Automatización de la detección y diagnóstico de leishmaniasis por medio de la identificación de parásitos en imágenes de placas de laboratorio

Daniel Fernando Cardozo Aricapa

Nota de Aceptación

Certificamos que el presente Trabajo de Grado Satisface, en alcances y calidad, todos los requisitos que demanda un Trabajo de Grado de Maestría.

GLORIA INÉS ÁLVAREZ VARGAS
Director

JULIÁN GIL GONZALEZ
Jurado

OMAR ANDRES CASTAÑO IDÁRRAGA
Jurado

Aprobado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana Cali, para optar el título de Magister en Ciencia de Datos..

HERNÁN CAMILO ROCHA NIÑO Ph. D.
Decano Facultad de Ingeniería y Ciencias

JUAN CARLOS MARTÍNEZ ARIAS
Director Posgrados de Ingeniería y Ciencias

Acta de Correcciones al Documento de Trabajo de Grado

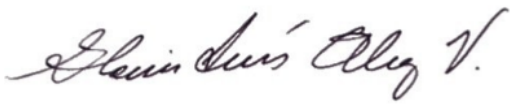
Santiago de Cali, 27 marzo de 2024

Autor: Daniel Fernando Cardozo Aricapa

Título del Trabajo de Grado: Automatización de la detección y diagnóstico de leishmaniasis por medio de la identificación de parásitos en imágenes de placas de laboratorio.

Director: Gloria Inés Álvarez Vargas

Como indica el artículo 2.13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.



Firma del director del Trabajo de Grado

Santiago de Cali, 15 de enero del 2024

Doctora
Gloría Inés Alvarez V.
Directora Maestría en Ciencia de Datos
Facultad de Ingeniería y Ciencias
Pontificia Universidad Javeriana de Cali

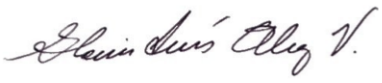
Asunto: Presentación para evaluación del proyecto aplicado

Cordial Saludo,

Con el fin de cumplir con los requisitos exigidos por la Universidad para optar por el título de Magíster en Ciencia de Datos, nos permitimos presentar a su consideración el proyecto denominado "**Automatización de la detección y diagnóstico de leishmaniasis por medio de la identificación de parásitos en imágenes de placas de laboratorio**", el cual fue realizado por el (los) estudiante (s) Daniel Fernando Cardozo Aricapa con código (s) 8974856 pertenecientes a la Maestría en Ciencia de Datos, bajo la dirección de Gloria Inés Álvarez V.

El suscrito director del Proyecto Aplicado autoriza para que se proceda a hacer la evaluación de este proyecto, toda vez que ha revisado cuidadosamente el documento y avala que ya se encuentra listo para ser presentado y sustentado oficialmente.

Atentamente,



Daniel Fernando Cardozo Aricapa
C.C. **94.064.615** de **Cali**

Gloria Inés Álvarez Vargas
C.C. **30.306.105** de **Manizales**

Documentación anexa:
Resumen del Proyecto Aplicado en formato digital (máximo 1 página).
Una copia digital (PDF) del documento del proyecto aplicado

FICHA RESUMEN

Título: Automatización de la detección y diagnóstico de leishmaniasis por medio de la identificación de parásitos en imágenes de placas de laboratorio.

1. **Área de Trabajo**: Salud Publica
2. **Tipo de Proyecto (Aplicado, Innovación, Investigación)**: El tipo de proyecto se clasifica como innovación, debido a que se busca mejorar el proceso de diagnóstico de la enfermedad.
3. **Estudiante(S):** Daniel Fernando Cardozo Aricapa
4. **Correo Electrónico**: dacari@javerianacali.edu.co
5. **Dirección y Teléfono**: Cra 13ª 1-139 Oeste
6. **Director**: Gloria Inés Álvarez
7. **Vinculación del director**: Directora de la Maestría en Ciencia de Datos Pontificia Universidad Javeriana Cali
8. **Correo Electrónico del director**: galvarez@javerianacali.edu.co
9. **Codirector (Si Aplica):** N/A
10. **Grupo o Empresa que lo Avala (Si Aplica):** Centro Internacional de Entrenamiento e Investigaciones médicas CIDEIM.
11. **Otros Grupos o Empresas**: N/A
12. **Palabras Clave (Al Menos 5):** Leishmaniasis; Machine Learning, Deep Learning, Diagnosis, Parasites, Disease Vectors, Diagnostic Imaging
13. **Fecha de Inicio**: noviembre 2022
14. **Duración Estimada (En Meses):** 12 meses
15. **Resumen**:

La leishmaniasis es una enfermedad causada por más de 20 especies del género *Leishmania* un protozoo parasito. Esta enfermedad se transmite por la picadura de flebótomos hembra infectados, que necesitan ingerir sangre para producir huevos. A nivel mundial, se encuentra entre las diez enfermedades tropicales desatendidas con más de 12 millones de personas infectadas con 0,9 a 1,6 millones de nuevos casos al año y entre 20.000 a 30.000 defunciones. En la actualidad, las estrategias de prevención y control disponibles para el manejo de la leishmaniasis son limitadas, por lo cual se requiere de herramientas efectivas para el diagnóstico temprano y tratamiento adecuado. Es por esto por lo que nuestro objetivo es desarrollar un modelo automatizado capaz de realizar la identificación del parasito y diagnóstico de Leishmaniasis usando imágenes de placas de laboratorio en pacientes con sospecha clínica de la enfermedad. Para estos proponemos utilizar diferentes algoritmos de clasificación que nos permitan realizar la detección de parásitos de Leishmania por medio de la extracción de características, creación de imágenes integrales y clasificación. Como resultados esperados se espera contar con un modelo diagnostico adecuado basado en placas de laboratorio que permita realizar el diagnostico de forma oportuna y accesible capaz de funcionar de forma eficiente en cualquier área que lo requiera. Finalmente, esta tecnología será una herramienta fundamental para la salud publica en áreas endémicas en pro de disminuir la morbimortalidad de la enfermedad.

Automatización de la detección y diagnóstico de leishmaniasis por medio de la identificación de parásitos en imágenes de placas de laboratorio.

Daniel Fernando Cardozo Aricapa
Código 8974856

Proyecto Aplicado para optar al título de
Magister en Ciencia de Datos

Director(a)
Gloria Inés Álvarez

Codirector(a)
Diego Luis Linares

Facultad de Ingeniería Y Ciencias
Maestría en Ciencia de Datos
Santiago de Cali, enero 15 de 2023

Tabla de Contenido

1 Introducción 9
2 Definición del Problema 10
2.1. Planteamiento del Problema 10
2.2. Formulación del Problema 12
3 Objetivos del Proyecto 12
3.1. Objetivo General 12
3.2. Objetivos Específicos 12
3.3. Resultados Esperados 12
4 Marco de Referencia 13
4.1. Marco Teórico 13
4.1.1. Aprendizaje automático o Machine Learning (ML) 14
4.1.2. Aprendizaje profundo o Deep Learning (DL) 16
4.1.3. Redes neuronales 17
4.1.3.1. Neurona biológica 17
4.1.3.2. Neurona artificial 19
4.1.3.3. Perceptrón 20
4.1.3.4. Red neuronal y redes multicapa 20
4.1.4. Redes convolucionales 22
4.1.5. Enriquecimiento de datos (Data Augmentation) 24
4.1.6. Aprendizaje por transferencia (Transfer Learning) 25
4.1.7. Métricas de evaluación de modelos 25
4.2. Antecedentes 27
4.2.1. Estudios de diagnóstico automático de leishmaniasis 28
5 Metodología 29
5.1. Preprocesamiento de los datos 29
5.1.1. Limpieza y adecuación de imágenes 30
5.1.2. Construcción de datasets 33
5.1.2.1. Dataset sin aumento de datos 34
5.1.2.2. Dataset con aumento de datos (AD) 35
5.2. Construcción de modelos 36
5.2.1. Modelo Arquitectura CNN (Construcción Propia) 36
5.2.2. Modelo Arquitectura VGG16 37
5.2.3. Modelo Arquitectura VGG19 38
5.2.4. Modelo Arquitectura Mobilenetv2 39
5.3. Búsqueda de Hiperparametros de modelos VGG16, VGG19 y MobileNetV2 (Transfer Learning) 41
5.4. Optimización de modelos 41
6 Resultados 41
6.1. Modelo convolucional construcción propia 42
6.1.1. Búsqueda de parámetros e hiperparametros 42

6.1.2. Construcción del modelo 42
6.1.3. Evaluación del modelo 43
6.1.4. Evaluación del modelo con aumento de datos 44
6.2. Modelo arquitectura VGG16 46
6.2.1. Búsqueda de parámetros e hiperparametros 46
6.2.2. Construcción del modelo 46
6.2.3. Evaluación del modelo 47
6.2.4. Evaluación del modelo con aumento de datos 49
6.3. Modelo arquitectura VGG19 50
6.3.1. Búsqueda de parámetros e hiperparametros 50
6.3.2. Construcción del modelo 51
6.3.3. Evaluación del modelo 51
6.3.4. Evaluación del modelo con aumento de datos 53
6.4. Modelo arquitectura MobilenetV2 54
6.4.1. Búsqueda de parámetros e hiperparametros 54
6.4.2. Construcción del modelo 55
6.4.3. Evaluación del modelo 55
6.4.4. Evaluación del modelo con aumento de datos 57
7 Conclusiones y Trabajos Futuros 59
7.1. Conclusiones 59
7.2. Trabajos Futuros 60
8 Referencias Bibliográficas 61

Listado de Graficas

Gráfico 1 Jerarquía de la IA [6] 14
Gráfico 2 Aplicaciones del aprendizaje automático [6] 15
Gráfico 3 Tipos de algoritmos de aprendizaje automático [11] 16
Gráfico 4 Aplicaciones de aprendizaje profundo [6] 17
Gráfico 5 Neurona biológica [14] 18
Gráfico 6 Potencial de acción de una neurona biológica[13] 19
Gráfico 7 Neurona artificial [12] 19
Gráfico 8 Perceptrón con N entradas 20
Gráfico 9 Capa Convolucional [16] 22
Gráfico 10 Capa convolucional adicional 23
Gráfico 11 Métrica Matriz de confusión 26
Gráfico 12Métrica Accuracy o exactitud 26
Gráfico 13 Métrica Recall o Sensibilidad 26
Gráfico 14Métrica Especificidad 26
Gráfico 15Métrica Precisión 26
Gráfico 16Métrica F1-Score 27
Gráfico 17Métrica Valor Predictivo Positivo (VPP) 27
Gráfico 18Métrica Valor Predictivo Negativo 27
Gráfico 19 Eliminación de guía ocular para microscopio (puntero indicador) y reducción de background (fondo) en imágenes de placas sin parasitos 30
Gráfico 20 Eliminación de background (fondo) en imágenes de placas con parasitos 31
Gráfico 21 Eliminación de background (fondo) en imágenes de placas sin parasitos 32
Gráfico 22 Dataset Training 34
Gráfico 23 Dataset Validation 34
Gráfico 24 Dataset Training + validation 34
Gráfico 25 Dataset Test 35
Gráfico 26 Dataset Training AD 35
Gráfico 27 Dataset Training + validation AD 36
Gráfico 28 Modelo base propio 36
Gráfico 29 Arquitectura VGG16 38
Gráfico 30 Arquitectura VGG19 39
Gráfico 31 Diagrama arquitectura Mobilenetv2 [25] 40
Gráfico 32Arquitectura Mobilenet_v2 [26] 40
Gráfico 41 Construcción del Modelo CNN (propio) 43
Gráfico 42 Accuracy (Exactitud) y función de perdida (loss function) Modelo CNN (propio) 43
Gráfico 43Matriz de confusión y Curva ROC (AUC) Modelo CNN (propio) 44
Gráfico 44 Accuracy (Exactitud) y función de perdida (loss function) Modelo CNN Aumento de Datos (propio) 45
Gráfico 45 Matriz de confusión y Curva ROC (AUC) Modelo CNN Aumento de Datos (propio) 45
Gráfico 46 Construcción del Modelo VGG16 47

Gráfico 47 Accuracy (Exactitud) y función de perdida (loss function) Modelo VGG16 47
Gráfico 48Matriz de confusión y Curva ROC (AUC) Modelo VGG16 48
Gráfico 49 Accuracy (Exactitud) y función de perdida (loss function) Modelo VGG16 Aumento de Datos 49
Gráfico 50Matriz de confusión y Curva ROC (AUC) Modelo VGG16 Aumento de Datos 50
Gráfico 51 Construcción del Modelo VGG19 51
Gráfico 52 Accuracy (Exactitud) y función de perdida (loss function) Modelo VGG19 52
Gráfico 53Matriz de confusión y Curva ROC (AUC) Modelo VGG19 52
Gráfico 54 Accuracy (Exactitud) y función de perdida (loss function) Modelo VGG19 Aumento de Datos 53
Gráfico 55Matriz de confusión y Curva ROC (AUC) Modelo VGG19 Aumento de Datos.....54
Gráfico 56 Construcción del Modelo Mobilenet_v2 55
Gráfico 57 Accuracy (Exactitud) y función de perdida (loss function) Modelo Mobilenet_v2 .55
Gráfico 58Matriz de confusión y Curva ROC (AUC) Modelo Mobilenet_v2.....56
Gráfico 59 Accuracy (Exactitud) y función de perdida (loss function) Modelo Mobilenet_v2 Aumento de Datos 57
Gráfico 60Matriz de confusión y Curva ROC (AUC) Modelo Mobilenet_v2 Aumento de Datos 58

Listado de Tablas

Tabla 1 Tipos de aprendizaje automático 15
Tabla 2 Distribución set de imagenes de Leishmaniasis 29
Tabla 3Estrategia de búsqueda de parámetros e hiperparametros Modelo Propio 37
Tabla 4 Estrategia de búsqueda de parámetros e hiperparametros Modelos Transfer Learning 41
Tabla 5 Parámetros de entrenamiento modelos optimizados 41
Tabla 6 Búsqueda de parámetros e hiperparametros Modelo CNN (propio) 42
Tabla 7 Métricas evaluación rendimiento Modelo CNN (propio) 44
Tabla 8 Métricas evaluación rendimiento Modelo CNN Aumento de Datos (propio) 46
Tabla 9 Búsqueda de parámetros e hiperparametros Modelo VGG16 46
Tabla 10 Métricas evaluación rendimiento Modelo VGG16 48
Tabla 11 Métricas evaluación rendimiento Modelo VGG16 Aumento de Datos 50
Tabla 12 Búsqueda de parámetros e hiperparametros Modelo VGG19 50
Tabla 13 Métricas evaluación rendimiento Modelo VGG19 53
Tabla 14 Métricas evaluación rendimiento Modelo VGG19 Aumento de Datos 54
Tabla 15 Búsqueda de parámetros e hiperparametros Modelo Mobilenet_v2 54
Tabla 16 Métricas evaluación rendimiento Modelo Mobilenet_v2 56
Tabla 17 Métricas evaluación rendimiento Modelo Mobilenet_v2 Aumento de Datos 58

1 Introducción

La leishmaniasis es una enfermedad causada por más de 20 especies del género *Leishmania* un protozoo parasito. Esta enfermedad se transmite por la picadura de flebótomos hembra infectados, que necesitan ingerir sangre para producir huevos. A nivel mundial, se encuentra entre las diez enfermedades tropicales desatendidas con más de 12 millones de personas infectadas con 0,9 a 1,6 millones de nuevos casos al año y entre 20.000 a 30.000 defunciones. En la actualidad, las estrategias de prevención y control disponibles para el manejo de la leishmaniasis son limitadas, por lo cual se requiere de herramientas efectivas para el diagnóstico temprano y tratamiento adecuado. Es por esto por lo que nuestro objetivo es desarrollar un modelo automatizado capaz de realizar la identificación del parasito y diagnóstico de Leishmaniasis usando imágenes de placas de laboratorio en pacientes con sospecha clínica de la enfermedad. Para estos proponemos utilizar diferentes algoritmos de clasificación que nos permitan realizar la detección de parásitos de Leishmania por medio de la extracción de características, creación de imágenes integrales y clasificación. Como resultados esperados se espera contar con un modelo diagnostico adecuado basado en placas de laboratorio que permita realizar el diagnostico de forma oportuna y accesible capaz de funcionar de forma eficiente en cualquier área que lo requiera. Finalmente, esta tecnología será una herramienta fundamental para la salud publica en áreas endémicas en pro de disminuir la morbimortalidad de la enfermedad.

2 Definición del Problema

2.1. Planteamiento del Problema

Las enfermedades transmitidas por vectores son enfermedades humanas provocadas por bacterias, parásitos o virus, las cuales son causadas por transmisión vectorial. Los vectores son organismos vivos que pueden transmitir patógenos infecciosos entre personas, o de animales a personas. Siendo la mayoría de los vectores insectos hematófagos que ingieren microorganismos patógenos junto con la sangre de un portador infectado (al picar a personas o animales) y posteriormente los transmiten a un nuevo portador, una vez replicado el patógeno. Estas enfermedades producen más de 300 millones de casos en todo el mundo anualmente y registran más de 700.000 muertes asociadas a enfermedades como la malaria, el dengue, la esquistosomiasis, tripanosomiasis africana, enfermedad de Chagas, fiebre amarilla, encefalitis japonesa, oncocercosis y leishmaniasis [1].

La leishmaniasis son enfermedades de transmisión vectorial la cual es causada por más de 20 especies del género *Leishmania* un protozoo parasito con más de 90 especies de flebótomos transmisores de este parasito. Su reservorio principal son mamíferos silvestres como perezosos, oso hormiguero, zarigüeyas, ratas silvestres, puerco espín y los perros. Mientras se han descrito más de 135 especies del género Lutzomyia como vectores transmisores de *Leishmania*[2], [3], [4].

Existen tres formas de presentación de leishmaniasis, entre la que se encuentra la visceral la cual es mortal en más del 95% de los casos y se caracteriza por episodios irregulares de fiebre, pérdida de peso, hepatoesplenomegalia y anemia. La leishmaniasis cutánea es la forma más frecuente y produce lesiones cutáneas, sobre todo ulcerosas que dejan cicatrices de por vida, además de discapacidad grave. Por su parte, la mucocutánea o mucosa, conduce a destrucción parcial o total de membranas mucosas de la nariz, boca y garganta[3].

El principal mecanismo de transmisión de la *leishmaniasis* es la picadura de un vector flebótomos hembra infectados con la forma de promastigote del parasito, que necesitan ingerir sangre para producir huevos. Estas picaduras generalmente ocurren en sitios expuestos y dejan pequeñas pápulas rojas en la piel[2], [3]. Siendo los principales factores de riesgo para la enfermedad, las condiciones socioeconómicas, dado que la pobreza aumenta el riesgo de leishmaniasis asociados a las malas condiciones de las viviendas, deficiencia de saneamiento de los hogares, hacinamiento y procesos de comportamiento humano. Asimismo, la malnutrición aumenta el riesgo de progresión de la enfermedad debido a dietas bajas proteínas, hierro, vitamina A y zinc. La movilidad poblacional favorece las formas cutáneas y viscerales, así como la exposición en el trabajo y el aumento de la deforestación. Finalmente, factores ambientales como la urbanización y la incursión del ser humano en zonas boscosas, y el cambio climático que afecta la temperatura, las precipitaciones, la humedad y

temporadas de sequias afectan considerablemente los ciclos de reproducción y supervivencia de vectores, reservorios, además de afectar el desarrollo de promastigotes de *Lesihmania* en los flebótomos, su tasa de supervivencia, el tamaño de la población, además de favorecer su transmisión[3] [2].

A nivel mundial, la leishmaniasis se encuentra entre las diez enfermedades tropicales desatendidas con más de 12 millones de infectados con 0,9 a 1,6 millones de nuevos casos al año. Un total de 20.000 a 30.000 defunciones y 350 millones de personas en riesgo de infectarse[3], [4]. Asimismo, es endémica en más de 102 países y territorios, reportándose transmisión en los 5 contienes. No obstante, la carga de la enfermedad se concentra en la población de bajos ingresos de la región africana, Asiática y de América Latina[2]. Entre el 70% al 75% de los casos mundiales de leishmaniasis cutánea se presentan en Afganistán, Argelia, Brasil, Colombia, Etiopia, Irán, Nicaragua, Perú, Sudan y Siria. Mientras la leishmaniasis mucosa ocurre especialmente en las Américas en los países de Bolivia, Brasil y Perú. Finalmente, más del 90% de los casos de leishmaniasis visceral ocurren en Bangladesh, Brasil, Etiopia, India, Sudan del Sur y Sudan[2], [3], [4].

La situación epidemiologia en las Américas, muestra que para el año 2019 Brasil presento el mayor número de casos, correspondiendo principalmente a la forma cutánea y mucosa con aproximadamente 15.000 casos, seguido por Colombia con 6.000 casos, Perú con 5.400 casos, Nicaragua con 3.300 casos y Bolivia con 2.000 acumulando el 77% de casos incidentes de la región. En contraste, para el año 2019 la leishmaniasis visceral presento una disminución en el número de casos en Brasil, Colombia, Guatemala, Hondura y Venezuela, así como un incremento de casos en Argentina, Paraguay, Uruguay y Bolivia [2], [4].

En Colombia, para la década de los 90 se notificaban en promedio 6.500 casos por año. No obstante, en los años 2005, 2006, 2009, 2010, 2014 y 2016 se presentaron picos de casos incidentes, pasando a más de 12 casos por año. Mostrando para el año 2020 un total de 6.176 casos incidentes para una incidencia de 23,2 casos por 100.000 habitantes[2], [5]. En general, la leishmaniasis es endémica en el territorio nacional a excepción de San Andrés Islas, Atlántico y Bogotá D.C. Situando que pone en riesgo a más de 11 millones de personas, especialmente las ubicadas en la zona rural[2]. Así mismo, la presentación más frecuente es la leishmaniasis cutánea la cual concentra entre el 95% al 98% de los casos, mientras la presentación mucosa y visceral varían entre el 1 a 4% y el 0,1 al 1,5%, respectivamente. Siendo los departamentos con mayor número de casos Guaviare, Vaupés, Meta y Caldas[2].

En general, el tratamiento de la leishmaniasis depende de diferentes factores, como el tipo de presentación de la enfermedad, la presencia de comorbilidades en el paciente (especialmente el VIH), la especie del parasito y la ubicación geográfica[3], [4]. Mientras su diagnóstico se realiza mediante examen clínico y pruebas de laboratorio (parasitológicas y serológicas). Las pruebas serológicas presentan baja sensibilidad y especificidad para el diagnóstico de

leishmaniasis cutánea y mucocutánea, por lo cual, el *Gold Standard* para el diagnóstico de leishmaniasis es la identificación histopatológica del parasito a través de muestras tomadas por frotis o biopsia para su posterior identificación en el microscopio, lo que corrobora las manifestaciones clínicas[3]. Por otra parte, la identificación oportuna del parasito es esencial para establecer un tratamiento específico y limitar el progreso de la enfermedad, aliviar los signos y síntomas, y mejorar la calidad de vida de los pacientes. Que en caso de no ser tratados a tiempo podría sufrir una deformidad o desfiguración en las presentaciones cutáneas o mucosas o la muerte en los pacientes con presentación visceral[4]. Es por esta razón, que planteamos este trabajo de investigación en donde buscamos crear una herramienta que permita realizar el diagnóstico de la enfermedad usando el aprendizaje automático (Machine Learning), por medio del entrenamiento de un modelo basado en la identificación del parásito en placas de laboratorio de pacientes con la enfermedad, garantizando así contar con un sistema de diagnóstico oportuno, correcto y de bajo costo el cual permita el inicio del tratamiento de forma precoz.

