UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM ATLACOMULCO

"INTELIGENCIA ARTIFICIAL EN SISTEMAS EMBEBIDOS PARA LA CLASIFICACIÓN DE IMÁGENES COMO HERRAMIENTA DE DIAGNÓSTICO DE RETINOPATÍA DIABÉTICA"

T E S I S

Que para obtener el Título de:

Ingeniero en Computación

Presenta:

P. I. C. Gustavo Blas Duran

Asesor de Tesis:

Dr. en C. en I. E. Allan Antonio Flores Fuentes

Atlacomulco, México, marzo 2024

DEDICATORIAS

AGRADECIMIENTOS

RESUMEN

La aplicación de técnicas de inteligencia artificial para la detección de enfermedades ha resultado eficiente en la actualidad, debido a que son herramientas de apoyo para los médicos para dar un diagnóstico al paciente antes, durante y después de la enfermedad. Específicamente el procesamiento de las imágenes, por medio de redes neuronales convolucionales es útil para realzar tareas de clasificación. Este trabajo propone el uso de Redes Neuronales Convolucionales y Transferencia de aprendizaje para clasificar imágenes de fondo del ojo, con el objetivo de determinar el grado de avance de la Retinopatía Diabética. Como dispositivo de despliegue el uso de un Sistema en Módulo NVIDIA® Jetson Nano™. En primer lugar, se realiza un balanceo de clases y procesamiento de imágenes, en segundo lugar, mediante el uso de recursos en línea se ejecutan tareas de aprendizaje automático de una manera rápida. En esta etapa se realiza el entrenamiento de los modelos de CNN con *Dataset* de imágenes obtenidos de los accesos públicos a APTOS y Kaggle. La disponibilidad de arquitecturas para transferencia de aprendizaje acondicionadas para clasificación de imágenes de fondo del ojo genera valores determinados por métricas de rendimiento de exactitud superiores al 85% con imágenes pertenecientes al grupo de ojos sanos. Los modelos entrenados por el método de transferencia de aprendizaje presentan complicaciones para establecer el nivel de Retinopatía Diabética, sin embargo, no presentan retos al utilizar un Sistema en Módulo NVIDIA® Jetson Nano™ como dispositivo de despliegue. Finalmente se obtienen resultados del monitoreo del módulo, donde el porcentaje de uso de los CPU es menor al 80%.

Palabras Clave: Inteligencia Artificial, Clasificación, Minería de datos, Sistema embebido, Retinopatía diabética.

ABSTRACT

The application of artificial intelligence techniques for disease detection has currently been efficient because they are support tools for doctors to give a diagnosis to the patient before, during and after the disease. Specifically, image processing through convolutional neural networks is useful to enhance classification tasks. This work proposes the use of Convolutional Neural Networks and Transfer Learning to classify fundus images, with the aim of determining the degree of progression of Diabetic Retinopathy. As a deployment device the use of an NVIDIA® Jetson Nano™ System on Module. Firstly, class balancing and image processing are performed, secondly, by using online resources, machine learning tasks are executed in a fast manner. In this stage, the training of the CNN models is carried out with image Dataset obtained from public access to APTOS and Kaggle. The availability of architectures for transfer learning conditioned for classification of fundus images generates values determined by accuracy performance metrics greater than 85% with images belonging to the group of healthy eyes. Models trained by the transfer learning method present complications to establishing the level of Diabetic Retinopathy, however, they do not present challenges when using an NVIDIA® Jetson Nano™ System on Module as a deployment device. Finally, module monitoring results are obtained, where the percentage of CPU usage is less than 80%.

Keywords: Artificial Intelligence, Classification, Data Mining, Embedded System, Diabetic Retinopathy.

ÍNDICE

DEDICATORIAS 2
AGRADECIMIENTOS 3
RESUMEN 4
ABSTRACT 5
ÍNDICE 6
ÍNDICE DE TABLAS 9
ÍNDICE DE IMÁGENES 10
INDICE DE FIGURAS 12
INDICE DE GRÁFICAS 13
1. INTRODUCCIÓN 15
2. PLANTEAMIENTO DEL PROBLEMA 16
2.1. Definición del problema 16
2.2. Objetivos de la investigación 17
2.3. Preguntas de investigación 17
2.4. Justificación 18
2.5. Impactos 18
3. HIPÓTESIS 19
4. ESTADO DEL ARTE 19
4.1. La Inteligencia Artificial (IA) 19
4.1.1. Flujo de Inteligencia Artificial 19
4.1.2. Machine Learning (ML) 20
4.1.3. Aprendizaje Profundo 24
4.1.4. Métricas de rendimiento 29
4.1.5. Herramientas para el desarrollo de IA 32

4.2. Sistemas embebidos 32
4.2.1. Computación de borde 34
4.3. Retinopatía diabética 34
4.4. Inteligencia Artificial y sus aplicaciones en la detección de Retinopatía Diabética 35
4.4.1. *Dataset* o conjunto de imágenes oculares 37
4.5. Análisis y discusión del estado del arte 53
5. MÉTODO 54
5.1. Requerimientos 54
5.1.1. Software 54
5.1.2. Hardware 55
5.1.3. Diagrama de requisitos 56
5.1.4. Diagrama de flujo de funcionamiento 57
5.2. Análisis y exploración de datos 57
5.3. Balanceo de datos 63
5.4. Preprocesamiento de datos 66
5.5. Selección del modelo de red 73
5.6. Carga de red pre-entrenada con pesos sin la última capa 86
6. RESULTADOS Y DISCUSIÓN 87
6.1. Entrenamiento de MobileNetV2 con primer conjunto de datos 87
6.2. Entrenamiento: Extracción de características 91
6.3. Implementación del modelo en SBC 101
CONCLUSIONES 107
REFERENCIAS 110
ANEXOS 119

ANEXO A..... 120

ANEXO B..... 121

ANEXO C..... 122

ÍNDICE DE TABLAS

Tabla 4.1 Diferencias entre Redes Neuronales Artificiales y Redes Neuronales Convolucionales (Bagnato, 2018). 26
Tabla 4.2 Ejemplos de Redes Neuronales Convolucionales (Artola Moreno, 2019). 26
Tabla 4.3 Métricas de Rendimiento para la evaluación de un Modelo. 29
Tabla 4.4 Matriz de confusión. 30
Tabla 4.5 Matriz de confusión para clasificadores multiclase. 31
Tabla 4.6 Matriz de confusión con clase como foco de referencia (Grandini, 2020). 31
Tabla 4.7 Niveles de severidad de la Retinopatía Diabética (Navarro, Peña, & Escorcia, 2020). 34
Tabla 4.8 Matriz de referencias. 39
Tabla 5.1 Modelos de CNN proporcionados por la API de Keras. Keras (2023), Keras Applications. Recuperado de https://keras.io/api/applications/ 75
Tabla 6.1 Reporte de métricas de rendimiento para el modelo de Red con arquitectura base MobileNetV2. Fuente: elaboración propia con datos obtenidos de matriz de confusión. ... 91
Tabla 6.2 Métricas de rendimiento obtenidas para el Modelo de CNN con arquitectura base MobileNetV2 con entrenamiento por transferencia de características. Fuente: Elaboración propia. 97
Tabla 6.3 Métricas de rendimiento obtenidas para el Modelo de CNN con arquitectura base DenseNet201 con entrenamiento por transferencia de características. Fuente: Elaboración propia a partir de los datos obtenidos en la matriz de confusión. 100
Tabla 6.4 Load Average de la SOM durante la ejecución de inferencia de las imágenes.. 105
Tabla 6.5 Métricas de rendimiento para modelo con arquitectura MobileNetV2 en NVIDIA® Jetson Nano™. 106
Tabla 6.6 Métricas de rendimiento para resultados obtenidos de modelo con arquitectura DenseNet201 en NVIDIA® Jetson Nano™. 106

ÍNDICE DE IMÁGENES

Imagen 5.1 Imagen aleatoria extraída del conjunto de datos de Kaggle, valor de etiqueta: 0 (No RD, ojo sano) (Kaggle, 2015). 59
Imagen 5.2 Imagen de fondo de ojo etiquetada con la clase 1 (R1: Retinopatía Diabética Leve) (Kaggle, 2015). 59
Imagen 5.3 Imagen de fonde de ojo de la clase 2 (R2: Moderado) (Kaggle, 2015). 60
Imagen 5.4 Imagen de fondo de ojo de la clase 3 (R3: severa) (Kaggle, 2015). 60
Imagen 5.5 Imagen de fondo de ojo de la clase 4 (R4: proliferada) (Kaggle, 2015). 61
Imagen 5.6 Imagen de fondo de ojo no visible (Kaggle, 2015). 62
Imagen 5.7 Imagen de fondo de ojo con brillo intenso (Kaggle, 2015). 62
Imagen 5.8 Imagen de fondo de ojo con circunferencia no completa (Kaggle, 2015). 63
Imagen 5.9 Ajuste de circunferencia de fondo de ojo. 67
Imagen 5.10 Imagen de fondo de ojo antes a) y b) después del procesamiento de fusión entre la imagen original y una operación gaussiana. 70
Imagen 5.11 Imagen de fondo de ojo con CLAHE. a) antes de procesamiento, b) después de procesamiento. 71
Imagen 5.12 Resultado de carga de imágenes en Google Colab. Imágenes para entrenamiento, validación y test en las cinco clases de Retinopatía Diabética. 73
Imagen 6.1 Numero de variables a entrenar de una CNN con arquitectura de base MobiloeNetV2. 88
Imagen 6.2 Resumen de tamaño (MB) de modelo de CNN con arquitectura base MobileNEtV2. 88
Imagen 6.3 Resumen de tamaño (MB) de la arquitectura de MobileNetV2. 88
Imagen 6.4 a) imagen con procesamiento fusión de filtro gaussiano, b) imagen fusión de filtro gaussiano y procesamiento CLAHE. Fuente: elaboración propia con imágenes del Dataset APTOS2019. 92
Imagen 6.5 Resumen de tamaño (MB) del modelo de CNN con arquitectura base DenseNet201. 93
Imagen 6.6 Resumen de tamaño (MB) del modelo base de la arquitectura DenseNEt201.. 93
Imagen 6.7 Captura Escritorio de SO en NVIDIA® Jetson Nano™ listo para ejecutar el sistema clasificador. 102

Imagen 6.8 Monitor de recursos del dispositivo NVIDIA® Jetson Nano™ sin tareas en ejecución. 102
Imagen 6.9 Monito de recursos de NVIDIA® Jetson Nano™ durante la ejecución del sistema con el modelo de CNN con arquitectura base MobileNetV2. 103
Imagen 6.10 Monito de recursos de NVIDIA® Jetson Nano™ durante la ejecución del sistema con el modelo de CNN con arquitectura base DenseNet201. 104

INDICE DE FIGURAS

Figura 4.1 Las cuatro fases del flujo de implementación de la IA (MathWorks, 2023). 20
Figura 4.2 Técnicas de Aprendizaje Automático (Tatsat, Puri, & Lookabaugh, 2021).....21
Figura 4.3 Ejemplo de la estructura de una Red Neuronal Artificial (Ponce Cruz, 2010)..22
Figura 4.4 Arquitectura básica de una Red Neuronal Convolucional. Fuente elaboración propia con base en (MathWorks, 2023). 25
Figura 4.5 Estructura General de capas de una Red Neuronal Convolucional. Fuente: Elaboración propia con base en (Kandel & Catelli, 2020)......28
Figura 4.6 Flujo de Transferencia de Aprendizaje de una Red Neuronal Convolucional. Fuente: Elaborado a partir de (MathWorks, 2023). 29
Figura 5.1 Hardware (accesorios) para el dispositivo de despliegue del proyecto. Fuente: Elaboración propia......56
Figura 5.2 Diagrama de requisitos por etapas del flujo de IA. Fuente: Elaboración propia con base en (MathWorks, 2023).....56
Figura 5.3 Diagrama de flujo de funcionamiento. El sistema final se ejecuta en el SOM de la plataforma de NVIDIA® Jetson Nano™......57
Figura 5.4 Flujo del procesamiento de imágenes. 66
Figura 5.5 Flujo de procesos para el realce de características utilizando fusión de imágenes y operación gaussiana. 70
Figura 5.6 Jerarquía de variables para decisión de modelo de CNN. Fuente: Elaboración propia......78
Figura 5.7 Modelo de Red Neuronal Convolucional basado en Modelos de CNN proporcionados por Keras. Fuente: Elaboración propia......85

INDICE DE GRÁFICAS

Gráfica 5.1 Distribución de los datos Kaggle. El eje X representa los niveles de Retinopatía Diabética (clases: 5), el eje Y representa la cantidad de imágenes en cada nivel. 58
Gráfica 5.2 Distribución de imágenes del conjunto de APTOS y Kaggle. 64
Gráfica 5.3 Distribución del conjunto de datos balanceado. 65
Gráfica 5.4 Distribución de clases después de sobre muestreo. Conjunto de datos de APTOS. 66
Gráfica 5.5 Comparación del Tamaño (MB) de los distintos modelos de CNN disponibles de API. Fuente: Elaboración propia a partir de datos de Keras Applications. 79
Gráfica 5.6 Relación entre las variables peso (MB), Tiempo (ms) por paso de inferencia (GPU), Top % Exactitud y dimensiones de la imagen de entrada. Fuente: Elaboración propia a partir de datos de Keras Applications. 80
Gráfica 5.7 Modelos de CNN y dimensiones de imagen de entrada. Fuente: Elaboración propia a partir de datos de Keras Applications. 81
Gráfica 5.8 Comparación de numero de capas de profundidad de los modelos de CNN. Fuente: Elaboración propia a partir de datos de Keras Applications. 82
Gráfica 5.9 Comparación del tamaño (MB) de los modelos de CNN con dimensiones de imagen de entrada de 224x224px. Fuente: Elaboración propia a partir de datos de Keras Applications. 84
Gráfica 5.10 Números de capas de profundidad para los modelos de CNN con dimensiones de tamaño de imagen de entrada a de 224x224px. Fuente: Elaboración propia a partir de datos de Keras Applications. 84
Gráfica 6.1 Función de perdida para MobileNetV2, en el eje x época de entrenamiento, eje y valor de función de perdida. Fuente: elaboración propia con datos obtenidos durante el entrenamiento del modelo. 90
Gráfica 6.2 Valores de precisión para entrenamiento y validación del modelo de red con modelo base de la arquitectura MobileNetV2. Fuente: elaboración propia con datos obtenidos en el entrenamiento del modelo. 90
Gráfica 6.3 Distribución de imágenes del Dataset APTOS, posterior a un balanceo de clases. Fuente: elaboración propia con datos de (APTOS,2019). 92

Gráfica 6.4 Valores de función de perdida entrenamiento y validación del modelo de CNN con arquitectura base MobileNetV2, Transferencia de Aprendizaje. Fuente: Elaboración propia con datos obtenidos en el entrenamiento. 94
Gráfica 6.5 Valores de precisión obtenidos en entrenamiento y validación del modelo de CNN con arquitectura base MobileNetV2, 25 épocas de entrenamiento y transferencia de aprendizaje. Fuente: Elaboración propia con datos obtenido en el entrenamiento. 94
Gráfica 6.6 Matriz de confusión: clasificaciones actuales (filas) y clasificaciones obtenidas (columnas). Fuente: Elaboración propia a partir de datos obtenidos en la predicción del modelo. 95
Gráfica 6.7 Área bajo la curva para cada una de 5 clases del modelo clasificador con base MobileNetV2. Fuente: Elaboración propia a partir de datos obtenidos del entrenamiento. 96
Gráfica 6.8 Valores de función de perdida en entrenamiento y validación para el modelo de CNN con base DenseNet201. Fuente: Elaboración propia a partir de datos obtenidos en el entrenamiento. 97
Gráfica 6.9 Valores de la métrica de precisión en entrenamiento y validación para el modelo de CNN con base DenseNet201. Fuente: Elaboración propia a partir de datos obtenidos en el entrenamiento. 98
Gráfica 6.10 Matriz de correlación para los valores reales y los valores obtenidos por el modelo clasificador con base DenseNet201. Fuente: Elaboración propia a partir de datos obtenidos en la inferencia del modelo. 99
Gráfica 6.11 Valores de área bajo la curva para las clases del modelo clasificador con base DenseNet201. Fuente: Elaboración propia a partir de datos generados por el modelo. .. 100
Gráfica 6.12 Tiempos de inferencia por imagen de los modelos en NVIDIA® Jetson Nano™. 105

1. INTRODUCCIÓN

Con el paso de los años se han generado millones de datos representados y almacenados de diferentes formas; en bases de datos, imágenes, audios, videos, entre otras, así gracias al volumen de estos datos almacenados fue posible la creación de algoritmos computaciones capaces de aprender a través de su análisis, de aquí surge la Inteligencia Artificial (IA, por su acrónimo en español), que ha tenido un desarrollo de una manera exponencial, gracias al Big Data (BD). Por otra parte, los dispositivos electrónicos, cada vez más veloces y compactos han sido desarrollados e implementados en todas las industrias, debido a su capacidad para acelerar los procesos de producción gracias a la automatización.

El uso de imágenes médicas como diagnóstico de enfermedades ha permitido la pronta detección de diversas enfermedades en el cuerpo humano, de las cuales la mayor parte de estas son capturadas con procesos no invasivos. Este tipo de imágenes representan cerca del 90% de todos los datos sanitarios, y por lo tanto representa una fuente importante de evidencia para el análisis clínico y la intervención médica (Zhou, 2021).

En Medicina, el área de Oftalmología presenta un ejemplo de análisis clínico a base de imágenes, con el objetivo de la detección de una patología derivada de la Diabetes Mellitus, que de igual manera ha incrementado en los últimos años y afectando a la población sin importar el sexo o la edad (Navarro, Peña, & Escorcia, 2020).

Aprovechando el éxito de la IA para su aprendizaje a través de los datos, es posible ayudar a contrarrestar el desarrollo de las enfermedades en los pacientes a través de la detección temprana de sus anomalías, es por ello por lo que han sido desarrolladas estas técnicas para este tipo de problemáticas.

En este trabajo se hace uso de una subárea de IA nombrada aprendizaje profundo (*Deep Learning,* por su traducción al inglés), para el desarrollo de herramientas para el diagnóstico de enfermedades a través del análisis y clasificación de imágenes de fondo de ojo para la detección de Retinopatía Diabética en sus cinco niveles de avance: 0.- No RD, 1.- RD Leve, 2.- RD Media, 3.- RD Moderado, 4.- RD Severa y 5.- RD Proliferada. Además, se implementa sistema en un dispositivo embebido NVIDIA® Jetson Nano™ con el objetivo de economizar recursos económicos y de hardware.

2. PLANTEAMIENTO DEL PROBLEMA

Actualmente en el área médica de oftalmología se detectan patologías oculares entre las que se encuentra la retinopatía diabética, que, a través de las características geométricas de sus anomalías como el tamaño, no son perceptibles para el médico encargado de dar el diagnóstico. Como método de ayuda en la actualidad se hace uso de herramientas visuales tales como retinografías, las cuales permiten el aumento del tamaño de la imagen del fondo de ojo. No obstante, el dispositivo para en el que se encuentran alojadas estas herramientas resulta ser voluminosas, lo que implica un lugar específico como lo es un consultorio para su uso, además de ser poco accesibles por las personas por su precio en el mercado (Barrot & Franch, 2019).

En la actualidad por medio de la inteligencia artificial es posible realizar la identificación de objetos en imágenes a través las características geométricas como; la forma, el tamaño y color. Por medio de estas características es posible realizar un análisis de la imagen y realizar una asignación de etiqueta estableciendo la clase a la que pertenece.

Específicamente, las Redes Neuronales Convolucionales (*ConvNets* o *CNN*) son un subconjunto del aprendizaje automático, se utilizan principalmente para tareas de clasificación y visión artificial, el reconocimiento de objetos a través de patrones a través de la multiplicación de matrices (IBM, 2024).

2.1. Definición del problema

Las técnicas de IA requieren del análisis de grandes volúmenes de datos para que el modelo pueda extraer características para realizar una clasificación. En la técnica de aprendizaje profundo, la arquitectura de la CNN define el número de capas que la conforman, es decir entre mayor sea el número de capas, la profundidad de aprendizaje de la CNN aumenta.

En las capas de la CNN cada una de las imágenes es utilizada (*Dataset)* con el objetivo de extraer las características y patrones de formas en las que se relacionan con cada clase a través de cálculos entre matrices. Como consecuencia de ello se requiere de hardware con elevado costo computacional, para disminuir el tiempo de procesamiento ejecutado por el aprendizaje de la red neuronal convolucional, asi como el dispositivo de despliegue.

Por otra parte, deben considerarse imágenes de una base de datos de imágenes (*Dataset*) de fondo de ojo para realizar el entrenamiento del algoritmo con servidores en la nube de la plataforma *Google Colab*. Además de realizar la inferencia de la red en un Sistema en Módulo (*SOM*, por sus siglas en inglés), considerando el uso de NVIDIA® Jetson Nano™, el lenguaje de programación Python y la biblioteca *TensorFlow*.

2.2. Objetivos de la investigación

General

Desarrollar un sistema de clasificación de imágenes de fondo ojo en los niveles de retinopatía diabética (0.- No RD, 1.- RD Leve, 2.- RD Moderado, 3.- RD Severa, 4.- RD Proliferada) mediante aprendizaje profundo implementado en un dispositivo de despliegue un Sistema en Módulo para obtener su rendimiento.

Objetivos específicos

- Estudiar el estado del arte en Arquitecturas de Redes Neuronales Convolucionales para la clasificación de imágenes de fondo de ojo en los niveles de Retinopatía Diabética.
- Identificar el método para el balanceo de datos entre cada una de las clases del *Datset* de imágenes de fondo de ojo.
- Establecer las técnicas de procesamiento de imagen que resalten las características geométricas de las anomalías con el objetivo de proporcionar atributos de cada clase para un mejor entrenamiento de una red neuronal convolucional.
- Implementar el modelo de red neuronal entrenado en un Sistema en Módulo NVIDIA® Jetson Nano™.
- Evaluar a través de las métricas; precisión, exactitud, sensibilidad, especificidad y área bajo la curva los resultados del modelo de red neuronal convolucional entrenado.

2.3. Preguntas de investigación

¿Cuáles son las arquitecturas de Redes Neuronales Convolucionales con los mejores valores obtenidos en las métricas de rendimiento?

¿Qué métodos de balanceo de datos se pueden aplicar al conjunto de imágenes para evitar el sobreajuste y subajuste de los datos?

¿Qué procesamiento de imagen resaltan mejor las características geométricas de las anomalías de RD en una imagen a color?

¿Qué rendimiento presenta el Sistema en Módulo NVIDIA® Jetson Nano™ al implementar una red neuronal convolucional?

¿Qué valores de precisión, exactitud, sensibilidad, especificidad y área bajo la curva son obtenidos al interpretar el modelo de red en un dispositivo de despliegue Sistema en Módulo NVIDIA® Jetson Nano™?

2.4.Justificación

En los trabajos realizados, tanto por Alan Lands en "*Implementation of deep learning based algotihms for diabetic retinopathy classification from fundus images*" como por Saranya Ribini & Saai Nithil en *"Deep Convolutional Neuronal Network-Based Diabetic Retinopathy Detection in Digital Fundus Image"* se desarrollaron e implementaron arquitecturas de red neuronal convolucional para la clasificación de imágenes de fondo de ojo en los niveles de Retinopatía Diabética. Sin embargo, en la literatura actual no se encontró una arquitectura para obtener valores específicos de métricas de rendimiento como la precisión.

Por otra parte, no se ha desarrollado un sistema de inferencia con *Deep Learning* para clasificación de imágenes de Retinopatía Diabética que pueda ser ejecutado en Sistema en Módulo NVIDIA® Jetson Nano™.

2.5.Impactos

La Inteligencia Artificial y las computadoras de borde (*Edge Computing,* por su traducción al inglés) han tenido un gran auge en los últimos años, implementándose en diversas áreas como herramienta de apoyo a la toma de decisiones, la automatización de procesos y la obtención de información a través del análisis de los datos. El implementar un sistema de Inteligencia Artificial en una plataforma embebida, tendrá un impacto tecnológico que puede disminuir hasta en un 50% su precio en hardware, debido a que las plataformas embebidas son más accesibles al público por su bajo precio en el mercado.

3. HIPÓTESIS

Cómo se desempeñará un Sistema en Módulo NVIDIA® Jetson Nano™ como dispositivo de despliegue de una red neuronal convolucional para la clasificación de imágenes de fondo de ojo en los cinco niveles de Retinopatía Diabética (0.- No RD, 1.- RD Leve, 2.- RD Moderado, 3.- RD Severa, 4.- RD Proliferada) estableciendo métricas de precisión, exactitud, sensibilidad, especificidad de al menos 80% para cada una, y adicionalmente área bajo la curva.

4. ESTADO DEL ARTE

4.1. La Inteligencia Artificial (IA)

La definición de inteligencia artificial, o IA, es una simulación del comportamiento humano inteligente. Se trata de un sistema diseñado para percibir su entorno, entender su comportamiento y realizar acciones (MathWorks, 2023).

En los últimos años las técnicas de inteligencia artificial han sido implementadas en diversas áreas como herramienta para la toma de decisiones, por lo tanto, su desarrollo requiere de pasos en específico que van desde la extracción de los datos hasta la implementación del modelo en un entorno productivo.

La computación con IA es el trabajo de sistemas y software de aprendizaje automático que realizan el análisis de un gran volumen de datos con objetivo de extraer conocimientos nuevos y generar nuevas capacidades. Las operaciones de aprendizaje automático requieren de tres pasos: ETL (*ETL*, por sus iniciales en inglés - *Extract, Transform and Load*), entrenamiento e inferencia (Merritt, 2023).

4.1.1. Flujo de Inteligencia Artificial

El éxito de la IA requiere de ciertos pasos que le permiten aprender a través de los datos. Como se muestra en la Figura 4.1, el flujo de trabajo de IA implica la preparación de datos: los datos pueden estar representados de diversas formas, la cuales deben someterse a diversas procesos como es el caso de balanceo de clases y preprocesamiento de imágenes, para el caso específico de clasificación de imágenes, creación del modelo; los modelos de *CNN* para una

clasificación de imágenes, diseño del sistema; aquí se establece el diseño en hardware y software antes de su despliegue al entorno de producción, y el despliegue en el hardware o el sistema: las características con las que el dispositivo debe cumplir para un correcto funcionamiento, asi como la flexibilidad y adaptación en el entorno de producción (MathWorks, 2023).

Figura 4.1 Las cuatro fases del flujo de implementación de la IA **(MathWorks, 2023)**.

Existen varios elementos que componen la ciencia de la IA, dentro de las cuales se pueden encontrar tres grandes ramas (Ponce Cruz, 2010):

a. Lógica difusa: esta rama permite a unas computadoras analizar la información del mundo real entre lo falso y lo verdadero. El objetivo principal es crear un sistema basado en el comportamiento y pensamiento humano.
b. Redes neuronales artificiales: así como los humanos aplican el conocimiento ganando experiencia a nuevos problemas o situaciones, una red neuronal toma como ejemplos problemas resueltos para construir un sistema que tome decisiones y realiza clasificaciones.
c. Algoritmos genéticos: son una técnica de búsqueda iterativa inspirada en los principios de selección natural, su concepto se basa en la generación de poblaciones de individuos mediante la reproducción de los padres. La idea de los algoritmos genéticos es optimizar una función objetivo utilizando los principios de la selección natural sobre los parámetros de la función.

4.1.2. Machine Learning (ML)

El Aprendizaje automático o *Machine Learning* por su traducción al inglés, es una técnica mediante la cual los sistemas informáticos construyen modelos a través de algoritmos de aprendizaje y de la experiencia a través del análisis de datos, que permitan realizar predicciones sobre nuevas observaciones (Zhi-Hua, 2021).

Como se muestra en la Figura 4.2, el aprendizaje automático emplea tres tipos de técnicas: el aprendizaje supervisado: impulsado por las tareas, el aprendizaje no supervisado; impulsado por los datos y aprendizaje por refuerzo: aprende de los errores. Dentro de las diferentes técnicas podemos encontrar diversos algoritmos, tal es el caso la técnica de aprendizaje supervisado, en donde podemos encontrar el uso de los algoritmos de clasificación y regresión, por otro lado, los algoritmos de reducción de dimensionalidad y agrupación en el aprendizaje no supervisado (Tatsat, Puri, & Lookabaugh, 2021).

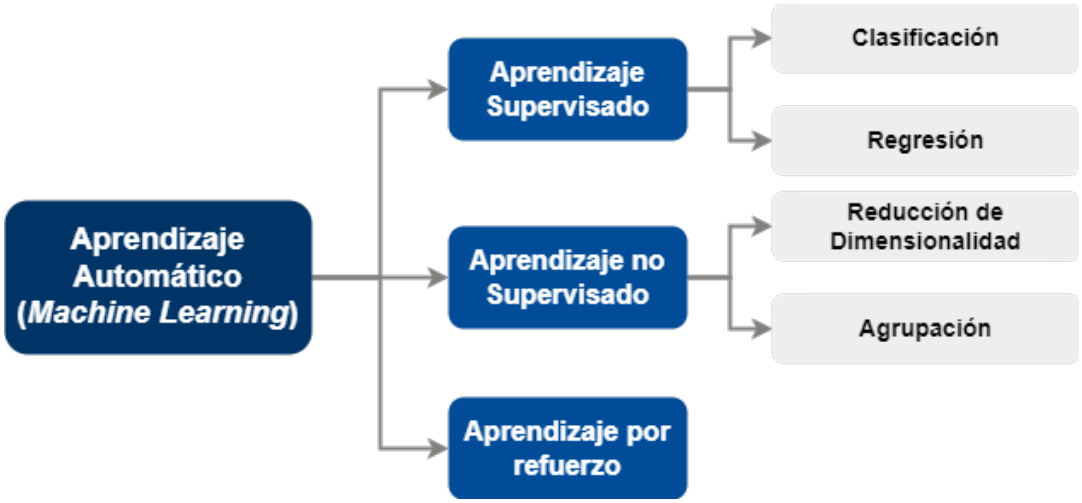


Figura 4.2 Técnicas de Aprendizaje Automático **(Tatsat, Puri, & Lookabaugh, 2021)**.

A. Aprendizaje automático supervisado

El aprendizaje automático supervisado es un área del *Machine Learning* en donde los algoritmos intentan ajustarse a un objetivo utilizando la etiqueta correspondiente a cada uno de los elementos que son proporcionados para el entrenamiento. A través de un análisis de los datos aprenden una regla que les ayudara a predecir las etiquetas para nuevas observaciones (Tatsat, Puri, & Lookabaugh, 2021). Como se muestra en la Imagen 4.2, existen dos variedades de algoritmos de aprendizaje supervisado: algoritmos de regresión y algoritmos de clasificación.

a) Redes neuronales artificiales

Dentro del aprendizaje supervisado, en el grupo de técnicas de clasificación se encuentran las redes neuronales artificiales, no existe una definición única para el termino de "red neuronal artificial". Kohonen (1988), en su trabajo *"An Introduction to Neuronal*

Computing" establece "*redes neuronales artificiales: son interconectadas masivamente paralelas de elementos simples (generalmente adaptativos) y sus organizaciones jerárquicas que están destinadas a interactuar con los objetos del mundo real de la misma manera que lo hacen los sistemas nerviosos biológicos*".

Como se muestra en la Figura 4.4, una Red Neuronal Artificial (RNA, por sus iniciales) consta de un conjunto de elementos de procesamiento conectados entre sí y entre los que se envían información a través de conexiones. Los elementos básicos de una RNA son: conjunto de unidades de procesamiento (neuronas), conexiones entre unidades (asociado a cada conexión un peso o valor) y funciones de salida o activación para cada unidad de procesamiento (Ponce Cruz, 2010).

Una red neuronal combina diversas capas de procesamiento y utiliza elementos simples que operan en paralelo, y están inspiradas en los sistemas nerviosos biológicos. Consta de una capa de entrada, una o varias capas ocultas y una capa de salida. Las capas están interconectadas mediante nodos, o neuronas, cada capa utiliza la salida de la capa anterior como entrada.

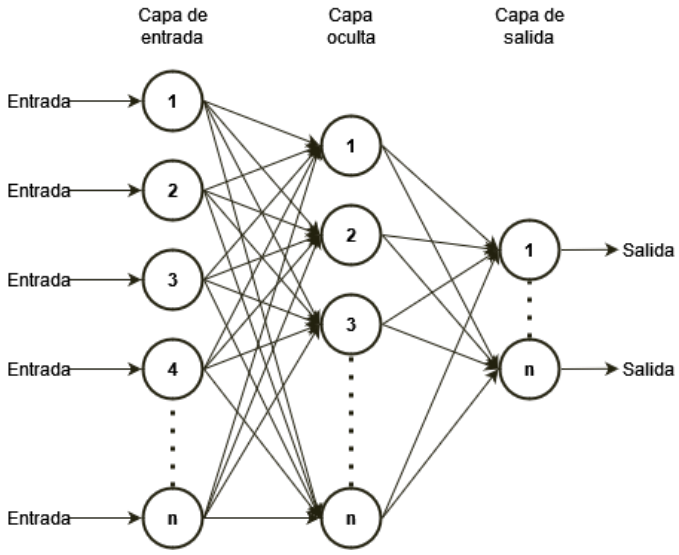


Figura 4.3 Ejemplo de la estructura de una Red Neuronal Artificial **(Ponce Cruz, 2010)**.

b) Flujo de trabajo típico para diseñar redes neuronales

Cada aplicación de red neuronal es única, pero el desarrollo de la red suele implicar los pasos siguientes (MathWorks, 2023):

1. Acceder a los datos y prepararlos.
2. Crear la red neuronal.
3. Configurar las entradas y salidas de la red.
4. Ajustar los parámetros de la red (ponderaciones y tendencias) para optimizar el rendimiento.
5. Entrenar la red.
6. Validar los resultados de la red.
7. Integrar la red en un sistema de producción.

