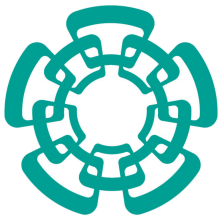


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco
Departamento de Ingeniería Eléctrica
Sección de Electrónica del Estado Sólido

"Sistema Integrado en Hardware para el Análisis y Procesamiento Básico de Imágenes Digitales"

T E S I S

Que presenta

Ivan Shin Cao Chong Cervantes

Para obtener el grado de

Maestro en Ciencias

En la Especialidad de Ingeniería Eléctrica

Directores de Tesis

Dr. Mario Alfredo Reyes Barranca

Dr. José Antonio Moreno Cadenas

Ciudad de México Junio, 2024

Agradecimientos

Esta tesis fue posible gracias al apoyo de varias personas e instituciones a las que les quiero mostrar mi más sincero agradecimiento.

Al CONAHCYT, por el apoyo económico para poder realizar mis estudios de posgrado.

Al Cinvestav, por brindarme la oportunidad de continuar mi formación académica en su prestigiosa institución.

A todo el personal de la SEES, por la disposición de ayuda y orientación en todo momento. En especial a Yesenia, por su gran gentileza y eficacia en resolver todos los trámites necesarios y a Mónica y Martín, por su amabilidad y mostrar siempre un gran apoyo.

A todos los investigadores y auxiliares de investigación, en especial a mis profesores, por compartir sus destacados conocimientos en su noble labor.

Al personal de laboratorio de VLSI, por su interés y disposición de apoyo en todo momento.

A mis asesores, el Dr. Mario Alfredo Reyes Barranca y el Dr. José Antonio Moreno Cadenas, por la oportunidad y confianza de trabajar con ellos, además por transmitirme sus conocimientos, brindarme su orientación y apoyarme en todos los sentidos, con gran profesionalismo, amabilidad y paciencia, pero sobre todo, con una gran calidad humana.

Al Dr. Álvaro Anzueto Ríos, por compartir gran parte de su tiempo conmigo para brindarme sus sobresalientes conocimientos y consejos, así como su apoyo, ayuda y orientación durante todo el desarrollo de esta tesis, su apoyo fue determinante para alcanzar los objetivos planteados.

Al Dr. Felipe Gómez Castañeda, por su amable disponibilidad en la revisión esta tesis y

compartir sus observaciones y recomendaciones.

Al M. en C. Luis Martín Flores Nava, por todo el apoyo brindado durante el desarrollo y revisión de esta tesis.

A mi familia y amigos, por su empuje y apoyo.

A Thais, por su constante impulso, alivio, paciencia y apoyo.

El mayor de mis agradecimientos a mi mamá y a mis hermanos, por su inmenso e invaluable apoyo y paciencia.

Y un especial agradecimiento a los optometristas Cristina Rodríguez López y Sergio Arturo Basave Garnica, por compartir sus conocimientos y opiniones, así como su disposición de ayuda en todo momento.

A todos ellos, mi más profundo y sincero agradecimiento.

Índice general

Índice de figuras

Índice de tablas

Sección de código

Resumen

En este trabajo se presenta el desarrollo de un sistema de procesamiento de imágenes de fondo de ojo implementado sobre un sistema embebido, con la finalidad de facilitar un diagnóstico en la detección de la retinopatía diabética por medio del análisis automático de este tipo de imágenes.

Para ello, se propone realizar el procesamiento en dos partes. El primero consiste en mejorar el contraste de las imágenes por medio de la lógica difusa. Aprovechando las características que esta técnica ofrece, el sistema de mejora de contraste será adecuado para este tipo de imágenes al tomar en cuenta los conocimientos de personal médico especializado en su desarrollo, con el objetivo de resaltar los signos de la retinopatía diabética sin alterar otros detalles en las imágenes. La segunda parte del sistema corresponderá al análisis de las imágenes para detectar aquellas con retinopatía diabética. Para lograrlo, se propone utilizar redes neuronales convolucionales por medio de un modelo de aprendizaje máquina, utilizando imágenes con la mejora de contraste para su entrenamiento.

El sistema de procesamiento final será el conjunto de ambas partes implementadas sobre un sistema embebido. El hardware emplear será una Raspberry Pi 4 modelo B de 8 GB, cuyas características de procesador y capacidad de memoria RAM, lo hacen capaz de implementar el sistema desarrollado.

Con esto se pretende demostrar que las técnicas de lógica difusa y de redes neuronales son capaces de funcionar en conjunto sobre un dispositivo de bajo costo, incursionando en el campo del análisis de imágenes médicas, considerando la posibilidad de apoyar al personal médico.

Abstract

This paper presents the development of an eye fundus image processing system implemented on an embedded system, with the purpose of facilitating a diagnosis in detection of diabetic retinopathy through the automatic analysis of this type of images.

It is proposed to perform the image processing in two parts. The first part pretend to enhance the contrast using fuzzy logic. Taking advantage of the characteristics this technique offers, the system of contrast enhancement will be suitable for this type of images by using the knowledge of specialized medical personnel in its development, with the aim of highlighting the signs of diabetic retinopathy without altering another details in this images. The second part of the system corresponds in the analysis of the images, in order to detect those with diabetic retinopathy. To achieve this, it is proposed to use convolutional neural networks through a machine learning model, using images with contrast enhancement for its training.

The final processing system will be the set of both parts implemented on an embedded system. The hardware used will be a Raspberry Pi 4 model B of 8 GB, whose processor and RAM capacity characteristics make it capable of implementing the developed system.

This aims to demonstrate that fuzzy logic and neural network techniques are capable of working together on a low-cost device, entering in the field of medical image analysis, considering the possibility of supporting medical personnel.

Objetivos

Objetivo general

Implementación de un sistema embebido en una Raspberry Pi 4 para procesar y analizar imágenes digitales empleando lógica difusa y redes neuronales.

Objetivos particulares

1. Desarrollo de un sistema de inferencia difuso para el procesamiento de mejora de contraste en imágenes digitales.
2. Extracción de características de imágenes de fondo de ojo para discernir entre imágenes con retinopatía diabética y sin retinopatía diabética.
3. Implementación del sistema de inferencia difuso para la mejora de contraste de imágenes digitales en la Raspberry Pi 4.
4. Implementación de la red neuronal para la extracción de características de imágenes de fondo de ojo en la Raspberry Pi 4.

Capítulo 1
Introducción

En este trabajo se plantea la creación de un sistema capaz de procesar imágenes digitales para extraer la mayor cantidad de información de ellas y poder analizarlas. Para ello, se propone implementarlo sobre un sistema embebido, es decir, un sistema de computación incorporado sobre un dispositivo electrónico de tamaño reducido y de bajo costo, con el fin de que su mantenimiento y actualización sean prácticos para que el sistema desarrollado pueda ser accesible.

Teniendo en cuenta que en esta tesis se tratará con imágenes digitales y que muchas de sus características visuales, tal como su calidad o interpretación, varían dependiendo de múltiples factores, los cuales son de cierta manera subjetivos, es conveniente utilizar técnicas de procesamiento de información computacionales que empleen la teoría de probabilidad para su ejecución. Éstas son conocidas como *Soft Computing* y permiten abordar problemas con datos imprecisos, ambiguos o incompletos de una manera flexible y adaptable, de esta manera se pretende reducir la posible incertidumbre introducida por la interpretación personal de las imágenes. Entre las diversas técnicas que abarca *Soft Computing* se encuentran la lógica difusa, las redes neuronales artificiales, los algoritmos metaheurísticos y la programación genética.

En términos generales, la metodología experimental de este trabajo se divide en dos partes, en la primera se pretende mejorar la calidad de las imágenes empleando la lógica difusa, buscando que la cantidad de ajuste que se le aplique a la imagen dependa de su calidad original. Dicho de otra manera, mientras menor sea la calidad original, el ajuste deberá ser mayor y si, por el contrario, la calidad inicial es alta, el ajuste deberá ser mínimo o nulo. La segunda parte se enfoca en la extracción y análisis de la información contenida en las imágenes, planteando

como solución una red neuronal de tipo convolucional o CNN (del inglés *Convolutional Neural Network*). Ambas partes del desarrollo experimental se realizaron y optimizaron en software utilizando una computadora personal y posteriormente se implementaron sobre el sistema embebido. De esta manera, la estructura de la presente tesis sigue dicha metodología, teniendo como introducción y marco teórico los capítulos 1 y 2, respectivamente; en los capítulos 3 y 4 se presenta el desarrollo experimental mencionado; los resultados se describen en el capítulo 5 y las conclusiones en el capítulo final.

1.1 Sistemas embebidos

Un sistema computacional se puede definir como el conjunto de dispositivos e instrucciones que interactúan para llevar a cabo un proceso. Con esta definición es posible afirmar que los sistemas computacionales están compuestos por hardware y software, donde hardware es el conjunto de componentes físicos que conforman un sistema informático o dispositivo electrónico y software es el aspecto lógico que lo controla. Dentro de la gran variedad de sistemas computacionales se encuentran los sistemas embebidos, los cuales incorporan el hardware y software dentro de un solo dispositivo electrónico y se enfocan en realizar una tarea específica.

Por lo general, los sistemas embebidos están diseñados para ser compactos y eficientes, ya que utilizan la mayoría de sus recursos para realizar la función específica para lo que fueron configurados. Existe una enorme variedad de estos sistemas, entre los cuales destacan los de automatización industrial, los sistemas para la domótica, los automotrices y los médicos. Actualmente diversos dispositivos electrónicos se pueden utilizar como sistemas embebidos, tal como los denominados FPGA (del inglés: *Field Programmable Gate Array*) y los sistemas basados en microprocesadores, como las microcomputadoras, que son computadoras compactas dentro de una sola tarjeta de circuito impreso.

La elección entre estos dispositivos depende completamente de las necesidades que se ten-

gan, ya que cada uno tiene sus ventajas y desventajas con respecto a los demás. Un FPGA es un dispositivo semiconductor compuesto por una matriz de bloques lógicos cuyas interconexiones son programables [1], sus principales características son su flexibilidad, velocidad y rendimiento por ser un dispositivo diseñado para realizar procesamientos paralelos. Por otra parte, los sistemas basados en microprocesadores, por tener todos los componentes de un computador tanto en hardware como en software, se suelen utilizar como ordenadores de propósito general, siendo capaces de configurarse para realizar tareas específicas.

Entre los dispositivos basados en microprocesadores más conocidos se encuentran las tarjetas de Arduino, Raspberry, NVIDIA Jetson, Odroid y Adafruit Feather. Cada uno de ellos presenta características particulares, sin embargo, para este trabajo los que mejor se adaptan en las tareas a desarrollar son las placas de Raspberry y NVIDIA Jetson. Las tarjetas de ambos comparten la característica de ser ordenadores con un sistema operativo (SO) basado en el kernel de Linux, por lo tanto, sus aplicaciones pueden ser muy similares. La principal diferencia entre ellos es que los dispositivos de NVIDIA Jetson están diseñados específicamente para la implementación de sistemas de inteligencia artificial y aprendizaje máquina (mejor conocido como *Machine Learning*), ya que permiten un procesamiento paralelo acelerado por una unidad de procesamiento gráfico o GPU (del inglés *Graphics Processing Unit*) y velocidades más altas.

En general, las placas de NVIDIA Jetson son las más adecuadas para el desarrollo experimental de esta tesis ya que cuentan con una GPU de gran potencia, sin embargo, la Raspberry Pi 4 Modelo B, también es capaz de realizar este tipo de tareas, aunque con un rendimiento inferior.

En particular, la placa Jetson Nano es la más similar a la Raspberry Pi 4 Modelo B en términos de características técnicas. La principal diferencia entre ambas es la GPU. La Jetson Nano cuenta con una NVIDIA Maxwell, mientras que la Raspberry Pi 4 Modelo B cuenta con una GPU *Broadcom VideoCore VI*. La GPU de la primera es más potente, lo que le permite realizar tareas de procesamiento de imágenes y vídeo de forma más eficiente, sin embargo, la Raspberry Pi 4 Modelo B cuenta con un procesador más potente y otros periféricos integrados, como el Wi-Fi y

el Bluetooth; además la Raspberry Pi 4 Modelo B cuenta con placas de 2, 4 u 8 GB de memoria RAM, mientras que la Jetson solamente de 2 o 4 GB. Otro punto para considerar es que el costo de las placas de NVIDIA son significativamente más costosas, incluso la Raspberry Pi 4 Modelo B es más económica que la Jetson Nano (aproximadamente 75 y 99 dólares, respectivamente).

Teniendo en cuenta estos factores, se decidió utilizar la Raspberry Pi 4 Modelo B para el desarrollo de esta tesis, ya que ofrece un buen equilibrio entre rendimiento, precio y características.

1.2 Procesamiento de imágenes

Actualmente el procesamiento de imágenes ha tomado un importante protagonismo en muchas y diversas áreas como en la medicina, en las ciencias, en diversas industrias, en el arte, en lo recreativo y hasta en lo legal. Es importante mencionar que su evolución se ha dado gracias a los avances tecnológicos computacionales ya que requiere una gran capacidad de procesamiento y almacenamiento, por esa razón en los últimos años su potencial ha ido creciendo, siendo cada vez más eficiente y, por lo tanto, teniendo lugar en más áreas.

Desde los inicios del procesamiento de imágenes se han desarrollado diversas técnicas para ajustar ciertos aspectos de las imágenes, entre los más básicos se tienen el contraste y el brillo, usando técnicas como la modificación del histograma o el ajuste logarítmico para mejorar el contraste y la corrección gamma para el brillo. Entre otros elementos básicos que se ajustan en una imagen son el ruido, el enfoque, el tono, el tamaño y la orientación.

1.2.1 Aplicación del procesamiento de imágenes

En la actualidad el potencial del procesamiento de imágenes ha aumentado de manera considerable y, en gran medida, ha sido por el auge de los sistemas de inteligencia artificial y *machine learning*. Con este tipo de tecnologías se ha logrado escalar la aplicación del procesamiento de imágenes digitales al permitir no solo mejorar aspectos de la propia imagen, sino que ahora es posible segmentar regiones dentro de la imagen, detectar bordes de objetos, reconocer patrones y hasta hacer reconstrucciones en tres dimensiones. Entre otras aplicaciones que estas características permiten hacer, de las más comunes son el reconocimiento facial, el cual ha sido ampliamente estudiado casi desde los inicios del procesamiento de imágenes, pasando desde la detección de características individuales como lo propusieron Carey y Diamond en 1977 en [2], la utilización de la álgebra lineal en los inicios de la denominada inteligencia artificial a finales de la década de 1980 hasta la utilización de las redes neuronales artificiales más recientes; las restauraciones eficientes de imágenes como lo proponen Kai Zhang et al., en [3], donde crearon un eliminador de ruido utilizando una CNN; y la extracción de características como colores, texturas y formas, como lo hicieron Chandan Singh y Jaspreet Singh en el 2019 en [4].

Tanto estos avances como los que se han tenido en los sistemas computacionales se ha promovido el desarrollo de diversas aplicaciones del procesamiento de imágenes implementados sobre sistemas embebidos, como es el caso presentado por Hayato Hagiwara et al., en [5] donde implementaron sobre un FPGA un sistema de procesamiento de imagen en tiempo real para la visión de un robot, este sistema no solo detecta y rastrea objetos en movimiento, sino que también puede determinar su posición y orientación con respecto al entorno donde se encuentra. Un ejemplo reciente se tiene en [6] donde, en el año 2022, Yanchao Zhang y colaboradores crearon una red neuronal llamada RTSD-Net para la detección de fresas en tiempo real, basándose en la red YOLOv4 [7], implementándola sobre la tarjeta Jetson Nano. Otro ejemplo del mismo año se puede observar en el trabajo de Joelton Cezar Vieira y colaboradores en [8] donde implementaron sobre una Raspberry Pi 4 un sistema de monitoreo para identificar violencia en tiempo real usando modelos de CNN móviles.

Con la finalidad de poder darle una utilidad relevante al objetivo de esta tesis, se buscó que las imágenes a tratar no solo tuvieran un propósito artístico, por lo tanto, se optó por utilizar imágenes con un propósito clínico, es decir, imágenes médicas. Uno de los criterios que se definieron para seleccionar qué tipo de imágenes utilizar fue el poder dar solución a algún problema de salud grave, frecuente y actual.

La diabetes mellitus es una de las enfermedades más comunes en los mexicanos y, desde el 2016, está declarada como emergencia epidemiológica por la Secretaría de Salud. En el 2021 México ocupó el tercer lugar a nivel mundial en prevalencia de diabetes, con aproximadamente 14 millones de personas con esta enfermedad; en el mismo año fue el tercer motivo de consulta médica externa y de urgencias por enfermedad [9], al igual que fue la tercera causa de muerte durante el primer bimestre de ese año [10].

Existen diversas complicaciones causadas por esta enfermedad, sin embargo, la retinopatía diabética es una de las más graves, siendo la principal causa de ceguera en México [11]. La retinopatía diabética se puede detectar principalmente a través de diversos exámenes oftalmológicos como la fluoresceína angiografía, el examen de retina con pupila dilatada (midriática) y la tomografía de coherencia óptica. Para la mayoría de ellas es necesario dilatar la pupila para realizar los exámenes y, a pesar de que no se considera invasivo, la dilatación suele durar de 4 a 6 horas afectando la realización de actividades durante ese tiempo.

Otro método para poder detectar esta complicación es a través de la retinografía, la cual es una técnica que permite obtener fotografías de la retina, conocidas como imágenes de fondo de ojo. Estas imágenes permiten observar el tejido interno del ojo para poder analizarla detalladamente. Hay dos tipos de imágenes de fondo de ojo: midriática y no midriática. En la midriática es necesario dilatar la pupila para que se obtenga una mejor visibilidad de la retina, esto permite detectar varias patologías, entre ellas las más comunes son la retinopatía diabética, el glaucoma, la degeneración macular por la edad, el desprendimiento de retina y nevus coroideo; en el caso de la no midriática al no tener la pupila dilatada se utiliza principalmente para la detección de enfermedades de los vasos sanguíneos, sin embargo, aplicando un procesamiento a la imagen

es posible resaltar detalles que no pueden observarse a simple vista con la original, evitando tener que dilatar la pupila para poder detectar las otras enfermedades.

Con el fin de demostrar que las técnicas empleadas en este trabajo son adecuadas para el procesamiento de imágenes, se consideró utilizar imágenes de fondo de ojo no midriáticas debido a la peculiaridad de que éstas son tomadas en ambientes de baja iluminación y a que son más cómodas para los pacientes. Al tratar con este tipo de imágenes, en algún punto, será inevitable relacionarse con las ciencias de la salud. Teniendo en cuenta que en este trabajo no se pretende profundizar en dichos temas, con la utilización de las técnicas de *Soft Computing* en el procesamiento de estas imágenes es posible tomar como base el conocimiento y la experiencia de personal médico especializado poder modelar correctamente el sistema desarrollado. Por lo tanto, en este trabajo se solicitaron recomendaciones, comentarios y observaciones de dos optometristas para lograr mejorar efectivamente la calidad de las imágenes de fondo de ojo.

Procesamiento en imágenes de fondo de ojo

Desde la creación del oftalmoscopio en 1851 por Von Helmholtz se ha podido observar y explorar el fondo de ojo, sin embargo, las primeras imágenes publicadas eran bocetos dibujados. Con la invención de los dispositivos capaces de tomar fotografías de fondo de ojo fue posible documentar de manera más segura y económica anomalías en la retina [12].

Con las imágenes de fondo de ojo es posible detectar y evaluar diversas enfermedades oculares tal como el glaucoma, la degeneración macular, el desprendimiento de retina, el nevus coroideo y, la ya mencionada, retinopatía diabética. El procesamiento de este tipo de imágenes ayuda al diagnóstico, tratamiento y seguimiento de dichas afecciones ya que permiten, entre otras cosas, resaltar sus principales características y segmentar partes de la fotografía para analizar detalladamente ciertas regiones.

Las primeras investigaciones sobre procesamiento de imágenes de fondo de ojo comenzaron

en la década de 1970, un ejemplo se tiene en [13] donde B. H. McCormick y colaboradores desarrollaron un sistema de adquisición y procesamiento de imágenes de fondo de ojo capaz de realizar varias mediciones de utilidad clínica. Conforme se fueron desarrollando nuevas técnicas de procesamiento de imágenes se fueron aplicando a imágenes de fondo de ojo, por ejemplo, Eli Peli et al. en [14] utilizaron técnicas como filtrado homomórfico, mejoramiento adaptativo y modificación del histograma para mejorar la zona del nervio óptico de las imágenes de fondo de ojo.

Existen diversas investigaciones de la última década enfocadas en el procesamiento para la mejora de estas imágenes, como en el 2013 donde Agung W. Setiawan y colaboradores que mejoraron el contraste utilizando un método para modificar el histograma de la imagen, conocido como CLAHE (del inglés: Contrast Limited Adaptative Histogram Equialization) [15]; por otra parte, Mei Zhou et al. ajustaron tanto la luminosidad como el contraste usando la corrección gamma y CLAHE, respectivamente, para mejorar imágenes de baja calidad [16]; y de manera similar Gopal Datt Joshi et al. también mejoran la luminosidad y el contraste, sin embargo, ellos reasignan los colores basándose en el conocimiento que tienen sobre la geometría de la imagen [17]. De igual manera, se han realizado investigaciones para la detección de las enfermedades oculares mencionadas, principalmente sobre la retinopatía diabética y el glaucoma, por ejemplo, Almeida. et al. desarrollaron una metodología para la detección de glaucoma utilizando el software Matlab 14 [18]. Mark J. J. P. van Grinsven y colaboradores crearon y entrenaron una CNN para identificar hemorragias de manera eficiente [19]; por su parte, Ling-Ping Cen et al, también utilizaron redes neuronales profundas, sin embargo, la entrenaron para identificar 39 tipos de enfermedades [20], Rubya Afrin y Pintu Chandra utilizaron la lógica difusa para detectar y clasificar retinopatía diabética en imágenes con un previo procesamiento para eliminar el posible ruido de las imágenes [21].

1.3 Conclusiones del capítulo

En este capítulo se dio un panorama general del presente trabajo de investigación, en el cual se implementará sobre una Raspberry Pi un sistema de procesamiento de imágenes empleando la lógica difusa y las redes neuronales, con el fin de demostrar que dichas técnicas pueden ser utilizadas en sistemas embebidos actuales y son capaces de desempeñar sus tareas de manera eficiente. Gracias al avance tecnológico de los sistemas computacionales, el potencial del procesamiento de imágenes ha incrementado durante la última década, favoreciendo su incorporación en diversos ámbitos, teniendo en la medicina uno donde se utiliza frecuentemente desde los inicios del procesamiento digital de imágenes. Por ello se decidió procesar imágenes de fondo de ojo, pretendiendo mejorar su calidad y poder detectar alguna patología, con el objetivo de aportar al personal médico una herramienta de apoyo en el diagnóstico y seguimiento de algún padecimiento observable en este tipo de imágenes.

Capítulo 2
Marco Teórico

En el presente capítulo se proporcionará una revisión de los fundamentos y conceptos teóricos necesarios para el desarrollo de este trabajo, en el cual se utilizarán las técnicas de procesamiento de *Soft Computing* de lógica difusa y redes neuronales artificiales para procesar imágenes de fondo de ojo sobre un sistema embebido, creando así un sistema neuronal y difuso que, comúnmente, es nombrado neuro-difuso.

Como primer punto se revisarán las características principales que posee el sistema embebido (que en este trabajo lo constituye la placa Raspberry Pi 4) que hacen que sea adecuado para procesar imágenes y desarrollar en ella el sistema neuro-difuso propuesto; posteriormente se abordarán las técnicas de *Soft Computing* utilizadas en el desarrollo experimental; y, finalmente, se presentará un panorama general sobre las imágenes digitales y su procesamiento.

2.1 Raspberry Pi

La Raspberry Pi es un ordenador de placa única y por sus dimensiones (56 mm de ancho y 85 mm de largo) es considerado de tamaño reducido. Fue desarrollado por la Fundación Raspberry Pi con el objetivo de ser una herramienta educativa de bajo costo y accesible, especialmente para que niños y jóvenes pudieran aprender sobre computación sin necesidad de utilizar un aprendizaje estructurado. El primer modelo se lanzó a la venta formalmente a inicios del año 2012 con una segunda revisión a finales del mismo año [22].

Debido a su reducido tamaño, capacidad para ejecutar sistemas operativos completos y permitir la programación en lenguajes como Python, C y Scratch, las placas Raspberry Pi se comenzaron a popularizar tanto por estudiantes como por profesionales y aficionados a la tecnología, ya que estas características lo hicieron ideal para ejecutar diversos tipos de proyectos para múltiples aplicaciones, donde no necesariamente se utilizaban como un computador, tal como sistemas de automatización, servidores web y dispositivos de internet de las cosas, por mencionar algunos.

Todas las versiones y modelos de la Raspberry Pi poseen procesadores ARM (del inglés *Advanced RISC Machines*), por lo tanto, se basan en una arquitectura con un conjunto reducido de instrucciones conocida como RISC (por su acrónimo en inglés *Reduced Instruction Set Computing*). Tener un conjunto reducido de instrucciones permite que estos procesadores sean más eficientes energéticamente y sencillos de diseñar y fabricar, además, facilita la decodificación y ejecución de instrucciones permitiendo que sean más veloces a comparación de otras arquitecturas.

2.1.1 Raspberry Pi 4 Modelo B

Actualmente, existen varias versiones de estas placas, siendo la Raspberry Pi 4 modelo B la versión más reciente al momento de realizar la parte experimental de esta tesis. Su procesador se encuentra en el chip Broadcom BCM2711 cuyas características y especificaciones mecánicas y eléctricas se detallan en su hoja de especificaciones que puede obtenerse en [23]. Una versión anterior es la Raspberry Pi 3 modelo B+ y su procesador se encuentra en el chip Broadcom BCM2837B0. Ambos chips son considerados "sistemas en un chip" o SoC (del inglés *System on a Chip*), ya que componentes como el procesador (CPU), el procesador gráfico (GPU), la memoria RAM y algunos controladores como los de entrada/salida, de pantalla y de conectividad se integran dentro de un solo chip. Las diferencias entre ambos se muestran en la tabla 2.1.

En la tabla 2.1, se puede observar que el chip BCM2711 utilizado en la Raspberry Pi 4 modelo B es prácticamente una actualización del modelo anterior ya que utiliza versiones más

Tabla 2.1: Comparación entre chips de Raspberry Pi 3 B+ y Raspberry Pi 4 B.

Modelo	BCM2837B0	BCM2711
CPU	4 núcleos tipo Cortex-A53 (ARMv8) 64-bits 1.4GHz	4 núcleos tipo Cortex-A72 (ARMv8) 64-bits 1.5GHz
GPU	VideoCore IV 32 bits 400MHz 1080p a 60fps OpenGL ES 2.0	VideoCore VI 3D 32 bits 500MHz 4k a 60fps OpenGL Es 3.0
Memoria	1 GB LPDDR2 SDRAM 32KB caché nivel 1 512KB caché nivel 2	Hasta 8GB LPDDR4-240 SDRAM 32KB de datos + 48KB de instrucciones caché nivel 1 1MB caché nivel 2
E/S	USB 2.0 Gigabit Ethernet sobre USB 2.0 Bluetooth 4.2 con BLE Wi-Fi IEEE 802.11.b/g/n/ac	USB 3.0 y 2.0 Gigabit Ethernet Bluetooth 5.0 con BLE Wi-Fi IEEE 802.11.b/g/n/ac Bus PCIe

recientes de los mismos componentes. De acuerdo con la información proporcionada en su página oficial [23], su diseño es similar al del BCM2837B0, sin embargo, los cambios que lo hacen considerablemente mejor es la utilización de un núcleo ARM más poderoso como lo es el Cortex-A72, que es capaz de trabajar a 1.5 GHz haciendo que sea aproximadamente 50 % más rápido que la Raspberry Pi 3B+; el poder acceder a más direcciones de memoria y el tener una GPU mucho más rápida, que es gracias a la incorporación del bus PCIe.

Como se mencionó anteriormente, el procesamiento de imágenes requiere de una gran capacidad de procesamiento, por lo tanto, entre estas dos placas se eligió la Raspberry Pi 4 Modelo B de 8 GB de RAM, por tener un procesador más potente y poder acceder a una mayor cantidad de RAM de una manera más veloz. En la figura 2.1 se muestra una fotografía de la Raspberry Pi 4 Modelo B.

