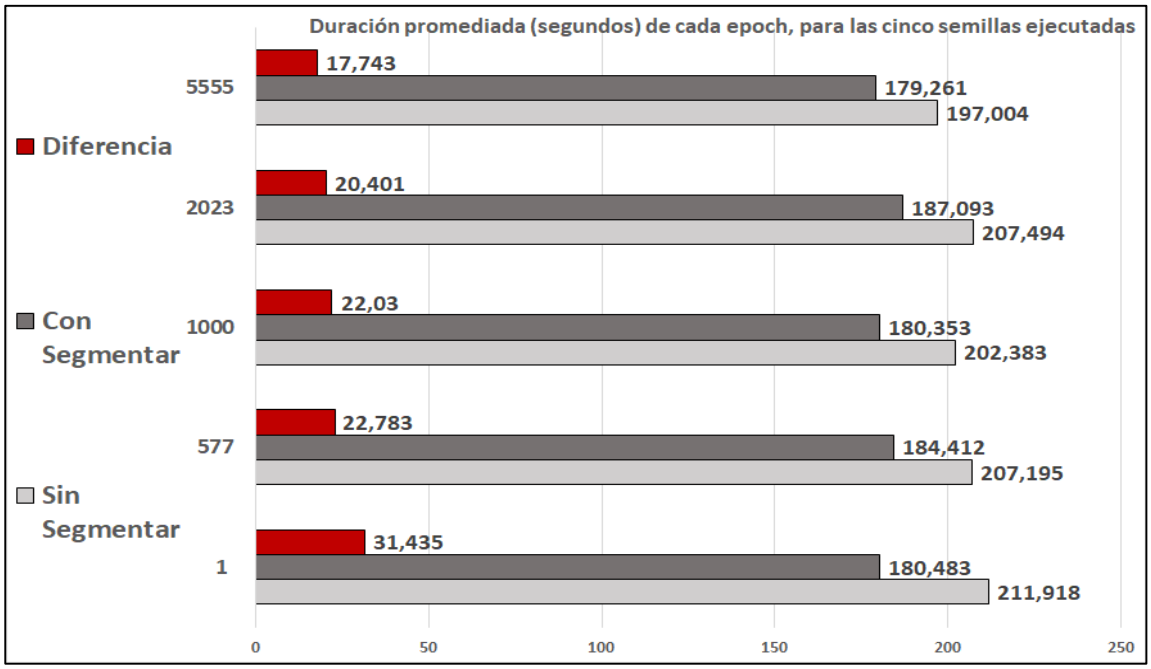


Figura 49: Comparación del tiempo de ejecución para cada una de las pruebas, y la diferencia particular de cada una de estas; Fuente: Elaboración propia (2023).

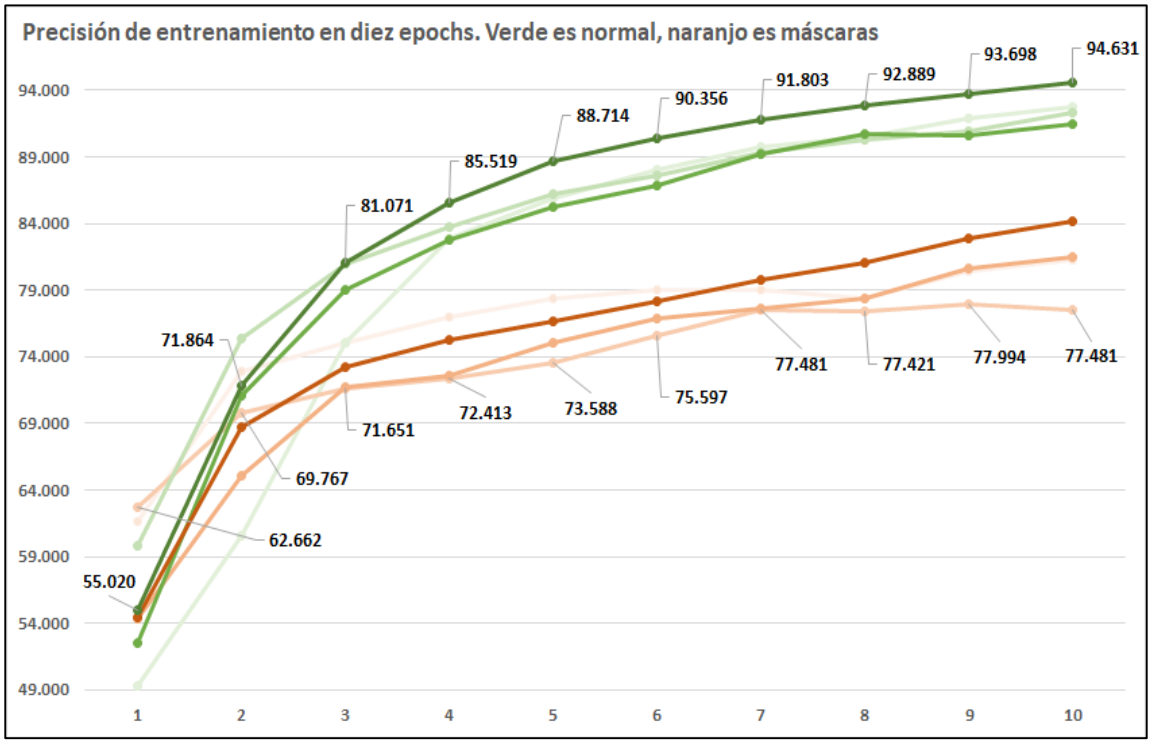


Figura 50: Para cuatro pruebas, comparación de precisión de entrenamiento con y sin máscaras y segmentación utilizadas; Fuente: Elaboración propia (2023).

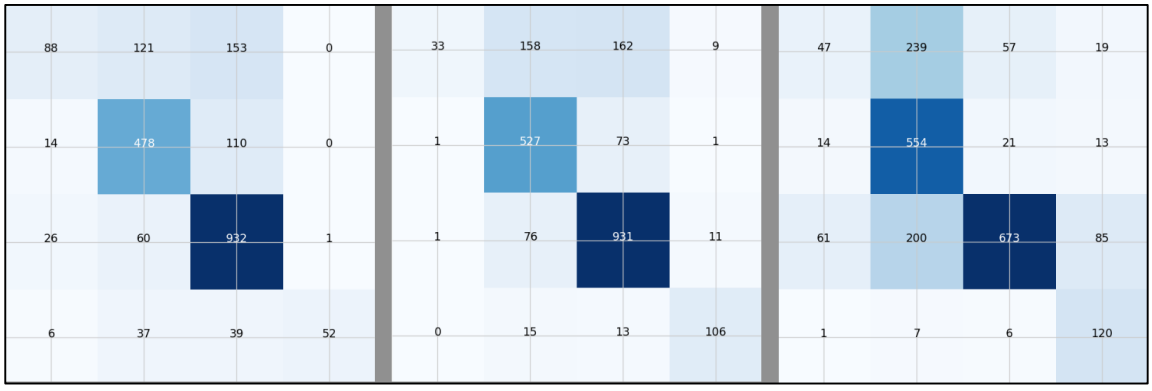


Figura 51: Efecto de la segmentación en tres matrices de confusión. Las clases de coronavirus y neumonía viral se ven altamente afectadas; Fuente: Elaboración propia (2023).

Como conclusión, el efecto de utilizar segmentación y máscaras en esta base de datos es negativo, y por lo tanto, se descartará su utilización para poder potenciar los resultados de este con las otras alternativas de técnicas a estudiar.

6.4 Programación y aplicación de un proceso metaheurístico: MRFO

En base a las soluciones propuestas por Balaha (2022), se ha estudiado e implementado un modelo basado en la metaheurística de Forrajeo de Mantarrayas, conocida por sus siglas como MRFO. Previamente definida como una técnica basada en la búsqueda de alimento realizada por las mantarrayas, las cuales realizan técnicas de búsqueda en cadena, forma de ciclón y forma de voltereta. Se definirá a mayor detalles los aspectos principales de esta, citando las definiciones realizadas por Mohammed et al. (2023).

“Los algoritmos de optimización con metaheurística, que emergieron del estudio y simulación de conductas y procesos inteligentes en la naturaleza, han sido propuestos para abordar limitaciones (obtener soluciones óptimas a un rango amplio de problemas de combinación). Estos algoritmos determinan efectivamente soluciones casi óptimas a problemas complejos de optimización, especialmente al manejarlos a gran escala. Además, los algoritmos metaheurísticos pueden superar un problema significativo de los algoritmos de búsqueda local: el atrapamiento de los agentes buscadores de soluciones en regiones locales lejanas a las regiones de soluciones globales. Evitar este atrapamiento es el desafío principal de los algoritmos de metaheurísticas”.

Del párrafo anterior, se destacan los siguientes conceptos y definiciones:

- **Agente buscador:** corresponde a un individuo generado inicialmente por el algoritmo. Junto a otros agentes buscadores, se esparcerán en el espacio de búsqueda y evolucionarán a través de procesos iterativos como mutación, cruzamiento y selección.
- **Atrapamiento:** situación en la que el algoritmo de búsqueda queda atrapado en una región específica del espacio de solución, incapaz de explorar y buscar mejores soluciones.
- **Regiones:** un subconjunto del espacio de soluciones, la cual puede o no ser óptima, y estar asociada a atrapamientos en el algoritmo.

Junto a lo anterior, Mohammed et al. hace énfasis en dos conceptos. El primer concepto exploración: la estrategia de aleatorización que busca llevar al algoritmo a buscar por todo el espacio y regiones de manera diversa. El segundo concepto es explotación, en donde se busca mejorar la solución actual al problema examinando el vecindario y sus soluciones similares más cercanas. Se sugiere un balance adecuado entre ambas técnicas, debido a que si la exploración es muy alta, las soluciones se revisarán y evaluarán con poca exactitud, y por contraste, si la exploración es muy baja, múltiples combinaciones y regiones no serán exploradas, llevando al concepto de atrapamiento mencionado anteriormente. La explotación adecuada es lograda al realizar un balanceado ajuste entre las soluciones, sus parámetros y regiones recorridas.

Continuando con la definición, se mencionan algoritmos y metaheurísticas bioinspiradas, categorizándolas de acuerdo a su enfoque, ya sea este evolutivo, físico, o basado en partículas. Se clasifica MRFO como la última categoría, de una metaheurística basada en partículas, movimiento y estrategias. Además, se menciona como motivación el comportamiento en la búsqueda de comida de las mantarrayas. Se utilizan tres estrategias para buscar plancton: búsqueda de comida en cadena, búsqueda de comida en ciclón, y búsqueda de comida en voltereta (somersault). Estas estrategias identifican la ubicación con la mayor densidad, que es devuelta como la mejor solución al problema de optimización.

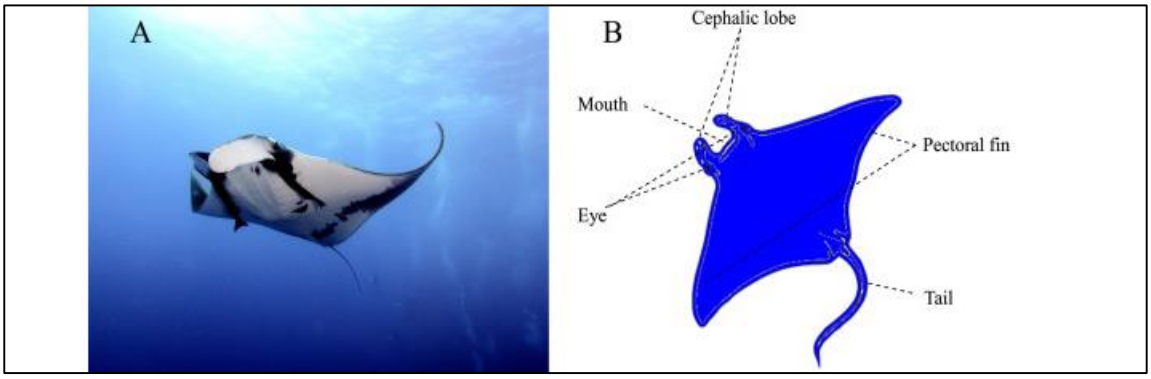


Figura 52: Partes de una mantarraya; Fuente: Zhao et al. (2023).

6.4.1 MRFO: Acercamiento a la búsqueda de comida en cadena

En este acercamiento, descrito por Mohammed et al. (2023), cada mantarraya actualiza su ubicación actual utilizando la mejor solución obtenida y la posición de la mantarraya frente a esta, excluyendo a la primera, que actualiza su ubicación de acuerdo a la mejor solución obtenida hasta el momento. Se define mediante el siguiente sistema de ecuaciones matemáticas.

$$p_k^{itr+1} = \begin{cases} p_k^{itr} + rn * \left(Gbest^{itr} - p_k^{itr}\right) + 2 * rn \\ \quad * \sqrt{|\log(r)|} * \left(Gbest^{itr} - p_k^{itr}\right) \quad k = 1 \\ p_k^{itr} + rn * \left(p_{k-1}^{itr} - p_k^{itr}\right) + 2 * rn \\ \quad * \sqrt{|\log(r)|} * \left(Gbest^{itr} - p_k^{itr}\right) \quad k = 2, \ldots, N \end{cases}$$

Figura 53: Fórmula de búsqueda de comida en cadena;
Fuente: Mohammed et al. (2023).

Donde $\boldsymbol{rn}$ denota un número entre cero y uno, $\boldsymbol{N}$ representa el número total de mantarrayas (el tamaño de la población), $\boldsymbol{p_k^{itr}}$ es la ubicación de cada mantarraya $\boldsymbol{k^{th}}$ en una iteración $\boldsymbol{itr}$, $\boldsymbol{p_k^{itr+1}}$ define la nueva ubicación de la siguiente iteración, y $\boldsymbol{Gbest}$ muestra la mejor solución global lograda.

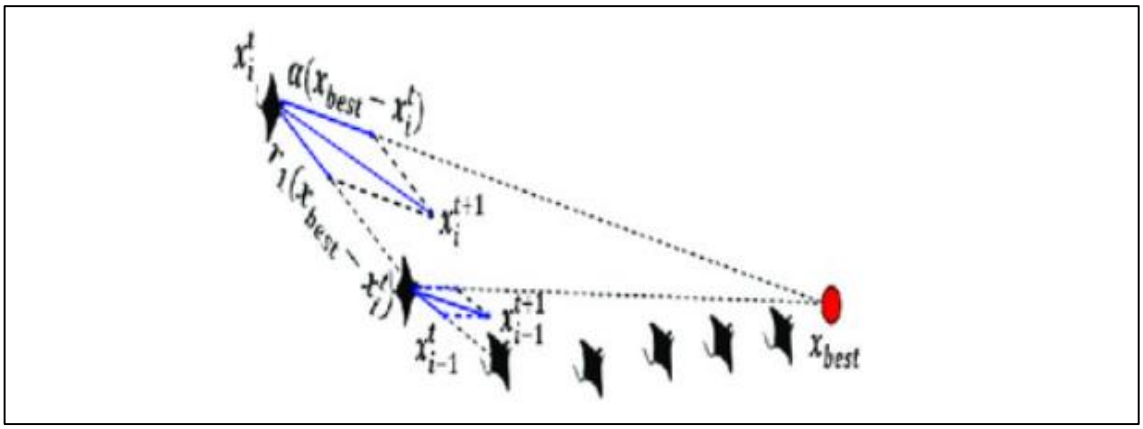


Figura 54: Gráfico de búsqueda de comida en cadena;
Fuente: Daqaq et al. (2023).

6.4.2 MRFO: Acercamiento a la búsqueda de comida en ciclón

En este acercamiento, las mantarrayas recorren el espacio de búsqueda de manera cíclica. El comportamiento del ciclón se representa como:

$$p_k^{itr+1} = \begin{cases} Gbest + rn * \left(Gbest^{itr} - p_k^{itr}\right) + 2* \\ \quad e^{rn*\frac{MaxItr-itr+1}{MaxItr}} * \sin(2 * \pi * rn) * \left(Gbest^{itr} - p_k^{itr}\right), \\ \quad k = 1 \\ Gbest + rn * \left(p_{k-1}^{itr} - p_k^{itr}\right) + 2* \\ \quad e^{rn*\frac{MaxItr-itr+1}{MaxItr}} * \sin(2 * \pi * rn) * \left(Gbest^{itr} - p_k^{itr}\right), \\ \quad k = 2, \ldots, N \end{cases} \qquad p_k^{itr+1} = \begin{cases} p_{rn} + rn * \left(p_{rn}^{itr} - p_{rn}\right) + 2 * e^{rn*\frac{MaxItr-itr+1}{MaxItr}} * \\ \quad \sin(2 * \pi * rn) * \left(p_{rn}^{itr} - p_k^{itr}\right), \\ \quad k = 1 \\ p_{rn} + rn * \left(p_{k-1}^{itr} - p_k^{itr}\right) + 2 * e^{rn*\frac{MaxItr-itr+1}{MaxItr}} * \\ \quad \sin(2 * \pi * rn) * \left(p_{rn}^{itr} - p_k^{itr}\right), \\ \quad k = 2, \ldots, N \end{cases}$$

$$p_{rn} = LowerBound + rn * (UpperBound - LowerBound).$$

Figura 55: Fórmula de búsqueda de comida en ciclón en sus dos formas;
Fuente: Mohammed et al. (2023).

Donde el nuevo valor $\boldsymbol{MaxItr}$ indica el número de iteraciones máximas a realizar por el algoritmo. Las mantarrayas realizan un recorrido aleatorio al actualizar sus localizaciones, basadas en posiciones aleatorias para estimular la diversificación, observado en la segunda fórmula del cuadro anterior, donde $\boldsymbol{p}_{rn}$ representa un punto de referencia definido por la tercera función, en donde LowerBound corresponde al límite inferior de los valores del espacio de búsqueda, y UpperBound corresponde al caso contrario.

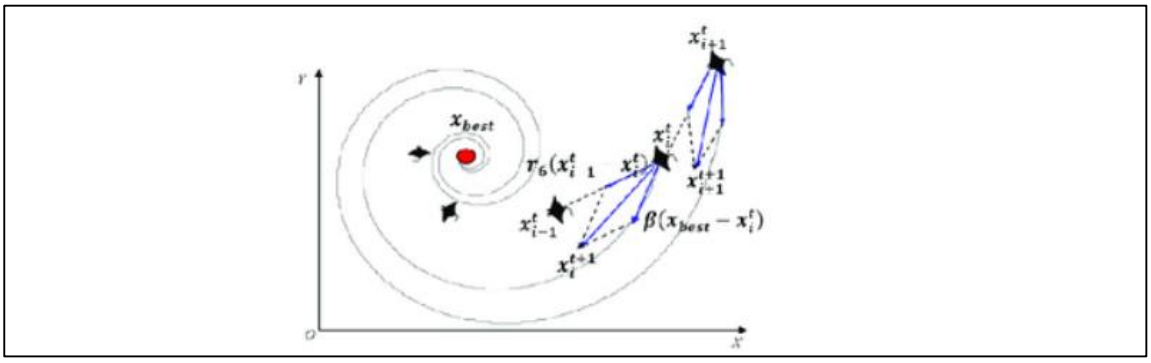


Figura 56: Gráfico de búsqueda de comida en ciclón;
Fuente: Daqaq et al. (2023).

6.4.3 MRFO: Acercamiento a la búsqueda de comida en voltereta

En este acercamiento, cada mantarraya ajusta su ubicación realizando un recorrido en voltereta en dirección hacia la mejor ubicación descubierta hasta el momento. Esta estrategia se describe con la siguiente fórmula.

$$p_k^{itr+1} = p_k^{itr} + Somersault\ factor * (rn_1 * Gbest - rn_2 * p_k^{itr}), i = 1, \ldots, N.$$

Figura 57: Fórmula de búsqueda de comida en voltereta;
Fuente: Mohammed et al. (2023).

Donde el factor de voltereta **($Somersault\ factor$)** es entregado como 2, mientras que rn_1 y rn_2 representan números en el intervalo entre cero y uno como se mencionó anteriormente.

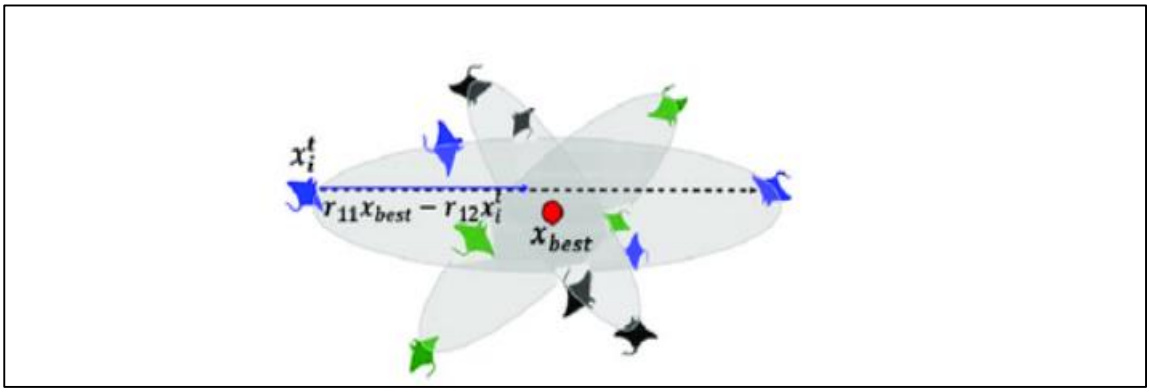


Figura 58: Gráfico de búsqueda de comida en voltereta;
Fuente: Daqaq et al. (2023).