La red neuronal almacena información en forma de pesos, por ello al entrenar la red neuronal con nueva información los pesos deben actualizarse en consecuencia. El enfoque sistemático para modificar los pesos según la información dada se llama regla de aprendizaje que es un componente vital en la investigación de redes neuronales (Valderrama Molano, 2017).

Se denomina entrenamiento al proceso de configuración de una red neuronal para que las entradas produzcan las salidas deseadas a través del fortalecimiento de las conexiones. Existen diversos modelos de Red Neuronal basados en función de su arquitectura y forma de aprendizaje, la red más utilizada son las basadas en capas de neuronas de tipo perceptrón, las cuales son entrenadas mediante la técnica de retro propagación (*Backpropagation*) (Sáenz Bajo & Álvaro Ballesteros, 2002).

Los pesos en una red pueden actualizarse a partir de los errores calculados para cada ejemplo de formación y esto se le denomina aprendizaje *online*: puede dar lugar a cambios rápidos, pero también caóticos en la red. Alternativamente, los errores se pueden guardar en todos los ejemplos de entrenamiento y la red se puede actualizar al final. Esto se llama aprendizaje por Lotes o *batch* y, a menudo es más estable.

Debido a la dimensión del conjunto de datos y a las eficiencias computacionales, el tamaño del lote, el número de ejemplos que la red muestra antes de una actualización a menudo se reduce a un número pequeño, como decenas o cientos de ejemplos. La cantidad de pesos que se actualizan es controlada por un parámetro de configuración llamado tasa de aprendizaje. También se denomina tamaño de paso y controla el paso o cambio realizado en los pesos de la red para un error dado. A menudo se utilizan tasas de aprendizaje pequeñas, tales como 0.1 o menores (Valderrama Molano, 2017).

Otro termino término utilizado es el que incorpora las propiedades de la actualización de peso anterior para permitir que los pesos continúen cambiando en la misma dirección incluso cuando hay menos errores calculados se le denomina *momentum* (Valderrama Molano, 2017).

Un modelo de red neuronal consta de parámetros, definimos como parámetro a un valor que tiene una característica, conta de parámetros como un sesgo y un peso. La retro propagación actualiza los parámetros utilizando una regla definida como una ecuación (Takano, 2017).

La decadencia de la tasa de aprendizaje: se utiliza para disminuir la tasa de aprendizaje durante épocas para permitir que la red realice grandes cambios en las ponderaciones al principio y pequeños cambios de ajuste fino más tarde en el programa de entrenamiento (Valderrama Molano, 2017).

B. Aprendizaje automático no supervisado

Al igual que el al aprendizaje supervisado el aprendizaje no supervisado es una rama del *Machine Learning* que permite descubrir estructuras en los datos, la principal diferencia consiste en que a los algoritmos de aprendizaje no supervisado no se les proporciona las etiquetas de los elementos de entrenamiento. Los algoritmos de aprendizaje no supervisado se denominan algoritmos de agrupamiento (*Clustering Algorithms*, por su traducción al inglés) asignan observaciones a los subgrupos que consisten en las características más similares entre los subgrupos (Tatsat, Puri, & Lookabaugh, 2021).

C. Aprendizaje por refuerzo

En esta técnica existe un agente que aprende de la experiencia, de acuerdo con las observaciones de su entorno selecciona y realiza la acción con el objetivo de recibir premios o castigos a cambio (Tatsat, Puri, & Lookabaugh, 2021).

4.1.3. Aprendizaje Profundo

El aprendizaje profundo (*Deep Learning,* por su traducción al inglés) es un subcampo especifico del aprendizaje automático, utilizado principalmente para tareas de visión, ya que su aprendizaje hace énfasis en el aprendizaje de capas sucesivas de representaciones jerárquicas cada vez más significativas. La profundidad de un modelo la define el número de capas, ya que dependiendo del modelo puede implicar decenas o incluso cientos de capas

sucesivas de representaciones, estas representaciones en capas se aprenden casi siempre a través de modelos como lo son las redes neuronales que están estructuradas en capas apiladas una encima de otra (Chollet, 2018).

Este tipo de redes se entrena mediante el aprendizaje supervisado que procesa sus capas imitando al ojo humano para identificar las características más importantes dentro de las imágenes. Las capas tienen una jerarquía, en donde las primeras detectan patrones sencillos, las capas más profundas detectan los patrones más complejos (Abellán, 2021).

La convolución es simplemente el proceso de tomar una pequeña matriz llamada núcleo y ejecutarla sobre todos los pixeles de una imagen. Para cada píxel se realiza la operación matemática que se involucra el valor del píxel y el valor de los pixeles a sus alrededores para determinar el valor de salida (Sharda, 2021).

La convolución es una operación matemática que se aplica generalmente sobre matrices, las capas de *pooling* tienen la funcionalidad de reducir la dimensionalidad de los datos que se están procesando. Estas dos operaciones son el secreto de las redes neuronales convolucionales (Abellán, 2021). En la Figura 4.5 se presenta la arquitectura de una red neuronal convolucional en donde la entrada corresponde a una imagen y como salida una clasificación de esta.

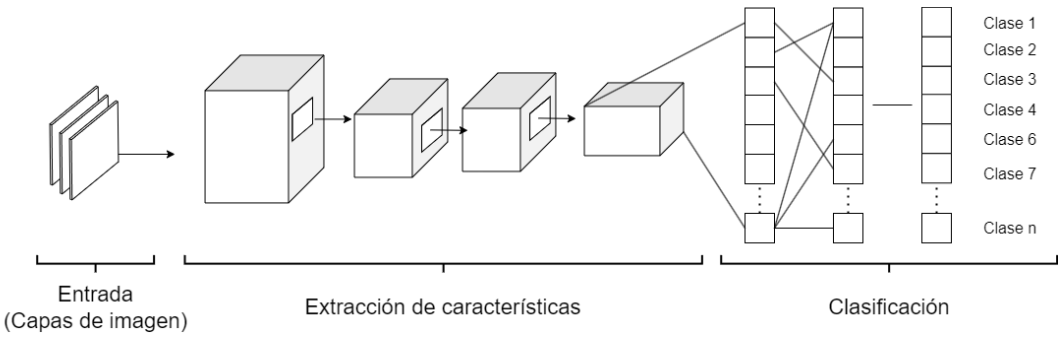


Figura 4.4 Arquitectura básica de una Red Neuronal Convolucional. Fuente elaboración propia con base en **(MathWorks, 2023)**.

En la Tabla 4.1, se muestran algunas de las características que diferencias una Red Neuronal Artificial Tradicional de una Red Neuronal Convolucional.

Tabla 4.1 Diferencias entre Redes Neuronales Artificiales y Redes Neuronales Convolucionales **(Bagnato, 2018)**.

	Red Neuronal Artificial ARN tradicional	**Red Neuronal Convolucional CNN**
Datos de entrada	Las características que analizamos. Por ejemplo: ancho, alto, grosos, etc.	Pixeles de una imagen. Si es en color, serán 3 capas, para rojo, verde y azul.
Capas ocultas	Elegimos una cantidad de neuronas para las capas ocultas.	Tenemos de tipo: Convolución (con un tamaño de *kernel* y una cantidad de filtros) *Subsampling*
Capa de salida	La cantidad de neuronas que queremos clasificar.	Debemos "aplanar" la última capa de convolución con una (ó más) capas de neuronas ocultas "tradicionales" y hacer una salida mediante SoftMax a la capa de salida que clasifica al número de clases, una neurona por cada clase.
Aprendizaje	Supervisado	Supervisado
Interconexiones	Entre capas, todas las neuronas de una capa con la siguiente.	Son mucho menos conexiones necesarias, pues realmente los pesos que ajustamos serán los de los filtros/*Kernels* que usamos.
Significado de la cantidad de capas ocultas	Realmente es algo desconocido y no representa alfo en sí mismo.	Las capas ocultas son mapas de detección de características de la imagen y tienen jerarquía: primeras capas detectan líneas, luego curvas y cada vez más elaborados.
Backpropagation	Se utiliza para ajustar los pesos de todas las interconexiones de las capas.	Se utiliza para ajustar los pesos de los *Kernels*.

a. Arquitectura de redes neuronales convolucionales

Existen diversos modelos de Red Neuronal Convolucional que han sido desarrollados por la comunidad de investigadores y científicos para la implementación en sus proyectos, de los cuales modelos como los que se muestran en la Tabla 4.2, son ejemplos de CNN que su arquitectura ha sido publicada para que puedan ser utilizadas en los proyectos de la comunidad. Estos modelos tienen características que los definen, principalmente en la conformación de capas y la organización de estas.

Tabla 4.2 Ejemplos de Redes Neuronales Convolucionales **(Artola Moreno, 2019)**.

Algoritmos	**No. de capas**	**Tamaño de imagen de entrada**	**Índice de Error**	**Características**
AlexNet	8	227x227	15.3%	Uso de la función de activación ReLU. Uso de *Dropout* para tratar el *Overfitting*. Las primeras 5 capas son convolucionales y las restantes están totalmente conectadas.

VGGNet	19	224x224	7.3%	Similar a AlexNet, pero con capas convolucionales de 3x3 y numerosos filtros.
GoogleNet	22	224x224	6.67%	Se compone de sus *Inception layers* o capas de inicio. La idea principal de estas es cubrir un área mayor pero también preservar una buena resolución para pequeña muestra de información sobre las imágenes.
ResNet	152	224x224	3.57%	Redes Residuales – son capaces de aprender funciones más complejas y consecuentemente conducir a un mejor rendimiento.
Inception-V3	42	299x299	3.5%	Es una arquitectura convolucional profunda ampliamente utilizada para tareas de clasificación.

Las CNN aprenden a detectar diferentes características de una imagen mediante decenas o cientos de capas ocultas. Cada capa oculta aumenta la complejidad de las características de la imagen aprendidas. Por ejemplo, la primera capa oculta podría aprender cómo detectar bordes, mientras la segunda aprende cómo detectar formas más complejas propias de la forma del objeto que se intenta reconocer.

b. Estructura de las redes neuronales convolucionales

Las capas de las redes neuronales convolucionales pueden clasificarse en dos categorías: capas primarias y capas secundarias. Las capas primarias son la principales usadas por la Red Neuronal Convolucional y consiste en las capas de la convolución, capas de activación, capas de agrupación (*Pooling Layer,* por su traducción al inglés), capas planas (*Flatten Layers,* por su traducción al inglés) y capas densas. Las capas secundarias son capas opcionales que pueden agregarse al modelo para que este sea más robusto contra el ajuste y aumentar la capacidad de la generalización, capas como lo son: abandono (*Dropout Layer*, por su traducción al inglés), capas de normalización por lotes (*Batch Normalization Layer,* por su traducción al inglés) y capas de regularización (Kandel & Catelli, 2020).

En la Figura 4.6, se muestra el ejemplo de la estructura de las capas de una red neuronal convolucional.

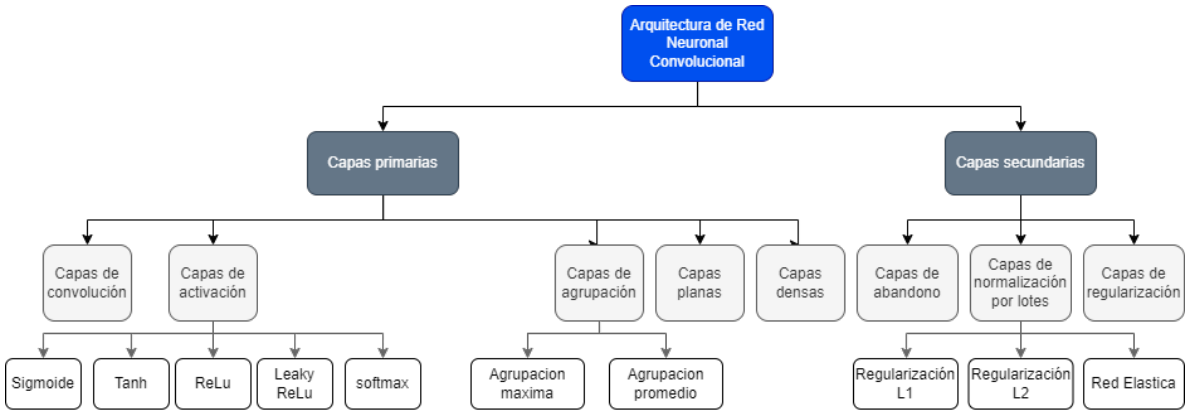


Figura 4.5 Estructura General de capas de una Red Neuronal Convolucional. Fuente: Elaboración propia con base en (Kandel & Catelli, 2020).

c. Entrenamiento de redes neuronales convolucionales

El entrenamiento o ajuste de los pesos, de las redes neuronales puede ser de las siguientes dos formas (Chollet, 2018):

a) Entramiento desde cero (*Training from Scratch*, por su traducción al inglés): como su nombre indica, el ajuste de los pesos se basa en los datos disponibles iniciando con un valor aleatorio.

b) Entrenamiento utilizando una red pre-entrenada: una red que se entrenó previamente con un conjunto de datos, generalmente de gran escala. Por la estructura de los modelos de Red Neuronal Convolucional existen dos formas de utilizarla:

   1 Extracción de características (*Feature Extraction,* por su traducción al inglés): donde se utilizan las representaciones aprendidas en un entrenamiento anterior para extraer características interesantes de nuevos ejemplos. Estas características luego se ejecutan a través de un nuevo clasificador, que se entrena desde cero.

   2 Ajuste fino (*Fine-Tuning,* por su traducción al inglés): consiste en descongelar algunas de las capas superiores de una base de modelo congelada utilizada para la extracción de características y entrenar conjuntamente tanto en la parte recién agregada del modelo como en capas superiores. Se le llama ajuste fino porque ajusta ligeramente las representaciones más abstractas del modelo que se está reutilizando, para que sean más relevantes para el problema en cuestión.

La transferencia de aprendizaje (*Transfer Learning,* por su traducción al inglés) es una metodología de Aprendizaje Profundo, en la que un enfoque en el que la información que se ha extraído de un dominio es utilizada en otro dominio (Md. Milon Islam, 2022). El flujo de los procesos para su implementación se muestra en la Figura 4.7.

Figura 4.6 Flujo de Transferencia de Aprendizaje de una Red Neuronal Convolucional. Fuente: Elaborado a partir de **(MathWorks, 2023)**.

La transferencia de aprendizaje resulta más habitual debido a que permite entrenar modelos con menos datos, reducir el tiempo de entrenamiento y los recursos informáticos, además de que te permite aprovechare arquitecturas de modelos desarrolladas por comunidades de investigación de *Deep Learning*. El tamaño, precisión y la velocidad de predicción son variables que se deben considerar al utilizar modelos previamente entrenados (MathWorks, 2023).

4.1.4. Métricas de rendimiento

Las métricas de rendimiento nos ayudan a definir el rendimiento del método implementado. Muestran que tan en acuerdo o en desacuerdo están el experto y el método propuesto para calificar una enfermedad (Perdomo Charry & González Augusto, 2019). En la Tabla 4.3 se muestran las métricas de rendimiento más utilizadas, la formula con la que es generado el valor.

Tabla 4.3 Métricas de Rendimiento para la evaluación de un Modelo.

Medida	**Formula**	**Descripción**
Sensibilidad	$\frac{TP}{TP+FN}$	Relación entre los verdaderos positivos clasificados y los verdaderos positivos reales en la verdad básica. También conocido como tasa de verdaderos positivos. (TPR)
Especificidad	$\frac{TN}{TN+FP}$	Relación entre los verdaderos negativos clasificados y los verdaderos negativos reales en la verdad básica. (1-SP) también se conoce como tasa de falsos positivos (FPR).
Exactitud		La precisión mide la proporción de pixeles clasificados correctamente (tanto de los vasos como de los no vasos) con

	$\frac{TP + TN}{TP + TN + FP + FN}$	respecto al número total de pixeles en el campo de visión de la imagen.
Precisión	$\frac{TP}{TP + FP}$	Porcentaje de casos positivos detectados.
Área bajo la curva	$\frac{Se + Sp}{2}$	Área cubierta por la curva Roc cuando se logra la optimización.
Puntuación - F	$2 * \frac{Se + Pr}{Se + Pr}$	f-score es el promedio ponderado de precisión y recuerdo. Tiene en cuenta tanto los falsos positivos como los falsos negativos.
Coeficiente Kappa	$\frac{p_0 - p_e}{1 - p_e}$	El coeficiente de Kappa refleja la concordancia Inter observador, puede tomar valores entre –1 y +1. Mientras más cercano a +1, mayor es el grado de concordancia Inter observador, por el contrario, mientras más cercano a -1, mayor es el grado de discordancia Inter observador.

Fuente: (Navarro, Peña, & Escorcia, 2020)

La matriz de confusión enumera los patrones entre los resultados de producción esperados y entrenados. La Tabla 4.4 muestra las combinaciones entre las predicciones y resultados, denominada como Matriz de confusión.

Tabla 4.4 Matriz de confusión.

		Resultado esperado	
		Positivo	*Negativo*
Resultados previstos	*Verdadero*	Verdadero-Positivo (TP)	Verdadero-Negativo (TN)
	Falso	Falso-Positivo (FP)	Falso-Negativo (FN)

Donde:

TP: positivos que fueron clasificados correctamente como positivos por el modelo.

FN: negativos que fueron clasificados correctamente como negativos por el modelo.

TN: positivos que fueron clasificados incorrectamente como negativos.

FP: negativos que fueron clasificados incorrectamente como positivos.

Estas variables son usadas para el cálculo de las métricas de rendimiento para un modelo clasificador binario. Para el problema de clasificación multiclase, es decir un modelo clasificador mayor a 2 clases, se utiliza una tabla similar a la Tabla 4.4, con un aumento de

filas y columnas, donde las filas corresponden a los valores de las etiquetas de clasificación actual y las columnas corresponden a los valores de etiquetas que se obtuvieron en la clasificación, como se observa en la Tabla 4.5.

Las clases son listadas en el mismo orden de las filas y columnas, por lo tanto, los elementos correctamente clasificados se ubican en la diagonal principal de arriba a la izquierda hacia abajo a la derecha, lo que corresponde al número de veces que ambos evaluadores coinciden (Grandini, 2020).

Para estudio visuales, los valores que coincidan en la matriz son sombreados de acuerdo con el número de muestras, como se muestra en la Tabla 4.5, por lo que un modelo adecuado tendrá la mayoría de las muestras a lo largo de la diagonal, una de las ventajas de esta fila es para ver si es desequilibrio de las clases reales hizo que el modelo clasificase erróneamente muestras de la clase minoritaria (Microsoft, 2023).

Tabla 4.5 Matriz de confusión para clasificadores multiclase.

		Clasificación prevista					
	Clases	No RD	Leve	Moderada	Servera	Proliferada	Total
Clasificación actual	No RD	n					n
	Leve		n				n
	Moderada			n			n
	Severa				n		n
	Proliferada					n	n
	Total	n	n	n	n	n	N

Tabla 4.6 Matriz de confusión con clase como foco de referencia **(Grandini, 2020)**.

		Clasificación prevista				
	Clases	No RD	Leve	Moderada	Servera	Proliferada
Clasificación actual	No RD	TN	FP	TN	TN	TN
	Leve	FN	TP	FN	FN	FN
	Moderada	TN	FP	TN	TN	TN
	Severa	TN	FP	TN	TN	TN
	Proliferada	TN	FP	TN	TN	TN

Los valores de las métricas que se han diseñado para para multiclase tienen como sufijo *micro*, *macro* o *ponderado* (*Weighted* por su traducción al inglés). Para el sufijo macro la métrica se calcula para cada clase (Véase en Tabla 4.6) y se toma el promedio no ponderado, de esta forma no hay distinción de clases cuando se tiene un desbalanceo. El sufijo micro se calcula la métrica de forma global mediante el recuento del total de verdaderos positivos,

falsos negativos y falsos positivos (independientemente de las clases). Por último, el sufijo ponderado se calcula la métrica para cada clase y se toma el promedio ponderado en función del número de muestras por clase (Microsoft, 2023; Microsoft).

4.1.5. Herramientas para el desarrollo de IA

Los métodos de aprendizaje profundo han dado como resultado mejoras significativas en el rendimiento en varios dominios de aplicaciones, por lo que se han desarrollado *Frameworks* y bibliotecas para facilitar su implementación, tales como *Caffe, TensorFlow, Keras y Pytorch* (Valderrama Molano, 2017).

TensorFlow es una plataforma de código abierto desarrollado por Google, en donde es posible expresar y ejecutar algoritmos de aprendizaje automático. Se puede utilizar en diversos dispositivos con pequeños cambios o incluso ninguno (Martín Abadi, 2015).

Keras es un *Framework* para el lenguaje de programación *Python*, desarrollado por investigadores para modelos de aprendizaje profundo. Entre sus principales características se encuentra la ejecución del mismo código en sin problemas en un CPU tanto como en una GPU, además de que admite diversas arquitecturas de redes neuronales arbitrarias (Chollet, 2018).

4.2.Sistemas embebidos

Un sistema embebido es un sistema de cómputo aplicado, es distinguido de otros tipos de sistemas de cómputo como las computadoras personales (PC) por su limitación en el hardware y/o software. En estos términos, sus principales limitaciones están en el rendimiento del procesamiento, consumo de energía, memoria, etc. Por otra parte, las limitaciones del software se encuentran el sistema operativo, el cual se encuentra limitado en cuento a aplicaciones (Noergaard, 2005). El termino sistema embebido se refiere a un equipo electrónico que incluye procesamiento de datos y que está diseñado para realizar una función dedicada (Vargas, 2020).

Este tipo de sistemas están formados de con los mismos elementos de un sistema informático de propósito general, adicionando elementos específicos de entrada y salida. Con el avance de la tecnología se han desarrollado diferentes sistemas embebidos los cuales cuentan con

varias unidades de cómputo conformando sistemas de un solo procesador (monocore) o con varios (multicore), la tendencia actual es que estos sean multicore (Crespo, 2020).

Así como los sistemas informáticos de propósito general, los sistemas embebidos también cuentan con un sistema operativo, este elemento es un punto clave en este tipo de sistemas, ya que estos se adaptan a las necesidades del hardware al que se va a operar. Se han desarrollado diversos sistemas operativos, pero los que se basan en las distribuciones de Linux cuentan con grandes ventajas como disponibilidad de compiladores para varios lenguajes, existencia de una gran variedad de herramientas para el desarrollo de aplicaciones, protocolos de comunicación, y muchos más (Crespo, 2020).

Crespo en su trabajo nombrado "Integración de redes neuronaes en sistemas empotrados. Clasificación de imagen con RaspberryPi" presenta tres plataformas comerciales de placas comerciales de costo reducido para el desarrollo de sistemas embebidos, los cuales hasta ahora cuentan con una gran capacidad de cómputo, que son ideales para el desarrollo de sistemas embebidos. La primera placa que presenta es Beagle Bonne, orientada a las altas prestaciones para aplicaciones técnicas de inteligencia artificial que está basado en el procesador de Texas Instruments Sitara AM5729. Este sistema está basado por dos core ARM A15, dos core de ARM M4 y dos core PowerVr de procesamiento gráfico.

El segundo sistema embebido es Raspberry Pi en su versión 4, que está basado en el chip Broadcom que está formado por cuatro núcleos de ARM Córtex A-72 de 64 bit en SoC (*System on Chip*) a 1.5 GHz, el sistema de desarrollo se basa en Raspbian además de que es un dispositivo de bajo costo (Crespo, 2020).

Otro computador de placa única es NVIDIA® Jetson Nano™, la cual es ha sido desarrollada para aplicaciones de Inteligencia Artificial, esta se caracteriza por su GPU integrada Maxwell 128-core, CUDA, CPU Quad-core ARM Cortex-A57 1.43 GHz, memoria RAM de 4 GB (También existe una versión disponible en 2GB), GPIOs, trabaja con un sistema operativo Ubuntu en su versión 18.04 el cual es otorgada espacialmente para la placa en la documentación de NVIDIA *Embeded*.

4.2.1. Computación de borde

El termino Computación de borde (*Edge Computing,* por su traducción al inglés) se define a la práctica de procesar datos físicamente más cerca de su fuente o del dispositivo del usuario final. Una de sus ventajas es el funcionamiento en ubicaciones remotas con baja o nula conexión a internet, ya que puede evitar el uso de computadoras en la nube o centros de datos para su funcionamiento (Yeung, 2022).

4.3. Retinopatía diabética

Retinopatía Diabética es una enfermedad ocular de elevada prevalencia, es caracterizada por cambios anormales a nivel de la retina, que se manifiesta con gran frecuencia en personas que padecen diabetes mellitus (Navarro, Peña, & Escorcia, 2020). Es una enfermedad progresiva que se diagnostica de acuerdo con algunas anormalidades clínicas de difícil detección. Por otra parte, la RD se divide en dos etapas principales: no proliferada y proliferada, llamada así por la ausencia o presencia de nuevos vasos sanguíneos anormales que emanan la retina (Navarro, 2020).

El marco general para los procesos de detección de la RD implica pasos específicos de preprocesamiento, extracción/selección de características, elección de un método de clasificación adecuado y, finalmente la evaluación de los resultados. El protocolo de clasificación se muestra en la Tabla 4.5.

Tabla 4.7 Niveles de severidad de la Retinopatía Diabética **(Navarro, Peña, & Escorcia, 2020)**.

Grado	**Signos clínicos**	**Decisión**
R0: No RD	No existen anormalidades	Tipo 1: volver a examinar a los dos años.
R1: Leve	• Más de 4 micro aneurismas o puntos de hemorragia. • Exudados > 2 diámetros de disco de la fóvea. Algunas manchas y hemorragias más grandes aceptables. Si hay más de 20MA o hemorragias por campo fotográfico se debe actualizar a R3 moderado.	Volver a evaluar después de 12 meses.
R2: moderado	• Cualquier característica de R2. • Manchas y hemorragias más grandes. • Hasta un cuadrante de reborde venoso.	Volver a evaluar después de 6 meses.

R3: Severa	Uno o más de: • Anomalías microvasculares intrarretinianas definidas (IRMA). • Dos cuadrantes o más de cuentas venosas. • Cuatro cuadrantes de manchas o hemorragias más grandes.	Revisión por oftalmólogo en 6 semanas.
R4: Proliferada	Uno o más de: • Neovascularización • Hemorragia su hialoidea o vítrea. • Desprendimiento de retina por tracción o gliosis de retina.	Remisión urgente a un oftalmólogo; considere revisar dentro de dos semanas.

4.4. Inteligencia Artificial y sus aplicaciones en la detección de Retinopatía Diabética

Las investigaciones en el área de oftalmología realizan la detección de objetos en una imagen de fondo de ojo, para determinar las anomalías se tiene como base la imagen de un fondo de ojo en estado sano, de esta manera se realizan comparaciones entre ambas imágenes y se determina el estado en el que se encuentra la imagen del paciente.

De acuerdo con los trabajos consultados, las técnicas utilizadas en la detección de retinopatía Diabética van desde la utilización de máquinas de soporte vectorial hasta modelos híbridos en donde se hace uso de modelos de red neuronal convolucional que son entrenados por transferencia de aprendizaje.

Como se observa en la Tabla 4.6, distintas técnicas han sido utilizadas para la clasificación de imágenes de fondo de ojo en los niveles de Retinopatía Diabética, sin embargo, los últimos trabajos muestran el aumento del uso de Redes Neuronales Convolucionales que son entrenadas por el método de Transferencia de Aprendizaje, en otros casos los modelos aportados son modificados agregando técnicas para complementación con el objetivo de obtener mejores valores en las métricas de clasificación.

Se han desarrollado trabajos los cuales se han puesto a disposición de la comunidad científica para brindar información y dar un diagnóstico de las enfermedades oculares, Zilly en su

trabajo "*Boosting Convolutional Filters with Entropy Sampling for Optic Cup and Disc Image Segmentation from Fundus Images*" propone un método basado en una red neuronal convolucional para la segmentación del disco y la copa óptica. Hace uso de filtros en los que se aprenden en varias capas y la salida de las capas anteriores como entrada a la siguiente. Una característica de este trabajo es que entrena un clasificador de regresión logística (la variable) en la salida de los filtros aprendidos. La regresión logística se trata de una red neuronal de una neurona.

Existen diferentes enfermedades de retina, cada una de ellas cuenta con características diferentes que son difíciles de identificar, Arunkumar & Karthigaikumar a traves de su trabajo "*Multi-retinal disease classification by reduced Deep Learning features*" desarrolla un modelo en el que obtienen características de *Deep Learning* y hacen uso del clasificador SVM (*Support Vector Machine*) para el diagnóstico de enfermedades oculares. SVM es un conjunto de algoritmos de aprendizaje supervisado que reconoce y analiza los datos de manera supervisada. La principal contribución de este trabajo es la reducción de características para la clasificación de enfermedades de retina.

Asi mismo se el uso de redes tipo *Faster R-CNN* (arquitectura de detección de objetos que utiliza redes neuronales de convolución) que logra tener una coincidencia de 85.4% (Navarro, Peña, & Escorcia, 2020).

Otro modelo nombrado de red convolucional de la arquitectura U-Net con ventana de imagen de 16 pixeles, la cual se compone de cuatro capas convolucionales y tres des convolucionales. Esta arquitectura se utiliza para la segmentación rápida y precisa de las imágenes (Pozo, 2019).

Rubini en su trabajo "*Deep Convolutional Neuronal Network-Based Diabetic Retinopathy Detection in Digital Fundus Image*" propone un modelo que no necesita de preprocesamiento que se basa en aprendizaje profundo automatizado para la detección de RD basado en Redes neuronales convolucionales, en la cual se basa a partir de las intensidades de los pixeles para clasificar la imagen de la retina como sana o defectuosa, obteniendo una precisión de 97%.

Trabajos más actuales hacen uso de la arquitectura Inception-V3 para la detección de retinopatía diabética y el impacto de la resolución de las imágenes. El trabajo más actual en

lo que va del año es el que presenta Gangwar, presenta un hibrido de *Deep Learning* que se compone de un bloque de capas de red neuronal convolucional sobre el modelo de aprendizaje por transferencia de Inception-ResNet-v2 ya entrenado (Navarro, Peña, & Escorcia, 2020).

4.4.1. *Dataset* o conjunto de imágenes oculares

Los trabajos que se han desarrollado hasta ahora hacen uso de conjunto de datos (*Dataset*) de imágenes de fondo de ojo, las cuales pueden ser usados para entrenar el modelo personalizado. Estas bases además del conjunto de imágenes también proporcionan las etiquetas correspondientes, ya que son imágenes que anteriormente han sido analizadas por especialistas.

Navarro en su trabajo "Una revisión de los métodos de Deep Learning aplicados a la detección automatizada de la Retinopatía Diabética", presenta un benchmarking usado en investigaciones y trabajos que se refieren a el diagnostico de retinopatia diabetica, en los cuales se cuentran imágenes para entrenamiento y prueba, en algunos casos se adjuntan etiquetas con descripciones o caracteristicas que se deben encontrar, con el fin de corroborar el procedimiento realizado.

A continuacionse mencionan algunos *Datasets* mas usados (Navarro, Peña, & Escorcia, 2020):

a) *DRIVE*: Consta de 40 fotografías de retina, de las cuales 7 muestran signos de RD temprana leve, el resto de las imágenes no presentan ninguna patología. El 50% de las imágenes son para entrenamiento, el porcentaje restante para pruebas, es decir 20 fotos.
b) *E-Optha*: Consta de dos categorías: *E-Optha* MA la cual posee 148 imágenes con microaneurismas y pequeñas hemorragias y 233 sin lesiones. *E-Optha* EX posee un conjunto de 47 imágenes de fondo de ojo con una segmentación de exudados y 35 imágenes sin ningún tipo de lesión.
c) Messidor: Este es uno de los *Dataset* más grande, ya que consta de 1200 imágenes de fondo de ojo. Las características de las imágenes pueden elegirse de entre tres tamaños, 800 imágenes con dilatación pupilar y 400 sin dilatación. Los formatos de las imágenes

son TIFF y se proporciona en un archivo de Excel los diagnósticos médicos de cada imagen.

d) *Kaggle*: una plataforma de competencia de ciencia de datos proporciona un conjunto de más de 80,000 imágenes, de las cuales 35,126 imágenes son públicas. El conjunto de datos consta de imágenes en los 5 niveles de Retinopatía Diabética.
e) *APTOS2019* BD (APTOS, 2019): una base que contiene 3662 imágenes de fondo de ojo, recabados de un hospital de la India.

Tabla 4.8 Matriz de referencias.