La memoria RAM es un componente esencial, ya que si llega a ser insuficiente la capacidad de

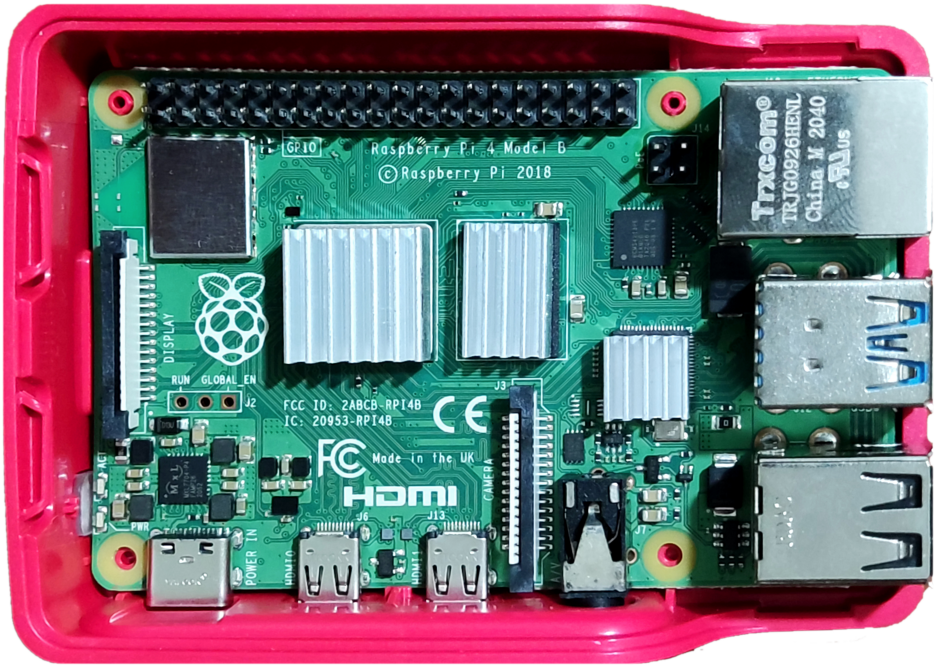


Figura 2.1: Vista superior de la Raspberry Pi 4 modelo B de 8 GB®.

procesamiento se podría ver afectada. Por esta razón se eligió el modelo con la mayor capacidad de memoria disponible.

Aparte de los módulos de conectividad de la Raspberry Pi 4 Modelo B mostrados en la tabla 2.1, esta placa tiene dos puertos micro HDMI que permiten conectar dos pantallas 4K simultáneamente, un puerto con la interfaz serial para pantalla conocida como DSI (del inglés *Display Serial Interface*), un puerto de cámara con la interfaz serie para cámaras o CSI (del inglés *Camera Serial Interface*) y 28 pines de entrada/salida de propósito general o GPIO (del inglés *General Purpose Input/Output*), los cuales utilizan 3.3 Volts para un nivel lógico alto y 0 Volts para un nivel lógico bajo. Estos pines GPIO pueden configurarse por software como pines de entrada o salida, sin embargo, algunos pines específicos tienen funciones alternativas, ya sea por tener controladores por hardware como moduladores de ancho de pulsos (mejor conocidos como PWM por su acrónimo en inglés de *Pulse Width Modulation*) o por estar conectados a los buses disponibles con los que cuenta esta placa, los cuales son los buses SPI (del acrónimo en inglés de interfaz periférica serial: *Serial Peripheral Interface*), el bus de datos serie de dos hilos o I2C (por su acrónimo en inglés de *inter integrated circuits*) y conocido UART (receptor/transmisor

asíncrono universal). En la figura 2.2 se muestra un esquema de la Raspberry Pi 4 con sus puertos disponibles.

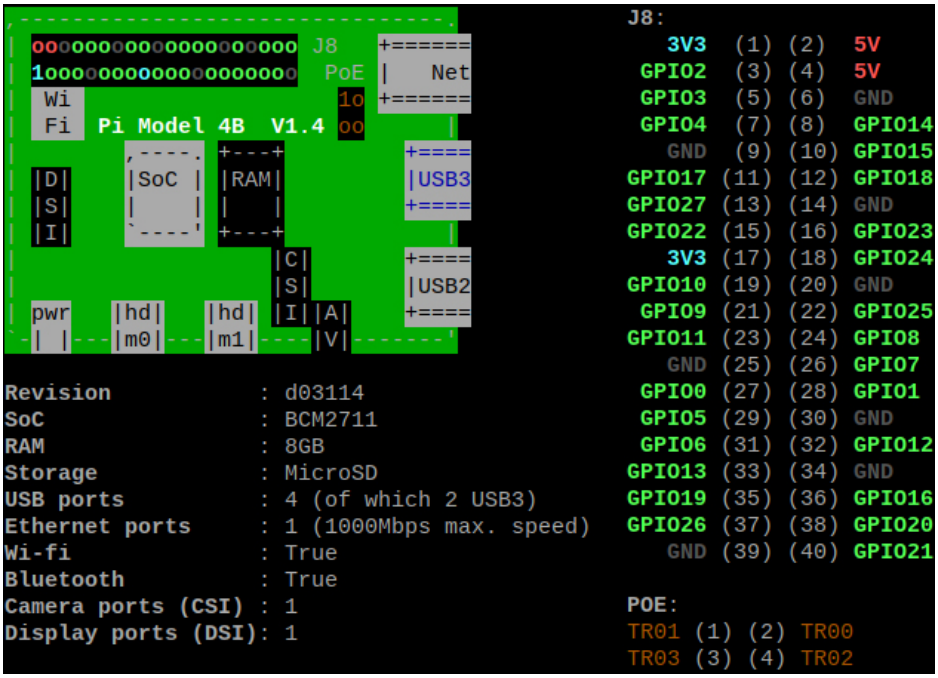


Figura 2.2: Esquema y asignación de pines de la Raspberry Pi 4 modelo B®.

Por su arquitectura, la Raspberry Pi es capaz de ejecutar sistemas operativos completos, de hecho, es necesario instalar uno para trabajar con ella, ya sea como computador o para ejecutar algún proyecto. El sistema operativo oficial "Raspberry Pi OS" está basado en una distribución de Linux llamada Debian y está optimizado para utilizarse en estos dispositivos con herramientas preinstaladas y con una documentación de la placa completa.

Los sistemas operativos basados en el kernel de Linux son conocidos por su estabilidad, seguridad, flexibilidad y personalización, además de que son ligeros, de código abierto y, por lo tanto, gratuitos. Aparte de Raspberry Pi OS, existen otros SO desarrollados específicamente para estas placas y para otros sistemas embebidos similares, la mayoría de ellos están basados en distribuciones de Linux, por lo antes mencionado, tal como Pidora que está basado en Fedora y Arch Linux ARM; aunque también existen versiones de Windows diseñados para sistemas embebidos, conocidos como Windows IoT. Muchos de estos SO incluyen Python en su

instalación base, como es el caso de Raspberry Pi OS ya que la Raspberry Pi requiere de este lenguaje de programación para configurar sus módulos de hardware disponibles.

Python es un lenguaje de programación interpretado de alto nivel que tiene una gran variedad de bibliotecas, entre ellas existen varias que son específicas para Linux, las cuales permiten crear aplicaciones y herramientas para estos sistemas operativos. Con ellas se crearon las bibliotecas para poder configurar los periféricos y módulos de hardware de la Raspberry.

Gracias a las características antes mencionadas de la Raspberry Pi 4, éste presenta una gran capacidad de procesamiento y un alto rendimiento, lo que permite procesar algoritmos complejos. Esto, aunado a que cuenta con un un sistema operativo y utiliza Python de manera nativa, permite utilizar las bibliotecas disponibles para procesamiento de imagen y de *Machine Learning* de Python, con lo cual es capaz de ejecutar modelos de *machine learning* [24].

2.2 Soft Computing

El término "*Soft Computing*" fue acuñado por Lofti A. Zadeh en 1994 [25] como una manera de crear computacionalmente sistemas inteligentes que puedan asemejar el razonamiento y aprendizaje de los humanos, los cuales tienen la capacidad de hacerlo con datos imprecisos y una gran incertidumbre. *Soft Computing* no es una metodología, sino un conjunto de técnicas que utilizan métodos heurísticos y permiten manejar efectivamente dicha incertidumbre y ambigüedad para crear sistemas flexibles y adaptables. Entre estas técnicas sobresalen tres: la lógica difusa que permite trabajar con "palabras" en lugar de matemáticas complejas; las redes neuronales que permite crear sistemas de identificación, aprendizaje y adaptación, y los algoritmos metaheurísticos que permiten optimizar sistemas.

Estas técnicas son complementarias, es decir, se pueden utilizar a la vez para aprovechar las ventajas que ofrece cada una en un solo sistema. La unión más común se tiene con la lógica

difusa y las redes neuronales artificiales, creando así los sistemas neuro-difusos.

Como se mencionó anteriormente, en este trabajo se desarrollará un sistema neuro-difuso para procesar imágenes. La lógica difusa se utilizará para hacer un preprocesamiento con el objetivo de mejorar la calidad de las imágenes para resaltar sus características y las redes neuronales artificiales para clasificar las imágenes.

Es importante mencionar que, si se observan ambos procesos de manera individual, al preprocesamiento se puede considerar como *Soft Computing* por utilizar la lógica difusa con datos ambiguos y al uso de la red neuronal se puede considerar como *Machine Learning*, ya que se entrena una red para aprender a clasificar imágenes.

2.3 Lógica difusa

La lógica difusa es una extensión de la lógica clásica, por ello, para entenderla es conveniente comprender su evolución. La lógica, como disciplina filosófica, estudia los principios del razonamiento válido y analiza los fundamentos del conocimiento y la verdad, en otras palabras, trata del estudio de cómo razonamos y cómo llegamos a conclusiones validas.

La lógica clásica está basada en la ley del medio excluido de Aristóteles, o sea que solamente trata con "verdadero" y "falso", es decir, una proposición puede tener solamente uno de estos valores. En cambio, la lógica difusa no se limita a estos dos estados, sino que toma en cuenta proposiciones que no son completamente verdaderas o falsas. Dicho de otra manera, entre esos dos estados puede haber muchas posibilidades como "parcialmente falso", "más o menos verdadero", "casi verdadero", etc.

El concepto actual de lógica difusa fue propuesto por Zadeh en 1965 con su artículo "*Fuzzy Sets*" [26], el cual está basado en otros puntos de vista de la lógica, donde filósofos opinaban que algunas proposiciones podrían ser verdaderas y falsas a la vez, o sea que entre lo verdadero y

lo falso existía una región donde podrían tener cierto grado de verdad o falsedad [27].

En la vida cotidiana utilizamos un lenguaje con mucha imprecisión, por ejemplo, cuando una persona quiere decir que el clima es frío no suele dar la temperatura exacta, sino que utiliza frases como "hace frío" o "está helado", lo que refiere a un rango de temperatura basado en su propia percepción o experiencia, por ello, lo que él puede considerar como clima frío para otra persona podría ser templado o hasta caluroso. La lógica difusa considera esta vaguedad del lenguaje natural con la cual, junto con los principios de la lógica predictiva y proposicional, que permiten representar la imprecisión del lenguaje natural, es posible construir sistemas de inferencia difusos, que son capaces de mapear la salida del sistema a partir de parámetros de entrada aplicando los principios de la lógica difusa.

2.3.1 Funciones de membresía

Así como existe una teoría de los conjuntos clásicos existe la teoría de conjuntos difusos, los cuales son una generalización de los primeros. En la teoría clásica, el universo de discurso se refiere a los elementos de un conjunto y es representado como $U = \{x_1, x_2, x_3, \ldots, x_n\}$, con la cual es posible saber, sin duda, que el elemento x_1 pertenece al conjunto U. En esta teoría se indica que la unión, intersección y el complemento son las tres operaciones básicas aplicables a estos conjuntos y con ellas se pueden crear otras como la diferencia y la diferencia simétrica.

Al igual que en la clásica, en la teoría de conjuntos difusa se puede manejar la pertenencia y la no pertenencia, pero también se toma en cuenta el grado de pertenencia de los elementos en un conjunto, por ello se considera como una generalización de la clásica. En esta teoría, la membresía de un elemento x en un conjunto S no está definida solamente como 0 o 1 ("pertenece" o "no pertenece"), sino que está dado por una función de membresía μ.

Formalmente, un conjunto difuso F se puede definir por medio de la ecuación (2.1).

$$F = \{(x, \mu_F(x)) | x \in U, 0.0 <= \mu_F <= 1.0\} \tag{2.1}$$

Donde x es un elemento perteneciente al universo de discurso U y $\mu_F(x)$ es la función de membresía del elemento x sobre el conjunto F.

Como se puede observar en la ecuación (2.1), los conjuntos difusos se definen por los pares ordenados de elemento-valor de membresía $(x, \mu_F(x))$, por lo tanto, la representación de los conjuntos se puede realizar por medio de funciones de pertenencia, las cuales dependerán de la naturaleza de la aplicación donde se empleará.

Entre las funciones de membresía más básicas se encuentran las funciones triangular y trapezoidal, las funciones S, Z y la Gaussiana [27]. Éstas últimas tres fueron las utilizadas en la primera parte del desarrollo experimental de esta tesis.

La función S (también conocida como función sigmoide), descrita por la ecuación (2.2), se utiliza comúnmente para representar conjuntos donde el grado de pertenencia inicia en su valor mínimo y hace una transición suave hasta su valor máximo.

$$\mu_S(x) = \frac{1}{1 + e^{-\alpha(x-\beta)}} \tag{2.2}$$

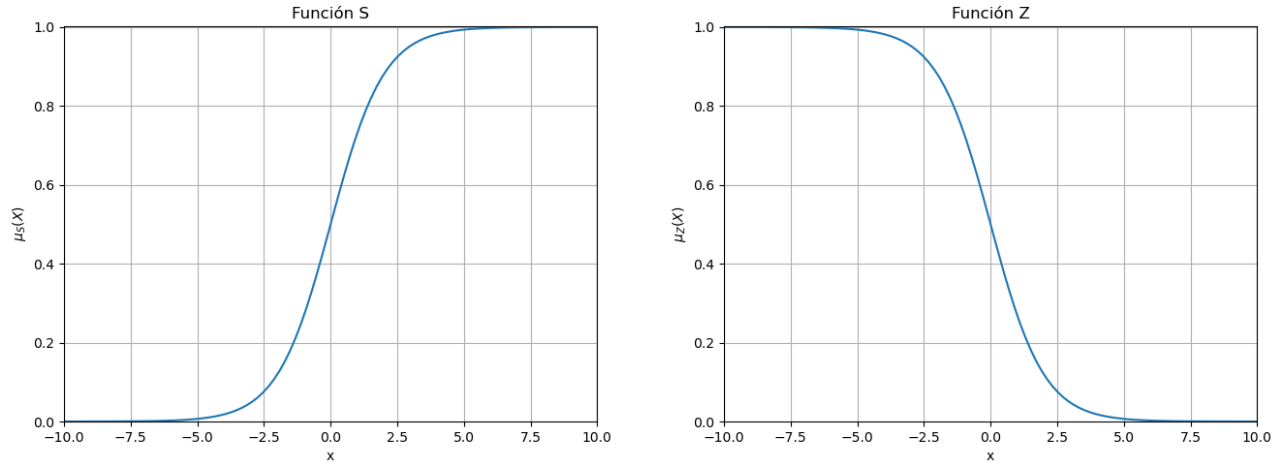
La función Z, descrita por la ecuación (2.3), puede verse como una sigmoide inversa, o sea que comienza en uno y termina en cero con una transición suave.

$$\mu_Z(x) = \frac{1}{1 + e^{\alpha(x-\beta)}} \tag{2.3}$$

En las ecuaciones (2.2) y (2.3), α representa la pendiente, o sea qué tan rápido crece o decrece la función, y β el punto medio de la pendiente. En la figura 2.3 se muestran las gráficas de estas funciones, donde la 2.3a es la gráfica de la función S y la 2.3b la de la función Z donde $\beta = 0$, es decir, están centradas en el origen.

La función gaussiana, representada por la ecuación (2.4), es un caso particular de la función campana. Ambas se utilizan para representar conjuntos con una forma de distribución normal de probabilidad, es decir, la pertenencia máxima se encuentra en el centro de la función y decrece suavemente a medida que se aleja del centro.

$$\mu_G(x) = e^{-\frac{1}{2}(\frac{x-\gamma}{\sigma})^2} \tag{2.4}$$



(a) Gráfica de la función S.

(b) Gráfica de la función Z.

Figura 2.3: Gráficas de la función S y Z.

De la ecuación 2.4, γ representa el centro de la función y σ la desviación estándar que, en el contexto de función de membresía, es el parámetro que controla el ancho de la campana. Su gráfica se muestra en la figura 2.4.

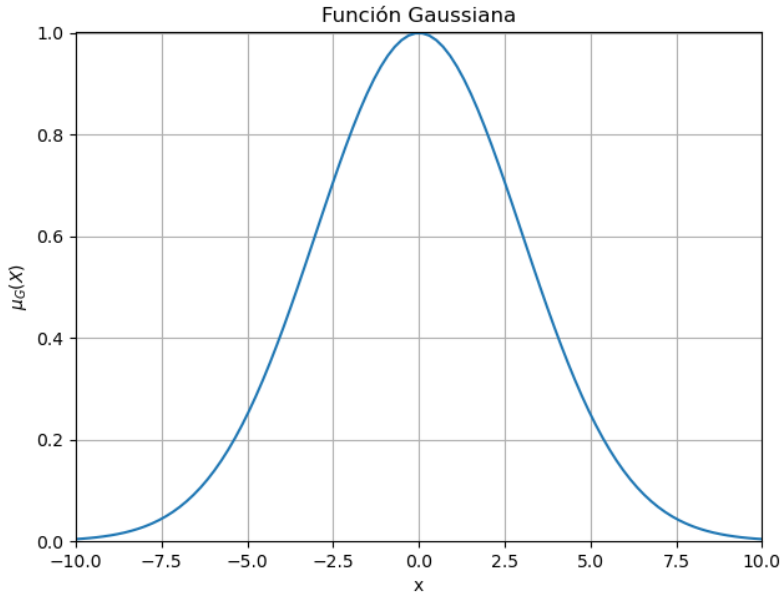


Figura 2.4: Gráfica de la función gaussiana.

Entre las ventajas de utilizar las funciones básicas se tiene que son funciones simples y, por lo tanto, fáciles de entender e implementar. Además, son capaces de adaptarse a una amplia gama de aplicaciones y, por su sencillez, son computacionalmente eficientes [28].

2.3.2 Variables lingüísticas

Una de las ventajas de utilizar conjuntos difusos es que con ellos es posible utilizar el lenguaje natural con el que nos comunicamos comúnmente, el cual suele ser vago o impreciso. Estos conjuntos son capaces de manejar la ambigüedad y la imprecisión ya que permiten modelar la vaguedad de nuestro lenguaje en formulaciones matemáticas. Por ejemplo, modelar la temperatura del ambiente se podría realizar de la siguiente manera:

Como conjunto difuso se tendría la temperatura ambiente y se podría representar con 3 variables: "frío", "templado" y "caliente". A estas variables se les conoce como variables lingüísticas ya que son palabras o frases que representan categorías cualitativas, en este caso representan diferentes rangos de temperatura que no son precisos. Cada una de estas variables lingüísticas tiene asociada una función de membresía que describe los valores que se ajustan a esa categoría, por ejemplo, para "frío" se podría utilizar una función Z donde el punto central de la curva sea a los 10° para que la pertenencia disminuya mientras suba la temperatura, para "templado" se podría utilizar una trapezoidal o una gaussiana que comience alrededor de los 10° y disminuya alrededor de los 25°, finalmente para "caliente" se podría utilizar una función S donde el punto central de la curva esté a los 25°.

Aunque la teoría de los conjuntos difusos es extensa y no se abordará en profundidad en este trabajo, es relevante mencionar que la mayoría de las operaciones básicas de los conjuntos clásicos también son aplicables a los conjuntos difusos.

2.3.3 Sistema de inferencia difusos

La lógica difusa se utiliza comúnmente para crear sistemas de inferencia difuso, mejor conocidos como FIS por sus siglas en ingles de *Fuzzy Inference System*. Éstos son sistemas de control capaces de modelar la experiencia humana para mapear la salida del sistema utilizando

las bases de la lógica difusa como las funciones de membresía, los conjuntos difusos y las variables lingüísticas.

Las bases de la lógica difusa antes mencionadas son algunas de las herramientas que se utilizan en los FIS, sin embargo, el núcleo de estos sistemas son las reglas difusas —que también son base de la lógica difusa— ya que son las que permiten realizar el razonamiento para deducir una conclusión a partir de las proposiciones iniciales, dicho de otra manera, son las que permiten realizar la inferencia. La estructura de las reglas tiene la forma *SI-ENTONCES* o *IF-THEN* en inglés, tal como se muestra en (2.5).

$$\textbf{\textit{SI}} \quad [antecedente] \quad \textbf{\textit{ENTONCES}} \quad [consecuente] \tag{2.5}$$

Estas reglas se sustentan con la lógica propositiva y la lógica predictiva, donde la primera permite representar proposiciones lingüísticas vagas como "clima frío" o "tiempo helado" para asociarlas a conjuntos difusos y la segunda se utiliza para establecer relaciones entre variables difusas.

Muchas de las reglas difusas se basan en las reglas de inferencia clásicas como el *modus ponens* y *modus tollems*, sin embargo, los FIS también pueden utilizar reglas difusas más complejas. Por ejemplo, una regla difusa podría tener dos o más antecedentes o consecuentes y también podrían utilizarse operadores lógicos como *OR*, *AND* y *NOT* para combinar reglas difusas o relacionar varias variables difusas.

Estas operaciones lógicas se basan en las operaciones básicas, pero se aplican de manera diferente para tener en cuenta la naturaleza imprecisa de los conjuntos difusos. En síntesis, estas operaciones se describen de la siguiente manera:

- AND. Es la intersección entre conjuntos, donde se consideran las regiones de superposición entre ellos.
- OR. Es la unión de los conjuntos, que representa el conjunto de elementos que pertenecen a uno o a ambos conjuntos.

- NOT. Es el complemento de un conjunto, es decir, los elementos que no pertenecen al conjunto.

En general, un FIS tiene como componentes principales entradas, reglas y salidas, con los cuales es capaz de "tomar decisiones" con datos imprecisos o ambiguos, donde el resultado o la salida no necesariamente deba estar en términos de "si" o "no", sino que puede tener grados de membresía. Las entradas, como su nombre lo indica, son las variables que representan los datos del mundo real que ingresan al sistema, las salidas son las respuestas del sistema y las reglas son las declaraciones que relacionan las entradas con las salidas. De esta manera el funcionamiento de un FIS se puede esquematizar de la siguiente manera:

1. Fuzzificación de las entradas.
2. Aplicación de las reglas difusas.
3. Defuzzificación de la salida difusa para obtener una salida concreta.

La palabra "fuzzificación" es una traducción de inglés "*fuzzify*" que, en este contexto, se utiliza para describir la transformación de las variables reales de entrada en conjuntos difusos con sus funciones de membresía. El proceso de fuzzificación en los FIS implica asignar grados de membresía a los valores de entrada en función de las funciones de membresía asociadas.

Para la aplicación de las reglas difusas se utilizan los operadores difusos "*AND*" y "*OR*" en los antecedentes de las reglas para evaluar cómo se superponen las funciones de membresía de las variables de entrada y para qué parte de los conjuntos difusos se activan dichas reglas, obteniendo así un conjunto difuso de salida, el cual al ser difuso representa la incertidumbre asociada con la respuesta del sistema.

La "defuzzificación" hace referencia al proceso inverso de la "fuzzificación", o sea, pasar del plano difuso al real. En el contexto de los FIS, es el proceso de convertir el conjunto difuso de salida en un valor numérico. Para esto existen diversos métodos, entre los más comunes

se encuentran los métodos de centroide, centro de sumas, media del máximo y el promedio ponderado o WAM (por las siglas en inglés de *Weighted Average Method*) [29], donde éste último fue el utilizado en la primera parte experimental de esta tesis.

En el método WAM cada conjunto difuso de salida tiene asociado un peso y el valor final se calcula como la suma de los productos de los valores del conjunto difuso y sus respectivos pesos, dividida por la suma total de los pesos, como se observa en la ecuación (2.6).

$$y = \frac{\sum_{i=1}^{n} \mu_i \cdot w_i}{\sum_{i=1}^{n} \mu_i} \tag{2.6}$$

Donde μ_i es la función de pertenencia del conjunto difuso y w_i su peso asociado.

Actualmente se han desarrollado varios tipos de FIS, sin embargo, los dos principales son los de tipo Mamdani y Sugeno. La principal diferencia entre ellos radica en la forma en que definen sus reglas y, por lo tanto, cómo se realiza la defuzzificación.

En los FIS de tipo Mamdani las reglas tienen la estructura que se muestra en (2.7).

$$\textbf{\textit{SI}} \quad X\, es\, A \quad \textbf{\textit{ENTONCES}} \quad Z\, es\, C \tag{2.7}$$

Donde las expresiones $X\, es\, A$ y $Z\, es\, C$ representan el antecedente y el consecuente, respectivamente. Ambos son conjuntos difusos definidos por funciones de membresía que asignan grados de pertenencia a los elementos. En este tipo de FIS tanto el antecedente como el consecuente son conjuntos difusos, por lo tanto, para que la salida pueda tener un valor real se debe realizar el proceso de defuzzificación.

En los FIS de tipo Sugeno, el consecuente de las reglas no es un conjunto difuso, sino una función matemática que relaciona las entradas con la salida, como se muestra en la ecuación (2.8).

$$\textbf{\textit{SI}} \quad X\, es\, A \quad \textbf{\textit{Y}} \quad Y\, es\, B \quad \textbf{\textit{ENTONCES}} \quad Z = f(X, Y) \tag{2.8}$$

De esta manera, en este tipo de FIS no es necesario realizar un proceso de defuzzificación, ya que la salida de la función Z ya es un valor real.

2.4 Redes neuronales artificiales

Las redes neuronales artificiales o ANN por sus siglas en inglés de *Artificial Neural Networks*, son modelos computacionales que imitan el comportamiento del cerebro humano. El concepto de "red neuronal artificial" se remonta a la década de 1940, cuando los neurocientíficos Warren McCulloch y Walter Pitts propusieron un modelo matemático de la neurona en su artículo "*A logical calculus of the ideas immanent in nervous activity*" [30]. Posteriormente, en la década de 1950, Frank Rosenblatt y otros investigadores desarrollaron el "perceptrón" [31], un tipo de red neuronal que podía ser entrenada por medio de reglas de entrenamiento para realizar tareas como reconocimiento de patrones. Para ello demostraron matemáticamente que el entrenamiento siempre convergería a una solución si se entrenaban con conjuntos de entrenamiento linealmente separables.

Aunque en su momento el desarrollo del perceptrón causó mucho entusiasmo porque se veía como un modelo teórico bien fundamentado de aprendizaje automático, se encontraron varias limitaciones, por ejemplo, que solo podían aprender funciones linealmente separables y que era difícil entrenarlas para tareas complejas, lo que generó que no tuvieran aplicaciones significativas. Por estas limitaciones, que eran tanto teóricas como tecnológicas, el desarrollo de ANN se estancó.

En 1985 David Rumelhart, Geoffrey Hinton y Ronald Williams presentaron en [32] una regla para el entrenamiento del perceptrón en redes multicapa haciendo uso de un algoritmo conocido como *backpropagation* o propagación hacia atrás por su traducción literal al español. Este nuevo aprendizaje para el perceptrón en redes multicapa resolvió su limitación al permitirles aprender tanto funciones lineales como no lineales. Esto revolucionó el campo de reconocimiento de patrones por lo que, a partir de ese momento, el desarrollo de las ANN se reinició, creando nuevos algoritmos de entrenamiento y otras arquitecturas de red.

Gracias a la constante mejora tecnológica en las computadoras y la mayor disponibilidad

de datos, a partir de los años 2000 el desarrollo de las ANN se ha acelerado significativamente logrando crear arquitecturas de red más grandes, complejas y profundas —cuyo significado se presentará más adelante—. Con ello se amplió la gama de aplicaciones al ser capaces de hacer reconocimiento de imágenes y de voz, traducción automática, detección de patrones complejos, entre otras aplicaciones.

Una parte importante en la historia de las ANN se tuvo al poder utilizar las unidades de procesamiento gráfico (GPU), ya que, gracias a su optimización para realizar operaciones paralelas y cálculos de matrices y punto flotante, se pudieron desarrollar redes neuronales convolucionales, mejor conocidas como CNN por sus siglas en inglés de *Convolutional Neural Networks* y redes neuronales recurrentes o RNN (del inglés *Recurrent Neural Networks*). Las CNN son sumamente eficientes en el procesamiento y reconocimiento de imágenes y las RNN son utilizadas, entre otras cosas, en el procesamiento de lenguaje natural.

2.4.1 Inspiración biológica

El cerebro humano es un órgano complejo que nos permite percibir y comprender el mundo que nos rodea. Una de las formas en que el cerebro humano realiza estas tareas es identificando y reconociendo patrones, por ejemplo, cuando vemos una cara nuestro cerebro identifica los patrones de ojos, nariz, boca y otros rasgos faciales, permitiéndonos reconocer a las personas, incluso si están de perfil, en diferentes ángulos, con diferente iluminación, etc.

En este contexto, entre las tareas principales que realiza el cerebro son el reconocimiento, clasificación, asociación y agrupación. Estas tareas son fundamentales para el funcionamiento del cerebro ya que nos permiten entender el mundo que nos rodea, tomar decisiones y aprender.