Se destaca que el algoritmo de MRFO cuenta con distintas variantes, pero para el ejemplo actual se trabajará con las fórmulas y funcionamientos descritos. A continuación, se mostrará el funcionamiento de este con gráficos asociados.

6.4.4 MRFO: Diagrama y variables del proceso metaheurístico

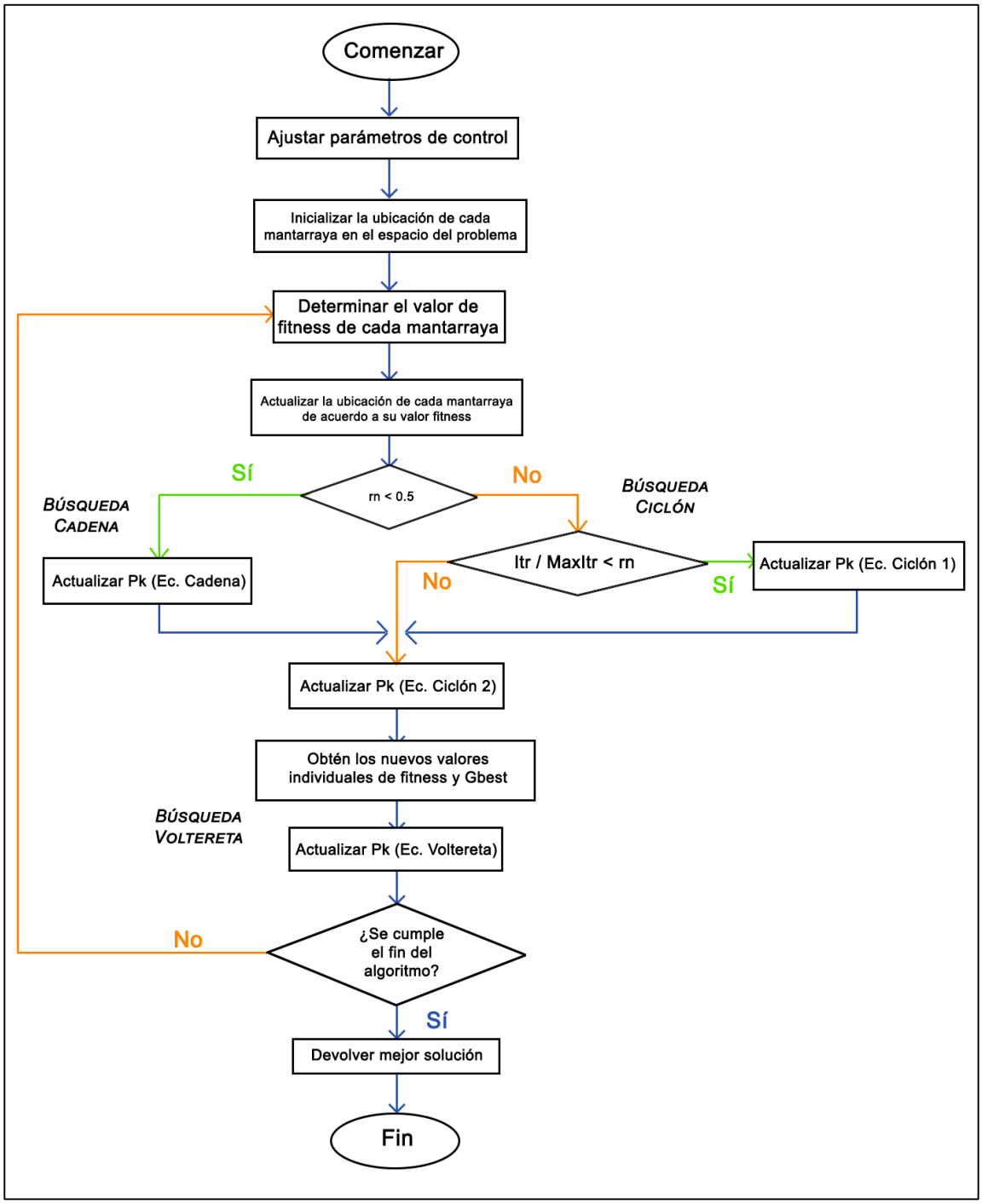


Figura 59: Diagrama de flujo del proceso MRFO; Fuente: Elaboración propia, en base al diagrama en inglés realizado por Mohammed et al. (2023).

La programación de la metaheurística requerirá de la definición de las siguientes variables en el código final del modelo propuesto.

Variable	Descripción	Valor escogido
Tamaño de la Población	El tamaño inicial de la población en la metaheurística.	4
Número de Iteraciones	Cuántas veces se repetirá el proceso de la metaheurística.	2
Límite inferior y límite superior	Límites requeridos por las fórmulas explicadas anteriormente.	[0.0 - 1.0]
Paciencia de la Metaheurística	Utilizada para finalizar el modelo prematuramente.	4
Hiperparámetros a modificar	Los hiperparámetros que modificará la metaheurística, definidos previamente en los puntos 2.5 y 4.	8, estos serán: • Rotación • Movimiento Ancho • Movimiento Alto • Zoom • Inclinación • Volteo Horizontal • Volteo Vertical • Tamaño de Lote
Epochs para cada iteración	Cada combinación de hiperparámetros será evaluada con una cantidad de epochs.	4

Tabla 17: Recuadro de variables asociadas al proceso metaheurístico a realizar y evaluar; Fuente: Elaboración propia (2023).

6.5 Evaluación de la primera implementación metaheurística

Evaluación realizada sobre el conjunto de prueba, con las características:

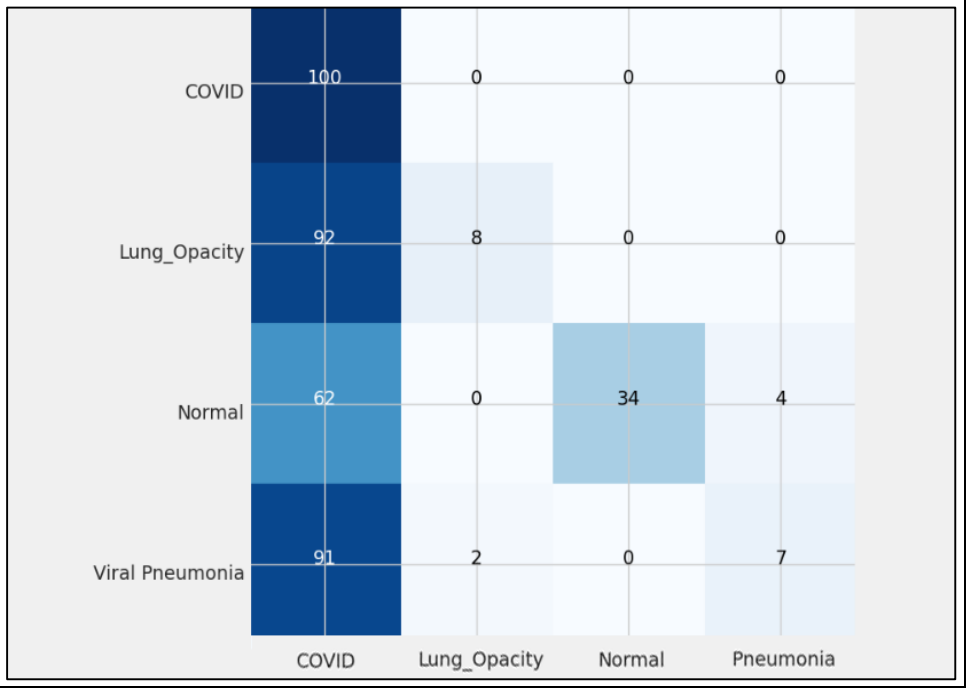
- *Dataset reducido (4.000 muestras).*
- *Lectura de radiografías a 32x32 pixeles, ancho y alto.*
- *10 epochs realizados bajo la semilla global "1".*
- *Se utiliza el Acelerador GPU P100 ofrecido por Kaggle.*

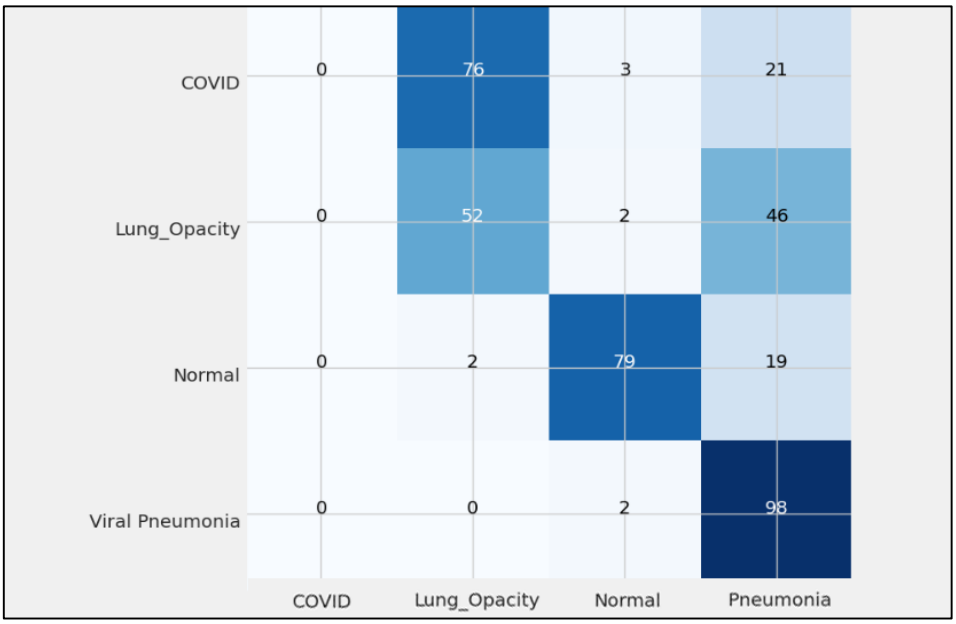
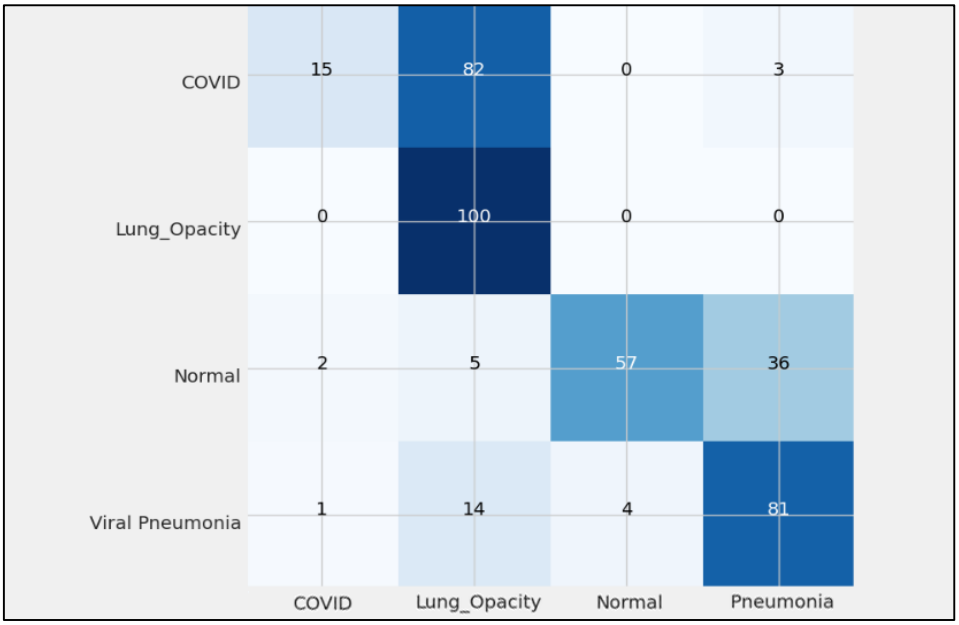
Selección de configuraciones entregadas por la metaheurística utilizada:

Puntaje	Los ocho hiperparámetros ajustados
0.4575,	[32, 0.3, 0.0, 0.0, 0.0, True, True, 8]
0.455,	[42, 0.4, 0.0, 0.0, 0.0, True, True, 8]
0.2625,	[16, 0.2, 0.0, 0.1, 0.0, True, True, 8]
0.25,	[24, 0.2, 0.0, 0.0, 0.0, True, True, 8]

Sin MRFO

00 Horas, 02 Minutos, 57 Segundos en total.
P.F1 (Exactitud, general): 0.37
P.F1 (Exactitud, para cada clase): 0.45, 0.15, 0.51, 0.13
Varianza entre las muestras: 0.0392



Con MRFO (0.25)	**00 Horas, 02 Minutos, 57 Segundos en total.** **P.F1 (Exactitud, general): 0.37** **P.F1 (Exactitud, para cada clase): 0.45, 0.15, 0.51, 0.13** **Varianza entre las muestras: 0.0392** 
Con MRFO (0.2625)	**00 Horas, 03 Minutos, 52 Segundos en total.** **P.F1 (Exactitud, general): 0.63** **P.F1 (Exactitud, para cada clase): 0.25, 0.66, 0.71, 0.74** **Varianza entre las muestras: 0.0524** 

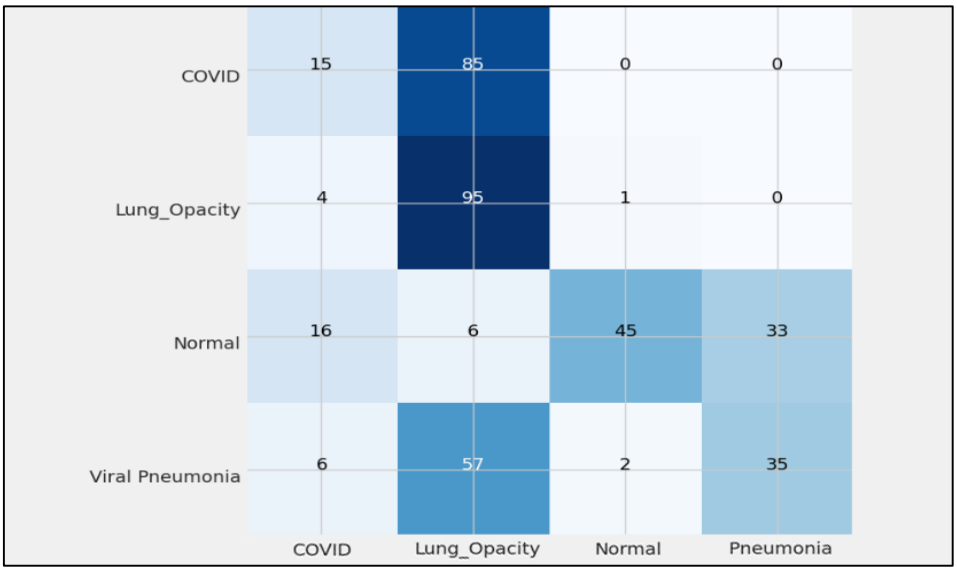
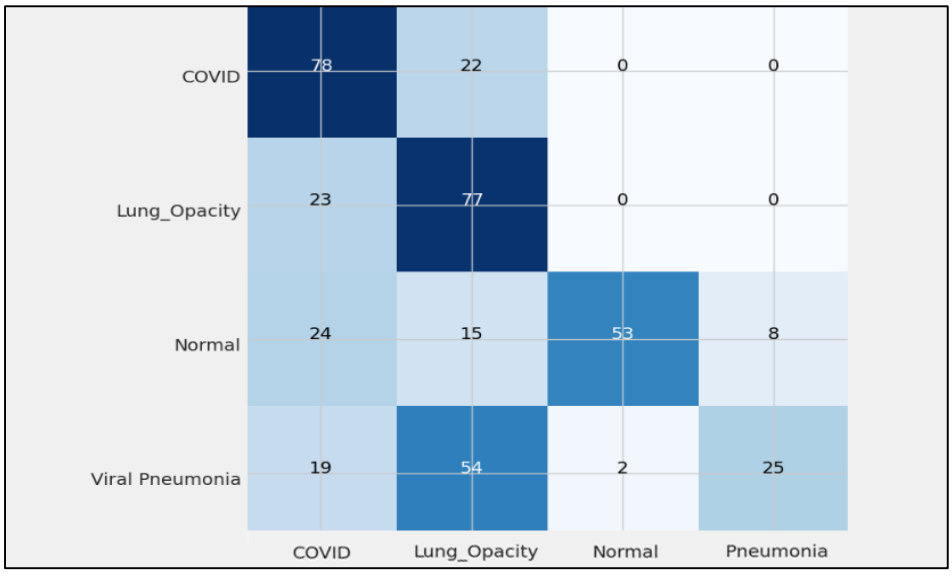
Con MRFO (0.455)	**00 Horas, 03 Minutos, 24 Segundos en total.** **P.F1 (Exactitud, general): 0.48** **P.F1 (Exactitud, para cada clase): 0.21, 0.55, 0.61, 0.42** **Varianza entre las muestras: 0.0313** 
Con MRFO (0.4575)	**00 Horas, 03 Minutos, 13 Segundos en total.** **P.F1 (Exactitud, general): 0.58** **P.F1 (Exactitud, para cada clase): 0.64, 0.57, 0.68, 0.38** **Varianza entre las muestras: 0.0176** 

Tabla 18: Resultados de la utilización de la primera versión de la metaheurística, en su primera prueba; Fuente: Elaboración propia (2023).

De lo anterior ha podido observarse:

- La utilización de metaheurísticas logra obtener distintas configuraciones las cuales resultan beneficiosas al ejemplo recién realizado.
- Se destaca que en la prueba sin utilizar esta técnica, el algoritmo clasifica a cada clase mayoritariamente como coronavirus. Sin embargo, al utilizar esta técnica, se presencia el crecimiento de la generalización y una clasificación más correcta.
- El ejemplo con mayor exactitud es la configuración asociada al puntaje 0.2625, pero este presenta fallas severas a la hora de clasificar radiografías asociadas a coronavirus.
- En comparación a 0.2625, que realiza 253 aciertos, se encuentra 0.4575, que realiza 233 aciertos, pero es capaz de clasificar con mayor consistencia a cada clase, obteniendo la menor varianza de los cinco ejemplos.

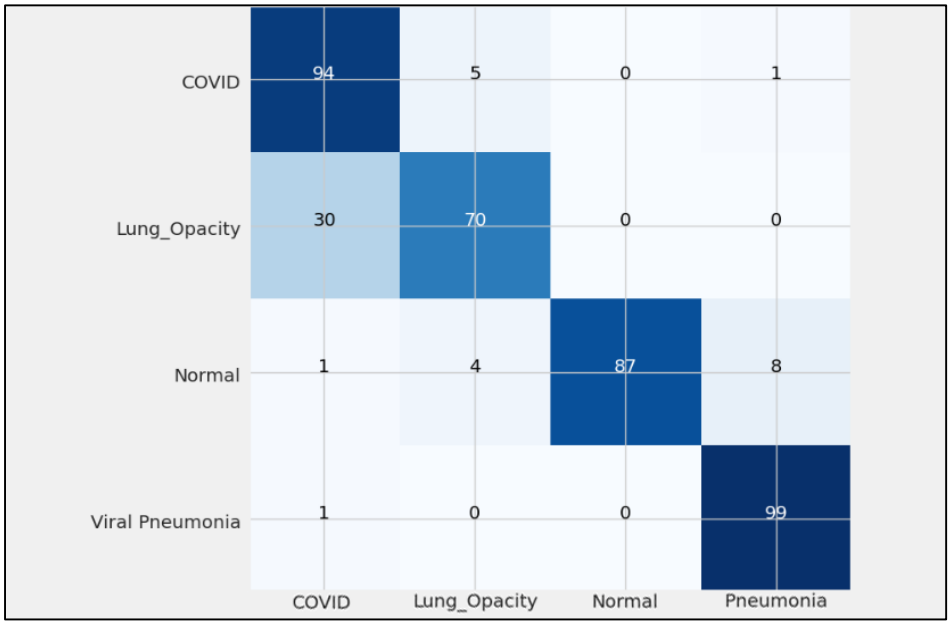
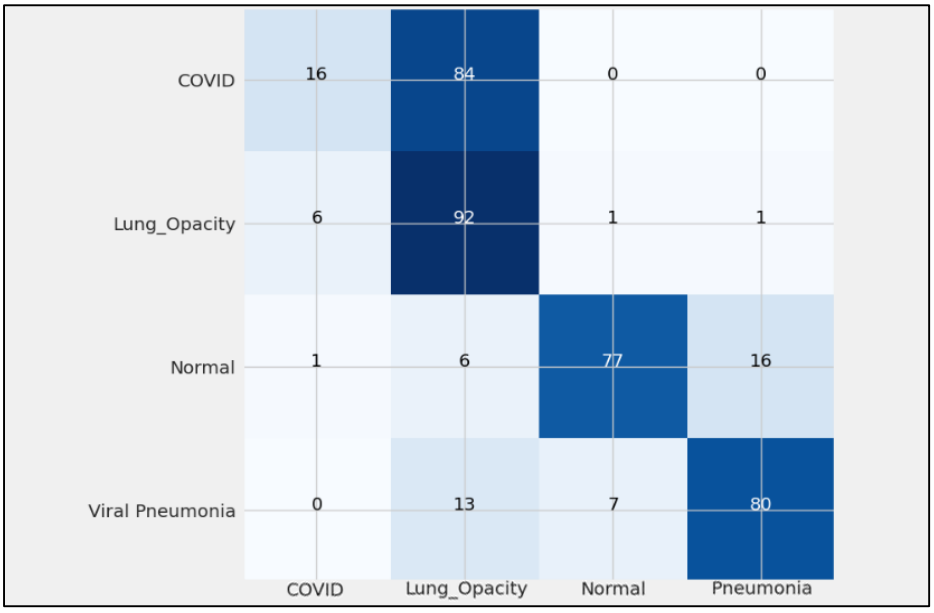
Entendiendo que esta es una prueba bajo condiciones que destruyen características, como el tamaño reducido de las imágenes, es de vital importancia continuar probando con distintas configuraciones y casos.