Fuente	Resumen	Metodología	Área de oportunidad
Implementation of deep learning-based algorithms for diabetic retinopathy classification from fundus images **(Lands, 2020)**.	Entrenamiento de los modelos de CNN ResNet y DenseNet para la clasificación de imágenes de fondo de ojo, como procesamiento de imagen uso de la resta del desenfoque gaussiano para el resaltado de las características, uso de aumentación de datos. El conjunto de datos utilizado es obtenido de Kaggle.	Entrenamiento del modelo utilizando un conjunto de datos balanceado. Ajuste del brillo, contraste y recorte de las imágenes como preprocesamiento en la aumentación de datos.	Resta del desenfoque gaussiano como procesamiento de imagen. Conjunto de datos de *Kaggle*.
Hybrid Retinal Image Enhancement Algorithm for Diabetic Retinopathy Diagnostic Using Deep Learning Model **(Hameed, 2022)**.	Clasificación de imágenes por Transferencia de Aprendizaje utilizando extracción de características, con un modelo de CNN basado en ResNet utilizando el conjunto de datos de MESSIDOR y *Kaggle*. Resta del desenfoque gaussiano y recorte de circunferencia como procesamiento de imagen.	Modelo personalizado de CNN basado en una arquitectura Resnet.	Uso de *Avgpooling* y *Dropout* para capa clasificadora del modelo de CNN. Recorte de circunferencia y fusión de imagen con matriz de desenfoque gaussiano.

Transfer-Ensemble Learning based Deep Convolutional Networks for Diabetic Retinopathy Classification. **(Ghosh, 2023)**.	Ensamble de dos modelos de CNN: InceptionV3 y VGG16 pre entrenados para la clasificación de imágenes del conjunto de datos de APTOS.	Ensamble de dos modelos de CNN clasificadores de CNN con el 25% congelado (no ajustable).	Ensamble de dos modelos de CNN. Tipo de entrenamiento: Ajuste fino del modelo base.
Transfer Learning approach for grading of Diabetic Retinopathy **(Aswathi, 2021)**.	Uso de arquitecturas pre-entrenadas: VGG19, ResNet e InceptionV3 para la clasificación binaria y multiclase de imágenes de fondo de ojo con procesamiento CLAHE y transformación POWERLAW para calidad de imagen.	Variación de valores de tamaño de *batch*, tasa de aprendizaje en el entrenamiento de diversos modelos. Cambio de optimizadores: (Adam y SGD) con diferentes tamaños de *batch*.	CLAHE como procesamiento de imagen. Transformación POWERLAW para calidad de imagen. Tamaño de *batch* variable.
Automatic Detection of Diabetic Retinopathy in Retinal Fundus Photographs Based on Deep Learning Algorithm **(Li, y otros, 2019)**.	Aprendizaje por transferencia profunda del modelo InceptionV3, procesamiento de imagen: CLAHE, validación de clases con cruce de datos.	Entrenamiento con y sin procesamiento de imagen para extracción de características.	Procesamiento de imagen: CLAHE.
Performance Evaluation of the NVIDIA® Jetson Nano™ Through a Real-Time Machine	Evaluación del rendimiento de la Plataforma NVIDIA® Jetson Nano™ y el *Dew Computing* aplicado al	Evaluación de recursos GPU y CPU, temperaturas del dispositivo, consumo de	Uso del algoritmo *YOLO*.

Learning Application. **(Valladares & Toscano, 2021)**.	conteo de vehículos terrestres en tiempo real.	energía y cantidad de RAM utilizada en la tarea del ML.	*Node JS* como interacción del usuario con el software.
Evaluation of Deep Learning Accelerators for Object Detection the Edge **(Puchtler & Peinl, 2020).**	Evaluación del rendimiento y el consumo de energía entre los dispositivos *Intel Neuronal Compute Stick 2, Google Coral Edge TPU y NVIDIA® Jetson Nano™ con la Raspberry PI 4.*	Comparación de características y comportamientos entre dispositivos.	Modelo de Red usado *MobileNet v2.*
Developing a Compressed Object Detection Model base don YOLOv4 for Deployment on Embedded GPU Platform of Autonomous System **(Jang, 2021).**	Ejecución del algoritmo YOLOv4 en una plataforma en línea que proporcione GPU para una mejor ejecución del algoritmo.	Evaluación del uso de memoria y rapidez, uso del procesador del sistema embebido en la ejecución del algoritmo.	Conexión con una plataforma online para une mejor ejecución del algoritmo de detección de objetos.
Promote Retinal Lesion Detection for Diabetic Retinopathy Stage Classification **(Wang, 2021).**	Detección de lesiones retinianas en imágenes de fondo de ojo y asignación de severidad en tres estados de acuerdo con la detección de los micro aneurismas.	Asignación de escala de severidad de acuerdo con cantidad de anomalías detectadas. Uso de modelo *InceptionNet V3* pre-	Modelo pre-entrenado: *ImageNet* V3 con imágenes de 299x299.

		entrenado de *ImageNet.* Y método propuesto.	
Severity Classification of Diabetic Retinopathy using an Ensemble Learning Algorithm through Analyzing Retinal Images **(Sikder, Masud, & Kumar Bairagi, 2021)**.	Diagnóstico de Retinopatía Diabética con Basado en la intensidad de y textura del nivel de gris, utilizando arboles de decisión.	Arboles de decisión con imágenes en escala de grises. *Data set* APTOS 2019.	Técnicas de procesamiento de imágenes a en escala y grises para resaltar características.
Classification of Different Stages of Diabetic Retinopathy using Convolutional Neuronal Network ***(Saranya & Umamaheswari, 2021).***	Clasificación de imágenes con un modelo de red personalizado con 64 capas, 32 filtros de 3x3.	35000 imágenes de *Kaggle* de 256x256. Evaluación de rendimiento: exactitud, sensibilidad y puntuación F.	*Kaggle Dataset*, Modelo de red neuronal personalizado.
Retinal Lesion Detection with Deep Learning Using Patches **(Lam & Yu, 2017).**	Desarrollo de un método para localizar y discernir múltiples tipos de hallazgos en imágenes retinales utilizando un conjunto limitado de datos de entrenamiento.	Conjunto de datos de Kaggle. Método de ventana deslizante. Validación con set de datos de *E-Ophta*.	Conjunto de datos de entramiento utilizado. Ventana deslizante. Modelo de red utilizado.
Automated Detection and Diagnosis of Diabetic Retinopathy: A Comprehensive	Comparativa de *Dataset* de imágenes utilizados para la detección y clasificación de RD, benchmarking de	Data sets utilizados y estudios con mejores métricas obtenidas.	*Dataset* utilizados*: Kaggle, APTOS, E-Optha*

Survey **(Lakshminaranan, Kheradfallah, & Jothi Balaji, 2021).**	data sets con estudios y valores de métricas obtenidos.		y uso de *Transfer Learning.*
Convolutional Neuronal Network for Diabetic Retinopathy Detection **(Firke & Jain, 2021).**	Propuesta de un modelo para la clasificación de imágenes de RD en dos casos.	Modelo de flujo para el diseño del algoritmo.	Flujo de proceso para el modelo.
Una revisión de los métodos de Deep Learning aplicados a la detección automatizada de la retinopatía diabética **(Navarro, Peña, & Escorcia, 2020).**	Las bases para la detección de la retinopatía diabética y las técnicas de *Deep Learning* utilizadas por diversos investigadores para el diagnóstico automatizado y el desempeño de los sistemas.	Resumen de trabajos que emplean Redes neuronales convolucionales, Maquinas de Soporte vectorial, métodos y más para la detección de RD.	Considerar las técnicas de los diversos autores para la selección de un algoritmo que sea eficiente en el sistema embebido.
Diabetic retinopathy Detection Using transfer Learning and Deep Learning **(Gangwar & Ravi, 2021).**	Hibrido de aprendizaje profundo para resolver el problema de detección automática de RD. Con aprendizaje por transferencia en Inception-ResNet-v2 entrenado, personalizando el modelo con bloques de capas CNN sobre ResNet-v2.	Personalización de los modelos y uso de *Transfer Learning*. Evaluación del rendimiento del modelo a través de múltiples data sets.	Aprendizaje por transferencia. Evaluación del rendimiento del modelo a través de múltiples *Dataset*.

Automated Detection and Diagnosis of Diabetic Retinopathy: A Comprehensive Survey **(Lakshminaranan, Kheradfallah, & Jothi Balaji, 2021).**	Enfoques de IA como ML y DL en clasificación y segmentación de 2016 a 2021 en imágenes para detectar la presencia de RD.	Enfoques de Machine Learning y Deep Learning.	Considerar las especificaciones de los data sets para el data set a utilizar en el entrenamiento del modelo.
Severity Classification of Diabetic Retinopathy Using an Ensemble Learning Algorithm through Analyzing Retinal Images **(Sikder, Masud, & Kumar Bairagi, 2021)**.	Desarrollo de un método para detección de RD basado en el nivel de intensidad de gris y las características de texturas extraídas de las imágenes de fondo de ojo.	El flujo se establece en tres etapas: Preprocesamiento de imágenes y datos, extracción y selección de características y Modelo de clasificación y optimización.	Analizar el flujo de trabajo del método de clasificación de la RD propuesto en el trabajo.
Mapeo de imágenes digitales de fondo de ojo atendiendo a rasgos de textura **(García García, Rodríguez Guillén, & Taboada Crispi, 2017)**.	Mapeo de 300 rasgos de textura en 150 imágenes de bases de datos para la localización del disco óptico y segmentación de vasos sanguíneos. Uso de descriptores estadísticos y algoritmos SIFT y SURF para la extracción de características relevantes.	Umbral, filtro gaussiano, histograma. Algoritmos SIFT y SURF.	Uso de herramientas como Python o Matlab. Contar con una buena calidad de imagen. Los rasgos de textura son sensibles ante los problemas de iluminación.

Reconocimiento de micropartículas de polen con algoritmos de procesamiento de imágenes implementadas en dispositivos reconfigurable **(Ruiz Varela, Ortega Cisneros, & Pedroza de la Crúz, 2016)**.	Se propone una metodología en hardware para la detección de micropartículas, a la medida de detección de bordes en imágenes a través de un FPGA.	Algoritmo de Canny para la detección de bordes. FPGA	Aceleradores de procesos para la reducción de tiempos.
Análisis del desempeño de redes neuronales profundas para segmentación semántica en hardware limitado **(Soto Orozco & Corral Sáenz, 2019)**.	Análisis cualitativo de los modelos de redes neuronales convolucionales Enet, Mobilenetv2, ERFNet, ESPNet v2, FastFCN. ERFNet genera buenos resultados en precisión por ser un modelo de baja profundidad.	Enet, Mobilenetv2, ERFNet, ESPNet v2, FastFCN.	Los modelos de red eficientes deben enfocar en mantener un equilibrio de filtros utilizados en cada capa.
Metodología de segmentación de la estructura ocular en imágenes de fondo de ojo de pacientes con retinopatía Diabética **(Madera, Gutierrez, & Gamarra, 2020)**.	Propuesta de una metodología de segmentación de vasos sanguíneos y disco óptico en imágenes de fondo de ojo. Uso del algoritmo CLAHE y BPDFHE para preprocesamiento de las imágenes.	Algoritmos CLAHE y BPDFHE para procesamiento de imágenes.	Desarrollo de un sistema CAD para soporte medico en el diagnóstico temprano de RD.

A Systematic Review of Deep Learning Methods Applied to Ocular Images **(Perdomo Charry & González Augusto, 2019)**.	Muestra de las enfermedades oculares especificando las patologías. Comparación de los rendimientos de los métodos aplicados a los data sets en las enfermedades oculares.	Métricas de rendimiento en modelos de aprendizaje profundo y data sets de imágenes oculares. ResNet, Inception-ResNet-V2, OCTNET	Implementar las métricas de rendimiento para la evaluación del sistema.
Boosting Convolutional Filters with Entropy Sampling for Optic Cup and Disc Image Segmentation from Fundus Images **(Zilly, 2015)**.	Proposición de un método basado en una red neuronal convolucional para la segmentación de disco y copa óptica. Entrenamiento de un clasificador de regresión logística.	Clasificador de regresión logística, softmax. y uso de técnicas de muestreo para reducir la complejidad computacional.	Considerar el clasificador de regresión logística.
Comparison of smartphone-based retinal imaging systems for diabetic retinopathy detection using Deep learning **(Karakaya & Hacisoftaoglu, 2020)**.	Comparación de los sistemas de imágenes de retina basados en teléfonos inteligentes disponibles en el mercado, así mismo la evaluación de la calidad de imagen y precisión de la detección automática utilizando *Deep Learning.*	Arquitecturas AlexNet y MatCovNet. Software *Matlab*.	Implementación de algunos de los sistemas de retina en el sistema embebido una vez que se termine la implementación de la red neuronal.
Optimization and acceleration of convolutional neuronal	Exploración de las técnicas recientes para la aceleración del entrenamiento de redes neuronales convolucionales	SGD (Descendiente de gradiente estocástico).	Implementar la aceleración de velocidad de entrenamiento con

networks: A survey **(Habib & Qureshi, 2020)**.	sin comprometer la precisión. Visión detallada sobre la aceleración de velocidad de cálculo utilizando la optimización *Stochastic Gradient Decent,* convolución rápida y desafíos del paralelismo en CNN.		SGD, ya que se tienen recursos limitados.
U-Net: Convolutional Networks for Biomedical Image Segmentation **(Ronneberger, Philipp, & Brox, 2015)**.	Aplicación de la arquitectura U.-Net para la segmentación de imágenes de un microscopio. Uso del descendiente del gradiente estocástico para el entrenamiento de la red.	U-Net, descendiente de gradiente estocástico.	Considerar la aplicación del descendiente de gradiente estocástico para el entrenamiento de la red.
A Hardware Accelerator for The Inference of a Convolutional Neuronal Network **(González, Villamizar Luna, & Fajardo Araiza, 2019)**.	Reducción de recursos de hardware en una placa *SoC* donde se utiliza un formato de punto fijo de 12 bits. El esquema de procesamiento opera a una velocidad de 100MHz que puede identificar alrededor de 441 imágenes por segundo.	Arquitectura Lenet-5, *Keras*, *TensorFlow*.	El hardware está basado en un procesador ARM y un acelerador de hardware obtener un mejor rendimiento de inferencia de la red neuronal.
Inteligencia Artificial con Aplicaciones a la ingeniería **(Ponce Cruz, 2010)**.	Descripción y ejemplificación de aplicaciones de la inteligencia Artificial para el análisis de datos,	Elementos que componen a la IA dentro de los cuales se encuentran tres grandes	Desarrollo de redes Neuronales para el

	imágenes, con el fin de obtener predicciones.	ramas: lógica difusa, redes neuronales artificiales y algoritmos genéticos.	procesamiento de imágenes.
Ascend IA Processor Architecture and Programming. Principles and Applications of CANN **(Liang, 2020)**.	Fundamentos de teóricos de la IA y el aprendizaje profundo, el estado de la industria, los procesadores de redes neuronales, marcos de aprendizaje profundo, métodos y prácticas de programación.	Rendimientos y atributos del procesador *AI Ascend* de Huawei. Elementos de la teoría de la IA, arquitectura del procesador.	Considerar el análisis de estudios detallados sobre datos y algoritmos de IA.
Thinking Machines **(Takano, 2017)**.	*Machine Learning* y sus aplicaciones, métodos de mejora de rendimiento, redes neuronales y su implementación en hardware.	Modelo de compresión, compresión numérica, aproximación, optimización.	Implementar la teoría de métodos de mejor de rendimiento en el sistema embebido.
Clasificación de imágenes en sistemas embebidos usando redes neuronales. **(Coccé, 2017)**.	Implementación de la librería Caffe sobre un ambiente Linux en una placa Orange Pi y Raspberry Pi para la clasificación de imágenes a través de redes neuronales.	AlexNet, Arboles, Rspberry, Orange Pi.	Considerar el uso de la placa Raspberry y el AlexNet para el desarrollo de la red neuronal.
Diseño de un sistema de detección de retinopatía diabética con procesamiento de	Procesamiento de imágenes de retina y uso de la red neuronal *Backpropagation* para la clasificación	*Backpropagation*, *Matlab*	Implementar la red neuronal propuesta para elegir un modelo

imágenes de retina **(Mamadi Noa, 2018)**.	de la enfermedad en sus diferentes etapas.		apropiado al sistema embebido.
Sistema de diagnóstico asistido por computadora para la detección de la retinopatía no proliferada usando la red neuronal de retro propagación **(Velázquez González, 2011)**.	Implementación de la red neuronal retro propagación para la clasificación de imágenes con RD en sus diferentes etapas de acuerdo con las patologías presentes en la retina.	Retro propagación. Algoritmos Python, C++	Conocer los métodos de implementación y desarrollo para su implementación en un sistema embebido.
Implementación y aceleración de algoritmos de IA sobre Raspberri Pi **(Abellán, 2021)**.	Desarrollo de la implementación del algoritmo YOLO en una placa y la complementación con una técnica de aceleración con Coral TPU para obtener un mejor rendimiento.	*YOLO,* *Coral TPU,* *OpenCV, TensorFlow 2.0.*	Aumento de la frecuencia base de la Raspberry pi de 1.5 a 2.0 GHz. Modificando el fichero de arranque.
Integración de redes neuronales en sistemas empotrados. Clasificación de imagen con Raspberry Pi **(Crespo, 2020)**.	Redes neuronales convolucionales en la placa para la clasificación de imágenes en una empresa con la problemática de identificación y etiquetado de objetos. se considera exitoso el proyecto en cuanto a su implementación en el hardware.	Sistema empotrado multi core, red neuronal convolucional,	Uso de lenguaje de programación Python y R. Aplicación de teoría y diseño de sistemas para la planeación del proyecto.

Optimización del producto matricial sobre dispositivos de bajo consumo para inferencia en Deep Learning **(Stabile Bernabé, 2021)**.	Optimización del producto matricial de las capas de entrenamiento e inferencia sobre una arquitectura ARM con procesador multinúcleo, se profundiza en la interacción entre algoritmos y arquitectura del procesador para determinar el rendimiento.	Arquitectura ARM, procesadores con arquitectura RISC, paralelismo con instrucciones SIMD.	El uso de procesadores ARM de arquitectura RISC proveen la capacidad de implementar paralelismo. Procesos multihilo y memoria compartida para la optimización del sistema.
Desarrollo de un Perceptrón Multicapa en Raspberry Pi **(Molina Ovando, 2019)**.	Implementación de un perceptrón multicapa como estrategia para la clasificación de patrones numéricos en un computador de placa reducida utilizando el lenguaje de programación Python.	Perceptrón multicapa.	Implementar más capas al modelo para elevar el nivel de dificultad de los patrones a reconocer.
Implementación de redes neuronales en Raspberry Pi con Movidius Neuronal Compute Stick **(Toro Valderas, 2020)**.	Red neuronal convolucional y el uso de tecnología de un microprocesador específico para acelerar la fase de inferencia, gracias a las Unidad de Procesamiento de visión. Se mejoran los tiempos de ejecución	Neuronal Compute Stick, Google Coral	El uso de estas tecnologías para la aceleración de la inferencia. Tener en cuenta los dispositivos embebidos

			compatibles con los aceleradores de IA.
Aceleración hardware para inferencia en redes neuronales convolucionales **(Pérez Cerdeira, 2021)**.	Las técnicas de *loop tiling y loop unrolling* disminuyen los accesos a memoria, lo que repercute favorablemente en el uso de recursos, potencia consumida y aceleración de inferencia.	Técnicas *Loop tiling, loop unrolling.* Implementación de MobileNet V2 en Hardware dedicado con uso de técnicas de HSL para explotar las ventajas del paralelismo.	*High-Level Synthesis* para la generación de aceleradores de hardware y facilitar el diseño e implementación sobre sistemas embebidos.
Desarrollo de un sistema de análisis de imágenes médicas basados en técnicas de Deep Learning **(Algarabel, 2019)**.	Comparación del rendimiento de métodos de clasificación de imágenes mediante DL. La ecualización adaptativa CLAHE mejora en los resultados.	CLAHE,	Cuanto mayor sea el *Dataset* el entrenamiento será mejor. El ensamble de varias redes pre-entrenadas da mejores resultados que por separado.
Clasificación de objetos usando aprendizaje profundo implementado en un sistema embebido **(Valderrama Molano, 2017)**.	Clasificación de imágenes con el uso de los *Frameworks TensorFlow*, *PyTorch* y *Caffe* con el uso de Redes Neuronales Convolucionales (CNN) de la arquitectura AlexNet y ResNet en	Uso de Frameworks para la programación de las redes neuronales. La librería *Caffe* presenta menos tiempo de	Usar CNN para la clasificación de imágenes para una mejor precisión y facilidad con la

	un sistema embebido NVIDIA® Jetson™ TX2.	ejecución en la etapa de entrenamiento.	implementación de los Frameworks.
Retinopatía Diabética en México **(Bueno-Garcia, Mira, & Flores Peredo, 2021)**.	Informe de estadísticas de Diabetes y Retinopatía Diabética en México, factores, así como los métodos de estudio y clasificación de RD.	Análisis de datos estadísticos proporcionados por la IDF además de un asesoramiento por especialistas en oftalmología, con un análisis especifico en México.	Considerar y basarse en datos proporcionados por oftalmólogos especializados en RD.

4.5. Análisis y discusión del estado del arte

Con base en el análisis de la literatura es demostrado que existen diferentes técnicas de inteligencia artificial para la generación modelos clasificadores de retinopatía diabética. Este proyecto al tratarse del análisis y tratamiento de imágenes, son utilizadas las técnicas de Inteligencia Artificial que permitan el tratar a los pixeles de las imágenes como datos, por ello es por lo que el aprendizaje profundo es el mejor candidato para utilizar. Al existir diversas arquitecturas de redes neuronales convolucionales, en las que una de sus principales características que los distingue son el número de capas de profundidad.

El enfoque de este proyecto es la inferencia de modelos de redes neuronales convolucionales para la clasificación de imágenes de fondo de ojo e identificar el nivel de retinopatía diabética en la que se encuentra. Diversos modelos de CNN han sido utilizados, sin embargo, estos modelos utilizan una gran cantidad de imágenes para su entrenamiento, por ello los proyectos analizados optan por la técnica de transferencia de aprendizaje.

Los valores establecidos de las métricas de rendimiento para la; precisión, exactitud, sensibilidad, especificidad y área bajo la curva, se estiman al menos del 80% en todos los casos y con base en la literatura. Cabe resaltar que es un problema de clasificación de tipo multiclase, con cinco tipos correspondientes a los niveles de Retinopatía Diabética, mediante transferencia de aprendizaje, con modelos previamente entrenados tareas de clasificación.

Para el dispositivo despliegue, se basa en computadores con componentes de alto rendimiento, en otros casos utilizan computadoras básicas para la adquisición de los datos y sus predicciones en línea. Autores como Coccé (2017) y Crespo (2020) han utilizado los sistemas embebidos para la ejecución de modelos de Inteligencia Artificial, específicamente con redes neuronales que han resultado efectivos.

Por tal motivo, considerando los aspectos de la literatura referente a el aprendizaje profundo para la clasificación de imágenes y su implementación en sistemas embebidos, se plantea el desarrollo de un sistema de clasificación de imágenes de fondo de ojo en los cinco niveles de retinopatía diabética, utilizado redes neuronales convolucionales y la interpretación del modelo clasificador en un sistema embebido. Considerando la técnica de transferencia de

aprendizaje para el entrenamiento del modelo de red convolucional y una SOM como los es la NVIDIA® Jetson Nano™ como sistema final de ejecución.

5. MÉTODO

En el presente capitulo es descrita la metodología empleada para el desarrollo del proyecto que permite la clasificación de imágenes de fondo de ojo, en los cinco niveles de retinopatía diabética utilizando la transferencia de aprendizaje en un modelo de red neuronal convolucional e interpretado en un sistema embebido de la plataforma NVIDIA® Jetson Nano™. Además, se detallan los requerimientos de hardware y software para la realización del sistema embebido de clasificación.

5.1. Requerimientos

5.1.1. Software

El lenguaje de programación seleccionado por su versatilidad es Python, complementando desarrollo del sistema con las siguientes librerías para la manipulación y visualización de los datos:

a) Numpy
b) Pandas
c) os
d) plotly
e) matplotlib
f) PIL
g) OpenCV
h) Scikit-learn

En primer lugar, cabe señalar que, al tratarse de un análisis de una gran cantidad de datos, las librerías *Numpy* y *Pandas* ejecutan manipulación a través de Arreglos y Matrices, que son implementados como *DataFrames* y Vectores utilizados por las bibliotecas seleccionados para el desarrollo del modelo. En segundo lugar, la lectura de los datos a través de archivos en carpetas, la librería OS permite la obtención de los directorios en los que realiza el proyecto. Así, una mejor manera de interpretar los datos es a través de una forma gráfica, por ello *plotly* y *matplotlib* son utilizados para la visualización de los datos a través de gráficos.

Para el procesamiento de imágenes son utilizadas las librerías *PIL* y *OpenCV*, las cuales tienen algoritmos desarrollados que permiten hacer uso de ellas de una manera fácil y rápida, además de ser compatibles con el lenguaje de programación. De manera general la técnica de inteligencia artificial que es utilizada en este proyecto es el Aprendizaje Automático: *Scikit-learn,* una biblioteca de aprendizaje automático que proporciona herramientas para el ajuste de modelos y la manipulación de los datos. Específicamente para el desarrollo del modelo clasificador son utilizados las siguientes bibliotecas compatibles con el lenguaje de programación *Python*: *TensorFlow* y *Keras.* En primer lugar, se ha seleccionado *TensorFlow* para el desarrollo del modelo clasificador, que permite el uso de la API *Keras*, lo cual facilita el desarrollo de modelos de redes neuronales. Además de proporcionar modelos pre-entrendos e implementar la transferencia de aprendizaje. En segundo lugar, el desarrollo a nivel software a través de una plataforma en línea que permite la ejecución lenguaje seleccionado para el desarrollo de proyecto: *Google Colab*, el cual proporciona un entorno de ejecución en línea y uso de herramientas de tipo hardware como lo son GPU y librerías previamente instaladas, *TensorFlow* y *OpenCV* por mencionar algunas.

5.1.2. Hardware

El desarrollo del modelo requiere de un sistema de cómputo con los requerimientos mínimos, la plataforma seleccionada para el desarrollo cuenta con las herramientas necesarias, sin embargo, un requerimiento indispensable del sistema de cómputo es la conexión a internet, debido a que la plataforma no realiza la ejecución si no se encuentra el usuario activo.

Para el despliegue del modelo final se establece en un sistema embebido, especialmente en una computadora de placa única de la plataforma NVIDIA® Jetson Nano™, se cuenta con dos versiones en donde la característica que la distingue una de otra es el espacio en memoria RAM de 4GB. El Sistema en Módulo NVIDIA® Jetson Nano™ necesita de accesorios para su funcionamiento (véase en la Figura 5.1), los cuales se mencionan a continuación:

1. Tablero portador (*Carrier Board,* por su traducción al inglés).
2. Memoria MicroSD (Clase 10): con un espacio mínimo de 60 GB, aquí se alojará el Sistema Operativo y los archivos generados.
3. Monitor: pantalla con conexión HDMI o *DisplayPort*.
4. Teclado con conexión USB.

5. Mouse con conexión USB.
6. Conexión a internet a través de cable Ethernet.
7. Fuente de alimentación (5V-12A) con cable de salida USB-C.

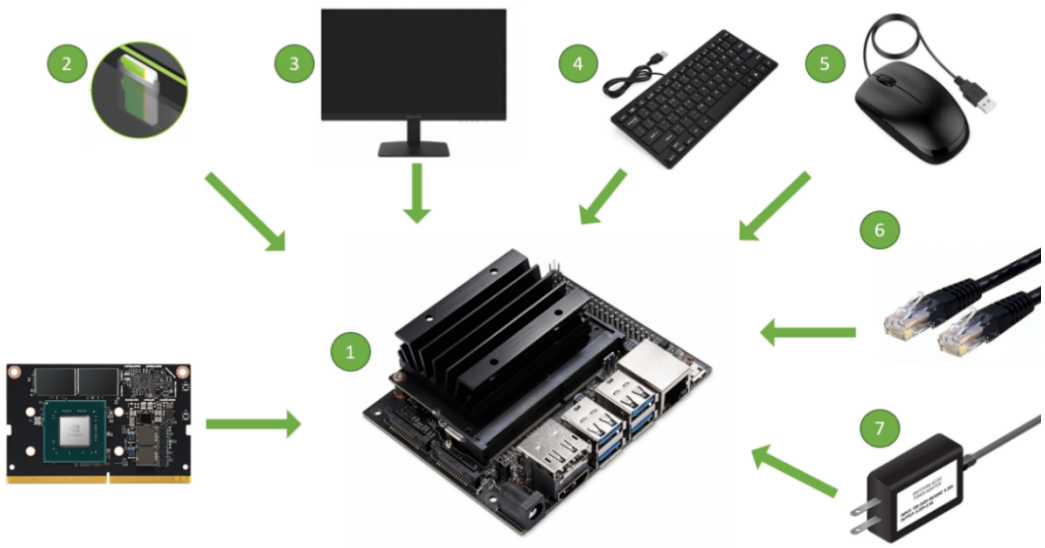


Figura 5.1 Hardware (accesorios) para el dispositivo de despliegue del proyecto. Fuente: Elaboración propia.

5.1.3. Diagrama de requisitos

El proceso general se basa en el flujo de IA, en el diagrama de la Figura 5.1 se muestra las etapas del flujo de IA y los requisitos en cada una de ellas.

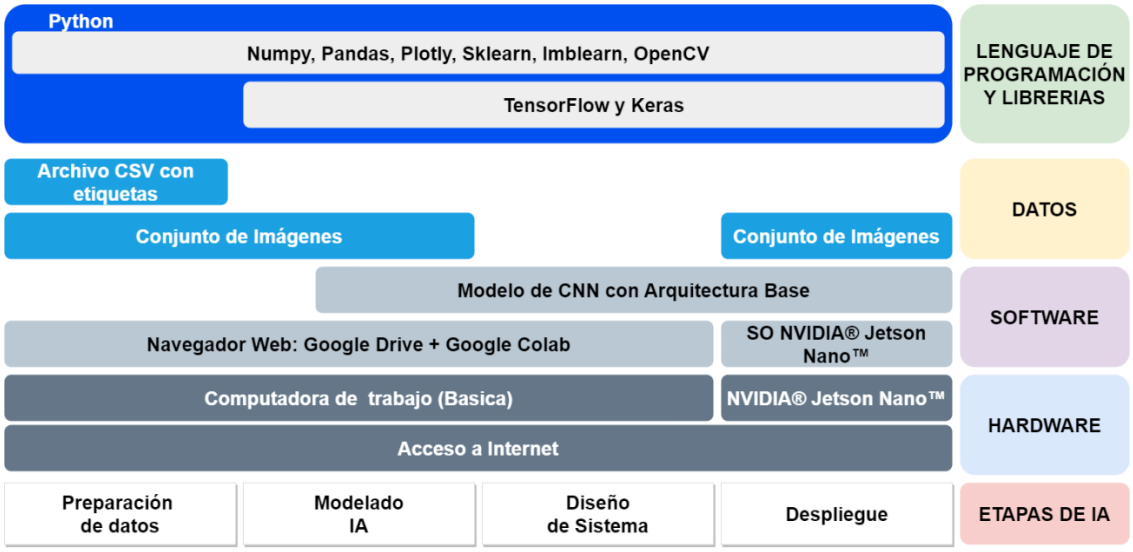


Figura 5.2 Diagrama de requisitos por etapas del flujo de IA. Fuente: Elaboración propia con base en **(MathWorks, 2023)**.

5.1.4. Diagrama de flujo de funcionamiento

El despliegue del modelo clasificador se realiza en un Sistema en Módulo (*System on Module*, por su traducción al inglés) NVIDIA® Jetson Nano™, el cual tiene como entrada una imagen a color, la cual es procesada antes de pasar a la Red Neuronal Convolucional. Como salida se obtiene la clase a la que el modelo ha clasificado (Véase en Figura 5.3).