La manera en que el cerebro logra esto es gracias que está conformado por un gran número (aproximadamente 100 billones) de células conocidas como neuronas, donde cada una está conectada con, aproximadamente, 10,000 de neuronas vecinas[33]. Estas células tienen la

capacidad de recolectar y "resumir" las señales de estímulo que reciben de neuronas vecinas y pasarlas a las siguientes neuronas vecinas.

Es posible decir que las partes principales que componen a una neurona son el soma, las dendritas y el axón. El soma o cuerpo celular es la parte central de la neurona y tiene función de sumar las señales entrantes y decidir si envía o no una respuesta, dependiendo de si reciben suficientes estímulos. Las dendritas son ramificaciones de fibras nerviosas receptivas que salen del cuerpo celular y son las responsables de recibir las señales de las otras neuronas para llevarlas hacia el cuerpo celular. El axón es una única fibra larga que sale del cuerpo celular y es el encargado de transmitir la señal de estímulo de la neurona hacia las neuronas vecinas a través de sus terminales.

El contacto entre el extremo del axón de una neurona y la dendrita de otra se conoce como sinapsis, la cual está formada por estructuras especializadas en las membranas celulares de las dos neuronas involucradas. En la figura 2.5 se muestra una representación de la conexión entre dos neuronas.

Parte de la estructura neuronal se define desde el nacimiento y otras partes se va desarrollando a través del aprendizaje, donde se van creando nuevas uniones sinápticas y otras se van eliminando, algunas otras se fortalecen y unas más debilitan.

La suma de toda la actividad neuronal, es decir, la continua activación de las neuronas de forma simultánea es lo que nos permite "procesar" información de manera muy eficiente y rápida para realizar actividades como reconocer patrones, asociar, clasificar, aprender, solucionar problemas y controlar nuestro cuerpo; tareas que serían muy complicadas e inclusive imposibles de realizar por una computadora utilizando la computación convencional.

En la computación básica se procesa información con base en una secuencia de ciclos de búsqueda y ejecución donde, en cada ciclo, la computadora busca las instrucciones a realizar en una memoria y luego las ejecuta. Los datos también se almacenan en la memoria y la computadora accede a ellos cuando los necesita.

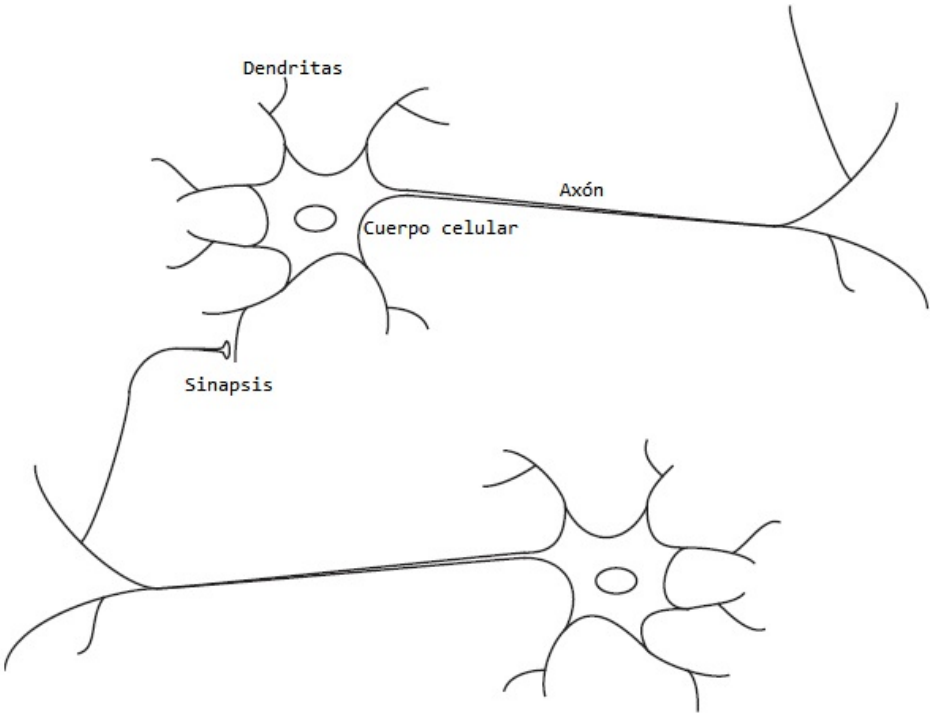


Figura 2.5: Representación gráfica de una neurona con sus principales componentes (tomada de [33]).

Esto ha demostrado ser muy eficiente para realizar tareas que se pueden dividir en pasos secuenciales, lo que ha permitido realizar una enorme cantidad de aplicaciones que han ido desde procesar texto hasta realizar cálculos matemáticos complejos, analizar datos, crear simulaciones o controlar máquinas complejas. Sin embargo, el procesamiento que realizan las computadoras es el resultado del conocimiento del programador, que lo plasma sobre ellas utilizando un lenguaje de programación.

De esta manera se puede hacer una similitud entre el cerebro humano y las computadoras, ya que ambos son capaces de realizar tareas complejas y utilizan señales eléctricas para transmitir información. En este contexto, es posible afirmar que la diferencia entre el cerebro y una computadora, aparte de que el primero es un "sistema" paralelo y las computadoras son sistemas secuenciales, es que el cerebro es adaptable, es decir, es capaz de cambiar para adaptarse a nuevas situaciones aprendiendo de la experiencia, mientras que las computadoras convencionales tendrían que ser reprogramadas para realizar nuevas tareas.

Esto motivó el desarrollo de un tipo de computación inspirada en la estructura y funcionalidad del cerebro, con lo que se desarrollaron elementos computacionales no lineales conocidas como neuronas artificiales [34], que son simples abstracciones de las neuronas biológicas en el sentido que imitan su comportamiento. Simulando la estructura neuronal del cerebro, se han desarrollado redes de estas neuronas artificiales, la cuales son conocidas como redes neuronales artificiales o ANN. Estas redes no se pueden comparar con la potencia y capacidad del cerebro, sin embargo, pueden entrenarse para realizar tareas como reconocimiento de patrones, regresión o clasificación, entre otras, de manera muy eficiente.

2.4.2 Neurona artificial

La estructura de una neurona artificial (ver figura 2.6) está compuesta por sus entradas, un sesgo b (del inglés: *bias*), una unidad de procesamiento y una salida.

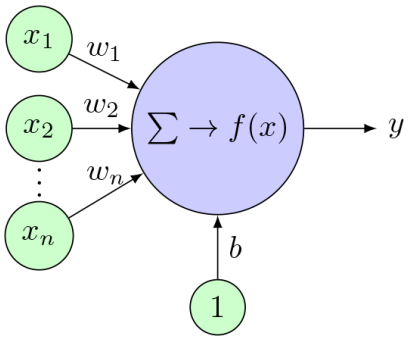


Figura 2.6: Representación de una neurona artificial con n entradas.

Como se observa en la figura 2.6, a la entrada de una neurona ingresan los datos $x_1, x_2, ..., x_n$, y cada uno es multiplicado por un valor escalar conocido como peso $w_1, w_2, ..., w_n$, de tal manera que ingresan a la unidad de procesamiento de la forma $x_n w_n$. La unidad de procesamiento consta de un sumador y una función de transferencia o función de activación, el primero se encarga de sumar todas las entradas junto con el sesgo de la forma $n = x_1 w_1 + x_2 w_2 + ..., x_n w_n + b$, obteniendo una suma ponderada. Finalmente la salida del sumador —también conocida como entrada de la red— pasa a la función de activación para generar la salida de la neurona , descrita

por la ecuación (2.9).

$$y = f(\sum_{i=1}^{n}(x_i w_i + b) \tag{2.9}$$

Anteriormente se mencionó que una neurona artificial es un modelo computacional simplificado basado en la neurona biológica, por lo tanto, haciendo una analogía los pesos representan la fortaleza de la sinapsis, el conjunto de cada entrada multiplicado con el peso imita el funcionamiento de la dendrita, la suma ponderada de entradas, incluyendo el sesgo, simula la integración de señales en el soma y el paso de la suma por la función de activación emula la función total del soma, finalmente la salida de la neurona representa la señal que sale de la neurona al axón.

Pesos

Debido a que los pesos están asociados a las entradas de la neurona, éstos determinarán de qué manera las entradas afectarán la salida. Cuando el peso es positivo significa que la entrada tendrá un efecto de excitación, en caso contrario la neurona tendrá un efecto de inhibición.

Sesgo

Como se observa en la ecuación (2.9), el sesgo b se representa como un peso adicional que se añade a la suma de las entradas, con la diferencia de que tiene la constante uno en la entrada. El efecto de este parámetro es significativo en el comportamiento de una neurona, ya que puede utilizarse para controlar la tendencia de la neurona para activarse o no, por ejemplo, si su valor es positivo entonces la neurona generará una salida alta incluso si la suma de las entradas es cero, si es negativo la neurona necesitará una suma ponderada más grande para activarse, haciendo que la neurona sea más selectiva y solo se active cuando las entradas son significativas. El sesgo también introduce una "no linealidad" en el comportamiento de la neurona, ya que, si éste se elimina, la salida de la neurona sería simplemente una función lineal de la suma ponderada de

las entradas, incluso después de pasar por la función de activación, en cambio con un sesgo la neurona podría activarse incluso cuando la suma de las entradas sea cero, lo cual introduce no linealidad en la relación entre las entradas y la salida.

Función de activación

La función de activación es una función matemática que se aplica a la suma ponderada de las entradas de la neurona para determinar su salida, es decir, si se activa o no, y son las base para el aprendizaje de las redes neuronales.

Estas funciones pueden ser lineales y no lineales, la elección de alguna depende de la tarea que se pretenda resolver con la red neuronal. Las primeras son comúnmente utilizadas para tareas relativamente simples de regresión y clasificación, en cambio las funciones no lineales son capaces de representar funciones más complejas.

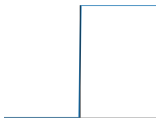
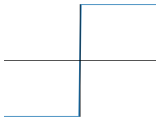
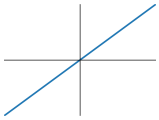
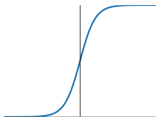
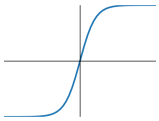
Actualmente, las funciones de activación más utilizadas son las funciones no lineales ya que permiten a una neurona tener una salida que no sea simplemente proporcional al cambio de sus entradas. Esta no linealidad hace posible que una red neuronal sea capaz aprender y representar funciones y patrones más complejos y establecer relaciones no lineales en los datos.

Existen numerosas funciones de transferencia utilizadas en las ANN y su desarrollo se ha venido incrementando en los últimos años, especialmente de las no lineales [35], debido al desarrollo de redes neuronales con un gran número de capas de neuronas —profundas— ya que necesitan aprender funciones complejas.

En la tabla 2.2 se muestran algunas de las funciones de activación más básicas utilizadas en las ANN.

Si bien cada una presenta ventajas y desventajas con respecto a las demás, su mayor limitación es que muchas de ellas llegan a ser lineales en ciertas regiones y son propensas a la saturación,

Tabla 2.2: Funciones de activación básicas.

Función	Ecuación	Gráfica
Escalón	$f(x) = 0 \quad x < 0$ $f(x) = 1 \quad x \geq 0$	
Escalón simétrica	$f(x) = -1 \quad x < 0$ $f(x) = +1 \quad x \geq 0$	
Lineal	$f(x) = ax$	
Sigmoidal	$f(x) = \frac{1}{1+e^{-x}}$	
Tangente hiperbólica	$f(x) = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$	

es decir, que sus salidas se vuelven constantes para valores de entrada grandes.

Debido a que las redes profundas suelen utilizar funciones de activación no lineales, en los últimos años ha aumentado el desarrollo de nuevas funciones de este tipo, como la Unidad Lineal Rectificada o "*ReLU*" (por sus siglas en inglés *Rectified Linear Unit*) y sus múltiples variantes, la función *Swish* y la función *Mish*.

Aunque esas tres funciones han demostrado ser eficientes en redes profundas, *ReLU* ha sido una de las más utilizadas. Su descripción matemática está dada por la ecuación (2.10).

$$f(x) = \text{máx}(0, x) \tag{2.10}$$

Que también puede escribirse por medio de la ecuación (2.11).

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \tag{2.11}$$

Como se observa en la ecuación (2.11), la salida de la función será cero si la entrada es negativa y cuando es positiva devolverá el mismo valor de entrada. La gráfica de *ReLU* se muestra en la figura 2.7.

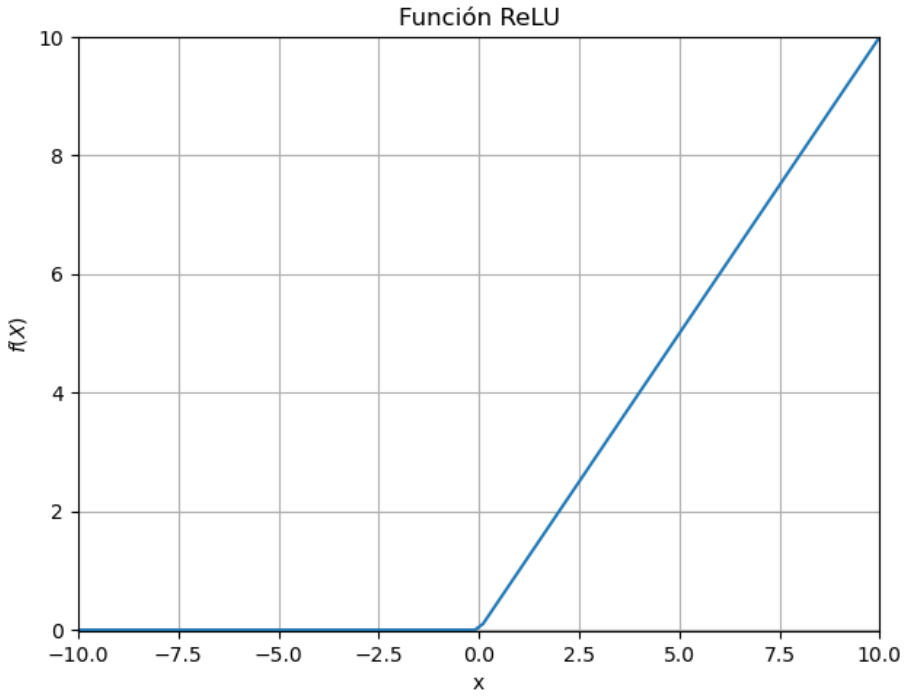


Figura 2.7: Gráfica de la función de activación ReLU.

Entre las ventajas de esta función es que no existe una saturación en la parte positiva por tener una respuesta lineal, además matemáticamente es una función sencilla, por lo que es fácil de implementar computacionalmente.

A pesar de que *ReLU* ha demostrado tener buenos resultados, también presenta algunas limitaciones como su saturación en la parte negativa. Esto puede generar un problema comúnmente conocido como "*dying ReLU*", cuya interpretación al español sería "*ReLU* muerta" o "bloqueada". Esto se produce cuando una neurona con esta función de activación comienza a generar siempre una salida nula, lo que podría provocar que la red neuronal pierda precisión o no pueda aprender funciones nuevas.

Para solucionar las limitaciones de *ReLU* se han desarrollado numerosas variantes, entre las más utilizadas están las funciones conocidas como *Leaky ReLU*, *SiLU* y *Softplus*, donde la

primera fue la utilizada en la red neuronal que se describirá en la segunda parte experimental de esta tesis.

Leaky ReLU

Una interpretación al español del nombre *Leaky ReLU* sería "*ReLU* con fugas", esto debido a su comportamiento similar a *ReLU* cuando la entrada es positiva, pero cuando la entrada es negativa *Leaky ReLU* permite que las neuronas sigan generando una salida con una pequeña pendiente, muy cercana a cero sin llegar a ser nula. Matemáticamente se describe mediante la ecuación (2.12).

$$f(x) = \begin{cases} \alpha x & x \leq 0 \\ x & x > 0 \end{cases} \tag{2.12}$$

Donde α es el coeficiente de la pendiente que tendrá la salida cuando los valores de entrada sean negativos. Este coeficiente es una constante positiva que se determina antes del entrenamiento. La representación gráfica de *Leaky ReLU* se muestra en la figura 2.8

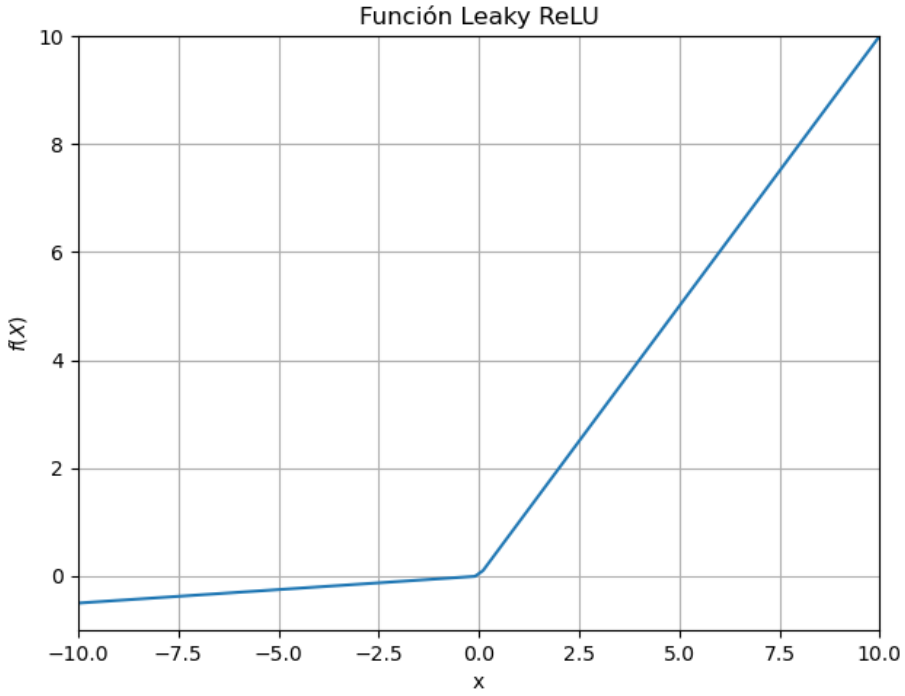


Figura 2.8: Gráfica de la función de activación Leaky ReLU.

2.4.3 Perceptrón

El modelo del perceptrón (desarrollado en 1958 por Frank Rosenblatt y colaboradores [31]) es básicamente el modelo de la neurona mostrada en la figura 2.6: consta de pesos en cada entrada, un sesgo, una unidad de procesamiento que suma los productos de las entradas y sus pesos y una función de transferencia para producir la salida.

Su particularidad es la utilización de una función escalón o escalón simétrico (ver tabla 2.2) como función de activación. Estas funciones se caracterizan por su capacidad de dividir la entrada en dos posibles valores, por lo que fue comúnmente utilizado para la clasificación de datos.

Esa cualidad de su función de activación, representa la principal característica —y limitación— del perceptrón, ya que solo es capaz de separar los datos de entrada en dos categorías linealmente separables.

Tomando como ejemplo un perceptrón de dos entradas con la función escalón como función de activación, su salida estará dada por la ecuación (2.13).

$$f(x) = \begin{cases} -1 & w_1 x_1 + w_2 x_2 + b < 0 \\ 1 & w_1 x_1 + w_2 x_2 + b \geq 0 \end{cases} \tag{2.13}$$

Como se puede observar en la ecuación (2.13), cuando la suma sea positiva la salida será 1 y cuando la suma sea negativa -1. Partiendo de ese punto, si se buscan los parámetros para los cuales la salida del perceptrón sea 0, se tendrá la ecuación (2.14).

$$w_1 x_1 + w_2 x_2 + b = 0 \tag{2.14}$$

Cuya estructura es similar a la ecuación de la línea recta, tomando a w_1 y w_2 como x y y, respectivamente.

La línea recta formada por la ecuación (2.14) representa la frontera de decisión de un perceptrón de dos clases, la cual separa linealmente las dos categorías. En la figura 2.9 se muestra

la frontera de decisión, suponiendo que los valores de x_1 y x_2 y b en la ecuación (2.14) son 1, 1 y -1, respectivamente.

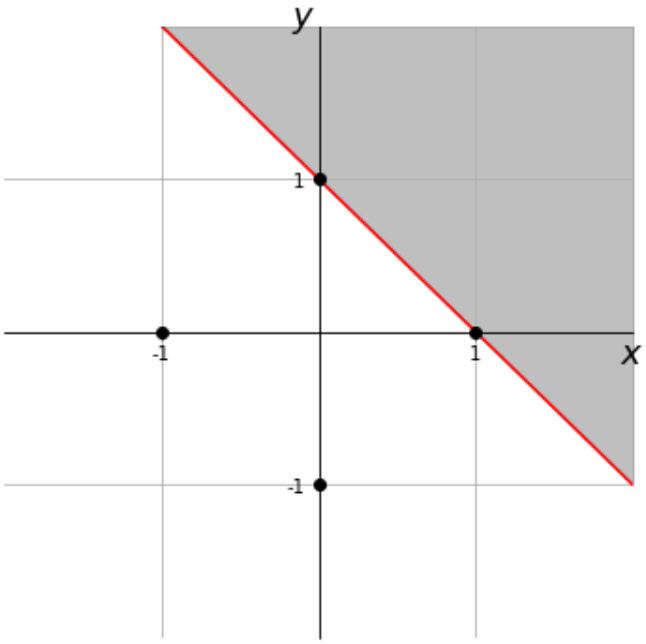


Figura 2.9: Frontera de decisión de un perceptrón de dos entradas.

Cabe señalar que en un perceptrón de tres entradas la frontera que separará las dos clases será un plano.

2.4.4 Regla de aprendizaje del Perceptrón

Como se mencionó anteriormente, el entrenamiento del perceptrón era capaz de llegar a una solución al entrenarse con conjuntos de entrenamiento linealmente separables. A su tipo de algoritmo de entrenamiento se le conoce como aprendizaje supervisado, ya que durante el entrenamiento se le presentan datos de entrada junto con la salida esperada, conocida como *target* t. Las reglas de aprendizaje del perceptrón pueden resumirse de la siguiente manera:

1. **Inicialización aleatoria de parámetros**. El primer paso consiste en inicializar con valores aleatorios los pesos y el sesgo del perceptrón.

2. **Cálculo de la salida del Perceptrón**. Se le presentan al modelo los parámetros de entrada y se calcula la salida del perceptrón y, es decir, se realiza la suma ponderada y se pasa por la función de transferencia. En un perceptrón de n entradas la salida estará dada por (2.15).

$$y = \begin{cases} 0 & \sum_{i=1}^{n}(w_i x_i) + b < 0 \\ 1 & \sum_{i=1}^{n}(w_i x_i) + b \geq 0 \end{cases} \tag{2.15}$$

3. **Cálculo del error**. Utilizando una variable conocida como error e, se compara la salida del perceptrón con la salida esperada t como se muestra en la ecuación (2.16).

$$e = t - y \tag{2.16}$$

4. **Ajuste de pesos y sesgo**. Si $e \neq 0$, es decir, si no coinciden, se ajustan los pesos y el sesgo por medio de las reglas mostradas en las ecuaciones (2.17) y (2.18), respectivamente.

$$w_i^{nuevo} = \begin{cases} w_i + \alpha e x_i & e < 0 \\ w_i - \alpha e x_i & e \geq 0 \end{cases} \tag{2.17}$$

$$b^{nuevo} = \begin{cases} b + \alpha e & e < 0 \\ b - \alpha e & e \geq 0 \end{cases} \tag{2.18}$$

   Donde α es un parámetro conocido como tasa de aprendizaje y controla el porcentaje del error que se ajustará.

5. **Iteración**. Al ajustarse los pesos se repiten los pasos 2 y 3 hasta que se llegue al número máximo de iteraciones previamente establecido o cuando $e = 0$, en este caso significa que se ha llegado a la solución.

De acuerdo con el teorema de convergencia del perceptrón, demostrado por Minsky y Papert en [36], se garantiza que, al entrenar un perceptrón el algoritmo mencionado convergerá a una solución en un número finito de iteraciones si los datos se pueden clasificar en dos clases linealmente separables.

Si bien el ejemplo mostrado se realizó con un solo perceptrón, estas mismas reglas de aprendizaje se pueden aplicar para múltiples perceptrones en una sola capa. Para ello es común utilizar tensores para representar los parámetros (pesos, sesgos y entradas) y poder operarlos.

2.4.5 Método de propagación hacia atrás

La limitación de los perceptrones de únicamente clasificar clases linealmente separables causó que sus aplicaciones prácticas fueran escasas y esto se dejó en evidencia en el mismo documento de Minsky y Papert donde demostraron el teorema de convergencia, casi una década después de la introducción del perceptrón [36]. En este documento destacaron el problema de la XOR, que básicamente mostraba la incapacidad de que un perceptrón o una capa con varios perceptrones pudieran representar la función lógica XOR de dos bits, por su incapacidad de aprender patrones no lineales.

Esto causó que disminuyera el financiamiento y el interés en la investigación de las redes neuronales artificiales y el aprendizaje automático, hasta que en la década de 1980 varios autores redescubrieron la tesis de Paul Werbos escrita en 1974 [37], donde presentaba la aplicación de un algoritmo conocido como *backpropagation* (propagación hacia atrás) para entrenar redes neuronales multicapa. Utilizando el algoritmo de *backpropagation* desarrollaron una regla de entrenamiento conocida como regla delta con la que fueron capaces de entrenar redes multicapa del perceptrón, superando las limitaciones antes mencionadas. También demostraron que una red multicapa de tipo *feedforward* podía aprender cualquier función continua.

Backpropagation es un método de aprendizaje automático utilizado para entrenar redes neuronales multicapa, su nombre hace referencia a que se propaga el error desde la capa de salida hasta las capas ocultas de la red. Éste utiliza la regla delta, que es una generalización de un método conocido como gradiente descendente para minimizar la función de error. Por ello, para comprender *backpropagation* es conveniente conocer el algoritmo de gradiente descendente.

Gradiente descendente

En términos sencillos, el mínimo de una función corresponde al valor más bajo que una función puede alcanzar y, matemáticamente, se obtiene analizando la derivada de la función, ya que, en un punto mínimo, la derivada de la función, o sea su gradiente o pendiente, es igual a cero.

En el contexto del aprendizaje automático para entrenar redes neuronales, el gradiente descendente se utiliza para minimizar la función de error, también conocida como función de pérdida.

La idea del algoritmo del gradiente descendente es ajustar los parámetros en dirección opuesta al gradiente de la función de error de manera iterativa, esto significa que los parámetros se actualizan en la dirección en que la función de error decrece haciendo uso del concepto de la derivada.

En este algoritmo se utiliza la derivada parcial de la función de error con respecto a cada parámetro, para determinar la dirección en la que se deben ajustar los parámetros. Esta derivada parcial, también es conocida como sensibilidad, ya que indica cuánto cambia la función de error al modificar un parámetro específico de la red neuronal, es decir, indica qué tan sensible es la función de error con respecto a ese parámetro.

El algoritmo del gradiente descendente se puede describir de la siguiente manera:

1. **Inicialización**. Se comienza en cualquier punto elegido aleatoriamente.

2. **Cálculo del gradiente.** Se calcula el gradiente de la función de error en el punto actual k por medio de la derivada, como se muestra en la ecuación (2.19).

$$\nabla E(x_k) = \left.\frac{\partial E(x)}{\partial x}\right|_{x_k} \tag{2.19}$$

3. **Actualización del punto**. Se actualiza el punto actual en la dirección del gradiente negativo por medio de la ecuación (2.20).

$$x_{k+1} = x_k - \alpha \nabla E(x_k) \tag{2.20}$$

   Donde x_k es el punto actual y α es la tasa de aprendizaje.

4. **Iteración**. Se repiten los pasos 2 y 3 hasta alcanzar un criterio de parada, ya sea por la convergencia del algoritmo o por un número máximo de iteraciones.

En palabras simples, este algoritmo toma el punto actual y le resta una parte del valor de la derivada en ese punto para acercarse cada vez más al mínimo en cada iteración. En la figura 2.10 se muestra una representación de este algoritmo con la función de una parábola $f(x) = x^2 + 1$.