2.2. Formulación del Problema

Basados en la información anterior, planteamos resolver la siguiente pregunta de investigación. ¿es posible desarrollar un modelo basado en machine Learning que se capaz de realiza la identificación del parasito para confirmar el diagnostico de leishmaniasis usando imágenes de placas de laboratorio?

3 Objetivos del Proyecto

3.1. Objetivo General

- Desarrollar un modelo automatizado capaz de realizar la identificación del parasito y diagnóstico de Leishmaniasis usando imágenes de placas de laboratorio en pacientes con sospecha clínica de la enfermedad.

3.2. Objetivos Específicos

- Preprocesar la información contenida en la base de datos con la información de los pacientes.
- Construir y entrenar el modelo de Machine Learning para realizar el diagnostico de Leishmaniasis basado en imágenes de placas de laboratorio.
- Validar los resultados entregados por el modelo de Machine Learning construido.
- Evaluar la sensibilidad, especificidad, valor predictivo, curva ROC y la verosimilitud del modelo diagnóstico de Leishmaniasis.

3.3. Resultados Esperados

Dentro de los principales resultados esperados de nuestro proyecto aplicado, se encuentra el desarrollo de un modelo diagnostico basado en metodologías de ciencia de

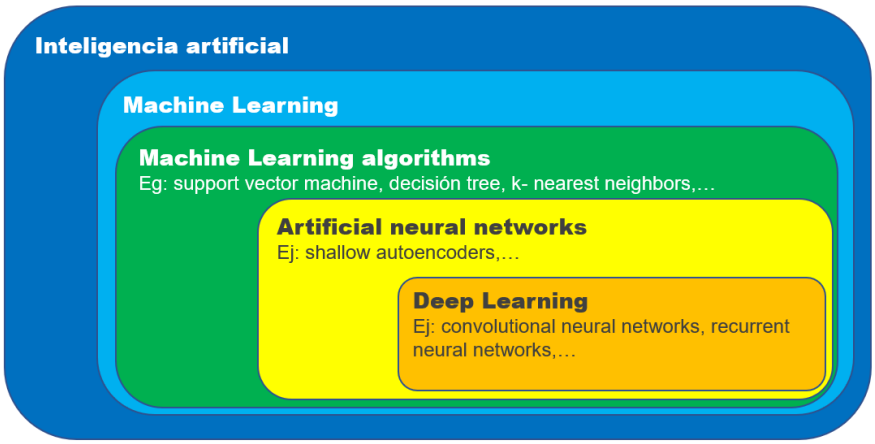
datos, el cual aprenderá a identificar el parasito de la leishmaniasis de forma adecuada en placas de laboratorio, para así garantizar un oportuno y adecuado diagnóstico de las personas con sospecha de leishmaniasis.
Esta herramienta facilitará el diagnóstico de la enfermedad y podrá ser implementada de forma masiva en sitios donde la enfermedad es más prevalente. Así mismo, facilitara su diagnóstico en áreas geográficas dispersas donde no se cuente con red hospitalaria y de laboratorio suficiente para realizar el diagnostico de forma oportuna. Por otra parte, el uso de esta herramienta permitirá realizar un diagnóstico temprano y oportuno de la enfermedad, disminuyendo así la ocurrencia de falsos positivos, así como también acortando los tiempos de inicio de tratamiento, lo cual se traduce en un mejor pronóstico de la enfermedad y una mayor tasa de recuperación.
En términos generales, el desarrollo de este modelo diagnostico por medio de aprendizaje automático de leishmaniasis, será una gran herramienta que ayudará a la salud pública de áreas endémicas a disminuir morbilidad y mortalidad causada por esta enfermedad.

4 Marco de Referencia

4.1. Marco Teórico

El termino Inteligencia Artificial (IA) hace referencia a que las maquinas sean tan inteligentes como un cerebro humano. Informalmente, el termino IA se asocia a que una maquina pueda realizar funciones que los humanos asocian con otras mentes humanas, como *resolver problemas* o *aprender*[6]. En términos generales, la IA comprende cualquier técnica que permita a las computadoras imitar el comportamiento humano y reproducir o superar la toma de decisiones humanas para resolver tareas complejas de forma independiente o con una intervención humana mínima[7].
Los primeros enfoques de IA se centraron en codificar acciones utilizando lenguajes formales que eran representados por medio de un algoritmo, sobre las cuales la computadora puede tomar decisiones en funciones de las reglas de inferencia lógica descritas. No obstante, este enfoque presenta varias limitaciones, en especial las asociadas a las limitaciones humanas para explicar el conocimiento tácito que se requiere para realizar tareas complejas. En contraste, el aprendizaje automático supera estas limitaciones, al tener la capacidad de aprender de los datos de entrenamiento específicos del problema al automatizando el proceso de creación de modelos analíticos y así resolver las tareas asociadas[7]. Por tanto, el aprendizaje automático es un subcampo de la IA[6].
En la actualidad, se cuentan con enormes avances en el aprendizaje automático y producto de estos se ha desarrollado el aprendizaje profundo el cual se basa en redes neuronales artificiales y es considerado como un subconjunto del aprendizaje automático[6], [7] .(Ver Gráfico 1)

Gráfico 1 Jerarquía de la IA [6]



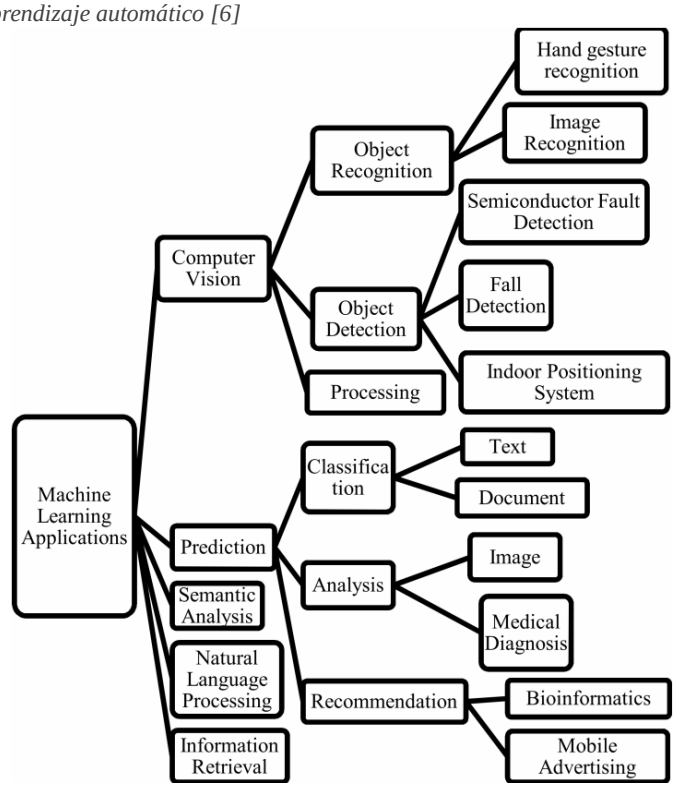
4.1.1. Aprendizaje automático o Machine Learning (ML)

El aprendizaje automático tiene como objetivo automatizar la tarea de construcción de modelos analíticos para realizar tareas cognitivas como la detección de objetos o la traducción del lenguaje natural. Esto se logra mediante la aplicación de algoritmos que aprende iterativamente de los datos de entrenamiento específicos del problema, lo que permite a las computadoras encontrar información oculta y patrones complejos sin ser programados explícitamente como datos de alta dimensión, clasificación, regresión y agrupación. Lo que ayuda a producir decisiones confiables y repetibles. Por esta razón, los algoritmos de ML se aplican en áreas para la detección de fraudes, calificación crediticia, análisis de la siguiente mejor oferta, reconocimiento de voz e imágenes o el procesamiento del lenguaje natural (NLP)[7].

En general, las aplicaciones de aprendizaje automático se pueden resumir en (Ver Imagen 2):

- Visión por computadora: Permite el reconocimiento de objetos, detección de objetos y el procesamiento de objetos.
- Predicción: Este tipo de análisis se divide en clasificación (texto y documentos), análisis (imágenes, diagnostico medico) y recomendación (predicción de la detección de intrusos en la red o predicción de ataque de denegación de servicio).
- Análisis semántico y procesamiento del lenguaje natural: Es el proceso de relacionar las estructuras sintácticas de párrafos, oraciones y palabras con el nivel de escritura como un todo.
- Recuperación de información: Permite buscar información en un documento, buscar documentos y buscar metadatos que describan los datos y bases de datos de sonidos e imágenes. [6]

Gráfico 2 Aplicaciones del aprendizaje automático [6]



Por otra parte, también podemos distinguir diferentes tipos de ML, según el tipo de datos y el problema a trabaja de la siguiente forma [7]:

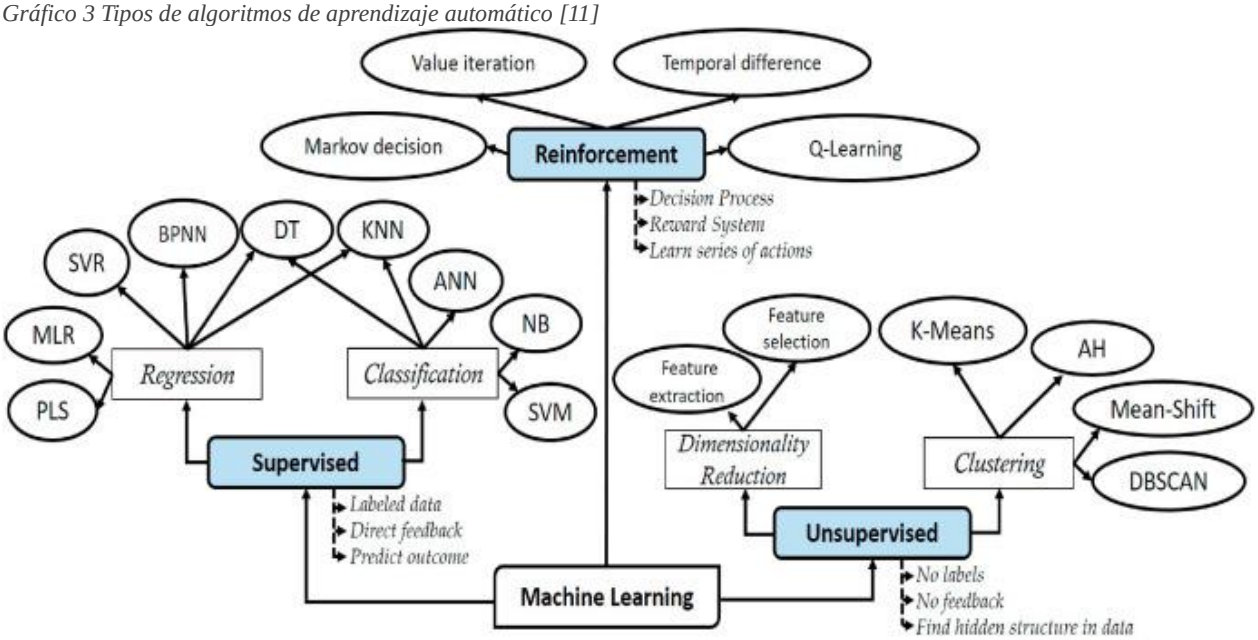
Tabla 1 Tipos de aprendizaje automático

Tipo	Descripción
Aprendizaje supervisado	*El aprendizaje supervisado* requiere un conjunto de datos de entrenamiento que cubra ejemplos para la entrada, así como respuestas etiquetadas o valores objetivo para la salida. Los pares de datos de entrada y salida en el conjunto de entrenamiento se utilizan luego para calibrar los parámetros abiertos del modelo ML. Una vez que el modelo se ha entrenado con éxito, se puede usar para predecir la variable de destino *y* dados puntos de datos nuevos o no vistos de las características de entrada *x*. Con respecto al tipo de aprendizaje supervisado, podemos distinguir además entre problemas de regresión, donde se predice un valor numérico (por ejemplo, número de usuarios), y problemas de clasificación, donde el resultado de la predicción es una afiliación de clase categórica como "observadores" o "compradores".
Aprendizaje sin supervisión	*El aprendizaje no supervisado* tiene lugar cuando se supone que el sistema de aprendizaje detecta patrones sin etiquetas o especificaciones preexistentes. Por lo tanto, los datos de entrenamiento solo consisten en variables *x* con el objetivo de encontrar información estructural de interés, como grupos de elementos que comparten propiedades comunes (conocidos como agrupamiento) o representaciones de datos que se proyectan desde un espacio de alta dimensión a uno más bajo (conocida como reducción de dimensionalidad) [8]
Aprendizaje reforzado	En un sistema *de aprendizaje por refuerzo*, en lugar de proporcionar pares de entrada y salida, describimos el estado actual del sistema, especificamos un objetivo, proporcionamos una lista de acciones permitidas y sus restricciones ambientales para sus resultados, y dejamos

Tipo	Descripción
	que el modelo ML experimente el proceso de lograr el objetivo por sí mismo utilizando el principio de prueba y error para maximizar una recompensa. Los modelos de aprendizaje por refuerzo se han aplicado con gran éxito en entornos de mundo cerrado como los juegos [9], pero también son relevantes para sistemas multiagente como los mercados electrónicos[10].

Finalmente, según la tarea de aprendizaje, el ML ofrece varias clases de algoritmos según los cuales se ajustan a las especificaciones y variantes, en donde podemos encontramos modelos de regresión, algoritmos basados en instancias, arboles de decisión, métodos bayesianos y redes neuronales artificiales (ANN)[6] [7][11]. En la siguiente grafica se pueden observar las diferentes técnicas disponibles según el tipo de aprendizaje (ver gráfico 3):

Gráfico 3 Tipos de algoritmos de aprendizaje automático [11]



SVM: Support vector machine.
NB: Naive Bayes.
ANN: Artificial neural network.
k-NN : Nearest neighbor.
DT: Decision tree
BPNN: Back propagation neural network.
SVR: Support vector regression.
MLR: Multiple linear regression.
PLS: Partiall least squares.
HA: Hierarchical algorithm.
DBSCAN: Density based spatial clustering of application with noise.

4.1.2. Aprendizaje profundo o Deep Learning (DL)

El aprendizaje profundo es un subconjunto del aprendizaje automático. Es una red neuronal con una gran cantidad de capas y parámetros. La mayoría de los métodos de este tipo utilizan arquitecturas de redes neuronales, por medio de múltiples capas de unidades de procesamiento no lineal para la extracción y transformación de características[6].

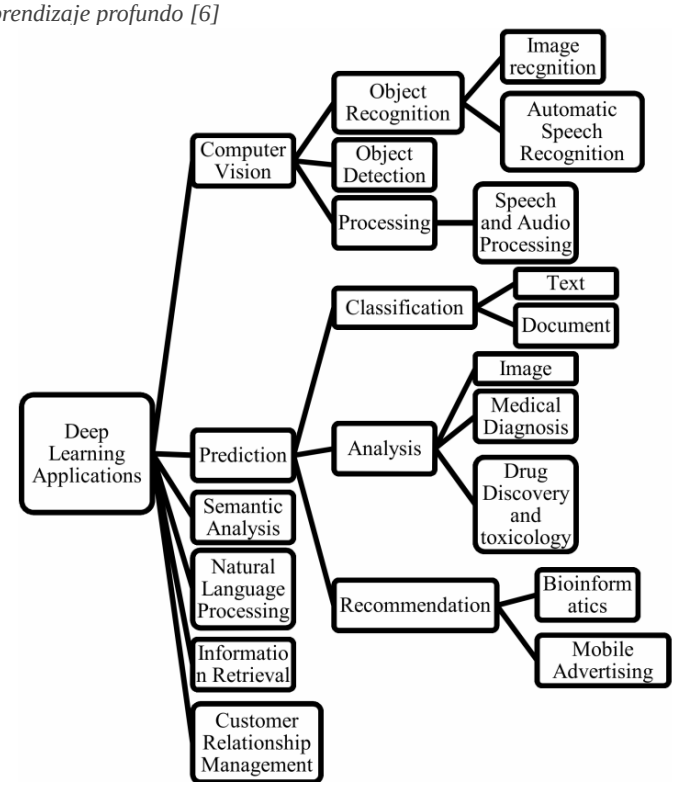
En DL, las capas inferiores cercanas a la entrada de datos aprenden características simples, mientras que las capas superiores aprenden características más complejas derivadas de las características de la capa inferior. Por lo cual pueden usar operaciones avanzadas como circunvoluciones o activaciones múltiples en una neura en lugar de usar una función de activación simple. Estas características permiten que las redes neuronales profundas se

alimenten con datos de entrada sin procesar y descubran automáticamente una representación que se necesita para la tarea de aprendizaje. Siendo particularmente útiles con datos grandes y de alta dimensión como datos de texto, imagen, video, voz y audio. Mientras con datos de baja dimensión con datos limitados de entrenamiento, las ML pueden producir resultados superiores [6] [7].

Las principales aplicaciones de aprendizaje profundo se enfocan a (ver gráfico 3):

- Visión artificial: Permite el reconocimiento automático de voz e imágenes, procesamiento de voz y audio y procesamiento de artes visuales.
- Predicción: Permite la clasificación, análisis y recomendación. Aplicados principalmente en el descubrimiento de nuevos fármacos, toxicología, bioinformática y publicidad móvil.
- Análisis semántico
- Procesamiento del lenguaje natural
- Recuperación de información
- Gestión de la relación con el cliente

Gráfico 4 Aplicaciones de aprendizaje profundo [6]



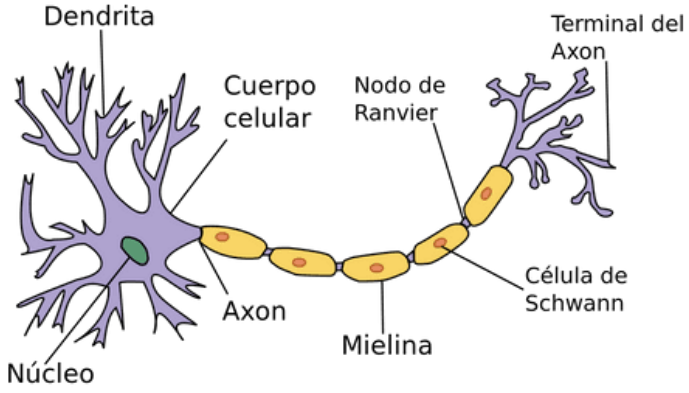
4.1.3. Redes neuronales

4.1.3.1. Neurona biológica

Las redes neuronales artificiales tratan de imitar el funcionamiento del cerebro, siendo su elemento básico las neuronas por lo que es necesario conocer el funcionamiento de una neurona biológica (ver gráfico 5) la cual se compone:

- **Cuerpo celular o soma:** Es la parte central de la neurona que contiene el núcleo de esta y es la responsable de la síntesis de proteínas y otras moléculas necesarias para el funcionamiento celular.
- **Núcleo:** Se encargar de realizar el procesamiento de la información que llega a la neurona.
- **Dendritas:** Extensiones ramificadas que se extienden desde el cuerpo celular y son las encargadas de recibir señales de otras neuronas o de células sensoriales y las transmiten hacia el cuerpo celular.
- **Axón:** Prolongación larga y delgada recubierta por mielina (aislante que acelera la transmisión del impulso nervioso) que se extiende desde el cuerpo celular para realizar la trasmisión de la señal generada por la neurona hacia otras células.
- **Terminal sináptica o del axón:** En este punto se encuentran las terminaciones o botones sinápticos las cuales establecen contacto con otras células (neuronales o efectoras). En sus botones sinápticos se encuentran vesículas sinápticas las cuales almacena neurotransmisores (sustancias químicas) que se liberan en la sinapsis para transmitir la señal de una neurona a otra.
- **Sinapsis:** Es el punto de contacto funcional entre el axón de una neurona y las dendritas o el cuerpo celular de otra neurona. Y es aquí donde se produce la transmisión de señales químicas las cuales son medidas por neurotransmisores para producir el impuso nervioso[12], [13], [14].

Gráfico 5 Neurona biológica [14]

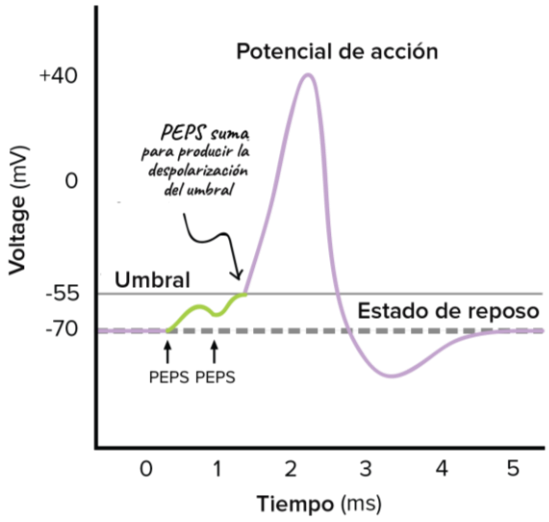


En general, un cerebro humano tiene unos 80 mil millones de neuronas y estas generan aproximadamente más de 100 millones de millones de conexiones desde que se genera un estímulo (visual, auditivo, sensitivo, entre otros) hasta que se genera la respuesta (ejemplo movimiento muscular) con una velocidad de procesamiento entre 1 y 2 milisegundos[12].

El proceso simplificado de una neurona biológica es la siguiente (ver gráfico 6):

- El núcleo de la célula se encuentra en reposo, con una carga potencial de unos -70mV.
- Cuando llegan señales de las sinapsis de otras neuronas, el cuerpo de la célula va perdiendo carga negativa.
- El cuerpo de la célula llega a situarse con un potencial de unos +30mV.
- Finalmente, se produce un proceso que descarga esta tensión, devolviendo a la célula a su estado de reposo[12].

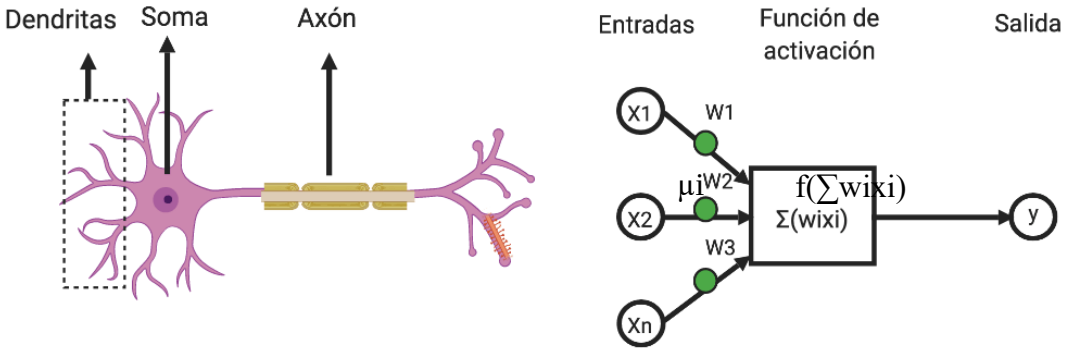
Gráfico 6 Potencial de acción de una neurona biológica[13]



4.1.3.2. Neurona artificial

Las neuronas artificiales son muy similares en estructura y funcionamiento que sus análogas naturales, y están compuestas de la siguiente forma (ver gráfico 7):

Gráfico 7 Neurona artificial [12]



- **μ*i*** representa la neurona i-ésima.
- ***y*** representa el resultado que genera la neurona i, que equivale a la salida que se produce en el axón de la neurona biológica.
- ***wi*** representa el valor de inhibición o excitación entre neuronas o entradas. Son los equivalentes al efecto de los neurotransmisores sobre la sinapsis. Cuando ***wi > 0*** se produce

una sinapsis excitadora; cuando ***wi < o*** se produce una sinapsis inhibidora; cuando ***wi = o*** se produce ausencia de conexión.