Evaluación realizada sobre el conjunto de prueba, con las características:

- *Dataset reducido (4.000 muestras).*
- *Lectura de radiografías a 224x224 pixeles, ancho y alto.*
- *10 epochs realizados bajo la semilla global "1".*
- *Se utiliza el Acelerador GPU P100 ofrecido por Kaggle.*

Selección de configuraciones entregadas por la metaheurística utilizada:

```
..........................................................
Puntaje         Los ocho hiperparámetros ajustados
..........................................................
    0.9,        [ 2, 0.0, 0.0, 0.0, 0.0, False, False, 16]
    0.7,        [11, 0.2, 0.0, 0.1, 0.0, True,  True,  16]
 0.5475,        [44, 0.4, 0.1, 0.2, 0.1, False, True,  64]
```

Sin MRFO	**00 Horas, 07 Minutos, 14 Segundos en total.** **P.F1 (Exactitud, general): 0.88** **P.F1 (Exactitud, para cada clase): 0.83, 0.78, 0.93, 0.95** **Varianza entre las muestras: 0.0065** 
Con MRFO **(0.5475)**	**00 Horas, 13 Minutos, 17 Segundos en total.** **P.F1 (Exactitud, general): 0.66** **P.F1 (Exactitud, para cada clase): 0.26, 0.62, 0.83, 0.81** **Varianza entre las muestras: 0.0698** 

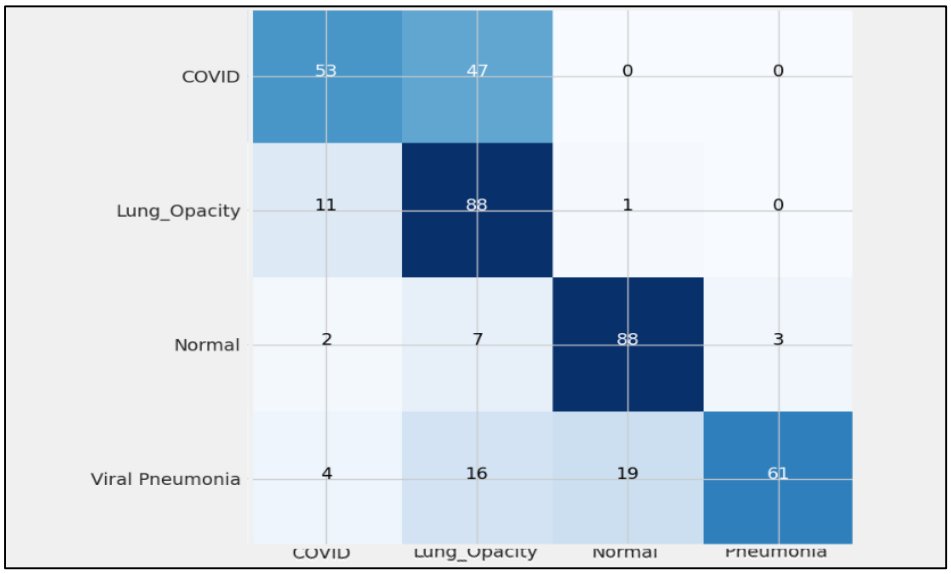
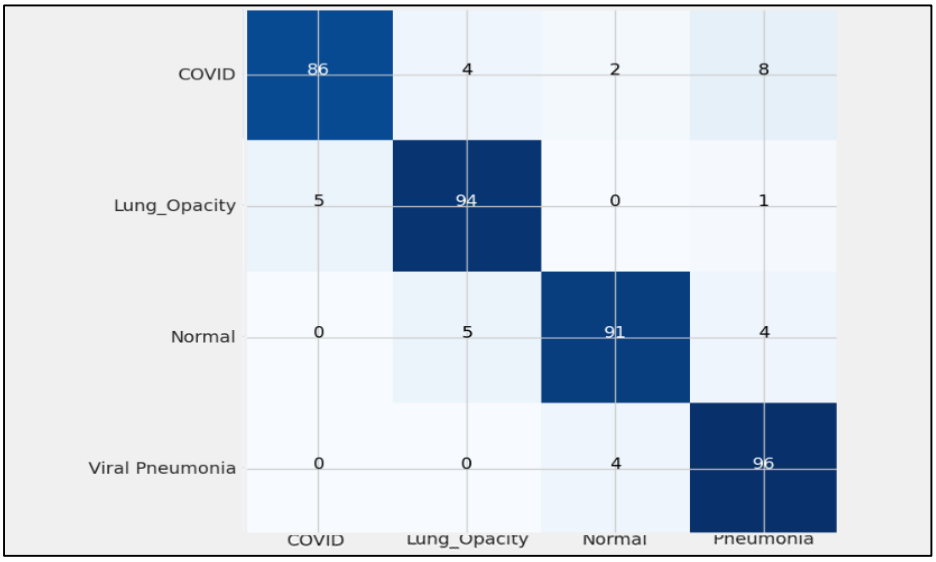
Con MRFO (0.7)	**00 Horas, 13 Minutos, 21 Segundos en total.** **P.F1 (Exactitud, general): 0.73** **P.F1 (Exactitud, para cada clase): 0.62, 0.68, 0.85, 0.74** **Varianza entre las muestras: 0.0096** 
Con MRFO (0.9)	**00 Horas, 14 Minutos, 06 Segundos en total.** **P.F1 (Exactitud, general): 0.92** **P.F1 (Exactitud, para cada clase): 0.90, 0.93, 0.92, 0.92** **Varianza entre las muestras: 0.00015** 

Tabla 19: Resultados de la utilización de la primera versión de la metaheurística, en su segunda prueba; Fuente: Elaboración propia (2023).

De lo anterior ha podido observarse:

- Dos de los puntajes altos entregados por la metaheurística no parecen ayudar al rendimiento en este caso, a diferencia del anterior. En la prueba sin MRFO se presencia alta exactitud, buena clasificación y baja varianza. Sin embargo, para los puntajes 0.5475 y 0.7 se pierde esta ventaja.
- Sólo un caso de los entregados logra mejorar el rendimiento del modelo: el puntaje 0.9, el cual específicamente sólo añade rotaciones aleatorias entre -2 y 2 grados, utilizando un tamaño de lotes de 16, y manteniendo los demás hiperparámetros de aumentación predeterminados.
- Un efecto que puede empezar a verse es el incremento del tiempo de ejecución del algoritmo: los entrenamientos empiezan a duplicar el tiempo requerido. Esto se debe principalmente a la utilización de la función ImageDataGenerator, la cual requiere accesos exigentes y constantes a la CPU durante el entrenamiento del modelo.

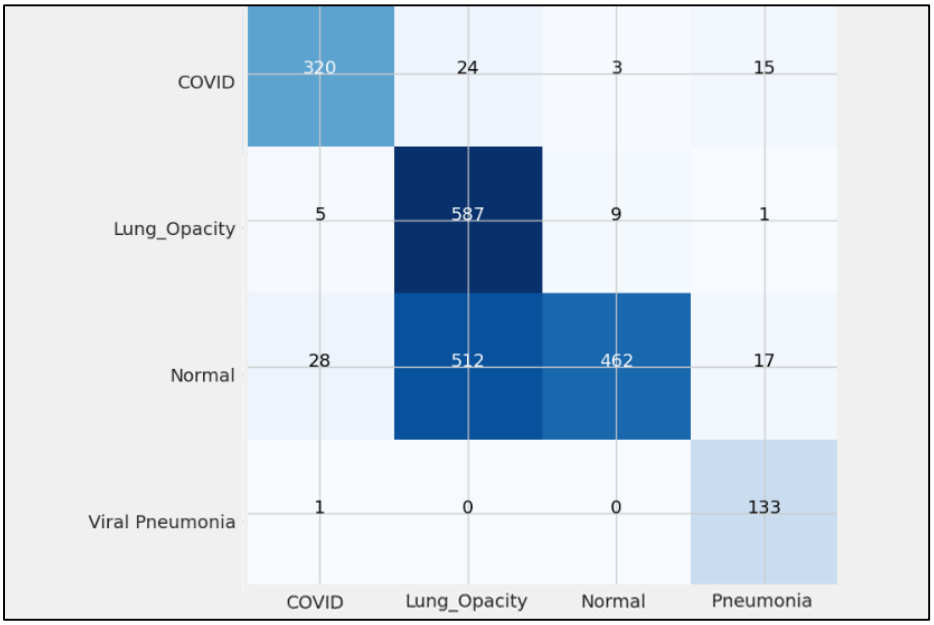
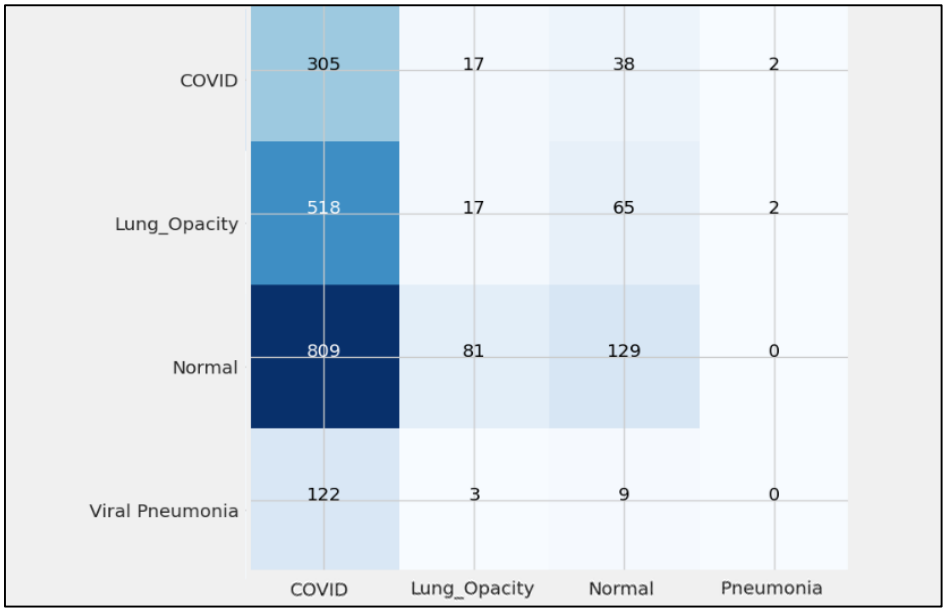
Lo siguiente será comprobar el funcionamiento de la metaheurística en el caso completo, para poder evidenciar si las configuraciones ayudan o no al algoritmo.

Evaluación realizada sobre el conjunto de prueba, con las características:

- *Dataset Completo (21.165 muestras).*
- *Lectura de radiografías a 224x224 pixeles, ancho y alto.*
- *10 epochs realizados bajo la semilla global "1".*
- *Se utiliza el Acelerador GPU P100 ofrecido por Kaggle.*

Selección de configuraciones entregadas por la metaheurística utilizada:

Puntaje	Los ocho hiperparámetros ajustados
0.6301	[35, 0.3, 0.0, 0.2, 0.2, True, True, 64]
0.5503	[22, 0.2, 0.0, 0.2, 0.0, True, True, 16]
0.4813	[0, 0.0, 0.0, 0.1, 0.1, True, True, 8]

Sin MRFO	**00 Horas, 34 Minutos, 24 Segundos en total.** **P.F1 (Exactitud, general): 0.71** **P.F1 (Exactitud, para cada clase): 0.89, 0.68, 0.62, 0.89** **Varianza entre las muestras: 0.0198** 
Con MRFO **(0.4813)**	**01 Hora, 10 Minutos, 06 Segundos en total.** **P.F1 (Exactitud, general): 0.21** **P.F1 (Exactitud, para cada clase): 0.29, 0.05, 0.20, 0.00** **Varianza entre las muestras: 0.0179** 

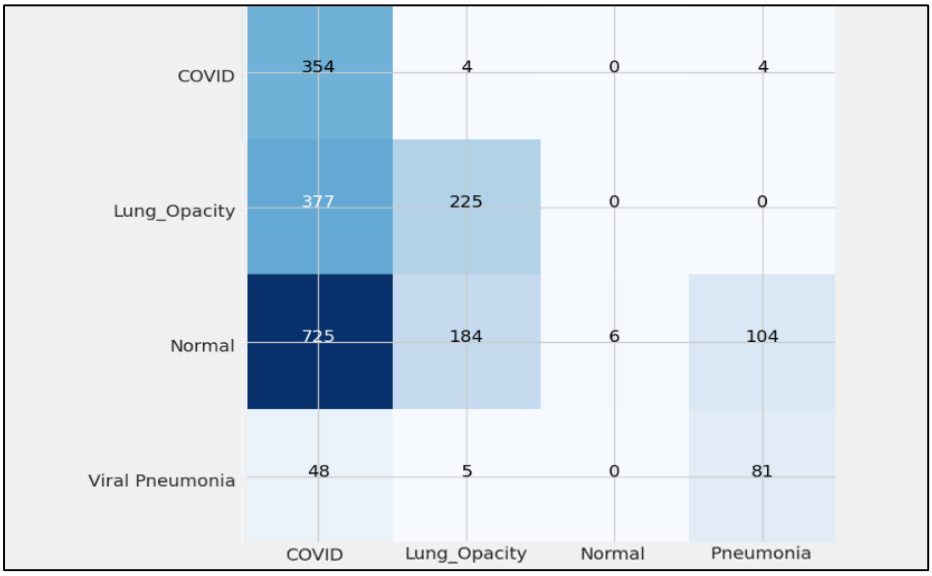
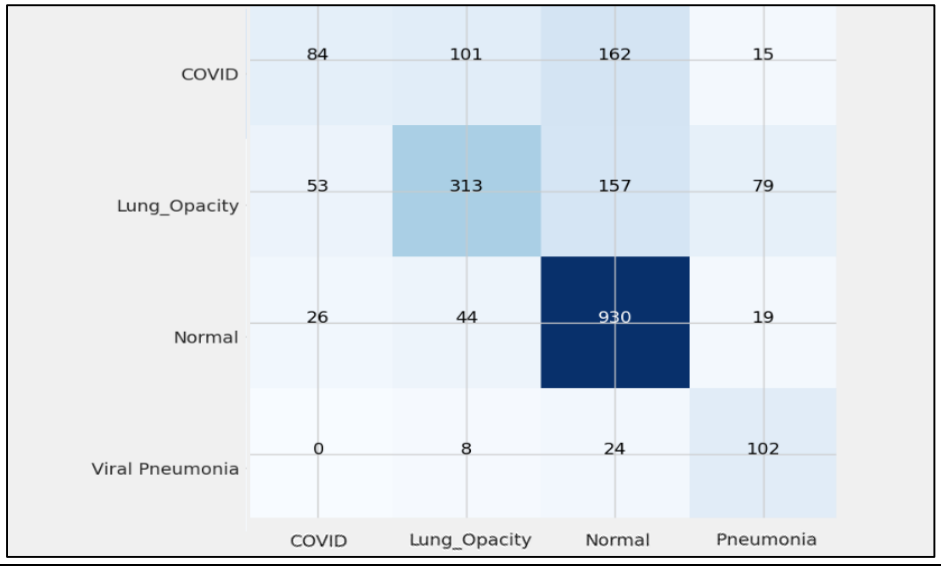
Con MRFO (0.5503)	**01 Hora, 04 Minutos, 11 Segundos en total.** **P.F1 (Exactitud, general): 0.31** **P.F1 (Exactitud, para cada clase): 0.38, 0.44, 0.01, 0.50** **Varianza entre las muestras: 0.0486** 
Con MRFO (0.6301)	**01 Hora, 07 Minutos, 51 Segundos en total.** **P.F1 (Exactitud, general): 0.68** **P.F1 (Exactitud, para cada clase): 0.32, 0.59, 0.81, 0.58** **Varianza entre las muestras: 0.0401** 

Tabla 20: Resultados de la utilización de la primera versión de la metaheurística, en su tercera prueba; Fuente: Elaboración propia (2023).

6.6 Considerando efectos secundarios del proceso de aumentación

Como se vio en el punto anterior, y bajo las condiciones de la primera prueba, la metaheurística es capaz de elevar los resultados de precisión de manera considerable y a su vez sin aumentar excesivamente el tiempo de ejecución. Sin embargo, esta ventaja no se presenta para el segundo caso, en donde sólo una configuración obtenida a través de la metaheurística logra mejorar los resultados del modelo, y mucho menos para el tercero, en donde las tres configuraciones obtenidas empeoran considerablemente la capacidad del modelo.

Por lo tanto, es de importancia considerar si se han escogido los hiperparámetros correctos a modificar. De acuerdo a Superb AI (2023), existen múltiples desafíos asociados a la técnica de aumentación de datos, que no pueden ser resueltos simplemente recolectando o alterando más datos. Mencionan como ejemplos:

- **La dificultad de conservar información original:** el riesgo de alterar y perder información vital está presente durante el proceso de aumentación. Por ejemplo, la excesiva rotación o escalado de una imagen la puede volver irreconocible o irrelevante para la tarea trabajada.
- **Crear instancias con datos anormales:** que no son representativas de la categoría actual o clase, llevando a resultados poco realistas. Un ejemplo en la base trabajada pudo ser el volteo horizontal y vertical.
- **Se puede sobre ajustar hacia los datos aumentados:** un riesgo en el que el modelo aprenderá de características anormales generadas por los datos aumentados, lo que a su vez destruirá su capacidad de generalizar frente a datos no vistos, como los del conjunto de prueba.

Junto con esto, Awan (2022) postula limitaciones de la aumentación como:

- La posibilidad de que los sesgos de la base de datos original persistan en los datos aumentados. Esto, inclusive, puede aumentar su presencia.
- Y el alto costo de asegurar la calidad de los datos aumentados.

En base a lo explorado anteriormente, y recordando lo contenido en 2.5, se considerará una nueva serie de hiperparámetros a modificar por la metaheurística, incluyendo y excluyendo parte de los que ya han sido utilizados.

Híper parámetro	¿Usado en la primera versión?	¿Se usará en la segunda versión?	Razonamiento o motivo de esta decisión
Rango de Movimiento al Ancho	✓	✗	Los bordes generados, independientemente del modo de llenado, pueden crear nuevas características que alteren la percepción del modelo y arruinen su rendimiento frente a nuevos datos.
Rango de Movimiento al Alto	✓	✗	
Rango de Zoom	✓	✗	Al realizar acercamientos o alejamientos, se pueden perder características importantes asociadas a los bordes de los pulmones.
Rango de Inclinación	✓	✗	La base de datos contiene radiografías contiene poca interferencia en cuanto a ángulos extremos. Utilizar este parámetro puede crear o acentuar un sesgo.
Volteo Horizontal	✓	✗	Los casos de radiografías volteadas en horizontal o vertical no existen en la base de datos. Además, cada pulmón por individual tiene una proporción de tamaño distinta debido a la posición del corazón. Esta es información que puede perderse al utilizar volteos, por lo tanto, se descartan.
Volteo Vertical	✓	✗	

Rango de Rotación	✓	✓	Utilizado en rangos pequeños (hasta diez grados) puede ser beneficioso debido a la naturaleza de la base de datos, en donde gran parte de las radiografías presenta ángulos de rotación pequeños.
Rango de Brillo (Mínimo y Máximo)	✗	✓	No se ha utilizado. Permite variar la cantidad de muestras de baja o alta claridad, característica existente en la base, de manera natural debido al proceso de radiografías y sus distintas fuentes.
Rango de Cambio de Canal	✗	✓	No se ha utilizado. Permite variar la cantidad de muestras de bajo o alto contraste, característica existente también en la base de manera natural y realista.
Tamaño de Lote	✓	✓	Observando el beneficio de lotes pequeños para la primera prueba realizada en la primera versión del modelo, se conservará este Hiperparámetro. Así, se espera observar sus efectos y confirmar si ayuda o no al algoritmo realmente.

Tabla 21: Cambios a realizar al proceso metaheurístico según resultados;
Fuente: Elaboración propia (2023).

En resumen, se descartarán seis de los ocho hiperparámetros originales a causa de distintas discrepancias u observaciones. Dos se conservarán, y se añadirán “tres” nuevos que no han sido probados y se espera que generen casos realistas que puedan enriquecer la base de datos de radiografías, sin caer en sesgos, casos irrealistas u otro tipo de situaciones que empeoren su rendimiento final.

Junto con el nuevo orden de hiperparámetros de Rotación, Brillo Mínimo, Brillo Máximo, Canal y Tamaño de Lote, se utilizará el acelerador TPU VM v3-8 para facilitar la ejecución de esta segunda implementación en cuanto a tiempo.