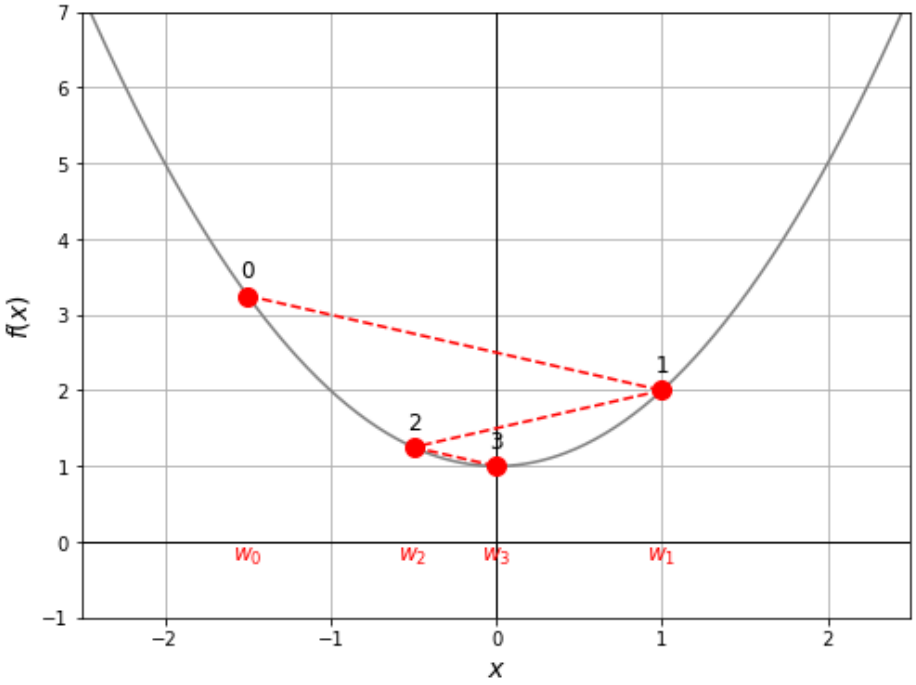


Figura 2.10: Descripción gráfica del algoritmo del gradiente descendente.

La tasa de aprendizaje α es un valor positivo y por lo regular se define en un rango entre 0 y 1, de tal modo que $0 < \alpha < 1$. El efecto de la tasa de aprendizaje es fundamental para la convergencia del algoritmo, ya que, si su valor es grande, cada nuevo punto podría oscilar

entre los extremos del mínimo sin llegar a él, impidiendo la convergencia; en el caso opuesto, si es muy pequeño, si tendrá convergencia, pero necesitaría un gran número de iteraciones para llegar al mínimo. La elección de su valor se determina de manera heurística, ya que no existe una manera analítica de hacerlo.

Error Cuadrático Medio

Previo a la descripción del algoritmo de *backpropagation*, es importante mencionar que existen diversas maneras de medir el error de una ANN. El error mostrado en la ecuación (2.16), es una de ellas, sin embargo, es más común utilizar funciones de error para redes multicapa.

La función de error (también conocido como función de pérdida) cuantifica la discrepancia entre el valor obtenido de la red y el valor real, expresado en función de los parámetros de la red. Aunque existen numerosas funciones de error, una de las más utilizadas es el error cuadrático medio o MSE (por sus siglas en inglés: *Mean Squared Error*) descrito por la ecuación (2.21). MSE es la función de error que se utiliza en el algoritmo de backpropagation y también en la red neuronal utilizada en la parte experimental de esta tesis.

$$MSE = \frac{1}{2}\sum_{k=1}^{n}(t_k - y_k)^2 \tag{2.21}$$

En la ecuación (2.21), t_k y y_k representan el valor objetivo y el valor obtenido a la salida de la red, respectivamente, en cada ejemplo k con un número n de neuronas en la capa de salida .

A las funciones de error también se les conoce como índice de desempeño porque su valor es un indicativo de qué tan bien está realizando su función la red neuronal.

Algoritmo de *backpropagation*

Para el aprendizaje automático utilizando este algoritmo se utilizan pares de datos de entradas y salidas durante el entrenamiento, conocidos como conjuntos de entrenamiento.

El objetivo de *backpropagation* es minimizar el MSE cometido por la capa de salida, ajustando los parámetros de la red. Para ello se expresa la función de error en términos de dichos parámetros y se aplica el método de gradiente descendente, propagando el error de la capa de salida hacia las capas ocultas utilizando la regla de la cadena en la derivada de la ecuación del gradiente descendente (2.19) para que todo se realice en términos de los pesos y sesgos de la red.

El algoritmo de *backpropagation* se puede describir de la siguiente manera:

1. **Inicialización**. Se inicializan los pesos y sesgos de la red neuronal con valores aleatorios pequeños.

2. **Propagación hacia adelante**. En este paso se calcula la salida de la red. Para ello se toman los datos de entrada y se mapean hacia la salida. El término propagación hacia adelante (conocido como *forward propagation* en inglés) hace referencia a que, como en una red neuronal multicapa las salidas de las neuronas de una capa son las entradas de la capa siguiente, el cálculo comenzará desde la capa de entrada y su respuesta pasará a las capas ocultas, propagándose la salida de cada capa a la siguiente hasta llegar a la capa de salida.

3. **Cálculo del error**. Se calcula la función de error , en este caso el MSE con la ecuación (2.21), con el conjunto de entrenamiento en turno.

4. **Propagación hacia atrás**. A este paso hace referencia el nombre del algoritmo y lo que se hace es aplicar el gradiente descendente, propagando hacia atrás el error calculado desde la capa de salida, pasando por las capas ocultas hasta llegar a la capa de entrada. En cada capa, se utilizan reglas de la cadena para calcular los gradientes de la función de pérdida

con respecto a los parámetros de esa capa. De esta manera en cada neurona, se calcula la derivada parcial de la función de error con respecto a la salida de la neurona anterior para que la expresión esté en términos de los parámetros de cada neurona.

5. **Actualización de parámetros**. Con los gradientes calculados se van ajustando los parámetros de la red con el mismo algoritmo de gradiente descendente. Matemáticamente el ajuste de los pesos y de los sesgos se pueden expresar por medio de las ecuaciones (2.22) y (2.23), respectivamente.

$$w_{i,j}(l) = w_{i,j}(l) - \alpha \frac{\partial E(l)}{\partial w_{i,j}} \tag{2.22}$$

$$b_i(l) = b_i(l) - \alpha \frac{\partial E(l)}{\partial b_i} \tag{2.23}$$

   Donde l representa la capa de la red neuronal que se está procesando, α la tasa de aprendizaje y $\partial E(l)$ la derivada parcial de la función de error que, implícitamente, contiene el error de las capas anteriores.

   Es importante mencionar que para poder computar todos los parámetros de la red se utilizan tensores, es decir, se expresan los pesos, sesgos y el error en estructura de datos en forma de matrices o vectores para poder realizar los cálculos de una manera eficiente, por lo que las ecuaciones (2.22) y (2.23) serían equivalentes a las ecuaciones (2.24) y (2.25), respectivamente.

$$W^m(k+1) = W^m(k) - \alpha \frac{\partial}{\partial n^m}(E^{m-1})^T \tag{2.24}$$

$$B^m(k+1) = B^m(k) - \alpha s^m \tag{2.25}$$

   Donde W representa la matriz de pesos, B la matriz de sesgos, m la capa, n la neurona y k la iteración.

6. **Iteración**. Se repiten del paso 2 al 5 hasta que se alcance un criterio de convergencia o se llegue a un número determinado de iteraciones.

Al igual que en el algoritmo de gradiente descendente, para el entrenamiento utilizando *backpropagation* se deben definir previamente la tasa de aprendizaje y el número máximo de iteraciones.

Con el desarrollo del algoritmo de *backpropagation* se logró superar la limitación que tenían los perceptrones, reanudando así la investigación de este tipo de computación y con ello, arquitecturas cada vez más complejas.

2.4.6 Redes Neuronales Artificiales

Debido a que las ANN están inspiradas en la funcionalidad del cerebro, para formar una red se suelen organizar varias neuronas para que puedan trabajar en paralelo. A esta organización de neuronas se le conoce como capa, donde cada neurona tiene sus propios elementos: pesos, sesgo y función de activación.

Las neuronas de una misma capa comparten las mismas entradas, o sea que todas las neuronas de una capa reciben la misma información. En la figura 2.11 se muestra la representación de una capa de 3 neuronas conectada a las entradas de la red.

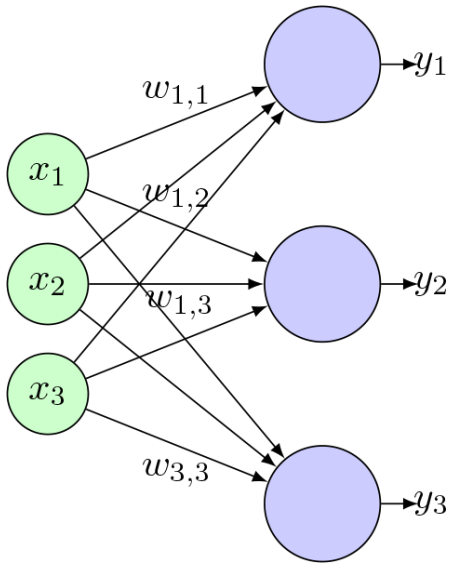


Figura 2.11: Capa de 3 neuronas.

Como se puede observar en la figura 2.11, cada entrada está conectada a todas las neuronas y el número de neuronas que conforma la capa determinará el número de salidas que tendrá la capa. Es importante mencionar que, por simplicidad, solamente se dibujaron los pesos de la

primera entrada y el último de la tercera entrada, sin embargo, a cada neurona le corresponde un peso por cada entrada, al igual que el sesgo, que también se omitió en el dibujo por simplicidad.

Por lo regular, una red neuronal está compuesta por varias capas donde cada neurona está conectada a las neuronas de las capas adyacentes, pero no hay conexión entre las neuronas de una misma capa. A este tipo de redes se les conoce como redes multicapa, donde la primera es la capa de entrada y es la que recibe los datos de entrada de la red, la última es conocida como la capa de salida y las capas intermedias se les conoce como capas ocultas y son las encargadas de procesar los datos.

Al tipo de capas donde todas sus neuronas están conectadas a todas las neuronas de la capa anterior y la capa siguiente se les conoce como capas completamente conectadas o *fully connected*.

En la figura 2.12 se muestra una ANN de cinco capas, donde la segunda, tercera y cuarta capa corresponden a la denominada capa oculta de la red.

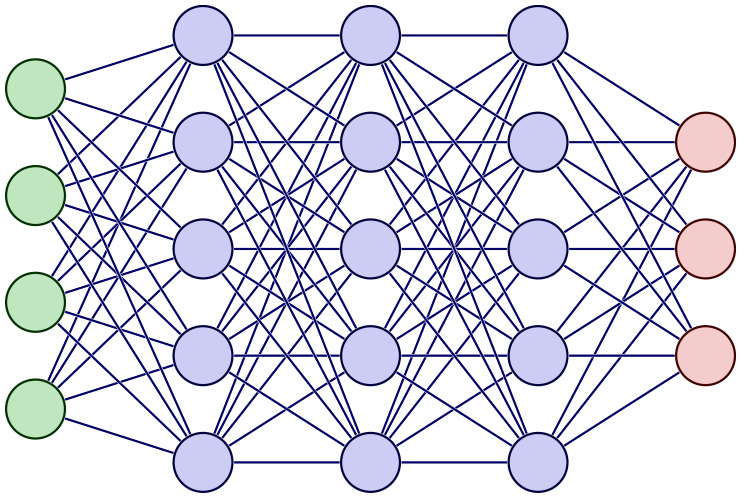


Figura 2.12: ANN de cuatro capas.

Al igual que en la figura 2.11, en la figura 2.12 se omitieron los pesos de las entradas y el sesgo en cada neurona por simplicidad. Además, a diferencia de los dibujos de neuronas anteriores, en la figura 2.12 la primera capa representa la capa de entrada de la red y no las entradas como tal, ya que, aunque no está estandarizado, es la manera más común de representar las ANN.

Debido a que cada neurona entrega un solo valor, el número de neuronas de la capa de salida es la que determina el número de salidas de la red. Esto se puede interpretar en que el número de neuronas que tenga la capa de salida debe ser igual al número de salidas que requiere el problema que se pretende solucionar con la red, para que ésta pueda generar una salida correcta. De manera similar, el número de datos de entrada debe ser igual al número de entradas de la red.

Anteriormente se ha mencionado que, gracias al avance tecnológico, en los últimos años se han desarrollado redes profundas, las cuales son redes multicapa con un gran número de capas intermedias —en la capa oculta— y un gran número de neuronas en cada capa. Aunque actualmente no hay ninguna convención para determinar a partir de cuántas capas ocultas se considera que una red es profunda, algunos autores consideran que es a partir de una, como Cichy y Kaiser lo mencionan en [38] y otros autores consideran que a parir de dos como Rafael et al. en [39], e incluso, en la práctica, algunos consideran que a partir de cuatro. Sin embargo, más que con la cantidad de capas y neuronas, los autores consideran que la profundidad de una red neuronal está relacionada con su capacidad para aprender patrones y relaciones cada vez más complejas a medida que la información pasa a través de las diferentes capas.

2.4.7 Arquitectura de las ANN

Actualmente existen numerosas arquitecturas de red y la mayoría de ellas son profundas. Cada arquitectura está diseñada para tratar con algún tipo de dato específico o para realizar alguna tarea específica, sin embargo, muchas de ellas se pueden clasificar entre los siguientes tipos de arquitectura.

Redes de propagación hacia adelante. Mejor conocidas por su nombre en inglés: *feedforward networks*. En este tipo de redes la información fluye en una sola dirección, hacia adelante desde la capa de entrada hacia la salida de la red. Son comúnmente utilizadas para tareas

de clasificación, regresión y reconocimiento de patrones.

Redes Neuronales Recurrentes. Mejor conocidas como RNN (del inglés: *Recurrent Neural Network*), en este tipo de redes existe retroalimentación, es decir, algunas salidas de las neuronas pueden estar conectadas a sus entradas. Se utilizan para tareas que requieren trabajar con datos secuenciales como reconocimiento de voz, traducción automática, predicción de series temporales, etc.

Autocodificadoras. Conocidas como *Autoencoder*. Su arquitectura consta de dos redes neuronales, un codificador y un decodificador. El primero se encarga de codificar los datos de entrada en una representación de menor dimensión y el segundo se encarga de reconstruir los datos de entrada a partir de la salida del codificador. Su principal aplicación es para aprender codificaciones de datos no etiquetados. Entre las tareas más comunes que se realizan con estas redes es la extracción de características y la reducción de dimensionalidad.

Redes Siamesas. Estas redes constan de dos redes neuronales idénticas, o sea que comparten la misma estructura y los pesos. Se utilizan para comparar dos entradas, por ejemplo, para verificar identidades, comparación de datos, reconocimiento de similitud, etc.

Redes Neuronales Generativas Adversarias. Las GAN (del inglés: *Generative Adversarial Network*) se utilizan para generar datos nuevos. Constan de una red generadora y una discriminadora, aunque la arquitectura de estas redes varía dependiendo de la tarea, comúnmente utilizan redes convolucionales, recurrentes o una mezcla de ambas.

Redes neuronales convolucionales. Mejor conocidas como CNN por su nombre en inglés *Convolutional Neural Network*, son redes neuronales que utilizan el concepto matemático de la convolución para la extracción de características. Están diseñadas para tratar con un gran número de datos en forma de tensores, tal como las imágenes. La eficiencia de las CNN en el procesamiento de imágenes radica en que su diseño les permite detectar patrones en pequeñas regiones de la imagen utilizando filtros convolucionales.

Debido a que la teoría de cada tipo de arquitectura es considerablemente extensa, en este trabajo solo se abordarán las redes neuronales convolucionales, ya que en la parte experimental de esta tesis se utilizó una para clasificar las imágenes de fondo de ojo. Debido a que comúnmente se utilizan para tratar con imágenes, previo a la descripción de las CNN se presentará el tema de las imágenes digitales, con el fin de que la comprensión de este tipo de redes pueda ser más fluida.

2.4.8 Aprendizaje automático (entrenamiento de una ANN)

Como se mencionó anteriormente, el entrenamiento de una red neuronal consiste en ajustar los parámetros de la red (los pesos y sesgos) de manera automática mediante la minimización de una función de error, para que la red pueda realizar alguna tarea específica. Para ello se utilizan reglas de aprendizaje (también conocidos como reglas de entrenamiento o algoritmos de entrenamiento), que se refiere a los algoritmos utilizados para ajustar dichos parámetros.

Actualmente existen numerosos algoritmos para entrenar todo tipo de redes, sin embargo, en general se pueden clasificar en tres categorías:

Aprendizaje supervisado. En este tipo de aprendizaje se utilizan conjuntos de entrenamiento, que son datos de entrada y salida. Estos se presentan en la red de la forma $\{X, t\}$, donde X representa los datos de entrada y t la salida o *target* que debería arrojar la red con esos datos de entrada. En este tipo de aprendizaje, el algoritmo ajusta los parámetros de las neuronas ingresando los datos de entrada y comparando la salida de la red con el *target* de manera iterativa, con el objetivo de que en cada nueva iteración la salida se acerque lo más posible al valor objetivo. *Backpropagation* es un tipo de aprendizaje supervisado.

Aprendizaje no supervisado. En el aprendizaje no supervisado el entrenamiento se realiza con

datos sin etiquetas de salida, es decir, no se les presenta información sobre la salida esperada. Por lo tanto, los parámetros de las neuronas se modifican en respuesta a las entradas para que el modelo aprenda a encontrar patrones en los datos. Aunque existen diversos algoritmos para este tipo de entrenamiento, la mayoría de ellos se utilizan principalmente para tareas de agrupamiento y para reducción de dimensionalidad.

Aprendizaje por refuerzo. Este aprendizaje es similar al supervisado, la diferencia es que, en lugar de presentarle los valores objetivo se le da una "recompensa", que representaría una medida del desempeño. De esta manera aprende un "buen comportamiento" a través de la prueba y error. Como es posible intuir por lo descrito anteriormente, el aprendizaje por refuerzo se utiliza para aplicaciones donde se tengan que tomar decisiones, por lo que, durante el entrenamiento, el modelo debe aprender a tomar decisiones a través de la retroalimentación y la experiencia que vaya tomando. Usualmente los algoritmos para este aprendizaje se utilizan para aplicaciones en sistemas de control.

En la práctica, para poder entrenar una red neuronal es necesario definir el algoritmo de aprendizaje, la tasa de aprendizaje y el número de épocas. Estos son conocidos como "hiperparámetros" y tienen el objetivo de controlar el proceso de aprendizaje de la red neuronal. Aparte de esos tres existen muchos otros hiperparámetros como el tamaño del lote, que define la cantidad de muestras que se utilizarán en cada iteración del entrenamiento. Cabe destacar que muchos de estos hiperparámetros varían según la arquitectura del modelo, ya que diferentes redes neuronales pueden requerir ajustes específicos que otras no.

2.5 Imágenes digitales

El trabajo experimental de esta tesis consiste en el procesamiento de imágenes, por ello, aparte de la base teórica de la lógica difusa y las redes neuronales, es necesario definir lo que es

una imagen, cómo se representa y qué implica el procesamiento de una imagen. Esto también será necesario para comprender el funcionamiento de las CNN.

En términos simples, una imagen se puede definir como una representación bidimensional de las características de una escena, como el brillo o color. Matemáticamente, se representa como una función de dos variables $f(x, y)$, donde (x, y) denota la posición en el plano y la amplitud $f(x, y)$ la intensidad de la magnitud física representada.

Cuando la posición de una imagen puede ser representada por un rango continuo de valores y su amplitud varía constantemente, se conoce como imagen analógica; en caso contrario, cuando sus valores son finitos y discretos, se tiene una imagen digital.

2.5.1 Elementos básicos de las imágenes digitales

Aunque existen diversas maneras de adquirir imágenes digitales, la mayoría de ellas se obtienen a partir de sensores, los cuales generan imágenes analógicas, por lo que es necesario aplicar los procesos de muestreo y cuantización para poder obtener la imagen digital. El muestreo consiste en digitalizar las coordenadas y la cuantización digitalizar los valores de amplitud, por lo tanto, en las imágenes digitales (x, y) son coordenadas discretas y finitas. Esto permite que este tipo de imágenes se puedan representar de diferentes maneras, siendo como una matriz de tamaño $M \times N$ la más común, donde M es el valor máximo de x y N el de y; y cada celda de la matriz contiene el valor numérico de $f(x, y)$. Así, una imagen digital se puede representar como se muestra en la ecuación (2.26).

$$f(x,y) = \begin{bmatrix} f(0,0) & \cdots & f(0,N-1) \\ \vdots & \ddots & \vdots \\ f(M-1,0) & \cdots & f(M-1,N-1) \end{bmatrix} \tag{2.26}$$

Donde cada elemento de la matriz es un número real y es conocido como píxel, el cual es el elemento más básico de una imagen digital. De este modo, es posible definir una imagen digital como una composición de píxeles.

Rango dinámico

Con lo mencionado anteriormente, es posible decir que las imágenes digitales son una representación de la realidad en forma de datos, que son almacenados y procesados por computadoras. De esta manera, cada píxel de una imagen digital representa una pequeña unidad de información del nivel de intensidad que éste tiene y, ya que se almacena en formato digital, su valor está dado en bits.

El rango de valores que puede tomar un píxel se denomina rango dinámico de canal y representa el número de niveles de intensidad L que pueden tener los píxeles de una imagen digital. Debido a que este valor está determinado por la cantidad de bits utilizados, suele estar dado por una potencia de dos, tal como se observa en la ecuación (2.27).

$$L = 2^n \tag{2.27}$$

Donde n representa el número de bits empleados para cuantizar los niveles de intensidad de un píxel y se conoce como bits de resolución de intensidad. Aunque no es la única opción, comúnmente se utilizan 8 bits (1 byte) para la cuantización, lo que permite tener 256 niveles. En este caso el rango dinámico de canal toma valores desde 0 hasta 255, o sea $[0, L-1]$, donde al límite superior (255) se le conoce como valor de saturación e indica el punto más brillante que la imagen puede mostrar, mientras que el límite inferior (0), denominado valor de ruido, representa el nivel más oscuro.

Mientras más bits se utilicen mayor será el rango dinámico de canal. Con esta información es posible graficar la distribución de los valores de intensidad de los píxeles de la imagen, es decir, la distribución de probabilidad de los píxeles. A este gráfico se le conoce como histograma y se considera como un gráfico estadístico, donde el eje horizontal representa los valores de intensidad posibles y el eje vertical la frecuencia de los píxeles por cada valor de intensidad. En la figura 2.13 se muestra el histograma de una imagen utilizada en la parte experimental de esta tesis, donde se utilizan 8 bits de profundidad, es decir 8 bits para la cuantización.

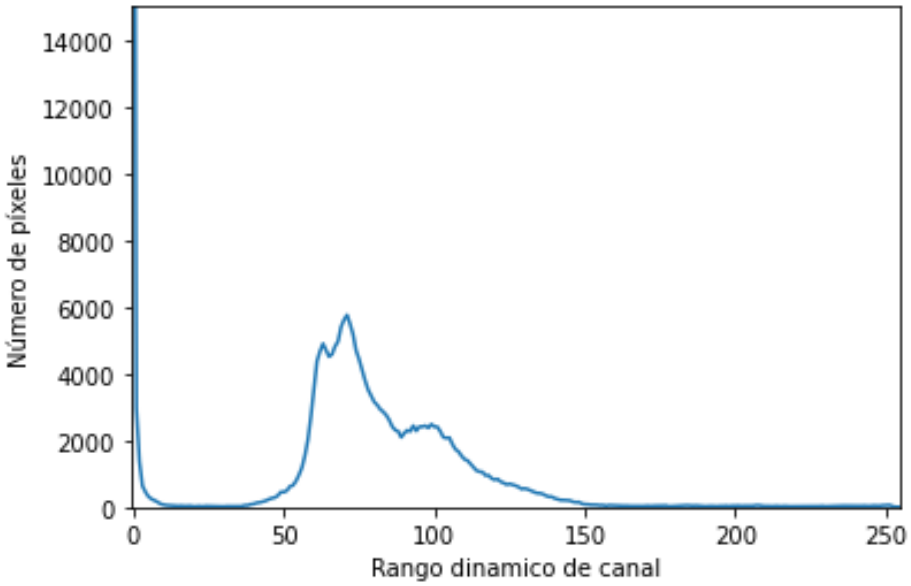


Figura 2.13: Histograma de una imagen con 8 bits de profundidad.

Matemáticamente, el histograma h de una imagen digital se puede definir por medio de la ecuación (2.28).

$$h(i_k) = n_k \tag{2.28}$$

Donde i_k representa el nivel de intensidad, n_k el número de píxeles con esa intensidad, y $k = 0, 1, 2, 3, \cdots, L-1$, por tener valores discretos.

Cabe mencionar que el término de "rango dinámico" tiene diferentes significados dependiendo del contexto donde se utilice. Anteriormente se definió el rango dinámico de canal, pero también existe el "rango dinámico de una imagen" o simplemente rango dinámico. Éste indica los valores mínimos y máximos que tiene una imagen, los cuales estarán dentro del rango dinámico del canal. En la figura 2.14 se puede observar de manera más clara.

2.5.2 Contraste

Con el rango dinámico es posible distinguir los valores límite de intensidad que tiene una imagen, donde la diferencia entre estos valores define el contraste. El contraste se refiere a la

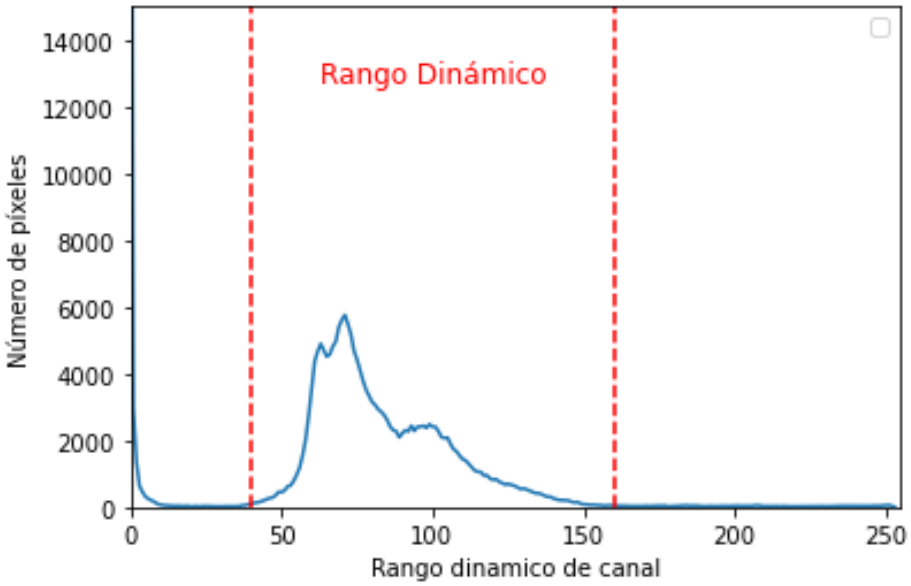


Figura 2.14: Rango dinámico de canal y rango dinámico de una imagen.

variación de intensidades entre los píxeles de una imagen, de tal manera que, cuanto mayor sea la diferencia entre los valores de saturación y ruido, mayor será el contraste, lo que permitirá resaltar los detalles visuales de la imagen y crear una sensación de profundidad.

Existen varios parámetros para cuantificar el contraste de las imágenes, uno de los más utilizados es la desviación estándar. Ésta se obtiene considerando los valores de los píxeles como variables aleatorias, de esta manera se puede calcular la probabilidad de cada nivel de intensidad $p(i_k)$ de una imagen $f(x, y)$ por medio de la ecuación (2.29).

$$p(i_k) = \frac{n_k}{MN} \tag{2.29}$$

Donde MN representa el total de píxeles de la imagen.

Al considerar los píxeles como variables aleatorias es posible calcular la media de los valores de intensidad $\bar{i}$, tal como se muestra en la ecuación (2.30).

$$\bar{i} = \sum_{k=0}^{L-1} i_k p(i_k) \tag{2.30}$$

Como se mencionó anteriormente, el contraste de una imagen es la diferencia entre el límite superior e inferior del valor de los píxeles. Bajo esta premisa, la varianza σ^2, que es una medida

de dispersión de los valores con respecto a la media, proporcionará información sobre contraste de la imagen, de tal manera que una varianza alta indicará que los valores límite están muy separados y, por lo tanto, el contraste será alto. Su cálculo se obtiene por medio de la ecuación (2.31).

$$\sigma^2 = \sum_{k=0}^{L-1} (i_k - \bar{i})^2 p(i_k) \tag{2.31}$$

Considerando que los valores de intensidad de los píxeles de una imagen están contenidos en una matriz, el cálculo de la varianza puede expresarse como se muestra en la ecuación (2.32).

$$\sigma^2 = \frac{1}{MN} \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} (i_{j,k} - \bar{i})^2 \tag{2.32}$$

Donde M y N indican el tamaño de la matriz, $\bar{i}$ es el promedio del valor de los píxeles e $i_{j,k}$ es el valor del píxel de la celda (j, k) de la matriz.

Por lo tanto, la desviación estándar σ se obtiene por medio de la ecuación (2.33)

$$\sigma = SD = \sqrt{\frac{1}{MN} \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} (i_{j,k} - \bar{i})^2} \tag{2.33}$$

Al igual que la varianza, la desviación estándar mide la dispersión de los valores de los píxeles con respecto a la media, por lo que es una medida cuantitativa del contraste, cuyo su valor es directamente proporcional al nivel de contraste visual de la imagen.

Como se muestra en la ecuación (2.33), para obtener la desviación estándar en las imágenes, se calcula la raíz cuadrada del promedio de los valores de los píxeles al cuadrado. Debido a esta manera de calcularlo y a que cuantifica el contraste, este parámetro es conocido como contraste RMS (del inglés *Root Mean Square*).