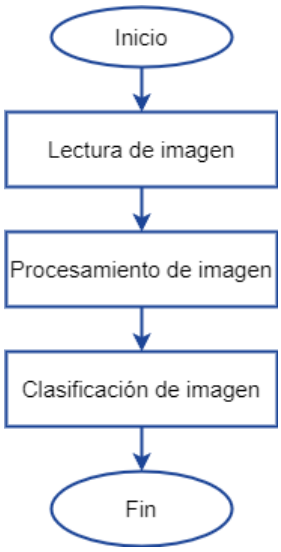
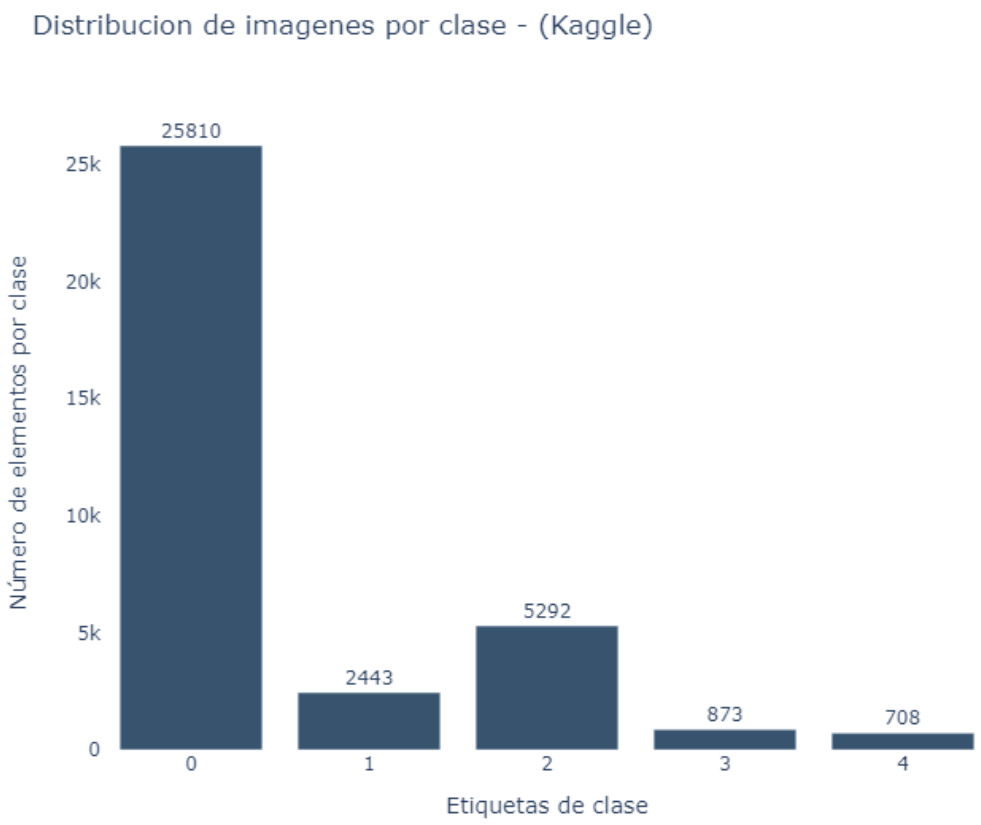


Figura 5.3 Diagrama de flujo de funcionamiento. El sistema final se ejecuta en el SOM de la plataforma de NVIDIA® Jetson Nano™.

5.2. Análisis y exploración de datos

El *Dataset Kaggle Diabetic Retinopathy* proporcionado por EyePACS, disponible en: *Kaggle.com*, proporciona un archivo CSV con las etiquetas de clase del conjunto de imágenes, la Gráfica 5.1 muestra la distribución de los datos, con un total de 35125 imágenes, distribuidas en las 5 clases de Retinopatía Diabética. El análisis de los datos se realiza con los datos tabulares que se proporcionan en el archivo CSV y no directamente con las imágenes, esto con ayuda de la librería de *Pandas* utilizando *DataFrames*.



Gráfica 5.1 Distribución de los datos Kaggle. El eje X representa los niveles de Retinopatía Diabética (clases: 5), el eje Y representa la cantidad de imágenes en cada nivel.

La Gráfica 5.1 presenta la distribución de las imágenes, se observa que se tiene un desbalanceo de datos, como clase predominante se encuentra el grupo de imágenes con la etiqueta 0 (R0: No RD) es decir en el set de datos predominan imágenes de fondo de ojo sanos, el 26.5% restante de las imágenes se encuentra distribuido en las 4 clases restantes.

Antes de un procesamiento digital de las imágenes son analizadas las distribuciones de los datos por clase, pues estos datos representan la información con la que la red neuronal convolucional extrae las características.

Para el análisis particular de los datos establecemos la visualización de imágenes de cada clase, identificar las anomalías y verificar el correcto etiquetado de las imágenes. Se tiene en cuenta las características geométricas de las anomalías que genera la retinopatía diabética, distinguir primeramente entre un ojo sano y un ojo enfermo, partiendo del ojo enfermo establecer el nivel en el que se encuentra.

Para la visualización de los datos que corresponde al grupo de imágenes de ojos sanos, los cuales tienen asignada la etiqueta con el valor 0, no se presentan anomalías, pues no se observa la presencia de anomalías como exudados o hemorragias, como se muestra en la Imagen 5.1.

Imagen 5.1 Imagen aleatoria extraída del conjunto de datos de Kaggle, valor de etiqueta: 0 (No RD, ojo sano) **(Kaggle, 2015)**.

La segunda clase R1: leve, véase en la Imagen 5.2, etiquetada con el número 1, esta etapa es considera como el inicio de la retinopatía diabética, se presentan exudados que son difíciles de distinguir, es necesario la ampliación de la imagen para su identificación, los exudados se representan como los pequeños puntos de color amarillo en el fondo de ojo.

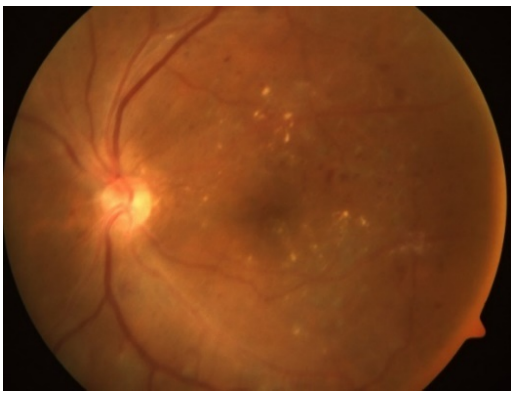


Imagen 5.2 Imagen de fondo de ojo etiquetada con la clase 1 (R1: Retinopatía Diabética Leve) **(Kaggle, 2015)**.

La clase R2 – Moderado, presenta una cantidad mayor de anomalías con respecto a la clase R1, para esta clase puede tener la misma cantidad anomalías, sin embargo, el área puede ser mayor, lo que significa que se tiene un agrupamiento de anomalías, sin realizar un

acercamiento a la imagen se observa la presencia de exudados en la parte superior del fondo de ojo (véase en Imagen 5.3).

Imagen 5.3 Imagen de fonde de ojo de la clase 2 (R2: Moderado) **(Kaggle, 2015)**.

La cuarta clase R3: severa, tiene presencia de exudados en mayor cantidad, en algunas zonas se tiene presencia de hemorragias (figuras geométricas con un color rojizo), elementos que pueden ser identificados inmediatamente (véase en Imagen 5.4).

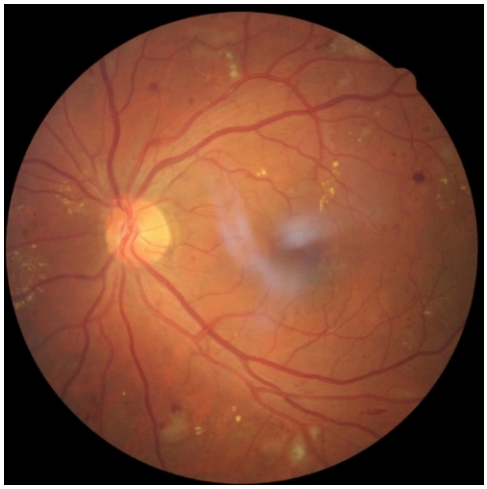


Imagen 5.4 Imagen de fondo de ojo de la clase 3 (R3: severa) **(Kaggle, 2015)**.

Clase: R4: proliferada, a simple vista se nota un ojo enfermo, esta es la etapa más avanzada, por ello es por lo que a simple vista se observa la presencia de las anomalías a través de sus características geométricas (objetos de diversas formas geométricas de color café) (Véase en Imagen 5.5).

Imagen 5.5 Imagen de fondo de ojo de la clase 4 (R4: proliferada) **(Kaggle, 2015)**.

La base de datos de Kaggle muestra por paciente dos imágenes de fondo de ojo, una para el ojo izquierdo y otra imagen para el ojo derecho, para este análisis no se considera el que la imagen pertenezca a un paciente, únicamente se realiza la clasificación por imagen de fondo de ojo.

El objetivo del análisis previo se concentra en el análisis de la estructura geométrica del fondo de ojo y las anomalías presentes para establecer un procesamiento de imagen que realice el resaltado de las características geométricas. En el conjunto de datos se encuentra con imágenes que no tienen un enfoque suficiente o que incluso no se tiene una visibilidad del fondo de ojo debido al brillo y otros factores más, aspectos que podrían afectar en el entrenamiento del modelo, por lo que se realiza un filtro de estas imágenes.

El procedimiento para realizar el filtrado de las imágenes se realiza en dos pasos: como primer paso la separación de las imágenes en carpetas correspondientes a su clase etiquetada, el conjunto total de las imágenes se encuentra en una sola carpeta. La separación de las imágenes se basa en la etiqueta y el nombre de la imagen que se encuentran en el archivo proporcionado por la plataforma, por ello a través de un módulo de *Python*, en el que se genera la ruta de la imagen y dependiendo de la clase en la que se encuentra etiquetada se realiza el movimiento de la imagen.

Al final de este proceso se obtienen 5 directorios correspondientes a cada una de las clases de Retinopatía Diabética, dentro de estos directorios se encuentran las imágenes etiquetas. Como segundo paso se realiza una exploración de las carpetas y al mismo tiempo un filtro

de imágenes que no tienen un enfoque suficiente o un bajo brillo, debido a que no proporcionan información suficiente para que el modelo extraiga las características de la clase a identificar. Ejemplo de imágenes que se han filtrado se muestra en la Imagen 5.6, esta imagen muestra una fotografía de fondo de ojo, en la que no se tiene el brillo suficiente para identificar la circunferencia del fondo de ojo, la circunferencia es muy poco apreciable.

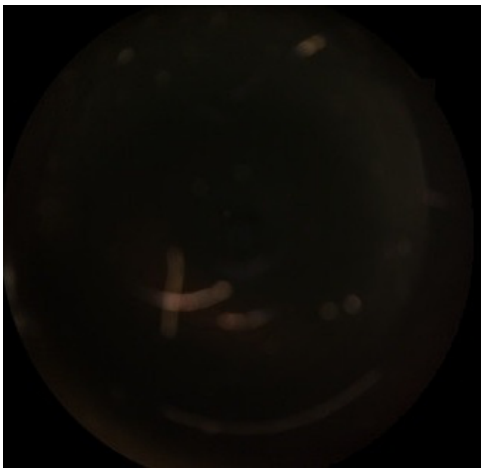


Imagen 5.6 Imagen de fondo de ojo no visible **(Kaggle, 2015)**.

También se eliminan del conjunto de imágenes las que presentan un brillo muy intenso, aun aplicando un preprocesamiento de imágenes para ajustar esta característica la cantidad de imágenes que necesitan de este preprocesamiento en mínima. El preprocesamiento de los datos se aplica de manera general a todas las imágenes, el preprocesamiento no es suficiente para imágenes en específico como la que se muestra en la Imagen 5.7.

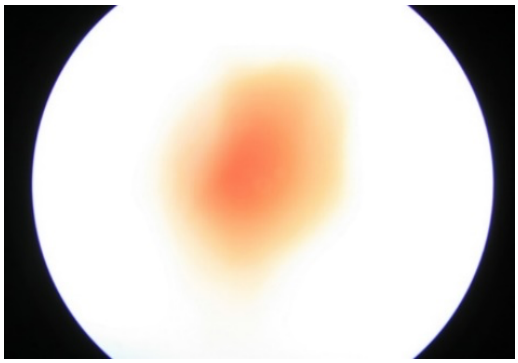


Imagen 5.7 Imagen de fondo de ojo con brillo intenso **(Kaggle, 2015)**.

Otra característica que se encuentra en el conjunto de imágenes son las imágenes alteradas en cuanto a la forma de la circunferencia, en este caso las imágenes se muestran recortadas,

o con figuras que parecieron ser recortadas manualmente, la figura de la circunferencia no se muestra completa, como se muestra en la Imagen 5.8.

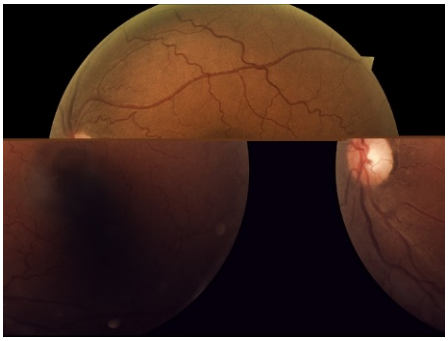


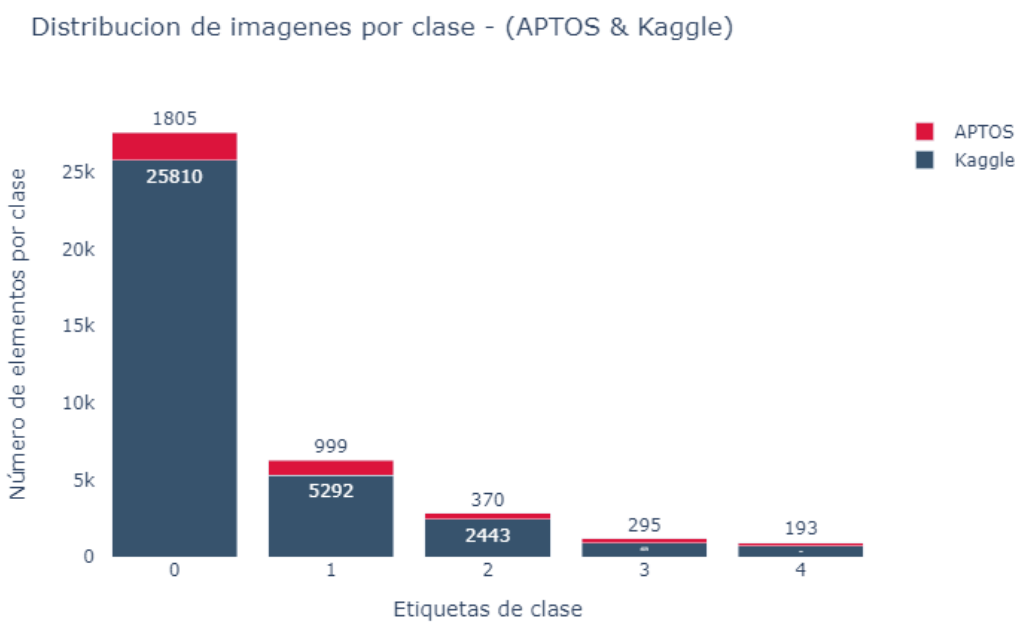
Imagen 5.8 Imagen de fondo de ojo con circunferencia no completa **(Kaggle, 2015)**.

De forma general se encontró que la mayor parte de las imágenes se encuentran desenfocadas, lo que obliga a establecer un preprocesamiento de imagen en el cual se tenga un realce de características, aun teniendo en cuenta esto, existen imágenes en las que el enfoque es muy bajo.

5.3. Balanceo de datos

Para este proyecto se utilizaron dos conjuntos de imágenes, en primer lugar, el conjunto de imágenes de *Kaggle*, la distribución de clases se muestra en la Gráfica 5.3. Con el objetivo de aumentar el número de imágenes en las clases minoritarias se anexan las imágenes del conjunto de datos de *Society, Asia Pacific Tele-Ophthalmology* (APTOS, 2019) disponible en: asiateleophth. A partir de aquí se obtiene un conjunto de datos mayor, cada una de estas clases y su número de imágenes se muestra en la Gráfica 5.2, el número de imágenes de *APTOS* es mucho menor en comparación con el número de imágenes de *Kaggle*.

Sin embargo, el conjunto de datos de *APTOS* presenta una base de imágenes más enfocadas, por lo que en un segundo entrenamiento se considera el uso únicamente de esta base.



Gráfica 5.2 Distribución de imágenes del conjunto de APTOS y Kaggle.

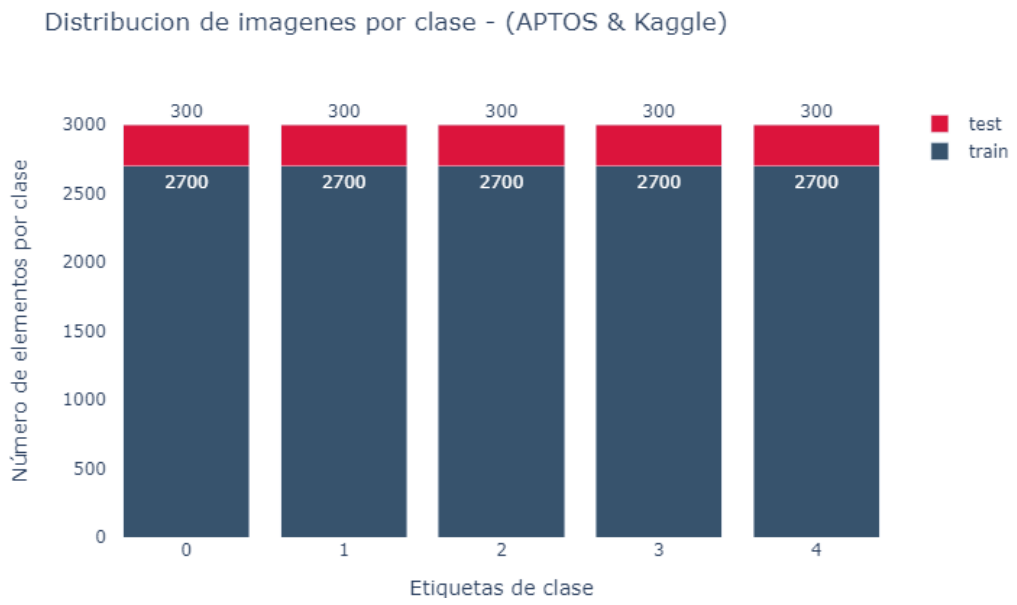
Antes de realizar el balanceo de datos, el conjunto de imágenes se divide en 2 partes: entrenamiento que corresponde al 80% del total de imágenes, mientras que para el conjunto de prueba se considera el 20% restante. Dentro del conjunto de imágenes de entrenamiento se establece el 20% para validación. El porcentaje de distribución de los datos entre cada clase es muy extenso, sobre todo por la cantidad de imágenes de la clase mayoritaria, la cual representa el 75% del total de conjunto de datos. Con lo anterior se establecen dos métodos de balanceos de datos: sobre muestreo (para aumentar el número de imágenes de las clases minoritarias) y submuestreo (para disminuir el número de datos de la clase mayoritaria).

Para el tratamiento de datos desbalanceados, *Scikit learn* proporciona una herramienta para el balanceo de datos: *imbalanced-learn*. Con esta herramienta se implementa el balanceo de los datos, a utilizando del nombre de las imágenes y la etiqueta correspondiente que se encuentran en el archivo *csv* del conjunto de datos de *Kaggle* y *APTOS*.

El conjunto de datos se divide en dos subconjuntos, el primer conjunto para los datos de entrenamiento, equivalente al 80% del total de datos y el segundo para los datos de prueba correspondiente al 20%. Cabe mencionar que el porcentaje de cada subconjunto de datos no corresponde a la cantidad de las imágenes de ambos conjuntos de datos, ya que se estableció

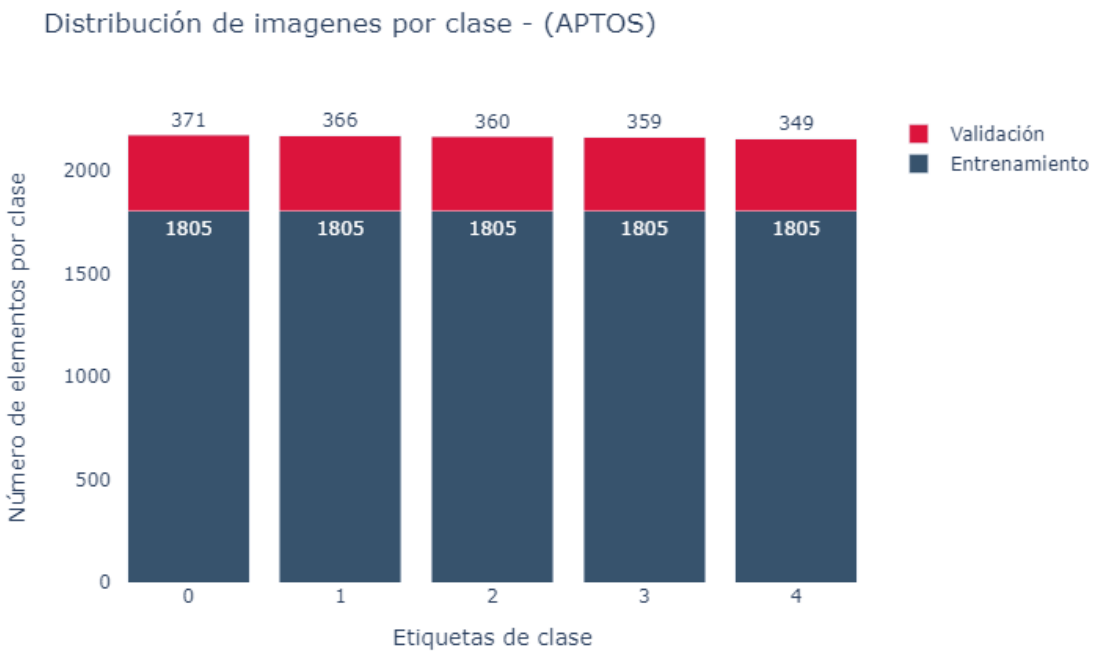
a través de la herramienta de *imbalanced-learn* obtener 3,000 imágenes por cada clase, 15,000 imágenes en total.

En la Gráfica 5.3 se representa la distribución balanceada de los datos, para el conjunto de datos de entrenamiento se tiene un total de 2,700 imágenes por cada clase, 300 imágenes son utilizadas para el conjunto de prueba.



Gráfica 5.3 Distribución del conjunto de datos balanceado.

El conjunto de datos utilizado en el entrenamiento del modelo con una arquitectura base DenseNet201, el conjunto de datos se obtiene a través de un balanceo utilizando sobre muestreo y submuestreo aleatorio para las imágenes del conjunto *APTOS*, por lo que el número de imágenes para este conjunto es menor. Para el balanceo de datos se utiliza los dos métodos que el conjunto de datos anterior. En la Gráfica 5.4 se observa la distribución de los datos, del total de imágenes de clase se establece las 9,025 imágenes en total, estableciendo el 80% para entrenamiento y el 20% restante para validación.



Gráfica 5.4 Distribución de clases después de sobre muestreo. Conjunto de datos de APTOS.

5.4. Preprocesamiento de datos

El objetivo del procesamiento de los datos consiste en el resaltado de las características geométricas de las anomalías en las imágenes, considerando el desenfoque que se ha encontrado en la exploración de datos.

Para el primer conjunto de datos que consta de 15,000 imágenes únicamente se aplicó el ajuste de la circunferencia y el resaltado de las características con el procesamiento de Fusión de la imagen con el Filtro Gaussiano. Con los resultados obtenidos posterior al primer entrenamiento se establece agregar un método de procesamiento más al segundo conjunto de imágenes, como se muestra en la Figura 5.4, se establecen cuatro etapas de procesamientos a cada una de las imágenes, que van desde el resaltado de características hasta el cambio de tamaño de la imagen. Este proceso se aplica únicamente al conjunto de datos de *APTOS2019*.

Figura 5.4 Flujo del procesamiento de imágenes.

El ajuste de la circunferencia consiste en la eliminación de la zona fuera de la circunferencia que pertenece a la retina, con el objetivo de ajustar la retina en todo el espacio de la imagen. Este proceso se considera al observar que la mayoría de las imágenes se encuentran con zonas negras, la circunferencia del fonde de ojo no logra llenar por completo la imagen.

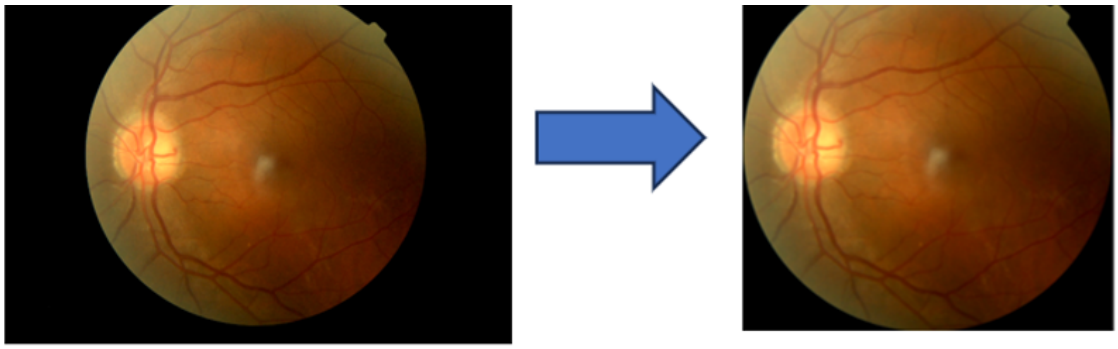


Imagen 5.9 Ajuste de circunferencia de fondo de ojo.

Como se muestra en la Imagen 5.9, la circunferencia de la retina ocupa la mayor parte de la imagen, contemplando que el tamaño final de la imagen es de 224x224px, no permite obtener una imagen cuadrada. Posterior al ajuste, la circunferencia del fondo de ojo se ajusta al tamaño de la imagen, aquí también se modifica el tamaño real de la imagen, aunque no son las dimensiones finales de la imagen.

El procedimiento del ajuste de la circunferencia se realiza con el Seudocódigo 1, donde recibe como parámetros de entrada la imagen y un valor de tolerancia, este último valor se utiliza para determinar el valor del umbral para los pixeles a considerarse. Para el funcionamiento del algoritmo se trabaja con una imagen en escala de grises, utilizando la matriz correspondiente de la cual se extrae una nueva matriz con los pixeles que superen el valor de umbral. El código que corresponde al seudocódigo 1 se puede consultar en el Anexo B.

Seudocódigo 1 Recorte de espacios en negro de imagen **(Hameed, 2022)**.

```
Inicio
    Tolerancia= 7
Si imagen.ndim == 2 Entonces
    Mascara = imagen > tolerancia
    Imagen segmentada = imagen[mascara.any(1), mascara.any(0)]
Si no
    Imagen en escala de grises = convertColor(imagen, RGB2GRAY)
    Mascara = imagen > tolerancia
    CanalRojo = imagen[:,:,0][mascara.any(1), mascara.any(0)]
    CanalAzul = imagen[:,:,1][mascara.any(1), mascara.any(0)]
    CanalVerde = imagen[:,:,2][mascara.any(1), mascara.any(0)]
    Imagen segmentada = numpy.stack([CanalRojo, CanalVerde, CanalAzul])
Fin Si
Final
```

Hammed, S. (2022) implemento en su trabajo "*Hybrid Retinal Image Enhancement Algorithm for Diabetic Retinopathy Diagnostic Using Deep Learning Model"* el proceso de la fusión de fusión de la imagen original y la misma imagen con el procesamiento de operación gaussiana, se obtiene una imagen con un resaltado de las características.

El desenfoque gaussiano es un método para suavizar una imagen mediante el uso de una función gaussiana para reducir el nivel de ruido, asi como detalles insignificantes. Se logra convolucionando una imagen con un núcleo Gaussiano 2-D que se expresa como (Siddharth, Hao, & Jiabo, 2020):

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x2+y2}{2\sigma2}} \quad (1)$$

Donde σ es la desviación estándar de la distribución gaussiana, que determina el alcance del efecto del desenfoque alrededor de un píxel, con ubicación en *(x, y)*.

Los valores de esta función crean el núcleo de convolución que se aplica a cada píxel de la imagen original. *OpenCV* realiza este proceso mediante el método `cv2.GaussianBlur` el cual recibe tres argumentos: la imagen, el delta del píxel central y la desviación estándar. El valor de la desviación estándar influye en que tan significativo los pixeles del píxel central afectan el resultado de los cálculos (Sharda, 2021).

El algoritmo para la aplicación del desenfoque Gaussiano recibe cuatro variables: sigma que representa el valor de la operación gaussiana, gamma que representa el valor del peso estático que se sumara a todos los pixeles de la imagen, alfa y beta que representan el peso a ser considerado al realizar el cegamiento (Hameed, 2022). Sin embargo, en su trabajo el filtro gaussiano establece la operación del filtro gaussiano en la imagen en escala de grises, por lo que el resultado obtenido es una imagen en escala de grises con una combinación lineal de la imagen con el filtro gaussiano.

La combinación lineal para la combinación de imágenes en *OpenCV* podemos hacer uso de este proceso a través del método `cv2.addWeighted(src1, alfa, src2, beta, gamma)`, la cual es una función que está definido por la ecuación 2:

$$dst = \alpha * src1 + \beta * src2 + \gamma \tag{2}$$

Donde la variable $src1$ corresponde a la primera matriz de entrada, α: el peso de los primeros elementos de la matriz, $src2$: segunda matriz de entrada, β: peso de los de los elementos de la segunda matriz y γ: el escalar agregado a cada suma (OpenCV, 2023).

El valor de la segunda matriz de entrada se establece a la matriz resultado de la operación gaussiana, es decir la imagen con el filtro gaussiano aplicado, con esto se realiza una fusión entre dos imágenes en donde se obtiene como resultado una imagen con un resaltado de características, con los aspectos de las características geométricas de toda la imagen con mayor nitidez. En la Figura 5.5 se muestra el proceso de esta operación.

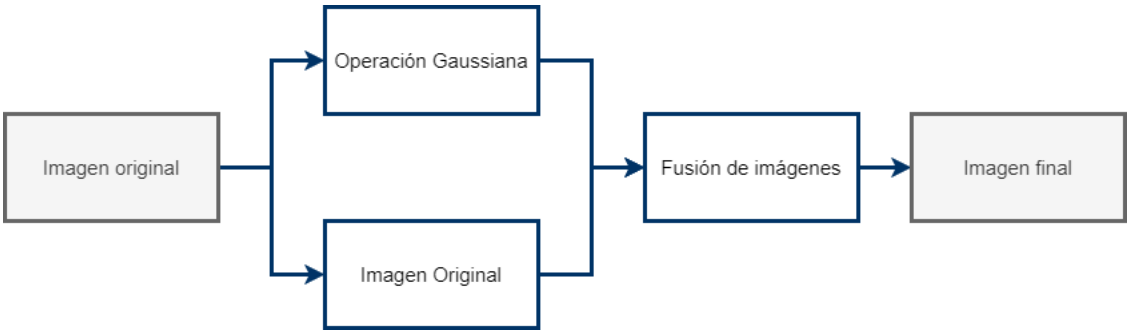
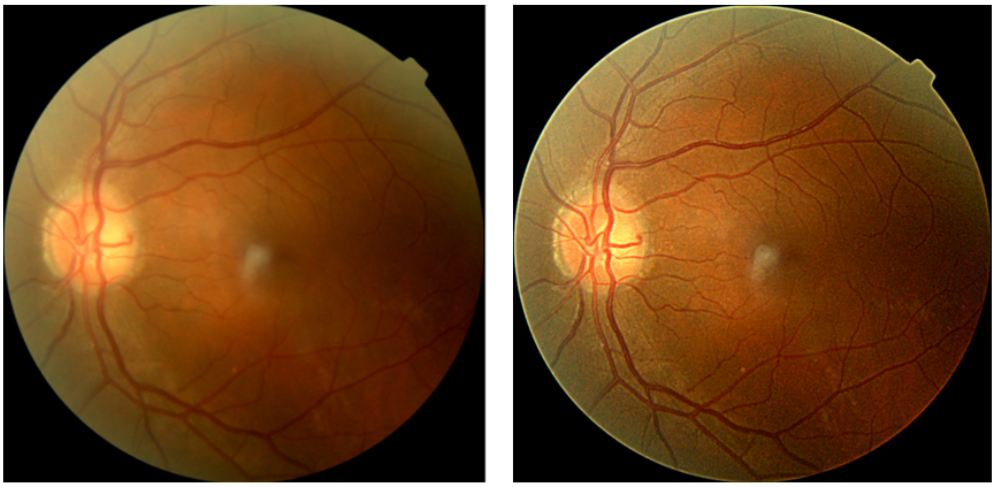


Figura 5.5 Flujo de procesos para el realce de características utilizando fusión de imágenes y operación gaussiana.

Este proceso ha resultado efectivo para el realce de las características sin alterar el color de la imagen, las imágenes que parece tener un desenfoque resultan un enfoque de sus características. En la Imagen 5.10 se muestra un ejemplo del procesamiento antes y después la fusión de una imagen original y una imagen con operación gaussiana.



a) *b)*

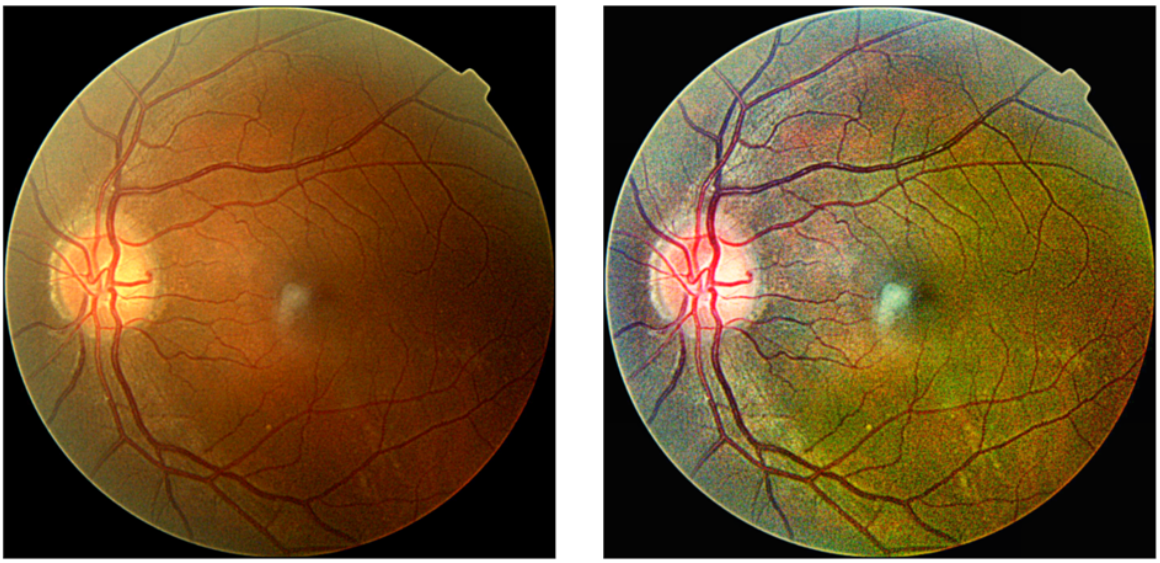
Imagen 5.10 Imagen de fondo de ojo antes a) y b) después del procesamiento de fusión entre la imagen original y una operación gaussiana.