6.7 Evaluación de la segunda implementación metaheurística

Evaluación realizada sobre el conjunto de prueba, con las características:

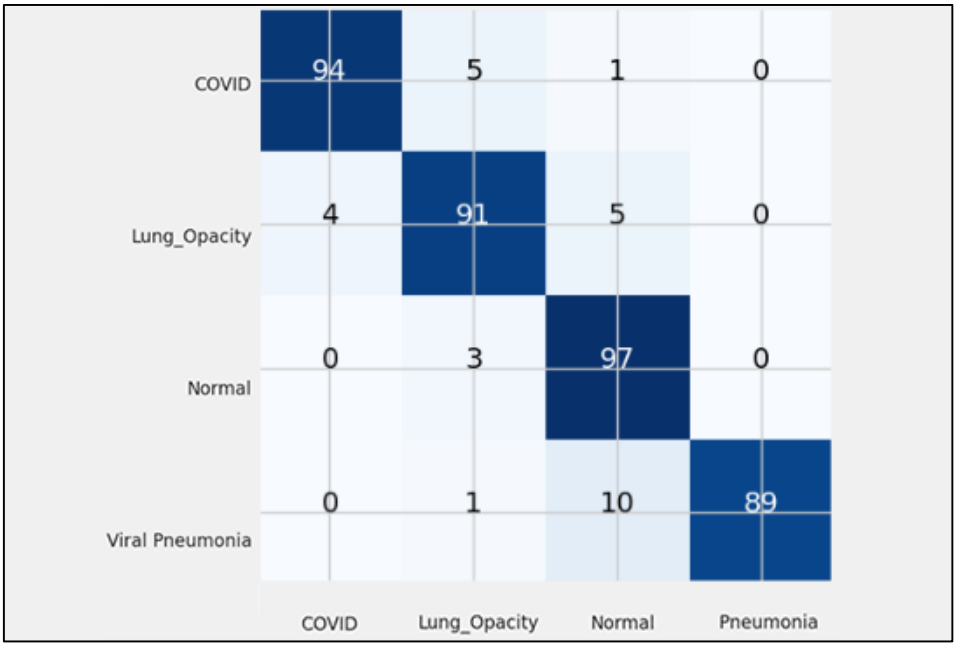
- *Dataset reducido (4.000 muestras).*
- *Lectura de radiografías a 224x224 pixeles, ancho y alto.*
- *10 epochs realizados bajo la semilla global "1".*
- *Se utiliza el Acelerador TPU VM v3-8 ofrecido por Kaggle.*

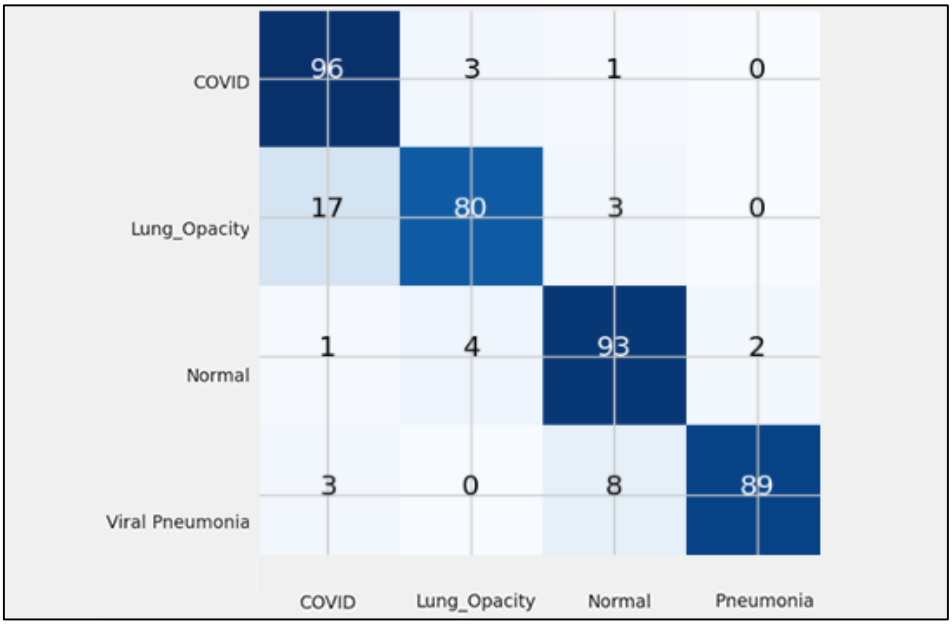
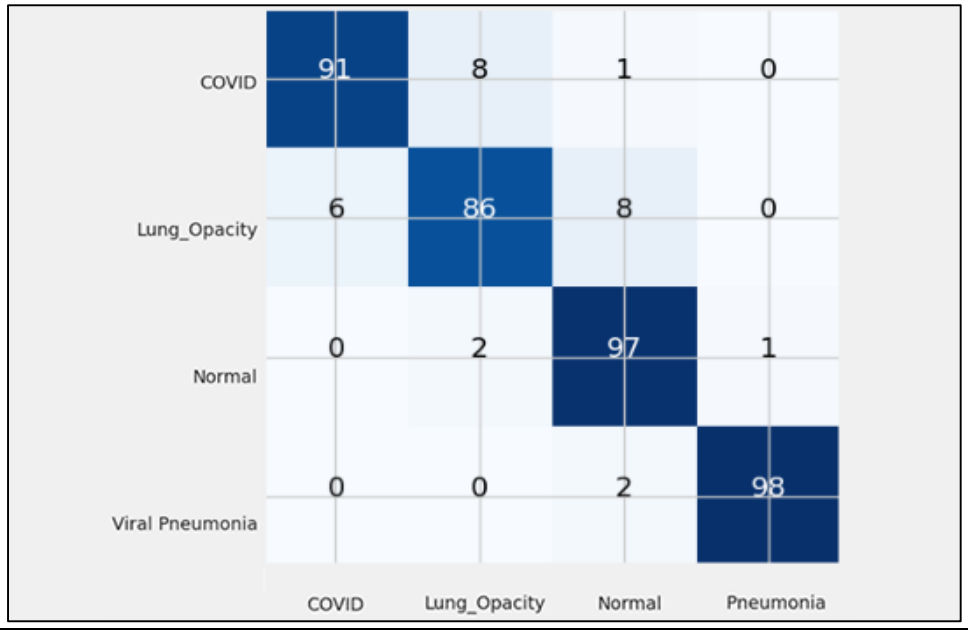
Selección de configuraciones entregadas por la metaheurística utilizada:

Puntaje	Los cinco hiperparámetros ajustados
0.9200,	[0, 1.00, 1.00, 16, 16]
0.9150,	[0, 0.50, 1.50, 0, 16]
0.9125,	[4, 0.90, 1.00, 43, 8]
0.9100,	[1, 0.65, 1.00, 19, 32]

Sin MRFO

00 Horas, 03 Minutos, 17 Segundos en total.
P.F1 (Exactitud, general): 0.93
P.F1 (Exactitud, para cada clase): 0.95, 0.91, 0.91, 0.94
Varianza entre las muestras: 0.000425



Con MRFO (0.9100)	**00 Horas, 09 Minutos, 11 Segundos en total.** **P.F1 (Exactitud, general): 0.90** **P.F1 (Exactitud, para cada clase): 0.88, 0.86, 0.91, 0.93** **Varianza entre las muestras: 0.000966** 
Con MRFO (0.9125)	**00 Horas, 09 Minutos, 46 Segundos en total.** **P.F1 (Exactitud, general): 0.93** **P.F1 (Exactitud, para cada clase): 0.92, 0.88, 0.93, 0.98** **Varianza entre las muestras: 0.001691** 

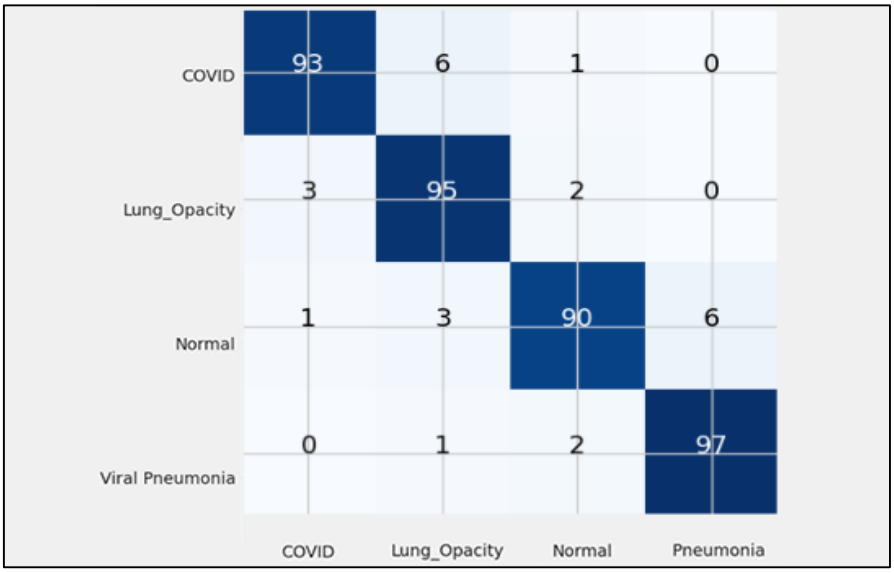
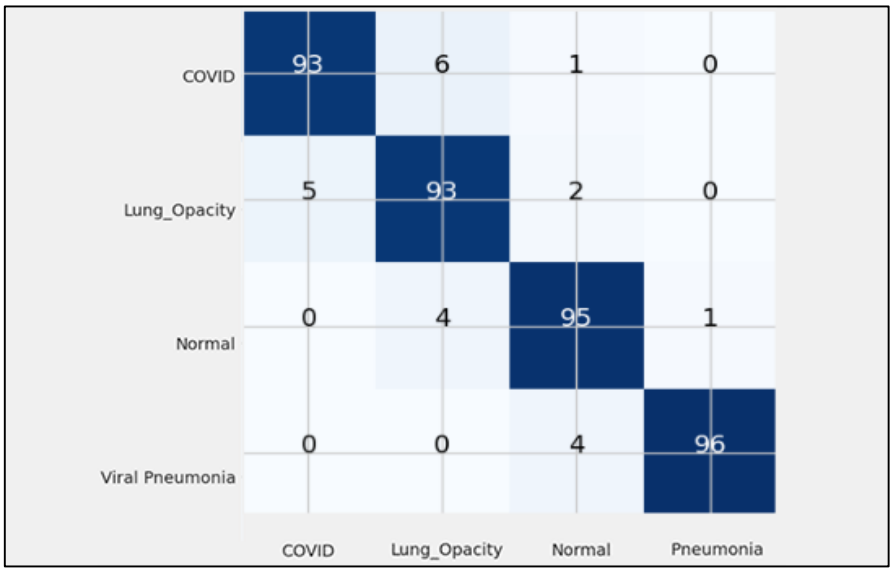
Con MRFO (0.9150)	**00 Horas, 03 Minutos, 30 Segundos en total.** **P.F1 (Exactitud, general): 0.94** **P.F1 (Exactitud, para cada clase): 0.94, 0.93, 0.92, 0.96** **Varianza entre las muestras: 0.000291** 
Con MRFO (0.9200)	**00 Horas, 04 Minutos, 21 Segundos en total.** **P.F1 (Exactitud, general): 0.94** **P.F1 (Exactitud, para cada clase): 0.94, 0.92, 0.94, 0.97** **Varianza entre las muestras: 0.000425** 

Tabla 22: Resultados de la utilización de la segunda versión de la metaheurística, en su primera prueba; Fuente: Elaboración propia (2023).

De lo anterior ha podido observarse:

- Un caso que obtiene peores resultados con la utilización de la metaheurísticas. Este corresponde a la única configuración con un tamaño de lote de 32, la cual podría resultar contraproducente para este tipo de ejercicio. Sin embargo, la caída de precisión es leve en comparación a otros casos obtenidos anteriormente con la primera versión.

- Tres casos que mejoran sus resultados. En comparación a la prueba sin metaheurística (con 371 aciertos), estos obtienen 375 y 377 aciertos. Además, los últimos dos casos sólo modifican los valores de brillo y canal, los cuales no ralentizan la ejecución considerablemente, y por lo tanto, se acercan potencialmente al mejor caso posible mencionado en el punto 4.

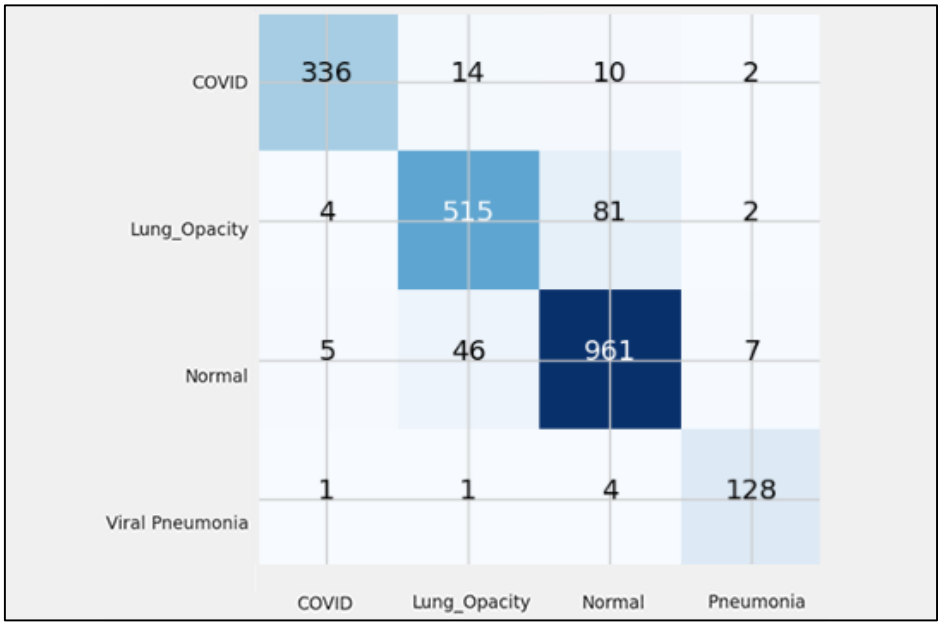
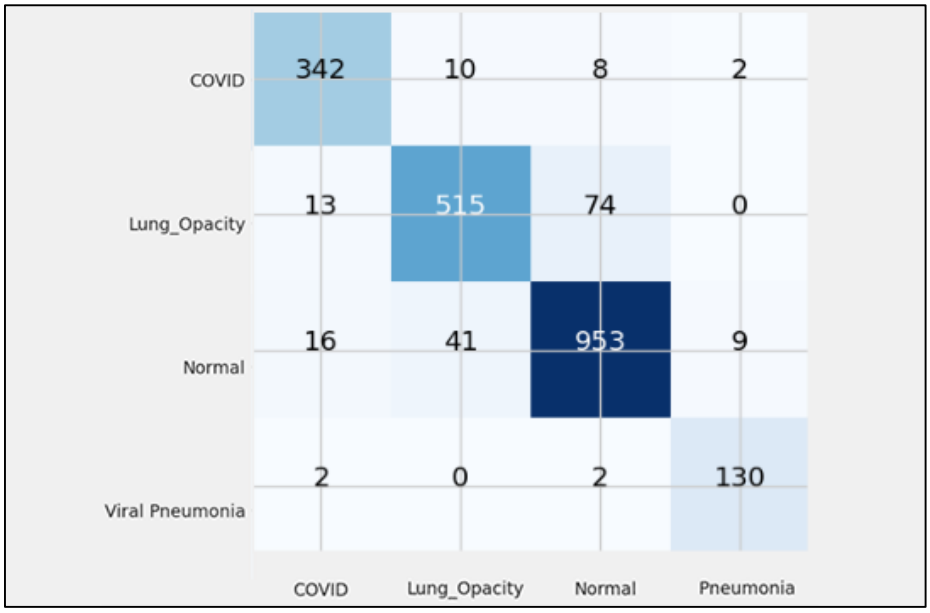
Lo siguiente será comprobar el funcionamiento de esta nueva versión en el caso completo, buscando encontrar discrepancias o bajas al rendimiento.

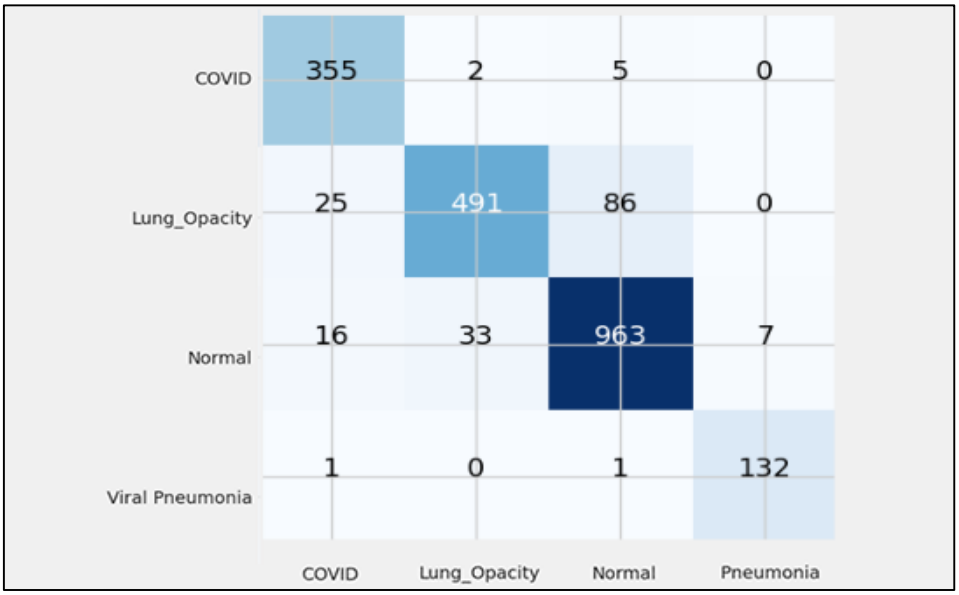
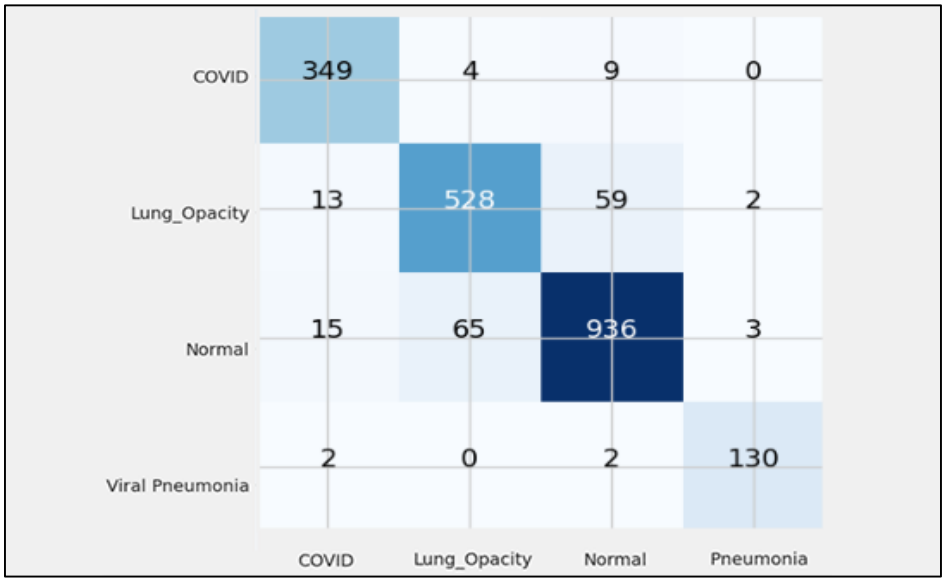
Evaluación realizada sobre el conjunto de prueba, con las características:

- *Dataset Completo (21.165 muestras).*
- *Lectura de radiografías a 224x224 pixeles, ancho y alto.*
- *10 epochs realizados bajo la semilla global "1".*
- *Se utiliza el Acelerador TPU VM v3-8 ofrecido por Kaggle.*

Selección de configuraciones entregadas por la metaheurística utilizada:

Puntaje	Los cinco hiperparámetros ajustados				
0.9190	[1,	0.75,	1.35,	26,	16]
0.9120	[2,	0.50,	1.50,	44,	16]
0.9050	[2,	1.05,	1.35,	11,	16]
0.9041	[2,	0.60,	0.90,	30,	8]

Sin MRFO	**00 Horas, 15 Minutos,16 Segundos en total.** **P.F1 (Exactitud, general): 0.92** **P.F1 (Exactitud, para cada clase): 0.95, 0.87, 0.93, 0.94** **Varianza entre las muestras: 0.001291** 
Con MRFO **(0.9041)**	**00 Horas, 48 Minutos, 15 Segundos en total.** **P.F1 (Exactitud, general): 0.92** **P.F1 (Exactitud, para cada clase): 0.93, 0.88, 0.93, 0.95** **Varianza entre las muestras: 0.000891** 

Con MRFO (0.9050)	**00 Hora, 44 Minutos, 41 Segundos en total.** **P.F1 (Exactitud, general): 0.92** **P.F1 (Exactitud, para cada clase): 0.94, 0.87, 0.93, 0.97** **Varianza entre las muestras: 0.001758** 
Con MRFO (0.9120)	**00 Hora, 44 Minutos, 57 Segundos en total.** **P.F1 (Exactitud, general): 0.92** **P.F1 (Exactitud, para cada clase): 0.94, 0.88, 0.92, 0.97** **Varianza entre las muestras: 0.001425** 

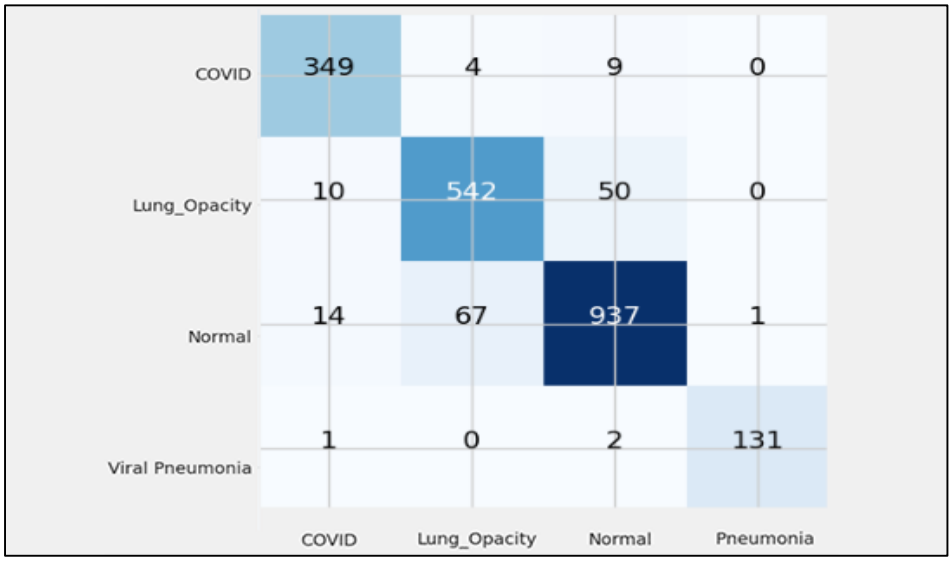
Con MRFO **(0.9190)**	**00 Hora, 45 Minutos, 33 Segundos en total.** **P.F1 (Exactitud, general): 0.93** **P.F1 (Exactitud, para cada clase): 0.95, 0.89, 0.93, 0.98** **Varianza entre las muestras: 0.001424** 

Tabla 23: Resultados de la utilización de la segunda versión de la metaheurística, en su segunda prueba; Fuente: Elaboración propia (2023).