- ∑ wixi, representa la suma de señales que le llegan a la neurona ***ui***.
- f(∑wixi) representa la función de salida o transferencia (equivalente a la acumulación de voltaje) y es la responsable de evitar la linealidad del cómputo. Las funciones de transferencia o activación *f* más utilizadas son[12]:
    - 0/1 step.
    - -1/+1 step.
    - Sigmoid $(1/ 1 + e^{-x})$.
    - Tanh $(e^{x} – e^{-x})/ (e^{x} + e^{-x})$
    - Relu max(0,x)
    - Maxout.

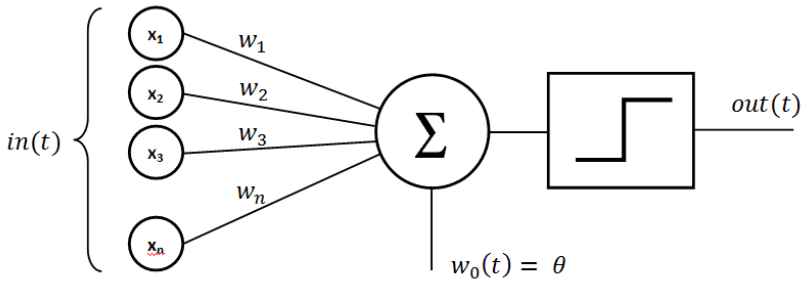
4.1.3.3. Perceptrón

El perceptrón es el caso más sencillo de red neuronal, en el cual solo existe una neurona de cómputo y una única salida de resultado. Pero esto no quiere decir que no sea capaz de recibir diferentes datos de entrada para efectuar cálculos que son capaces de identificar características o tendencias en los datos de entrada.

Se trata de un algoritmo de aprendizaje supervisado como clasificador binario, el cual le permite aprender y realizar predicciones a partir de datos ya etiquetados. En general, la regla de aprendizaje del perceptrón se centra en aprender los coeficientes de peso óptimo para encenderse o no, a partir de múltiples señales de entrada. Si la suma de las señales supera un umbral determinado se produce un resultado u otro.

El perceptrón es una función matemática donde los datos de entrada (x) se multiplican por los coeficientes de peso (w) para producir un resultado el cual puede ser positivo o negativo o capaz de activar o no la neurona artificial. Una vez el resultado predicho se compara con el resultado conocido. Si se encuentran diferencias, el error se retropropaga para permitir ajustar los pesos. (ver gráfico 7)

Gráfico 8 Perceptrón con N entradas



4.1.3.4. Red neuronal y redes multicapa

Una red neuronal artificial es un modelo computacional inspirado en el cerebro

humano que se utiliza para realizar tareas de aprendizaje automático. Consiste en un conjunto interconectado de nodos llamados neuronas artificiales, que trabajan en conjunto para procesar información y realizar predicciones.

El funcionamiento de una red neuronal artificial se puede resumir en los siguientes pasos:

- **Entrada de datos:** La red neuronal recibe una serie de datos de entrada, que pueden ser imágenes, texto, sonido u otro tipo de información. Estos datos se representan numéricamente y se utilizan como la señal de entrada para la red.
- **Peso y sesgo:** Cada conexión entre las neuronas tiene asociado un peso que determina la importancia de la señal transmitida. Además, cada neurona tiene un valor de sesgo que se suma a la señal ponderada. Estos pesos y sesgos se inicializan aleatoriamente y se ajustarán durante el proceso de aprendizaje.
- **Propagación hacia adelante:** Los datos de entrada se propagan a través de la red neuronal desde la capa de entrada hacia las capas ocultas y finalmente hacia la capa de salida. Cada neurona en una capa oculta toma las señales de entrada ponderadas y las procesa utilizando una función de activación no lineal, generando una señal de salida que se transmite a las neuronas de la siguiente capa.
- **Función de activación:** La función de activación introduce no linealidad en la red y permite que esta pueda aprender relaciones y patrones complejos en los datos. Algunas funciones de activación comunes incluyen la función sigmoide, la función ReLU (Rectified Linear Unit) y la función tangente hiperbólica.
- **Cálculo del error:** Después de que la señal se propaga hasta la capa de salida, se compara con la salida esperada para calcular el error. Este error es una medida de qué tan bien está realizando la red neuronal en la tarea que se le ha asignado.
- **Retropropagación del error:** El error se propaga hacia atrás a través de la red neuronal desde la capa de salida hasta la capa de entrada. Esto se hace ajustando los pesos y sesgos de las conexiones en la dirección opuesta al gradiente del error con respecto a los pesos y sesgos. El objetivo es minimizar el error, lo que implica ajustar los pesos y sesgos para mejorar las predicciones de la red.
- **Ajuste de pesos:** Durante la retropropagación del error, se utilizan algoritmos de optimización, como el descenso del gradiente, para ajustar los pesos y sesgos de la red neuronal. Estos algoritmos actualizan los valores de los pesos y sesgos en función del gradiente del error con respecto a los mismos, de manera que se reduzca el error de la red.
- **Iteración y aprendizaje:** Los pasos anteriores se repiten iterativamente para múltiples ejemplos de entrenamiento, ajustando gradualmente los pesos y sesgos de la red

neuronal a medida que se reduce el error. Este proceso se conoce como entrenamiento de la red neuronal. Cuantas más iteraciones de entrenamiento se realicen, más precisa será la red neuronal en la tarea específica para la que ha sido entrenada.

- **Predicción:** Una vez entrenada, la red neuronal puede realizar predicciones sobre nuevos datos de entrada. Los datos de entrada se propagan a través de la red utilizando los pesos y sesgos ajustados durante el entrenamiento, y la salida generada por la capa de salida se interpreta como la predicción realizada por la red neuronal.

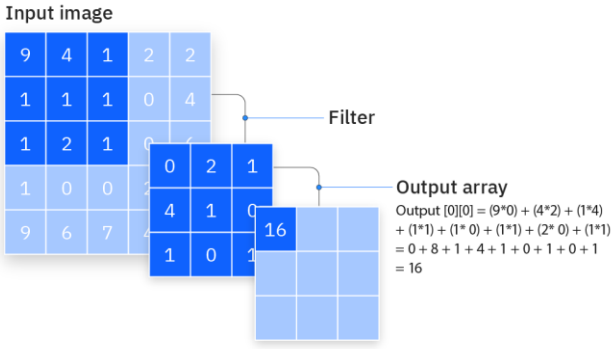
4.1.4. Redes convolucionales

Las Redes Convolucionales Convolutional Neural Networks (CNN) son un tipo de arquitectura de red neuronal especialmente diseñada para procesar datos estructurados en forma de cuadricula, como las imágenes. Siendo compuestas por tres tipos principales de capas (convolucional, agrupación y completamente conectada). Con cada capa, la CNN aumenta su complejidad, identificando porciones mayores de la imagen. Las capas iniciales se centran en características simples como colores o borde y medida que los datos de la imagen avanzan a través de las capas, comienzan a reconocer elementos o formas más grandes del objeto hasta que finalmente identifican el objeto desea. En especial estas redes han demostrado ser muy efectivas en tareas relacionadas con la visión por computadora, como la clasificación de imágenes, la detección de objetos y la segmentación semántica [12], [15], [16].

Estas redes se encuentras compuestas por:

- **Capas Convolucionales:** Las capas convolucionales son fundamentales en las CNN debido a que es donde se produce la mayor parte de los calculos. Es la primera capa de una red convolucional y pueden ir seguidas de capas convolucionales adicionales o capas de agrupación. Por otra parte, estas capas utilizan filtros para realizar operaciones de convolución en los datos de entrada. Estos filtros se desplazan a través de la imagen, aprendiendo características locales como bordes, texturas y patrones visuales. Por lo cual la entrada tendrá tres dimensiones (alto, ancho y profundidad) que corresponde a RGB en una imagen [12], [15], [16].

Gráfico 9 Capa Convolucional [16]



- **Capa convolucional adicional**: Una capa convolucional puede seguir a la capa convolucional inicial, y en estos casos la estructura de la CNN puede convertirse en jerárquica, ya que las capas posteriores pueden ver los pixeles dentro de los campos receptivos de las capas anteriores, convirtiendo la imagen en valores numéricos, lo que permite a la red neuronal interpretar y extraer patrones relevantes.

Gráfico 10 Capa convolucional adicional

- **Capas de Agrupación (Pooling):** Despues de las capas convolucionales, se suelen aplicar capas de agrupación para reducir la dimensionalidad y la cantidad de parámetros en la red. El pooling, comúnmente max pooling, selecciona el valor máximo de un conjunto de valores dentro de una región definida o se puede realizar pooling promedio, el cual calcula el valor promedio dentro del campo receptivo para enviarlo a la matriz de salida[12], [15], [16].
- **Capas Densa Totalmente Conectadas (Fully Connected):** Las capas densas siguen a las capas convolucionales y de agrupación. Estas capas toman las características aprendidas en las capas anteriores y las utilizan para realizar la clasificación final[12], [15], [16].
- **Funciones de Activación:** Las funciones de activación, como ReLU (Rectified Linear Unit), se aplican despues de las operaciones convolucionales para introducir no linealidades en la red y permitir que la red aprenda patrones más complejos [12], [15], [16].
- **Relleno cero:** Se utiliza cuando los filtros no se ajustan a la imagen de entrada. Esto establecer todos los elementos que quedan fuera de la matriz de entrada en cero, lo que produce una salida mayor o del mismo tamaño. Entre los que encontramos relleno valido, mismo relleno y relleno completo [12], [15], [16].
- **Capas de Normalización y Regularización:** Capas como Batch Normalization y técnicas de regularización, como Dropout, se utilizan para mejorar la generalización y la estabilidad del entrenamiento [12], [15], [16].

- **Tamaño del Filtro y Stride**: El tamaño del filtro y el paso (stride) en las capas convolucionales afectan la resolución espacial y la cantidad de informacion aprendida pro la red. Elecciones comunes de filtros son 3x3 con stride de 1 [12], [15], [16].
- **Funciones de Perdida y Optimización**: La elección de la función de perdida y el algoritmo de optimización depende de la tarea especifica. Por ejemplo para clasificación, se utiliza la entropía cruzada y algoritmos como el descenso de gradiente estocástico (SGD) o adaptadores como Adam [12], [15], [16].

4.1.5. Enriquecimiento de datos (Data Augmentation)

El enriquecimiento de datos, también conocido como data augmentation, es una técnica utilizada en el aprendizaje automático para aumentar la cantidad y la variedad de datos de entrenamiento disponibles. Consiste en aplicar transformaciones o manipulaciones a los datos existentes para crear nuevas muestras sintéticas que sean similares a las originales pero que presenten variaciones en aspectos específicos [12], [15].

El principal objetivo del enriquecimiento de datos es mejorar la capacidad del modelo para generalizar y realizar predicciones precisas en situaciones de datos no existentes. Al aumentar la diversidad de los datos de entrenamiento, se reducen los problemas de sobreajuste y se mejora la capacidad de adaptación a nuevas instancias[12], [15].

Dentro de las principales técnicas de enriquecimiento de datos encontramos:

- **Reflejo horizontal**: Se realiza una inversión de los datos en horizontal, como si se estuviera viendo en un espejo.
- **Rotación**: Se giran las imágenes en un ángulo determinado, como 90 grados o 180 grados
- **Desplazamiento**: Se desplazan los datos en diferentes direcciones, como desplazamiento vertical u horizontal.
- **Zoom:** Se aplica un factor de zoom a los datos, aumentando o disminuyendo su escala.
- **Cambio de brillo y contraste**: Se ajustan los niveles de brillo y contraste de los datos.
- **Agregado de ruido**: Se introduce ruido aleatorio en los datos para simular variaciones en el entorno de captura.
- **Recorte aleatorio**: Se realiza un recorte aleatorio de los datos para enfocarse en partes específicas.

Estas transformaciones se aplican de manera aleatoria y controlada durante el proceso de entrenamiento, generando nuevas instancias de datos que enriquecen el conjunto de entrenamiento. Al utilizar el enriquecimiento de datos, se aumenta la cantidad de ejemplos disponibles para el modelo, lo que puede mejorar su capacidad de generalización y rendimiento en situaciones del mundo real[12], [15].

4.1.6. Aprendizaje por transferencia (Transfer Learning)

La transferencia de aprendizaje es una tecnica de machine learning donde un modelo entrenado en una tarea se utiliza como punto de partida para entrenar un modelo en una tarea relacionada o similar. Esta tecnica se utiliza para aprovechar el conocimiento adquirido por el modelo en la tarea original y acelerar el proceso de entrenamiento o mejorar el rendimiento en la nueva tarea a entrenar en la que se dispone de datos limitados para la nueva tarea.[12], [15] En general, existen dos formas comunes de transferencia de aprendizaje:

Transferencia de Aprendizaje Basada en Características (Feature-Based Transfer Learning):

- Se extraen las características aprendidas por un modelo pre-entrenado y se utilizan como entrada para un nuevo modelo entrenando la tarea de interes.
- El modelo pre-entrenado se conoce como el "backbone" y generalmente se congela durante el entrenamiento del nuevo modelo, evitando así que los pesos se modifiquen en estas capas.
- Para finalizar se añade una capa adicional (o varias capas) al modelo para adaptarse a la nueva tarea.

Transferencia de Aprendizaje Basada en Modelo (Model-Based Transfer Learning):

- Se parte de un modelo pre-entrenado completo y se ajusta para adaptarse a una nueva tarea.
- En este enfoque, no solo se utilizan las características aprendidas, sino que también se ajustan los pesos de algunas capas del modelo pre-entrenado durante el entrenamiento en la nueva tarea.
- Este enfoque puede ser útil cuando las tareas original y nueva son lo suficientemente similares.

Ademas de aprovechar el conocimiento adquirido por el modelo en la tarea original, la transferencia de aprendizaje también permite:

- Reducir el tiempo de entrenamiento significativamente para alcanzar un rendimiento aceptable en la nueva tarea.
- Mejora el rendimiento en la nueva tarea cuando se cuenta con datos limitados.
- Mejora la generalización.

Siendo especialmente útil en una variedad de tareas, como el reconocimiento de objetos, clasificación de texto y otras tareas de visión por computadora y procesamiento del lenguaje natural[12], [15].

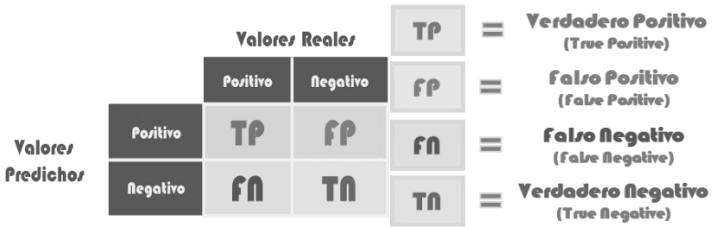
4.1.7. Métricas de evaluación de modelos

Para medir el rendimiento y la eficacia de los modelos de clasificación binaria en placas de laboratorio de cultivos de leishmaniasis, utilizamos las siguientes métricas las cuales nos proporcionaron informacion cuantitativa sobre que tan bien los modelos estan haciendo

predicciones en comparación con los valores reales. Lo cual nos permitirá evaluar el rendimiento, seleccionar el mejor modelo, optimizar hiperparametros, identificar problemas de clasificación, entre otros. A continuación describimos cada una de las métricas utilizadas:

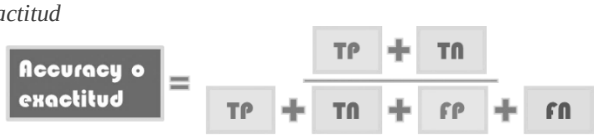
Matriz de confusión: Proporciona una descripción más detalla del rendimiento del modelo al mostrar el número de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos [17], [18], [19].

Gráfico 11 Métrica Matriz de confusión



Accuracy o exactitud: Mide la proporción de casos que son clasificados correctamente por el modelo, ya sean casos positivos o negativos [17], [18], [19].

Gráfico 12Métrica Accuracy o exactitud



Sensibilidad o Recall: Establece la capacidad de un modelo para detectar correctamente los casos positivos. Se expresa por medio de la proporción de casos positivos que son identificados correctamente en un modelo [17], [18], [19].

Gráfico 13 Métrica Recall o Sensibilidad

Especificidad: Establece la capacidad de un modelo para detectar correctamente los casos negativos. Se expresa por medio de la proporción de casos negativos que son identificados correctamente en un modelo [17], [18], [19].

Gráfico 14Métrica Especificidad

Precisión: Mide la proporción de instancias positivas correctamente identificadas entre todas las instancias identificadas como positivas. El denominador incluye todos los casos identificados como positivos [17], [18], [19].

Gráfico 15Métrica Precisión

$$\text{Precisión} = \frac{TP}{TP + FP}$$

F1-score: Medida armónica de la precisión y el Recall que proporciona una evaluación global del rendimiento de un modelo entre 0 y 1, siendo 1 el valor optimo[17], [18], [19].

Gráfico 16Métrica F1-Score

$$\text{F1-score} = 2 \times \frac{\text{Precisión} \quad \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}}$$

VPP: Mide la proporción de verdaderos positivos respecto a todos los casos identificados como positivos por el modelo [17], [18], [19].

Gráfico 17Métrica Valor Predictivo Positivo (VPP)

$$\text{VPP} = \frac{TP}{TP + FP}$$

VPN: Mide la proporción de casos en los que el modelo predijo correctamente un resultado negativo [17], [18], [19].

Gráfico 18Métrica Valor Predictivo Negativo

$$\text{VPN} = \frac{TN}{TN + FN}$$

Curva ROC: La curva ROC (Receiver Operating Characteristic) es una herramienta grafica utilizada en estadísticas y aprendizaje automatico para evaluar el rendimiento de un modelo de clasificación binaria con diferentes umbrales de decisión. La curva ROC representa la tasa de verdaderos positivos (sensibilidad) en el eje **"Y"** y la tasa de falsos positivos (1-Especificidad) en el eje **"X"** [17], [18], [19].

AUC-ROC: El área bajo la curva ROC proporciona una medida cuantitativa del rendimiento del modelo, donde un AUC de 1.0 indica un rendimiento perfecto, mientras un 0.5 indica un rendimiento similar al azar [17], [18], [19].

4.2. Antecedentes

EL ML y DL en los últimos se han utilizado ampliamente en desafíos complejos en diferentes sectores por medio de aplicaciones médicas, financieras, ambientales, marketing, seguridad industrial, entre otros. Específicamente en el área de la salud, el uso de ML y DL pueden ayudar a mejorar la confiabilidad, el rendimiento, la previsibilidad y la precisión de los sistemas diagnósticos para muchas enfermedades. En los últimos años las publicaciones científicas sobre la implementación de modelos de aprendizaje automático en el área de la salud vienen en incremento. Siendo los principales temas de investigación los problemas asociados a patologías como el cáncer, problemas cerebrales, química médica, uso de sensores portátiles y diagnóstico de enfermedades por medio de imágenes médicas[11].

En cáncer, las aplicaciones más frecuentes del aprendizaje automático se centran en mejorar el rendimiento de la clasificación y predicción de enfermedades por medio de SWM y k-NN, determinar el tratamiento más adecuado por medio de ANN, determinar la eficacia del tratamiento por medio de SVM, BT, RF, RFE, identificación del problema genético por medio BHM y GM, entre otras muchas aplicaciones [11].
También se han publicado varios estudios que utilizan el aprendizaje automático para el desarrollo de nuevos medicamentos, analizar su toxicidad, su efectividad, entro otros muchos usos de la química medicinal [11].
A nivel cerebral se ha utilizado el aprendizaje automático como método de clasificación de patrones en imágenes de resonancias magnéticas para medir diferentes respuestas de comportamiento. Asimismo, se han utilizado en resonancias magnéticas para clasificar el tipo y grado de tumores cerebrales. Se han desarrollado aplicaciones capaces de comprender y distinguir los comportamientos ocurridos en un electroencefalograma (EEG) en pacientes con ataques de epilepsia. También se han crearon interfaces cerebro-computador basadas en EEG y monitoreo del estado mental [11].
Respecto al uso de sensores portátiles presentes en teléfonos móviles, relojes, entre otros dispositivos. Se encuentran también un gran número de publicaciones científicas en donde se han utilizado algoritmos de DL, ANN, SVM, DR, GP, selección de características, HMM y ELM con el objetivo de clasificar el gran volumen de información recopilada, realizar el siguiente de dispositivos, observar movimiento de pacientes, extraer características de asignaciones complejas, análisis de movimientos humanos, reconocimiento de gestos de la mano y de la actividad diaria, identificación de patologías o condiciones de estrés, entre otras [11].
Finalmente, respecto a la aplicación del aprendizaje automático en imágenes médicas, las principales aplicaciones descritas se refieren a la identificación de patrones en imágenes diagnosticas provenientes de radiología, tomografía computarizada, resonancia magnética, tomografía por emisión de positrones (PET), informes de radiología o por medio de imágenes de pruebas de laboratorio [11].

4.2.1. Estudios de diagnóstico automático de leishmaniasis

Asociado al diagnóstico de enfermedades causadas por vectores, como es el caso de la leishmaniasis en humanos. No se encontraron numerosas publicaciones que hiciera uso del aprendizaje automático para el diagnóstico de esta. En general, solo fue posible recuperar 4 publicaciones las cuales describimos a continuación:

Uno de los estudios se enfocó en realizar una predicción espacial de la leishmaniasis cutánea utilizando tres algoritmos de aprendizaje automático entre los que se encontraban los árboles de decisión (DT), regresión de vectores de soporte (SVR) y regresión lineal (LR). Tomando como fuente de información diferentes variables ambientales para predecir la prevalencia de la enfermedad, logrando así estimar la precisión de cada uno de los modelos según la curva ROC y la estimación de área bajo la curva[20].
Otro de los estudios utilizo el aprendizaje automático profundo para mejorar la detección de fármacos candidatos para el tratamiento de la leishmaniasis al detectar compuestos

farmacológicos por medio del conjunto de herramientas denominado deeppurpose el cual integra cinco modelos entre los que se encuentran la red neuronal convolucional (CNN), perceptrones multicapa (MLP), huella dactilar a la luz del día, red neuronal de paso de mensajes (MPNN) y dos codiciadores de proteínas sobre composición de aminoácidos (ACC)[21].

Los restantes dos estudios se enfocaron en el diagnóstico y pronostico en pacientes con leishmaniasis cutánea[22], [23]. Uno de ellos tenía como objetivo detectar casos que no responden al tratamiento para lo cual se distribuyeron en dos grupos de estudio en donde uno de ellos si respondían al tratamiento, mientras el otro no. Para categorizar a los pacientes se utilizaron los algoritmos de SVM, MLP, LVQ y LVQ, además se evaluó la precisión general por medio de la curva ROC, sensibilidad, especificidad[23]. Mientras la otra publicación se enfocó en desarrollar un algoritmo basado en inteligencia artificial para el diagnóstico automático de la leishmaniasis el cual utilizo el algoritmo de Viola-Jones para desarrollar un sistema de detección por medio de la extracción de características, creación de imágenes integrales y clasificación. Esta última utilizo la técnica adaBoost para seleccionar las características discriminatorias y entrenar el clasificador. Encontrando una precisión del 50% en la detección de macrófagos infectados con el parasito[22].

5 Metodología

5.1. Preprocesamiento de los datos

Para desarrollar nuestro proyecto de investigación, el Centro Internacional de Entrenamiento e Investigaciones Médicas CIDEIM proporciono un set de imágenes, las cuales contenían placas de laboratorio con muestras de cultivos celulares con y sin parasitos de Leishmaniasis. Algunas de estas imágenes contaban con informacion del número de células y parasitos por campo, no obstante, para nuestro objetivo esta informacion no era relevante, por lo cual no fue tenía en cuenta. Asimismo, dentro del set de imágenes se encontraban otras imágenes que solo informaban si tenían o no tenían parasitos sin ninguna informacion adicional.