El valor de los parámetros para el método `addWeighted` se establecen con $\alpha = 5$, $\beta = -4$ y $\gamma=0$. Estos valores son definidos a prueba y error, pues se establecieron diferentes combinaciones para llegar a estos valores. Con *Pyhton* y *OpenCV* este proceso se puede resumir en una línea de código como se define a continuación:

```
def addGaussianBlur(img):
    return  cv2.addWeighted(img, 5, cv2.GaussianBlur(img, (0,0), 4), -4, 0)
```

Adicionalmente para mejorar el contraste de las imágenes se utiliza la ecualización del histograma adaptativo de contraste limitado (*CLAHE,* por sus iniciales en inglés – *Contrast Limited Adaptative Histogram Equalization*) en los tres canales de la imagen y el tamaño de los mosaicos de 8x8, esto para cada una de las capas de color RGB.

Este método es utilizado ya que se detectó que en algunas imágenes el contraste no es fijo para toda el área de la imagen, sin embargo, a partir de este preprocesamiento de imagen se nota un cambio en los colores de la imagen, pero a su vez se notan mejor las características de la imagen. En la Imagen 5.11 se muestra el antes y después del procesamiento con *CLAHE* a una imagen de fondo de ojo.



a) b)

Imagen 5.11 Imagen de fondo de ojo con CLAHE. a) antes de procesamiento, b) después de procesamiento.

El método para aplicar la ecualización del histograma adaptativo que proporciona *OpenCV* está diseñado para procesar una imagen a blanco y negro. El proyecto utiliza imágenes a color, la cuales están en RGB, por lo que se aplica el procesamiento a cada una de las capas de color, para lo cual se hace uso del seudocódigo 2. El código en *Python* correspondiente se encuentra en Anexo C.

Seudocódigo 2 Ecualización del Histograma Adaptativo para imágenes en RGB

```
Inicio:
        modeloCLAHE = CLAHE (umbral=3, ventana=(8,8))
        CapaR = modeloCLAHE(image[:,:,0])
        CapaG = modeloCLAHE(image[:,:,1])
        CapaB = modeloCLAHE(image[:,:,2])
        Imagen = numpy.stack([CanalRojo, CanalVerde, CanalAzul])
Fin
```

Al final de los procesos, se encuentra el redimensionamiento del tamaño de la imagen, este se realiza al final con el objetivo de considerar la mayor parte de características, pues si este proceso se realiza antes de los procesamientos de filtros.

Las imágenes finales se almacenan en Google Drive, aquí se establecen las carpetas con cada una de las clases, las imágenes ya procesadas. Esta plataforma puede conectarse con *Google Colab* para acceder a los datos sin la necesidad de cargarlos o subirlos a la plataforma en el momento de realizar el entrenamiento. Este proceso se realiza ya que *Google Colab* elimina todos los datos después de cerrar el explorador.

Como parte del procesamiento de imágenes se considera establecer un porcentaje del total de imágenes para entrenamiento y otro porcentaje para validación del modelo clasificador, este proceso es realizado a través las librerías de *TensorFlow*, en donde se antes de pasar el conjunto de datos a un modelo pasa por un preprocesamiento en el que implica convertir el valor de los pixeles de las imágenes a tensores que son soportados por *TensorFlow*, así como el previo procesamiento que los modelos requieren.

Se generan los lotes de imágenes tensoriales en tiempo real ya que se considera la prueba de diversos modelos de red, estos requieren de un tamaño en específico de la imagen, por lo que el preprocesamiento se realiza en tiempo real antes de un entrenamiento, adecuando los datos de acuerdo con lo solicitado por el modelo de red. Los modelos proporcionan una capa por la que los datos deben pasar antes de entrar al modelo. Posteriormente, mediante las bibliotecas de *TensforFlow* realizamos la configuración del generador de datos, en donde a través del método `ImageDataGenerator` podemos establecer la función de preprocesamiento,

el aumento de datos y la división del conjunto de datos en dos partes, para entrenamiento y para validación. Después, con el balanceo de datos se duplicaron las imágenes, por lo que en este proceso de la generación de los bloques de datos se aplica el voltear las imágenes de forma horizontal y vertical de forma aleatoria.

El tamaño de *batch* para el primer conjunto de datos se estableció en 16 mientras que para el segundo conjunto de imágenes se estableció en 32. Por medo del método de ImageDataGenerator se realiza la generación de los set de datos, en donde se almacenan el conjunto de entrenamiento y el conjunto de validación, se debe obtienen los datos a través del método `ImageDataGenerator.flow_from_directory` en el cual indicamos como argumento la dirección donde se encuentran las imágenes, el conjunto a obtener (entrenamiento/validación), el tamaño del *batch*, y las dimensiones de la imagen, en este caso ya se ha establecido en el preprocesamiento de datos, con esto el método en automático realiza la carga de los datos o imágenes en forma de vectores en las variables establecidas. Como resultado obtenemos la carga de las imágenes en memoria, aquí se muestra la distribución de las imágenes en los conjuntos de entrenamiento y validación, así como el total de imágenes en cada conjunto (véase en Imagen 5.12).

```
Found 13500 images belonging to 5 classes.
Found 1500 images belonging to 5 classes.
Found 1500 images belonging to 5 classes.
```

Imagen 5.12 Resultado de carga de imágenes en Google Colab. Imágenes para entrenamiento, validación y test en las cinco clases de Retinopatía Diabética.

La plataforma de *Google Colab* permite el monitoreo de sus variables, a través de este monitor podemos observar el contenido de las variables, en especial de variables que almacenan los tensores del conjunto de imágenes. Para el segundo conjunto de imágenes se establecieron 1805 imágenes en total como conjunto de validación.

5.5.Selección del modelo de red

Zhou presenta una revisión de diversos trabajos que se enfocan en el aprendizaje profundo, asi como diversos modelos clasificadores utilizados en el desarrollo de proyectos de clasificación de imágenes, la técnica de entrenamiento utilizando transferencia de

aprendizaje y considerando los modelos disponibles en las herramientas a utilizadas (Zhou, 2021).

EL trabajo utiliza la librería de *TensorFlow* y la API de *Keras* para el desarrollo del modelo de red neuronal, se tienen disponibles los modelos que se muestran en la Tabla 5.1:

Tabla 5.1 Modelos de CNN proporcionados por la API de Keras. Keras (2023), Keras Applications. Recuperado de https://keras.io/api/applications/

Modelo	Tamaño (MB)	Top-1 Accuracy	Top-5 Accuracy	Parametros (Millones)	Profundidad	Tiempo (ms) por paso de inferencia (CPU)	Tiempo (ms) por paso de inferencia (GPU)	Tamaño de entrada
Xception	88	0.7900	0.9450	22.90	81	109.4	8.1	299x299
VGG16	528	0.7130	0.9010	138.40	16	69.5	4.2	224x224
VGG19	549	0.7130	0.9000	143.70	19	84.8	4.4	224x224
ResNet50	98	0.7490	0.9210	25.60	107	58.2	4.6	224x224
ResNet50V2	98	0.7600	0.9300	25.60	103	45.6	4.4	224x224
ResNet101	171	0.7640	0.9280	44.70	209	89.6	5.2	224x224
ResNet101V2	171	0.7720	0.9380	44.70	205	72.7	5.4	224x224
ResNet152	232	0.7660	0.9310	60.40	311	127.4	6.5	224x224
ResNet152V2	232	0.7800	0.9420	60.40	307	107.5	6.6	224x224
InceptionV3	92	0.7790	0.9370	23.90	189	42.2	6.9	299x299
InceptionResNetV2	215	0.8030	0.9530	55.90	449	130.2	10	299x299
MobileNet	16	0.7040	0.8950	4.30	55	22.6	3.4	224x224
MobileNetV2	14	0.7130	0.9010	3.50	105	25.9	3.8	224x224
DenseNet121	33	0.7500	0.9230	8.10	242	77.1	5.4	224x224
DenseNet169	57	0.7620	0.9320	14.30	338	96.4	6.3	224x224
DenseNet201	80	0.7730	0.9360	20.20	402	127.2	6.7	224x224
NASNetMobile	23	0.7440	0.9190	5.30	389	27	6.7	224x224
NASNetLarge	343	0.8250	0.9600	88.90	533	344.5	20	331x331
EfficientNetB0	29	0.7710	0.9330	5.30	132	46	4.9	opt
EfficientNetB1	31	0.7910	0.9440	7.90	186	60.2	5.6	opt
EfficientNetB2	36	0.8010	0.9490	9.20	186	80.8	6.5	opt
EfficientNetB3	48	0.8160	0.9570	12.30	210	140	8.8	opt

EfficientNetB4	75	0.8290	0.9640	19.50	258	308.3	15.1	opt
EfficientNetB5	118	0.8360	0.9670	30.60	312	579.2	25.3	opt
EfficientNetB6	166	0.8400	0.9680	43.30	360	958.1	40.4	opt
EfficientNetB7	256	0.8430	0.9700	66.70	438	1578.9	61.6	opt
EfficientNetV2B0	29	0.7870	0.9430	7.20	438			opt
EfficientNetV2B1	34	0.7980	0.9500	8.20	438			opt
EfficientNetV2B2	42	0.8050	0.9510	10.20	438			opt
EfficientNetV2B3	59	0.8200	0.9580	14.50	438			opt
EfficientNetV2S	88	0.8390	0.9670	21.60	438			opt
EfficientNetV2M	220	0.8530	0.9740	54.40	438			opt
EfficientNetV2L	479	0.8570	0.9750	119.00	438			opt
ConvNeXtTiny	109	0.8130	0.0000	28.60				opt
ConvNeXtSmall	192	0.8230	0.0000	50.20				opt
ConvNeXtBase	339	0.8530	0.0000	88.50				opt
ConvNeXtLarge	755	0.8630	0.0000	197.70				opt
ConvNeXtXLarge	1310	0.8670	0.0000	350.10				opt

Primeramente, la plataforma de *Keras* proporciona una gran variedad de modelos pre-entrenados, los cuales pueden ser utilizados para la transferencia de aprendizaje, de la misma forma existen diferentes proyectos en los que utilizan cada uno de estos modelos la una tarea en especifica, como lo es la clasificación.

Después para una selección de un modelo a través de la información obtenida de la plataforma de *Keras* se realiza un análisis de la arquitectura de los modelos, como primera característica las capas de profundidad, considerando que entre más capas tenga una red neuronal convolucional, más profundo es el aprendizaje de las características más específicas de las imágenes. Otra característica es un análisis del rendimiento obtenido basándose en los valores de las métricas como lo es la precisión obtenida en tareas de clasificación anterior.

Posteriormente para una toma de decisiones en cuanto al modelo de red a seleccionar para este trabajo, se realiza un análisis de los datos de los modelos proporcionados por *Keras* de forma gráfica, a través de la comparación de los valores y la relación entre ellos. De esta manera es como se realiza la selección del modelo de red a utilizar, como apoyo a la toma de decisión se encuentran los trabajos consultados, los cuales han utilizado las arquitecturas específicamente en la clasificación de imágenes de fondo de ojo y la detección de la Retinopatía Diabética.

Entonces con un total de 38 modelos de Red Neuronal Convolucional y 8 variables que las distingue una de otras es complicado el análisis y la selección del modelo, para lo cual se toma en cuenta las variables y una jerarquía para la consideración en la selección. Como se muestra en la Figura 5.6, se ha establecido una jerarquía de las 4 variables principales y su nivel de importancia con respecto a las demás que nos ayuda en la selección del modelo.

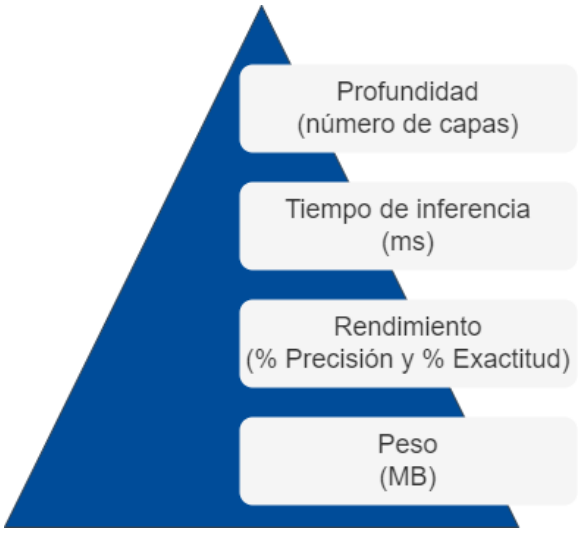
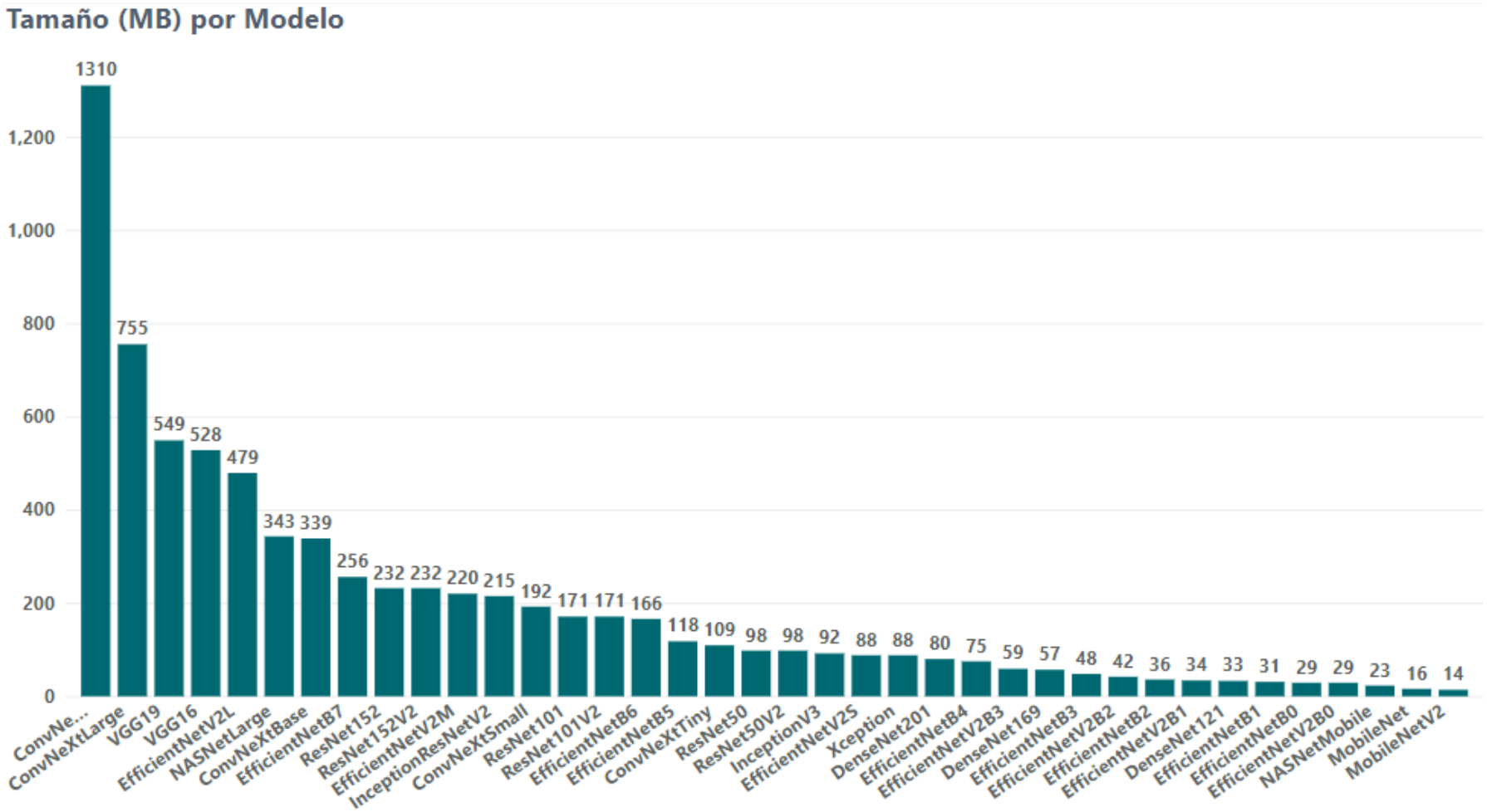


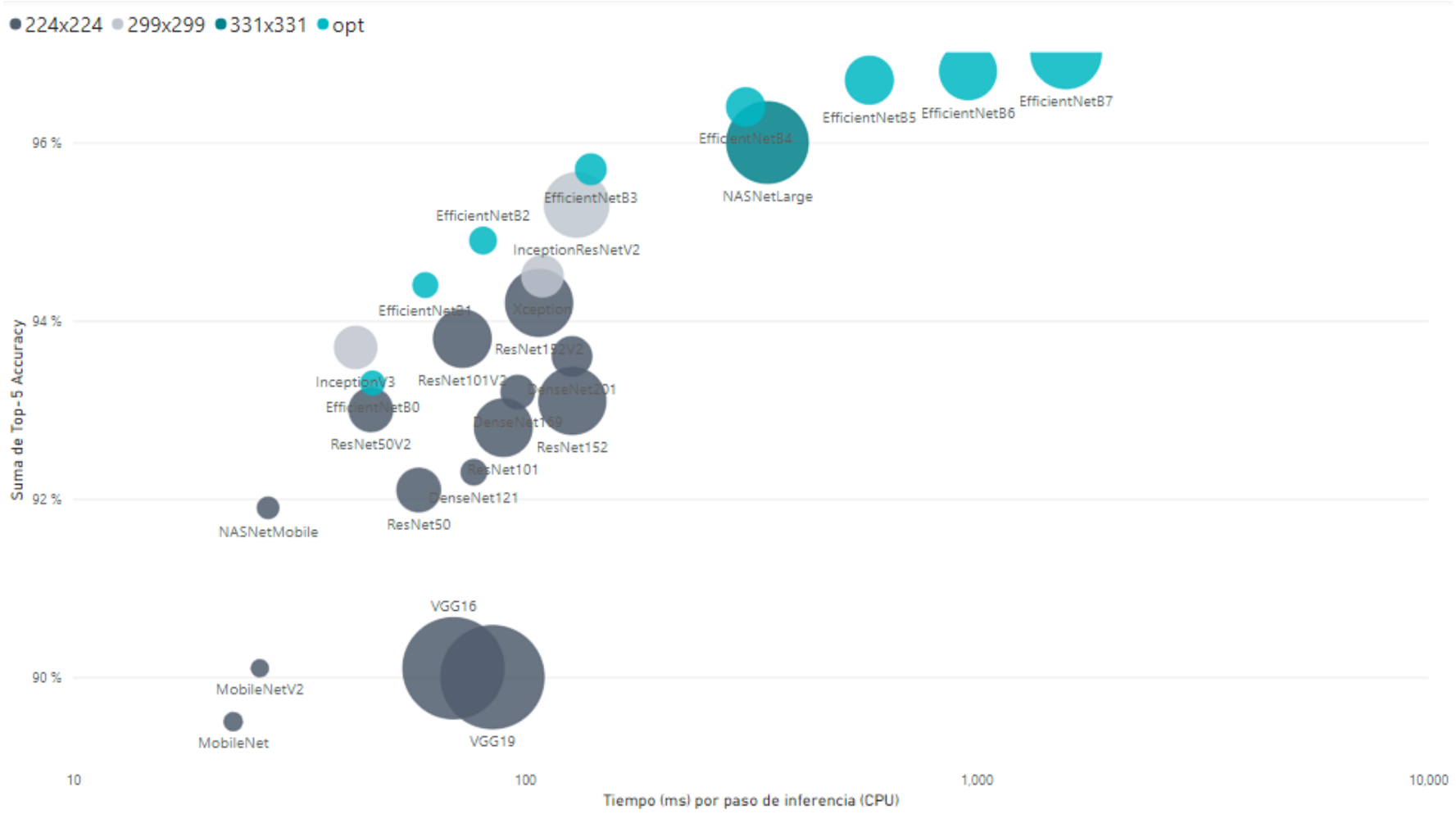
Figura 5.6 Jerarquía de variables para decisión de modelo de CNN. Fuente: Elaboración propia.

Por el tamaño, como base de la elección del modelo, el dispositivo de despliegue se toma en consideración con el mayor peso en la toma de decisión, debido al espacio del sistema de despliegue, es por lo que a través de la Gráfica 5.5 la cual representa la comparación entre el tamaño en memoria de cada uno de los modelos disponibles en *Keras*, con ello se define al modelo con menor peso: MobileNetV2, sin embargo, otras características como el número de capas de profundidad del modelo son variables que no ayudan a establecer como candidato único este modelo de red.

Con base en las características que se proporcionan deben tener relación entre ellas, o en su mayoría, considerando que debe ser un modelo de red ligero en cuanto a su peso, haber obtenido un mejor rendimiento, menor tiempo de inferencia y la profundidad de sus capas. La Gráfica 5.6 muestra un diagrama de burbujas en la cual se puede visualizar la relación entre las variables: tiempo (ms) por paso de inferencia, rendimiento (% exactitud en el Top 5), tamaño del modelo (MB) y dimensión del tamaño de entrada de la imagen (pixeles).

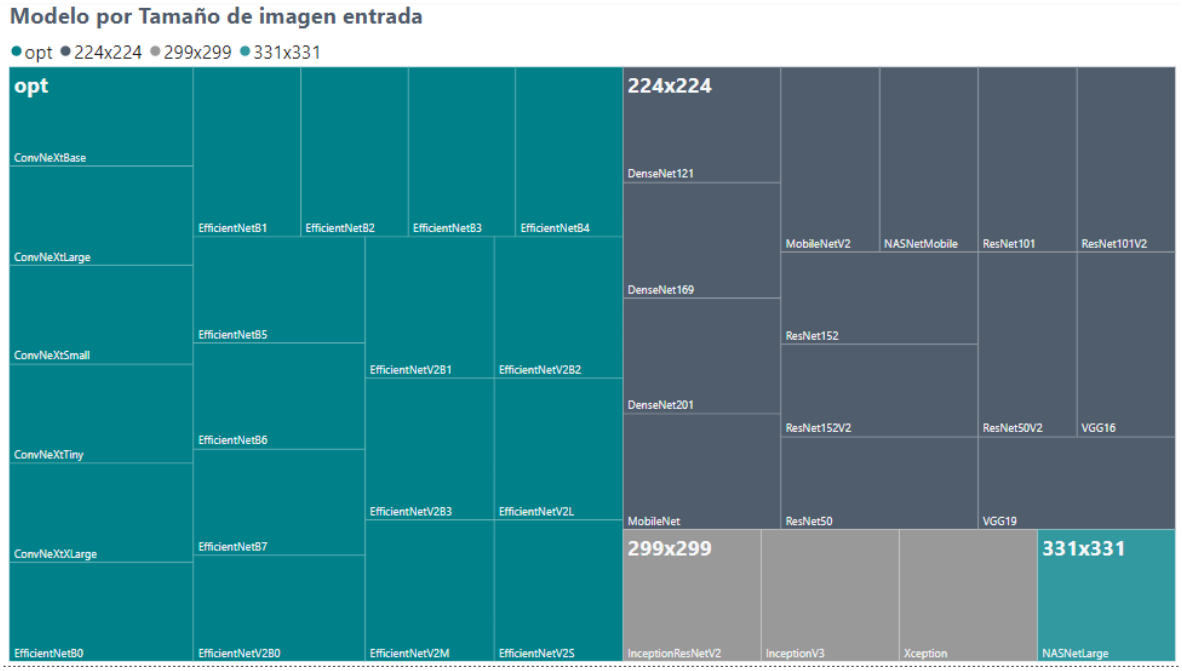


Gráfica 5.5 Comparación del Tamaño (MB) de los distintos modelos de CNN disponibles de API. Fuente: Elaboración propia a partir de datos de Keras Applications.



Gráfica 5.6 Relación entre las variables peso (MB), Tiempo (ms) por paso de inferencia (GPU), Top % Exactitud y dimensiones de la imagen de entrada. Fuente: Elaboración propia a partir de datos de Keras Applications.

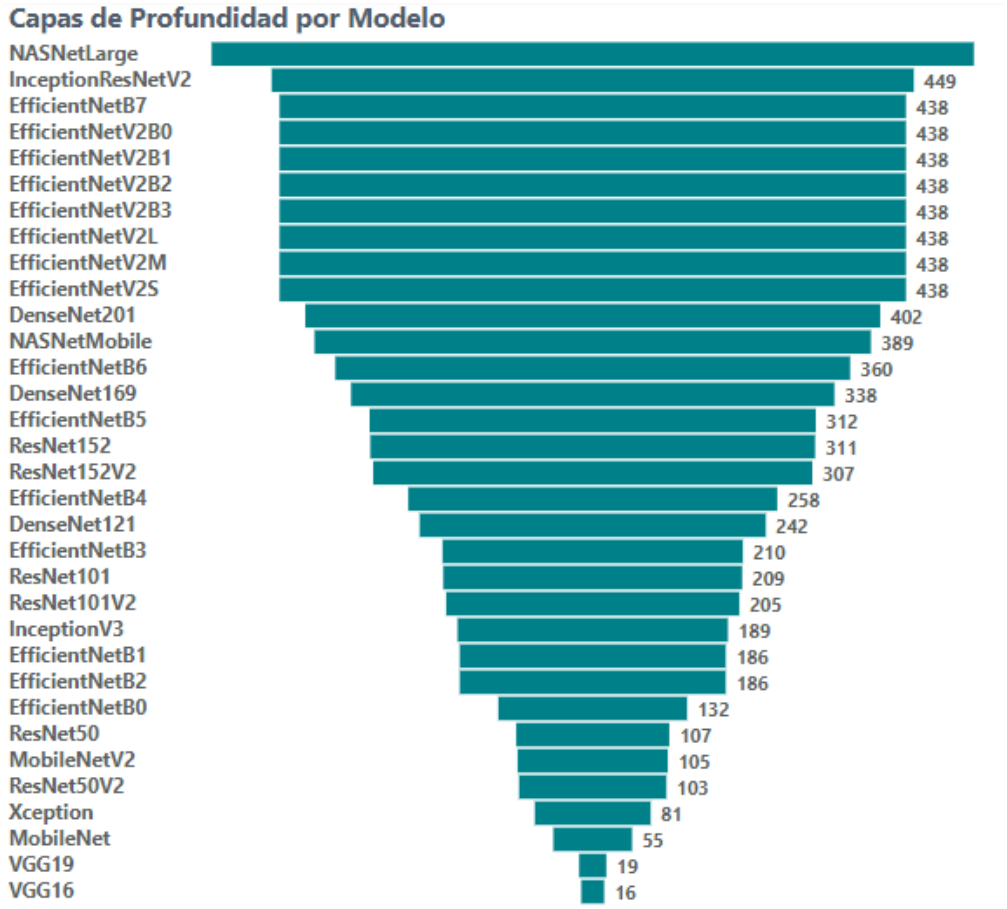
De acuerdo con la Gráfica 5.7 más de la mitad de los modelos que nos proporciona la API de *Keras*, tienen un tamaño de entrada de la imagen opcional, sin embargo, se consideran el promedio de las dimensiones de los modelos restantes, pues, aunque se tiene un tamaño de entrada que puede ser personalizado, el entrenamiento de los modelos se basa en imágenes con tamaños similares a los modelos restantes. Por ello se agregan más variables para la selección del modelo.



Gráfica 5.7 Modelos de CNN y dimensiones de imagen de entrada. Fuente: Elaboración propia a partir de datos de Keras Applications.

Basándose en la Gráfica 5.6 el tiempo de inferencia y la precisión es similar en la mayoría de los modelos, pues la mayoría de estos se encuentran en la parte superior izquierda del gráfico, lo que nos muestra que la relación entre el nivel de rendimiento y un tiempo de inferencia en segundos es similar entre estos modelos.

La profundidad de una red neuronal convolucional se basa en el número de capas, las capas más profundas analizan y aprenden las características más particulares de las imágenes, se tiene en consideración que las imágenes a clasificar son identificadas a través de las patologías de la retinopatía diabética.



Gráfica 5.8 Comparación de numero de capas de profundidad de los modelos de CNN. Fuente: Elaboración propia a partir de datos de Keras Applications.

Por la gran variedad de los modelos, resulta un poco difícil el análisis y comparación entre todos los modelos disponibles, con lo cual se realiza un filtro a todos los modelos pro el tamaño de entrada, seleccionando únicamente modelos que tienen una entrada de imagen de 224x224. Como se muestra en la Gráfica 5.9, 14 modelos de red neuronal convolucional tienen como entrada una imagen de 224x224, considerando que los modelos pre entrenados fueron con imágenes de ese mismo tamaño. En este análisis los modelos VGG16 y VGG19 son descartados, debido al tamaño, por otro lado, MobileNetV2 es considerado un modelo a seleccionar.

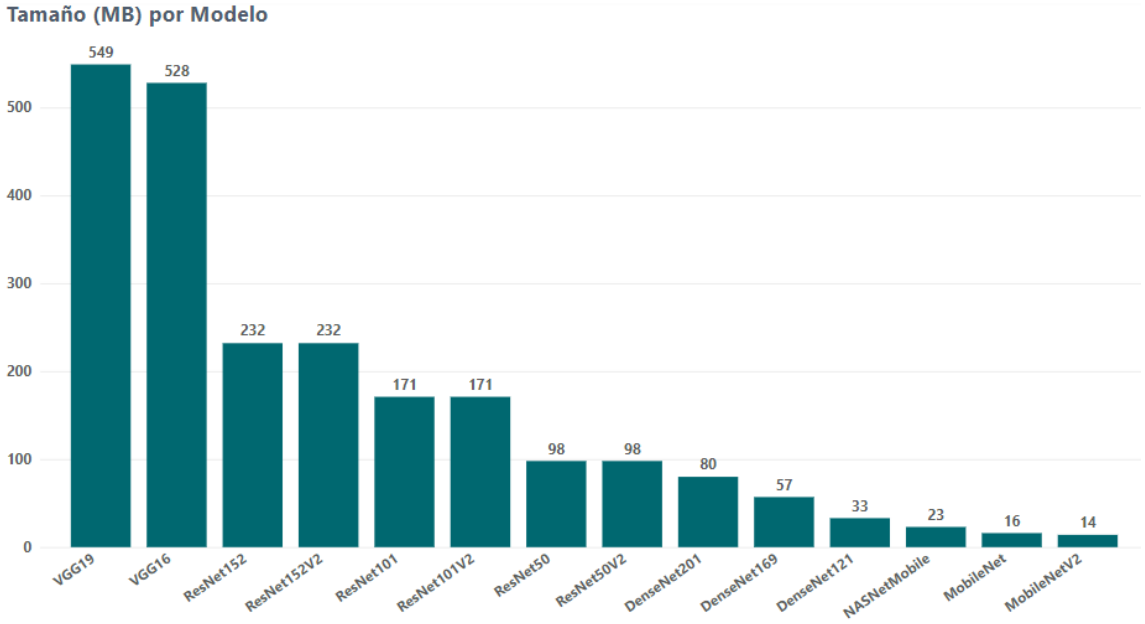
La arquitectura MobileNetV2 resulta ser un modelo ligero, que ha obtenido buenos resultados con una precisión mayor al 90% en métrica de precisión, sim embargo es un modelo que, a diferencia del grupo de modelos filtrados, tiene muy pocas capas de profundidad.

De acuerdo con la dificultad de las imágenes debido a las características geométricas se decide considerar un modelo profundo, que a su vez no tenga un valor alto en MB del grupo comparado. La arquitectura DenseNet201 cumple con estas características, como se muestra en la Gráfica 5.10 es el modelo con mayor número de capas filtradas.