De lo anterior ha podido observarse:

- Un caso con metaheurística (0.9041) que realiza la misma cantidad de aciertos que el modelo sin esta. Su única diferencia es que fue capaz de clasificar más radiografías de coronavirus y neumonía, pero falló más en la categoría de radiografías de condición normal.
- Tres casos con metaheurísticas (0.9050, 0.9120 y 0.9190) que obtienen resultados mejores. De 1940 aciertos, pasan a 1941, 1943 y 1959 aciertos finalmente, por lo que finalmente puede evidenciarse la utilidad de estas técnicas en un caso con la base de datos original y completa.
- La ralentización del algoritmo no es tan evidente bajo la utilización del acelerador TPU VM v3-8, en comparación a las pruebas con GPU P100: sus resultados superaban la hora de ejecución. Estos resultados rondan los cuarenta y cinco minutos para ejecutar diez epochs.

7 DIAGRAMA DE LA ARQUITECTURA Y PROPUESTA FINAL

Se ha podido obtener una serie de resultados que respaldan el uso de MRFO. Por lo tanto, se actualiza el diagrama o arquitectura propuesto al final del primer semestre y mencionado en el punto 3, contando los cambios realizados.

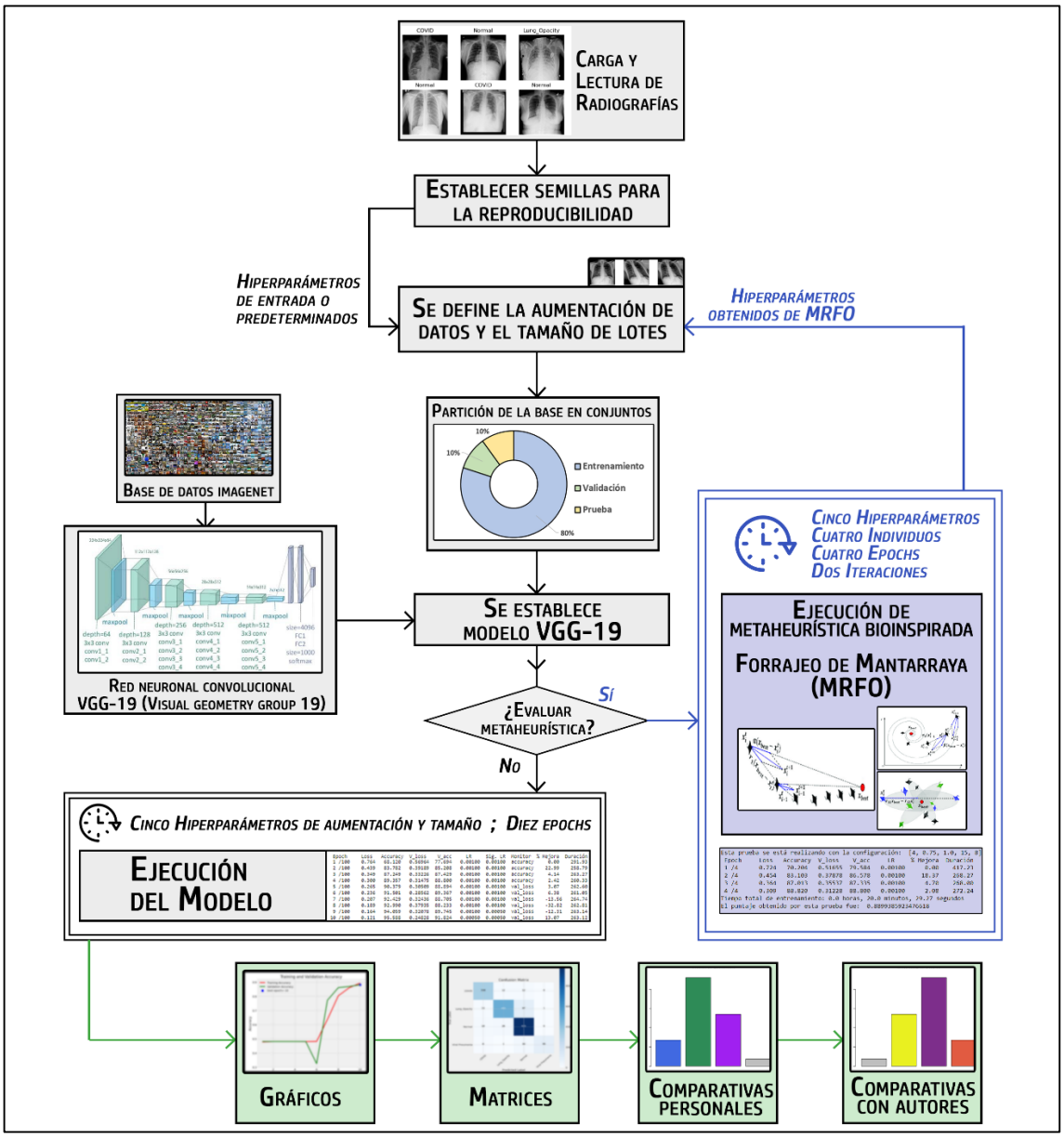


Figura 60: Arquitectura final, propuesta y evaluada al cierre del segundo semestre y segunda parte del proceso de proyecto de título; Fuente: Elaboración propia (2023).

8 RESULTADOS OBTENIDOS CON LA PROPUESTA FINAL

De los resultados obtenidos en el punto 6.7, se puede visualizar nuevamente la efectividad del modelo y arquitectura propuestos en este proyecto.

Semilla 1 Dataset Reducido	Exactitud Total	Exactitud Coronavirus	Exactitud Opacidad	Exactitud Normal	Exactitud Neumonía	Aciertos	Minutos
Sin MTH	0,93	**0,95**	0,91	**0,91**	0,94	371	**3,2833**
0,9100	**0,90**	**0,88**	**0,86**	**0,91**	**0,93**	**358**	9,1833
0,9125	0,93	0,92	0,88	0,93	**0,98**	372	**9,7666**
0,9150	**0,94**	0,94	**0,93**	0,92	0,96	375	3,5000
0,9200	**0,94**	0,94	0,92	**0,94**	0,97	**377**	4,3500
Semilla 1 Dataset Completo	**Exactitud Total**	**Exactitud Coronavirus**	**Exactitud Opacidad**	**Exactitud Normal**	**Exactitud Neumonía**	**Aciertos**	**Minutos**
Sin MTH	**0,92**	**0,95**	**0,87**	**0,93**	**0,94**	**1940**	**15,2666**
0,9041	**0,92**	**0,95**	**0,87**	**0,93**	**0,94**	**1940**	**48,2500**
0,9050	**0,92**	**0,94**	**0,87**	**0,93**	0,97	1941	44,6833
0,9120	**0,92**	**0,94**	0,88	**0,92**	0,97	1943	44,9500
0,9190	**0,93**	**0,95**	**0,89**	**0,93**	**0,98**	**1959**	45,5500

Tabla 24: Recolección de resultados con la segunda metaheurística;
Fuente: Elaboración propia (2023).

- Las mejores exactitudes totales en el dataset reducido se obtuvieron con los dos mejores puntajes, mientras que la mejor exactitud total en el dataset completo se obtuvo con el mejor puntaje.
- Todos los puntajes bajaron la precisión de detección de coronavirus en el caso del dataset reducido, pero este no fue el caso para el dataset completo.
- En el caso del dataset reducido, el mejor puntaje obtiene mejor exactitud total, exactitud de radiografías normales y aciertos. En el caso del dataset completo, mejora cada una de las categorías.
- La alternativa sin metaheurística se mantiene como la opción más veloz.
- Finalmente, el incremento obtenido por los mejores puntajes es de seis nuevos aciertos (dataset reducido) y 19 nuevos aciertos (dataset completo). Considerando el tamaño de prueba como 400 y 2.117 muestras, significa un aumento de 1.5% y 0.9% del total.

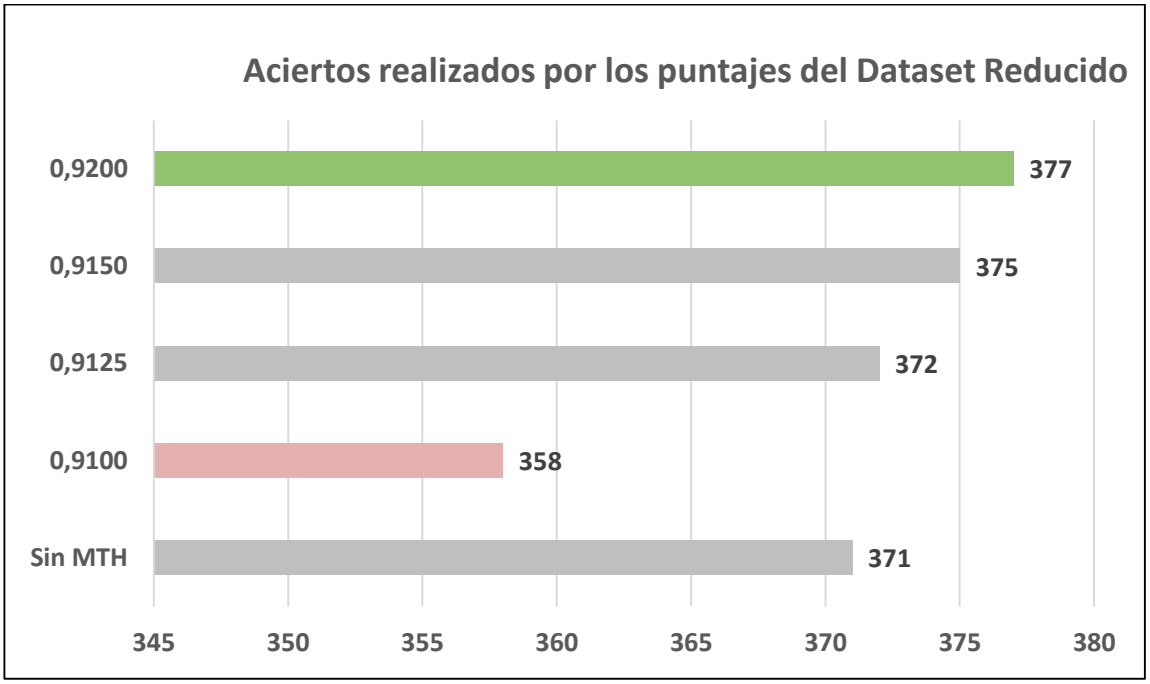


Figura 61: Aciertos realizados en el conjunto de prueba, por los puntajes del Dataset Reducido; Fuente: Elaboración propia (2023).

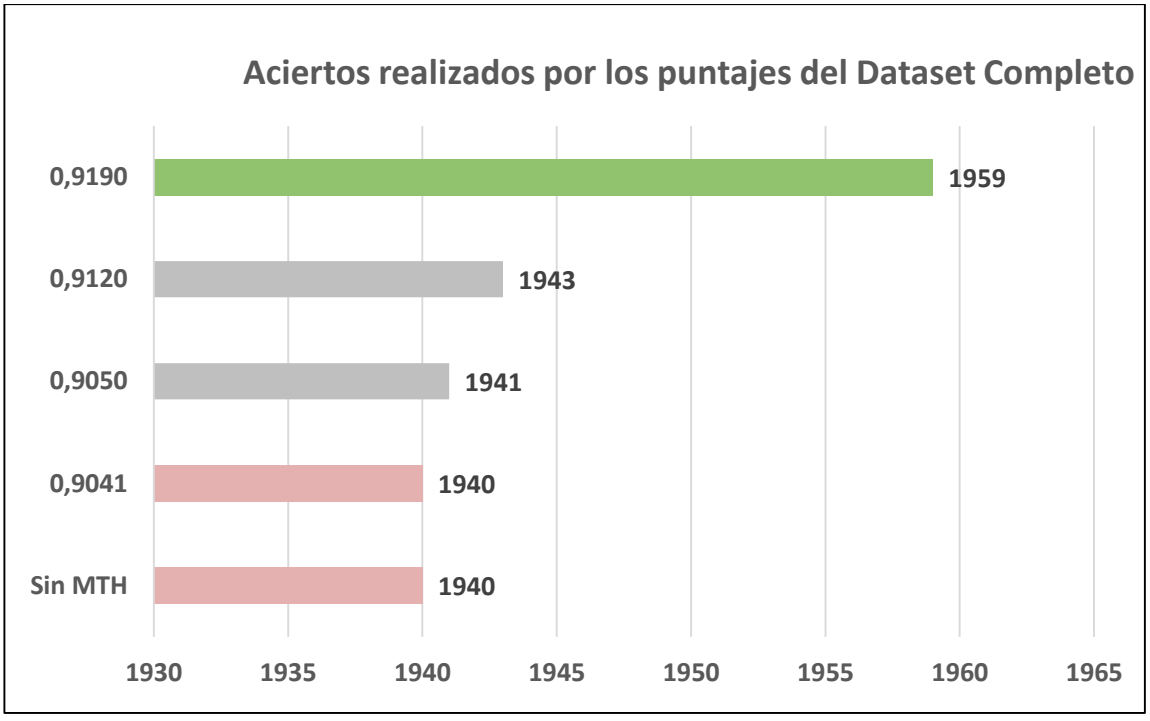


Figura 62: Aciertos realizados en el conjunto de prueba, por los puntajes del Dataset Completo; Fuente: Elaboración propia (2023).

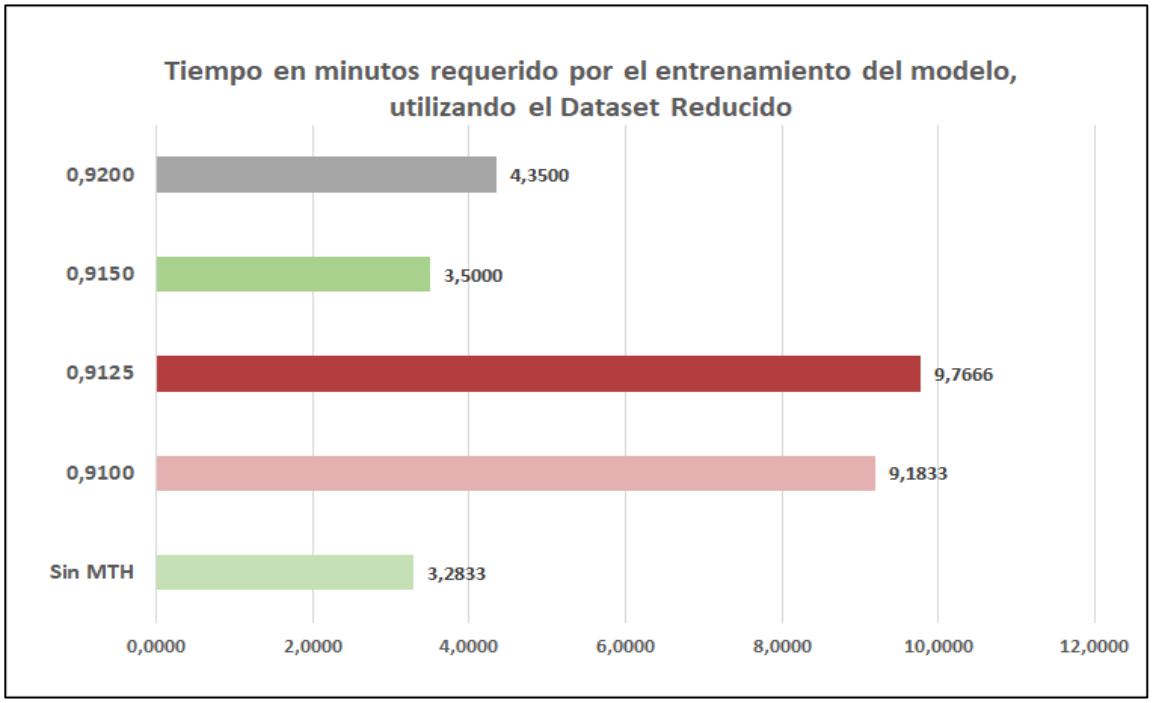


Figura 63: Tiempo en minutos requerido por el entrenamiento del modelo, utilizando el Dataset Reducido; Fuente: Elaboración propia (2023).

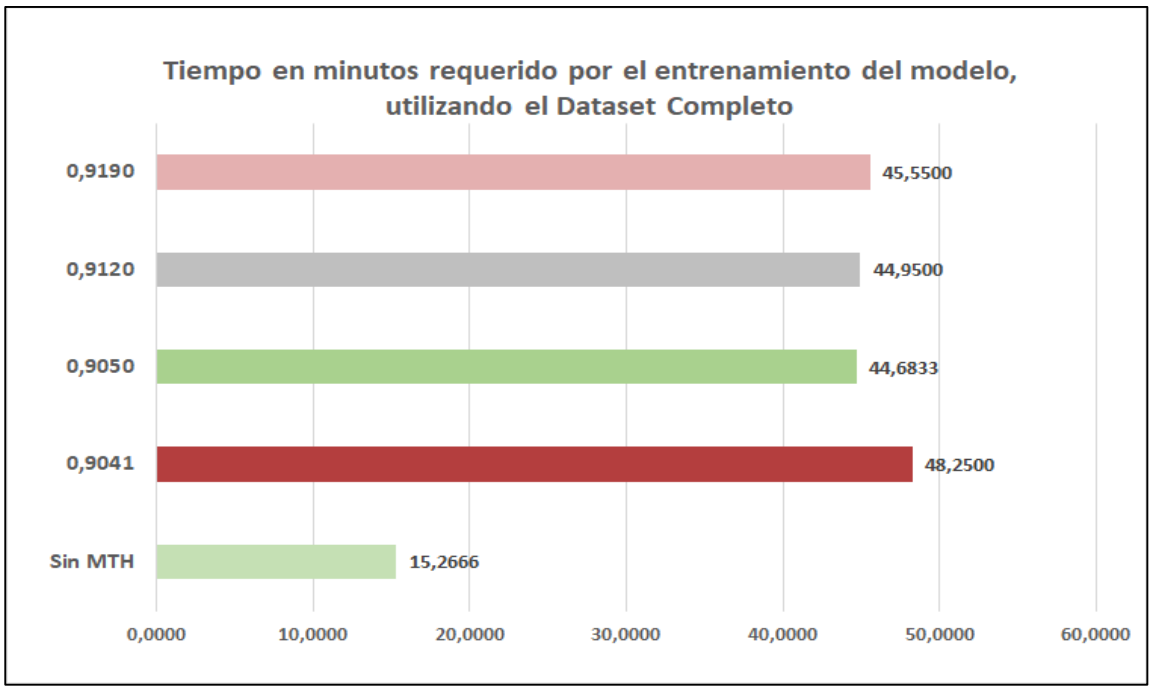


Figura 64: Tiempo en minutos requerido por el entrenamiento del modelo, utilizando el Dataset Completo; Fuente: Elaboración propia (2023).

9 COMPARACIÓN CON OTROS AUTORES

Una de las principales dificultades a la hora de comparar el rendimiento del modelo propuesto es buscar casos realizados en condiciones similares:

- Realizados en un trabajo de clasificación de múltiples clases de radiografías.
- Realizados bajo bases de datos similares a la de Rahman et al. (2022).
- Que hayan o no utilizado un tipo de metaheurística y en qué modo.
- Con suficientes datos asociados al rendimiento final (ej: matriz de confusión).

Debido a las distintas combinaciones presentes en este tipo de ejercicio, se considerarán problemas binarios junto a clasificación múltiple entre tres y cuatro clases, observando el rendimiento de la metaheurística propuesta respecto a la competencia. Entre los trabajos citados se encontrarán:

- *Covid-19: automatic detection from X-ray images utilizing transfer learning with convolutional neural networks (Apostolopoulos y Mpesiana, 2020).* Trabaja con problemas de clasificación binaria y ternaria y evaluados durante diez epochs y con tamaños de lotes iguales a 64.
- *Deep Learning for Screening COVID-19 using Chest X-Ray Images (Basu, Mitra y Saha, 2020).* Presenta resultados con problemas de clasificación cuaternaria en la que se usaron entrenamientos de cien epochs.
- *CovXNet: A multi-dilation convolutional neural network for automatic COVID-19 and other pneumonia detection from chest X-ray images with transferable multi-receptive feature optimization (Mahmud, Rahman y Fattah, 2020).* En este se realiza distintas tareas de clasificación y comparativas entre estas.
- *Metaheuristic-based Deep COVID-19 Screening Model from Chest X-Ray Images (Kaur et al., 2021).* En este se propone la utilización de una metaheurística evolucionaria (SPEA-II) para optimizar los hiperparámetros de una versión modificada de la arquitectura AlexNet.