El número total de imágenes suministradas fue de 315, distribuyéndose en 208 imágenes con parasitos y 107 imágenes sin parasitos las cuales se distribuyeron en cuatro dataset (training, validation, training- validation y test) como se pueden ver en la Tabla 2.

Tabla 2 Distribución set de imagenes de Leishmaniasis

Detalle	Tipo de Dataset				
	Training	Validation	Training + Validation	Test	Completo
Con parasitos	116	50	166	42	208
Sin parasitos	60	26	86	21	107
N° total imagenes	176	76	252	63	315
Distribución	**56%**	**24%**	**80%**	**20%**	**100%**

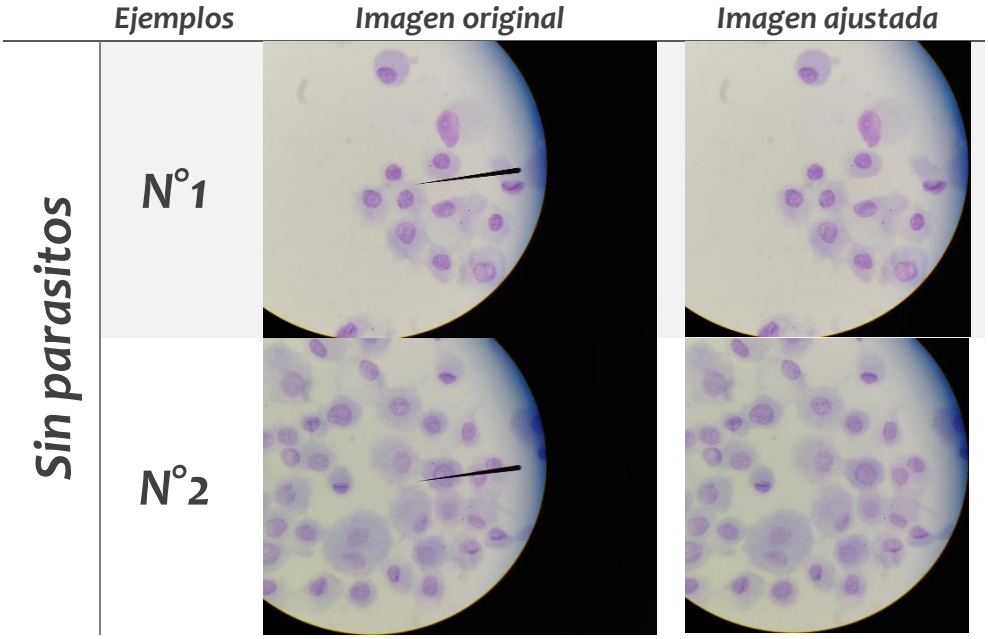
5.1.1. Limpieza y adecuación de imágenes

Con el objetivo de mejorar la calidad de las imágenes que hacen parte de los diferentes sets de datos utilizados en el entrenamiento de modelos de aprendizaje automatico planteados. Se realizo una limpieza y adecuación de imágenes utilizando Adobe Photoshop el cual es una herramienta profesional de edición digital que se utiliza principalmente para retocar imágenes.

En nuestro caso, utilizamos esta herramienta con el objetivo de eliminar características muy llamativas en las imágenes de placas de laboratorio que podrían afectar la capacidad de aprendizaje de nuestros modelos.

En las imágenes con placas de laboratorio sin parasitos, se realizó la eliminación de la guía ocular para microscopio o puntero indicador que se encontraba en varias de las imágenes entregadas por el CIDEIM, así como también la reducción de background o fondo de negro de las imágenes, para garantizar una imagen más adecuada a nuestras necesidades. (Grafico 9) Aun así las gráficas seguían presentando, partes de background o fondo negro, por lo cual se realizó la eliminación del mismos, dejando las imágenes lo más limpias posibles, para evitar un posible sesgo al momento de entrenar los modelos. Esta limpieza se realizó tanto para las imágenes de placas de laboratorio con y sin parasitos de leishmaniasis. (Grafico 10-11)

Gráfico 19 Eliminación de guía ocular para microscopio (puntero indicador) y reducción de background (fondo) en imágenes de placas sin parasitos



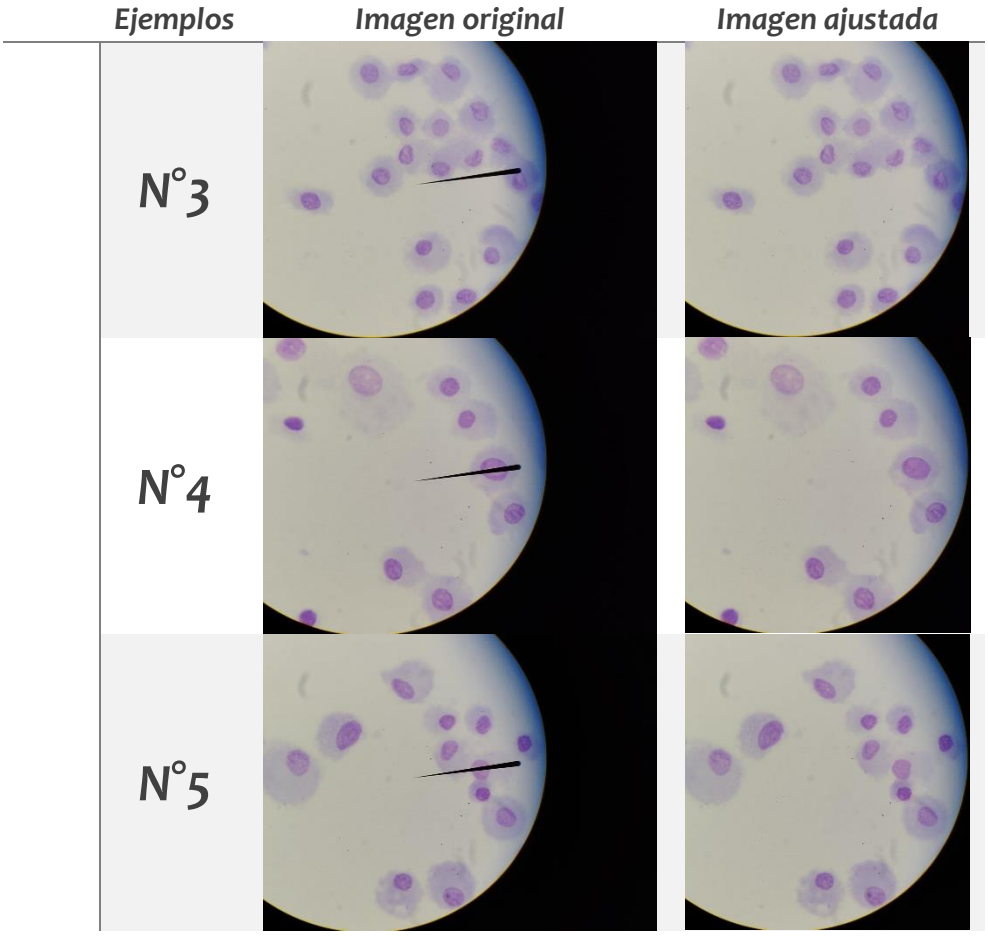
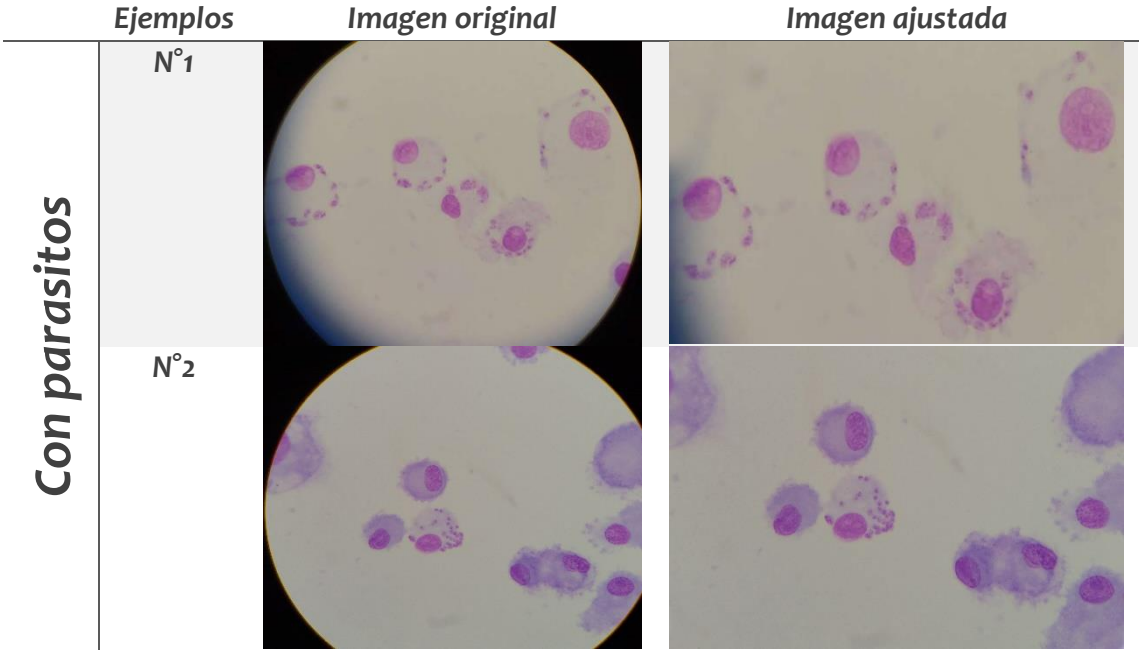


Gráfico 20 Eliminación de background (fondo) en imágenes de placas con parasitos



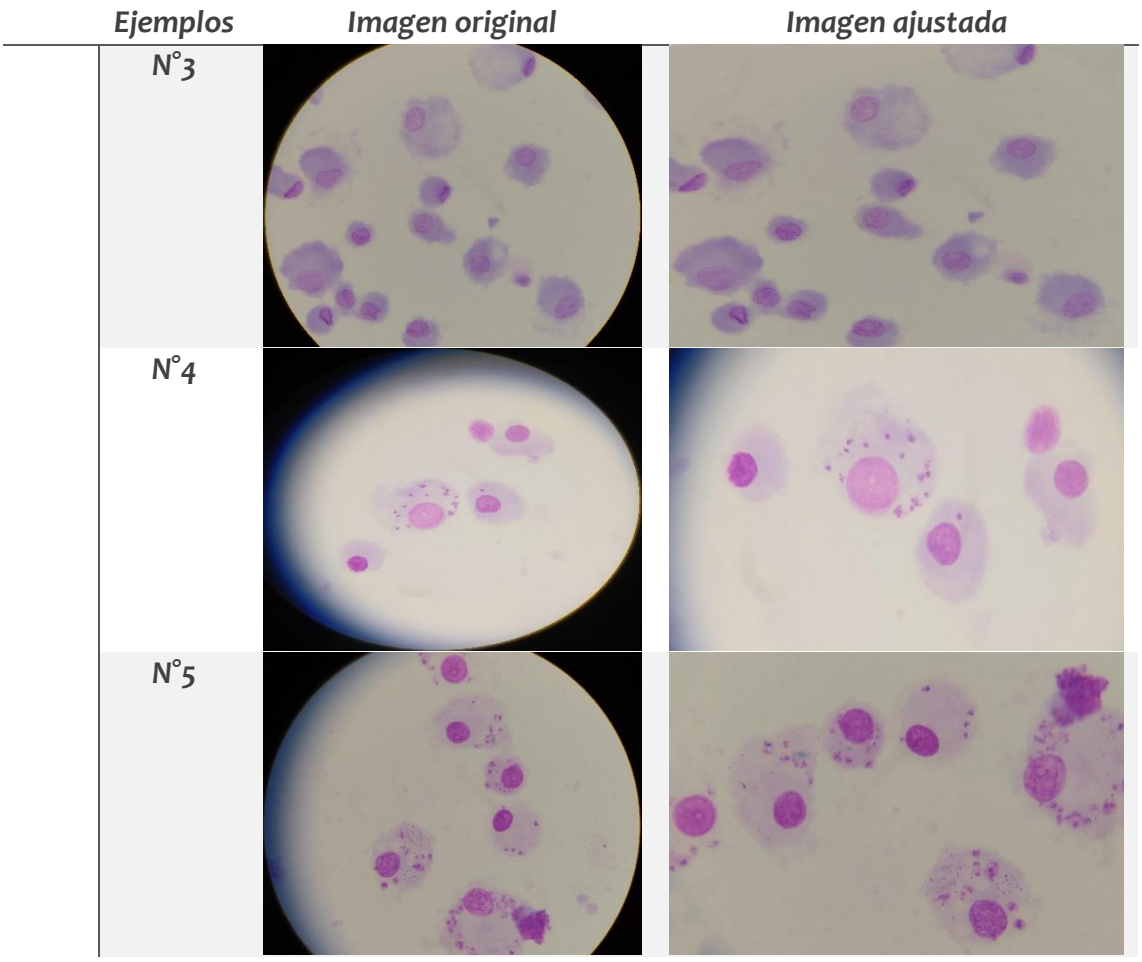
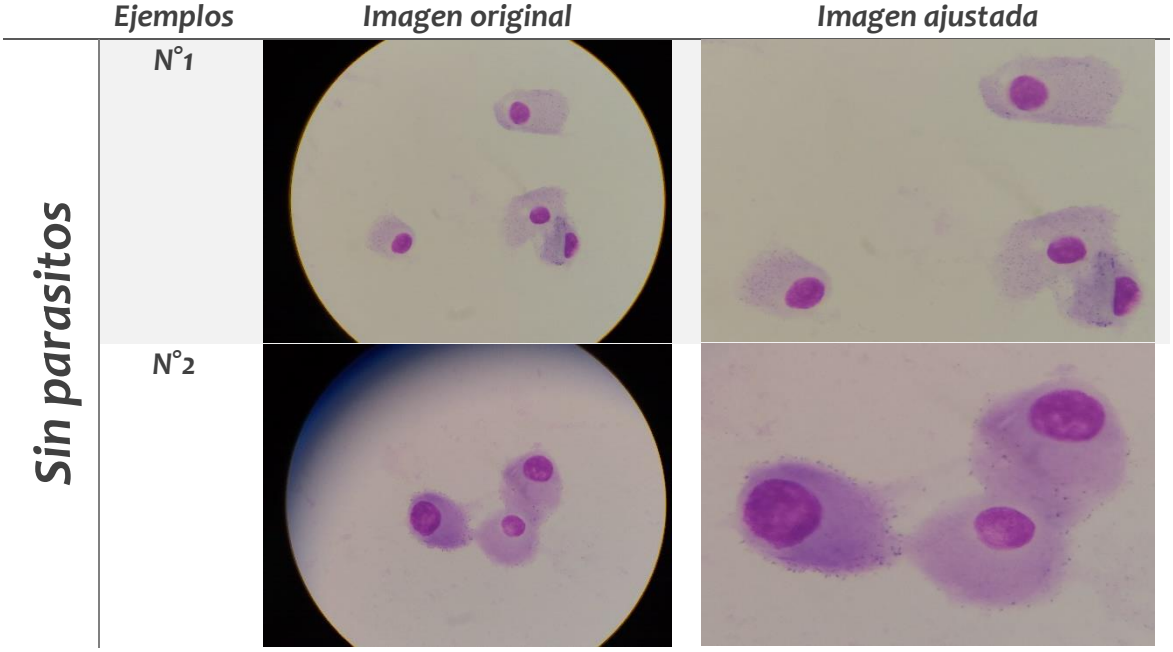
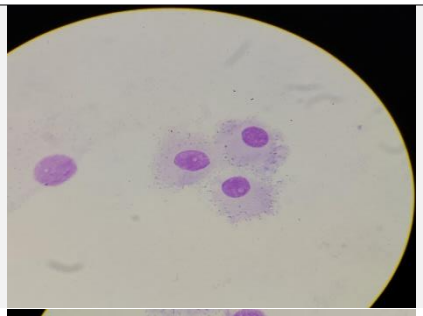
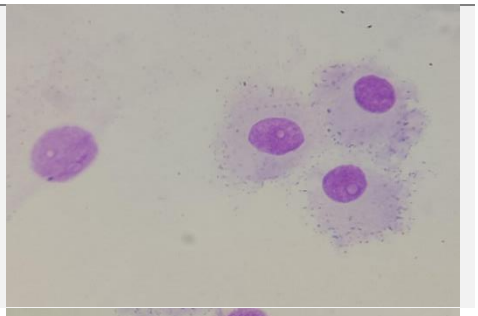
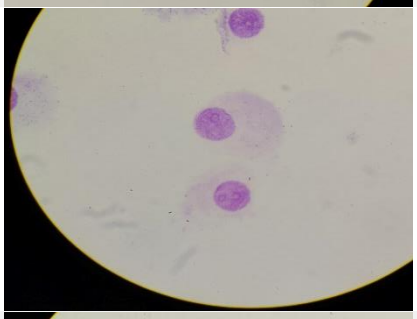
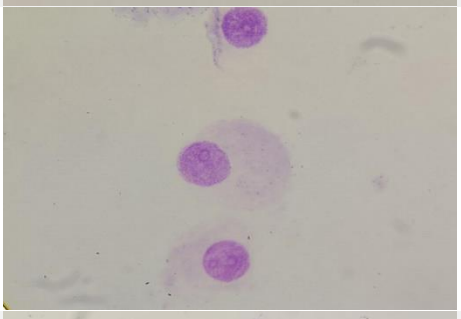
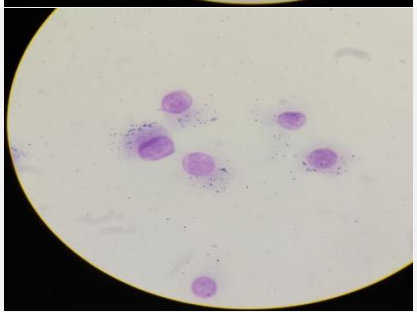
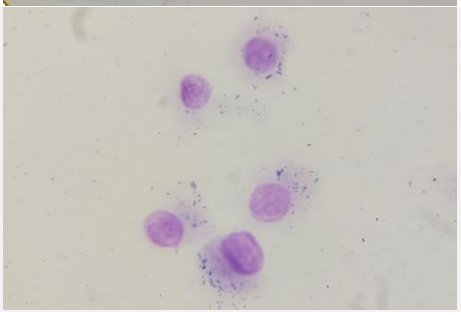


Gráfico 21 Eliminación de background (fondo) en imágenes de placas sin parasitos



Ejemplos	Imagen original	Imagen ajustada
N°3		
N°4		
N°5		

5.1.2. Construcción de datasets

La construcción de dataset se realizó distribuyendo las imágenes en nuestros 4 set de datos anteriormente mencionados (training, validation, training + validation y Test). Inicialmente, las imágenes con y sin parasitos se incluyeron en una carpeta de Google drive para facilitar su posterior duplicación en las carpetas específicas de cada dataset.

Para la construcción de cada dataset se crearon carpetas en Google Drive por cada set de imágenes y por cada una de las categorías de imágenes con y sin parasitos de leishmaniasis. Una vez creadas, se realizó la copia de estas en cada una de las carpetas según la distribución realizada en la Tabla 2 (training, validation, training+validation, test).

Finalmente, los datasets se crearon con la clase **'ImageDataGenerator'** de Keras, la cual permite crear lotes de imágenes en tiempo real durante el entrenamiento de modelos de redes neuronales convolucionales. Así como también generar imágenes aumentadas, que favorecen la diversidad del conjunto de datos de entrenamiento mediante la aplicación de transformaciones aleatorias a las imágenes existentes, como rotación, cambio de zoom, entre otras.

5.1.2.1. Dataset sin aumento de datos

En cada uno de los sets de datos se realizó normalización del valor de los pixeles, dividiéndolos por 255, garantizando así valores de pixeles en el rango de 0 a 1. Esta normalización se realizó para ayudar a mejorar la convergencia de los modelos durante el entrenamiento. Se realizó un redimensionamiento de las imágenes a un tamaño de 224 x 224, dado que la mayoría de los modelos utilizados en nuestro proyecto funcionan con este tamaño. También se estableció un 'batch_size' o número de muestras de 64 a utilizar en cada iteración durante el entrenamiento de un modelo. Del mismo modo, se estableció el tipo de problema como '**binary**' en el parámetro '**class_mode**' dado que se estaba trabajando con una tarea de clasificación binaria. Estableciéndose las etiquetas como 0 para la categoría o clase con parasito y 1 para la categoría o clase sin parasito.