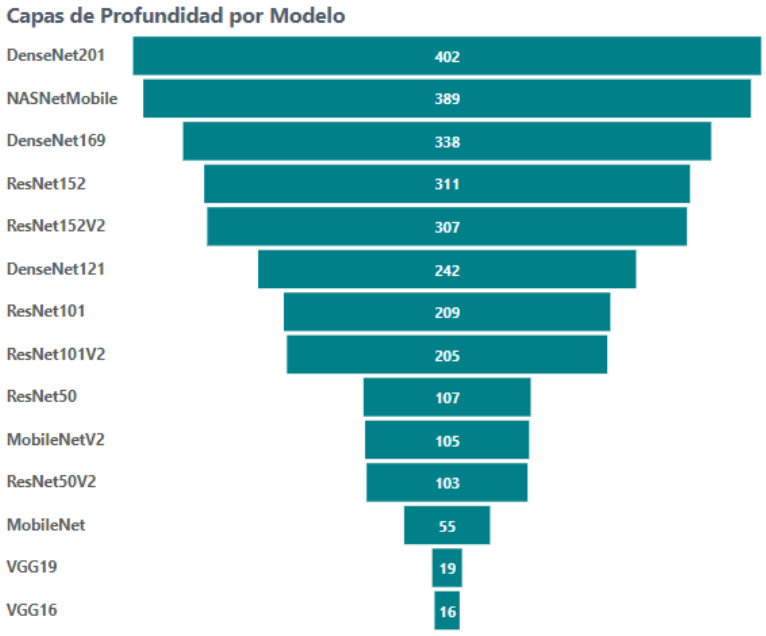
Se seleccionan los siguientes modelos:

1. MobileNetV2: por su nivel de rendimiento en cuanto a métricas de precisión, su peso y su arquitectura diseñada, este modelo es entrenado con los métodos: transferencia de características y ajuste fino. Es un modelo pequeño a diferencia de los otros, considerando el dispositivo de despliegue y con base en el estado del arte es un modelo que no debería tener problemas al ser implementado en un sistema embebido.
2. DenseNet201: al por ser el modelo con más profundidad, un tamaño considerable y un bueno resultado en cuanto a rendimiento en métricas de precisión. El modelo es entrenado por el metodo de transferencia de caracteristicas. Se ha decidido implementar este modelo como prueba y analisis de su comportamiento en el dispositivo de despliegue.

Para ambos modelos se utiliza la transferencia de aprendizaje, debido a que el conjunto de imágenes es considerado pequeño, la tecnica de transferencia de aprendizaje es un metodo de entrenamiento viable situaciones de este tipo.



Gráfica 5.9 Comparación del tamaño (MB) de los modelos de CNN con dimensiones de imagen de entrada de 224x224px. Fuente: Elaboración propia a partir de datos de Keras Applications.



Gráfica 5.10 Números de capas de profundidad para los modelos de CNN con dimensiones de tamaño de imagen de entrada a de 224x224px. Fuente: Elaboración propia a partir de datos de Keras Applications.

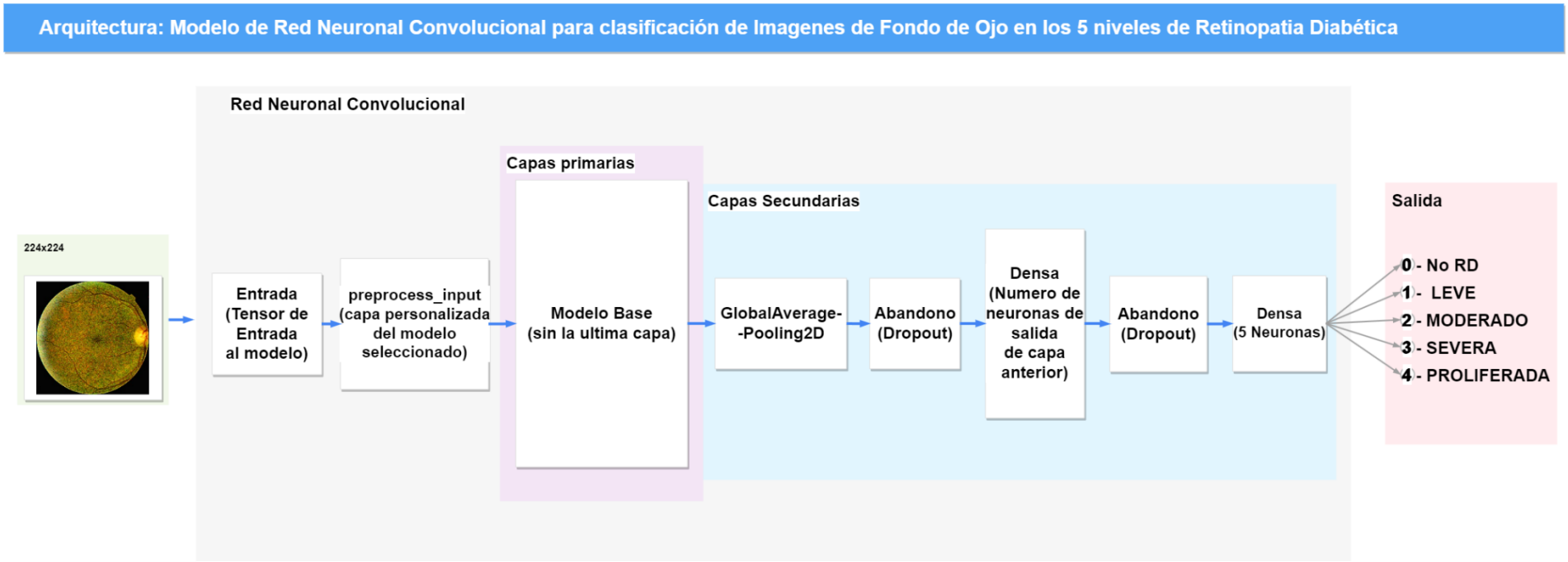


Figura 5.7 Modelo de Red Neuronal Convolucional basado en Modelos de CNN proporcionados por Keras. Fuente: Elaboración propia.

5.6. Carga de red pre-entrenada con pesos sin la última capa

El modelo en general se define en la Figura 5.7, en la cual se establece como capas principales al modelo de CNN seleccionado, donde el bloque modelo base es sustituido por los modelos MobileNetV2 o DeneNet201. Las capas secundarias corresponden a las capas de clasificación: bloques *GlobalAveragePooling2D* y *Densas*, las cuales son conectadas a la salida del *modelo base*.

El inicio de la carga del modelo base, con el modelo seleccionado, a través de la API se realiza la modificación de las capas de entrada y salida del modelo. Como primer capa o entrada al modelo es un tensor que cumple con las características de tamaño de imagen a la que el modelo está configurado, considerando que este modelo fue diseñado y entrenado con imágenes de 224x224 pixeles, con el método `Input()` se establecen los parámetros de largo y ancho de la imagen, así como el número de canales de color, se establecen 3 canales correspondiendo a los canales rojo, verde y azul.

La entrada del modelo de la API de *Keras* proporciona un preprocesamiento a los datos de que debe ser agregado como capa de entrada de la arquitectura seleccionada. Posterior a este preprocesamiento se realiza la conexión a el modelo de red, que a su vez tiene un tensor de entrada. A continuación, se presenta el código empleado:

```
# Tamaño del tensor de entrada, se refiere al tensor de la imagen
input_tensor = Input(shape=(224,224,3))
# el modelo MobileNetV2 requiere de un una capa de entrada
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
# Conectar el tensor a la capa de preprocesamiento del modelo
x = preprocess_input(input_tensor)
# base central del modelo
base_model = tf.keras.applications.MobileNetV2(
    input_shape=(224,224,3),     # Tamaño de la imagen de entrada
    include_top=False,           # sin la parte clasificadora
    weights='imagenet',          # pesos de imagenet
    input_tensor=x               # conectamos la capa de procesamiento a la
entrada
)
```

De acuerdo con la literatura, en modelo de red debe ser modificado, eliminando la última capa para personalizar la salida del modelo, con el objetivo de tener 5 salidas correspondientes a los niveles de Retinopatía Diabética. Esta última capa puede establecerse

directamente en el modelo, sin embargo, se anexan las capas secundarias, como se muestra en la Figura 5.7, en donde las capas secundarias corresponden a una capa de *GlobalAveragePooling2D*, dos capas densas y dos capas más con abandono (*Dropout* por su traducción al inglés).

Agrupación promedio Global (*GlobalAvergePooling,* por su traducción al inglés), son utilizadas al diseñar la red neuronal de tal forma que la última capa convolucional tenga una cantidad de canales igual a la cantidad de salidas y simplemente tomar uno de esos canales y promediar los valores en el número de valores de salida (Rahim, 2023).

Las capas de abandono se agregan como una técnica de regularización, al reducir la capacidad del modelo con el objetivo de lograr un menor error de generalización.

6. RESULTADOS Y DISCUSIÓN

6.1. Entrenamiento de MobileNetV2 con primer conjunto de datos

Las imágenes que son utilizadas para el entrenamiento son almacenadas en la plataforma de *Google Drive*, al realizar la ejecución del *Google Colab* se realiza una conexión a *Google Drive* y a través de esta conexión se realiza la lectura de las imágenes, de la misma forma se almacenan los datos generados como imágenes y el mismo archivo *ipynb* con los resultados de ejecución.

En la Figura 5.7 se muestra la arquitectura en general del modelo utilizado en el entrenamiento, en este entrenamiento el modelo base corresponde a el modelo de arquitectura de MobileNetV2 y DenseNet201. Para el primer entrenamiento se realizó el entrenamiento de la capa secundaria en 2 épocas, con una tasa de entrenamiento de *0.001*. La tendencia del valor de función de se muestra descendente, un buen modelo obtiene valores en la función de perdida cercamos a cero, ya que representa la discrepancia entre el valor real y el valor obtenido en la predicción, un valor a cero representa el costo de equivocarse.

El modelo debe tener variables a entrenar, considerando que dentro de este modelo en general está la base del modelo, la cual no debe tener variables a entrenar, en la Imagen 6.1 se muestra la comprobación, donde es posible realizar la consulta de la longitud de las variables que serán entrenadas, esto con el método *trainable_variables*.

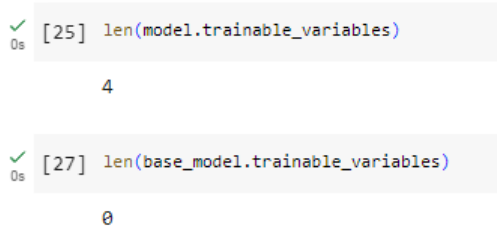


Imagen 6.1 Numero de variables a entrenar de una CNN con arquitectura de base MobiloeNetV2.

En resumen, de esta primera configuración del modelo, se observa la longitud de las variables a entrenar. En la sección de parámetros no ajustable, se tiene un peso que puede comprobarse con el resumen de la base del modelo, se tiene el mismo peso.

Para el modelo con arquitectura base MobileNetV2 ocupa un espacio en memoria de 8.61 MB como se visualiza en la Imagen 6.2, estos parámetros no son ajustados en el entrenamiento, pues a partir de estos se realiza la transferencia de características.

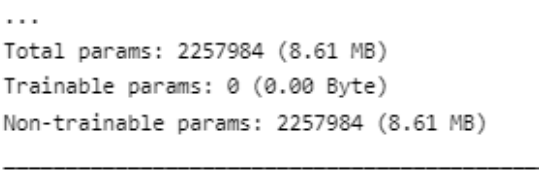


Imagen 6.2 Resumen de tamaño (MB) de modelo de CNN con arquitectura base MobileNEtV2.

El modelo en general ocupa un espacio en memoria de 14.89 MB, del cual 8.61 MB corresponden al modelo base que no es ajustable durante el entrenamiento (Véase en Imagen 6.3).

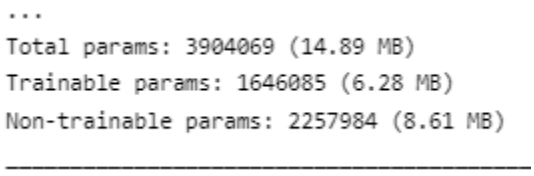


Imagen 6.3 Resumen de tamaño (MB) de la arquitectura de MobileNetV2.

Con la API de *Keras* es posible utilizar devoluciones de llamada (*API Callbacks* por su traducción al inglés) para ajustar el valor de la taza de aprendizaje asi como el de las épocas de entrenamiento basándose en los resultados obtenidos de una métrica en el ajuste del modelo. Para el valor de la taza de aprendizaje es utilizado la devolución de llamada `ReduceLROnPlateau`, para monitorear el valor obtenido en la validación esto para la métrica

de función de perdida, los argumentos para esta devolución de llamada se muestran a continuación:

```
ReduceLROnPlateau(monitor='val_loss', mode='min', patience=5,
                  factor=0.5, min_lr=1e-6,  verbose=1)
```

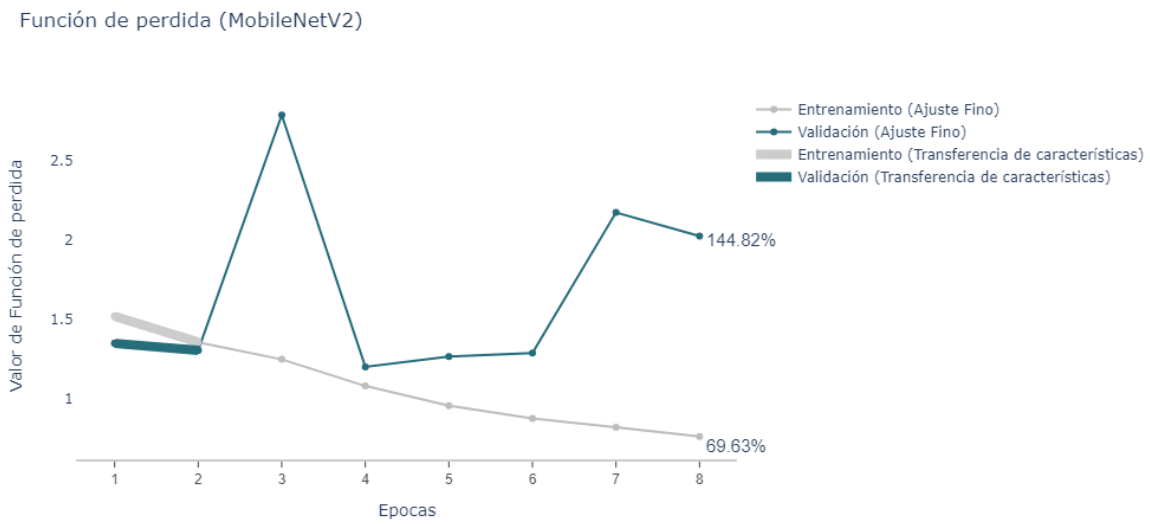
Donde:

a. Monitor (*monitor* por su traducción al inglés): métrica a monitorear, se estableció la métrica de función de perdida obtenido en la validación del modelo.
b. Modo (*mode* por su traducción al inglés): puede establecerse en {'auto','min','max'}, se estableció en 'min', en donde la tasa de aprendizaje reducirá cuando el valor de la métrica monitoreada haya dejado de disminuir, esto debido a que la métrica debe tener valores descendentes cercanos a 0 con el paso de las épocas.
c. Paciencia (*patiente* por su traducción al inglés) es el número de épocas sin mejora después de las cuales se reducirá la tasa de aprendizaje, se estableció en *5*.
d. Factor (*factor* por su traducción al inglés): factor por el cual reducirá la taza de aprendizaje, para este entrenamiento el factor se estableció en *0.5*, es decir el valor para la tasa de aprendizaje reducirá: *0.00005* si se tiene una taza de *0.0001*.
e. Límite inferior de la taza de aprendizaje (min_lr): al establecerse una reducción de la taza de aprendizaje, se estableció el valor de *0.000001*.

La segunda devolución a las llamadas utilizadas es `EarlyStopping` en la cual el entrenamiento se detendrá cuando el valor de una métrica monitoreada haya dejado de mejorar, los valores para el argumento de esta devolución de llamada se establecieron de la siguiente manera:

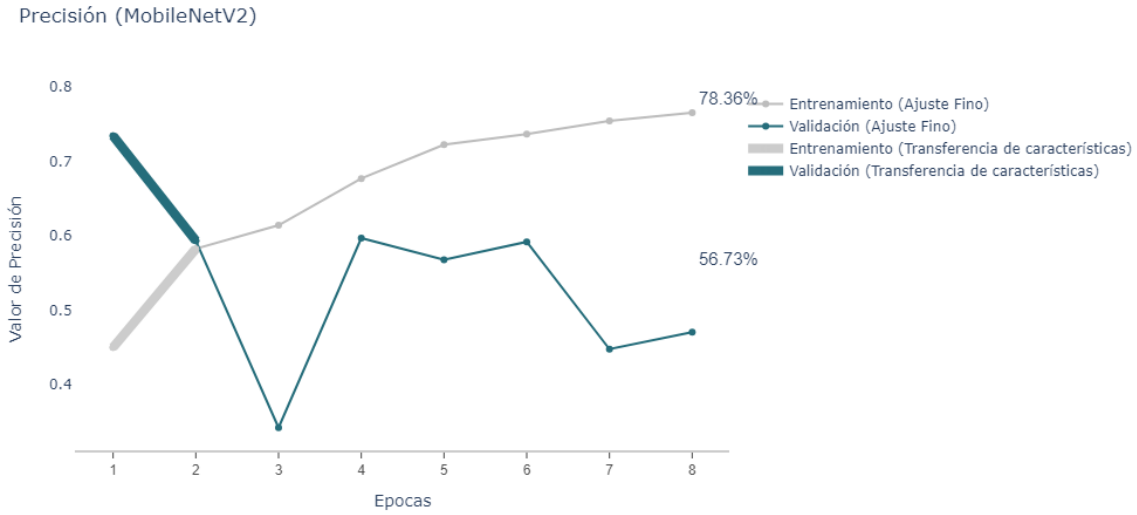
```
EarlyStopping(monitor='val_loss', mode='min',
              patience=5, restore_best_weights=True,
              verbose=1)
```

Con el *fine tunning* se espera que este valor redujera, sin embargo, como se muestra en la Gráfica 6.1, los valores tienden a oscilar bruscamente, teniendo una tendencia al alza, lo que implica que el modelo no es capaz de clasificar correctamente a las imágenes El número de épocas de entrenamiento se estableció en 28, una taza de aprendizaje de *0.0001*.



Gráfica 6.1 Función de perdida para MobileNetV2, en el eje x época de entrenamiento, eje y valor de función de perdida. Fuente: elaboración propia con datos obtenidos durante el entrenamiento del modelo.

Como se observa en la Gráfica 6.2, la métrica de precisión para el entrenamiento del modelo tiene una tendencia descendente para el entrenamiento mientras que para la validación se muestra descendente. Aunque el entrenamiento se estableció en *28* épocas, las devoluciones a llamada detuvieron el entrenamiento en la época 7.



Gráfica 6.2 Valores de precisión para entrenamiento y validación del modelo de red con modelo base de la arquitectura MobileNetV2. Fuente: elaboración propia con datos obtenidos en el entrenamiento del modelo.

Las métricas para el entrenamiento de este primer modelo son muy bajas con lo establecido, pues basándose en la métrica de precisión los valores no superan el 50%, mientras que para

la métrica de exactitud el modelo no logra superar el 80%. Como se muestra en la Tabla 6.1, reporte de métricas de rendimiento.

Tabla 6.1 Reporte de métricas de rendimiento para el modelo de Red con arquitectura base MobileNetV2. Fuente: elaboración propia con datos obtenidos de matriz de confusión.

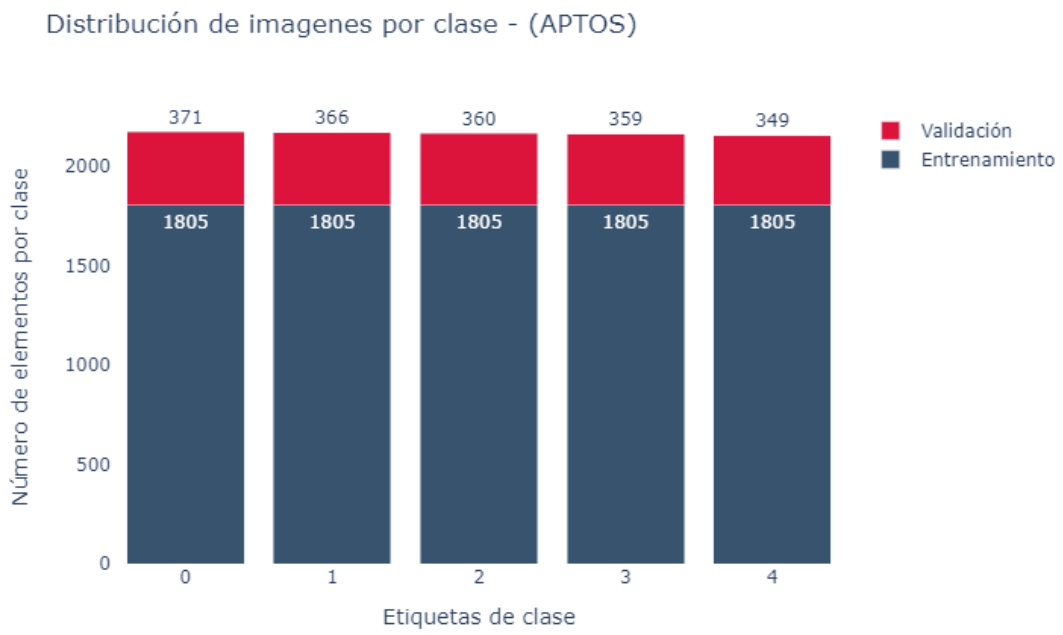
Clase	Especificidad	Precisión	Exactitud	Sensibilidad
0	73.67%	37.55%	71.60%	63.33%
1	85.42%	34.46%	74.47%	30.67%
2	97.58%	44.23%	79.60%	7.67%
3	91.25%	48.78%	79.67%	33.33%
4	70.50%	37.01%	72.09%	79.33%
macro	83.68%	40.41%	75.48%	42.87%

Con los resultados obtenidos anteriormente obtenemos como resultado un modelo deficiente, basándose en las métricas obtenidas en la validación. El entrenamiento modelo afecta a las metas planteadas, debido a que los valores obtenidos no superan a los propuestos.

6.2. Entrenamiento: Extracción de características

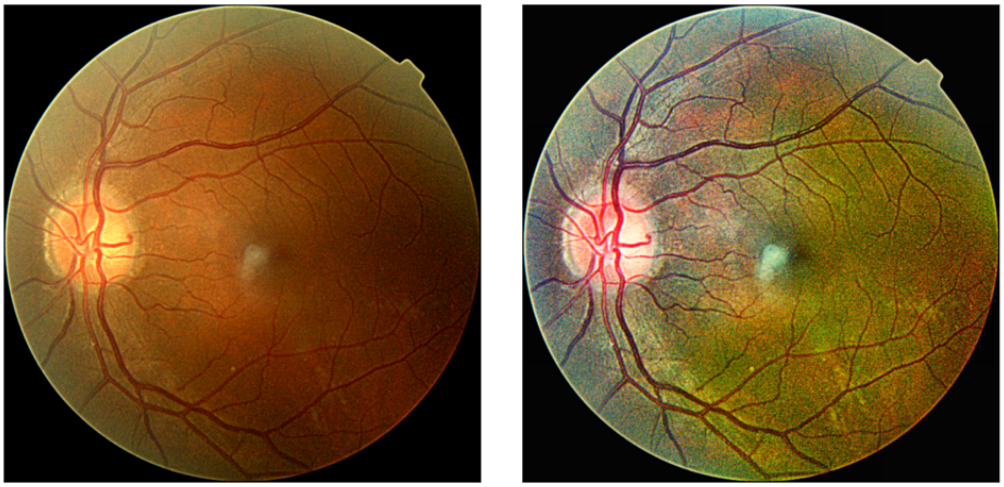
Considerando los resultados obtenidos en entrenamiento del modelo con la arquitectura base MobileNetV2 se establece un nuevo conjunto de datos, el conjunto de datos de *APTOS*, el cual realizando una exploración de las imágenes se aprecian mejor las características de las anomalías de cada una de las clases. Sin embargo, se decide agregar un método más de procesamiento de datos, la ecualización del histograma adaptativo, el conjunto de imágenes se utiliza para el entrenamiento del segundo modelo de red seleccionado.

Posterior a un balanceo de clases con sobre muestreo aleatorio, donde el número de elementos de clase se estableció el valor de la clase mayoritaria (Clase 0) como se muestra en la Grafica 6.3.



Gráfica 6.3 Distribución de imágenes del Dataset APTOS, posterior a un balanceo de clases. Fuente: elaboración propia con datos de (APTOS,2019).

El procesamiento que se ha añadido a este conjunto de imágenes es la Ecualización del Histograma Adaptativo, como se muestra en la Imagen 6.4, con lo que las características de la imagen son resaltadas, como es el caso de los vasos sanguíneos.



a) *b)*

Imagen 6.4 a) imagen con procesamiento fusión de filtro gaussiano, b) imagen fusión de filtro gaussiano y procesamiento CLAHE. Fuente: elaboración propia con imágenes del Dataset APTOS2019.

El modelo base corresponde a la arquitectura DenseNet201 sin la última capa, posteriormente se agrega la capa clasificadora, y se realiza un entrenamiento, aquí se tiene la base de la modelo congelada, agregamos una capa de función de activación, una capa densa al final con 5 neuronas que corresponden al número de clases de retinopatía Diabética:

```
base_model.trainable = False # congelamos el modelo
# Agregamos una capa de GlobalAveragePooling
x = GlobalAveragePooling2D()(base_model.output)
x = Dropout(0.5)(x)
# Capa totalmente conectada
x = Dense(2048, activation='relu')(x)
x = Dropout(0.5)(x)
# Capa logística – con 5 classes
final_output = Dense(5, activation='softmax', name='final_output')(x)
# Creacion del modelo, entrada y salida
model = Model(inputs=base_model.input, outputs=final_output)
```

Para el modelo con base en la arquitectura DenseNet201 se han generado 84.00 MB como se muestra en la Imagen 6.5, donde únicamente son entrenados 14.11 MB, por lo que 68.89 MB corresponden a la base de la arquitectura DenseNet201, la cual se muestra en la Imagen 6.6 se encuentra deshabilitada para entrenamiento.

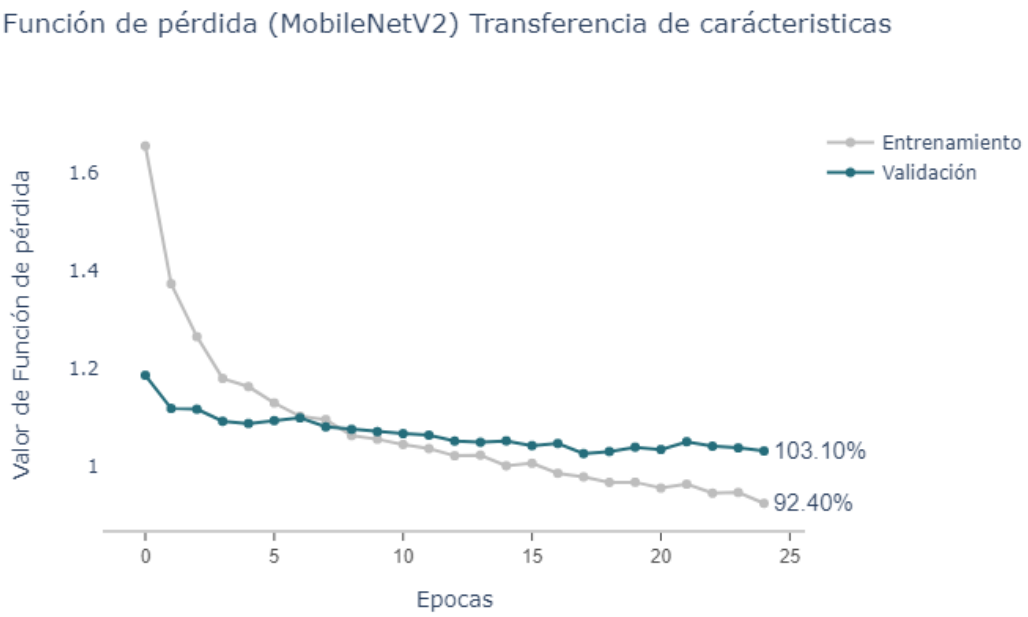
```
...
Total params: 22019909 (84.00 MB)
Trainable params: 3697925 (14.11 MB)
Non-trainable params: 18321984 (69.89 MB)
_________________________________________________
```

Imagen 6.5 Resumen de tamaño (MB) del modelo de CNN con arquitectura base DenseNet201.

```
...
Total params: 18321984 (69.89 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 18321984 (69.89 MB)
_________________________________________________
```

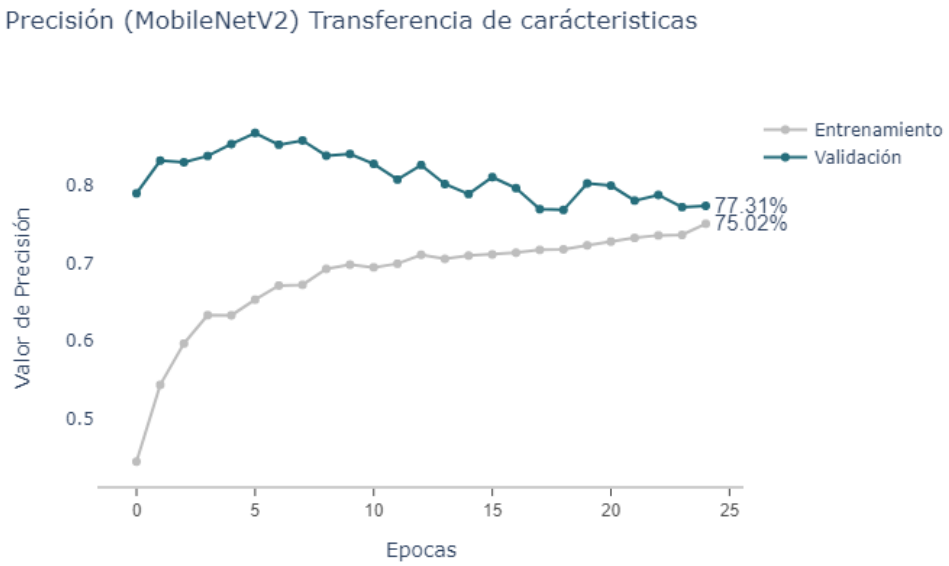
Imagen 6.6 Resumen de tamaño (MB) del modelo base de la arquitectura DenseNEt201.

Los resultados de este entrenamiento para el modelo con la arquitectura base de MobileNetV2 se establecieron 25 épocas de entrenamiento, sin embargo, como se observa en la Gráfica 6.4, el modelo tuvo valores en función de perdida cercanos mayores al 100% en la validación. esto implica que el modelo tiene muchas diferencias entre su valor de clasificación y el valor real.



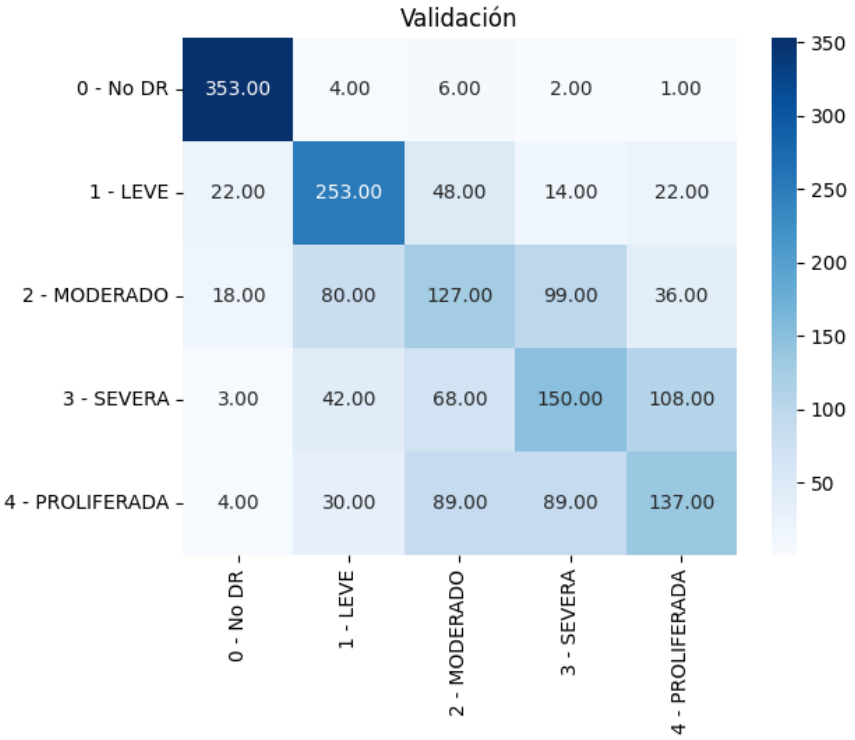
Gráfica 6.4 Valores de función de perdida entrenamiento y validación del modelo de CNN con arquitectura base MobileNetV2, Transferencia de Aprendizaje. Fuente: Elaboración propia con datos obtenidos en el entrenamiento.

En la Gráfica 6.5 se muestran los valores obtenidos de la métrica de Precisión durante el entrenamiento y validación, en las últimas épocas de entrenamiento el modelo obtuvo valores alrededor del 76% en entrenamiento y validación, valores por debajo de lo establecido en la hipótesis.



Gráfica 6.5 Valores de precisión obtenidos en entrenamiento y validación del modelo de CNN con arquitectura base MobileNetV2, 25 épocas de entrenamiento y transferencia de aprendizaje. Fuente: Elaboración propia con datos obtenido en el entrenamiento.