Junto con esto, se compararán los resultados de Abdelsamee et al. (2022) y Narin (2021), mencionados en 2.6.1 y 2.6.3 respectivamente.

Solución	Exact. F1, Binario	Exact. F1, Ternario	Exact. F1, Cuaternario
Modelo propuesto en el proyecto Sin utilizar metaheurística MRFO (Dataset de 4.000 elementos, VGG19)			***93.000%***
Modelo propuesto en el proyecto Utilizando metaheurística MRFO (Dataset de 4.000 elementos, VGG19)			***94.000%***
Modelo propuesto en el proyecto Sin utilizar metaheurística MRFO (Dataset de 21.165 elementos, VGG19)			***92.000%***
Modelo propuesto en el proyecto Utilizando metaheurística MRFO (Dataset de 21.165 elementos, VGG19)			***93.000%***
Apostolopoulos y Mpesiana (2020) (Sin M.Heurísticas) (Primer dataset, binario y ternario, con VGG19)	**98.750%** (Normal y COVID)	**93.480%** (Normal, Neumonía, COVID)	
Apostolopoulos y Mpesiana (2020) (Sin M.Heurísticas) (Primer dataset, binario y ternario, con MobileNet v2)	**97.400%** (Normal y COVID)	**92.850%** (Normal, Neumonía, COVID)	
Apostolopoulos y Mpesiana (2020) (Sin M.Heurísticas) (Segundo dataset, binario y ternario, con MobileNet v2)	**96.780%** (Normal y COVID)	**94.720%** (Normal, Neumonía, COVID)	
Basu, Mitra y Saha (2020) (Sin M.Heurísticas) (Dataset cuaternarios, VGG19)			**82.980%**
Basu, Mitra y Saha (2020) (Sin M.Heurísticas) (Dataset cuaternarios, AlexNet)			**90.130%**
Basu, Mitra y Saha (2020) (Sin M.Heurísticas) (Dataset cuaternarios, ResNet)			**85.980%**
Mahmud, Rahman y Fattah (2020) (Sin M.Heurísticas) (Dataset con las clases COVID, Normal, Neumonía Viral y Bacterial, y arquitectura VGG19)	**85.300%** (COVID y Normal)	**79.100%** (COVID y Ambas Neumonías)	**80.800%** (Las cuatro clases)
Mahmud, Rahman y Fattah (2020) (Sin M.Heurísticas) (Dataset con las clases COVID, Normal, Neumonía Viral y Bacterial, y arquitectura Inception)	**89.500%** (COVID y Normal)	**84.300%** (COVID y Ambas Neumonías)	**82.900%** (Las cuatro clases)
Mahmud, Rahman y Fattah (2020) (Sin M.Heurísticas) (Dataset con las clases COVID, Normal, Neumonía Viral y Bacterial, y arquitectura Propuesta)	**97.400%** (COVID y Normal)	**89.600%** (COVID y Ambas Neumonías)	**90.200%** (Las cuatro clases)

Kaur et al. (2021) (Hyperparameter Tuning SPEA-II) (Dataset cuaternario con AlexNet, 150 epochs)			**99.130%**
Abdelsamee et al. (2022) (Feature Selection + DTPSO) (Dataset binario mencionado en 2.6.1, 20 epochs)	**99.875%**		
Abdelsamee et al. (2022) (Feature Selection + WOA) (Dataset binario mencionado en 2.6.1, 20 epochs)	**97.985%**		
Abdelsamee et al. (2022) (Feature Selection + GWO) (Dataset binario mencionado en 2.6.1, 20 epochs)	**98.089%**		
Abdelsamee et al. (2022) (Feature Selection + GA) (Dataset binario mencionado en 2.6.1, 20 epochs)	**98.704%**		
Abdelsamee et al. (2022) (Feature Selection + PSO) (Dataset binario mencionado en 2.6.1, 20 epochs)	**98.137%**		
Narin, A. (2021) (SVM sin Feature Selection) (Dataset ternario mencionado en 2.6.3)		**97.720%**	
Narin, A. (2021) (SVM + Feature Selection con ACO) (Dataset ternario mencionado en 2.6.3)		**98.170%**	
Narin, A. (2021) (SVM + Feature Selection con PSO) (Dataset ternario mencionado en 2.6.3)		**98.170%**	

Tabla 25: Comparación de resultados propios junto a otros autores;
Fuente: Elaboración propia en base a la información citada (2023).

En base a la información recopilada, los resultados obtenidos por la propuesta del presente informe se destacan por las siguientes diferencias respecto a las propuestas de clasificación sin utilizar metaheurísticas:

- Obtener mejores calificaciones en comparación a Basu, Mitra y Saha (2020), y Mahmud, Rahman y Fattah (2020) en sus versiones cuaternarias y ternarias, las cuales rondan en una diferencia de unidades.
- Obtener calificaciones relativamente mejores a Apostolopoulos y Mpesiana (2020) en clasificación ternaria, y Mahmud, Rahman y Fattah (2020) en clasificación binaria, pues ambos presentan casos con mejores porcentajes. Para el primero, son las resoluciones con VGG19 y MobileNetv2. Para el segundo, son aquellas con VGG19 e Inception.
- Las calificaciones propias son peores que los resultados binarios sin metaheurística de Apostolopoulos y Mpesiana (2020), y uno de los resultados binarios de Mahmud, Rahman y Fattah (2020).

En cuanto a diferencias percibidas con las soluciones y propuestas que utilizan metaheurísticas, se observan peores calificaciones en comparación a Kaur et al. (2021), Abdelsamee et al. (2022), y Narin (2021), y en donde domina claramente la solución de la primera mención. Estos resultados pueden parecer desalentadores, sin embargo, se hará énfasis en los siguientes aspectos:

- Los problemas de clasificación binaria, ternaria y cuaternaria difieren considerablemente en cuanto a su complejidad. Una clasificación binaria no requerirá aprender tantas características y rasgos como lo podría requerir una cuaternaria, y es de esperar que obtenga mejores resultados.
- La diferencia en elementos disponibles en la base de datos también afectará a los resultados de esta, y cada uno de los ejemplos anteriores trabaja bajo distintos tipos o versiones de bases de datos, las cuales no necesariamente utilizan categorías idénticas a las de este informe.
- Junto a lo anterior, resultados como los de Kaur et al. y Abdelsamee et al. se benefician de la utilización de un mayor número de recursos disponibles, lo que se traduce en un mayor número de epochs e iteraciones evaluadas para las metaheurísticas que utilizan: 150 iteraciones para Kaur et al. y 20 epochs con 80 iteraciones cada uno para Abdelsamee et al., en comparación a los 10 epochs y 2 iteraciones postuladas en el uso de MRFO de este proyecto.

Por lo tanto, se considerarán satisfactorios, para los casos individuales estudiados, los resultados obtenidos por la propuesta de arquitectura VGG-19 y aumentación de hiperparámetros mediante la metaheurística bioinspirada conocida como Forrajeo de Mantarraya. Sin embargo, se sugiere el estudio y potenciamiento de este planteamiento mediante distintas técnicas y observaciones que se han obtenido a través del desarrollo de este proyecto. Cada uno de estos se considera como segundos casos aceptables, al mejorar el valor de precisión o exactitud obtenido, pero aumentando el tiempo de ejecución.

10 RECOMENDACIONES Y OBSERVACIONES FUTURAS

En base las observaciones realizadas durante la elaboración, tanto del presente informe como del modelo propuesto, se sugiere tomar en cuenta los siguientes puntos para futuras experimentaciones.

1. Investigar la aplicación de procesos de segmentación a la arquitectura de VGG-19, que permitan desechar características innecesarias sin perder valores considerables de precisión como se observó anteriormente.

2. Generar alternativas a la base de datos trabajada, para así poder realizar comparaciones de su funcionamiento en casos de clasificación binaria, ternaria o cuaternaria, esto inclusive bajo distintas combinaciones, tales como las realizadas por Mahmud, Rahman y Fattah (2020). Estas pueden equilibrarse mediante un segundo trabajo de aumentación de imagen, de manera que cada clase pueda tener una cantidad similar de elementos.

3. Debatir y evaluar distintas configuraciones de la metaheurística trabajada, buscando a su vez probar estas con un mayor número de iteraciones y epochs asociados. Junto con esto, contrastar su funcionamiento en la selección de características (Feature Selection) versus la optimización de hiperparámetros (Hyperparameter Optimization), y en el segundo caso, cuáles de estos modificar o no, de acuerdo a lo considerado en 6.6.

Finalmente, y recordando lo mencionado en el punto anterior, es de considerable dificultad la comparación de un método propio frente al estado del arte y las soluciones de otros autores, por lo que se sugiere, para futuros experimentos, buscar recrear las arquitecturas y modelos propuestos por otros autores, evaluarlos bajo las distintas bases de datos propias, y finalmente, comparar estos nuevos resultados contra las propuestas personales. De esta manera, podrán eliminarse distintos factores que interfieran con un correcto análisis de lo visto.

11 CONCLUSIONES DEL PROYECTO DE TÍTULO

A través de este informe de proyecto de título, se ha podido evidenciar el interés que existe en resolver problemas de clasificación de imágenes mediante la aplicación de distintas técnicas asociadas al aprendizaje automático, y el beneficio que estas técnicas pueden representar a la hora de ser aplicadas. Considerando el contexto actual, cuatro años desde que inició la pandemia global de COVID-19, causada por el virus SARS-CoV-2, es fácilmente entendible que múltiples desarrolladores, programadores e interesados en las áreas de salud e informática, deseen generar soluciones que ayuden a la efectiva identificación de esta y otras enfermedades pulmonares existentes o que puedan conocerse con el pasar de los años, para así poder dar un soporte y apoyo a los profesionales y encargados de la categorización de pacientes, en especial en casos tan importantes al estar relacionados con la salud de la población.

Con la presentación de este trabajo de investigación y desarrollo, se propuso, estudiaron y evaluaron soluciones para resolver el problema de reconocimiento de estas enfermedades mediante bases de datos de radiografías y distintas técnicas, tales como la metaheurística basada en el Forrajeo de Mantarraya. Técnicas y soluciones, las cuales, en sus versiones más óptimas, alcanzaron altos niveles de efectividad, pero que sin duda, se beneficiarán de un mayor número de experimentaciones, junto al ajuste y retroalimentación en base a observaciones y lecciones aprendidas durante el desarrollo y aplicación de estas.

Finalmente, se espera que el estudio y aplicación de este tipo de problemas incentive al desarrollo de nueva tecnología, que pueda ser capaz de potenciar considerablemente la capacidad de detección de distintas afecciones presentes en imágenes, y cuyos beneficios no sólo se limiten al campo de radiografías y escaneos, sino que también puedan expandirse a otros campos mencionados anteriormente, como la detección de tumores, heridas y lesiones cancerígenas

12 BIBLIOGRAFÍA Y RECURSOS UTILIZADOS

1. World Health Organization. (2023). *WHO Coronavirus (COVID-19 Dashboard)*. Recuperado el 26 de julio de 2023 de https://covid19.who.int/
2. Oracle Cloud. (2023). *¿Qué es el aprendizaje automático?* https://www.oracle.com/cl/artificial-intelligence/machine-learning/what-is-machine-learning/
3. May, M. (2021). *Eight ways machine learning is assisting medicine.* Nat Med 27, 2–3. https://doi.org/10.1038/s41591-020-01197-2
4. Registro Nacional de Inmunizaciones. (2023). *Resultados Campaña Influenza 2022.* https://www.cenabast.cl/wp-content/uploads/2022/09/Presentacion-Proveedores-Influenza-2023-18.08.22.pdf
5. MedlinePlus. (2023). *Lesiones y enfermedades del tórax. https://medlineplus.gov/spanish/chestinjuriesanddisorders.html*
6. IntegraMédica. (2023). *Servicios - Radiografía (Rx).* https://www.integramedica.cl/integramedica/servicios/examenes/radiologia
7. AboutKidsHealth. (2023). *Pneumonia*. https://www.aboutkidshealth.ca/pneumonia
8. IBM Cloud. (2023). *¿Qué son las redes neuronales convolucionales? https://www.ibm.com/es-es/topics/convolutional-neural-networks*
9. Aprende Machine Learning. (2018). *¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador. https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/*
10. Hewage, R. (2020). *Extract Features, Visualize Filters and Feature Maps in VGG16 and VGG19 CNN Models.* https://towardsdatascience.com/extract-features-visualize-filters-and-feature-maps-in-vgg16-and-vgg19-cnn-models-d2da6333edd0

11. Medical Imaging Databank of the Valencia Region. (2023). *BIMCV-COVID19+: a large annotated dataset of RX and CT images of COVID19 patients.* https://bimcv.cipf.es/bimcv-projects/bimcv-covid19/#1590858128006-9e640421-6711
12. Wu, S. (2023). *Machine Learning Data: Do You Really Have Rights to Use It?* https://www.airoboticslaw.com/blog/machine-learning-data-do-you-really-have-rights-to-use-it
13. MathWorks. (2023). *VGG-19 convolutional neural network.* https://www.mathworks.com/help/deeplearning/ref/vgg19.html
14. Simonyan & Zisserman. (2015). *Very Deep Convolutional Networks for Large-Scale Visual Recognition.* https://www.robots.ox.ac.uk/~vgg/research/very_deep/
15. ImageNet. (2023). *About ImageNet.* https://image-net.org/about.php
16. Shrestha, D. (2021). *ImageNet Gets A Privacy Overhaul, What About Other Datasets? https*://analyticsindiamag.com/imagenet-gets-a-privacy-overhaul-what-about-other-datasets/
17. Iryna, B. (2022). *No PCR, no problem: how COVID can be diagnosed with X-rays.* https://theconversation.com/no-pcr-no-problem-how-covid-can-be-diagnosed-with-x-rays-175947
18. Universidad de Chile. (2020). *¿Por qué en Chile usamos el test RT-PCR si existen 6 técnicas para la detección de COVID-19?* https://uchile.cl/noticias/164485/tecnicas-para-la-deteccion-covid-por-que-en-chile-usamos-rt-pcr
19. IntegraMédica. (2023). *Examen PCR Covid-19: Todo lo que necesitas conocer sobre tu examen.* https://www.integramedica.cl/integramedica/examen-pcr
20. MedlinePlus. (2023). *¿Qué son las pruebas de PCR?* https://medlineplus.gov/spanish/pruebas-de-laboratorio/pruebas-de-pcr/

21. Rahman et al. (2022). *COVID-19 Radiography Database.* https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database
22. Hafez, A. (2022). *Covid-19 Radiology | VGG19 | f1-score: 95%.* https://www.kaggle.com/code/ahmedtronic/covid-19-radiology-vgg19-f1-score-95/notebook
23. Kaushik, A. (2023). *Understanding the VGG19 Architecture.* https://iq.opengenus.org/vgg19-architecture/
24. Singh, Meitei, Majumder. (2020). *Short PCG classification based on deep learning.* https://www.sciencedirect.com/topics/engineering/convolutional-layer
25. Redacción KeepCoding. (2022). *Capas de pooling en una red neuronal convolucional.* https://keepcoding.io/blog/capas-pooling-red-neuronal-convolucional/
26. Zheng, Yang. (2018). *Breast cancer screening using convolutional neural network and follow-up digital mammography.* https://www.researchgate.net/publication/325137356_Breast_cancer_screening_using_convolutional_neural_network_and_follow-up_digital_mammography
27. Amazon Web Services. (2023). *¿Qué es Python?* https://aws.amazon.com/es/what-is/python/
28. Frías, N. (2020). *Algoritmo Híbrido basado en el sistema de Colonia de Hormigas y Machine Learning para la Resolución del EMVRP.*
29. Kulhary, R. (2022). *OpenCV – Overview*. https://www.geeksforgeeks.org/opencv-overview/
30. seaborn. (2023). *seaborn: statistical data visualization.* https://seaborn.pydata.org/
31. Sancho, F. (2021). *Metaheurísticas.* http://www.cs.us.es/~fsancho/?e=207

32. Sadigh et al. (2012). *A graphical classification of metaheuristics*. https://www.researchgate.net/figure/A-graphical-classification-of-metaheuristics_fig5_272145791
33. Game et al. (2020). *Bio-inspired Optimization: metaheuristic algorithms for optimization.* https://arxiv.org/abs/2003.11637
34. Suárez, A. (2013). *Una aproximación a la heuristica y metaheuristicas.* https://revistas.uan.edu.co/index.php/ingeuan/article/view/217/179
35. Devika, G. & Gowda, A. (2022). *Bio-inspired Optimization: Algorithm, Analysis and Scope of Application.* https://www.intechopen.com/chapters/84300
36. Dréo, J. (2006). *Shortest path find by an ant colony.* https://commons.wikimedia.org/wiki/File:Aco_branches.svg
37. Azizi, R. (2014). *Empirical Study of Artificial Fish Swarm Algorithm.* https://www.semanticscholar.org/paper/Empirical-Study-of-Artificial-Fish-Swarm-Algorithm-Azizi/cce58c87b595abcc9e6d351861bb5cd2ffac2d4c
38. Abdelsamee et al. (2022). *Metaheuristic Optimization Through Deep Learning Classification of COVID-19 in Chest X-Ray Images*. https://www.researchgate.net/publication/361322613_Metaheuristic_Optimization_Through_Deep_Learning_Classification_of_COVID-19_in_Chest_X-Ray_Images
39. Riaz, M. & Bashir, M. & Younas, I. (2022). *Metaheuristics based COVID-19 detection using medical images: A review*. https://www.sciencedirect.com/science/article/pii/S0010482522001366
40. Sidey-Gibbons, J. & Sidey-Gibbons, C. (2019). *Machine learning in medicine: a practical introduction.* https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/s12874-019-0681-4
41. Analytics Vidhya. (2021). *Step-by-Step guide for Image Classification on Custom Datasets.* https://www.analyticsvidhya.com/blog/2021/07/step-by-step-guide-for-image-classification-on-custom-datasets/

42. CHM Computer History Museum. (2023). Alan Kay. https://computerhistory.org/profile/alan-kay/
43. Jiménez, J. & Pavón, J. (2014). Applications of metaheuristics in real-life problems. https://link.springer.com/article/10.1007/s13748-014-0051-8
44. Golnoori, F. (2023). Metaheuristic algorithm-based hyper-parameters optimization for skin lesion classification. https://github.com/herodesigngit/Metaheuristic-algorithm-based-hyper-parameters-optimization-for-skin-lesion-classification
45. Dezube, R. (2021). https://www.msdmanuals.com/es/professional/trastornos-pulmonares/procedimientos-diagn%C3%B3sticos-y-terap%C3%A9uticos-pulmonares/estudios-por-la-imagen-del-t%C3%B3rax
46. Willis et al. (2023). Artifacts and misadventures in digital radiography. https://appliedradiology.com/articles/artifacts-and-misadventures-in-digital-radiography
47. Pérez, A. (2021). Estudio de viabilidad de un proyecto: ¿qué es y cómo hacerlo? https://www.obsbusiness.school/blog/estudio-de-viabilidad-de-un-proyecto-estructura-e-importancia
48. Guo, E. (2022). A Roomba recorded a woman on the toilet. How did screenshots end up on Facebook? https://www.technologyreview.com/2022/12/19/1065306/roomba-irobot-robot-vacuums-artificial-intelligence-training-data-privacy/
49. Marr, B. (2023). Is Generative AI Stealing From Artists? https://www.forbes.com/sites/bernardmarr/2023/08/08/is-generative-ai-stealing-from-artists/?sh=3c21684f5d1e
50. Patil, S. & Rajat (2020). Does Bit-Depth of an image affects the convolutional neural networks? https://stackoverflow.com/questions/61843146/does-bit-depth-of-an-image-affects-the-convolutional-neural-networks

51. Sudhakar, S. (2017). Custom Image Augmentation. https://towardsdatascience.com/image-augmentation-14a0aafd0498
52. Sharma, P. (2023). Computer Vision Tutorial: A Step-by-Step Introduction to Image Segmentation Techniques (Part 1). https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/
53. Yumi's Blog (2018). Learn about ImageDataGenerator. https://fairyonice.github.io/Learn-about-ImageDataGenerator.html
54. Chugh, S. (2020). Dump Keras-ImageDataGenerator. Start Using TensorFlow-tf.data (Part 1). https://medium.com/swlh/dump-keras-imagedatagenerator-start-using-tensorflow-tf-data-part-1-a30330bdbca9
55. Chugh, S. (2020). Dump Keras-ImageDataGenerator. Start Using TensorFlow-tf.data (Part 2). https://towardsdatascience.com/dump-keras-imagedatagenerator-start-using-tensorflow-tf-data-part-2-fba7cda81203
56. Rath, S. (2022). Faster Image Augmentation in TensorFlow using Keras Layers. https://debuggercafe.com/faster-image-augmentation-in-tensorflow-using-keras-layers/
57. Prasad, S. (2020). Keras 'Random' preprocessing layers do not work on cloud TPU. https://github.com/tensorflow/tensorflow/issues/42454
58. Superb AI. (2023). When More Isn't an Option: Data Augmentation Techniques for Rare Cases in Computer Vision Models. https://superb-ai.com/blog/when-more-isn-t-an-option-data-augmentation-techniques-for-rare-cases-in-computer-vision-models/
59. Awan, A. (2022). A Complete Guide to Data Augmentation. https://www.datacamp.com/tutorial/complete-guide-data-augmentation
60. Albumentations. (2023). What is image augmentation and how it can improve the performance of deep neural networks. https://albumentations.ai/docs/introduction/image_augmentation/

61. Zhao, W., Zhang, Z. & Wang, L. (2019). Manta ray foraging optimization: An effective bio-inspired optimizer for engineering applications. https://www.sciencedirect.com/science/article/abs/pii/S0952197619302593
62. Daqaq et al. (2022). Enhanced Chaotic Manta Ray Foraging Algorithm for Function Optimization and Optimal Wind Farm Layout Problem https://www.researchgate.net/publication/362188806_Enhanced_Chaotic_Manta_Ray_Foraging_Algorithm_for_Function_Optimization_and_Optimal_Wind_Farm_Layout_Problem
63. Abdullahi et al. (2023). Manta Ray Foraging Optimization Algorithm: Modifications and Applications. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10124217
64. Balaha, M. (2022). CNN Hyperparameters Optimization using Metaheuristic Optimizers. https://github.com/HossamBalaha/CNN-Hyperparameters-Optimization-using-Metaheuristic-Optimizers
65. cMax. (2023). ¿Qué es Epoch en Machine Learning? https://ciberseguridadmax.com/epoch/
66. Kaur et al. (2021). Metaheuristic-based Deep COVID-19 Screening Model from Chest X-Ray Images. https://www.hindawi.com/journals/jhe/2021/8829829/
67. Apostolopoulos, I. & Mpesiana, T. (2020). Covid-19: automatic detection from X-ray images utilizing transfer learning with convolutional neural networks. https://link.springer.com/article/10.1007/s13246-020-00865-4
68. Basu, S., Mitra, S. & Saha, N. (2020). Deep Learning for Screening COVID-19 using Chest X-Ray Images. https://arxiv.org/pdf/2004.10507.pdf
69. Mahmud, T., Rahman, Md. & Fattah, S. (2020). CovXNet: A multi-dilation convolutional neural network for automatic COVID-19 and other pneumonia detection from chest X-ray images with transferable multi-receptive feature optimization. https://www.sciencedirect.com/science/article/pii/S001048252030225

13 APÉNDICE

1. Codificación en lenguaje Python para la Segmentación de Imágenes

Realizado de manera previa a un trabajo de clasificación de radiografías pulmonares.