Gráfico 22 Dataset Training

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255
                                   )
train_set = train_datagen.flow_from_directory(ruta_train,
                                              target_size = (width_shape, height_shape),
                                              batch_size = batch_size,
                                              class_mode = 'binary')
```

```
Found 176 images belonging to 2 classes.
```

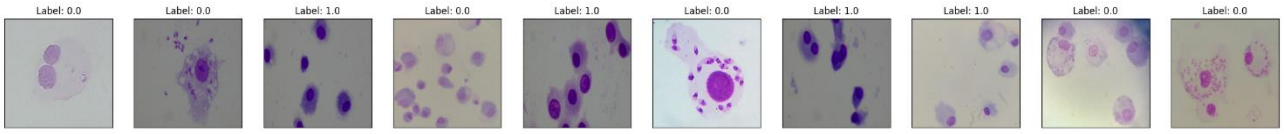


Gráfico 23 Dataset Validation

```
validation_datagen = ImageDataGenerator(rescale = 1./255
                                        )
validation_set = validation_datagen.flow_from_directory(ruta_validation,
                                                        target_size = (width_shape, height_shape),
                                                        batch_size = batch_size,
                                                        class_mode = 'binary')
```

```
Found 76 images belonging to 2 classes.
```

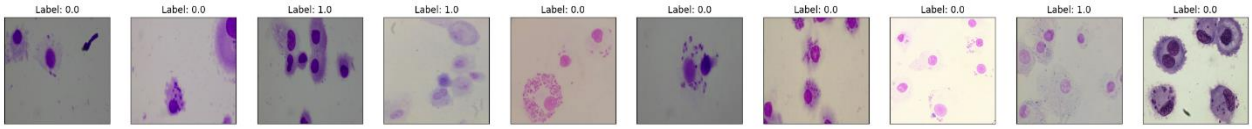


Gráfico 24 Dataset Training + validation

```
train_validation_datagen = ImageDataGenerator(rescale = 1./255
                                              )
train_validation_set = train_validation_datagen.flow_from_directory(ruta_train_validation,
                                                    target_size = (width_shape, height_shape),
                                                    batch_size = 64,
                                                    class_mode = 'binary')
```

```
Found 252 images belonging to 2 classes.
```

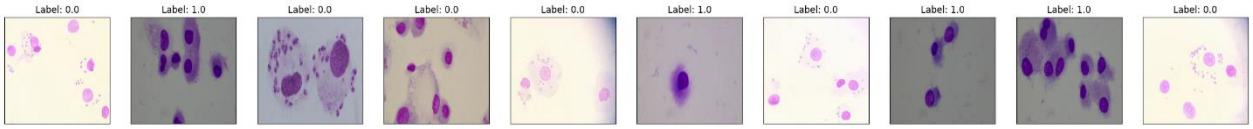
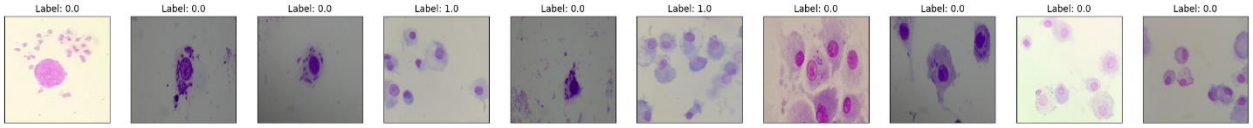


Gráfico 25 Dataset Test

```
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory(ruta_test,
                                            target_size = (width_shape, height_shape),
                                            batch_size = 64,
                                            class_mode = 'binary')
```

```
Found 63 images belonging to 2 classes.
```



5.1.2.2. Dataset con aumento de datos (AD)

Utilizamos aumento de datos (data augmentation) para aumentar la diversidad del conjunto de datos, y mejorar así la capacidad del modelo de generalizar a situaciones diferentes. Por medio de transformaciones aleatorias a las muestras existentes, evitando así el sobreajuste y mejorando la capacidad del modelo para reconocer patrones importantes. En nuestro caso solo utilizamos dos tipos de transformaciones. La primera corresponde a un giro horizontal y la segunda a un giro vertical, debido a que opciones como la rotación, traslación, zoom, entre otros, afectaban la calidad de las imágenes y producía distorsión.

Gráfico 26 Dataset Training AD

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator

train_datagen_aumento = ImageDataGenerator(rescale = 1./255,
                                           horizontal_flip = True,
                                          vertical_flip = True
                                           )
train_set_aumento = train_datagen_aumento.flow_from_directory(ruta_train,
                                                    target_size = (width_shape, height_shape),
                                                    batch_size = batch_size,
                                                    class_mode = 'binary')
```

```
Found 176 images belonging to 2 classes.
```

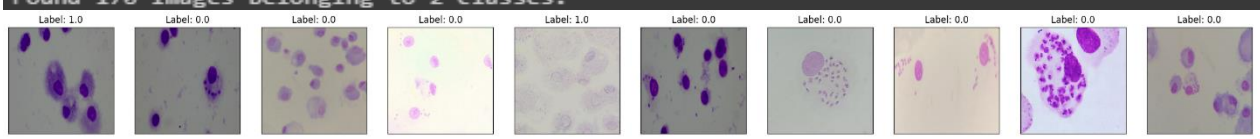
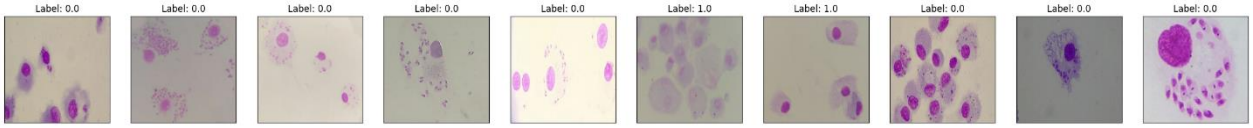


Gráfico 27 Dataset Training + validation AD

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator

train_validation_datagen_aumento = ImageDataGenerator(rescale = 1./255,
                                #rotation_range = 90, #rotar la imagen rottar a 90°
                                horizontal_flip = True,
                                vertical_flip = True
                                 )
train_validation_set_aumento = train_validation_datagen_aumento.flow_from_directory(ruta_train_validation,
                                                target_size = (width_shape, height_shape),
                                                batch_size = batch_size,
                                                class_mode = 'binary')

Found 252 images belonging to 2 classes.
```



5.2. Construcción de modelos

Para realizar la construcción de modelos de clasificación de imágenes con placas de laboratorio de cultivos de leishmaniasis utilizamos cuatro tipos de modelos. El primer modelo corresponde a una construcción propia y se plantearon tres modelos por medio de la tecnica de transferencia de aprendizaje basada en características (Feature-based Transfer Learning) para los modelos VGG16, VGG19 y MobileNetV2.

5.2.1. Modelo Arquitectura CNN (Construcción Propia)

La construcción del modelo convolucional propio se realizó utilizando la estrategia de búsqueda en malla por medio de la clase *HalvingGridSearchCV* de scikit-learn la cual es mucho más rápida para encontrar buenas combinaciones de parámetros, al reducir a la mitad los posibles candidatos con una pequeña cantidad de recursos y selecciona iterativamente a los mejores candidatos, utilizando más y más recursos[24]. Debido a que el uso de la clase *GridSearchCV* es muy exhaustiva y requiere un gran volumen de recursos computacionales, mientras que la clase *RandomizedSearchCV* puede muestrear a número dado de candidatos de forma aleatoria de un espacio de parámetros con un numero especifico de distribución, lo cual no garantizar encontrar el mejor modelo[24].

Gráfico 28 Modelo base propio

```
def create_model(filters, dense_units, dropout_rate, kernel_size, pool_size, num_layers):
    model = keras.Sequential()

    model.add(keras.Input(shape=(224, 224, 3)))

    for _ in range(num_layers - 1):
        model.add(layers.Conv2D(filters, kernel_size=kernel_size, activation="relu"))
        model.add(layers.MaxPooling2D(pool_size=pool_size))

    model.add(layers.Dropout(dropout_rate))
    model.add(layers.Flatten())
    model.add(layers.Dense(dense_units, activation="relu"))
    model.add(layers.Dense(1, activation="sigmoid"))

    model.compile(loss="binary_crossentropy", optimizer='Adam', metrics=["accuracy"])

    return model
```

En la estrategia de búsqueda de parámetros e hiperparametros estableció para la búsqueda de estos algunos parámetros fijos como: el parámetro '*random_state*' = 64 como semilla para el generador de números aleatorios, garantizando así la reproducibilidad de los resultados. Se estableció el parámetro '*factor*' = 3, el cual determina cuantas veces se reduce el número de configuraciones despues de cada iteración. Lo que significa, que se seleccionan el 30% de las configuraciones más prometedoras para la siguiente iteración, lo cual se encuentra relacionado con la estrategia de parada adaptativa en la búsqueda de hiperparametros. También se instauro una validacion cruzada '*cv*'=3, lo cual nos permitió evaluar el rendimiento del modelo con diferentes conjuntos de datos de entrenamiento y prueba, que nos permitió obtener una estimación más robusta en la evaluación del rendimiento del modelo. Ademas de una función de perdida que mide la discrepancia entre las predicciones del modelo y las etiquetas reales del conjunto de entrenamiento, que en nuestro caso corresponde a '*binary_crossentropy*' que es comúnmente utilizada en problemas de clasificación binaria y un algoritmo de optimización '*optimizer*' = *Adam el cual adapta la tasa de aprendizaje de manera adaptativa para cada parámetro durante el entrenamiento, favoreciendo la convergencia.*
Finalmente, se definieron un total de 7 espacios de búsqueda, donde se definieron los hiperparametros que consideramos claves en el modelo y se establecieron rangos o valores específicos para cada hiperparametros en el espacio de búsqueda. Toda esta búsqueda se realizó a '*epochs*'=50, lo que indica el número total de iteraciones completas a través de todo el conjunto de datos de entrenamiento durante el entrenamiento del modelo, ajustando sus pesos para mejorar su rendimiento en cada época. Los cuales describimos a continuación:

Tabla 3Estrategia de búsqueda de parámetros e hiperparametros Modelo Propio

Parámetro e Hiperparametros	***Rango de Búsqueda***
Numero de Capas o Layers Conv	'num_layers': [2, 3, 4, 5, 6, 8, 10]
Filter Layers | **Capa o layer # 1…n**	'filters': [32, 64, 128, 256, 512, 1024]
Dropout Rate	'dropout_rate': [0.1, 0.3, 0.4, 0.5, 0.7, 0.9]
Dense Units	'dense_units': [16, 32, 64, 128, 256, 512, 1024]
Kernel size	'kernel_size': [(3, 3), (4, 4), (5, 5)]
Pool size	'pool_size': [(2, 2), (3, 3), (4, 4)]
Learning Rate	'learning_rate': [0.00001, 0.0001, 0.005, 0.001]

La estrategia de búsqueda de parámetros e hiperparametros se realizó con el dataset training y se validó con el dataset de validation, los cuales se mencionaron anteriormente.

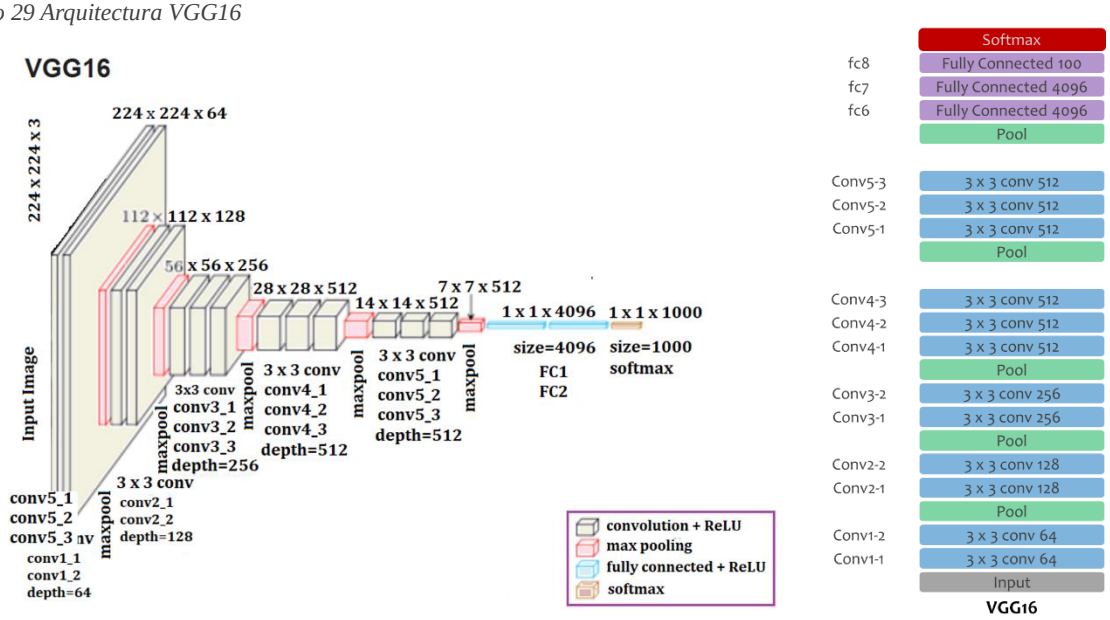
5.2.2. Modelo Arquitectura VGG16

VGG16 es una arquitectura de red neuronal convolucional (CNN) que fue propuesta por el grupo Visual Geometry Group (VGG) en la competencia ImageNet Large Scale Visual Recognition Challenge en 2014 siendo conocida por su simplicidad, eficacia y uniformidad en la disposición de las capas, facilitando su comprensión y entrenamiento. Sin embargo, debido a su profundidad, tiene un gran número de parámetros, lo que puede requerir un poder

computacional significativo y un conjunto de datos lo suficientemente grande para evitar el sobre ajuste.
Descripción de la arquitectura:

- **Capas de Entrada:** La entrada de la red consiste en imágenes RGB de tamaño fijo comúnmente de 224x224 pixeles.
- **Bloques de Convolución:** Esta arquitectura está compuesta por varios bloques de convolución, cada uno de los cuales consta de múltiples capas convolucionales y de agrupación (pooling). Estos bloques de convolución se dividen en conjuntos de capas convolucionales con filtros de tamaño 3x3 y una función de activación ReLU (Rectified Linear Unit).
- **Capas de Agrupación (Pooling):** Despues de cada conjunto de capas convolucionales, se realiza una capa de agrupación máxima (max pooling) con un tamaño de ventana de 2x2 y un paso (stride) de 2.
- **Capas Totalmente Conectadas (Fully Connected Layer):** La clasificación final se realiza por medio de esta capa, despues de varias capas de convolución y agrupación.
- **Capa de Salida:** Esta capa consta de un numero de nodos igual al número de clases en la tarea de clasificación. La función de activación en la capa de salida es Softmax.

Gráfico 29 Arquitectura VGG16



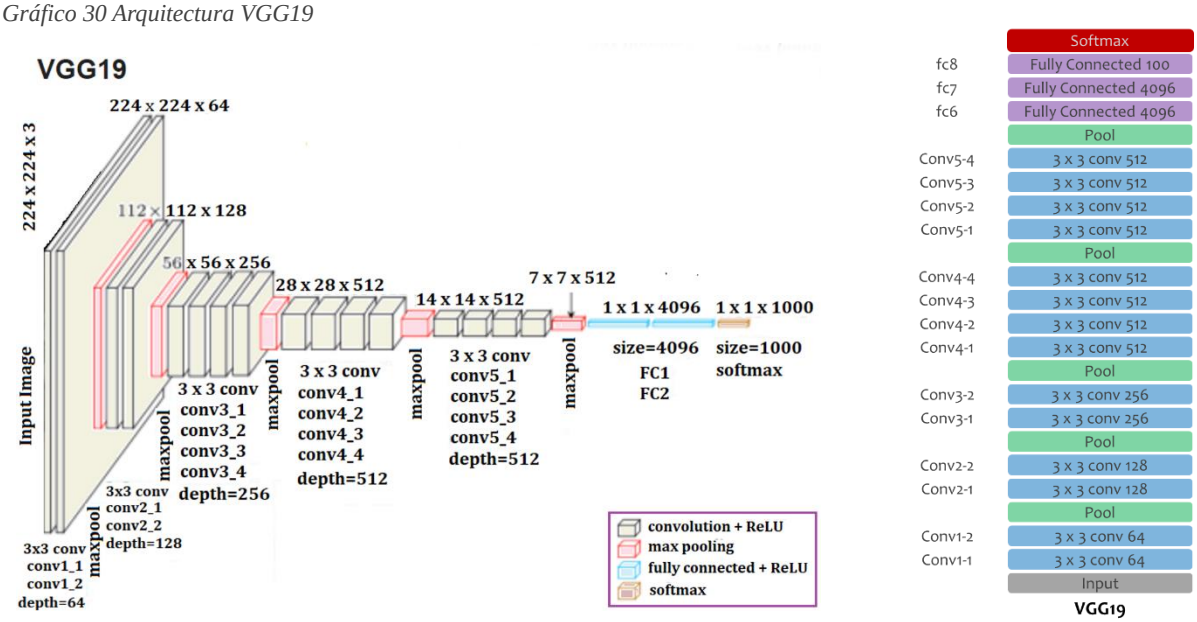
5.2.3. Modelo Arquitectura VGG19

La arquitectura VGG19 es una extensión de la arquitectura VGG16 desarrollada por el grupo Visual Geometry Group (VGG). VGG19 tiene una estructura similar a VGG6, pero con más capas convolucionales y parámetros, lo que le permite una representación más rica y compleja de las características en las imágenes.

Descripción de la arquitectura:

- **Capa de Entrada:** La entrada es igual que en VGG16, la entrada consiste en imágenes RGB de tamaño fijo, comúnmente 224x224 pixeles.
- **Bloques de Convolución:** Esta compuesta por varios bloques de convolución, cada uno con múltiples capas convolucionales seguidas de capas de agrupación máxima (max pooling) y capas convolucionales tienen filtros de tamaño 3x3 y activación ReLU.
- **Capas de Agrupación (Pooling):** Despues de cada conjunto de capas convolucionales, se realiza una capa de agrupación maxima (max pooling) con un tamaño de ventana de 2x 2 y un paso (stride) de 2.
- **Capas Totalmente Conectadas (Fully Connected Layer):** Igual que VGG16, VGG19 termina con capas totalmente conectadas destinadas a realizar la clasificación final.
- **Capa de Salida:** La capa de salida consta de un numero de nodos igual al número de clases en la tarea de clasificación y utiliza la función de activación Softmax.

Gráfico 30 Arquitectura VGG19



5.2.4. Modelo Arquitectura Mobilenetv2

La arquitectura MobileNetV2 es una arquitectura de red neuronal diseñada para aplicaciones de visión por computadora en dispositivos con recursos limitados (dispositivos móviles o sistemas embebidos). Fue diseñada por Google en 2018 como una mejora de la arquitectura original MobileNetV1. Esta arquitectura utiliza bloques de construcción llamados "*inverted residuals*" (residuos invertidos) y "*linear bottlenecks*" (cuellos de botella lineales) para lograr un equilibrio entre la eficiencia computacional y el rendimiento.

- **Capas de entrada:** Entrada de la red son imágenes RGB de tamaño fijo 224 x 224 pixeles.

- **Bloques Inverted Residuals:** compuestos por capa de expansión Conv2D lineal con una función de activación lineal, una capa de convolución profunda Conv2D con una función de activacion no lineal (ReLU) y una capa de proyección lineal Conv2D lineal al final del bloque.
- **Cuellos de Botella Lineales:** Los cuellos de botella lineales en las capas de expansión y proyección permiten reducir el número de canales, para así mejorar la eficiencia.
- **Conexiones Residuales:** Facilitan el entrenamiento de redes más profundas y mejorar el flujo de gradiente.
- **Capas de bloque Invertido Repetitivas:** Estos bloques (bloques inverted residuals) se repiten en la arquitectura para formas capas más profundas
- **Capas de Agrupación (Pooling):** Son capas de agrupación que permiten reducir las dimensiones espaciales antes de la capa de clasificación.
- **Capas Totalmente Conectadas (Fully Connected Layer):** Esta capa permite realizar la clasificación final.
- **Capa de Salida:** Esta capa de salida tiene un numero de nodos igual al número de clases en la tarea de clasificación y utiliza la función de activación Softmax.

Gráfico 31 Diagrama arquitectura Mobilenetv2 [25]

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

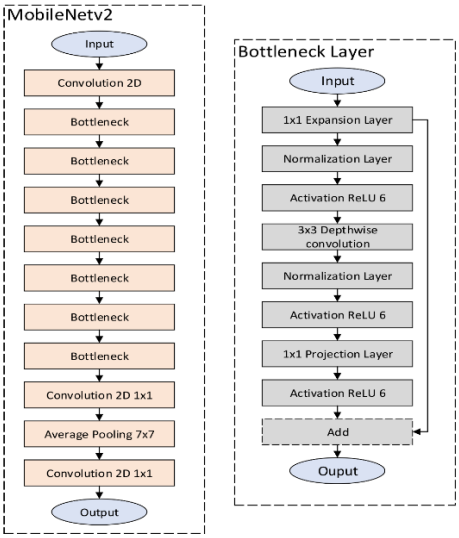
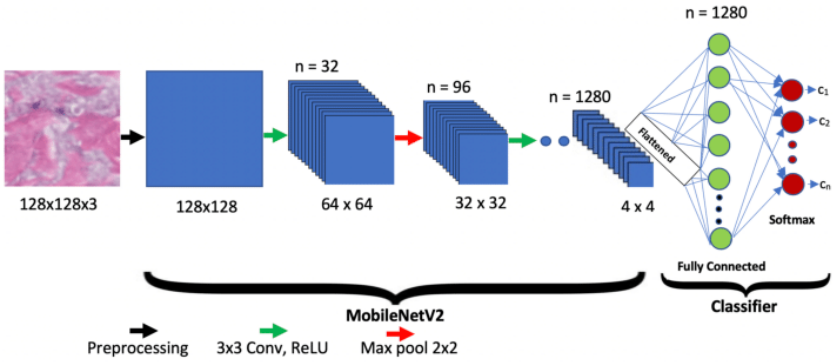


Gráfico 32Arquitectura Mobilenet_v2 [26]



5.3. Búsqueda de Hiperparametros de modelos VGG16, VGG19 y MobileNetV2 (Transfer Learning)

Del mismo modo, que se estableció una búsqueda de hiperparametros para el modelo propio, se estableció una búsqueda de hiperparametros claves para cada uno de los modelos de Transfer Learning en donde se buscaron los siguientes parámetros:

Tabla 4 Estrategia de búsqueda de parámetros e hiperparametros Modelos Transfer Learning

Parámetro e Hiperparametros	***Rango de Búsqueda***
Dense Units	'dense': [64,128,256,512]
Dropout Rate	'dropout': [0.1,0.3,0.5,0.7]
Learning Rate	'learning_rate': [0.0001,0.001,0.01]

En este caso, se utilizó la búsqueda por malla basada en la clase GridSearchCV para cada una de las arquitecturas utilizada, dado que el número de parámetros e hiperparametros no era muy grande y su rango era limitado. Se instauro una validacion cruzada 'cv'=2, una función de perdida 'loss'= "binary_crossentropy", un algoritmo de optimización 'optimizer'= Adam y se instauraron un total de épocas 'epochs'=30. La estrategia de búsqueda de parámetros e hiperparametros se realizó con el dataset training y se validó con el dataset de validation, los cuales se mencionaron anteriormente.

5.4. Optimización de modelos

Una vez definidos los mejores parámetros e hiperparametros para el modelo CNN propio y para los modelos por Transfer Learning, todos fueron compilados y entrenados utilizando los siguientes parámetros:

Tabla 5 Parámetros de entrenamiento modelos optimizados

Descripción	**Nombre del Parámetros**	**Parámetro usado**
Datos de entrada	'input_data'	Dataset Training
Datos en los que se evalúa el rendimiento del modelo al final de cada época	'validation_data'	Dataset Validation
N° Muestras a utilizar en cada iteración	'batch_size'	64
N° épocas o iteraciones para entrenar el modelo	'epochs'	100

6 Resultados

Al realizar la implementación de los diferentes modelos según la metodología descrita anteriormente, nos permitimos mostrar los principales resultados obtenidos para el modelo de construcción propia y para los modelos por Transfer Learning en las arquitecturas VGG16, VGG19 y MobileNetV2 usando los datasets de imágenes con y sin aumento de datos:

6.1. Modelo convolucional construcción propia

6.1.1. Búsqueda de parámetros e hiperparametros

Los resultados de la búsqueda de mejores parámetros e hiperparametros que se realizó de forma secuencial por medio de HalvingGridSearch se muestra a continuación con la Tabla 6.

Tabla 6 Búsqueda de parámetros e hiperparametros Modelo CNN (propio)

Búsqueda	Mejor resultado encontrado
Numero de capas (Layers)	Mejor Número de layers: 6 3/3 - 0s - 119ms/epoch - 40ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8157894736842105
Search filters Layers #1	Mejor Número de Filtros: 128 3/3 - 0s - 129ms/epoch - 43ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8947368421052632
Search filters Layers #2	Mejor Número de Filtros: 256 3/3 - 0s - 143ms/epoch - 48ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8026315789473685
Search filters Layers #3	Mejor Número de Filtros: 512 3/3 - 0s - 153ms/epoch - 51ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8421052631578947
Search filters Layers #4	Mejor Número de Filtros: 64 3/3 - 0s - 165ms/epoch - 55ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.881578947368421
Search filters Layers #5	Mejor Número de Filtros: 256 3/3 - 0s - 160ms/epoch - 53ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8421052631578947
Search filters Layers #6	Mejor Número de Filtros: 512 3/3 - 0s - 164ms/epoch - 55ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.881578947368421
Dropout Rate	Mejor Tasa de Dropout: 0.7 3/3 - 0s - 122ms/epoch - 41ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8289473684210527
Dense Units	Mejor Número de Unidades en la Capa Densa: 1024 3/3 - 0s - 40ms/epoch - 13ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.881578947368421
Kernel Size	Mejor kernel_size: (3, 3) 3/3 - 0s - 129ms/epoch - 43ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8289473684210527
Pool Size	Mejor pool size: (2, 2) 3/3 - 0s - 121ms/epoch - 40ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.868421052631579
Learning Rate	Mejor learning rate: 0.001 3/3 - 0s - 42ms/epoch - 14ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8289473684210527

6.1.2. Construcción del modelo

Finalizada la búsqueda de parámetros e hiperparametros usando la técnica HalvingGridSearch en el modelo propio para el numero de capaz convolucionales y pooling,

numero de filters Layers en cada capa convolucional, unidades en capa fully connected, kernel size, pool size, learning rate y se realizó la construcción de este, el cual se puede visualizar en el Gráfico 41 en donde se muestra el código del modelo y su arquitectura.

Gráfico 33 Construcción del Modelo CNN (propio)

a) Construcción del Modelo