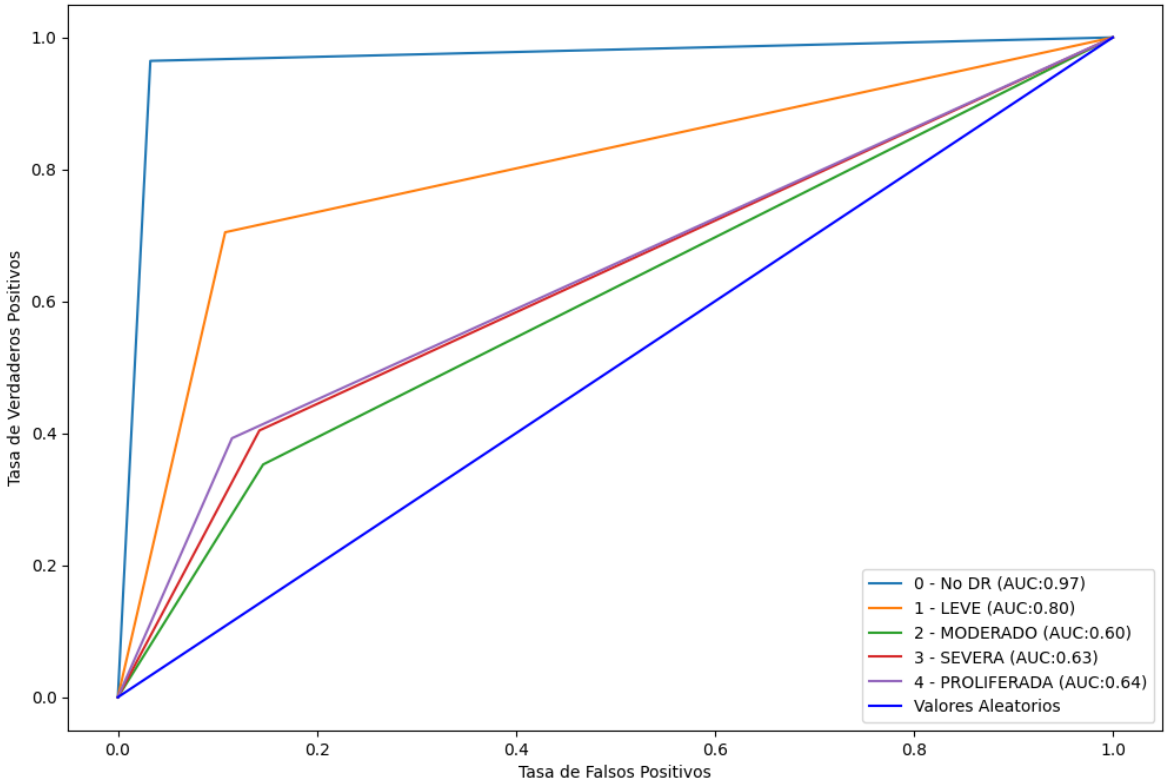
En la Gráfica 6.6 se muestra la matriz de confusión para el modelo con modelo base DenseNet201, donde las filas representan las clasificaciones actuales y las columnas las clasificaciones obtenidas, la matriz de confusión de un modelo adecuado tendrá la mayoría de las muestras a lo largo de la diagonal (Microsoft, 2023). Para la clase 0 (No Retinopatía Diabética) 353 de 366 imágenes son clasificadas correctamente, lo que representa el 96% de clasificaciones correctas. La clase 2 (Retinopatía Diabética Moderada) resulto la clase con más clasificaciones incorrectas con un 127 de 360 imágenes clasificadas correctamente, lo que representa el 35% de clasificaciones correctas.



Gráfica 6.6 Matriz de confusión: clasificaciones actuales (filas) y clasificaciones obtenidas (columnas). Fuente: Elaboración propia a partir de datos obtenidos en la predicción del modelo.

Al ser una clasificación multiclase, el área bajo la curva si como las métricas establecidas en este trabajo, se calculan para cada una de las clases, ya que el modelo puede ser eficiente al detectar solo algunas de las clases. En la Grafica 6.5 se muestra el valor del área bajo la curva para cada una de las clases de RD, la curva que se aproxima a la esquina superior izquierda se está aproximando a un valor de la tasa de Verdaderos Positivos, para este proyecto se obtiene el área bajo la curva para cada clase, las clases con una curva con mayor área son la clase 0 (No Retinopatía Diabética) y clase 1 (Retinopatía Diabética Leve) con un valor de

con un valor de 97% y 80% respectivamente. La clase 2 (Retinopatía Diabética Moderada) tiene un área bajo la curva 60%, lo que representa el menor valor en área bajo la curva con respecto a las clases restantes.



Gráfica 6.7 Área bajo la curva para cada una de 5 clases del modelo clasificador con base MobileNetV2. Fuente: Elaboración propia a partir de datos obtenidos del entrenamiento.

En la Tabla 6.2 se muestran los valores obtenidos para el modelo de CNN con arquitectura base MobileNetV2, el cual fue entrenado por transferencia de aprendizaje (extracción de características). El modelo obtiene valores de precisión 88.25% y exactitud 96.68% al clasificar a las imágenes de ojos sanos, sin embargo, para las clases que presentan ojos enfermos, es decir se encuentran en los niveles 1 a 4 de Retinopatía Diabética, el modelo presenta dificultades para ubicar el nivel exacto en el que se encuentra. Esto se visualiza de una mejor manera en la matriz de confusión, ya que los resultados de las predicciones se encuentran cercanos a lo largo de la diagonal.

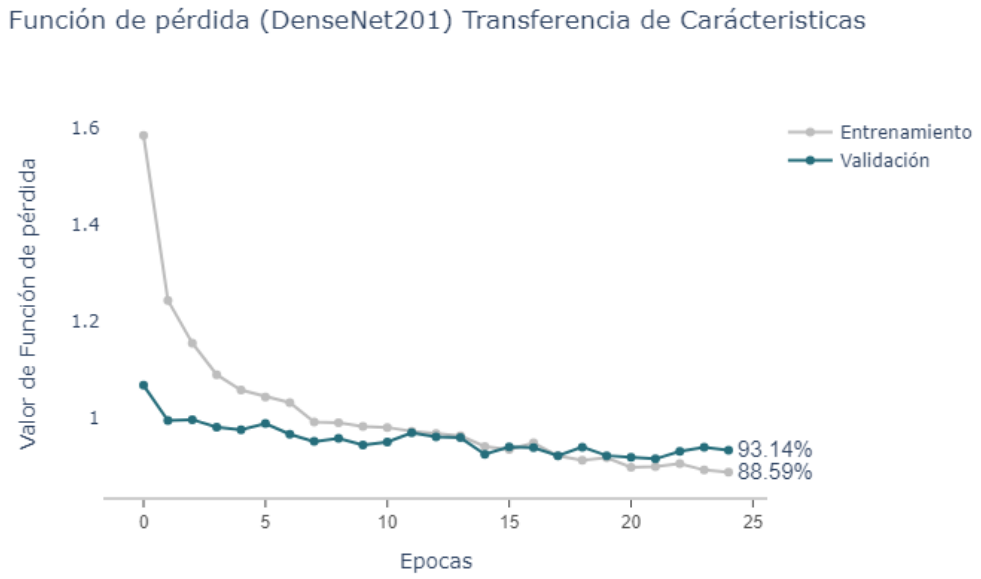
De un total de 1805 imágenes, 1157 fueron clasificadas correctamente, que representan el 64% del total de imágenes. El 26% de las clasificaciones que fueron clasificadas como

erróneas, es decir la etiqueta asignada por el modelo no corresponde a la real, corresponde a un nivel superior o inferior del real.

Tabla 6.2 Métricas de rendimiento obtenidas para el Modelo de CNN con arquitectura base MobileNetV2 con entrenamiento por transferencia de características. Fuente: Elaboración propia.

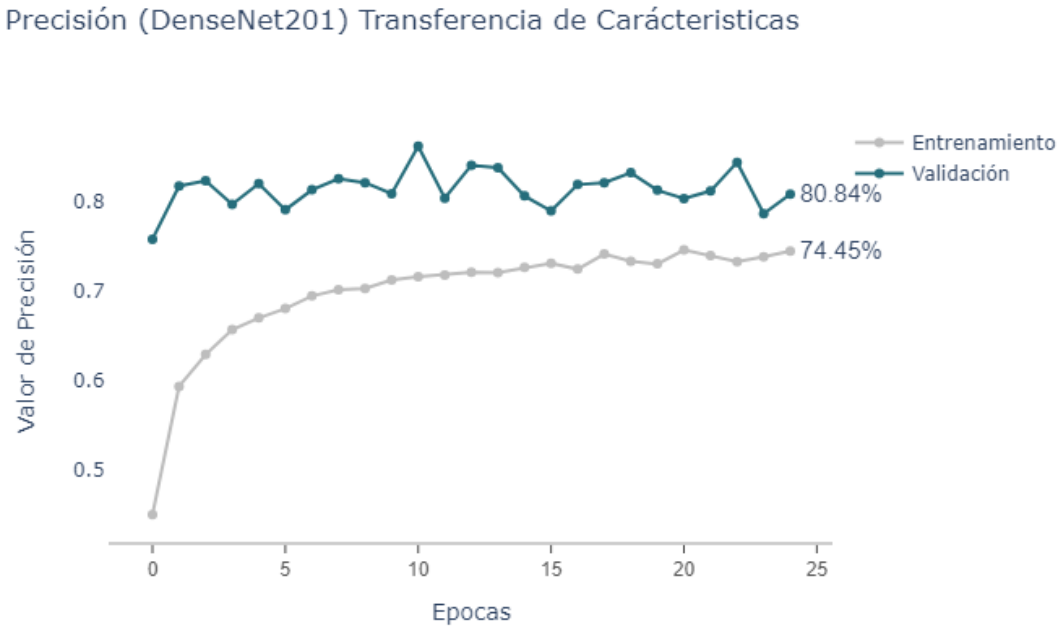
Clase	Especificidad	Precisión	Exactitud	Sensibilidad	AUC
0	96.73%	88.25%	96.68%	96.45%	97.00%
1	89.21%	61.86%	85.48%	70.47%	80.00%
2	85.40%	37.57%	75.40%	35.28%	60.00%
3	85.77%	42.37%	76.45%	40.43%	63.00%
4	59.35%	13.43%	56.56%	39.26%	64.00%
macro	83.29%	48.70%	78.12%	56.38%	72.80%

Para el segundo modelo de CNN con arquitectura base DenseNet201, el entrenamiento por transferencia de aprendizaje se realizó a 25 épocas y con el mismo conjunto de datos. En la Grafica 6.8 se muestran los valores de la función de perdida que se obtuvieron durante el entrenamiento y la validación en cada una de las épocas de entrenamiento. Los valores para esta métrica si mejoran en cada época de entrenamiento, aunque el porcentaje obtenido no es suficiente para considerar al modelo como buen clasificador.



Gráfica 6.8 Valores de función de perdida en entrenamiento y validación para el modelo de CNN con base DenseNet201. Fuente: Elaboración propia a partir de datos obtenidos en el entrenamiento.

Para los valores obtenidos en la métrica de precisión como se muestra en la Gráfica 6.7, van mejorando en cada época de entrenamiento, los valores se consideran buenos ya que se obtienen valores durante la validación alrededor del 80.81%.

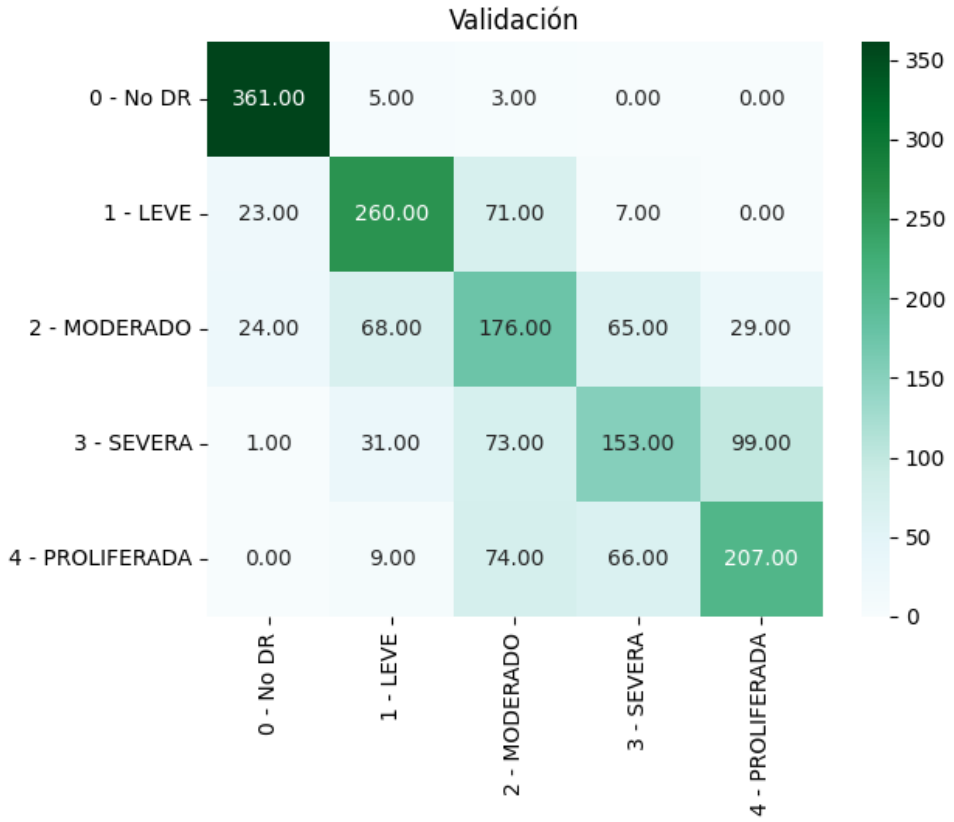


Gráfica 6.9 Valores de la métrica de precisión en entrenamiento y validación para el modelo de CNN con base DenseNet201. Fuente: Elaboración propia a partir de datos obtenidos en el entrenamiento.

Posterior al entrenamiento del modelo se realiza una validación con el conjunto de imágenes de validación, en la Gráfica 6.10 se muestra la matriz de confusión de los valores reales comparados con los valores proporcionados por el modelo con el conjunto de pruebas. Al ser un modelo clasificador multiclase con 5 valores posibles de salida, es posible que la clasificación realizada por el modelo se establezca en las otras 4 clases.

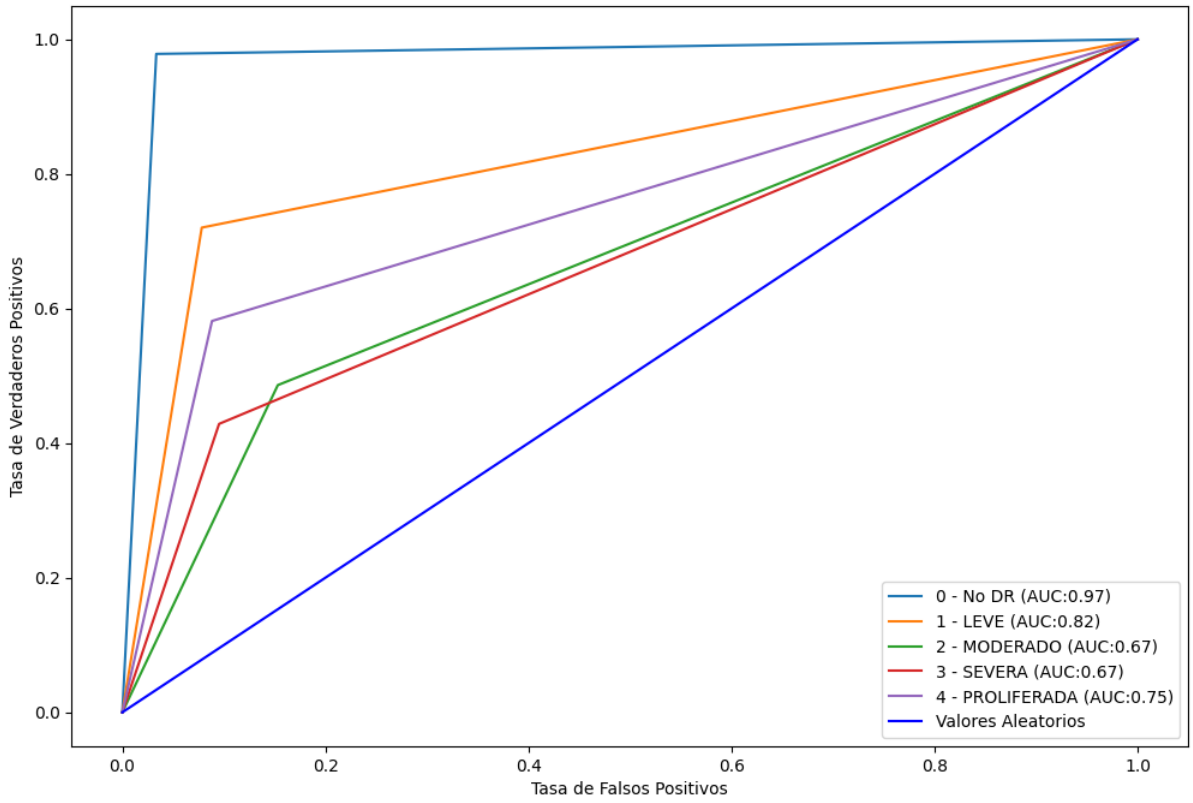
Esto se visualiza de una mejor manera en la matriz de confusión, ya que los resultados de las predicciones se encuentran cercanos a lo largo de la diagonal.

De un total de 1805 imágenes, 1157 fueron clasificadas correctamente, que representan el 64% del total de imágenes. El 26% de las clasificaciones que fueron clasificadas como erróneas, es decir la etiqueta asignada por el modelo no corresponde a la real, corresponde a un nivel superior o inferior del real.



Gráfica 6.10 Matriz de correlación para los valores reales y los valores obtenidos por el modelo clasificador con base DenseNet201. Fuente: Elaboración propia a partir de datos obtenidos en la inferencia del modelo.

El modelo clasificador con base DenseNet201 de acuerdo con los valores del área bajo la curva que se muestran en la Gráfica 6.11, tiene buenos resultados al clasificar las imágenes que pertenecen a la clase 1, es decir ojos sanos, seguido de la clasificación de ojos de la clase 2 (R0: LEVE).



Gráfica 6.11 Valores de área bajo la curva para las clases del modelo clasificador con base DenseNet201. Fuente: Elaboración propia a partir de datos generados por el modelo.

Los valores de las métricas establecidas para el modelo clasificador se muestran en la Tabla 6.3, para cada una de las métricas por clase y el valor para el sufijo macro. Comparando los valores obtenidos para el modelo con base MobileNetV2 y el modelo con base DenseNet201, DenseNet201 obtuvo mejores resultados, sin embargo, los valores obtenidos para ambos modelos no cumplen con el objetivo establecido en los objetivos del presente trabajo.

Tabla 6.3 Métricas de rendimiento obtenidas para el Modelo de CNN con arquitectura base DenseNet201 con entrenamiento por transferencia de características. Fuente: Elaboración propia a partir de los datos obtenidos en la matriz de confusión.

Clase	Especificidad	Precisión	Exactitud	Sensibilidad	AUC
0	96.66%	88.26%	96.90%	97.83%	97.00%
1	92.17%	69.71%	88.14%	72.02%	82.00%
2	84.68%	44.33%	77.45%	48.62%	67.00%
3	90.47%	52.58%	81.05%	42.86%	67.00%
4	58.17%	17.89%	58.17%	58.15%	75.00%
macro	84.43%	54.55%	80.34%	63.90%	77.60%

Los resultados de las métricas de rendimiento de ambos modelos entrenados presentan variación entre el -1.18% y el 18.89%, del modelo con arquitectura base DenseNet201 con respecto al modelo con arquitectura MobileNet201. La arquitectura DenseNet201 se encuentra constituida por 402 capas de profundidad y ocupa un peso en memoria de 80 MB, por otro lado, la arquitectura MobileNetV2 está constituida por 105 capas de profundidad y ocupa un espacio en memoria de 16 MB.

6.3. Implementación del modelo en SBC

El dispositivo de despliegue se estableció en un sistema en modulo (*SOM – System on Module*, por su traducción al inglés) NVIDIA® Jetson Nano™, los requisitos para el funcionamiento de la placa, por lo que el primer paso para realizar la fase de despliegue del modelo es la instalación del Sistema Operativo de la NVIDIA® Jetson Nano™ (Véase en Imagen 6.8), antes de realizar este proceso es necesario que la tarjeta este conectada con los accesorios. En el Anexo A se muestra la hoja de datos de la placa utilizada.

En el sistema operativo ya se cuenta con una versión del lenguaje de programación Python, sin embargo, es necesario realizar la actualización de este a la versión con la que fue desarrollado el modelo, asi mismo la instalación de la librería de *TensorFlow* y *OpenCV*, todo esto a través de un gestor de paquetes PIP de Python. En la Imagen 6.8 se muestra una captura de pantalla del sistema listo para implementar el sistema clasificador.

El Sistema Operativo de NVIDIA® Jetson Nano™ está basado en el Sistema Operativo Ubuntu en du versión 18.01, para la implementación del sistema es necesario un editor de código para la programación de los módulos de procesamiento de imagen y el flujo del programa. Para el monitoreo de los recursos fue utilizado el programa *HTOP* que está incluido el Sistema Operativo.

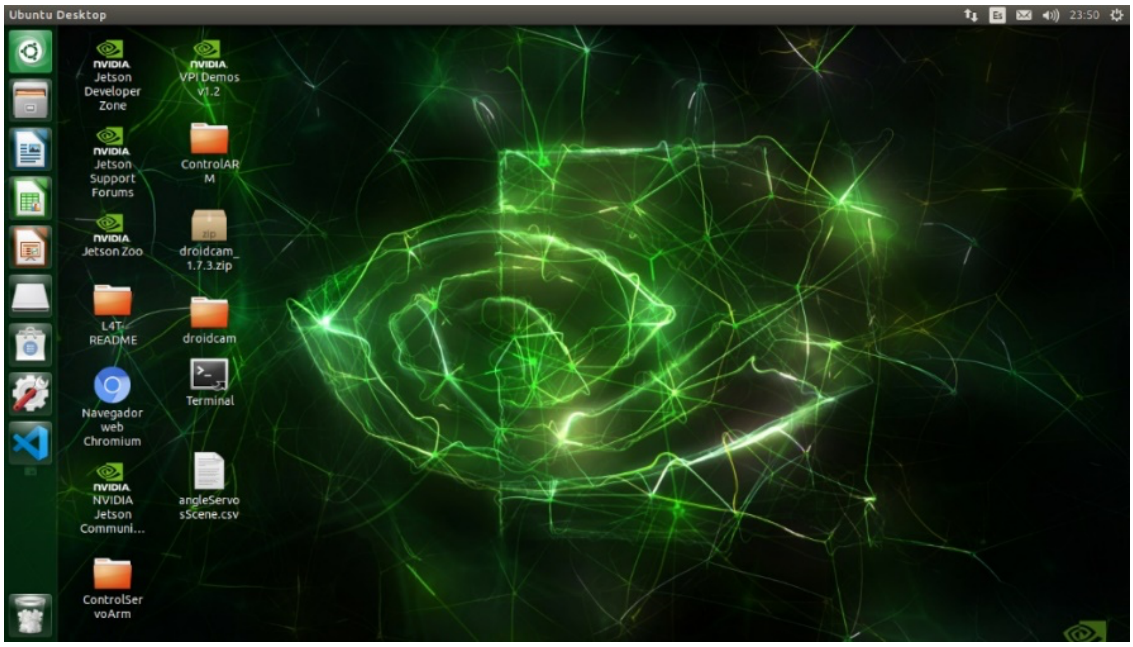


Imagen 6.7 Captura Escritorio de SO en NVIDIA® Jetson Nano™ listo para ejecutar el sistema clasificador.

Realizando un monitoreo de recursos del sistema operativo, en la Imagen 6.9 se muestra una captura de la salida en consola por el programa *HTOP*, en donde se muestra el espacio disponible y usado en memoria asi como el comportamiento de CPU, estos datos son con el sistema sin ningún proceso ejecutándose.

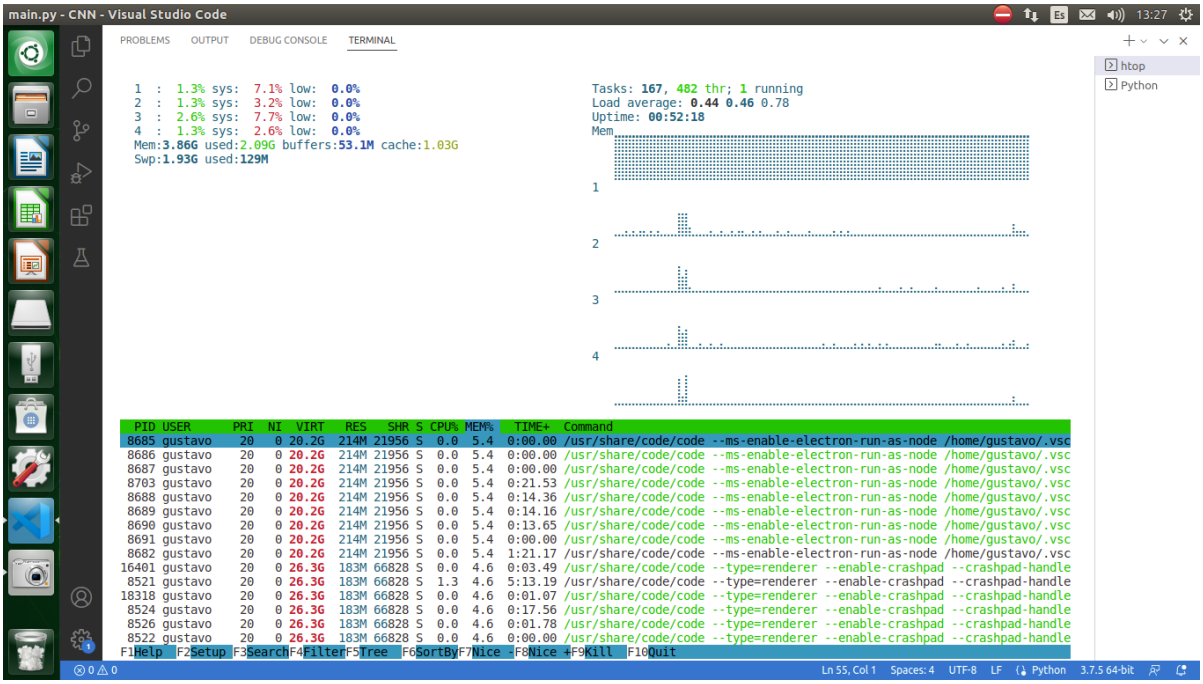


Imagen 6.8 Monitor de recursos del dispositivo NVIDIA® Jetson Nano™ sin tareas en ejecución.

Para la prueba del modelo de transfiere un conjunto de imágenes al dispositivo, estad imágenes son sin procesamiento en formato PNG, en un directorio del escritorio.

Como primer modelo para prueba se establece a el modelo de Red Neuronal Convolucional con arquitectura base MobileNetV2, este modelo es el menos pesado. Como se observa en la Imagen 6.10, el sistema es ejecutado con lo que se observa un aumento de uso de memoria RAM, con un aumento de 2.09G a 2.48G, asi mismo para los CPU, el porcentaje de uso de estos es menor al 57%.

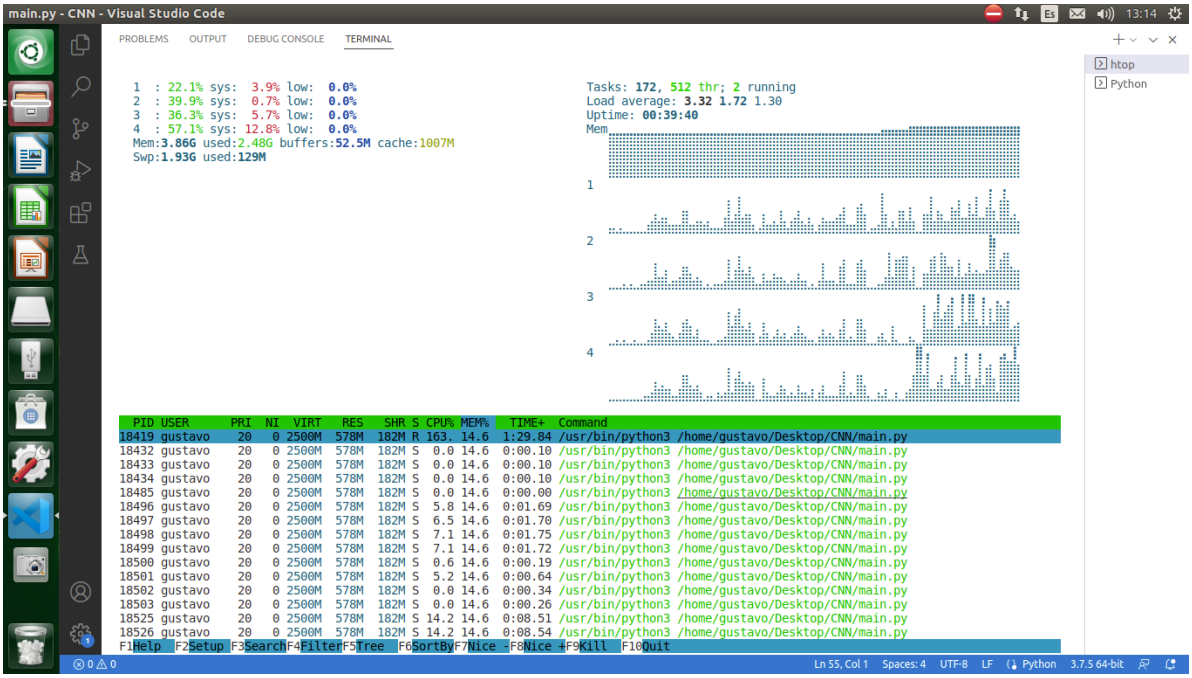


Imagen 6.9 Monito de recursos de NVIDIA® Jetson Nano™ durante la ejecución del sistema con el modelo de CNN con arquitectura base MobileNetV2.

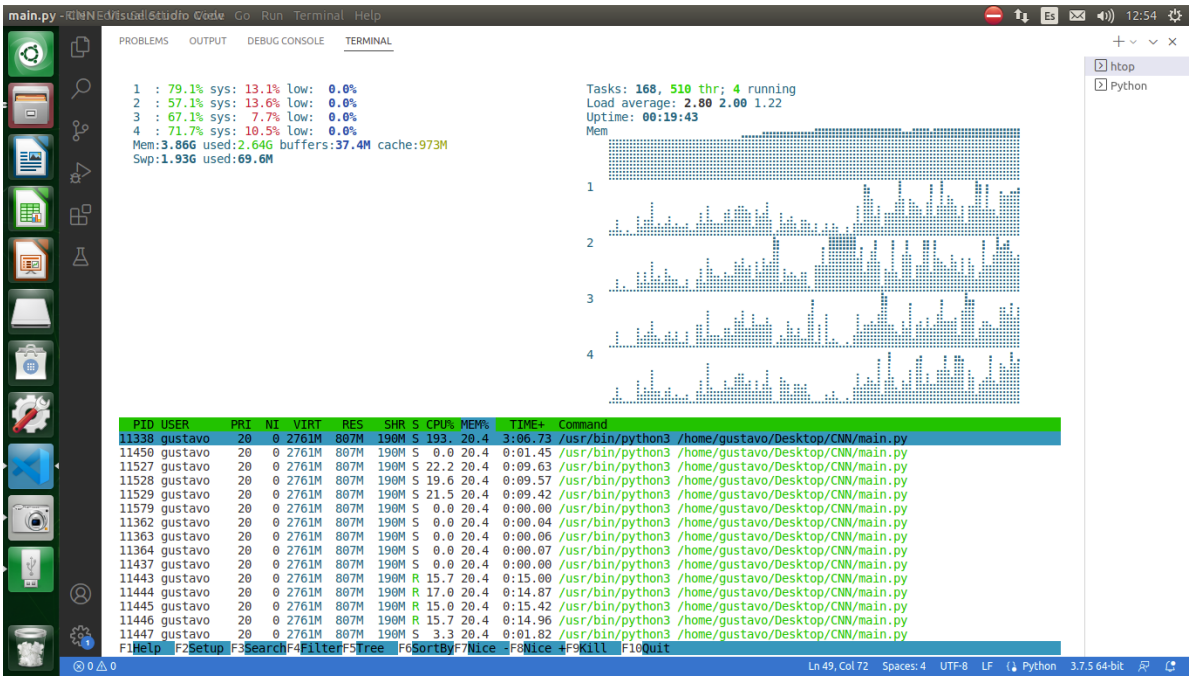
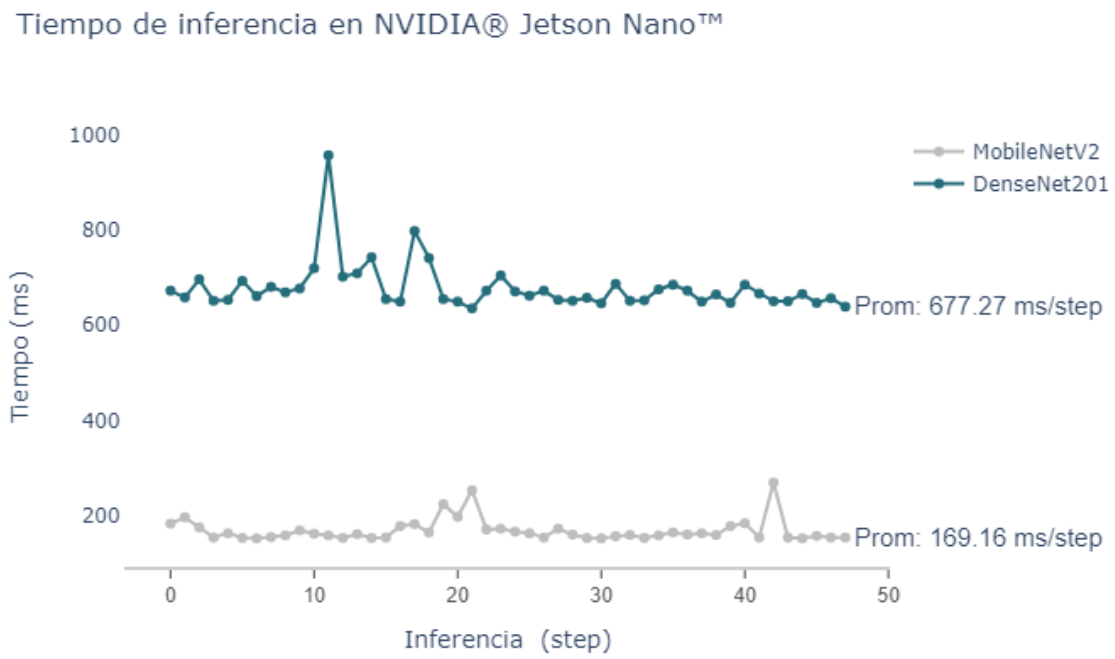


Imagen 6.10 Monito de recursos de NVIDIA® Jetson Nano™ durante la ejecución del sistema con el modelo de CNN con arquitectura base DenseNet201.