Lo que se busca aquí es poder segmentar las imágenes antes de trabajarlas con la red neuronal convolucional y metaheurísticas.
Se les aplicará una "máscara" que removerá las zonas innecesarias de la radiografía, conservando sólo los pulmones y área del tórax.

Para asegurar el correcto funcionamiento, ejecutar este archivo jupyter notebook en la interfaz de Kaggle, y con las siguientes propiedades activadas:

Notebook Data:

- **Input:** covid19-radiography-database de TAWSIFUR RAHMAN (zip, 816MB).

Notebook Options:

- **Accelerator:** Ninguno. Este proceso no se beneficia de utilizar aceleradores.
- **Internet:** Internet on (requiere verificar la cuenta de Kaggle para utilizarse).

Se comienza definiendo la ruta de la base de datos original, y limpiando la carpeta output de Kaggle

```
rutaDatabase  = '/kaggle/input/covid19-radiography-database/COVID-19_Radiography_Dataset'

def limpiaCarpetaOutput(carpeta):
    for archivo in os.listdir(carpeta):
        rutaArchivo = os.path.join(carpeta, archivo)
        try:
            if os.path.isfile(rutaArchivo):
                os.unlink(rutaArchivo)
            elif os.path.isdir(rutaArchivo):
                limpiaCarpetaOutput(rutaArchivo)
                os.rmdir(rutaArchivo)
        except Exception as e:
            print(e)

rutaCarpeta = '/kaggle/working'
try:
    limpiaCarpetaOutput(rutaCarpeta)
    os.rmdir(rutaCarpeta)
except:
    pass
```

Se importa lo necesario y requerido.

- CV2 es una biblioteca de visión artificial, manejo de imágenes, detección, entre otros.
- OS permite el manejo de carpetas.
- Glob permite funciones de manejo de rutas de carpetas.
- PIL permite procesar imágenes, crearlas desde arreglos, etc.

```
import cv2
import os
import glob
from PIL import Image
```

Se crean las carpetas en que se exportarán las imágenes trabajadas.

En este ejemplo, se conocen previamente las categorías gracias a la confección de la base de datos, por lo tanto, se establecen manualmente.

```
os.makedirs('COVID'          , exist_ok=True)
os.makedirs('Lung_Opacity'   , exist_ok=True)
os.makedirs('Normal'         , exist_ok=True)
os.makedirs('Viral Pneumonia', exist_ok=True)
```

Se obtienen las rutas a leer.

Tanto para imágenes como para máscaras.

```
rutasImagenes= []
rutasMascaras= []

for (directorioRuta, directorioNombres, archivosNombres) in os.walk(rutaDatabase):
    directorioNombres.sort()
    archivosPosibles = os.path.join(directorioRuta, "images/*.png")

    for archivo in sorted(glob.glob(archivosPosibles)):
        rutasImagenes.append(archivo)

for (directorioRuta, directorioNombres, archivosNombres) in os.walk(rutaDatabase):
    directorioNombres.sort()
    archivosPosibles = os.path.join(directorioRuta, "masks/*.png")

    for archivo in sorted(glob.glob(archivosPosibles)):
        rutasMascaras.append(archivo)
```

Para cada ruta, se realiza lo siguiente:

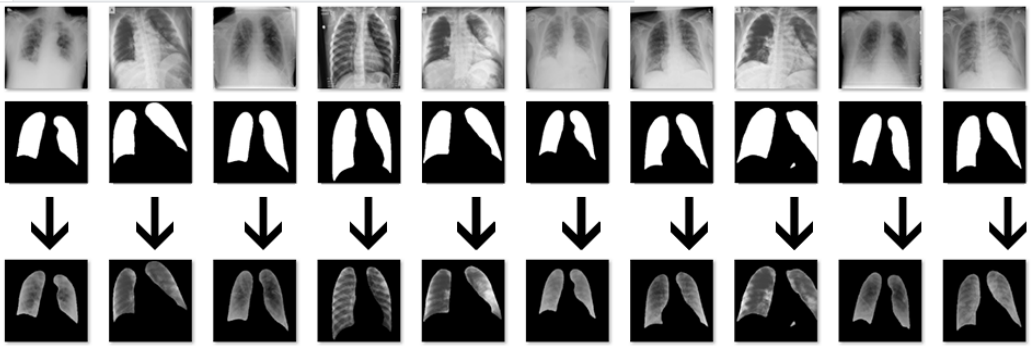
- Se lee, se obtiene dónde se exportará el resultado dentro de Kaggle.
- Se lee la imagen y su máscara con cv2. Este las transformará a arreglos de Python.
- A ambas se les redimensiona, ya que las radiografías se encuentran a 299x299 pixeles, las máscaras a 256x256, y VGG-19 trabaja idealmente con imágenes de 224x224.
- Se utiliza la función bitwise_and de cv2 para aplicar la máscara a la imagen original.
- El resultado, en arreglos de Python, es reconvertido a Imagen y exportado.

```
for contador in range (0, len(rutasImagenes)):
    exportado = os.path.relpath(rutasImagenes[contador],rutaDatabase)
    exportado = exportado.replace("/images","")

    imgOrig = cv2.imread(rutasImagenes[contador],cv2.IMREAD_GRAYSCALE)
    imgMask = cv2.imread(rutasMascaras[contador],cv2.IMREAD_GRAYSCALE)

    imgOrig = cv2.resize(imgOrig, (224,224), interpolation = cv2.INTER_AREA)
    imgMask = cv2.resize(imgMask, (224,224), interpolation = cv2.INTER_AREA)

    imgOrig = cv2.bitwise_and(imgMask, imgOrig)

    imagenFinal = Image.fromarray(imgOrig)
    imagenFinal.save(exportado)
    print("Progreso de la segmentación: ",contador+1," de ",len(rutasImagenes)," máscaras listas.", end='\r')
```

Comprimiendo y descargando.

Los resultados quedarán en cuatro carpetas distintas que podrán ser descargadas.
Para facilitar el proceso, se comprimen en un .zip que Kaggle permitirá descargar de una sola vez.

```
print("Comienza el proceso de creación del .zip...")
!zip -r descargaConMasking.zip '/kaggle/working/' -q
print("Fin del proceso de creación del .zip...")
```



2. Codificación en lenguaje Python para la CNN y Metaheurística MRFO

Realizado como parte de un trabajo de clasificación de radiografías pulmonares.

Para asegurar el correcto funcionamiento, ejecutar este archivo jupyter notebook en la interfaz de Kaggle, y con las siguientes propiedades activadas:

Notebook Data:

- **Input:** covid19-radiography-database de TAWSIFUR RAHMAN (zip, 816MB) o las versiones reducidas o minimizadas de la base de datos.

Notebook Options:

- **Accelerator:** TPU VM v3-8 es la opción más efectiva. De lo contrario, utilizar GPU P100.
- **Internet:** Internet on (requiere verificar la cuenta de Kaggle para utilizarse).

Importación de funciones necesarias para el funcionamiento de Tensorflow

```
import tensorflow as tf
tf.config.set_soft_device_placement(True)
try:
    # Se detecta el sistema de TPU en Kaggle.
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.TPUStrategy(tpu)
else:
    # Estrategia predeterminada en TensorFlow.
    strategy = tf.distribute.get_strategy()

AUTO = tf.data.experimental.AUTOTUNE
print("REPLICAS: ", strategy.num_replicas_in_sync)
```

Se importa lo necesario y requerido.

```
# Directorios, medición de tiempo, operaciones con ficheros y rutas, e iteración eficiente.
from statistics import mean
import os
import time
import shutil
import pathlib
import itertools
import random

# Herramientas de manejo de datos, librerías de tablas y gráficos, y printeo con colores.
!pip install seaborn
!pip install colorama
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')3
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from colorama import Fore

# Librerías de deep-learning.
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import *
from tensorflow.keras.models import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.metrics import *
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import *
```

```python
# Generar rutas de acceso con etiquetado.
def DefinirRutas(rutaDatabase):
    rutasArchivos = []
    etiquetas = []

    directorios = os.listdir(rutaDatabase)
    for directorio in directorios:
        rutaDirectorio = os.path.join(rutaDatabase, directorio)
        # Sólo chequea las cuatro carpetas del dataset (ignora los archivos de metadata en este)
        if pathlib.Path(rutaDirectorio).suffix != '':
            continue

        listaArchivos = os.listdir(rutaDirectorio)
        for archivo in listaArchivos:
            rutaArchivo = os.path.join(rutaDirectorio, archivo)

            # Chequea si existen más carpetas
            if pathlib.Path(rutaDirectorio).suffix == '':

                # Cada clase (COVID, LUNG, NORMAL, PNEUMONIA) cuenta con una carpeta para las radiografías y otra para las máscaras.
                # Siempre se excluyen (en caso de importar la base de datos segmentadas, esta ya trae las imágenes listas...)

                if pathlib.Path(rutaArchivo).parts[-1] == 'masks' or pathlib.Path(rutaArchivo).parts[-1] == 'Masks' or pathlib.Path(rutaArchivo).parts[-1] == 'MASKS':
                    continue

                else:
                    archivoSistema = os.listdir(rutaArchivo)
                    for archivoVisto in archivoSistema:
                        rutaExtendida = os.path.join(rutaArchivo, archivoVisto)
                        rutasArchivos.append(rutaExtendida)
                        etiquetas.append(directorio)

            else:
                rutasArchivos.append(rutaArchivo)
                etiquetas.append(directorio)

    return rutasArchivos, etiquetas
```

```python
# Concatena las rutas con etiquetas en un solo dataframe (que será el que se utilizará en el modelo)
def DefinirDataframe(archivos, clases):
    serieRutas = pd.Series(archivos, name= 'rutasArchivos')
    serieEtiquetas = pd.Series(clases, name='etiquetas')
    return pd.concat([serieRutas, serieEtiquetas], axis= 1)

# Divide el dataframe a entrenamiento, validación y testeo. Está hecha para trabajar con distintos tamaños, pero en este ejercicio se usará específicamente 80/10/10.
def RepartirDatos(rutaDatabase, tamEntrenamiento, tamValidacion):

    archivos, clases = DefinirRutas(rutaDatabase)
    dataframeRadio = DefinirDataframe(archivos, clases)

    estratificacion = dataframeRadio['etiquetas']
    dfEntrenamiento, dfRestante = train_test_split(dataframeRadio,  train_size= tamEntrenamiento, shuffle= True, random_state= semillaGlobal, stratify= estratificacion)

    estratificacion = dfRestante['etiquetas']
    dfValidacion, dfPrueba = train_test_split(dfRestante,  train_size= tamValidacion, shuffle= True, random_state= semillaGlobal, stratify= estratificacion)

    return dfEntrenamiento, dfValidacion, dfPrueba
```

Funciones de llamada de vuelta (Callback)

```python
class CallbackCNN(keras.callbacks.Callback):
    def __init__(self, model, patience, stop_patience, threshold, factor, batches, epochs, ask_epoch):
        super(CallbackCNN, self).__init__()
        self.model  = model
        self.patience = patience                                        # Cuántos epochs transcurren sin mejora hasta que se reajuste el learning rate.
        self.stop_patience = stop_patience                              # Cuántas veces se ajusta el learning rate sin mejoras para parar el entrenamiento.
        self.threshold = threshold                                      # Límite del training accuracy dependiendo de lo anterior y validation loss.
        self.factor = factor                                            # Factor para reducir el learning rate.
        self.batches = batches                                          # Número de lotes en cada epoch.
        self.epochs = epochs
        self.ask_epoch = ask_epoch
        self.ask_epoch_initial = ask_epoch                              # Guardar el valor inicial si se resetea el entrenamiento.

        # Variables de callback.
        self.count = 0                                                  # Cuántas veces se ha reducido el learning rate sin mejoras.
        self.stop_count = 0
        self.best_epoch = 1                                             # Epoch con el menor loss.
        self.initial_lr = float(tf.keras.backend.get_value(model.optimizer.lr)) # Obtén el learning rate inicial y guárdalo.
        self.highest_tracc = 0.0                                        # La training accuracy más alta será cero inicialmente.
        self.lowest_vloss = np.inf                                      # La validation loss más baja será infinito inicialmente.
        self.best_weights = self.model.get_weights()                    # Mejores pesos -> los iniciales.
        self.initial_weights = self.model.get_weights()                 # Guardar los pesos iniciales si se resetea.

    # Función de inicio del entrenamiento.
    def on_train_begin(self, logs= None):
        self.ask_permission = 1

        msg = '{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}{8:10s}{9:^8s}'.format('Epoch', 'Loss', 'Accuracy', 'V_loss', 'V_acc', 'LR', 'Sig. LR', 'Monitor','% Mejora', 'Duración')
        print(msg)
        self.start_time = time.time()

        self.times = []
```

```
    def on_train_end(self, logs= None):
        stop_time = time.time()
        tr_duration = stop_time - self.start_time
        hours = tr_duration // 3600
        minutes = (tr_duration - (hours * 3600)) // 60
        seconds = tr_duration - ((hours * 3600) + (minutes * 60))

        msg = f'Tiempo total de entrenamiento: {str(hours)} horas, {minutes:4.1f} minutos, {seconds:4.2f} segundos'
        print(msg)

        # Ajusta los pesos del modelo a los mejores pesos hasta ahora.
        self.model.set_weights(self.best_weights)


    def on_train_batch_end(self, batch, logs= None):
        # obtén la accuracy y loss del lote.
        acc = logs.get('accuracy') * 100
        loss = logs.get('loss')

        # printea en la misma linea para mostrar el progreso de lotes.
        msg = '{0:20s}procesando lote {1:} de {2:5s}-   accuracy=  {3:5.3f}   -   loss: {4:8.5f}'.format(' ', str(batch), str(self.batches), acc, loss)
        print(msg, '\r', end= '')


    def on_epoch_begin(self, epoch, logs= None):
        self.ep_start = time.time()
        self.epoch_time_start = time.time()
```

```
# Métodos que se llaman al final de cada epoch
def on_epoch_end(self, epoch, logs= None):
    ep_end = time.time()
    duration = ep_end - self.ep_start
    self.times.append(time.time() - self.epoch_time_start)
    tiempos.append(time.time() - self.epoch_time_start)

    lr = float(tf.keras.backend.get_value(self.model.optimizer.lr)) # obtén el learning rate actual
    current_lr = lr
    acc = logs.get('accuracy')  # obtén la accuracy de entrenamiento
    v_acc = logs.get('val_accuracy')  # obtén la accuracy de validación
    loss = logs.get('loss')  # obtén el loss de entrenamiento para este epoch
    v_loss = logs.get('val_loss')  # obtén el loss de validación para este epoch

    if acc < self.threshold:
        monitor = 'accuracy'
        if epoch == 0:
            pimprov = 0.0
        else:
            pimprov = (acc - self.highest_tracc ) * 100 / self.highest_tracc

        if acc > self.highest_tracc: # mejora de accuracy en entrenamiento para este epoch
            self.highest_tracc = acc
            self.best_weights = self.model.get_weights()
            self.count = 0
            self.stop_count = 0
            if v_loss < self.lowest_vloss:
                self.lowest_vloss = v_loss
            self.best_epoch = epoch + 1  # pon el valor del mejor epoch para este epoch

        else:
            # si entra a este caso, la training accuracy no mejoró. chequea si se acabó la paciencia para ajustar el learning rate...
            if self.count >= self.patience - 1: # ajustar learning rate
                lr = lr * self.factor
                tf.keras.backend.set_value(self.model.optimizer.lr, lr)
                self.count = 0
                self.stop_count = self.stop_count + 1
                self.count = 0
                if v_loss < self.lowest_vloss:
                    self.lowest_vloss = v_loss
            else:
                self.count = self.count + 1 # incrementar el contador de paciencia
```

```
    else: # la training accuracy supera el límite, ajusta el lr en base al validation loss
        monitor = 'val_loss'
        if epoch == 0:
            pimprov = 0.0

        else:
            pimprov = (self.lowest_vloss - v_loss ) * 100 / self.lowest_vloss

        if v_loss < self.lowest_vloss:
            self.lowest_vloss = v_loss
            self.best_weights = self.model.get_weights()
            self.count = 0
            self.stop_count = 0
            self.best_epoch = epoch + 1

        else:
            if self.count >= self.patience - 1:
                lr = lr * self.factor
                self.stop_count = self.stop_count + 1
                self.count = 0
                tf.keras.backend.set_value(self.model.optimizer.lr, lr)

            else:
                self.count = self.count + 1

            if acc > self.highest_tracc:
                self.highest_tracc = acc

    msg = f'{str(epoch + 1):^3s}/{str(self.epochs):4s} {loss:^9.3f}{acc * 100:^9.3f}{v_loss:^9.5f}{v_acc * 100:^9.3f}{current_lr:^9.5f}{lr:^9.5f}{monitor:^11s}{pimprov:^10.2f}{duration:^8.2f}'
    print(msg)

    if self.stop_count > self.stop_patience - 1: # chequea si el lr se ha ajustado stop_count veces sin mejora alguna.
        msg = f' El entrenamiento terminó en el epoch {epoch + 1} después de {self.stop_patience} ajustes al learning rate sin mejora alguna.'
        print(msg)
        self.model.stop_training = True

    else:
        if self.ask_epoch != None and self.ask_permission != 0:
            if epoch + 1 >= self.ask_epoch:
                self.model.stop_training = True
```

Creación e implementador del modelo de aprendizaje automático.

```
def CrearGeneradores (train_df, valid_df, test_df, tamanoLote, generadorImagenes):
    tamanoImagen = (224, 224)
    canales = 3
    color = 'rgb'
    shapeImagen = (tamanoImagen[0], tamanoImagen[1], canales)

    ts_length = len(test_df)
    test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n == 0 and ts_length/n <= 80]))
    test_steps = ts_length // test_batch_size

    tr_gen = generadorImagenes
    ts_gen = generadorImagenes

    #Conjunto de entrenamiento.
    train_gen = tr_gen.flow_from_dataframe( train_df, x_col= 'rutasArchivos', y_col= 'etiquetas', target_size= tamanoImagen, class_mode= 'categorical',
                                        color_mode= color, shuffle= False, batch_size= tamanoLote)

    #Conjunto de validación.
    valid_gen = ts_gen.flow_from_dataframe( valid_df, x_col= 'rutasArchivos', y_col= 'etiquetas', target_size= tamanoImagen, class_mode= 'categorical',
                                        color_mode= color, shuffle= False, batch_size= tamanoLote)

    #Conjunto de prueba.
    test_gen = ts_gen.flow_from_dataframe( test_df, x_col= 'rutasArchivos', y_col= 'etiquetas', target_size= tamanoImagen, class_mode= 'categorical',
                                        color_mode= color, shuffle= False, batch_size= test_batch_size)

    return train_gen, valid_gen, test_gen
```

Aseguramiento de la reproducibilidad

```
# Asegurar semillas fijas para la reproducibilidad
!pip install tensorflow-determinism
!pip install framework-reproducibility

semillaGlobal = 1

random.seed(semillaGlobal)
np.random.seed(semillaGlobal)
tf.random.set_seed(semillaGlobal)

tf.keras.utils.set_random_seed(semillaGlobal)
tf.config.experimental.enable_op_determinism()

os.environ['PYTHONHASHSEED']= str(semillaGlobal)
os.environ['TF_CUDNN_DETERMINISTIC']= str(semillaGlobal)
```