2.5.3 Modelos de color

Anteriormente se mencionó que un píxel puede tener varios niveles de intensidad dependiendo del número de bits de resolución utilizados. En este contexto los “niveles de intensidad”

se refieren a la intensidad de luz acromática o sin color que puede representar un píxel. Por ello, cada nivel de intensidad representaría un nivel de gris, creando así una escala de grises que constituye los diversos matices de intensidad en una imagen, donde en el límite inferior se encontraría el negro y en el superior el blanco.

Las imágenes digitales también son capaces de representar la luz cromática obteniendo así imágenes digitales a color. Para comprender este tipo de imágenes es conveniente mencionar que la luz visible es una parte del espectro electromagnético, con longitud de onda que abarcan desde los 400 nm hasta los 800 nm, aproximadamente. Esto es debido a que biológicamente la retina del ojo humano tiene alrededor de 7 millones de células sensibles a la luz conocidas como conos, las cuales son principalmente de tres tipos, donde cada uno es sensible a una longitud de onda específica, correspondiente a los colores rojo, verde y azul, con longitudes de onda aproximadas de 565 nm, 535 nm y 445 nm, respectivamente [40]. Por esta razón, a estos tres colores se les conoce como primarios y, gracias a las propiedades aditivas de la luz, con la combinación de éstos el ojo humano es capaz de ver todos los colores de la luz visible.

Existe una organización que se dedica a la investigación y normalización de la luz, la iluminación y el color llamada CIE (del francés *Commission Internationale de l'Eclairage*) que creó un diagrama cromático (ver figura 2.15) que representa los colores perceptibles por el ojo humano basado en un sistema de coordenadas, desarrollado también por ellos, llamado sistema CIE XYZ. Este diagrama es utilizado para representar, comparar y clasificar colores, así como definir los colores que se pueden utilizar en un estándar de color.

Los colores de una imagen digital pueden representarse de diferentes maneras, por ello se han creado numerosos modelos de color para cubrir las necesidades de representación y percepción, ya sea por los requerimientos de algún hardware como monitores o impresoras, o para aplicaciones en distintos campos como el procesamiento de imágenes, la fotografía, el diseño gráfico o la visión computacional.

Con el fin de estandarizar la especificación de los colores, los modelos de color definen

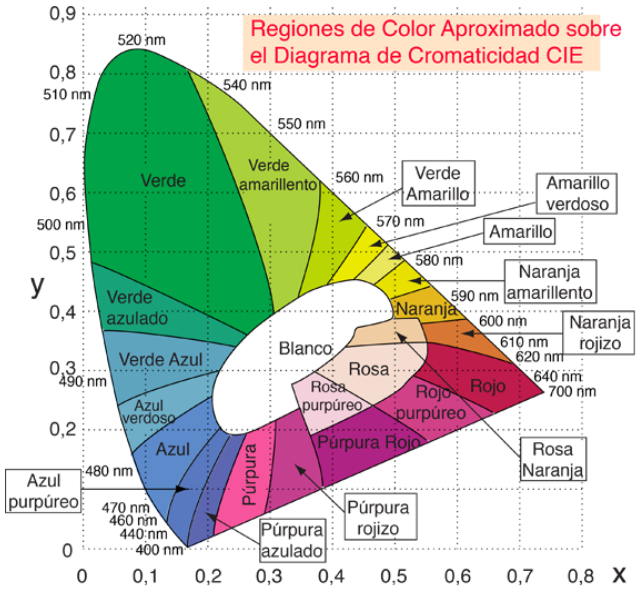


Figura 2.15: Diagrama cromático de CIE (tomada de [41]).

el sistema de coordenadas y los colores que pueden representar, de tal manera que no exista ambigüedad, o sea que cada color esté representado por un solo punto en ese sistema de coordenadas. Los modelos de color definen los espacios de color, que son el conjunto de colores que se pueden representar mediante ese modelo.

Actualmente existen muchos modelos de color, entre los más conocidos se encuentran el RGB, CYMK, YCbCr, HSV, HSI, HSV, CIE XYZ y CIELAB. Como es de esperarse, cada uno de ellos presenta ventajas y desventajas con respecto a los demás. Debido a que la teoría de cada modelo puede ser muy extensa, solo se dará una breve descripción de los modelos utilizados en este trabajo, los cuales fueron el RGB [42], YCbCr [43], HSI y CIELAB [44].

Modelo de color RGB

El modelo RGB (por su acrónimo en ingles de los colores rojo, verde y azul: *Red*, *Green*, *Blue*) es uno de los más utilizados a tal grado que la mayoría de las imágenes digitales se crean por defecto con este modelo. Su sistema de coordenadas es un sistema cartesiano con tres

componentes, donde cada uno es un color primario. La representación de los colores de este modelo se puede observar en el cubo unitario mostrado en la figura 2.16.

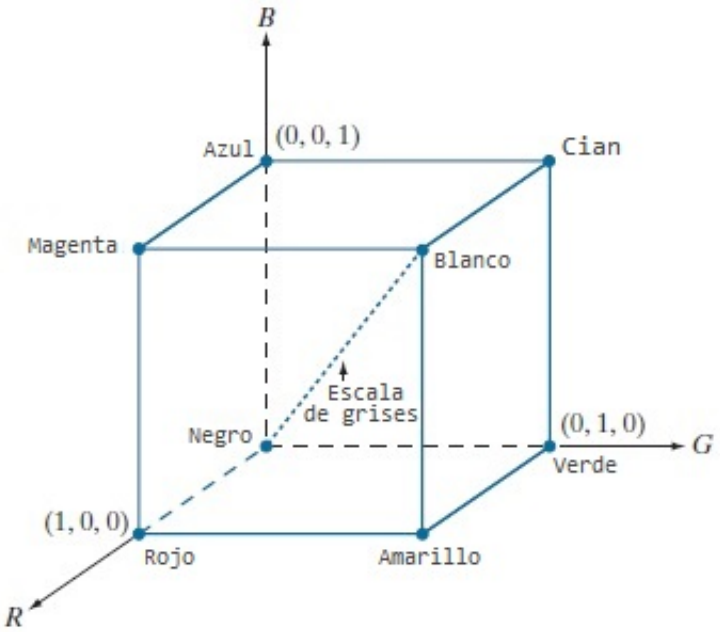


Figura 2.16: Cubo unitario RGB (tomada de [39]).

Como se observa en la figura 2.16, en este modelo también se puede representar la escala de grises, donde los valores de los tres colores es igual y se representa con la línea recta que va desde la coordenada $(0, 0, 0)$ hasta la $(1, 1, 1)$, es decir desde el el color negro hasta el blanco.

El modelo RGB está basado en la síntesis aditiva de la luz, la cual en pocas palabras, indica que combinando diferentes intensidades de los colores primarios de la luz (rojo, verde y azul) se puede obtener cualquier color del espectro visible. Por ello, las imágenes digitales representadas por este modelo tienen tres componentes (conocidos como canales), uno por cada color primario. Consecuentemente, cada píxel de estas imágenes contiene los niveles de intensidad de las tres componentes. En la figura 2.17, se muestra una representación de los 3 canales que conforman una imagen con el modelo RGB.

Como se observa en la figura anterior, el píxel se puede representar como un vector con los 3 valores de cada componente o canal.

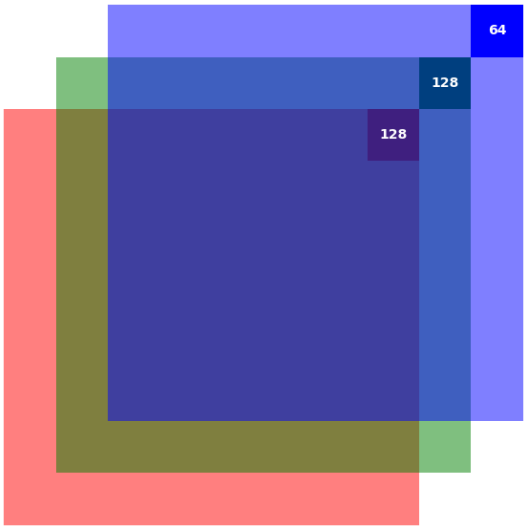


Figura 2.17: Componentes Roja, Verde y Azul de una imagen representada con el modelo RGB.

A partir de este modelo se creó el estándar conocido como sRGB (estándar RGB) con el objetivo de que diferentes dispositivos pudieran ser consistentes al reproducir los colores. Este estándar determina que por cada canal se utilizarán 8 bits de profundidad por lo que cada uno tendrá 256 niveles de intensidad.

Modelo de color YCbCr

Al igual que en el modelo anterior, en el modelo conocido como "YCbCr" también se utilizan tres componentes, pero en este modelo la información del brillo de la imagen se separa de la información del color, específicamente se separa la luminancia de la crominancia ya que se utiliza comúnmente para sistemas de transmisión de video. Sus componentes son la luminancia Y, crominancia en azul C_b y crominancia en rojo C_r.

La componente Y de este modelo se calcula con la suma ponderada de las componentes del modelo RGB, tal como se muestra en la siguiente ecuación (2.34).

$$Y = 0.299R + 0.587G + 0.114B \tag{2.34}$$

Como se muestra en (2.34), a la componente verde del RGB se le otorga un mayor peso. Esto se

debe a que, la CIE define el espacio de color YCbCr como una forma de representar los colores que se basa en la sensibilidad del ojo humano, donde la sensibilidad para percibir el verde es mayor a la del rojo y azul [45].

Las componentes C_b y C_r se calculan con las ecuaciones (2.35) y (2.36), respectivamente.

$$C_b = 128 - 0.169R - 0.331G + 0.500B \tag{2.35}$$

$$C_r = 128 + 0.500R - 0.419G - 0.081B \tag{2.36}$$

Modelo de color HSI

El modelo HSI nació de la idea de crear un modelo de color basado en la interpretación humana del color. De acuerdo con [46], los humanos describimos el color de acuerdo con su tono (*H*), su saturación (*S*) y su intensidad (*I*) o luminosidad, los cuales son los tres componentes de este modelo. Al igual que en el modelo anterior, la información del color (tono y saturación) se separa de la información de la iluminación.

El tono o matiz (*hue* en inglés) representa la longitud de onda dominante de la luz que observamos, o sea, el color puro como el rojo, azul, amarillo, etc. La saturación se refiere a qué tan puro o intenso es el color, mientras menos puro sea se puede decir que el color se va combinando con el blanco. Finalmente, la intensidad representa la luminosidad del color, es decir la cantidad de luz que emite.

En el modelo HSI se utilizan coordenadas cilíndricas para representar los colores. El ángulo representa el tono, por lo que puede tomar valores desde $0°$ hasta $360°$, donde los colores primarios del RGB están separados por $120°$ y entre cada uno de ellos, a $60°$, se encuentran los colores secundarios amarillo, cían y magenta. La saturación se representa con la distancia radial, donde los valores más bajos o cerca del eje serán colores con poca saturación, mientras que los valores más altos corresponderán a colores más saturados. La intensidad está representada con

la altura del cilindro y se mide desde la base (negro) hasta la parte superior (blanco). Esto se ejemplifica de mejor manera con la figura 2.18.

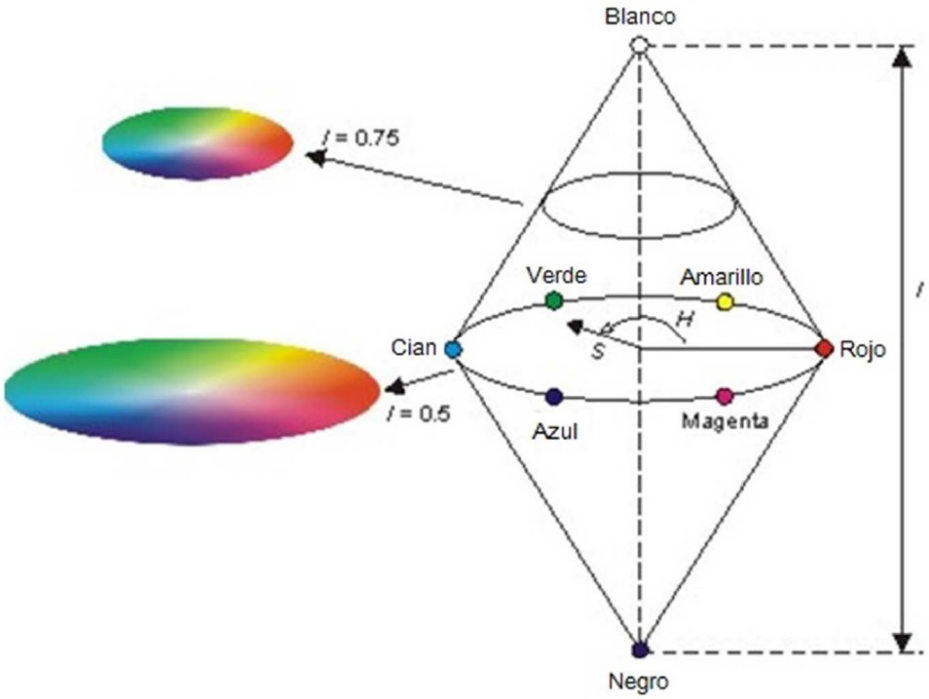


Figura 2.18: Representación de los colores en el modelo de color HSI (tomada de [47]).

La conversión de una imagen con modelo RGB al HSI, se realiza por medio de las ecuaciones (2.37) y (2.38) para obtener la componente *H*, (2.39) para la componente *S* y (2.40) para *I*.

$$H = \begin{cases} \theta & B \leq G \\ 360 - \theta & B > G \end{cases} \tag{2.37}$$

Donde:

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{\frac{1}{2}}}\right\} \tag{2.38}$$

$$S = 1 - \frac{3}{(R+G+B)}[\text{mín}(R,G,B)] \tag{2.39}$$

$$I = \frac{1}{3}(R+G+B) \tag{2.40}$$

Debido a que los colores RGB en el modelo HSI están separados por $120°$ la conversión de una imagen con el modelo HSI hacia el modelo RGB dependerá del ángulo que tenga la componente

H. Para el caso donde $0° \leq H < 120°$, conocida como la región RG, la conversión se realiza por medio de las ecuaciones (2.41), (2.42) y (2.43) para las componentes *R*, *B* y *G*, respectivamente.

$$R = I\left[1 + \frac{S\cos H}{\cos(60 - H)}\right] \tag{2.41}$$

$$B = I(1 - S) \tag{2.42}$$

$$G = 3I - (R + B) \tag{2.43}$$

Cuando la componente H se encuentra en la región GB, es decir $120° \leq H < 240°$, es necesario sustraer $120°$ al propio valor de H. Posteriormente las componentes RGB se realizarán mediante las ecuaciones (2.44), (2.45) y (2.46).

$$R = I(1 - S) \tag{2.44}$$

$$G = I\left[1 + \frac{S\cos H}{\cos(60 - H)}\right] \tag{2.45}$$

$$B = 3I - (R + G) \tag{2.46}$$

Donde $H = H - 120$ en las tres ecuaciones anteriores.

Finalmente, en la región BR (cuando $240° \leq H < 360°$) se utilizan las ecuaciones (2.47), (2.48) y (2.49), restando $240°$ al valor de H.

$$R = 3I - (G + B) \tag{2.47}$$

$$G = I(1 - S) \tag{2.48}$$

$$B = I\left[1 + \frac{S\cos H}{\cos(60 - H)}\right] \tag{2.49}$$

Donde $H = H - 240$ en las tres ecuaciones anteriores [48].

Modelo de color CIELAB

El modelo CIE 1976 L*a*b*, mejor conocido como CIELAB, fue desarrollado en 1976 por la CIE con el objetivo de tener un modelo independiente del hardware, es decir, que los colores

puedan ser consistentes sin importar el dispositivo utilizado, ya que en muchos otros modelos (incluyendo RGB, YCbCr y HSI) la percepción del color suele variar significativamente entre dispositivos si estos no estaban correctamente calibrados.

CIELAB se basa en el modelo CIE XYZ, utilizado para definir los colores que percibe el ojo humano. Por consiguiente, CIELAB pretende modelar la percepción humana del color por medido de tres componentes: L^*, a^* y b^*. La primera representa la percepción de la intensidad de la luz, o sea la luminosidad del color y las otras dos representan la crominancia del color por medio de coordenadas.

La componente a^* representa la posición del color en un eje relativo con los colores opuestos rojo-verde, con valores positivos para los colores rojos y negativos para los colores verdes; de manera similar, b^* indica la posición del color en un eje relativo con los colores opuestos azul-amarillo, con valores positivos para los colores amarillos y negativos para los colores azules.

Este modelo es considerado perceptivamente uniforme, ya que pretende que los cambios en las coordenadas correspondan de manera similar a los cambios de la percepción humana del color [48].

Para la conversión del modelo RGB a CIELAB es necesario realizar primero la conversión de RGB a CIE XYZ. Debido a que este procedimiento contempla una teoría relativamente extensa y varía dependiendo de los estándares utilizados, es este trabajo solo se mostrará la conversión utilizada en la parte experimental, la cual implica obtener las coordenadas X, Y y Z (conocidos como valores triestímulos) por medio de la matriz de ecuación (2.50).

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.50}$$

Donde X, Y y Z son las coordenadas de color XYZ y R, G y B son las componentes de color RGB.

Con el cálculo de estos valores es posible realizar la conversión de las coordenadas XYZ a

los valores L^*, a^* y b^* por medio de las ecuaciones (2.51), (2.52) y (2.53), respectivamente.

$$L^* = 116 \cdot h\left(\frac{Y}{Y_{ref}}\right) - 16 \tag{2.51}$$

$$a^* = 500\left[h\left(\frac{X}{X_{ref}}\right) - h\left(\frac{Y}{Y_{ref}}\right)\right] \tag{2.52}$$

$$b^* = 200\left[h\left(\frac{Y}{Y_{ref}}\right) - h\left(\frac{Z}{Z_{ref}}\right)\right] \tag{2.53}$$

Donde X_{ref}, Y_{ref} y Z_{ref} son valores triestímulo de referencia y $h(\chi)$ es una función que permite que los cambios en las coordenadas sean perceptualmente uniformes, la cual está dada por la ecuación (2.54).

$$h(\chi) = \begin{cases} \sqrt[3]{\chi} & \chi > 0.008856 \\ 7.787\chi + \frac{16}{116} & \chi \leq 0.008856 \end{cases} \tag{2.54}$$

Para los valores triestímulo de referencia X_{ref}, Y_{ref} y Z_{ref} de las ecuaciones (2.51), (2.52) y (2.53), se utilizaron los valores de "observador estándar bajo condiciones de iluminación estándar", definidos por la CIE en el estándar D65 [49]: $X_{ref} = 95.047$, $Y_{ref} = 100$ y $Z_{ref} = 108.883$.

2.5.4 Procesamiento de imágenes digitales

El procesamiento de imágenes se refiere a la aplicación de técnicas y algoritmos para mejorar su calidad o facilitar su análisis e interpretación. Bajo esta premisa es posible afirmar que el procesamiento de imágenes se convierte en digital en el momento en que las computadoras tuvieron la capacidad para realizarlo.

Aunque por definición no está considerado como un procesamiento digital, en la década de 1920, con el desarrollo del sistema de transmisión de imágenes Bartlane, se realizaron las primeras manipulaciones de imágenes digitales al codificar los niveles de gris para poder transmitirlas. Con esta codificación, pronto surgió la necesidad de mejorar la calidad de dichas imágenes, que en ese momento significó aumentar los niveles de gris.

El primer registro sobre procesamiento de imágenes digitales data de la década de 1960 con los programas espaciales, donde corrigieron por computadora algunas distorsiones de las imágenes transmitidas de la luna; las técnicas utilizadas durante estos programas sentaron las bases para el desarrollo de nuevos métodos. Una década más tarde, con la invención de la tomografía axial computarizada, el procesamiento de imágenes digitales no solo tuvo un gran avance por ser capaz de construir una imagen a través de datos detectados por los sensores y por crear un método de mejora de contraste, sino que también porque se comenzó a utilizar en la medicina y esto dio pauta a que pudiera ser utilizada en otros ámbitos.

Actualmente, el procesamiento de imágenes se utiliza principalmente para tres aplicaciones: mejorar su calidad para que la interpretación de la información que ella contiene sea más fácil de observar para un humano, para extraer información de ellas y para su almacenamiento y transmisión.

Como se ha mencionado anteriormente una imagen digital está definida como una matriz de píxeles, por lo tanto, el procesamiento implica la manipulación de los píxeles y principalmente se realiza modificando su intensidad ya sea utilizando funciones o por medio de filtros.

Funciones de transformación de intensidad

En las técnicas de procesamientos más simples se utilizan funciones para modificar la intensidad, las cuales comúnmente se representan como se muestra en la ecuación (2.55), tomando en cuenta imágenes acromáticas o de un solo canal.

$$g(x, y) = T\left[f(x, y)\right] \tag{2.55}$$

Donde $g(x, y)$ representa la imagen obtenida al aplicar la función T en la imagen $f(x, y)$. De esta manera, los niveles de intensidad ι de la imagen resultante $g(x, y)$ estarán en función de los de la imagen original, y se puede representar por medio de la ecuación (2.56).

$$\iota = T(i) \tag{2.56}$$

Para representar estas funciones es común graficar los niveles de intensidad de entrada contra los de salida, recordando que pueden tomar valores dentro del rango dinámico del canal $[0, L-1]$, tal como se muestra en la figura 2.19.

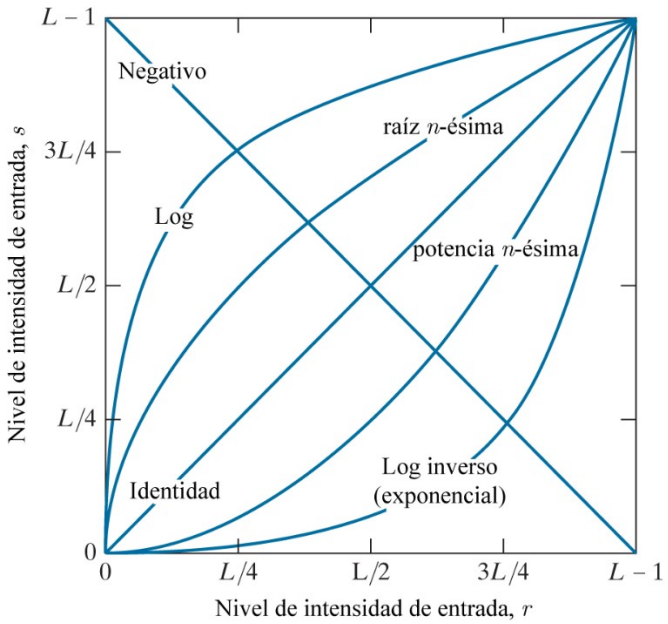


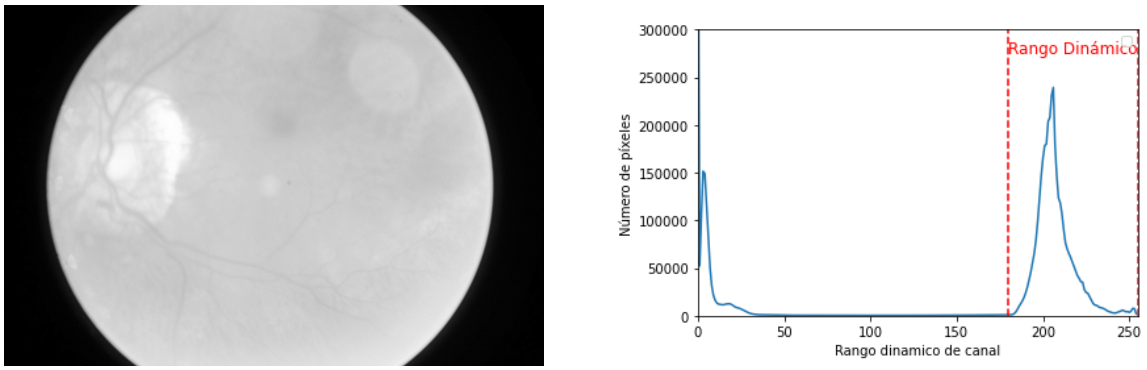
Figura 2.19: Niveles de intensidad de entrada vs salida (tomada de [50].

Las curvas de la figura anterior muestran las funciones más básicas que suelen utilizarse en este tipo de procesamiento de imagen, las cuales son la negativa, logarítmica, identidad y de potencia. Entre ellas destacan las transformaciones logarítmicas y las de potencia, las primeras son utilizadas para expandir el rango de valores de los píxeles oscuros y comprimir los más claros (logarítmica) o viceversa (logarítmica inversa); y los de potencia, también conocidas como corrección gamma, para modificar la iluminación de las imágenes.

Mejora de contraste

Como se mencionó anteriormente, el contraste está determinado por el rango dinámico de la imagen de tal manera que, si éste es estrecho, una gran cantidad de píxeles tendrán un mismo valor de intensidad y los demás tendrán valores muy similares. Esto indicaría que la imagen

digital presenta un bajo contraste y en su histograma se observaría una distribución angosta del rango dinámico, tal como se observa en la figura 2.20, donde la figura 2.20a muestra la imagen con bajo contraste y la figura 2.20b su histograma.



(a) Imagen con bajo contraste. (b) Histograma de la imagen.

Figura 2.20: Histograma de imagen con bajo contraste.

La figura 2.20a muestra una imagen de fondo de ojo con bajo contraste, ya que se puede observar prácticamente solo una figura redonda de color blanco y prácticamente no se perciben los detalles. Su histograma muestra una distribución del rango dinámico considerablemente angosta, abarcando menos del 30 % del rango dinámico del canal. Un detalle a considerar es que se observa una leve distribución en los valores más bajos, los cuales corresponden al fondo negro de la imagen.

Debido a que el contraste es una característica fundamental de las imágenes, ya que permite resaltar los detalles visuales y crear una sensación de profundidad, mejorar o corregir el contraste de las imágenes ha sido un área de investigación muy activa desde los inicios del procesamiento de imagen.

La mejora del contraste se puede definir como el ensanchamiento del rango dinámico de una imagen digital, con esto se lograrían tener una distribución de niveles de intensidad más uniforme al incrementar la separación entre los valores de píxeles más bajos y altos en el histograma de la imagen.

Actualmente se han desarrollado numerosos métodos de mejora de contraste, entre los cua-

les la ecualización de histograma, junto con sus múltiples variantes, han sido de los algoritmos más básicos y utilizados. Sin embargo, no suele ser ideal en todas las situaciones, ya que pueden provocar alteraciones en la imagen original, no preservar adecuadamente la apariencia original, sobre-mejorar el contraste o suelen tener deficiencias de contraste o de brillo [51]. Para solucionar eso, se han propuesto métodos más avanzados, como técnicas basadas en transformaciones locales y algoritmos adaptativos. También existen métodos con enfoques basados en la percepción humana del contraste utilizando herramientas como la lógica difusa, como es el caso de este trabajo.

La ecualización de histograma fue uno de los primeros métodos propuestos y sigue siendo ampliamente utilizado. Su algoritmo consiste en redistribuir los valores de los píxeles de forma que todos los niveles de intensidad tengan la misma probabilidad de aparición. Si bien su teoría completa es extensa, se puede resumir definiéndola como una función que mapea una imagen en todo el rango dinámico, utilizando la función de densidad acumulada como función de transformación [52]. Como se menciona en la misma literatura, el algoritmo para la ecualización de histograma se enlista a continuación:

1. Calcular el histograma por medio de la ecuación (2.28) para obtener la función de densidad de probabilidad de la imagen, por medio de la ecuación (2.57).

$$p(i_k) = \frac{h(i_k)}{n} \tag{2.57}$$

   Donde n representa el número total de píxeles de la imagen, recordando que i_k representa el nivel de intensidad que puede tener un píxel, por lo que $k = 0, 1, 2, ..., L-1$.

2. Obtener la función de densidad acumulada con la ecuación (2.58) para encontrar la suma acumulada de los valores del histograma.

$$c(i_k) = \sum_{k=0}^{L-1} p(i_k) \tag{2.58}$$

3. Utilizar la función de ecualización de histograma definida por la ecuación (2.59).

$$h(i_k) = i_0 + (i_{L-1} - i_0)c(i_k) \tag{2.59}$$

Esta función, en pocas palabras, multiplica los valores obtenidos con la función de densidad por el máximo nivel de intensidad.

4. Mapear el resultado de la función anterior a la imagen de salida con una correspondencia uno a uno, para obtener la imagen con el histograma ecualizado.

La ecualización de histograma es una técnica eficaz para mejorar el contraste de las imágenes, sin embargo, como se mencionó anteriormente, puede causar sobre mejoramiento de contraste e, incluso en ciertos casos, puede reducir el contraste de las imágenes [53]. Entre otros problemas que puede causar este método, es que puede distorsionar la apariencia de imagen y causar bordes irregulares o halos alrededor de los objetos de la misma.

Para solucionar estas desventajas, se han desarrollado numerosas variantes de este algoritmo, donde se divide la imagen en dos o varios sub-bloques para aplicar la ecualización de histograma a cada bloque. Muchos de estas variantes se mencionan en [51] y [53].