Para el modelo con la arquitectura base DenseNet201, se realiza la misma prueba, este modelo es más grande en capas de profundidad, así como el peso del modelo, su ejecución requiere de mayores recursos como se observa en la Imagen 6.11, el porcentaje de uso de los CPU es menor al 80%.

El tiempo de inferencia del modelo con arquitectura base MobileNetV2 por imagen es casi cuatro veces en comparación al con el modelo con arquitectura base DenseNet201, para esta prueba se tomó un conjunto de 48 imágenes de prueba y se obtuvo el tiempo de inferencia, los datos obtenidos del tiempo se graficaron para realiza una comparación de ambos modelos, en la Grafica 6.12 se muestran los valores de tiempo de inferencia de los modelos.



Gráfica 6.12 Tiempos de inferencia por imagen de los modelos en NVIDIA® Jetson Nano™.

La carga promedio del CPU (*Load Average* por su traducción al inglés) al ejecutar la inferencia de las imágenes se muestran en la Tabla 6.4, el valor a 15 minutos se presenta con 1.22, que representa el 30% del promedio de carga del CPU, al ser Quad-Core este soporta una carga promedio de 4 (Gunther, 2007).

Tabla 6.4 Load Average de la SOM durante la ejecución de inferencia de las imágenes.

Tiempo	Máx. Quad-core	Modelo con MobileNetV2	% Uso	Modelo con DenseNet201	% Uso
1 min	4 core	3.32 core	83.00%	2.80 core	70.00%
5 min	4 core	1.72 core	43.00%	2.00 core	50.00%
15 min	4 core	1.30 core	32.50%	1.22 core	30.50%
CPU prom.		38.85%			68.75%

El comportamiento del dispositivo no presenta comportamientos que indiquen una alteración a los recursos o funcionamiento durante la ejecución de estos sistemas de IA.

La prueba de los modelos en NVIDIA® Jetson Nano™ se realizó con 48 imágenes de fondo de ojo, las imágenes fueron tomadas de forma aleatoria del conjunto de datos de APTOS2019. Las Tabla 6.5 y 6.6 representan la matriz de confusion para los resultados obtenidos en la inferencia del modelos con arquitectura base MobileNet201 y DenseNEt201 respectivamente.

Tabla 6.5 Métricas de rendimiento para modelo con arquitectura MobileNetV2 en NVIDIA® Jetson Nano™.

Clase	Especificidad	Precisión	Exactitud	Sensibilidad	AUC
0	94.74%	77.78%	89.58%	70.00%	97.00%
1	86.84%	54.55%	81.25%	60.00%	80.00%
2	84.62%	40.00%	77.08%	44.44%	60.00%
3	82.05%	41.67%	77.08%	55.56%	63.00%
4	62.07%	15.38%	58.82%	40.00%	64.00%
macro	82.06%	45.87%	76.76%	54.00%	72.80%

Tabla 6.6 Métricas de rendimiento para resultados obtenidos de modelo con arquitectura DenseNet201 en NVIDIA® Jetson Nano™.

Clase	Especificidad	Precisión	Exactitud	Sensibilidad	AUC
0	97.37%	90.00%	95.83%	90.00%	97.00%
1	86.84%	50.00%	79.17%	50.00%	82.00%
2	82.05%	30.00%	72.92%	33.33%	67.00%
3	84.62%	50.00%	81.25%	66.67%	67.00%
4	62.30%	20.69%	61.97%	60.00%	75.00%
macro	82.63%	48.14%	78.23%	60.00%	77.60%

Los resultados de las métricas de rendimiento obtenidas al realizar las inferencias en NVIDIA® Jetson Nano™ presentan variaciones con un promedio de -2% y 6% de variación en las métricas de exactitud y precisión respectivamente, con respecto a las métricas obtenidas para los resultados obtenidos con el conjunto de prueba de validación en *Google Colab*.

CONCLUSIONES

El uso de la Inteligencia artificial en cada una de las áreas del conocimiento se ha vuelto relevante en los últimos años. En medicina, el incremento de pacientes con Diabetes Mellitus incrementa en el mundo de forma acelerada, debido a que es una enfermedad de la que derivan otras como lo es la Retinopatía Diabética, que si no son tratadas a tiempo en etapas tempranas pueden provocar consecuencias más graves y lesiones que resultan ser irreversibles debido al grado de avance.

En este trabajo se utilizaron técnicas de IA para la clasificación de imágenes en el área de oftalmología, utilizando las imágenes de fondo de ojo que son obtenidas de las bases *APTOS2019* y *Kaggle*. El entrenamiento previo de estas bases de datos fue realizado mediante arquitecturas de red neuronal convolucional.

La API de *Keras* presenta un benchmarking con 38 modelos de red neuronal convolucional entrenados para clasificar imágenes, a su vez valores de métricas de rendimiento como precisión y tiempo de inferencia, características del modelo como el número de capas que lo conforman y tamaño en memoria. A partir de estos datos fue posible realizar una selección de los modelos MobileNetV2 y DenseNet201 a través de un análisis grafico de la comparación de sus características con el resto de los modelos disponibles. Esta arquitectura es elegida por la cantidad de espacio que ocupa en memoria de 14MB, considerando que se tiene un dispositivo de despliegue con recursos de hardware limitado. Por otro lado, la arquitectura DenseNet201 fue seleccionada por sus 402 capas de profundidad, con la teoría de que un modelo de red profundo aprende mejor las características.

El conjunto de imágenes de *APTOS2019* es ideal para realizar un entrenamiento de clasificación de fondo de ojo en los niveles de retinopatía diabética, ya que proporciona imágenes con buena resolución e imágenes con características que permiten identificar características y realizar una clasificación. Por otra parte, este conjunto se encuentra desbalanceado, por lo tanto, se llevó a cabo un balance de datos utilizando el algoritmo de sobre muestreo para aumentar el número de elementos en las clases minoritas y submuestreo aleatorio para reducir el número de elementos de la clase mayoritaria. Cabe resaltar que el uso del conjunto de datos de *Kaggle* no es satisfactorio, aunque proporciona una base con

más de 35,000 imágenes, donde más del 75% pertenecen a la clase de imágenes de fondo de ojos sanos, es decir sin presencia de retinopatía diabética. En el conjunto de imágenes se encuentran imágenes con poca o nula visibilidad, que no proporcionan información necesaria para realizar una detección y aprendizaje de características.

Se encontró que un desenfoque gaussiano y la Ecualización del Histograma Adaptativo son procesamientos de imágenes ideales para realizar el resaltado de características como lo son bordes, en imágenes a color. Lo modelos de red neuronal especifican las dimensiones de las imágenes de entrada, lo que hace indispensable eliminar las zonas en negro o que no proporcionan información para el aprendizaje de características.

La plataforma de *Google Colab* fue de gran apoyo, no solo porque mediante el uso de recursos de hardware que proporciona, sino también porque fue posible realizar el entrenamiento con mayor rapidez, además de la importación de las imágenes desde un almacenamiento en la nube. El soporte de herramientas de desarrollo de Machine Learning como *TensorFlow* y *Keras*, fue posible realizar el entrenamiento cruzado, utilizando un dispositivo embebido como dispositivo de despliegue.

La ejecución de un sistema de *Deep Learning* en el Sistema en Módulo NVIDIA® Jetson Nano™ fue satisfactorio dado que utilizó menos del 80% de sus recursos. El modelo MobileNetV2 está diseñado para ejecutarse en este tipo de plataformas, los valores de porcentaje de uso de CPU para un modelo con arquitectura de este tipo tienen un promedio de -30% de variación con respecto a un modelo con arquitectura base DenseNet201, a pesar de que este último tiene casi cuatro veces el número de capas y uso de memoria con respecto al primero.

Las métricas de rendimiento para ambos modelos entrenados no cumplen con lo establecido en los objetivos, una complicación durante el desarrollo del proyecto. Los valores obtenidos en las métricas de rendimiento de precisión y función de perdida implicaron realizar diversos ajustes al conjunto de datos, así como el tipo de entrenamiento. La transferencia de aprendizaje de los modelos seleccionados presenta una gran ventaja, al ser modelos complejos por su número de capas, el entrenamiento desde cero de estos modelos llevaría demasiado tiempo. Sin embargo, es necesario realizar un ajuste fino a las ultimas capas para obtener mejores resultados en las métricas de rendimiento establecidas.

De acuerdo con los resultados obtenidos en la matriz de confusión y métricas de rendimiento, ambos modelos entrenados por el método de transferencia de aprendizaje son capaces de clasificar a las imágenes de fondo de ojo sanos, al obtener un valor en la métrica de precisión superior al 85% para la clase 0 en ambos modelos. No obstante, les es complicado determinar el nivel de retinopatía diabética superior a 1 (leve, severa, moderada y proliferada), pues no encuentran las características suficientes para determinar el nivel exacto a la que pertenece la imagen.

Finalmente, a través del análisis de la distribución de los datos en la matriz de confusión, se obtienen valores predichos por la CNN los cuales encuentran en un nivel superior o inferior al real.

En síntesis, la IA en sinergia con los sistemas embebidos están impulsando una gran ventaja competitiva en todas las áreas de conocimiento, donde un elemento clave son el análisis y procesamiento de los datos.

REFERENCIAS

Abellán, M. C. (2021). *Implementacion y aceleración de algoritmos de IA sobre Raspberry Pi.* España: Universidad de Alicante.

Algarabel, A. B. (2019). *Desarrollo de un sistema de análisis de imágenes médicas basados en técnicas de Deep Learning.* Madrid, España: Universidad Autonoma de Madrid.

APTOS. (2019). *Society, Asia Pacific Tele-Ophthalmology*. Obtenido de APTOS 2019: https://2019.asiateleophth.org/

Artola Moreno, Á. (2019). *Clasificación de imágenes usando redes neuronales* . Sevilla: Universidad de Sevilla.

Arunkumar, R., & Karthigaikumar, P. (2017). Multi-retinal disease classification by reduced Deep Learning features. *Neuronal Comput & Applic, 28*(2), 329-334.

Aswathi, T. (2021). Transfer Learning approach for grading of Diabetic Retinopathy. *Journal of Physics: Conference Series*, 1767.

Bagnato, J. (29 de Noviembre de 2018). *Aprende Machine Learning*. Recuperado el 2024 de Febrero de 05, de ¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

Baischer, L., Wess, M., & Taherinejad, N. (2021). Learning on Hardware: A tutorial on Neuronal Network Accelerators and Co-Processors. *arXiv, 0*(0), 0-29.

Barrio A, J. (2021). *Redes Neuronlales Convolucionales*. Recuperado el 30 de Agosto de 2021, de https://www.juanbarrios.com/redes-neurales-convolucionales/

Barrot, J., & Franch, J. (2019). *Atlas en Retinoptía Dibética y lectura de retinografías.* Barcelona: MSD.

Basulto-Lantsova, A. (2020). Performance comparative of OpenCV Tempate Matching method on Jetson TX2 and Jetson NAno developer kits. *TecNm en Celaya*, 8012-816.

Bueno-Garcia, R., Mira, X., & Flores Peredo, V. (16 de Septiembre de 2021). *Retinopatía Diabética en México.* Recuperado el 18 de Octubre de 2021, de https://www.iapb.org/news/retinopatia-diabetica-en-mexico/

Chen, Y., Xie, Y., Song, L., Chen, F., & Tang, T. (2020). A survey of Accelerator Architectures for Deep Neuronal Networks. *Engineering, 6*, 264-274.

Chollet, F. (2018). *Deep Learning with Python.* USA: Manning.

Coccé, J. G. (2017). *Clasificación de imágenes en sistemas embebidos usando redes neuronales.* Buenos Aires: Universidad de Buenos Aires.

Crespo, A. B. (2020). *Integración de redes neuronaes en sistemas empotrados. Clasificación de imagen con RaspberryPi.* España: Universidad Politecnica de Valencia.

Firke, S. N., & Jain, R. B. (2021). Convolutional Neuronal Network for Diabetic Retinopathy Detection. *Proceedings of the International Conference on Artificial Intelligence and Smart Systems*, 549-553.

Gangwar, A. K., & Ravi, V. (2021). Diabetic Retinopathy Detection Using Transfer Learning and Deep Learning. *Evolution in Computational Intelligence, 1*, 679-689. doi:https://doi.org/10.1007/978-981-15-5788-0_64

García García, Y., Rodríguez Guillén, R., & Taboada Crispi, A. (2017). Mapeo de imágenes digitales de fondo de ojo atendiendo a rasgos de textura. *Revista Cubana de Ciencias Informáticas, 11*(1), 106-121.

Ghosh, S. (2023). Transfer-Ensemble Learning based Deep Convolutional Networks for Diabetic Retinopathy Classification. *Department of Computer Science and Engineering Jadaypur University*.

González, E., Villamizar Luna, W. D., & Fajardo Araiza, C. A. (2019). A Hardware Accelerator for The Inference of a Convolutional Neuronal Network. *Ciencia e Ingeniería Neogranadina, 30*(1), 107-116.

González, J. S. (2011). *Sistema de Diagnóstico Asistido por Computadora para la detección de la Retinpatía Diabética No Proliferada usando la Red Neuronal de Retropropagación.* México: Instituto Politécnico Nacional.

Grandini, M. (2020). Metrics for multi-class classification: a overview. *A white paper*, 2.

Gunther, N. J. (Octubre de 2007). *Understanding load averages and strech factors.* Obtenido de Linux Magazine: https://www.linux-magazine.com/content/download/62593/485442/version/1/file/Load_Average.pdf

Habib, G., & Qureshi, S. (2020). Optimization and acceleration of convolutional neuronal networks: a Survey. *Jorunal of Kind Saud University - Computer and information Sciences*.

Hameed, S. (2022). Hybrid Retinal Image Enhancement Algorithm for Diabetic Retinopathy Diagnostic Using Deep Learning Model. *IEEE Access*, 73079-73086.

Huanca, A. L. (2021). *Diagnóstico de la retinopatia diabética usando algoritmos de clasificación.* Moquegua, Peru: Universidad Nacional de Moquegua.

IBM. (2024). *¿Qué son las redes neuronales convolucionales?* Obtenido de IBM Cloud: https://www.ibm.com/mx-es/topics/convolutional-neural-networks

Jang, Y.-W. (2021). Developing a Compressed Object Detection Model based on YOLO for Deployment on Embedded GPU Plaftoform of Autonomous System. *School of Mechanical and Control Enginnering*.

Kaggle. (Febrero de 2015). doi:10.1177/193229680900300315.

Kandel, I., & Catelli, M. (2020). Transfer Learning with Convolutional Neuronal Networks for Diabetic Retinopathy image Classification. A Review. *Applie Sciences*, 1. doi:https://doi.org/10.3390/app10062021

Karakaya, M., & Hacisoftaoglu, R. E. (2020). Comparison of smartphone-based retinal imaging system for diabetic retinopathy detection using Deep Learning. *BMC Bioinformatics, 21*(4), 18.

Keras. (2023). *Keras Applications*. Obtenido de Keras Applications: https://keras.io/api/applications/

Kohonen, T. (1988). An Introduction to Neuronal Computing. *Pergamon Journals Ltd.*, 3-16.

Lakshminaranan, V., Kheradfallah, H., & Jothi Balaji, J. (2021). Automated Detection and Diagnosis of Diabetic Retinopathy: A Comprehensive Survey. *Journal of Imagin, 7*(165), 2-26. doi:https://doi.org/10.3390/jimaging7090165

Lam, C., & Yu, C. (2017). Retinal Lesion Detection With Deep Learning Using Patches. *Invest Opbthalmol*, 590-597. doi: https://doi.org/10.1167/

Lands, A. (2020). Implementation of deep learning based algotihms for diabetic retinopathy classification from fundus images. *Proccedings of the Fourth International Conference on Trends in Electronics ans Informatics (ICOEI 2020)*, 1028-1032.

Li, F., Liu, Z., Chen, H., Jiang, M., Zhang, X., & Wu, Z. (2019). Automatic Detection of Diabetic Retinopathy in Retinal Fundus Photographs Based on Deep Learning Algorithm. . *Translational Vsion Science & Technology*.

Liang, X. (2020). *Ascend AI Processor Architecture and Programming* (First ed.). Cambridge: Elseiver.

Madera, N. S., Gutierrez, J. E., & Gamarra, M. c. (2020). Metodología de Segmentación de la Estructura Ocular en Imágenes de Fondo de Ojo de Pacientes con Retinopatía Diabética. *Prospectiva, 18*(2).

Mamadi Noa, R. W. (2018). *Diseño de un sistema de detección de retinopatía diabética con procesamiento de imagenes de retina.* Perú: Universidad Nacional de San Agustín.

Martín Abadi, A. A. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *White Paper*, 1. Obtenido de https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf

MathWorks, T. (2023). *MathWorks*. Recuperado el 22 de Septiembre de 2023, de MATLAB for Artificial Intelligence: https://www.mathworks.com/

Md. Milon Islam, M. Z.-R. (2022). Diagnosis of COVID-19 from X-rays using combined CNN-RNN architecture. *BenchCouncil Transactions on Benchmarks, Standards and Evaluations, 2*. doi:https://doi.org/10.1016/j.tbench.2023.100088

Merritt, R. (20 de Enero de 2023). *NVIDIA*. Obtenido de What Is AI Computing?: https://blogs.nvidia.com/blog/what-is-ai-computing/

Merwedel, P. (2018). *Embedded System Design. .* Dortmund, Germany: Springer.

Microsoft. (01 de 08 de 2023). *Microsoft Learn.* Obtenido de Evaluación de los resultados del experimento de aprendizaje automático automatizado: https://learn.microsoft.com/es-es/azure/machine-learning/how-to-understand-automated-ml?view=azureml-api-2

Molina Ovando, F. E. (2019). *Desarrollo de un perceptrón Multicapa en Raspberry Pi.* Valparaiso: Pontificia Universidad Católica de Valparaiso.

Moons, B., Bankman, D., & Verhelst, M. (2019). *Embedded Deep Learning. Algoritms, Architectures and Circuits for Always-on Neuronal Network Processing.* Springer. doi:https://doi.org/10.1007/978-3-319-99223-5

Navarro, J. C., Peña, C. B., & Escorcia, J. G. (2020). Una revisión de los métodos de Deep Learning aplicados a la detección automatizada de la Retinopatía Diabética. *Sextante, 23*, 12-27.

Noergaard, T. (2005). *Embedded Systems Architecture. A comprensive Guide for Engineers and Programmers.* Burlington, USA: Elseiver Inc.

OpenCV. (19 de 19 de 2023). *Open Sorce Computer Vision* . Obtenido de OpenCV: https://docs.opencv.org/4.x/d5/dc4/tutorial_adding_images.html

Perdomo Charry, O. J., & González Augusto, F. (2019). A Systematic Review of Deep Learning Methods Applied to Ocular Images. *Ciencia e Ingeniería Neogranadina, 30*(1), 9-26.

Pérez Cerdeira, I. J. (2021). *Convolucionales, Aceleración hardware para Inferencia en Redes Neuronales.* Concepcion, Chile: Universidad de Concepción.

Ponce Cruz, P. (2010). *Inteligencia Artifical con Aplicaciones a la Ingeniería* (Primera ed.). México: Alfaomega.

Pozo, E. R. (2019). *Análisis de clasificadores supervisados para detectar la estructura en imágenes retinianas patológicas.* Quito: Escuela Politécnica Nacional.

Puchtler, P., & Peinl, R. (2020). Evaluation of Deep Learning Accelerators for Object Detecion at the Edge. *KI 2020: Advances in Artificial Intelligence*, 320-325. doi:https://doi.org/10.1007/978-3-030-58285-2_29

Rahim, S. (2023). Global average pooling. *Medium*. Obtenido de https://medium.com/@jackneutron786/global-average-pooling-3deb58466fef

Ranchschaert, E., & Morozov, S. (2019). *Artificial Intelligence in Medical Imaging. Opportunities, Applications and Risks.* Moscow, Russia: Springer. doi:https://doi.org/10.1007/978-3-319-94878-2

Rodríguez, N., Murazzo, M., Ortega, M., & Lund, M. I. (2019). *XXI Workshop de Investigadores en Ciencias de la Computación* (Primera ed.). San Juan, Argentina: UNSJ.

Ronneberger, O., Philipp, F., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation.* Freiburg, Germany: Computer Science Department and BIOSS.

Rubini, S., Nithil, S., & Kunthavai, S. (2019). Deep Convolutional Neuronal Network-Based Diabetic Retinopathy Detection in Digital Fundus Image. *Soft Computing and Signal Processing, 900*, 201-209. doi:http://doi.org/10.1007/978-981-13-3600-3_19

Ruiz Varela, J. M., Ortega Cisneros, S., & Pedroza de la Crúz, A. (2016). Reconocimiento de micro partículas de polen con algoritmos de procesamiento de imágenes implementados en dispositivos reconfigurables. *ReCIBE. Revista electrónica de Computación, Informática, Biomédica y Electrónica, 5*(2).

Russell, R. (2018). *MACHINE LEARNING. STEP-BY-STEP GUIDE TO IMPLEMENT MACHINE LEARNING ALGORITMS WITH PYTHON.*

Sáenz Bajo, N., & Álvaro Ballesteros, M. (2002). Redes nuronales: concepto, aplicaciones y utilidad en medicina. *Aten Primaria, 30*(2), 119-120.

Saranya, & Umamaheswari. (2021). Classification of Different Stages of Diabetic Retinopathy using Convolutional Neuronal Network. *2021 2nd International Conference on Computation, Automation and Knowledge Mangement (ICCAKM) Amity University*.

Scanlon, P. H. (Febrero de 2019). Diabetic Retinopathy. *Medicine, 47*, págs. P77-85. doi:https://doi.org/10.1016/j.mpmed.2018.11.013

Sharda, A. (2021). Image Filters: Gaussian Blur. *Medium*. Obtenido de https://aryamansharda.medium.com/image-filters-gaussian-blur-eb36db6781b1

Siddharth, M., Hao, L., & Jiabo, H. (2020). Machine learning assisted segmentation of scanning electron microscopy images of organic-rich shales with feature extraction and feature ranking. En M. Siddharth, L. Hao, & H. Jiabo, *Machine Learning for Subsurface Characterization* (pág. 295). Texas: Gulf Professional Publishing. doi:https://doi.org/10.1016/C2018-0-01926-X

Sikder, N., Masud, M., & Kumar Bairagi, A. (2021). Severity Classification of Diabetic Retinopathy Using an Ensemble Learning Algorithm through Analyzing Retinal Images. *Symmetry, 13*(670), 26. doi:https://doi.org/10.3390/sym13040670

Soto Orozco, O. A., & Corral Sáenz, A. D. (2019). Análisis del desempeño de redes neuronales profundas para segmentación semántica en hardware limitado. *ReCIBE. Revista electrónica de Computación, Informática., Biomédica y Electrónica., 8*(2), 1-21.

Stabile Bernabé, E. (2021). *Optimización del producto matricial sobre dispositivos de bajo consumo ara inferencia en Deep Learning.* España: Univerisidad Politecnica de Valencia.

Suzen, A. A. (2020). Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry PI using Deep-CNN. *Department of Computer Technologies Isparta University of Applied Sciences*.

Takano, S. (2017). *Thinking Machines. Machine Learning and its Hardware Implementation* (Primera ed.). Japan: Elseiver Inc.

Tatsat, H., Puri, S., & Lookabaugh, B. (2021). *Machine Learning & Data Science Blueprints for Finance.* Estados Unidos de America: O´Reilly.

Toro Valderas, A. J. (2020). *Implementación de redes neuronales convolucionales en Raspberry Pi con Movidius Neuronal Compute Stick.* Sevilla: Universidad de Sevilla.

Valderrama Molano, J. A. (2017). *Clasificación de Objetos Usando Aprendizaje Profundo Implementado en un Sistema Embebido.* Santiago de Cali: Universidad Autónoma de Occidente.

Valladares, S., & Toscano, M. (2021). Performance Evaluation of the Nvidia Jetson Nano Through a Real-Time Machine Learning Application. *intelligent Human Systems Integration 2021*, 333-349. doi:https://doi.org/10.1007/978-3-030-68017-6_51

Vargas, F. (2020). *A Literature Review on Embedded Systems.* IEEE LATIN AMERICA TRANSACTIONS. doi:https://doi.org/10.1109/TLA.2020.9085271

Velázquez González, J. S. (2011). *Sistema de Diagnositico Asistido por Computadora para la detección de la Retinpatia Diabética o proliferada usando la Red Neuronal de Retropropagación.* México: IPN.

Wang, X. (2021). Promote Retinal Lesion Detection for Diabetic Retinopathy Stage Classification. *2020 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, 31-34.

Yeung, T. (19 de Octubre de 2022). *Go to the Edge with NVIDIA.* Obtenido de What Is Edge Computing?: https://resources.nvidia.com/en_us_fleet_command/en-us-fleet-command/what-is-edge-computing?pflpid=3725&lb-mode=preview

Zhi-Hua, Z. (2021). *Machine Learning.* Singapore: Springer. doi:https://doi.org/10.1007/978-981-15-1967-3

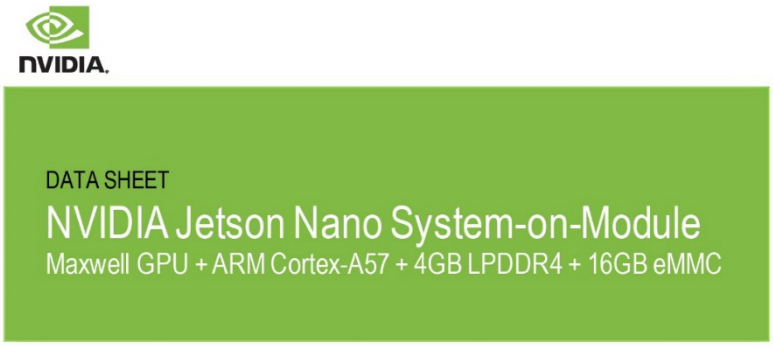
Zhou, K. (2021). A review of deep learning in medical imaging: Imaging traits, technology trends, case studies with progress highlights, and future promises. *EMC*. doi:https://arxiv.org/pdf/2008.09104.pdf

Zilly, J. G. (2015). Boosting Convolutional Filters with Entropy Sampling for Optic Cup and Disc Image Segmentation from Fundus Images. *Iowa Research Online*, 153-160.

ANEXOS

ANEXO A

Obtenido de: https://developer.nvidia.com/embedded/jetson-nano



Maxwell GPU ◊

128-core GPU | End-to-end lossless compression | Tile Caching | OpenGL® 4.6 | OpenGL ES 3.2 | Vulkan™ 1.1 | CUDA® | OpenGL ES Shader Performance (up to): 512 GFLOPS (FP16)
Maximum Operating Frequency: 921MHz

CPU

ARM® Cortex®-A57 MPCore (Quad-Core) Processor with NEON Technology | L1 Cache: 48KB L1 instruction cache (I-cache) per core; 32KB L1 data cache (D-cache) per core | L2 Unified Cache: 2MB | Maximum Operating Frequency: 1.43GHz

Audio

Industry standard High Definition Audio (HDA) controller provides a multichannel audio path to the HDMI interface.

Memory

Dual Channel | System MMU | Memory Type: 4ch x 16-bit LPDDR4 | Maximum Memory Bus Frequency: 1600MHz | Peak Bandwidth: 25.6 GB/s | Memory Capacity: 4GB

Storage

eMMC 5.1 Flash Storage | Bus Width: 8-bit | Maximum Bus Frequency: 200MHz (HS400) | Storage Capacity: 16GB

Boot Sources

eMMC and USB (recovery mode)

Networking

10/100/1000 BASE-T Ethernet | Media Access Controller (MAC)

Imaging

Dedicated RAW to YUV processing engines process up to 1400Mpix/s (up to 24MP sensor) | MIPI CSI 2.0 up to 1.5Gbps (per lane) | Support for x4 and x2 configurations (up to four active streams).

Operating Requirements

Temperature Range (T_j): -25 – 97C* | Module Power: 5 – 10W | Power Input: 5.0V

Display Controller

Two independent display controllers support DSI, HDMI, DP, eDP:

- MIPI-DSI (1.5Gbps/lane): Single x2 lane | Maximum Resolution: 1920x960 at 60Hz (up to 24bpp)
- HDMI 2.0a/b (up to 6Gbps) | DP 1.2a (HBR2 5.4 Gbps) | eDP 1.4 (HBR2 5.4Gbps) | Maximum Resolution (DP/eDP/HDMI): 3840 x 2160 at 60Hz (up to 24bpp)

Clocks

System clock: 38.4MHz | Sleep clock: 32.768kHz | Dynamic clock scaling and clock source selection

Multi-Stream HD Video and JPEG

Video Decode

- H.265 (Main, Main 10): 2160p 60fps | 1080p 240fps
- H.264 (BP/MP/HP/Stereo SEI half-res): 2160p 60fps | 1080p 240fps
- H.264 (MVC Stereo per view): 2160p 30fps | 1080p 120fps
- VP9 (Profile 0, 8-bit): 2160p 60fps | 1080p 240fps
- VP8: 2160p 60fps | 1080p 240fps
- VC-1 (Simple, Main, Advanced): 1080p 120fps | 1080i 240fps
- MPEG-2 (Main): 2160p 60fps | 1080p 240fps | 1080i 240fps

Video Encode

- H.265:2160p 30fps | 1080p 120fps
- H.264 (BP/MP/HP): 2160p 30fps | 1080p 120fps
- H.264 (MVC Stereo per view): 1440p 30fps | 1080p 60fps
- VP8: 2160p 30fps | 1080p 120fps

JPEG (Decode and Encode): 600 MP/s

Peripheral Interfaces

xHCI host controller with integrated PHY: 1 x USB 3.0, 3 x USB 2.0 | USB 3.0 device controller with integrated PHY | EHCI controller with embedded hub for USB 2.0 | 4-lane PCIe: one x1/2/4 controller | single SD/MMC controller (supporting SDIO 4.0, SD HOST 4.0) | 3 x UART | 2 x SPI | 4 x I2C | 2 x I2S: support I2S, RJM, LJM, PCM, TDM (multi-slot mode) | GPIOs

Mechanical

Module Size: 69.6 mm x 45 mm | PCB: 8L HDI | Connector: 260 pin SO-DIMM

Note: Refer to the software release feature list for current software support; all features may not be available for a particular OS.

◊ Product is based on a published Khronos Specification and is expected to pass the Khronos Conformance Process. Current conformance status can be found at www.khronos.org/conformance.

* See the *Jetson Nano Thermal Design Guide* for details. Listed temperature range is based on module T_j characterization.

ANEXO B

Función en *Python* para recorte de zonas en negro de imagen.

```
def crop_dark(img, tol=7):
    if img.ndim == 2: # dimensiones de la imagen
        mask = img>tol
        return [np.ix_(mask.any(1), mask.any(0))]
    elif img.ndim == 3:
        gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        mask = gray_img > tol
        check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
        if (check_shape == 0): # demasiado oscuro para recortar todo
            return img # imagen original
        else:
            img1 = img[:,:,0][np.ix_(mask.any(1), mask.any(0))]
            img2 = img[:,:,1][np.ix_(mask.any(1), mask.any(0))]
            img3 = img[:,:,2][np.ix_(mask.any(1), mask.any(0))]
            img = np.stack([img1, img2, img3], axis=-1)
        return img
```

ANEXO C

Función en *Python* para aplicar Ecualización del Histograma Adaptativo en imágenes a color RGB:

```
def applyCLAHE(colorimage):
    clahe_model = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
    colorimage_b = clahe_model.apply(colorimage[:,:,0])
    colorimage_g = clahe_model.apply(colorimage[:,:,1])
    colorimage_r = clahe_model.apply(colorimage[:,:,2])
    im = np.stack((colorimage_b,colorimage_g,colorimage_r), axis=2)
    return im
```