Todo lo asociado a la metaheurística

```
tamanoPoblacion         = 4
numeroIteraciones       = 2
limiteInferior          = 0.0
limiteSuperior          = 1.0
pacienciaMetaheuristica = 4

rutaResultadosMeta = "/kaggle/working/resultados/"

if not os.path.isdir("/kaggle/working/resultados/Logs"):
    os.makedirs("/kaggle/working/resultados/Logs")
```

```
dataGenPredeterminado = ImageDataGenerator(
  rotation_range        = 0,
  brightness_range      = [0,0],
  channel_shift_range   = 0,

  # Para estandarizar valores y asegurar bordes continuos en casos de rotaciones.
  rescale               = 1.0/255.0,
  fill_mode             = 'nearest',

  # Se ha evidenciado que los siguientes no ayudan al algoritmo, por lo tanto, se dejan por defecto.
  width_shift_range     = 0,
  height_shift_range    = 0,
  zoom_range            = 0,
  shear_range           = 0,
  horizontal_flip       = False,
  vertical_flip         = False,
)
batchPredeterminado     = 16
```

```
rangos = {
    "Rotación":         np.arange(   0,    10,      1),
    "Brillo Mínimo":    np.arange(0.50,  1.25,   0.05),
    "Brillo Máximo":    np.arange(0.75,  1.50,   0.05),
    "Cambio al Canal":  np.arange(   0,    50,      1),
    "Tamaño Lote":      [8, 16, 32, 64],
}
espacioSolucion = len(rangos.keys())
```

```
# Inicializar población.
np.random.seed(semillaGlobal)
poblacion = np.random.uniform(
  low=limiteInferior,
  high=limiteSuperior,
  size=(tamanoPoblacion, espacioSolucion)
)

np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})
print(poblacion.shape)
print(poblacion[0])
```

```python
def ActualizarPoblacion(poblacion, puntajes, numeroIteracion):
  mejorIndice = np.argmax(puntajes)
  mejorSolucion = poblacion[mejorIndice].copy()
  mejorPuntaje = puntajes[mejorIndice]

  np.random.seed(semillaGlobal)
  tf.random.set_seed(semillaGlobal)
  tf.keras.utils.set_random_seed(semillaGlobal)

  # MRFO:
  # Write the metaheuristic rules for updating the poblacion.
  nuevaPoblacion = poblacion.copy()

  coef = numeroIteracion / float(numeroIteraciones)
  for contador in range(len(poblacion)):
    r = np.random.random(1)
    alpha = 2.0 * r * np.sqrt(np.abs(np.log(r)))
    r1 = np.random.random(1)
    factor = (numeroIteraciones - numeroIteracion + 1.0) / (numeroIteraciones * 1.0)
    beta = 2.0 * np.exp(r1 * factor) * np.sin(2.0 * np.pi * r1)
    if (np.random.random(1) < 0.5):
      if (coef < np.random.random(1)):
        s = np.subtract(limiteSuperior, limiteInferior)
        u = np.random.uniform(low=0, high=1, size=espacioSolucion)
        m = np.multiply(u, s)
        xRand = np.clip(np.add(limiteInferior, m), limiteInferior, limiteSuperior)
        if (contador == 0):
          nuevaPoblacion[contador, :] = xRand + r * (xRand - poblacion[contador, :]) + beta * (xRand - poblacion[contador, :])
        else:
          nuevaPoblacion[contador, :] = xRand + r * (poblacion[contador - 1, :] - poblacion[contador, :]) + beta * (xRand - poblacion[contador, :])
      else:
        if (contador == 0):
          nuevaPoblacion[contador, :] = mejorSolucion + r * (mejorSolucion - poblacion[contador, :]) + beta * (mejorSolucion - poblacion[contador, :])
        else:
          nuevaPoblacion[contador, :] = mejorSolucion + r * (poblacion[contador - 1, :] - poblacion[contador, :]) + beta * (mejorSolucion - poblacion[contador, :])
    else:
      if (contador == 0):
        nuevaPoblacion[contador, :] = poblacion[contador, :] + r * (mejorSolucion - poblacion[contador, :]) + alpha * (mejorSolucion - poblacion[contador, :])
      else:
        nuevaPoblacion[contador, :] = poblacion[contador, :] + r * (poblacion[contador - 1, :] - poblacion[contador, :]) + alpha * (mejorSolucion - poblacion[contador, :])

    nuevaPoblacion[contador, :] = np.clip(nuevaPoblacion[contador, :], limiteInferior, limiteSuperior)

    if numeroIteracion==0:
        colorUsado = Fore.GREEN

    if numeroIteracion==1:
        colorUsado = Fore.BLUE

    print(colorUsado + "\nRecalculando el puntaje actual con la combinación: Poblacion índice",contador," de ",len(poblacion)-1," en su primera revisión.")
    print(colorUsado + "\nY con la siguiente nueva poblacion: ",nuevaPoblacion[contador, :])

    puntajeActual = FuncionFitness(nuevaPoblacion[contador, :])
    if (puntajeActual > mejorPuntaje):
      mejorSolucion, mejorPuntaje = nuevaPoblacion[contador, :].copy(), puntajeActual

    s = 2.0
    r2, r3 = np.random.random(1), np.random.random(1)
    nuevaPoblacion[contador, :] = poblacion[contador, :] + s * (r2 * mejorSolucion - r3 * poblacion[contador, :])

    nuevaPoblacion[contador, :] = np.clip(nuevaPoblacion[contador, :], limiteInferior, limiteSuperior)

    print(colorUsado + "\nRecalculando el puntaje actual con la combinación: Poblacion índice",contador," de ",len(poblacion)-1," en su segunda revisión.")
    print(colorUsado + "\nY con la siguiente nueva poblacion: ",nuevaPoblacion[contador, :])

    puntajeActual = FuncionFitness(nuevaPoblacion[contador, :])
    if (puntajeActual > mejorPuntaje):
      mejorSolucion, mejorPuntaje = nuevaPoblacion[contador, :].copy(), puntajeActual

  return nuevaPoblacion.copy()
```

```python
def FuncionFitness(solucion):
    solucion = np.round(solucion, 4)

    indice = int(np.round(solucion[0] * (len(rangos["Rotación"]) - 1)))
    valorRotacion = rangos["Rotación"][indice]

    indice = int(np.round(solucion[1] * (len(rangos["Brillo Mínimo"]) - 1)))
    valorBrilloMin = np.round(rangos["Brillo Mínimo"][indice],decimals=2)

    indice = int(np.round(solucion[2] * (len(rangos["Brillo Máximo"]) - 1)))
    valorBrilloMax = np.round(rangos["Brillo Máximo"][indice],decimals=2)

    if(valorBrilloMin>valorBrilloMax):
        traspaso       = valorBrilloMin
        valorBrilloMin = valorBrilloMax
        valorBrilloMax = traspaso

    indice = int(np.round(solucion[3] * (len(rangos["Cambio al Canal"]) - 1)))
    valorCanal = rangos["Cambio al Canal"][indice]

    indice = int(np.round(solucion[4] * (len(rangos["Tamaño Lote"]) - 1)))
    valorTamLote  = rangos["Tamaño Lote"][indice]
```

```
    dataGenInterior         = ImageDataGenerator(
        rotation_range      = valorRotacion,
        brightness_range    = [valorBrilloMin,valorBrilloMax],
        channel_shift_range = valorCanal,

        # Para estandarizar valores y asegurar bordes continuos en casos de rotaciones.
        rescale              = 1.0/255.0,
        fill_mode            = 'nearest',

        # Se ha evidenciado que los siguientes no ayudan al algoritmo, por lo tanto, se dejan por defecto.
        width_shift_range    = 0,
        height_shift_range   = 0,
        zoom_range           = 0,
        shear_range          = 0,
        horizontal_flip      = False,
        vertical_flip        = False,
    )

    generadorEntrena, generadorValida, generadorPrueba = CrearGeneradores(dfEntrenamiento, dfValidacion, dfPrueba, valorTamLote, dataGenInterior)
    tamanoImagen = (224, 224)
    canales = 3
    shapeImagen = (tamanoImagen[0], tamanoImagen[1], canales)

    palabraClave = "VGG19_metaheuristica-" + "-".join([str(el)[2:] for el in solucion])
    rutaCheckpoint = os.path.join(rutaResultadosMeta, "Checkpoints", palabraClave) + ".h5"
    rutaCSV = os.path.join(rutaResultadosMeta, "Logs", palabraClave) + ".csv"

    with strategy.scope():
        conteoClases = len(list(generadorEntrena.class_indices.keys()))
        modeloBase = tf.keras.applications.vgg19.VGG19(include_top= False, weights= "imagenet", input_shape= shapeImagen, pooling= 'max')
        model = Sequential([
            modeloBase,
            Dense(conteoClases, activation= 'softmax')
        ])
        model.compile(Adamax(learning_rate= 0.001), loss= 'categorical_crossentropy', metrics= ['accuracy'],)

    epochsModelo = 4
    callbacks = [CallbackCNN(model = model,
                 patience         = 4,
                 stop_patience    = 10,
                 threshold        = 0.9,
                 factor           = 10,
                 batches          = (int(np.ceil(len(generadorEntrena.labels) / valorTamLote))),
                 epochs           = epochsModelo,
                 ask_epoch        = 10),
                 ModelCheckpoint(rutaCheckpoint, save_best_only=True, save_weights_only=True, monitor="val_accuracy", mode="max", verbose=0),
                 TerminateOnNaN(),
                 CSVLogger(rutaCSV, append=True),
                 EarlyStopping(monitor="val_accuracy", mode="max", patience=pacienciaMetaheuristica),
                 ]
```

```
    tiempos = []
    np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})

    configs = [
        valorRotacion,
        valorBrilloMin,
        valorBrilloMax,
        valorCanal,
        valorTamLote
    ]
    print("\nEsta prueba se está realizando con la configuración: ", configs)

    history = model.fit(x                = generadorEntrena,
                        epochs           = epochsModelo,
                        verbose          = 0,
                        callbacks        = callbacks,
                        validation_data  = generadorValida,
                        validation_steps = None,
                        shuffle          = False)

    model.load_weights(rutaCheckpoint)
    longitudPrueba = len(dfPrueba)
    lotesPrueba = lotesPrueba = max(sorted([longitudPrueba // n for n in range(1, longitudPrueba + 1) if longitudPrueba%n == 0 and longitudPrueba/n <= 80]))
    pasosPrueba = longitudPrueba // lotesPrueba

    configs = [
        valorRotacion,
        valorBrilloMin,
        valorBrilloMax,
        valorCanal,
        valorTamLote
    ]

    generadorDiccionario = generadorPrueba.class_indices
    clases = list(generadorDiccionario.keys())
    predicciones = model.predict_generator(generadorPrueba)
    prediccionY = np.argmax(predicciones, axis=1)
    reporte = classification_report(generadorPrueba.classes, prediccionY, target_names= clases, output_dict=True)
    puntaje = reporte['accuracy']

    print("El puntaje obtenido por esta prueba fue: ",puntaje," con la configuración: ", configs)

    configsGlobal.append(configs)
    puntajesGlobal.append(puntaje)

    return puntaje

                EarlyStopping(monitor="val_accuracy", mode="max", patience=pacienciaMetaheuristica),
                ]
```

Interacción con el usuario y ejecución de la metaheurística.

```
print("................................................................")
print("Escoge el dataset a utilizar.")
print("C para el Completo   (21.165 muestras  -> 3.616, 6.012, 10.192, 1.345)")
print("R para el Reducido   (4000 muestras    -> 4.000, 4.000, 4.000, 4.000)")
print("M para el Minimizado (40 muestras      -> 10, 10, 10, 10)")
ansMet_1 = input('')
print("................................................................")
print("Escoge la semilla a utilizar (ej: 1, 1000, 2000, 2023, 5555...).\n")
semillaGlobal = input('')
semillaGlobal = int(semillaGlobal)
print("................................................................")
def comprobacionOpcion():
    if ansMet_1 == 'C' or ansMet_1 == 'c':
            data_dir         = '/kaggle/input/covid19-radiography-database/COVID-19_Radiography_Dataset'
            baseDatosNombre  = 'Completa (21.165 muestras -> 3.616, 6.012, 10.192, 1.345) SIN SEGMENTACIÓN (MASKING)'
            return data_dir, baseDatosNombre
    if ansMet_1 == 'R' or ansMet_1 == 'r':
            data_dir         = '/kaggle/input/covid19-reducido/COVID-19_Reducido'
            baseDatosNombre  = 'Reducida (4000 muestras -> 4.000, 4.000, 4.000, 4.000) SIN SEGMENTACIÓN (MASKING)'
            return data_dir, baseDatosNombre
    if ansMet_1 == 'M' or ansMet_1 == 'm':
            data_dir         = '/kaggle/input/covid19-minimo/COVID-19_Minimo'
            baseDatosNombre  = 'Minimizada (40 muestras -> 10, 10, 10, 10) SIN SEGMENTACIÓN (MASKING)'
            return data_dir, baseDatosNombre

data_dir, baseDatosNombre = comprobacionOpcion()
```

```
rutaDatabase        = data_dir

mejoresSoluciones   = []
mejoresPuntajes     = []
tiempos             = []
puntajesGlobal      = []
configsGlobal       = []
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})
numeroIteraciones = 2

dfEntrenamiento, dfValidacion, dfPrueba = RepartirDatos(rutaDatabase, 0.8, 0.5)

for numeroIteracion in range(numeroIteraciones):
    puntajes  = []
    configs = []
    print(Fore.BLACK + "------------------------------------------------------------------------")
    print(Fore.BLACK + "Nueva iteración (",numeroIteracion,"), la población, de len ",len(poblacion)," es:")
    print(poblacion)
    for contador in range(len(poblacion)):
        print(Fore.BLACK + "\nPasa por un len del poblacion (",contador,")")
        puntaje = FuncionFitness(poblacion[contador])
        puntajes.append(puntaje)

    nuevaPoblacion = ActualizarPoblacion(poblacion, puntajes, numeroIteracion)


    populationScoresPath = os.path.join(rutaResultadosMeta, "Population.csv")
    archivo = open(populationScoresPath, "a")
    for contador in range(len(poblacion)):
        datos = f"{numeroIteracion + 1},{contador + 1},"
        datos += ",".join([str(el) for el in poblacion[contador]])
        datos += f",{puntajes[contador]}"
        datos += "\n"
        archivo.write(datos)
    archivo.close()

    mejorIndice = np.argmax(puntajes)
    mejorSolucion = poblacion[mejorIndice].copy()
    mejorPuntaje = puntajes[mejorIndice]
    mejoresSoluciones.append(mejorSolucion)
    mejoresPuntajes.append(mejorPuntaje)

    poblacion = nuevaPoblacion.copy()
```

Muestreo de los valores finales obtenidos.

```
valoresFinales = []
for i in range(len(scoresGlobal)):
    valoresFinales.append([scoresGlobal[i], configsGlobal[i]])
valoresFinales.sort(key=lambda x: x[0], reverse=True)
valoresFinales
```

3. Selección de distintas poblaciones, puntajes y configuraciones.

```
.....................................................................................................
Para la prueba "GPU P100 - 32x32 pixeles de ancho y alto - Dataset Reducido", Semilla 1...:
.....................................................................................................
Población                                       Puntaje            Config. ocho hiperparámetros.
.....................................................................................................
[0.417 0.720 0.000 0.302 0.147 0.092 0.186 0.346] 0.3000 [18, 0.3, 0.0, 0.1, 0.1, True,  True, 16]
[0.397 0.539 0.419 0.685 0.204 0.878 0.027 0.670] 0.3475 [17, 0.2, 0.2, 0.3, 0.1, False, True, 32]
[0.417 0.559 0.140 0.198 0.801 0.968 0.313 0.692] 0.2800 [18, 0.2, 0.1, 0.1, 0.3, False, True, 32]
[0.876 0.895 0.085 0.039 0.170 0.878 0.098 0.421] 0.2675 [39, 0.4, 0.0, 0.0, 0.1, False, True, 16]
[1.000 1.000 0.000 0.113 0.000 0.000 0.000 0.000] 0.4000 [44, 0.4, 0.0, 0.0, 0.0, True,  True,  8]
[0.362 0.493 0.000 0.151 0.063 0.040 0.080 0.149] 0.3500 [16, 0.2, 0.0, 0.1, 0.0, True,  True,  8]
[0.927 1.000 0.000 0.000 0.000 0.000 0.102 0.000] 0.3450 [41, 0.4, 0.0, 0.0, 0.0, True,  True,  8]
.....................................................................................................
Para la prueba "GPU P100 - 244x244 pixeles de ancho y alto - Dataset Reducido", Semilla 1...:
.....................................................................................................
Población                                       Puntaje            Config. ocho hiperparámetros.
.....................................................................................................
[0.335 0.524 0.134 0.354 0.180 0.366 0.113 0.401] 0.4975 [15, 0.2, 0.1, 0.1, 0.1, True,  True, 16]
[1.000 1.000 0.067 0.031 0.134 1.000 0.077 1.000] 0.6700 [44, 0.4, 0.0, 0.0, 0.1, False, True, 64]
[0.000 0.000 0.101 0.844 0.014 0.000 0.106 0.000] 0.2500 [ 0, 0.0, 0.0, 0.3, 0.0, True,  True,  8]
[0.222 0.281 0.037 0.101 0.055 0.202 0.042 0.229] 0.4725 [10, 0.1, 0.0, 0.0, 0.0, True,  True, 16]
[0.725 0.823 0.100 0.197 0.143 0.714 0.095 0.747] 0.3150 [32, 0.3, 0.0, 0.1, 0.1, False, True, 32]
[1.000 1.000 0.287 0.592 0.365 1.000 0.239 1.000] 0.7000 [44, 0.4, 0.1, 0.2, 0.1, False, True, 64]
[1.000 1.000 0.316 0.697 0.359 1.000 0.245 1.000] 0.5250 [44, 0.4, 0.1, 0.3, 0.1, False, True, 64]
.....................................................................................................
Para la prueba "GPU P100 - 244x244 pixeles de ancho y alto - Dataset Completo", Semilla 1...:
.....................................................................................................
Población                                       Puntaje            Config. ocho hiperparámetros.
.....................................................................................................
[0.417 0.720 0.000 0.302 0.147 0.092 0.186 0.346] 0.4969 [18, 0.3, 0.0, 0.1, 0.1, True,  True, 16]
[0.397 0.539 0.419 0.685 0.204 0.878 0.027 0.670] 0.3112 [17, 0.2, 0.2, 0.3, 0.1, False, True, 32]
[0.417 0.559 0.140 0.198 0.801 0.968 0.313 0.692] 0.3623 [18, 0.2, 0.1, 0.1, 0.3, False, True, 32]
[0.876 0.895 0.085 0.039 0.170 0.878 0.098 0.421] 0.3415 [39, 0.4, 0.0, 0.0, 0.1, False, True, 16]
[1.000 1.000 0.000 0.113 0.000 0.000 0.000 0.000] 0.4813 [44, 0.4, 0.0, 0.0, 0.0, True,  True,  8]
[1.000 1.000 0.000 0.753 0.365 0.230 0.464 0.860] 0.5810 [44, 0.4, 0.0, 0.3, 0.1, True,  True, 64]
[0.818 1.000 0.000 0.401 0.262 0.000 0.456 0.518] 0.5106 [36, 0.4, 0.0, 0.2, 0.1, True,  True, 32]
.....................................................................................................
Para la prueba "TPU VM v3-8 - 244x244 pixeles de ancho y alto - Dataset Reducido", Semilla 1...:
.....................................................................................................
Población                                       Puntaje            Config. cinco hiperparámetros.
.....................................................................................................
[0.417 0.720 0.000 0.302 0.147]                  0.8700               [4, 0.75, 1.00, 15,  8]
[0.092 0.186 0.346 0.397 0.539]                  0.9100               [1, 0.65, 1.00, 19, 32]
[0.419 0.685 0.204 0.878 0.027]                  0.9125               [4, 0.90, 1.00, 43,  8]
[0.670 0.417 0.559 0.140 0.198]                  0.9025               [6, 0.80, 1.15,  7, 16]
[1.000 1.000 0.000 0.113 0.000]                  0.9000               [9, 0.75, 1.20,  6,  8]
[0.537 0.865 0.329 1.000 0.000]                  0.8975               [5, 1.00, 1.10, 49,  8]
[0.000 0.000 0.000 1.000 0.000]                  0.9100               [0, 0.50, 0.75, 49,  8]
.....................................................................................................
Para la prueba "TPU VM v3-8 - 244x244 pixeles de ancho y alto - Dataset Completo", Semilla 1...:
.....................................................................................................
Población                                       Puntaje            Config. cinco hiperparámetros.
.....................................................................................................
[0.417 0.720 0.000 0.302 0.147]                  0.8899               [4, 0.75, 1.00, 15,  8]
[0.092 0.186 0.346 0.397 0.539]                  0.8894               [1, 0.65, 1.00, 19, 32]
[0.419 0.685 0.204 0.878 0.027]                  0.8885               [4, 0.90, 1.00, 43,  8]
[0.670 0.417 0.559 0.140 0.198]                  0.8998               [6, 0.80, 1.15,  7, 16]
[1.000 1.000 0.000 0.113 0.000]                  0.8823               [9, 0.75, 1.20,  6,  8]
[0.235 0.166 0.181 0.058 0.070]                  0.9041               [2, 0.60, 0.90,  3,  8]
[0.984 1.000 1.000 0.000 0.772]                  0.8512               [9, 1.20, 1.45,  0, 32]
.....................................................................................................
```