```
from keras.src.engine import input_layer
from keras.src import layers
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
from keras.models import load_model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout, AveragePooling2D
from sklearn.model_selection import train_test_split
from tensorflow.keras.regularizers import l1, l2
from tensorflow.keras.optimizers import Nadam

modelo_CNN = keras.Sequential([
    keras.layers.Conv2D(filters=128, kernel_size=3, activation='relu', input_shape=[224,224,3]),
    keras.layers.MaxPool2D(pool_size=2, strides=2),
    # Layer #2
    keras.layers.Conv2D(filters=256, kernel_size=3, activation='relu'),
    keras.layers.MaxPool2D(pool_size=2, strides=2),
    # Layer #3
    keras.layers.Conv2D(filters=512, kernel_size=3, activation='relu'),
    keras.layers.MaxPool2D(pool_size=2, strides=2),
    # Layer #4
    keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu'),
    keras.layers.MaxPool2D(pool_size=2, strides=2),
    # Layer #5
    keras.layers.Conv2D(filters=256, kernel_size=3, activation='relu'),
    keras.layers.MaxPool2D(pool_size=2, strides=2),
    # Layer #6
    keras.layers.Conv2D(filters=512, kernel_size=3, activation='relu'),
    keras.layers.MaxPool2D(pool_size=2, strides=2),

    keras.layers.Dropout(0.7),
    keras.layers.Flatten(),
    keras.layers.Dense(1024, activation='relu',kernel_regularizer=l1(0.01)),
    keras.layers.Dense(1, activation='sigmoid')
])
modelo_CNN.summary()
tf.keras.utils.plot_model(modelo_CNN)

learning_rate = 0.001
optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
modelo_CNN.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = ['accuracy'])

history_CNN = modelo_CNN.fit(train_validation_set,validation_data=test_set, epochs = 100, batch_size=64)
```

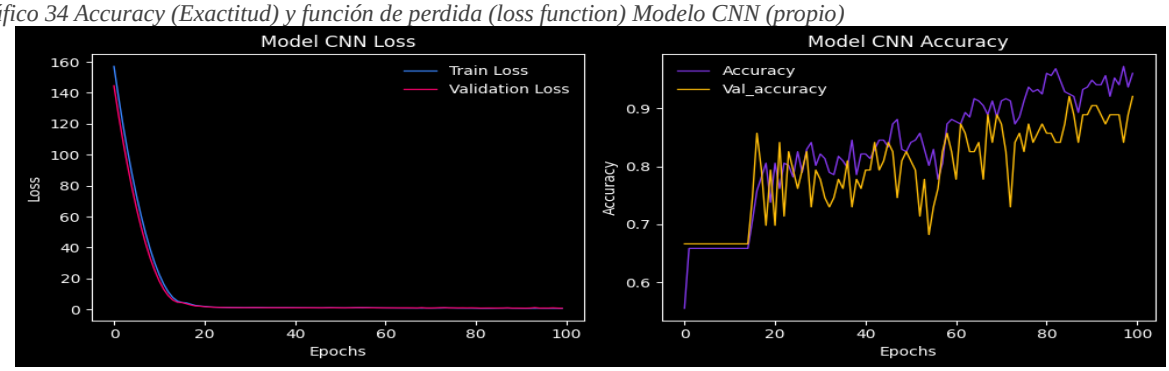
b) Arquitectura del Modelo

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 128)     3584
 max_pooling2d (MaxPooling2  (None, 111, 111, 128)     0
 D)
 conv2d_1 (Conv2D)           (None, 109, 109, 256)     295168
 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 256)       0
 g2D)
conv2d_2 (Conv2D)            (None, 52, 52, 512)       1180160
max_pooling2d_2 (MaxPoolin  (None, 26, 26, 512)       0
 g2D)
conv2d_3 (Conv2D)            (None, 24, 24, 64)        294976
 max_pooling2d_3 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)
conv2d_4 (Conv2D)            (None, 10, 10, 256)       147712
max_pooling2d_4 (MaxPoolin  (None, 5, 5, 256)         0
 g2D)
conv2d_5 (Conv2D)            (None, 3, 3, 512)         1180160
max_pooling2d_5 (MaxPoolin  (None, 1, 1, 512)         0
 g2D)
dropout (Dropout)            (None, 1, 1, 512)         0
flatten (Flatten)            (None, 512)               0
dense (Dense)                (None, 1024)              525312
dense_1 (Dense)              (None, 1)                 1025
=================================================================
Total params: 3628097 (13.84 MB)
Trainable params: 3628097 (13.84 MB)
Non-trainable params: 0 (0.00 Byte)
```

6.1.3. Evaluación del modelo

En las gráficas de la función de perdida (loss) y la precisión (Accuracy), observamos una disminución constante de la perdida a medida que avanza el entrenamiento, indicando que el modelo está aprendiendo y ajustando los pesos de maneja efectiva y no hay indicación de overfitting ni underfitting. Por su parte, el Accuracy presenta un aumento constante en la precisión a medida que el modelo se entrena, a pesar de que en las primeras épocas se observa un estancamiento. En términos generales, se observa convergencia en las dos medidas mostrando cierta estabilización. (Gráfico 42)

Gráfico 34 Accuracy (Exactitud) y función de perdida (loss function) Modelo CNN (propio)



Respecto a las métricas del modelo sin aumento de datos podemos mencionar:

La curva ROC se acerca a la esquina superior izquierda, indicando un buen rendimiento, ya que la tasa de verdaderos positivos es alta y la tasa de falsos positivos es baja. Mientras el AUC muestra un valor de 0.9, indicando que el modelo tiene una buena capacidad para distinguir entre clases. (Gráfico 43) El Accuracy presenta un valor de 92%, lo cual indica un buen rendimiento del modelo de clasificación con alta proporción de predicciones de forma correcta al comparar los datos de entrenamiento y validacion. Por su parte, el Recall o Sensibilidad presenta un valor del 86% indicando que una buena proporción de casos son identificados correctamente como positivo o con parasitos de leishmaniasis. En contraste, la especificidad presenta un valor del 95%, indicando que el modelo es capaz de clasificar eficientemente todas las instancias negativas o imágenes sin parasitos. La precisión y el VPP muestra un valor del 90% indicando que la proporción de imágenes con parasitos con leishmaniasis predichas se está realizando correctamente en 9 de cada 10 casos. Mientras que el VPN alcanza un 93%, indicando que el modelo clasifica eficientemente la mayoría de las imágenes sin parasitos. Finalmente, la métrica de F1-Score alcanzo un valor del 88%, revelando un buen equilibrio entre la precisión y Recall. (Tabla 7)

Gráfico 35Matriz de confusión y Curva ROC (AUC) Modelo CNN (propio)

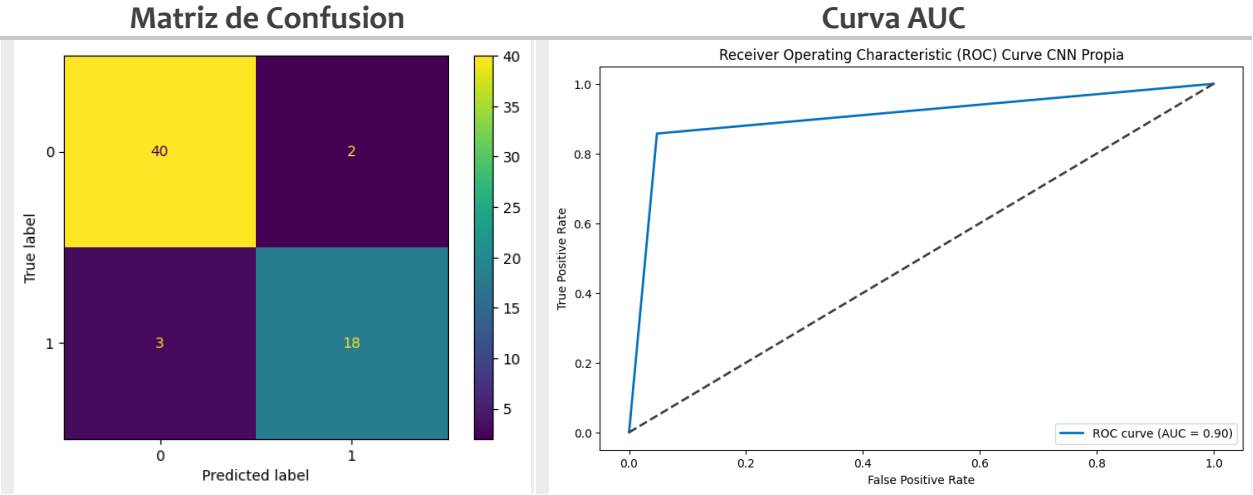


Tabla 7 Métricas evaluación rendimiento Modelo CNN (propio)

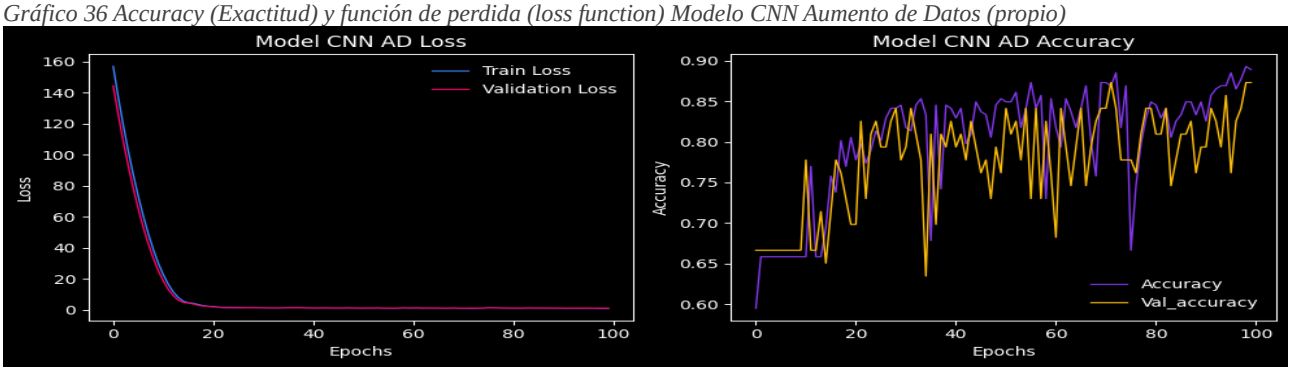
	Exactitud (Accuracy)	***Sensibilidad (Recall)***	***Especificidad***	***Precisión***	***F1-Score***	***VPP***	***VPN***
General	0.92	0.86	0.95	0.9	0.88	0.9	0.93
0.0 *(Con parásitos)*	-	0.9524	-	0.9302	0.9412	-	-
1.0 *(Sin parásitos)*	-	0.8571	-	0.900	0.8780	-	-
Macro avg	-	0.9048	-	0.9151	0.9096	-	-
Weighted avg	-	0.9206	-	0.9202	0.9201	-	-

6.1.4. Evaluación del modelo con aumento de datos

En las gráficas de la función de perdida (loss) y la precisión (Accuracy), observamos una

disminución constante de la perdida a medida que avanza el entrenamiento, indicando que el modelo está aprendiendo y ajustando los pesos de maneja efectiva y no hay indicación de overfitting ni underfitting. Por su parte, el Accuracy presenta un aumento con grandes variaciones o caídas en la precisión a medida que el modelo se entrena, a pesar de que en las primeras épocas se observa un estancamiento. En términos generales, se observa convergencia en las dos medidas al finalizar el entrenamiento en la época 100 mostrando cierta estabilización. (Gráfico 44)

Gráfico 36 Accuracy (Exactitud) y función de perdida (loss function) Modelo CNN Aumento de Datos (propio)



En relación con las métricas del modelo con aumento de datos podemos mencionar:

La curva ROC se acerca a la esquina superior izquierda, indicando un buen rendimiento, ya que la tasa de verdaderos positivos es alta y la tasa de falsos positivos es baja. Mientras el AUC muestra un valor de 0.85, indicando que el modelo tiene una buena capacidad para distinguir entre clases. (Gráfico 45) El Accuracy presenta un valor de 87,3%, lo cual indica un buen rendimiento del modelo de clasificación con alta proporción de predicciones de forma correcta al comparar los datos de entrenamiento y validacion. Por su parte, el Recall o Sensibilidad presenta un valor del 76%, valor más bajo si se compara con el resultado del modelo anterior sin aumento de datos. En contraste, la especificidad presenta un valor del 93%, indicando que el modelo es capaz de clasificar eficientemente todas las instancias negativas o imágenes sin parasitos. La precisión y el VPP muestra un valor del 84% indicando que la proporción de imágenes con parasitos con leishmaniasis predichas se está realizando correctamente en 8 de cada 10 casos. Mientras que el VPN alcanza un 89%, indicando que el modelo clasifica eficientemente la mayoría de las imágenes sin parasitos. Finalmente, la métrica de F1-Score alcanzo un valor del 80%, revelando un buen equilibrio entre la precisión y Recall, aunque más bajo que el modelo sin aumento de datos. (Tabla 8)

Gráfico 37 Matriz de confusión y Curva ROC (AUC) Modelo CNN Aumento de Datos (propio)

Matriz de Confusion	Curva AUC

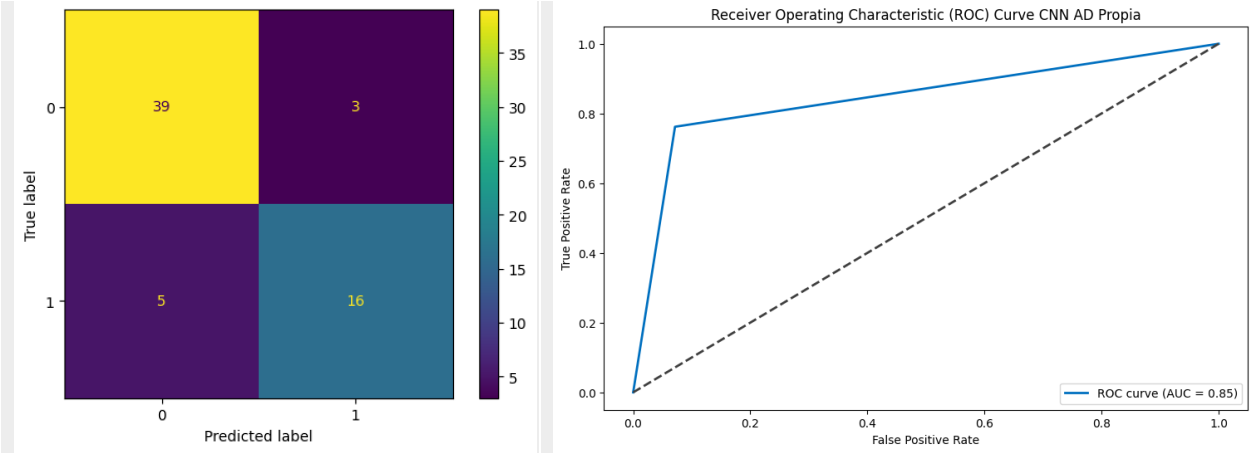


Tabla 8 Métricas evaluación rendimiento Modelo CNN Aumento de Datos (propio)

	Exactitud (Accuracy)	***Sensibilidad (Recall)***	***Especificidad***	***Precisión***	***F1-Score***	***VPP***	***VPN***
General	0.8730	0.76	0.93	0.84	0.8	0.84	0.89
0.0 *(Con parásitos)*	-	0.9286	-	0.8864	0.9070	-	-
1.0 *(Sin parásitos)*	-	0.7619	-	0.8421	0.800	-	-
Macro avg	-	0.8452	-	0.8642	0.8535	-	-
Weighted avg	-	0.8730	-	0.8716	0.8713	-	-

6.2. Modelo arquitectura VGG16

6.2.1. Búsqueda de parámetros e hiperparametros

Los resultados de la búsqueda de mejores parámetros e hiperparametros que se realizó de forma secuencial por medio de GridSearch se muestra a continuación con la Tabla 9.

Tabla 9 Búsqueda de parámetros e hiperparametros Modelo VGG16

Búsqueda	**Mejor resultado encontrado**

Dense Units	`Mejor Número de Unidades en la Capa Densa: 128`
	`2/2 [==============================] - 0s 247ms/step`
	`Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8026315789473685`
	`Mejor: 0.812500 usando {'dense': 128}`

Dropout Rate	`Mejor Tasa de Dropout: 0.3`
	`2/2 [==============================] - 0s 249ms/step`
	`Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.881578947368421`
	`Mejor: 0.818182 usando {'dropout': 0.3}`

Learning Rate	`Mejor learning rate: 0.0001`
	`2/2 [==============================] - 0s 248ms/step`
	`Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.868421052631579`
	`Mejor: 0.823864 usando {'learning_rate': 0.0001}`

6.2.2. Construcción del modelo

Finalizada la búsqueda de parámetros e hiperparametros usando la técnica GridSearch en el modelo de arquitectura VGG16 para el número de unidades en capa fully connected, Dropout rate y learning rate, se realizó la construcción de este, el cual se puede visualizar en el Gráfico 46 en donde se muestra el código del modelo y su arquitectura.

Gráfico 38 Construcción del Modelo VGG16

a) Construcción del Modelo

```
from keras.src.layers.attention.multi_head_attention import activation
from keras.src.engine.sequential import Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG16
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout, AveragePooling2D
from tensorflow.keras.regularizers import l1, l2
from tensorflow import keras


vgg16 = VGG16(weights ='imagenet',
              include_top = False,
              input_shape=(224,224,3))

model_vgg16 = Sequential()
model_vgg16.add(vgg16)
model_vgg16.add(Dropout(0.3))
model_vgg16.add(Flatten()) # Aplanar la salida de la capa convolucional
model_vgg16.add(Dense(128, activation = 'relu', kernel_regularizer=l2(0.01)))
model_vgg16.add(Dense(1,activation = 'sigmoid'))
vgg16.trainable = False

print(vgg16.summary())
print("*****************************************************************")
print(model_vgg16.summary())
```