Filtros espaciales

Aparte del procesamiento básico de imágenes, existen otro tipo de técnicas que utilizan diferentes características de las imágenes para procesarlas. Entre las más comunes están aquellas donde se aplican filtros, ya sea en el dominio espacial, donde la imagen se representa por sus valores de intensidad en ubicaciones específicas (matrices de píxeles), o en el dominio de la frecuencia, donde se utiliza la transformada de Fourier para representar a la imagen por las frecuencias presentes en ella y la cantidad de energía asociada con cada frecuencia. En esta sección se abordarán solamente los filtros espaciales, ya que son la base de las redes neuronales convolucionales.

El objetivo de los filtros espaciales es modificar las propiedades locales de una imagen cambiando los valores de los píxeles por medio de un filtro conocido como kernel, máscara o

ventana, que es un tensor de valores, comúnmente matriz o vector, que aplica una operación para transformar la imagen.

Aunque los kernels pueden ser tensores de cualquier tamaño y forma, comúnmente son matrices cuadradas cuyo tamaño determina el área de la imagen a la cual se le aplicará la operación. Esta operación consiste en una suma de productos entre la imagen $f(x, y)$ de tamaño $M \times N$ y el kernel $w(x, y)$ de tamaño $m \times n$ donde $m = 2a + 1$ y $n = 2b + 1$, que producen la imagen filtrada $g(x, y)$, la cual estará dada por la ecuación (2.60).

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t) \tag{2.60}$$

Donde x y y representan la posición de la imagen resultante después de aplicar la operación, s y t representan las posiciones relativas dentro del kernel y $f(x + s, y + t)$ el valor del píxel en la imagen de entrada en la posición desplazada $(x + s, y + t)$. Es importante mencionar que a y b deben ser enteros no negativos para que se garantice que el kernel tenga un valor central.

La ecuación (2.60) indica que el kernel actúa como una ventana cuyo centro $w(0, 0)$ se desplaza por la imagen original píxel por píxel, y en cada posición se realiza la suma de productos. Para que el centro del filtro pueda recorrer todos los píxeles, comúnmente es necesario agregar una o varias filas y columnas de ceros en los bordes de la imagen para que cuando el tamaño del kernel sobrepase los bordes no se tenga un espacio indefinido, sino ceros que permitan realizar la operación. A este relleno de ceros en los bordes de la imagen se le conoce por su nombre en inglés: *padding*.

El *padding* evita la perdida de información en los bordes de la imagen y, por lo mismo, controla el tamaño de la imagen de salida, ya que, si no se agregan los ceros, la imagen resultante sería más pequeña. El número de filas y columnas de ceros que sería necesario agregar a los bordes de una imagen dependerá del tamaño del kernel. Se requerirá un mínimo de $(m - 1)/2$ filas de ceros en el borde superior e inferior de la imagen y un mínimo de $(n - 1)/2$ columnas de ceros en ambos bordes laterales.

Convolución

La "suma de productos" que se realiza por medio del kernel a la imagen original describe dos posibles operaciones: la correlación o la convolución. Sin entrar en detalle, matemáticamente ambas involucran la combinación de dos funciones y sus fórmulas son similares, con la única diferencia de que en la convolución se invierte una de las funciones. Esta diferencia permite que las aplicaciones de ambas operaciones sean distintas, la correlación genera una tercera función que permite medir la similitud de las funciones y la convolución genera una tercera función con la combinación de las amplitudes de las dos funciones.

En el contexto de las imágenes digitales, la diferencia entre ambas operaciones es que en la convolución el kernel se rota $180°$ antes de realizar la suma de productos. De esta manera, la modificación de la ecuación (2.60) para representar la convolución entre una imagen $f(x, y)$ y un kernel $w(x, y)$ estaría en un cambio de signo de s y t, tal como se muestra en la ecuación (2.61)

$$(w * f)(x, y = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x - s, y - t) \tag{2.61}$$

En la figura 2.21 se muestra una representación de la convolución de una imagen de 5x5 píxeles con un kernel de 3x3.

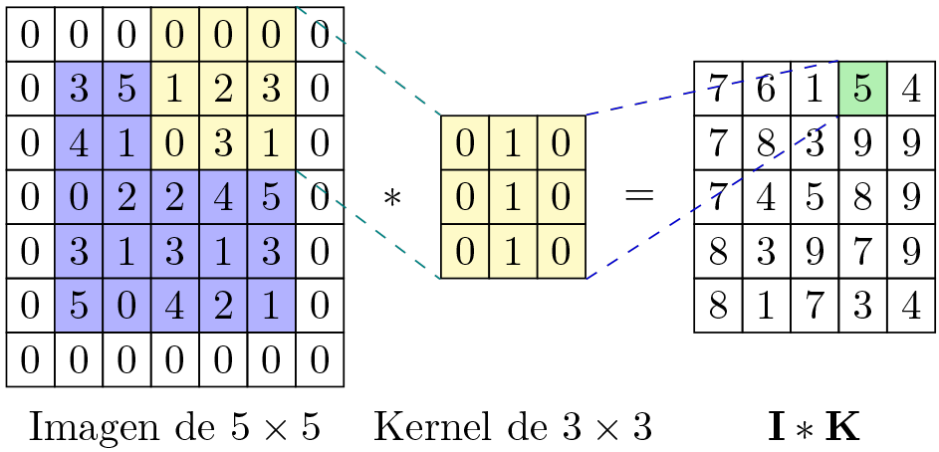


Figura 2.21: Convolución de una imagen de 5×5 píxeles y *padding* de 1 con un kernel de 3×3 píxeles.

En la figura 2.21 también se ejemplifica el *padding* agregado a la imagen para que el filtro pueda recorrer los bordes de la imagen sin perder información, por lo que la imagen resultante será del mismo tamaño que la original.

2.6 Redes Neuronales Convolucionales

Para los humanos, la detección de objetos en el proceso de la visión es una habilidad que resulta sencilla, un ejemplo se tiene con la detección de rostros y es gracias a que ciertas regiones de la corteza cerebral están especializada en esta tarea [54]. Sin embargo, como se mencionó anteriormente, esta misma tarea resultaría demasiado compleja para una computadora utilizando una computación clásica e incluso con una red neuronal artificial convencional, como las que se mencionaron anteriormente en este mismo capítulo.

A pesar de que las ANN son eficaces en tareas de clasificación y reconocimiento de patrones, están diseñadas para manejar datos en forma de vectores que no tienen una relación secuencial. Para poder procesar una imagen con una ANN sería necesario vectorizar las imágenes, es decir, se tendría que convertir en una lista de píxeles. Esto implicaría perder su información espacial, o sea la información sobre la posición de cada píxel en la imagen original.

Esto representa una limitación al procesar la información de las imágenes ya que dicha información presenta una estructura espacial que puede ser bidimensional (para las imágenes de un solo canal como en escala de grises) o multidimensional (como las imágenes en RGB u otro modelo de color multicanal), donde la organización de los píxeles influye en la información que se puede extraer de las imágenes.

Otro problema de la vectorización de una imagen es que las redes necesitarían un número muy grande de neuronas en la capa de entrada y de pesos por cada neurona. Suponiendo que una imagen de entrada sea de 32×32 píxeles, tan solo sería necesario entrenar 1024 pesos y un

sesgo por cada neurona en la capa de entrada.

Para solucionar esos problemas se desarrolló un tipo de red neuronal que utiliza el concepto de la convolución para poder procesar la información de las imágenes, conocidas como redes neuronales convolucionales o CNN. En estas redes, al igual que las de tipo *feedforward*, la información se propaga hacia adelante por todas sus capas y, aunque en un principio se diseñaron para trabajar con imágenes, actualmente son capaces de trabajar con tensores prácticamente de cualquier tamaño, por lo que también son utilizadas para el reconocimiento de voz y para el procesamiento de lenguaje natural, de audio o de señales fisiológicas como las cardiacas o del cerebro, entre otras.

2.6.1 Historia

Es posible decir que las bases para el desarrollo de este tipo de redes se sentaron desde el desarrollo de las redes multicapa. Sin embargo, su distintivo se encuentra en su arquitectura, ya que está inspirada en el sistema nervioso visual de los vertebrados, cuyo funcionamiento fue descrito por David Hubel y Torsten Wieses con sus diversas investigaciones entre las décadas de 1950 y 1960 [55].

Sin entrar en detalles, los puntos clave del proceso de visión descrito por Hubel y Wieses que se utilizaron para el desarrollo de las CNN son la siguientes.

1. Al observar una imagen se genera una excitación en las células de los ojos y mandan una señal eléctrica a la corteza visual, donde tenemos una red neuronal que se especializa en analizar diferentes características de la imagen que estamos observando.

2. Existen varios grupos de neuronas y cada grupo se enfoca en un campo receptivo local. Esto significa que las neuronas no miran en detalle cada elemento de la imagen, sino que varios grupos de neuronas se enfocan en pequeñas regiones diferentes, al que llamaron

"campos receptivos locales". En cada campo se extraen algunas características relevantes que permiten reconocer parte del contenido de esas imágenes.

3. Existen diferentes tipos de neuronas y cada tipo es sensible a algún estímulo particular, como neuronas de color, profundidad y movimiento. Entre ellas también se encuentran neuronas simples (que responden a estímulos simples como líneas con alguna orientación específica) y complejas (que responden a la presencia de esquinas o curvas).

4. Las neuronas de las primeras capas solo reaccionan a formas muy básicas y, a medida que se profundiza en la red, las siguientes neuronas son capaces de utilizar la información de las capas de anteriores para combinarlas e interpretar formas más complejas. Dicho de otra manera, cada capa se va especializando en detectar formas con mayor complejidad hasta el punto donde el cerebro ya es capaz de determinar que en la imagen que estamos observando hay objetos determinados.

Aunque desde la década de 1960 se propusieron varios modelos neuronales que utilizaban una estructura jerárquica, tal como el perceptrón multicapa, el modelo propuesto por Kunihiko Fukushima en la década de 1980, conocido como Neocognitrón [56] fue el primero en hacerlo eficientemente para el procesamiento de imágenes (específicamente en el reconocimiento de patrones) utilizando capas de neuronas simples y otras de neuronas complejas, como las descritas por Hubel y Wieses.

Basándose en el Neocognitrón, entre las décadas de 1980 y 1990, Yann LeCun desarrolló la que es considerada como la primera red neuronal convolucional, a la cual llamó LeNet [57]. Aunque estrictamente no fue la primera red que utilizó la operación de la convolución, su arquitectura era más simple y eficiente en comparación con sus predecesoras y logró darle una aplicación real (fue utilizada para la detección de números en cheques bancarios), por lo cual esta red sentó las bases de las CNN actuales, por ello es considerada como la primera CNN.

Aunque con el desarrollo de LeNet se tuvo el planteamiento teórico para el funcionamiento de las CNN, el desarrollo de este tipo de redes no experimentó un crecimiento significativo

tanto por la falta de datos suficientes para entrenar los modelos y ponerlos a prueba como por las limitaciones computacionales. A partir de la década de 2010, con el boom de la cantidad de datos disponibles, la capacidad de procesamiento de las computadoras y el desarrollo de GPUs potentes, comenzaron a surgir nuevas arquitecturas que demostraron ser eficientes, entre las más conocidas se encuentran AlexNet [58], VGGNet [59], ResNet [60], RedNet [61] y GoogLeNet [62].

De las redes antes mencionadas, GoogLeNet introdujo cambios significativos en la arquitectura común de una CNN —la cual se mostrará más adelante en este capítulo— que le permitió ser más ligera en tamaño, más rápida en la fase de entrenamiento y, aún así, tener un mejor desempeño comparado con redes antecesoras [63]. Gracias a su buen desempeño, GoogLeNet ha sido utilizada en numerosas aplicaciones, además su arquitectura ha sido la base de otras redes neuronales más actuales, como el caso de la red utilizada en esta tesis.

2.6.2 Arquitectura y funcionamiento de una CNN

Las CNN se inspiran en el funcionamiento del sistema visual humano en el sentido que extraen características cada vez más complejas a medida que la información avanza por las capas de la red. En las primeras capas, se detectan patrones básicos como bordes, texturas y colores, los cuales se van combinando en las capas posteriores hasta que la red es capaz de identificar objetos complejos. Su eficiencia radica en su capacidad de procesar la información espacial de los datos de entrada utilizando la operación matemática de convolución, lo que le da el nombre característico a este tipo de redes.

En general, las CNN comparten la misma estructura básica de las convencionales ya que ambas están compuestas por una capa de entrada, un determinado número de capas en la capa oculta y una capa de salida. La deferencia entre ambas radica en las operaciones que se realizan en cada capa, ya que las convolucionales, en lugar de conectar neuronas simples, utilizan operaciones de convolución y submuestreo para extraer la información espacial de los datos de

entrada que, a diferencia de las redes convencionales donde utilizan vectores, comúnmente son tensores como imágenes de un solo canal o multicanal. La arquitectura básica de una CNN se muestra en la figura 2.22.

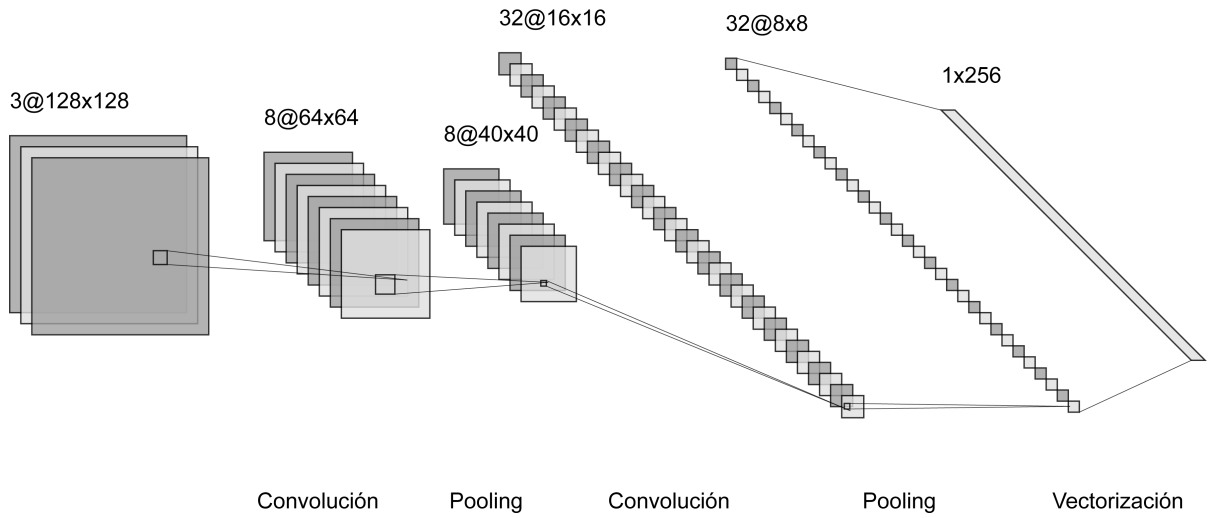


Figura 2.22: Arquitectura básica de una CNN.

Como se observa en la figura 2.22, en las CNN se utilizan varias capas convolucionales y de submuestreo con el objetivo de compactar lo más posible el tamaño de las imágenes extrayendo la mayor cantidad de características. En las capas convolucionales se aplican los filtros convolucionales por medio de kernels que se van desplazando a lo largo de toda la imagen para identificar características específicas, dependiendo de los parámetros de cada kernel, obteniendo como resultado una imagen de menor dimensión, pero mayor profundidad, conocida como mapa de características. Después de cada capa convolucional se tiene una capa de submuestreo, mejor conocida por su nombre en inglés *pooling*, la cual tiene el objetivo de reducir el tamaño del mapa de características preservando la información más relevante.

En otras palabras, la convolución permite que la red se enfoque en partes locales de la imagen para extraer solo la información relevante, reduciendo la complejidad computacional en comparación con una red convencional y el submuestreo permite reducir progresivamente la resolución espacial de las características extraídas mediante la convolución.

Se espera que en la última capa convolucional y de *pooling* se tenga un número muy pequeño

de datos a comparación de la imagen de entrada, pero que la información sea muy relevante, representada con un gran número de mapas de características de pequeñas dimensiones apiladas una tras otra como un volumen como se muestra en la misma figura 2.22. Una vez que se tiene la información representada de esta manera, se vectoriza y pasa por una red *fully connected*, la cual será la encargada se generar la predicción.

Capas convolucionales

Al igual que en el procesamiento de imágenes, en las CNN la convolución se realiza por medio de un kernel que ejecuta la suma de productos a la fracción de la imagen que cubre. De igual manera, para evitar la pérdida de información en los bordes de la imagen, en las CNN se suele agregar un *padding* dependiendo del tamaño del kernel, que comúnmente es cuadrado con dimensiones impares (3×3, 5×5, 7×7).

En cada iteración del entrenamiento, el kernel se desplaza cierta distancia (en píxeles) para para recorrer la imagen completa desde la esquina superior izquierda hasta la esquina inferior derecha. Este desplazamiento es conocido por su nombre en inglés *stride*.

Si bien, en el procesamiento de imágenes el desplazamiento comúnmente es de uno para que la salida del filtro genere una imagen con las mismas dimensiones que la entrada, en las CNN esto no es necesario, ya que un *stride* mayor a 1 permite reducir intencionalmente el tamaño del mapa de características obtenido, con el objetivo de que la cantidad de datos que serán procesados en las capas posteriores sea menor. El *stride* puede ser diferente para el desplazamiento horizontal y vertical, aunque en la práctica es común que sea igual en ambas direcciones.

Dicho esto, el tamaño del mapa de características obtenido por la convolución dependerá del tamaño del kernel, del *padding* y del *stride*, tal como se muestra en la ecuación (2.62).

$$\left\lfloor \frac{n_h + p_h + s_h - k_h}{s_h} \right\rfloor \times \left\lfloor \frac{n_w + p_w + s_w - k_w}{s_w} \right\rfloor \tag{2.62}$$

Donde n representa la dimensión de la imagen original, p el tamaño del *padding*, s el factor del desplazamiento y k el tamaño del kernel, a lo ancho h y alto w. Debido a que los argumentos pueden resultar en valores fraccionarios, se toma el entero positivo cercano más pequeño.

La ecuación (2.62) indica que mientras más grande sea el desplazamiento el tamaño del mapa de características será menor, sin embargo, es importante señalar que un valor muy grande puede generar que el kernel no sea capaz de cubrir toda la imagen, lo que implicaría pérdida de información.

Como se muestra en la figura 2.22, la capa de entrada consiste en una capa convolucional, la cual tiene el objetivo de detectar ciertos patrones de la imagen original. Los parámetros del kernel son los elementos que determinan las características que se extraen de la imagen durante la convolución. Estos parámetros actúan como un filtro que, al desplazarse por la imagen, responden a patrones específicos de píxeles dependiendo de la propia configuración del kernel. Por ejemplo, un kernel con un patrón vertical será "sensible" a líneas verticales en la imagen y lo plasmará en el mapa de características.

En el caso de las imágenes multicanal, los kernels deben tener el mismo número de canales de la imagen, por ejemplo, para las imágenes en RGB cada kernel de la primera capa convolucional deberá tener 3 canales, sin embargo, ya que los coeficientes de cada canal son distintos pueden tomarse como 3 kernels individuales, generando 3 mapas de características diferentes, pero del mismo tamaño. Como esos 3 mapas representan el mismo patrón de la imagen, se suman punto a punto para obtener un solo mapa de características que represente la combinación de los patrones detectados de cada canal.

Las CNN utilizan múltiples kernels para extraer una gran variedad de características de las imágenes. Cada uno se especializa en detectar un tipo específico de característica realizando la convolución individualmente, por lo tanto, cada kernel genera un mapa de características. En cada capa convolucional se concatenan todos los mapas de características a lo largo del canal, es decir, se apilan uno encima del otro generando un volumen con un número de canales igual

al número de kernels utilizados.

Al diseñar una CNN, es necesario definir el tamaño y la forma de los kernels, sin embargo, sus parámetros se obtienen automáticamente mediante el entrenamiento de la red. Por lo tanto, haciendo una analogía a las ANN convencionales, los parámetros de los kernels son el equivalente a los pesos de la red.

Capas de submuestreo (*pooling*)

El objetivo de las capas de submuestreo es reducir el tamaño de los mapas de características obtenidos en cada capa convolucional preservando la información más relevante. Para ello existen varias operaciones que se pueden realizar en estas capas, conocidas como operaciones de *pooling*. Su principio de funcionamiento es similar a la convolución, aunque más sencillo: se tiene una ventana que cubre cierta región de la imagen y se aplica la operación, la ventana recorre toda la imagen iterativamente para generar un mapa de características de menor dimensión.

Existen diferentes operaciones de *pooling*, entre las más conocidas se encuentran:

- ***Max pooling***. Simplemente se toma el máximo valor de la región de imagen que cubre la ventana. Éste es de los más utilizados ya que permite resaltar las características más dominantes del mapa de características.
- ***Mean pooling***. En esta operación se calcula el promedio de los valores de la región de la imagen seleccionada por la ventana.
- ***Min pooling***. Es similar a *max pooling*, con la diferencia que en éste se toma el valor mínimo de la región de la imagen.

En la CNN utilizada en esta tesis, se utilizaron capas de *max pooling*, ya que permite reducir el tamaño preservando la información más relevante. En la figura 2.23 se muestra una

ejemplificación de la operación de *max pooling* en una matriz.

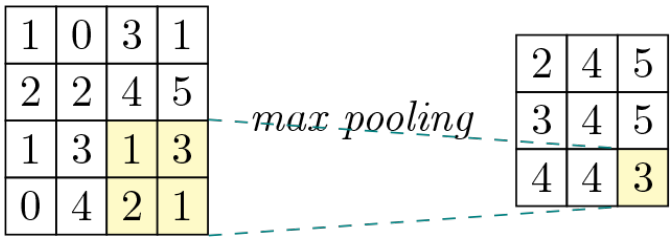


Figura 2.23: *Max pooling* con un filtro 2×2 y *stride* de 1.

A diferencia de las capas convolucionales, las ventanas en las capas de *pooling* no tienen parámetros, sino que definen la región del mapa de características que se va a reducir de dimensión aplicando la operación correspondiente. Lo que si es necesario determinar es el tamaño de las ventanas y el desplazamiento, ya que mientras ambos sean mayores, el grado de compactación también lo será. Para aplicar la operación de *pooling* a un volumen de características se realiza la operación por cada canal individualmente, generando en un volumen de menor cantidad de filas y columnas, pero con la misma profundidad.

Capa *fully connected*

La función de las capas convolucionales y de *pooling* es extraer las características relevantes de las imágenes, compactarlas y agruparlas. Sin embargo, para poder generar una predicción es necesario pasar esa información por una capa completamente conectada (mejor conocida por su nombre en inglés *fully connected*), que es similar a una red neuronal convencional.

Debido a que esta capa utiliza vectores como parámetro de entrada, es necesario vectorizar la salida de la última capa de pooling, la cual se espera que ya tenga dimensiones muy pequeñas con una profundidad de características muy grande, en otras palabras "ya casi vectorizada". Este vector de valores se utiliza como entrada a la capa *fully connected*, que activa las neuronas solo en las regiones donde detecta los patrones específicos que ha aprendido durante el entrenamiento. De esta manera, la red neuronal aprende a identificar y discriminar diferentes características visuales en la imagen.

2.7 Conclusiones del capítulo

En este capítulo se presentaron fundamentos teóricos que serán necesarios para el desarrollo experimental de esta tesis, donde se desarrollará un sistema de procesamiento de imágenes utilizando lógica difusa para mejorar el contraste de imágenes y una red neuronal convolucional para clasificarlas, implementado sobre el sistema embebido Raspberry Pi.

Como primer punto, se revisaron las características principales que posee la placa Raspberry Pi 4 modelo B y se compararon con las de su predecesor. Aunque ambos son capaces de ejecutar sistemas operativos completos y tienen implementado el lenguaje de programación Python, lo que los hace teóricamente capaces de ejecutar códigos utilizando estas dos técnicas mencionadas, la mayor capacidad y velocidad de procesamiento de la Raspberry Pi 4 modelo B y su mayor capacidad —y también velocidad— de memoria RAM (8 GB frente a 1 GB) lo hacen más apto para ejecutar el sistema a desarrollar, ya que se utilizará una red neuronal convolucional profunda.

Posteriormente se describió la lógica difusa para demostrar que esta técnica es capaz de modelar la vaguedad de un lenguaje natural para crear un sistema de inferencia difuso que se adapte a las necesidades específicas de un problema determinado. También se dio una descripción de las redes neuronales artificiales, donde se mencionó que la función de una neurona simplemente es realizar una suma de productos y pasarla por una función de activación, sin embargo, el trabajo paralelo de un gran número de neuronas organizadas en capas, permite realizar tareas complejas como reconocimiento de patrones, clasificación y regresión de una manera efectiva, tareas que serían prácticamente imposibles utilizando una computación clásica.

Ya que las redes neuronales convolucionales tratan con imágenes, previo a su descripción se definió lo que es una imagen digital y lo que implica el procesamiento de imágenes, mencionando también cómo es que se representa una imagen digital.

Como punto final se proporcionó la base teórica de las redes neuronales convolucionales, las cuales son capaces de preservar la información espacial de los datos de entrada, que comúnmente son imágenes (como en el caso de este trabajo). Este tipo de redes utilizan la operación matemática de la "convolución" entre la imagen y varios filtros que recorren la imagen completa para extraer patrones de la imagen. Las CNN, aparte de las redes convencionales, están inspiradas en el proceso de visión del ser humano, donde a medida que la información va pasando por cada capa se va extrayendo información más compleja hasta ser capaz de identificar objetos.

Los conceptos teóricos junto con la matemática presentada en este capítulo, son requeridos para comprender el desarrollo experimental que se presentará en los próximos dos capítulos.

Capítulo 3
Procesamiento de Imágenes

En el presente capítulo se describirá la primera parte del desarrollo experimental de esta tesis, la cual consiste en el preprocesamiento de las imágenes de fondo de ojo con el objetivo de que la red neuronal, que se presentará en el próximo capítulo, tenga un rendimiento adecuado. Para ello se plantea resaltar los signos de la retinopatía diabética mejorando el contraste de las imágenes, buscando también que la imagen resultante pueda ser utilizada por el personal médico para detectar, de una manera más sencilla, dicha patología.

Previo a la descripción del desarrollo experimental de este capítulo, se presentará información preliminar sobre las imágenes de fondo de ojo y de las herramientas de software utilizadas en esta tesis.

3.1 Preliminares

Si bien en este trabajo no se pretende ahondar en los conceptos médicos sobre las imágenes de fondo de ojo, en esta sección se proporcionará un panorama general sobre ellas, específicamente para conocer las regiones del ojo y los signos de retinopatía diabética que se pueden observan en estas imágenes, ya que será importante conocerlos tanto para el diseño del sistema de inferencia difuso como para evaluar los resultados del desarrollo experimental.

3.1.1 Imágenes de fondo de ojo

La retinografía es una prueba diagnóstica no invasiva donde se obtiene una imagen de la parte posterior del ojo, conocida como fondo de ojo por ser la capa más interna del globo ocular. En esta imagen es posible observar la distribución de los vasos sanguíneos y el tejido interno del ojo con sus diferentes áreas, las cuales son la mácula, la fóvea, la retina periférica y la papila óptica (también conocida como nervio óptico). En la figura 3.1 se muestra una imagen de fondo de ojo y la estructura anatómica del ojo que se puede observar en ella.

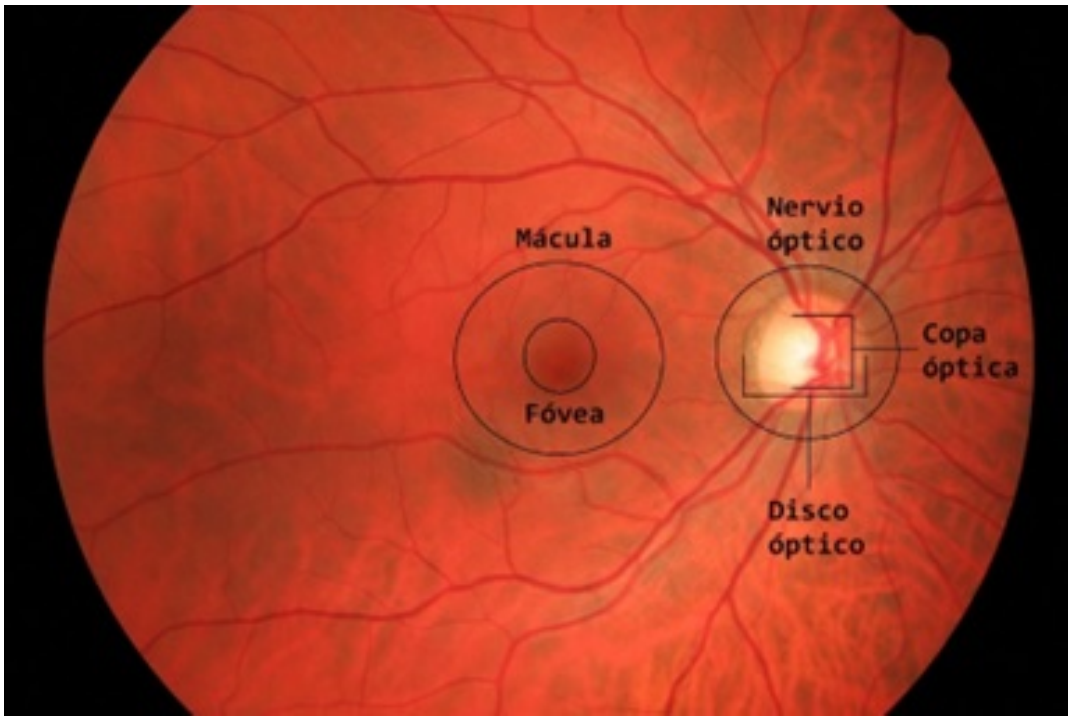


Figura 3.1: Estructura anatómica en una imagen de fondo de ojo.