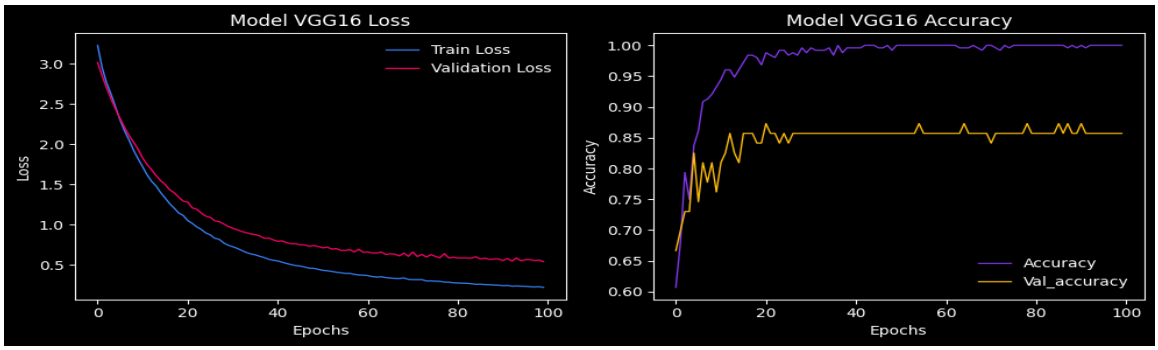
b) Arquitectura del Modelo

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0
=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
None
*****************************************************************
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688
 dropout_2 (Dropout)         (None, 7, 7, 512)         0
 flatten_2 (Flatten)         (None, 25088)             0
 dense_4 (Dense)             (None, 128)               3211392
 dense_5 (Dense)             (None, 1)                 129
=================================================================
Total params: 17926209 (68.38 MB)
Trainable params: 3211521 (12.25 MB)
Non-trainable params: 14714688 (56.13 MB)
```

6.2.3. Evaluación del modelo

En las gráficas de la función de perdida (loss) y la precisión (Accuracy), observamos una disminución constante de la perdida a medida que avanza el entrenamiento, indicando que el modelo está aprendiendo y ajustando los pesos de maneja efectiva y no hay indicación de overfitting ni underfitting, aunque se observa una divergencia entre la curva de train y validation. Por su parte, el Accuracy presenta un aumento constante en la precisión en las primeras épocas del modelo, para luego estabilizarse después de la iteración o época 20. (Gráfico 47)

Gráfico 39 Accuracy (Exactitud) y función de perdida (loss function) Modelo VGG16



En relación con las métricas del modelo sin aumento de datos podemos mencionar:

La curva ROC se acerca a la esquina superior izquierda, indicando un buen rendimiento, ya que la tasa de verdaderos positivos es alta y la tasa de falsos positivos es de aproximadamente el 20%. La AUC muestra un valor de 0.85, indicando que el modelo tiene una buena capacidad para distinguir entre clases. (Gráfico 48) El Accuracy presenta un valor de 85,7%, lo cual indica un buen rendimiento del modelo de clasificación con alta proporción de predicciones de forma correcta al comparar los datos de entrenamiento y validacion. Por su parte, el Recall o Sensibilidad presenta un valor del 71%, el cual es bajo, si se compara con los modelos anteriores. En contraste, la especificidad presenta un valor del 93%, indicando que el modelo es capaz de clasificar eficientemente todas las instancias negativas o imágenes sin parasitos. La precisión y el VPP muestra un valor del 83% indicando que la proporción de imágenes con parasitos con leishmaniasis predichas se está realizando correctamente en 8 de cada 10 casos. Mientras que el VPN alcanza un 87%, indicando que el modelo clasifica eficientemente la mayoría de las imágenes sin parasitos. Finalmente, la métrica de F1-Score alcanzo un valor del 77%, revelando una disminución del equilibrio entre la precisión y Recall. (Tabla 10)

Gráfico 40Matriz de confusión y Curva ROC (AUC) Modelo VGG16

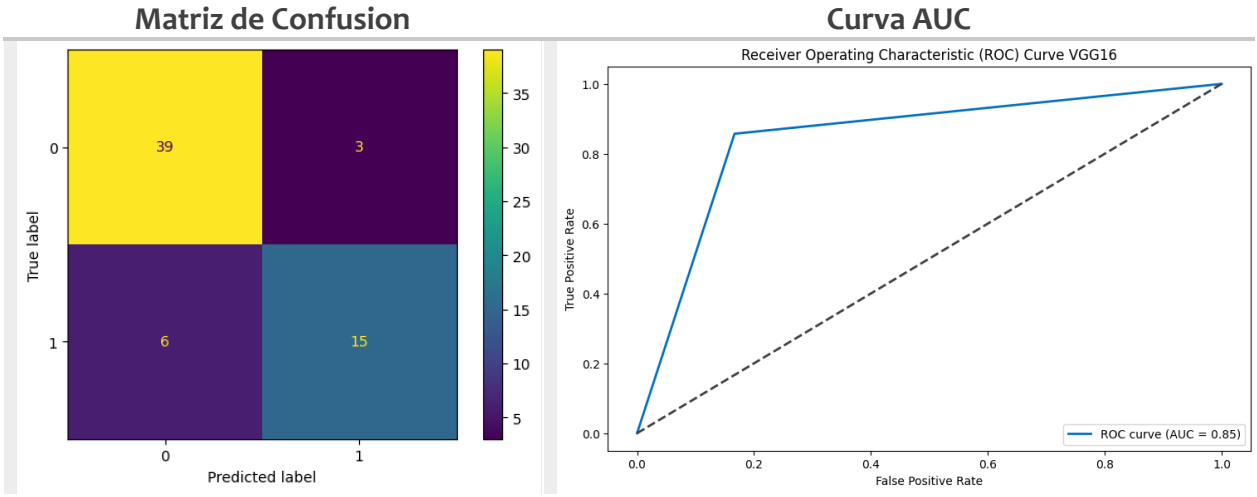


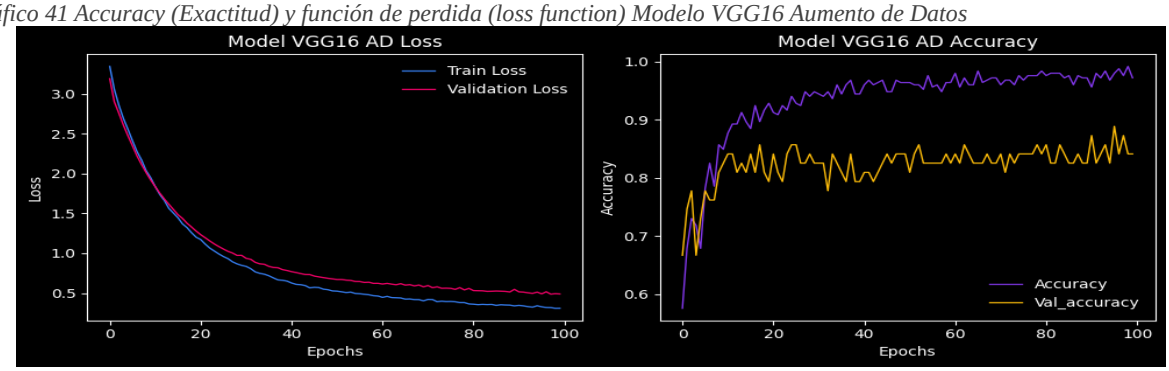
Tabla 10 Métricas evaluación rendimiento Modelo VGG16

	Exactitud (Accuracy)	*Sensibilidad (Recall)*	*Especificidad*	*Precisión*	*F1-Score*	*VPP*	*VPN*
General	0.8571	0.71	0.93	0.83	0.77	0.83	0.87
0.0 *(Con parásitos)*	-	0.9286	-	0.8667	0.8966	-	-
1.0 *(Sin parásitos)*	-	0.7143	-	0.8333	0.7692	-	-
Macro avg	-	0.8214	-	0.8500	0.8329	-	-
Weighted avg	-	0.8571	-	0.8556	0.8541	-	-

6.2.4. Evaluación del modelo con aumento de datos

En las gráficas de la función de perdida (loss) y la precisión (Accuracy), observamos una disminución constante de la perdida a medida que avanza el entrenamiento, indicando que el modelo está aprendiendo y ajustando los pesos de maneja efectiva y no hay indicación de overfitting ni underfitting, aunque nuevamente se observa una divergencia aunque un poco más pequeña entre la curva de train y validation. Por su parte, el Accuracy presenta un aumento constante en la precisión en las primeras épocas del modelo, para luego estabilizarse después de la iteración o época 20, mostrando un mejor comportamiento que el modelo sin aumento de datos. Gráfico 49

Gráfico 41 Accuracy (Exactitud) y función de perdida (loss function) Modelo VGG16 Aumento de Datos



En relación con las métricas del modelo sin aumento de datos podemos mencionar:

La curva ROC se acerca a la esquina superior izquierda, indicando un buen rendimiento, ya que la tasa de verdaderos positivos es alta y la tasa de falsos positivos es de aproximadamente el 20%. La AUC muestra un valor de 0.85, indicando que el modelo tiene una buena capacidad para distinguir entre clases. (Gráfico 50) El Accuracy presenta un valor de 84,1%, lo cual indica un buen rendimiento del modelo de clasificación con alta proporción de predicciones de forma correcta al comparar los datos de entrenamiento y validacion. Por su parte, el Recall o Sensibilidad presenta un mejor valor alcanzando 86%. En contraste, la especificidad presenta una disminución de 10 puntos porcentuales, logrando un 83%, indicando que el modelo es capaz de clasificar eficientemente 8 de cada 10 imágenes negativas o imágenes sin parasitos. La precisión y el VPP muestra un valor bajo cercano al 72% indicando que la proporción de imágenes con parasitos con leishmaniasis predichas se está realizando correctamente en 7 de cada 10 casos. De la misma forma, el VPN alcanza un 72%, indicando que el modelo no clasifica eficientemente la mayoría de las imágenes sin parasitos. Finalmente, la métrica de F1-Score

alcanzo un valor del 78%, revelando un comportamiento similar al modelo sin aumento de datos. (Tabla 11)

Gráfico 42Matriz de confusión y Curva ROC (AUC) Modelo VGG16 Aumento de Datos

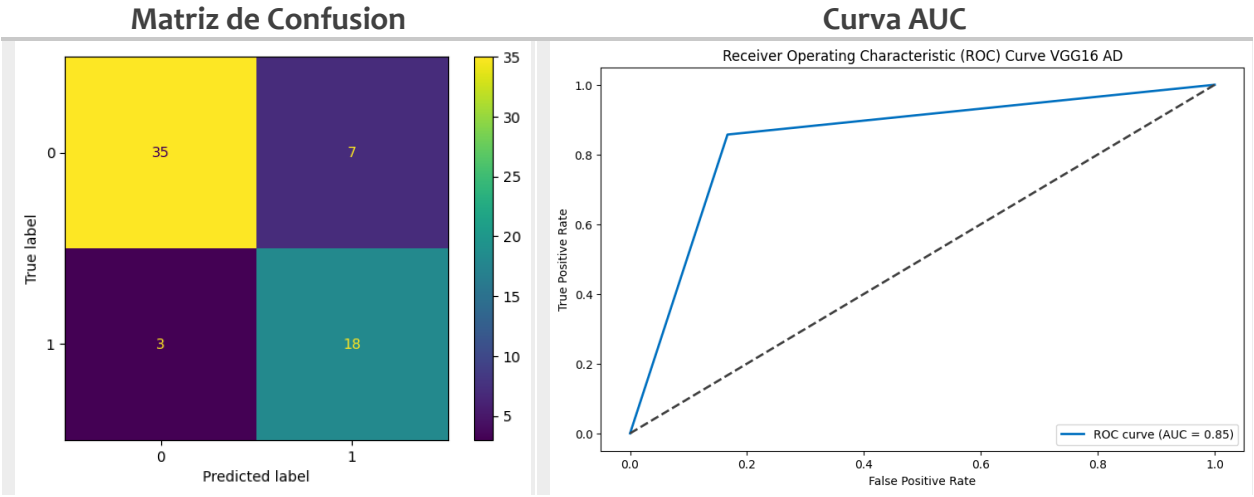


Tabla 11 Métricas evaluación rendimiento Modelo VGG16 Aumento de Datos

	Exactitud (Accuracy)	***Sensibilidad (Recall)***	***Especificidad***	***Precisión***	***F1-Score***	***VPP***	***VPN***
General	0.8413	0.86	0.83	0.72	0.78	0.72	0.92
0.0 *(Con parásitos)*	-	0.8333	-	0.9211	0.8750	-	-
1.0 *(Sin parásitos)*	-	0.8571	-	0.7200	0.7826	-	-
Macro avg	-	0.8452	-	0.8205	0.8288	-	-
Weighted avg	-	0.8413	-	0.8540	0.8442	-	-

6.3. Modelo arquitectura VGG19

6.3.1. Búsqueda de parámetros e hiperparametros

Los resultados de la búsqueda de mejores parámetros e hiperparametros que se realizó de forma secuencial por medio de GridSearch se muestra a continuación con la Tabla 12.

Tabla 12 Búsqueda de parámetros e hiperparametros Modelo VGG19

Búsqueda	**Mejor resultado encontrado**

Dense Units	`Mejor Número de Unidades en la Capa Densa: 512`
	`2/2 [==============================] - 1s 338ms/step`
	`Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8552631578947368`
	`Mejor: 0.767045 usando {'dense': 512}`

Dropout Rate	`Mejor Tasa de Dropout: 0.1`
	`2/2 [==============================] - 1s 324ms/step`
	`Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.8552631578947368`
	`Mejor: 0.829545 usando {'dropout': 0.1}`

Learning Rate	`Mejor learning rate: 0.0001`
	`2/2 [==============================] - 0s 294ms/step`
	`Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.868421052631579`
	`Mejor: 0.857955 usando {'learning_rate': 0.0001}`

6.3.2. Construcción del modelo

Finalizada la búsqueda de parámetros e hiperparametros usando la técnica GridSearch en el modelo de arquitectura VGG19 para el número de unidades en capa fully connected, Dropout rate y learning rate, se realizó la construcción de este, el cual se puede visualizar en el Gráfico 51 en donde se muestra el código del modelo y su arquitectura.

Gráfico 43 Construcción del Modelo VGG19

a) Construcción del Modelo

```
from keras.src.layers.attention.multi_head_attention import activation
from keras.src.engine.sequential import Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG19
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout, AveragePooling2D
from tensorflow.keras.regularizers import l1, l2
from tensorflow import keras

vgg19 = VGG19(weights ='imagenet',
              include_top = False,
              input_shape=(224,224,3))

model_vgg19 = Sequential()
model_vgg19.add(vgg19)
model_vgg19.add(Dropout(0.1))
model_vgg19.add(Flatten())
model_vgg19.add(Dense(512, activation = 'relu', kernel_regularizer=l2(0.01)))
model_vgg19.add(Dense(1,activation = 'sigmoid'))
vgg19.trainable = False
```

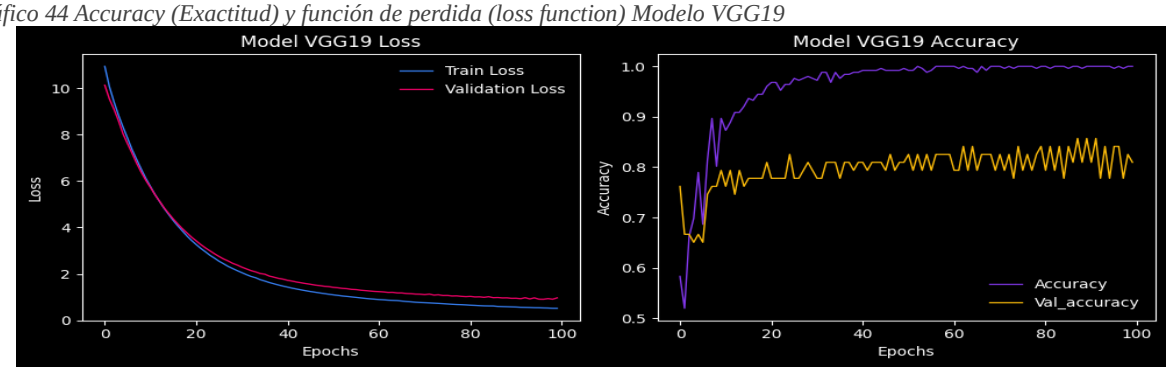
b) Arquitectura del Modelo

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_conv4 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv4 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0
=================================================================
Total params: 20024384 (76.39 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 20024384 (76.39 MB)
_________________________________________________________________
None
**********************************************************************
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 dropout_4 (Dropout)         (None, 7, 7, 512)         0

 flatten_4 (Flatten)         (None, 25088)             0

 dense_8 (Dense)             (None, 512)               12845568

 dense_9 (Dense)             (None, 1)                 513
=================================================================
Total params: 32870465 (125.39 MB)
Trainable params: 12846081 (49.00 MB)
Non-trainable params: 20024384 (76.39 MB)
```

6.3.3. Evaluación del modelo

En las gráficas de la función de perdida (loss) y la precisión (Accuracy), observamos una disminución constante de la perdida a medida que avanza el entrenamiento, indicando que el modelo está aprendiendo y ajustando los pesos de maneja efectiva y no hay indicación de overfitting ni underfitting, aunque se observa una divergencia entre la curva de train y validation similar a la observada en el modelo con arquitectura VGG16. Por su parte, el Accuracy presenta un aumento constante en la precisión en las primeras épocas del modelo, para luego estabilizarse después de la iteración o época 20. Mostrando un mejor resultado para la curva de entrenamiento vs la de validación. (Gráfico 52)

Gráfico 44 Accuracy (Exactitud) y función de perdida (loss function) Modelo VGG19



En relación con las métricas del modelo sin aumento de datos podemos mencionar:

La curva ROC se alejada a la esquina superior izquierda, indicando un rendimiento bajo, ya que la tasa de verdaderos positivos es baja y la tasa de falsos positivos es alta. La AUC muestra un valor de 0.73, indicando que el modelo tiene una baja capacidad para distinguir entre clases. (Gráfico 53) El Accuracy presenta un valor de 80,9%, lo cual indica un buen rendimiento del modelo de clasificación con alta proporción de predicciones de forma correcta al comparar los datos de entrenamiento y validacion. Por su parte, el Recall o Sensibilidad presenta un valor del 48%, el cual es muy bajo, si se compara con los modelos anteriores. En contraste, la especificidad presenta un valor del 98%, indicando que el modelo es capaz de clasificar eficientemente todas las instancias negativas o imágenes sin parasitos. La precisión y el VPP muestra un valor del 91% indicando que la proporción de imágenes con parasitos con leishmaniasis predichas se está realizando correctamente en 9 de cada 10 casos, aproximadamente. Mientras que el VPN alcanza un 79%, indicando que el modelo clasifica eficientemente la mayoría de las imágenes sin parasitos. Finalmente, la métrica de F1-Score alcanzo del 62%, revelando desequilibrio entre la precisión y Recall. (Tabla 13)

Gráfico 45Matriz de confusión y Curva ROC (AUC) Modelo VGG19

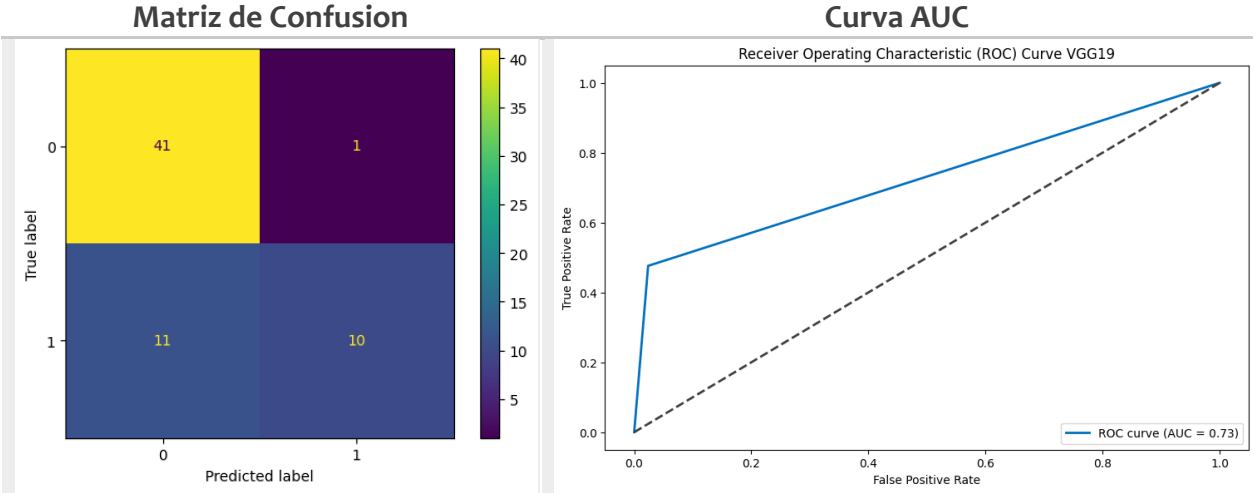


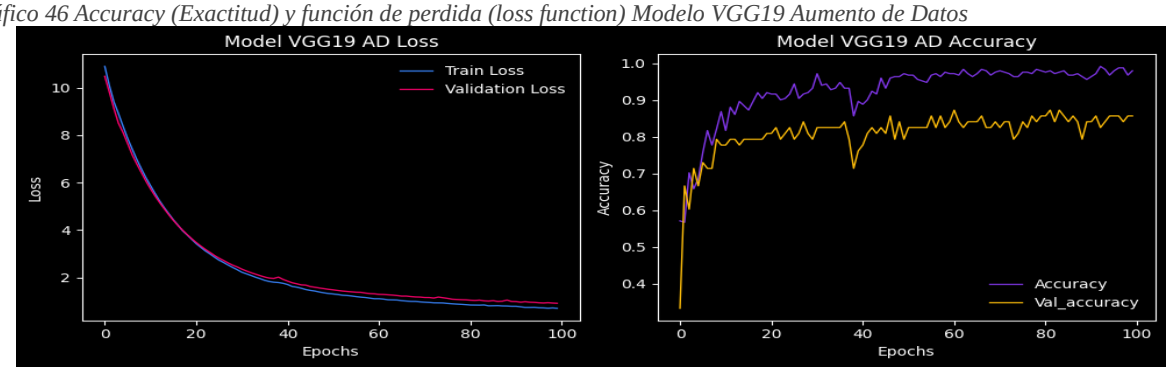
Tabla 13 Métricas evaluación rendimiento Modelo VGG19

	Exactitud (Accuracy)	***Sensibilidad (Recall)***	***Especificidad***	***Precisión***	***F1-Score***	***VPP***	***VPN***
General	0.8095	0.48	0.98	0.91	0.62	0.91	0.79
0.0 *(Con parásitos)*	-	0.9762	-	0.7885	0.8723	-	-
1.0 *(Sin parásitos)*	-	0.4762	-	0.9091	0.6250	-	-
Macro avg	-	0.7262	-	0.8488	0.7487	-	-
Weighted avg	-	0.8095	-	0.8287	0.7899	-	-

6.3.4. Evaluación del modelo con aumento de datos

En las gráficas de la función de perdida (loss) y la precisión (Accuracy), observamos una disminución constante de la perdida a medida que avanza el entrenamiento, indicando que el modelo está aprendiendo y ajustando los pesos de maneja efectiva y no hay indicación de overfitting ni underfitting. Siendo un mejor resultado que el observado en el modelo VGG16 y VGG19 sin aumento de datos. Por su parte, el Accuracy presenta un aumento constante en la precisión en las primeras épocas del modelo, para luego estabilizarse después de la iteración o época 20. (Gráfico 54)

Gráfico 46 Accuracy (Exactitud) y función de perdida (loss function) Modelo VGG19 Aumento de Datos



En relación con las métricas del modelo sin aumento de datos podemos mencionar:

La curva ROC se acerca a la esquina superior izquierda, indicando un buen rendimiento, ya que la tasa de verdaderos positivos es alta y la tasa de falsos positivos es de aproximadamente el 16%. La AUC muestra un valor de 0.86, indicando que el modelo tiene una buena capacidad para distinguir entre clases. (Gráfico 55) El Accuracy presenta un valor de 85,7%, lo cual indica un buen rendimiento del modelo de clasificación con alta proporción de predicciones de forma correcta al comparar los datos de entrenamiento y validacion. Por su parte, el Recall o Sensibilidad presenta un valor del 86%, el cual se incrementa nuevamente, si se compara con el modelo anterior sin aumento de datos. Asimismo, la especificidad presenta un valor del 86%, indicando que el modelo es capaz de clasificar eficientemente todas las instancias negativas o imágenes sin parasitos. La precisión y el VPP muestra un valor del 75% indicando que la proporción de imágenes con parasitos con leishmaniasis predichas se está realizando correctamente en aproximadamente 7 de cada 10 casos. Mientras que el VPN alcanza un 92%, indicando que el modelo clasifica eficientemente la mayoría de las imágenes sin parasitos. Finalmente, la métrica de F1-Score alcanzo un valor del 80%, revelando un mejor equilibrio

entre la precisión y Recall. (Tabla 14)

Gráfico 47Matriz de confusión y Curva ROC (AUC) Modelo VGG19 Aumento de Datos

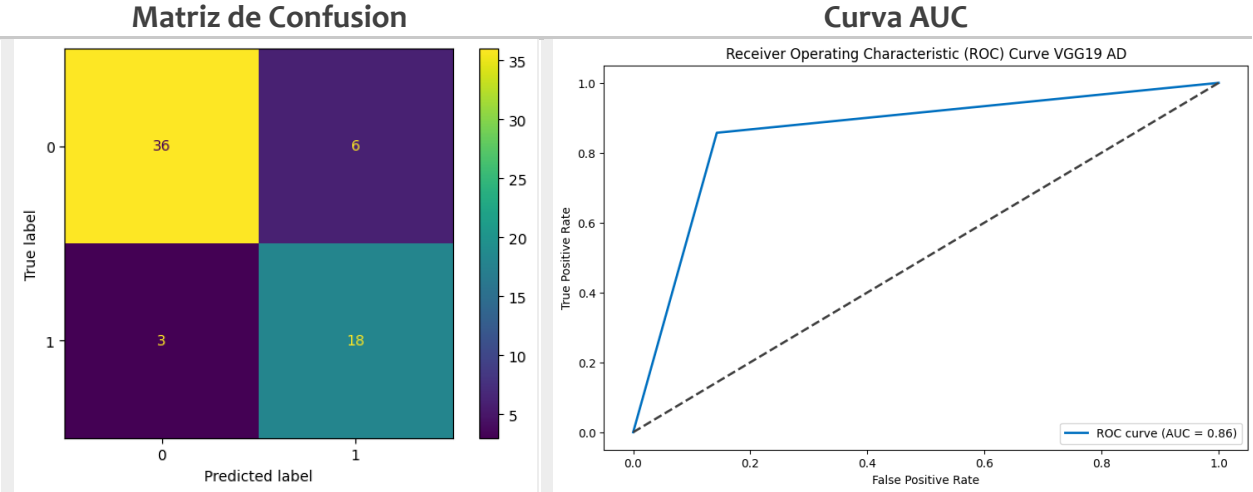


Tabla 14 Métricas evaluación rendimiento Modelo VGG19 Aumento de Datos

	Exactitud (Accuracy)	***Sensibilidad (Recall)***	***Especificidad***	***Precisión***	***F1-Score***	***VPP***	***VPN***
General	0.8571	0.86	0.86	0.75	0.8	0.75	0.92
0.0 *(Con parásitos)*	-	0.8571	-	0.9231	0.8889	-	-
1.0 *(Sin parásitos)*	-	0.8571	-	0.7500	0.8000	-	-
Macro avg	-	0.8571	-	0.8365	0.8444	-	-
Weighted avg	-	0.8571	-	0.8654	0.8593	-	-

6.4. Modelo arquitectura MobilenetV2

6.4.1. Búsqueda de parámetros e hiperparametros

Los resultados de la búsqueda de mejores parámetros e hiperparametros que se realizó de forma secuencial por medio de GridSearch se muestra a continuación con la Tabla 15.

Tabla 15 Búsqueda de parámetros e hiperparametros Modelo Mobilenet_v2

Búsqueda	**Mejor resultado encontrado**
Dense Units	Mejor Número de Unidades en la Capa Densa: 128 2/2 [==============================] - 1s 77ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.9473684210526315 Mejor: 0.914773 usando {'dense': 128}
Dropout Rate	Mejor Tasa de Dropout: 0.3 2/2 [==============================] - 1s 78ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.9210526315789473 Mejor: 0.909091 usando {'dropout': 0.3}
Learning Rate	Mejor learning rate: 0.001 2/2 [==============================] - 1s 78ms/step Exactitud del Mejor Modelo en el Conjunto de Prueba: 0.9473684210526315 Mejor: 0.903409 usando {'learning_rate': 0.001}

6.4.2. Construcción del modelo

Finalizada la búsqueda de parámetros e hiperparametros usando la técnica GridSearch en el modelo de arquitectura MobileNetV2 para el número de unidades en capa fully connected, Dropout rate y learning rate, se realizó la construcción de este, el cual se puede visualizar en el Gráfico 56 en donde se muestra el código del modelo y su arquitectura.