Cada una de estas zonas desempeña una tarea específica para la visión: la mácula se encarga de la visión central, la retina de la visión periférica y en la papila óptica se agrupan las células ganglionares de la retina para llevar el impulso visual al cerebro y, también, es por donde pasa la arteria y la vena de la retina encargadas las demás capas de la retina.

En la figura 3.1, se muestra una imagen de fondo de ojo con las áreas previamente men-

cionadas. El análisis de estas imágenes por parte de médicos especialistas permite identificar problemas oculares como el glaucoma, el nevus coroideo y diferentes retinopatías. Entre las retinopatías más comunes se encuentran la degeneración macular asociada a la edad; el desgarro y desprendimiento de retina; la retinosis pigmentaria y la retinopatía hipertensiva, de Purtscher y diabética.

Cada uno de estos padecimientos exhibe signos característicos en diferentes áreas de estas imágenes. Aunque este trabajo se enfoca específicamente en la retinopatía diabética, se espera que los resultados de la mejora de contraste que se presentará en este capítulo sean aptos para cualquier otro padecimiento, facilitando el diagnóstico de un médico y permitiendo una mayor versatilidad en la aplicación de los resultados de este trabajo, pudiendo servir como punto de partida para futuras investigaciones, ampliando el número de patologías que puedan ser detectadas por una red neuronal.

Signos de la retinopatía diabética

La retinopatía diabética es una complicación crónica derivada de la diabetes mellitus, que puede llegar a causar edema macular, isquemia vascular, neovascularización, hemorragias y desprendimiento de retina. Como esta retinopatía puede dañar prácticamente toda la retina, sus síntomas son muy variados, entre ellos se encuentran la visión borrosa, visión variable, zonas de visión oscuras, cuerpos flotantes en la vista y la pérdida total de visión.

Los signos que se pueden observar en las imágenes de fondo de ojo para detectar la retinopatía diabética son los exudados, las hemorragias, los aneurismas y microaneurismas, la neovascularización y edema macular [64], los cuales se señalan en la imagen de fondo de ojo mostrado en la figura 3.2.

Exudados. Son depósitos de lípidos o proteínas que se presentan como manchas blancas o amarillentas en la retina. Dependiendo de su apariencia se pueden clasificar como

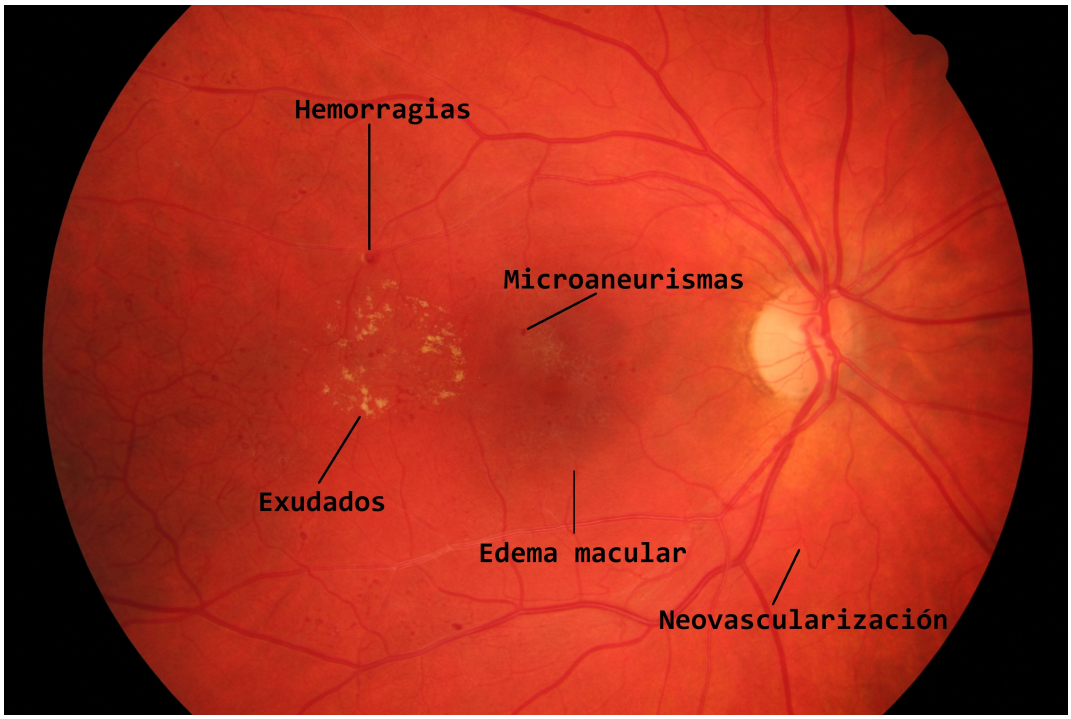


Figura 3.2: Signos de la retinopatía diabética en una imagen de fondo de ojo.

exudados duros o blandos. Los duros tienen los bordes bien definidos, mientras que los blandos los tienen difusos, por lo que también se conocen como algodonosos.

Hemorragias Causadas por la ruptura de vasos sanguíneos, se observan como puntos o manchas en las imágenes.

Microaneurismas Son pequeñas protuberancias redondas en las arteriolas de la retina. Al igual que las hemorragias, se muestran como puntos, pero se distinguen de ellas por su forma redonda.

Neovascularización Crecimiento anormal de nuevos vasos sanguíneos que se pueden detectar por tener formas irregulares y ser más delgados.

Edema macular Es una inflamación en la mácula, por lo que, en las imágenes de fondo de ojo, se puede presentar la zona con un color amarillento o blanquecino, dependiendo si es un edema macular duro o blando, respectivamente.

3.1.2 Fuentes de datos y recursos computacionales

Para el desarrollo de esta tesis, se utilizaron imágenes de fondo de ojo obtenidas de tres bases de datos, la primera pertenece a la Universidad Friedrich-Alexander Universität Erlangen-Nürnberg [65] y contiene un total de 45 imágenes, de las cuales 15 están etiquetadas con retinopatía diabética, 15 con glaucoma y 15 saludables; la segunda pertenece a la empresa ADCIS y su base es conocida como MESSIDOR (del francés Méthodes d'Évaluation de Systèmes de Segmentation et d'Indexation Dédiées à l'Ophtalmologie Rétinienne) [66], la cual contiene 1200 imágenes etiquetadas con cuatro posibles grados de retinopatía y con tres posibles grados de riesgo de edema macular, donde el grado 0 indica que son saludables; la última base de datos se obtuvo de Kaggle [67] y contiene 53,586 imágenes etiquetadas con cinco posibles grados de retinopatía donde, al igual que en la anterior base de datos, el grado 0 indica que son saludables.

Software utilizado

Existen diversos lenguajes de programación para poder procesar imágenes, entre los más comunes se encuentran el Python, C++ y Java. Aunque todos ellos cuentan con bibliotecas especializadas para el procesamiento de imágenes, se ha elegido utilizar Python, ya que aparte de contar con numerosas bibliotecas para el procesamiento de imágenes como OpenCV, Pillow, Pandas y scikit-image, también cuenta con bibliotecas para cálculos numéricos como NumPy y para crear modelos de Machine Learning como scikit-learn, Keras, TensorFlow y PyTorch. Cabe señalar que éstas dos últimas se consideran también como frameworks por ofrecer herramientas adicionales para el desarrollo de los modelos, aparte de las funciones propias de la biblioteca.

Los programas de Python desarrollados en esta tesis se escribieron utilizando principalmente tres herramientas: la interfaz de desarrollo integrado (IDE por su acrónimo en inglés de *Integrated Development Environment*), Spyder instalado en una computadora personal (PC por sus siglas en inglés de *Personal Computer*) con un procesador AMD Rizen 5 3500U, el IDE Thonny instalado

en la Raspberry Pi 4 Modelo B y una plataforma de Google para realizar computación en la nube (mejor conocido por su nombre en inglés *Cloud Computing*) llamada Google Colab.

En el presente capítulo se presentará el desarrollo del sistema para mejorar el contraste de las imágenes, donde todos los programas utilizados se crearon y optimizaron en la computadora personal con el IDE Spyder. Para la segunda parte del desarrollo experimental, que se describirá en el cuarto capítulo, se utilizaron tanto el IDE Spyder como Google Colab, el primero se empleó para crear y utilizar los programas para la preparación de las imágenes (preprocesamiento) y Google Colab para el entrenamiento de la red. Finalmente, en la Raspberry se utilizó el IDE Thonny para la creación de una interfaz de usuario.

3.2 Procesamiento de imágenes de fondo de ojo (mejora de contraste)

Debido a que el objetivo de este preprocesamiento es resaltar los signos de la retinopatía diabética y, tomando en cuenta que los microaneurismas y hemorragias se presentan como manchas más oscuras y los exudados como manchas blancas o amarillentas, para esta tarea se propone realizar una mejora de contraste en las imágenes de fondo de ojo.

Como se mencionó en el capítulo anterior, existen numerosos algoritmos para mejorar el contraste de las imágenes, entre ellos, la ecualización de histograma y sus múltiples variantes son de los más básicos, sin embargo, para este trabajo se propone utilizar un método que permita adaptarse a los requerimientos de las imágenes de fondo de ojo, tal como la lógica difusa.

Actualmente existen numerosos trabajos donde, precisamente, utilizan la lógica difusa para mejorar el contraste de imágenes digitales. En muchos de ellos mencionan que la principal ventaja es que permite crear sistemas de mejora adaptables a las necesidades de cada imagen,

como Sharma y Bhatia en [68], donde desarrollaron un método para mejorar el contraste tanto de imágenes cromáticas como de acromáticas controlando un gradiente por medio de un FIS; o en [69] donde Samrudh y Sandeep utilizaron la lógica difusa para modificar el valor de los pixeles dependiendo de su valor inicial para mejorar el contraste en imágenes de un solo canal. En un trabajo más reciente [70] S. Fernandez et al., propusieron un método de mejora de contraste adaptativo para imágenes acromáticas utilizando dos FIS, donde el primero decide cuánto contraste se aplicará y el segundo lo ejecuta utilizando una técnica similar al mostrado en [69].

Generalmente, el campo de aplicación de estas técnicas está en las imágenes cotidianas, sin embargo, son un punto de partida para el desarrollo de técnicas para procesar imágenes específicas, tal como las imágenes médicas.

En el caso de las imágenes de fondo de ojo se han utilizado técnicas básicas para mejorar su contraste, como Agung W. Setiawan y colaboradores en [71], donde utilizan CLAHE en el canal G del espacio de color RGB. También se han utilizado procesamientos más complejos utilizando más de una técnica como en [72] donde M. Zhou et al., antes de mejorar el contraste utilizando también CLAHE, mejoran la luminosidad; un ejemplo más se tiene en [73], donde Lvchen Cao et al. utilizan un filtro pasa bajas y otra técnica llamada "α-rooting".

La lógica difusa también ha sido utilizada para mejorar el contraste de las imágenes de fondo de ojo, aunque se ha aplicado principalmente para modificar el histograma de las imágenes, tal como es el caso de [74], donde Sandeep y colaboradores desarrollaron un algoritmo para mejorar el contraste modificando solo una parte del histograma (donde se encuentra la región de interés) aplicando un FIS en esa área del histograma; en otro trabajo, Niladri y colaboradores utilizaron la lógica difusa para obtener un "histograma difuso", el cual es seccionado por medio de los puntos máximos locales para aplicar una ecualización de histograma a cada sección [75]; en otro trabajo más reciente [76], Basma y Duaa también utilizaron la lógica difusa para obtener el histograma difuso para seccionarlo y ecualizarlo, pero a diferencia de [75], utilizaron la lógica difusa para la ecualización.

Pese a lo anterior, en esta parte del desarrollo experimental se pretende utilizar la lógica difusa para mejorar el contraste de las imágenes de una manera directa, es decir, modificando directamente los píxeles en lugar del histograma, aprovechando la capacidad de la lógica para interpretar los conocimientos de un experto y lograr un procesamiento adecuado para este tipo de imágenes, ya que al trabajar con imágenes médicas, no basta con tener una mejora visual general, sino que la mejora debe tener ciertas condiciones para que pueda considerarse adecuada. Por ello, en este trabajo se buscó la opinión de personal experto que, en este caso, corresponde a médicos especialistas, ya sea oftalmólogos u optometristas.

Con base en la asesoría de dos optometristas se determinó que, para mejorar el contraste de las imágenes de fondo de ojo adecuadamente, se deben tener en cuenta tres consideraciones:

1. Mantener la tonalidad de los colores.
2. Mantener la forma de las regiones de mácula.
3. Mantener la forma de las regiones del nervio óptico

La importancia de estas consideraciones radica en que si se tiene alguna alteración significativa sobre ellas, podría resultar en un diagnóstico erróneo por parte del personal médico y, por lo tanto, en la red neuronal. Por ello, para mejorar el contraste de las imágenes de fondo de ojo se propone tomar como base la metodología propuesta en [70], donde utilizan un FIS para ajustar los valores de los píxeles y otro FIS para determinar el grado de dicho ajuste, adaptándola específicamente a las necesidades de este tipo de imágenes.

Una de las razones por la cual se eligió seguir dicha metodología fue por su capacidad de adaptarse a las necesidades de cada imagen. Para lograr esta adaptabilidad, antes de modificar el valor de los píxeles, se realiza una evaluación del contraste inicial de las imágenes por medio de un FIS, el cual, en la entrada recibe un parámetro con la información de dicho contraste y como salida se obtiene otro parámetro que determinará, por medio del siguiente FIS, qué tanto ajuste de contraste se realizará a las imágenes.

3.3 Metodología propuesta

Para mejorar el contraste de las imágenes de fondo de ojo de una manera adecuada, es necesario considerar que es vital no alterar la tonalidad de color de las imágenes, por ello se planteó utilizar un espacio de color donde la información de la iluminación y del color estén separadas para poder modificar sólo el canal de iluminación. En este contexto, la metodología propuesta para realizar esta tarea se ilustra en el diagrama de flujo de la figura 3.3.

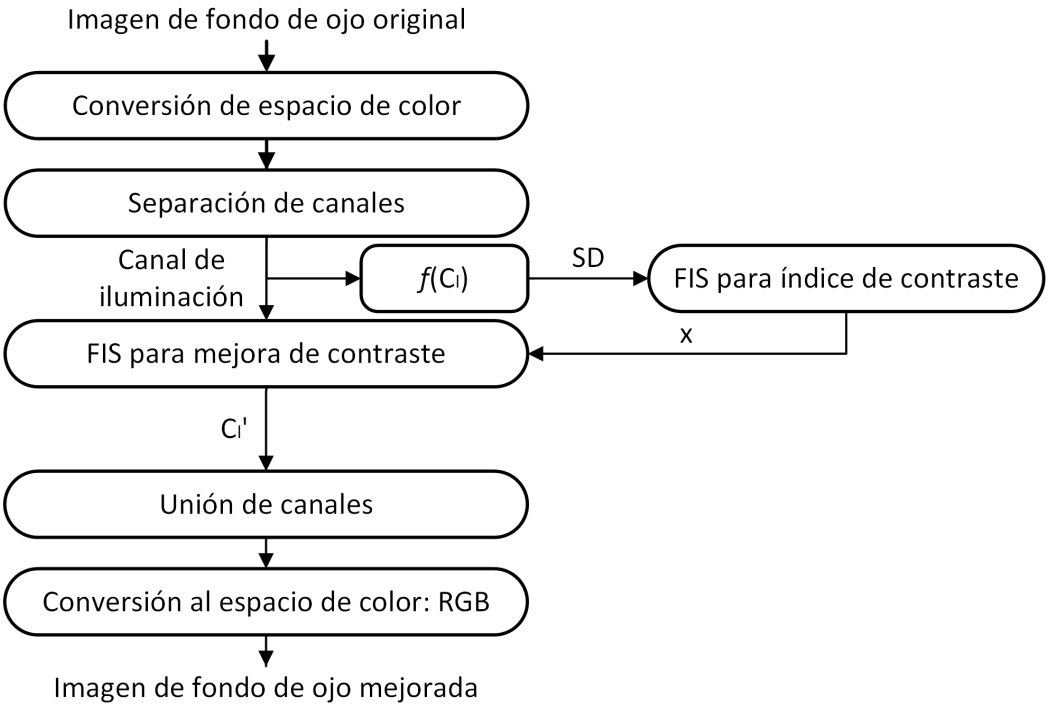


Figura 3.3: Diagrama de flujo de la metodología propuesta.

Como se observa en el diagrama de la figura 3.3, el primer paso de esta metodología consiste en realizar una conversión de espacio de color debido a que, por lo general, todas las imágenes digitales están representadas con el espacio RGB para poder ser observadas en un monitor y las imágenes de fondo de ojo no son la excepción. Para ello, se propone que el espacio de color a utilizar tenga la información de la iluminación en un canal independiente de la información del color.

El segundo paso consiste en realizar una separación de canales para modificar únicamente el que contiene la información de la iluminación (mostrado en el diagrama como C_I) y evitar una posible modificación drástica en color de la imagen. Entre los espacios de color con estas características se encuentran HSI, HSV, YCbCr y CIELAB. La elección del espacio de color más adecuado para procesar las imágenes se determinó de manera experimental, por ello, primero se describirá el desarrollo de los sistemas de inferencia difusos, donde el primero tiene la tarea de interpretar el valor original del contraste para generar el parámetro x, que indicará el grado de ajuste de contraste que aplicará el segundo FIS.

3.3.1 FIS para índice de contraste

Para obtener el parámetro x se utilizó un FIS de tipo Takagi-Sugeno de orden cero, debido a que la relación entre la variable de entrada y la salida será lineal. Como variable de entrada se utilizó el contraste RMS (2.33) de las imágenes originales y se definieron tres funciones de membresía asociados a las variables lingüísticas *bajo*, *medio* y *alto*, que se refiere a *bajo contraste*, *contraste medio* y *alto contraste*, respectivamente.

Para definir el universo de discurso de este FIS, se calculó la desviación estándar de 1000 imágenes de fondo de ojo en escala de grises, utilizando el algoritmo mostrado en la sección de código 3.1.

Sección de código 3.1: Cálculo de la desviación estándar.

```
filenames = next(os.walk("/imagenes"), (None, None, []))[2]
filenames.sort()
imagenes = []

num_imagenes = min(1000, len(filenames))

for i in range(num_imagenes):
    ruta_imagen = os.path.join("/imagenes", filenames[i])
```

```
    imagen = io.imread(ruta_imagen)
    imagenes.append(imagen)

imagenes = np.array(imagenes)
desviacion_estandar = np.std(imagenes)
```

Con los resultados obtenidos del programa anterior, se encontró que los valores de desviación estándar más altos se encontraban alrededor de 0.26 con un máximo de 2.89 y los más bajos alrededor de 0.06. Tomando en cuenta estos valores y que, debido a la naturaleza de su cálculo, el valor mínimo que puede tomar el contraste RMS es cero, se definió el universo de discurso en un rango de 0 a 0.3.

Las funciones de membresía utilizadas fueron las funciones S, Z y la gaussiana G, definidas por las ecuaciones (2.2), (2.3) y (2.4), respectivamente. Para la fuzzificación se asignó la variable lingüística "$bajo$" a la función Z, "$medio$" a la función G y "$alto$" a la función S. En la tabla 3.1 se muestran los valores numéricos utilizados en los parámetros α, β, γ, σ para crear dichas funciones.

Tabla 3.1: Parámetros de las funciones de membresía de entrada del primer FIS.

Función	Parámetro	Valor numérico
Z	α	0.750
	β	0.130
G	γ	0.175
	σ	0.035
S	α	0.750
	β	0.210

Para la creación de estas funciones de membresía en el lenguaje de programación Python se utilizó el algoritmo mostrado en la sección de código 3.2, donde también se grafican dichas funciones.

Sección de código 3.2: Funciones de membresía del primer FIS en Python.

```
sd = np.linspace(0.0, 0.3, 256)
gmin = 1 / (1 + np.exp(0.75*(sd - 0.130)))
gmed = np.exp((-1/2)*(((sd - 0.175)/0.035)**2
gmax = 1 / (1 + np.exp(-0.75*(sd - 0.210)))

plt.figure(0)
plt.plot(sd, gmin)
plt.plot(sd, gmed)
plt.plot(sd, gmax)
```

Con la función de la primera línea de esta sección de código se generó una secuencia de 256 números que, en este caso específico, representan el universo de discurso para el contraste RMS, cuyo rango se estableció de 0 a 3; en las siguientes tres líneas se definen matemáticamente las funciones de membresía **S**, **G** y **Z**, respectivamente; y las siguientes cuatro líneas grafican estas funciones, teniendo al contraste RMS como dominio de las funciones y el grado de membresía como su rango.

Esta sección de código tiene el único objetivo de graficar dichas funciones de membresía para tener la certeza de que, con los parámetros elegidos, tengan la forma deseada. De esta manera, al ejecutarlo se obtiene la gráfica mostrada en la figura 3.4.

Como se mencionó anteriormente, para este FIS la variable de entrada se relaciona linealmente con la salida y, en palabras sencillas, se puede expresar de la siguiente manera:

"Si tiene bajo contraste el ajuste deberá ser mayor, si tiene un contraste medio el ajuste será medio y si tiene un contraste alto el ajuste será poco."

Consecuentemente, las variables lingüísticas de salida se establecieron como *"mucho ajuste"*, *"ajuste medio"* y *"poco ajuste"* y se asociaron a los valores de salida f_{11}, f_{12} y f_{13}, respectivamente, conocidos como valores *singleton* por ser conjuntos difusos con solo un valor. Los valores numéricos utilizados se muestran en la tabla 3, cuya elección se explicará con el siguiente FIS.

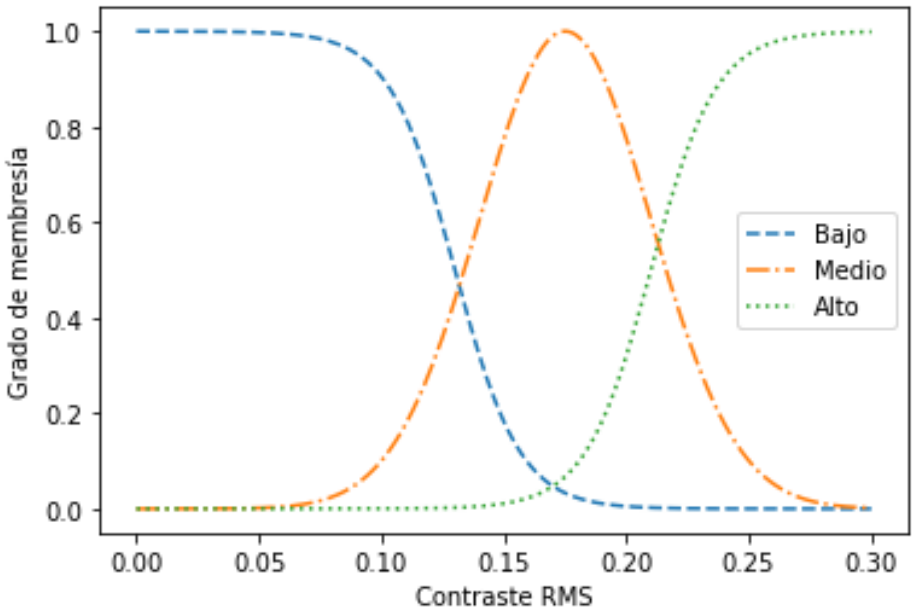


Figura 3.4: Funciones de membresía de entrada del primer FIS.

Tabla 3.2: Valores *singleton* de salida del primer FIS.

Singleton	**Valor numérico**
f_{11}	0.10
f_{12}	0.25
f_{13}	0.35

Las reglas *IF-THEN* o *SI-ENTONCES* para la inferencia difusa se definieron de la siguiente manera:

1. ***SI*** $bajo$ ***ENTONCES*** f_{11}
2. ***SI*** $medio$ ***ENTONCES*** f_{12}
3. ***SI*** $alto$ ***ENTONCES*** f_{13}

Para la defuzzificación se utilizó el método de promedio ponderado WAM (2.6), por lo que

la salida de este FIS, con el cual se obtendrá el parámetro x, está dada por la ecuación (3.1).

$$x = \frac{(\mu_{11} \cdot f_{11}) + (\mu_{12} \cdot f_{12}) + (\mu_{13} \cdot f_{13})}{\mu_{11} + \mu_{12} + \mu_{13}} \tag{3.1}$$

Donde $\mu_{11} \cdot f_{11}$, $\mu_{12} \cdot f_{12}$ y $\mu_{13} \cdot f_{13}$ representan los grados de activación de las reglas difusas, por lo que μ_{11}, μ_{12} y μ_{13} son los valores de membresía de sus respectivas reglas difusas, los cuales son obtenidos al evaluar el contraste RMS de la imagen de entrada en las funciones de membresía.

En la sección de código 3.3 se muestra el algoritmo completo para la creación de este primer FIS, el cual se definió dentro de una función que recibe como parámetro de entrada el valor de la desviación estándar y como salida se obtiene el parámetro de ajuste x.

Sección de código 3.3: Definición del primer FIS en Python

```
def fisSD (std_d):
    gmin = 1 / (1 + np.exp( 75*(std_d - 0.130)))
    gmed = np.exp((-1/2)*(((std_d - 0.175)/0.035)**2
    gmax = 1 / (1 + np.exp(-75*(std_d - 0.210)))
    f1 = 0
    f2 = 0.175
    f3 = 0.3
    x = ((gmin*f1)+(gmed*f2)+(gmax*f3)) / (gmin+gmed+gmax)
    return x
```

Para observar el comportamiento de este primer FIS se graficó la salida evaluando todos los posibles valores de la desviación estándar y se obtuvo la curva de transferencia mostrada en la figura 3.5.

En esta curva de transferencia se puede observar que el valor mínimo que puede tomar el parámetro de ajuste x es 0.1 y 0.35 para el valor máximo.

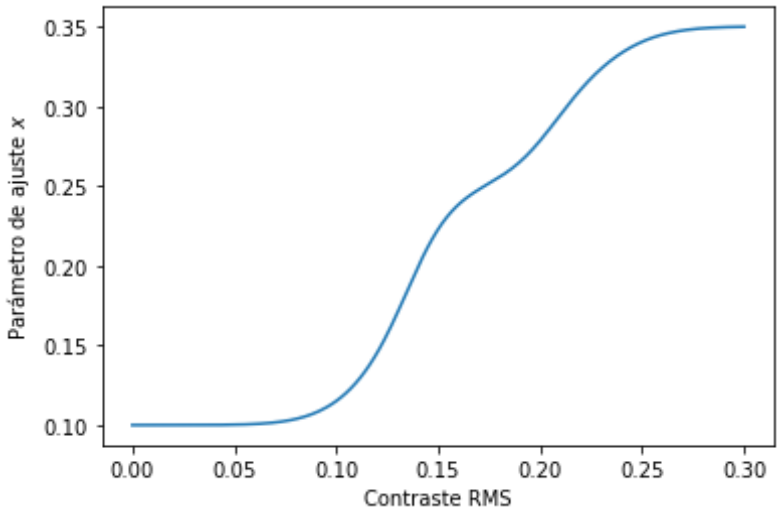


Figura 3.5: Curva de transferencia del primer FIS.

3.3.2 FIS para mejora de contraste

Para mejorar el contraste de la imagen, se empleará un segundo FIS, diseñado específicamente para crear un nuevo canal a partir de los valores ajustados del canal original. En este caso, la entrada del FIS consistirá en el canal que contiene la información de iluminación de la imagen original.

Durante el proceso de inferencia difusa, se evaluará individualmente el valor de cada píxel y se ajustarán para incrementar o reducir su intensidad según corresponda a su valor original y al parámetro de ajuste x. Como resultado se obtendrá un nuevo canal que sustituirá al de entrada, generando una imagen con mejor contraste sin alterar la tonalidad de los colores.

Debido a que la relación entre la entrada y la salida también será lineal, se propuso nuevamente un FIS de tipo Takagi-Sugeno de orden cero, donde el universo de discurso contempla todos los posibles valores que pueden tomar los píxeles. Como estos valores dependen del espacio de color utilizado, se realizará una normalización previa para obtener un universo de discurso con un rango de 0 a 1, lo que también permitirá obtener un ajuste fino por utilizar decimales.