Gráfico 48 Construcción del Modelo Mobilenet_v2

a) Construcción del Modelo

```
from keras.src.applications import mobilenet_v2
import tensorflow as tf
import tensorflow_hub as hub
from keras.src.layers.attention.multi_head_attention import activation
from keras.src.engine.sequential import Sequential
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout, AveragePooling2D
from tensorflow.keras.regularizers import l1, l2
from tensorflow import keras
url = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4'
mobilenet_v2 = hub.KerasLayer(url, input_shape=(224,224,3))
mobilenet_v2.trainable = False
modelo_mobilenetv2 = Sequential()
modelo_mobilenetv2.add(mobilenet_v2)
modelo_mobilenetv2.add(layers.Dropout(0.3))
modelo_mobilenetv2.add(layers.Flatten())
modelo_mobilenetv2.add(layers.Dense(128, activation = 'relu', kernel_regularizer=l2(0.01)))
modelo_mobilenetv2.add(layers.Dense(1, activation='sigmoid'))
```

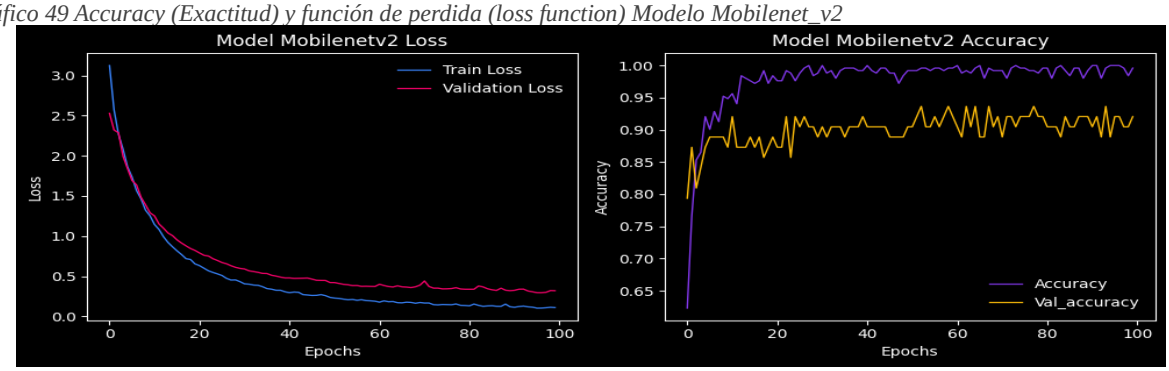
b) Arquitectura del Modelo

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              2257984
 dropout_6 (Dropout)         (None, 1280)              0
 flatten_6 (Flatten)         (None, 1280)              0
 dense_12 (Dense)            (None, 128)               163968
 dense_13 (Dense)            (None, 1)                 129
=================================================================
Total params: 2422081 (9.24 MB)
Trainable params: 164097 (641.00 KB)
Non-trainable params: 2257984 (8.61 MB)
```

6.4.3. Evaluación del modelo

En las gráficas de la función de perdida (loss) y la precisión (Accuracy), observamos una disminución constante de la perdida a medida que avanza el entrenamiento, indicando que el modelo está aprendiendo y ajustando los pesos de maneja efectiva y no hay indicación de overfitting ni underfitting, aunque se observa una divergencia entre la curva de train y validation. Por su parte, el Accuracy presenta un aumento constante en la precisión en las primeras épocas del modelo, para luego estabilizarse después de la iteración o época 20 con mayores valores en la curva de Accuracy si se compara con la curva de validation Accuracy. (Gráfico 57)

Gráfico 49 Accuracy (Exactitud) y función de perdida (loss function) Modelo Mobilenet_v2



En relación con las métricas del modelo sin aumento de datos podemos mencionar:

La curva ROC se acerca a la esquina superior izquierda, indicando un buen rendimiento, ya que la tasa de verdaderos positivos es alta y la tasa de falsos positivos. La AUC muestra un valor de

0.89, indicando que el modelo tiene una buena capacidad para distinguir entre clases. (Gráfico 58) El Accuracy presenta un valor de 92%, lo cual indica un buen rendimiento del modelo de clasificación con alta proporción de predicciones de forma correcta al comparar los datos de entrenamiento y validacion. Por su parte, el Recall o Sensibilidad presenta un valor del 81%, el cual bueno, si se compara con los modelos anteriores. En contraste, la especificidad presenta un valor del 98%, indicando que el modelo es capaz de clasificar eficientemente todas las instancias negativas o imágenes sin parasitos. La precisión y el VPP muestra un valor del 94% indicando que la proporción de imágenes con parasitos con leishmaniasis predichas se está realizando correctamente en 9 de cada 10 casos. Mientras que el VPN alcanza un 91%, indicando que el modelo clasifica eficientemente la mayoría de las imágenes sin parasitos. Finalmente, la métrica de F1-Score alcanzo un valor del 87%, revelando un incremento del equilibrio entre la precisión y Recall. (Tabla 16)

Gráfico 50Matriz de confusión y Curva ROC (AUC) Modelo Mobilenet_v2

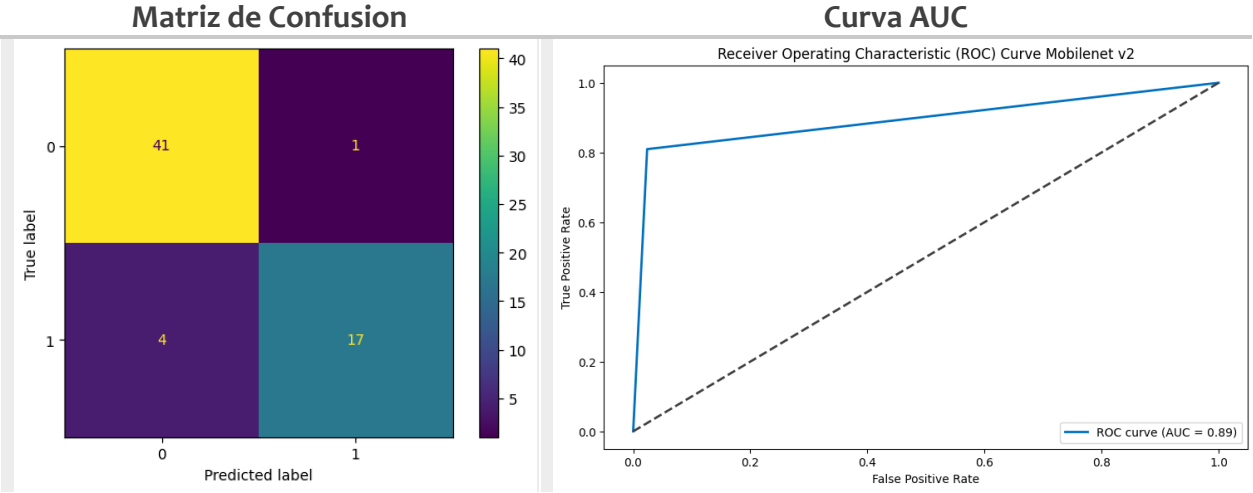


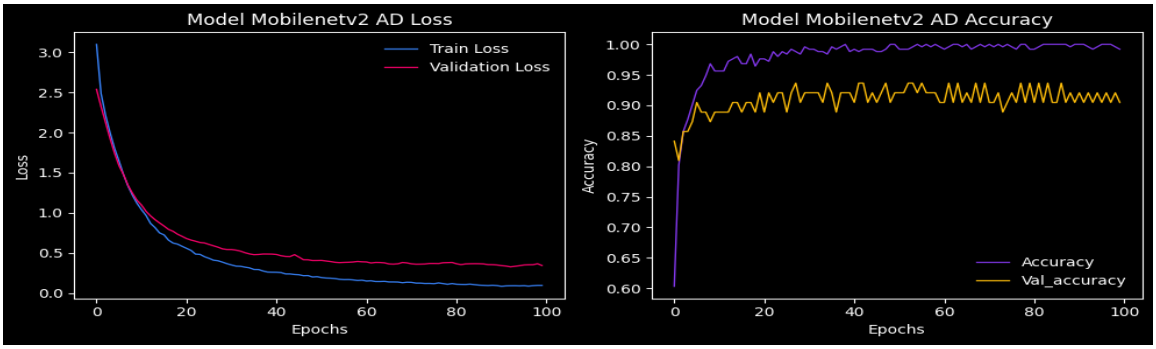
Tabla 16 Métricas evaluación rendimiento Modelo Mobilenet_v2

	Exactitud (Accuracy)	***Sensibilidad (Recall)***	***Especificidad***	***Precisión***	***F1-Score***	***VPP***	***VPN***
General	0.9206	0.81	0.98	0.94	0.87	0.94	0.91
0.0 *(Con parásitos)*	-	0.9762	-	0.9111	0.9425	-	-
1.0 *(Sin parásitos)*	-	0.8095	-	0.9444	0.8718	-	-
Macro avg	-	0.8929	-	0.9278	0.9072	-	-
Weighted avg	-	0.9206	-	0.9222	0.9190	-	-

6.4.4. Evaluación del modelo con aumento de datos

En las gráficas de la función de perdida (loss) y la precisión (Accuracy), observamos una disminución constante de la perdida a medida que avanza el entrenamiento, indicando que el modelo está aprendiendo y ajustando los pesos de maneja efectiva y no hay indicación de overfitting ni underfitting, aunque se observa una divergencia entre la curva de train y validation. Por su parte, el Accuracy presenta un aumento constante en la precisión en las primeras épocas del modelo, para luego estabilizarse y una variación más estable después de la iteración o época 20. (Gráfico 59)

Gráfico 51 Accuracy (Exactitud) y función de perdida (loss function) Modelo Mobilenet_v2 Aumento de Datos



En relación con las métricas del modelo sin aumento de datos podemos mencionar:

La curva ROC se acerca a la esquina superior izquierda, indicando un buen rendimiento, ya que la tasa de verdaderos positivos es alta y la tasa de falsos positivos es de aproximadamente el 10%. La AUC muestra un valor de 0.90, indicando que el modelo tiene una buena capacidad para distinguir entre clases. (Gráfico 60) El Accuracy presenta un valor de 90,5%, lo cual indica un buen rendimiento del modelo de clasificación con alta proporción de predicciones de forma correcta al comparar los datos de entrenamiento y validacion. Por su parte, el Recall o Sensibilidad presenta un valor del 90%, el cual es alto, si se compara con los modelos anteriores. En contraste, la especificidad presenta un valor del 90%, indicando que el modelo es capaz de clasificar eficientemente todas las instancias negativas o imágenes sin parasitos. La precisión y el VPP muestra un valor del 83% indicando que la proporción de imágenes con parasitos con leishmaniasis predichas se está realizando correctamente en 8 de cada 10 casos. Mientras que el VPN alcanza un 95%, indicando que el modelo clasifica eficientemente la mayoría de las imágenes sin parasitos. Finalmente, la métrica de F1-Score alcanzo un valor del 86%, revelando una disminución del equilibrio entre la precisión y Recall. (Tabla 17)

Gráfico 52Matriz de confusión y Curva ROC (AUC) Modelo Mobilenet_v2 Aumento de Datos

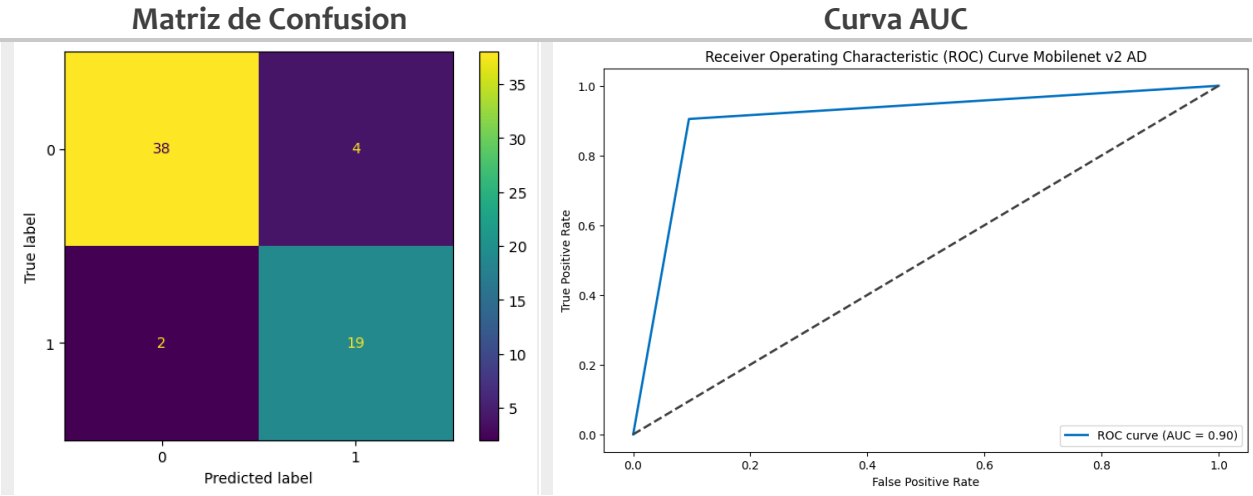


Tabla 17 Métricas evaluación rendimiento Modelo Mobilenet_v2 Aumento de Datos

	Exactitud (Accuracy)	***Sensibilidad (Recall)***	***Especificidad***	***Precisión***	***F1-Score***	***VPP***	***VPN***
General	0.9048	0.9	0.9	0.83	0.86	0.83	0.95
0.0 *(Con parásitos)*	-	0.9048	-	0.9500	0.9268	-	-
1.0 *(Sin parásitos)*	-	0.9048	-	0.8261	0.8636	-	-
Macro avg	-	0.9048	-	0.8880	0.8952	-	-
Weighted avg	-	0.9048	-	0.9087	0.9058	-	-

7 Conclusiones y Trabajos Futuros

7.1. Conclusiones

Después de finalizar nuestro proyecto de investigación, en donde se construyeron diferentes modelos, haciendo uso de diferentes técnicas de Machine Learning, podemos concluir lo siguiente:

- MobileNetV2 fue la mejor arquitectura para realizar la clasificación de imágenes de placas con cultivos de leishmaniasis tanto para el modelo sin y con aumento de datos. Ambos modelos alcanzaron una exactitud, especificidad y VPN por encima del 90%. Con curvas ROC entre el .89 y 0.9. Asimismo, la sensibilidad, precisión, F1-Score y VPP alcanzaron valores del 81%, 94%, 87% y 91%, respectivamente en el modelo sin aumento de datos. Mientras que para el modelo con aumento de datos la sensibilidad fue del 90%, la precisión del 83%, F1-score del 86% y VPP del 83%.

- El modelo de construcción propia presento un mejor desempeño sin datos aumentados con resultados en todas las métricas por encima del 85% y una curva ROC del 0.9. Sin embargo, al analizar los resultados del modelo aumentado, el valor de la curva ROC cae al 0.85 al igual que la sensibilidad al 76%. Las demás métricas presentan una caída hasta de los 8 puntos porcentuales.

- El uso de técnicas de búsqueda y optimización de hiperparametros como GridSearch y HalvingGridSearch, a pesar de ser técnicas exhaustivas y costosas computacionalmente. Son excelentes herramientas para definir los mejores parámetros para nuestros modelos, al permitir una búsqueda eficiente, secuencial y reproducible, eliminando la subjetividad e intuición en la búsqueda del mejor modelo.

- A pesar de que se intentó hacer uso de EarlyStopping en los modelos, especialmente en el modelo propio. No fue posible su implementación debida a que esta técnica monitorea el rendimiento del modelo en un conjunto de validación y detiene el entrenamiento cuando el rendimiento deja de mejorar o comienza a empeorar. Y en nuestro caso, los modelos se estabilizaban después de 50 épocas o más.

- El modelo con arquitectura VGG19 sin aumento de datos fue el que peor desempeño presento. Su sensibilidad no supero el 50%, el F1-score fue del 62%, el VPN alcanzo el 79% y un valor de curva ROC del 0,73. No obstante, Este modelo presento una alta especificidad del 98% y una precisión y VPP del 91%.

- Las arquitecturas VGG16 y VGG19 con aumento de datos presentaron un comportamiento similar en sus métricas con un valor de curva ROC que se encontraba en el rango del 0.85 a 0.86. Y Resultados de Accuracy, Sensibilidad, Especificidad, Precisión, F1-Score, VPP y VPN por encima del 72%.

7.2. Trabajos Futuros

Posterior al desarrollo de este trabajo de investigación, realizamos las siguientes recomendaciones para futuros proyectos asociados a la identificación de parásitos de leishmaniasis en imágenes de placas de cultivos de laboratorio:

- La primera recomendación hace referencia a la disponibilidad de imágenes con y sin parásitos de cultivos de laboratorio de leishmaniasis, las cuales permitirán hacer un mejor entrenamiento de los respectivos modelos de identificación al contar con dataset más grandes.

- Los resultados de las métricas de los diferentes modelos pueden ser mejorados si se explora un mayor número de hiperparametros con mayores recursos computacionales en el modelo propio. Igualmente es posible mejorar los modelos construidos por Transfer Learning usando la técnica de transferencia de aprendizaje basada en modelo (Model-based Transfer Learning) la cual no solo entrena la capa de salida del modelo sino que también ajusta los pesos de algunas capas del modelo pre-entrenado.

- En la misma línea de la recomendación anterior, sugerimos mejorar la calidad de las imágenes entregadas (con y sin parásitos) debido a que no son uniformes y de la misma calidad. En la mayoría de las imágenes se identificó presencia de mucho background o fondo lo cual puede afectar el entrenamiento de los modelos y presencia de guía ocular o puntero indicador del microscopio lo cual puede afectar en gran medida la identificación de patrones relevantes.

- Garantizar una comunicación adecuada y constante entre el equipo investigador y el centro responsable de la información con el fin de facilitar la retroalimentación, aclaración de conceptos, obtención de mejores imágenes además de una adecuada sinergia en el desarrollo de este tipo de proyectos.

8 Referencias Bibliográficas

[1] «Enfermedades transmitidas por vectores». Accedido: 22 de noviembre de 2022. [En línea]. Disponible en: https://www.who.int/es/news-room/fact-sheets/detail/vector-borne-diseases

[2] M. L. O. Martínez y F. E. P. Alvarado, «Protocolo de vigilancia de Leishmaniasis», p. 28.

[3] «Leishmaniasis». Accedido: 22 de noviembre de 2022. [En línea]. Disponible en: https://www.who.int/es/news-room/fact-sheets/detail/leishmaniasis

[4] «Leishmaniasis - OPS/OMS | Organización Panamericana de la Salud». Accedido: 22 de noviembre de 2022. [En línea]. Disponible en: https://www.paho.org/es/temas/leishmaniasis

[5] «Atlas interactivo de leishmaniasis en las Américas: aspectos clínicos y diagnósticos diferenciales - OPS/OMS | Organización Panamericana de la Salud». Accedido: 23 de noviembre de 2022. [En línea]. Disponible en: https://www.paho.org/es/documentos/atlas-interactivo-leishmaniasis-americas-aspectos-clinicos-diagnosticos-diferenciales

[6] P. P. Shinde y S. Shah, «A Review of Machine Learning and Deep Learning Applications», en *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, ago. 2018, pp. 1-6. doi: 10.1109/ICCUBEA.2018.8697857.

[7] C. Janiesch, P. Zschech, y K. Heinrich, «Machine learning and deep learning», *Electron. Mark.*, vol. 31, n.º 3, pp. 685-695, sep. 2021, doi: 10.1007/s12525-021-00475-2.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, 2006.

[9] D. Silver *et al.*, «A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play», *Science*, vol. 362, n.º 6419, pp. 1140-1144, dic. 2018, doi: 10.1126/science.aar6404.

[10] M. Peters, W. Ketter, M. Saar-Tsechansky, y J. Collins, «A reinforcement learning approach to autonomous decision-making in smart electricity markets», *Mach. Learn.*, vol. 92, n.º 1, pp. 5-39, jul. 2013, doi: 10.1007/s10994-013-5340-0.

[11] M. Shehab *et al.*, «Machine learning in medical applications: A review of state-of-the-art methods», *Comput. Biol. Med.*, vol. 145, p. 105458, jun. 2022, doi: 10.1016/j.compbiomed.2022.105458.

[12] J. Bobadilla, *Machine Learning y Deep Learning: Usando Python, Scikit y Keras*. Ediciones de la U, 2021.

[13] «La sinapsis (artículo) | Biología humana», Khan Academy. Accedido: 28 de mayo de 2023. [En línea]. Disponible en: https://es.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/the-synapse

[14] «NEURONAS - Naturopathic». Accedido: 28 de mayo de 2023. [En línea]. Disponible en: https://www.naturopathic.cat/es/anatomia-humana/sistema-nervioso/neuronas/

[15] «Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow». Accedido: 7 de enero de 2024. [En línea]. Disponible en: https://web.p.ebscohost.com/ehost/ebookviewer/ebook/bmxlYmtfXzM0MDYxNzRfX0FO0?sid=1545e9aa-1159-4f3b-8824-0db0fe66a62e@redis&vid=3&format=EB&rid=1

[16] «What are Convolutional Neural Networks? | IBM». Accedido: 13 de enero de 2024. [En

línea]. Disponible en: https://www.ibm.com/topics/convolutional-neural-networks
[17] R. Borja-Robalino, A. Monleon-Getino, y J. Rodellar, «Estandarización de métricas de rendimiento para clasificadores Machine y Deep Learning», *RISTI - Rev. Iber. Sist. E Tecnol. Inf.*, vol. 30, pp. 184-196, jun. 2020.
[18] G. Vilela Junior *et al.*, «MÉTRICAS UTILIZADAS PARA AVALIAR A EFICIÊNCIA DE CLASSIFICADORES EM ALGORITMOS INTELIGENTES», *Cent. Pesqui. Avançadas Em Qual. Vida*, vol. 14, p. 1, ene. 2022, doi: 10.36692/v14n2-01R.
[19] «3.3. Metrics and scoring: quantifying the quality of predictions», scikit-learn. Accedido: 7 de enero de 2024. [En línea]. Disponible en: https://scikit-learn/stable/modules/model_evaluation.html
[20] N. Shabanpour, S. V. Razavi-Termeh, A. Sadeghi-Niaraki, S.-M. Choi, y T. Abuhmed, «Integration of machine learning algorithms and GIS-based approaches to cutaneous leishmaniasis prevalence risk mapping», *Int. J. Appl. Earth Obs. Geoinformation*, vol. 112, p. 102854, ago. 2022, doi: 10.1016/j.jag.2022.102854.
[21] J. Smith, H. Xu, X. Li, L. Yang, y J. M. Gutierrez, «Compound Screening with Deep Learning for Neglected Diseases: Leishmaniasis». bioRxiv, p. 2021.10.02.462874, 2 de octubre de 2021. doi: 10.1101/2021.10.02.462874.
[22] M. Zare *et al.*, «A machine learning-based system for detecting leishmaniasis in microscopic images», *BMC Infect. Dis.*, vol. 22, n.º 1, p. 48, ene. 2022, doi: 10.1186/s12879-022-07029-7.
[23] M. Bamorovat *et al.*, «A novel diagnostic and prognostic approach for unresponsive patients with anthroponotic cutaneous leishmaniasis using artificial neural networks», *PloS One*, vol. 16, n.º 5, p. e0250904, 2021, doi: 10.1371/journal.pone.0250904.
[24] «3.2. Tuning the hyper-parameters of an estimator», scikit-learn. Accedido: 7 de enero de 2024. [En línea]. Disponible en: https://scikit-learn/stable/modules/grid_search.html
[25] M. Akay *et al.*, «Deep Learning Classification of Systemic Sclerosis Skin Using the MobileNetV2 Model», *IEEE Open J. Eng. Med. Biol.*, vol. PP, pp. 1-1, mar. 2021, doi: 10.1109/OJEMB.2021.3066097.
[26] A. Tragoudaras *et al.*, «Design Space Exploration of a Sparse MobileNetV2 Using High-Level Synthesis and Sparse Matrix Techniques on FPGAs», *Sensors*, vol. 22, n.º 12, Art. n.º 12, ene. 2022, doi: 10.3390/s22124318.