Tomando en cuenta las consideraciones mencionadas, para la fuzzificación se utilizaron funciones de tipo S y Z en los extremos (al igual que en el FIS anterior) y funciones gaussianas G para las demás variables lingüísticas con el fin de que la inferencia sea suave y el cambio entre grados de pertenencia sea menos abrupta.

Como este universo de discurso contempla la intensidad de los píxeles, se propusieron las siguientes cinco variables lingüísticas: *negro*, *gris oscuro*, *gris medio*, *gris claro* y *blanco*.

Donde *"negro"* considera los valores cercanos a 0, *"gris medio"* valores alrededor de 0.5 y *"blanco"* cercanos a 1; en el caso de *"gris oscuro"* contempla los valores intermedios entre 0 y 0.5 y, finalmente, *"gris claro"* valores entre 0.5 y 1.

Para lograr la suavidad en dicha inferencia, se ajustaron los puntos máximos de grado de membresía a una distancia equidistante entre cada función consecutiva, tal como se muestra en la gráfica de la figura 3.6.

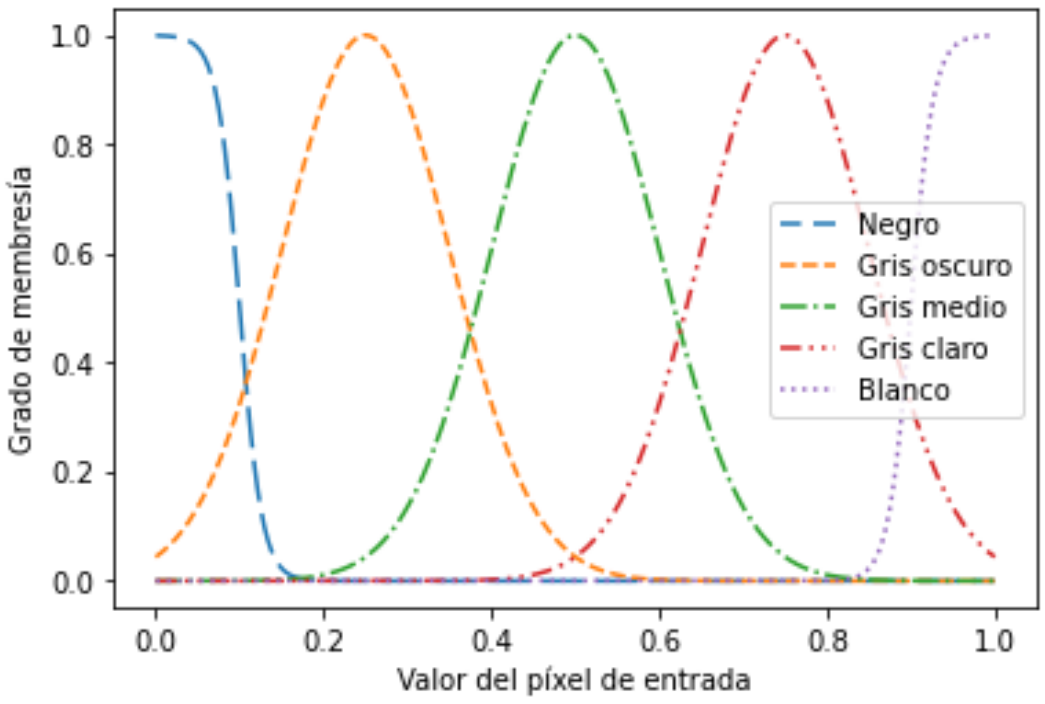


Figura 3.6: Funciones de membresía de entrada del segundo FIS.

Los parámetros utilizados para crear estas funciones de membresía se muestran en la tabla

3.3 y su creación en Python en la sección de código 3.4.

Tabla 3.3: Parámetros de las funciones de membresía de entrada del segundo FIS.

Función	Parámetro	Valor numérico
Z	α	0.75
	β	0.10
G_1	γ	0.25
	σ	0.10
G_2	γ	0.50
	σ	0.10
G_3	γ	0.75
	σ	0.10
S	α	0.75
	β	0.90

Sección de código 3.4: Funciones de membresía del segundo FIS en Python.

```
pix = np.linspace(0.0, 1.0, 256)
pmin = 1 / (1 + np.exp( 75*(pix - 0.10)))
pbla = np.exp((-1/2)*(((pix - 0.25)/0.10)**2))
pmed = np.exp((-1/2)*(((pix - 0.50)/0.10)**2))
pwit = np.exp((-1/2)*(((pix - 0.75)/0.10)**2))
pmax = 1 / (1 + np.exp(-75*(pix - 0.90)))
plt.figure(2)
plt.plot(pix, pmin)
plt.plot(pix, pbla)
plt.plot(pix, pmed)
plt.plot(pix, pwit)
plt.plot(pix, pmax)
```

La manera en cómo se ajustarán los valores de los píxeles con la inferencia difusa se puede expresar con palabras con la siguiente frase:

"Tonalidades negras se quedan negras, tonalidades oscuras deben ser más oscuras, tonalidades medias se quedan medias, tonalidades claras se aclararán más y tonalidades blancas se quedan blancas."

Bajo esta lógica, se definieron las 5 siguientes reglas difusas:

1. ***SI*** $negro$ ***ENTONCES*** $negro$
2. ***SI*** $gris\,oscuro$ ***ENTONCES*** $gris\,mas\,oscuro$
3. ***SI*** $gris$ ***ENTONCES*** $gris$
4. ***SI*** $gris\,claro$ ***ENTONCES*** $gris\,mas\,claro$
5. ***SI*** $blanco$ ***ENTONCES*** $blanco$

Dicho de otra manera, para mejorar el contraste se modificarán los valores de las tonalidades intermedias haciéndolas más claras u oscuras, dependiendo si son tonalidades medio claras u oscuras, respectivamente. Sin embargo, siguiendo la metodología de [70], se definieron dos reglas difusas más, con el objetivo de que el cambio de tonalidades sea mayor y así lograr una mejora de contraste suave pero considerable. Estas dos reglas son:

6. ***SI*** $gris\,oscuro$ ***ENTONCES*** $negro$
7. ***SI*** $gris\,claro$ ***ENTONCES*** $blanco$

Con estas dos reglas agregadas se pretende obtener una función de transferencia con una pendiente mayor en los extremos, lo que indica que las tonalidades medias no tendrán un cambio significativo y las tonalidades más claras y oscuras sí.

Al igual que en el FIS anterior, las variables lingüísticas de salida se asociaron con sus respectivos valores *singleton*, cuyo valor numérico indica de la luminosidad a la que deberá tender el ajuste del píxel, dependiendo de su valor de entrada. Con el fin de seguir una nomenclatura

similar en los parámetros de ambos FIS, los valores *singleton* asociados a las variables lingüísticas de salida son las siguientes:

1. $Negro = f_{21}$
2. $Gris\,mas\,oscuro = f_{22}$
3. $Gris = f_{23}$
4. $Gris\,mas\,claro = f_{24}$
5. $Blanco = f_{25}$

Y sus valores numéricos se muestran en la tabla 3.4.

Tabla 3.4: Valores *singleton* de salida del segundo FIS.

Singleton	**Valor numérico**
f_{21}	0.0
f_{22}	x
f_{23}	0.5
f_{24}	$1 - x$
f_{25}	1.0

De esta manera, se definieron las siete reglas difusas como se muestra a continuación:

1. ***SI*** $negro$ ***ENTONCES*** f_{21}
2. ***SI*** $gris\,oscuro$ ***ENTONCES*** f_{22}
3. ***SI*** $gris$ ***ENTONCES*** f_{23}
4. ***SI*** $gris\,claro$ ***ENTONCES*** f_{24}
5. ***SI*** $blanco$ ***ENTONCES*** f_{25}

6. ***SI*** $gris\,oscuro$ ***ENTONCES*** f_{21}

7. ***SI*** $gris\,claro$ ***ENTONCES*** f_{25}

El objetivo del parámetro de ajuste x es modificar los valores *singleton* f_{22} y f_{24}, de esta manera se obtiene una relación inversamente proporcional entre su valor y el nivel de ajuste.

Para comprender el motivo de la elección del rango de valores que puede tomar este parámetro (0.1 a 0.35) obtenido con el FIS anterior, es necesario tener claro que los tonos *"gris oscuro"* corresponden a los píxeles en el rango de 0 a 0.5, de esta manera si el valor *singleton* es bajo, por ejemplo 0.1 (su menor valor posible), estos tonos *"gris oscuros"* se deberán ajustar a un valor más cercano a 0, lo que significa que serán más oscuros; en el otro caso, cuando su valor es máximo (0.35), el ajuste se realizará exactamente hacia la mitad entre el negro y el gris, lo que genera un cambio mínimo en la tonalidad original. Lo mismo ocurre con los tonos *"gris claro"* al ser el valor *singleton* el complemento del parámetro x.

Para la defuzzificación se utilizó el mismo método (WAM), por lo que ecuación está dada por (3.2)

$$salida = \frac{\sum_{i=1}^{n} \mu(I)_{2i} \cdot f_2}{\sum_{i=1}^{n} \mu(I)_{2i}} \tag{3.2}$$

Donde $\mu(I)_{2i}$ es el valor de membresía de la regla i obtenido al evaluar las funciones de membresía con el píxel de entrada y f_2 el valor *singleton* de salida correspondiente y n representa el número de reglas difusas.

La función creada en Python para generar este FIS se muestra en la sección de código 3.5.

Sección de código 3.5: Definición del segundo FIS en Python.

```
def fisImg (x, canal):
    [fil, col] = canal.shape
    mf1 = 0.0
    mf2 = x
    mf3 = 0.5
    mf4 = 1.0 - x
```

```
    mf5 = 1.0

    pixS = np.zeros((fil, col))
    for i in range (fil):
        for j in range (col):
            pixI = canal[i,j]
            pmin = 1 / (1 + np.exp( 75*(pixI - 0.10)))
            pbla = np.exp((-1/2)*(((pixI - 0.25)/0.10)**2))
            pmed = np.exp((-1/2)*(((pixI - 0.50)/0.10)**2))
            pwit = np.exp((-1/2)*(((pixI - 0.75)/0.10)**2))
            pmax = 1 / (1 + np.exp(-75*(pixI - 0.90)))

            pixS[i,j] = ((pmin*mf1)+(pbla*mf2)+(pmed*mf3)+
    (pwit*mf4)+(pmax*mf5)+(pbla*mf1)+(pwit*mf5))/
    (pmin+pbla+pmed+pwit+pmax+pbla+pwit)
    return pixS
```

Para observar el comportamiento de este FIS, se utilizó la función anterior con tres valores distintos del parámetro x y se graficaron las funciones de transferencia resultantes, las cuales se muestran en la figura 3.7. Estos tres valores fueron el valor máximo que puede tomar (0.35), el mínimo (1.0) y un valor intermedio (0.25).

De la figura 3.7, se puede notar que mientras menor sea el parámetro x mayor será la pendiente de la curva de transferencia, resultado de que los tonos menores a la media se ajusten a un valor más cercano a 0 y los mayores a la media se ajusten a valores más cercanos a 1, aumentando el rango dinámico de la imagen.

3.3.3 Espacios de color

Como se mencionó previamente, el primer paso de esta metodología consiste en realizar una conversión del espacio RGB a otro cuya información de la iluminación esté separada de

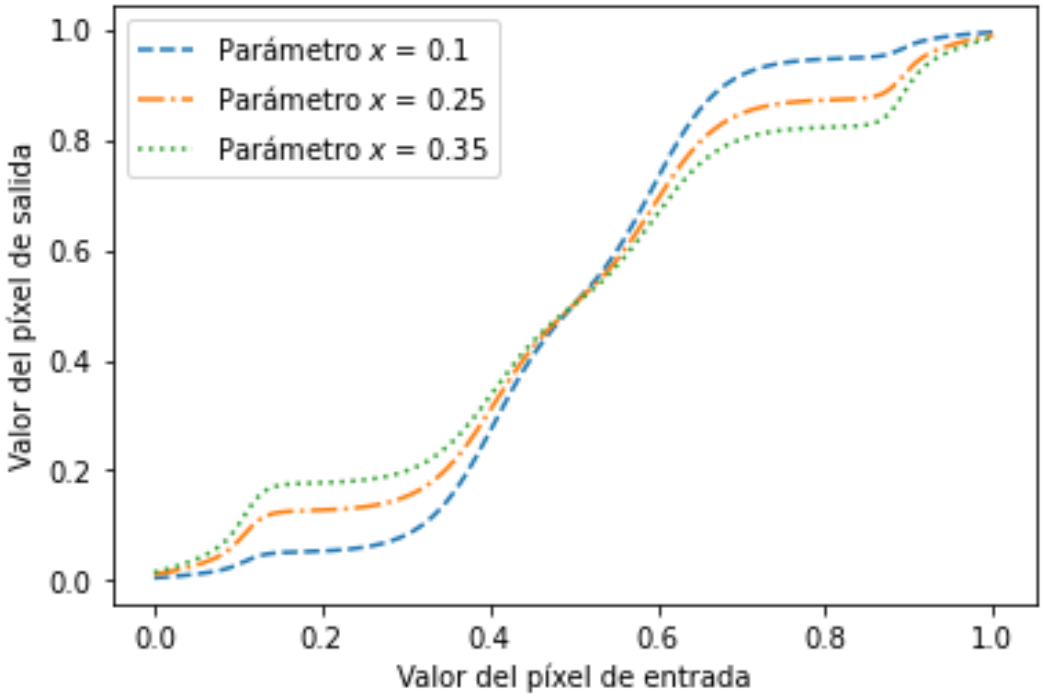


Figura 3.7: Curvas de transferencia del segundo FIS.

la tonalidad de los colores. Para la elección del espacio de color más adecuado se utilizaron los espacios de color YCbCr, HSI y CIELAB. Las imágenes resultantes de cada espacio de color se evaluaron visualmente con el apoyo de los dos optometristas, cuidando que cumplieran las consideraciones antes mencionadas.

YCbCr

La utilización de Python para la conversión entre espacios de color supone una gran ventaja, ya que dentro de la biblioteca *skimage* existe una función optimizada para pasar del espacio RGB al YCbCr y viceversa, utilizando las ecuaciones de conversión (2.34), (2.35) y (2.36), cuya documentación se encuentra en su página de internet [77]. Con estas funciones de la biblioteca para la conversión más los FIS previamente definidos, el algoritmo para procesar las imágenes utilizando el espacio de color YCbCr se simplifica considerablemente como se observa en la sección de código 3.6.

Sección de código 3.6: Mejora de contraste con el espacio YCbCr.

```
1 Original = io.imread("/Prueba_5L.jpeg")
2 YCbCr = color.rgb2ycbcr(Original)
3 Canal_Y = YCbCr[:,:,0]/255.0
4 std = np.std(Canal_Y)
5 x = fisSD(std)
6 Canal_Mejorado = fisImg(stdO, ilu)
7 YCbCr[:,:,0] = Canal_Mejorado*255.0
8 rgb = color.ycbcr2rgb(YCbCr)
```

Todos los programas desarrollados para mejorar el contraste siguen la metodología mostrada en el diagrama de la figura 3.3, para ejemplificarlo, se describirá brevemente el algoritmo de la sección de código anterior línea por línea.

Línea 1 Se carga la imagen original.

Línea 2 Se realiza la conversión de RGB a YCbCr con la función `color.rgb2ycbcr(Original)`.

Línea 3 Se aísla el canal que contiene la información de la iluminación (el canal "Y" para este espacio) y se normalizan sus valores.

Línea 4 Se obtiene la desviación estándar del canal utilizando la función `np.std()` de la biblioteca numpy.

Línea 5 Se aplica el primer FIS utilizando la función `fisSD(std)` creada anteriormente en la sección de código 3.3, con el cual se obtiene el parámetro x.

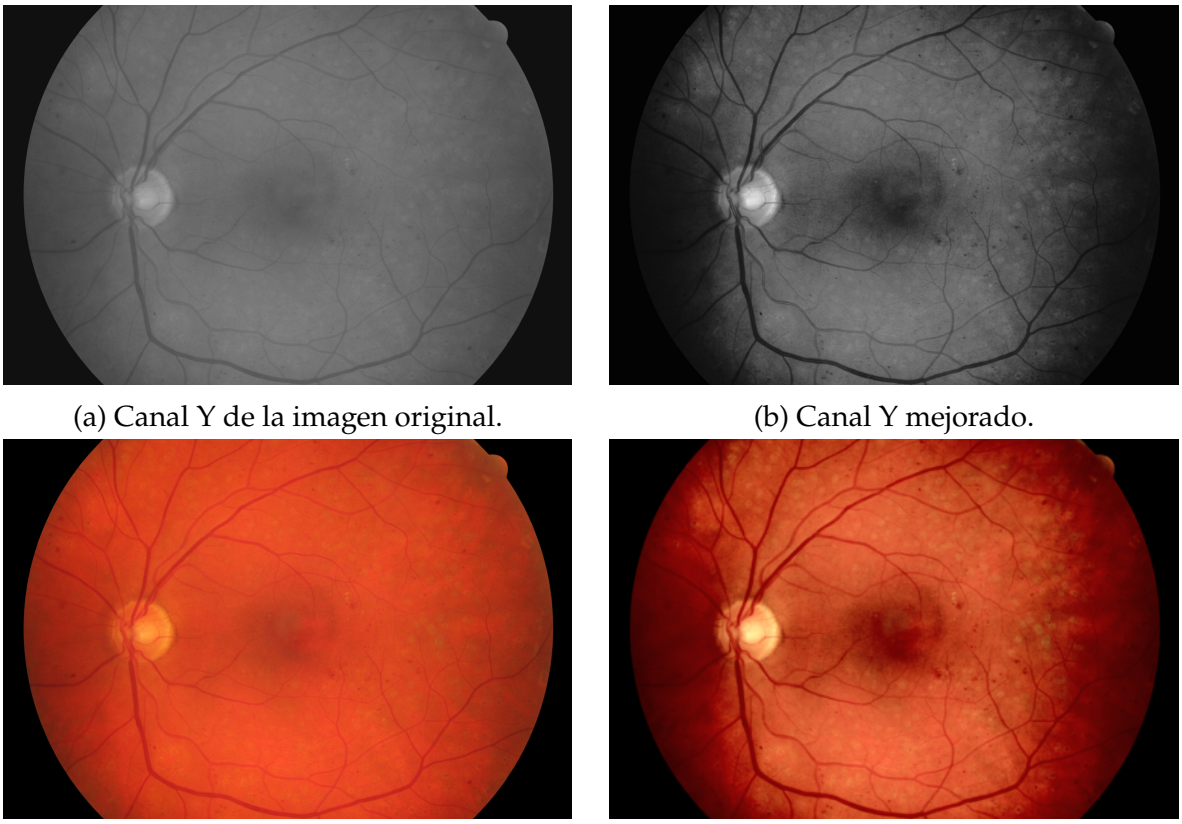
Línea 6 Se aplica el segundo FIS con la función `fisImg(stdO, ilu)` descrita previamente en la sección de código 3.5, con el cual se obtiene un nuevo canal con el contraste mejorado.

Línea 7 Se sustituye el canal Y de la imagen original por el canal obtenido de la salida del segundo FIS, obteniendo una nueva imagen con el contraste mejorado en el espacio YCbCr.

Línea 8 Se realiza la conversión YCbCr a RGB con la función `color.ycbcr2rgb(YCbCr)`.

Para observar el desempeño de este espacio de color, se procesaron las 45 imágenes de la base de datos HRF [65] y, con el apoyo de los dos optometristas, se evaluó visualmente el resultado.

En la figura 3.8 se muestra la comparación en una imagen de fondo de ojo procesada, donde la figura 3.8a muestra el canal Y original, la figura 3.8b el canal Y mejorado, la figura 3.8c la imagen original y la figura 3.8d la imagen mejorada.



(a) Canal Y de la imagen original.

(b) Canal Y mejorado.

(c) Imagen original.

(d) Imagen mejorada.

Figura 3.8: Comparación entre la imagen original y la mejorada en el canal Y del espacio YCbCr.

Comparando los dos canales "Y" de la figura 3.8 se puede observar la evidente mejora de contraste de la imagen procesada, lo que comprueba que el método propuesto es efectivo en la tarea de mejorar el contraste. Sin embargo, la evaluación del espacio YCbCr para procesar estas imágenes no resulta favorable ya que, como se puede observar en la figura 3.8, en la imagen resultante se pueden observar tonalidades muy claras en zonas donde originalmente tenía tonos medios. Este cambio en la tonalidad del color provoca que no se cumpla la primera

consideración, por lo tanto, este espacio de color no resulta adecuado para procesar imágenes de fondo de ojo con la metodología propuesta.

HSI

En el espacio HSI la información de la iluminación se encuentra en el canal I por lo que, a grandes rasgos, solo sería necesario realizar la modificación de las líneas 2 y 8 en la sección de código 3.6 (que es donde se hace la conversión entre los espacios de color) y en las líneas 3 y 7, donde se selecciona el canal a modificar y reemplazar (ya que en HSI es el tercero), respectivamente. Sin embargo, en ninguna biblioteca disponible existe una función para la conversión entre los espacios RGB y HSI, por lo que fue necesario crear estas funciones utilizando las ecuaciones de conversión (2.37)–(2.40) para pasar de RGB a HSI y (2.41)–(2.49) para pasar de HSI a RGB.

En la sección de código 3.7 se muestra la primera versión creada para realizar esta conversión.

Sección de código 3.7: Conversión RGB a HSI utilizando ciclos FOR.

```
1  def RGB2HSI (RGB):
2      rgb = np.float32(RGB)/255.0
3      r = rgb[:,:,0]
4      g = rgb[:,:,1]
5      b = rgb[:,:,2]
6      Fil = rgb.shape[0]
7      Col = rgb.shape[1]
8      def Calc_H(r,g,b):
9          h = (0.5*((r-g)+(r-b))) / math.sqrt(((r-g)**2)+(r-b)*(g-b))
10         h = math.acos(h)
11         if b <= g:
12             return h*(180/math.pi)
13         else:
14             return ((2*math.pi)-h)*(180/math.pi)
15     def Calc_S(r,g,b):
```

```
16          Min = np.minimum(np.minimum(r,g),b)
17          s = 1 - (3/(r+g+b+0.001))*Min
18          s = 100*s
19          return s
20      def Calc_I(r,g,b):
21          i = np.divide((r+g+b),3)
22          i = 255*i
23          return i
24      H = np.copy(r)
25      S = np.copy(g)
26      I = np.copy(b)
27      for i in range(Fil):
28          for j in range(Col):
29              H[i][j] = Calc_H(r[i][j],g[i][j],b[i][j])
30              S[i][j] = Calc_S(r[i][j],g[i][j],b[i][j])
31              I[i][j] = Calc_I(r[i][j],g[i][j],b[i][j])
32      HSI = np.copy(RGB)
33      HSI = np.dstack((H,S,I))
34      return HSI
```

El algoritmo propuesto consiste en realizar el cálculo píxel por píxel para obtener la conversión completa. Como se muestra en la línea 11 del código anterior, se utilizó la función condicional `if-else` para aplicar la condición de la ecuación (2.37) dependiendo del valor del ángulo θ. También se utilizaron dos ciclos `for` anidados en la línea 27 y 28 para hacer un barrido de todos los valores de la matriz.

Para la conversión de HSI a RGB se utilizó un algoritmo similar, el cual se muestra en la sección de código 3.8.

Sección de código 3.8: Conversión HSI a RGB utilizando ciclos FOR.

```
1  def HSI2RGB (HSI):
2      h = HSI[:,:,0]*(math.pi/180.0)
3      s = HSI[:,:,1]/100.0
4      i = HSI[:,:,2]/255.0
```

```
    X = (2*math.pi)/3
    Fil = HSI.shape[0]
    Col = HSI.shape[1]
    def Calc_X(s,i):
        return i*(1-s)
    def Calc_Y(h,s,i):
        y = i*(1+(s*math.cos(h))/math.cos((math.pi/3)-h))
        return y
    def Calc_Z(i,x,y):
        return 3*i-(x+y)
    r = np.copy(h)
    g = np.copy(s)
    b = np.copy(i)
    for j in range(Fil):
        for k in range(Col):
            if h[j][k] < X:
                b[j][k] = Calc_X(s[j][k],i[j][k])
                r[j][k] = Calc_Y(h[j][k],s[j][k],i[j][k])
                g[j][k] = Calc_Z(i[j][k],b[j][k],r[j][k])
            elif X <= h[j][k] < 2*X:
                h[j][k] = h[j][k]-X
                r[j][k] = Calc_X(s[j][k],i[j][k])
                g[j][k] = Calc_Y(h[j][k],s[j][k],i[j][k])
                b[j][k] = Calc_Z(i[j][k],r[j][k],g[j][k])
            elif 2*X <= h[j][k] < 3*X:
                h[j][k] = h[j][k]-2*X
                g[j][k] = Calc_X(s[j][k],i[j][k])
                b[j][k] = Calc_Y(h[j][k],s[j][k],i[j][k])
                r[j][k] = Calc_Z(i[j][k],g[j][k],b[j][k])
            else:
                r = 0
                g = 0
                b = 0
    RGB = np.copy(HSI)
```

```
39     RGB = np.dstack((r,g,b))
```

Una vez definidas las funciones para la conversión RGB a HSI y viceversa, se creó el programa para procesar las imágenes en el espacio HSI modificando las líneas mencionadas de la sección de código 3.6, quedando el programa como se muestra en la sección de código 3.9.

Sección de código 3.9: Mejora de contraste con el espacio HSI.

```
1 Original = io.imread("/01_dr.jpg")
2 hsi = RGB2HSI(Original)
3 Canal_I = hsi[:,:,2]/255.0
4 std = np.std(ChI)
5 x = fisSD(stdI)
6 Canal_Mejorado = fisImg(X, Canal_I)
7 hsi[:,:,2] = ChI_ace*255.0
8 rgb = HSI2RGB(hsi)
```

Para evaluar el desempeño en este espacio de color se procesaron las mismas 45 imágenes, sin embargo, el tiempo de procesamiento fue aproximadamente de 8 minutos por cada una, siendo considerablemente alto a comparación del espacio anterior, cuyo tiempo de procesamiento fue menor a 1 minuto por imagen. Este tiempo alto es debido a que las imágenes utilizadas tienen una dimensión de 3504×2336 píxeles y, con el algoritmo utilizado, se realizan todas las operaciones píxel por píxel, es decir, $8,185,344$ veces.

Tomando en cuenta que el desarrollo de estos programas se realizó en la PC con un procesador más potente que es de la Raspberry, si el tiempo de procesamiento es considerablemente alto, en la Raspberry será aún mayor, por lo tanto se desarrolló un nuevo algoritmo para optimizar la conversión utilizando operaciones matriciales en lugar de ciclos `for`, ya que la organización de los píxeles está dada precisamente como matrices, aprovechando así las funciones de la biblioteca Numpy.

En la sección de código 3.10 se muestra el desarrollo de la nueva función para pasar del espacio RGB al HSI y en el código 3.11 la conversión inversa.

Sección de código 3.10: Conversión RGB a HSI con matrices.

```
def RGB2HSI (RGB):
    rgb = np.float32(RGB)/255.0
    r = rgb[:,:,0]
    g = rgb[:,:,1]
    b = rgb[:,:,2]
    Fil = rgb.shape[0]
    Col = rgb.shape[1]
    def Calc_H(r,g,b):
        Num = 0.5*((r-g)+(r-b))
        D1 = (r-g)**2
        D2 = (r-b)*(g-b)
        Den = np.sqrt(D1+D2)
        Den = np.where(Den==0, Den+0.000001, Den)
        h = np.arccos(Num/Den)
        h = np.where(b<=g, h*(180/np.pi),((2*np.pi)-h)*(180/np.pi))
        return h
    def Calc_S(r,g,b):
        Min = np.minimum(np.minimum(r,g),b)
        s = 1 - (3/(r+g+b+0.000001))*Min
        s = 100*s
        return s
    def Calc_I(r,g,b):
        i = np.divide((r+g+b),3)
        i = 255*i
        return i
    hsi = np.zeros((Fil,Col,3))
    hsi[:,:,0] = Calc_H(r, g, b)
    hsi[:,:,1] = Calc_S(r, g, b)
    hsi[:,:,2] = Calc_I(r, g, b)
    return hsi
```

Como se observa en la sección de código 3.10, no se utilizan los ciclos `for` ni la condicional `if-elif`, en su lugar se toman las matrices como variables para operar directamente sobre ellas.

De esta manera es posible sumar, restar, multiplicar y dividir las matrices, tomando en cuenta que para la multiplicación y división de matrices en Python no se siguen las reglas del álgebra lineal, sino que se realizan elemento por elemento correspondiente como en la suma o resta. Cabe destacar que esto también permite utilizar otras funciones como los comparadores en las líneas 13 y 15, donde en este caso particular, el comparador de la línea 13 se utilizó para evitar el error matemático de tener un denominador cero en una división.

Sección de código 3.11: Conversión HSI a RGB con matrices.

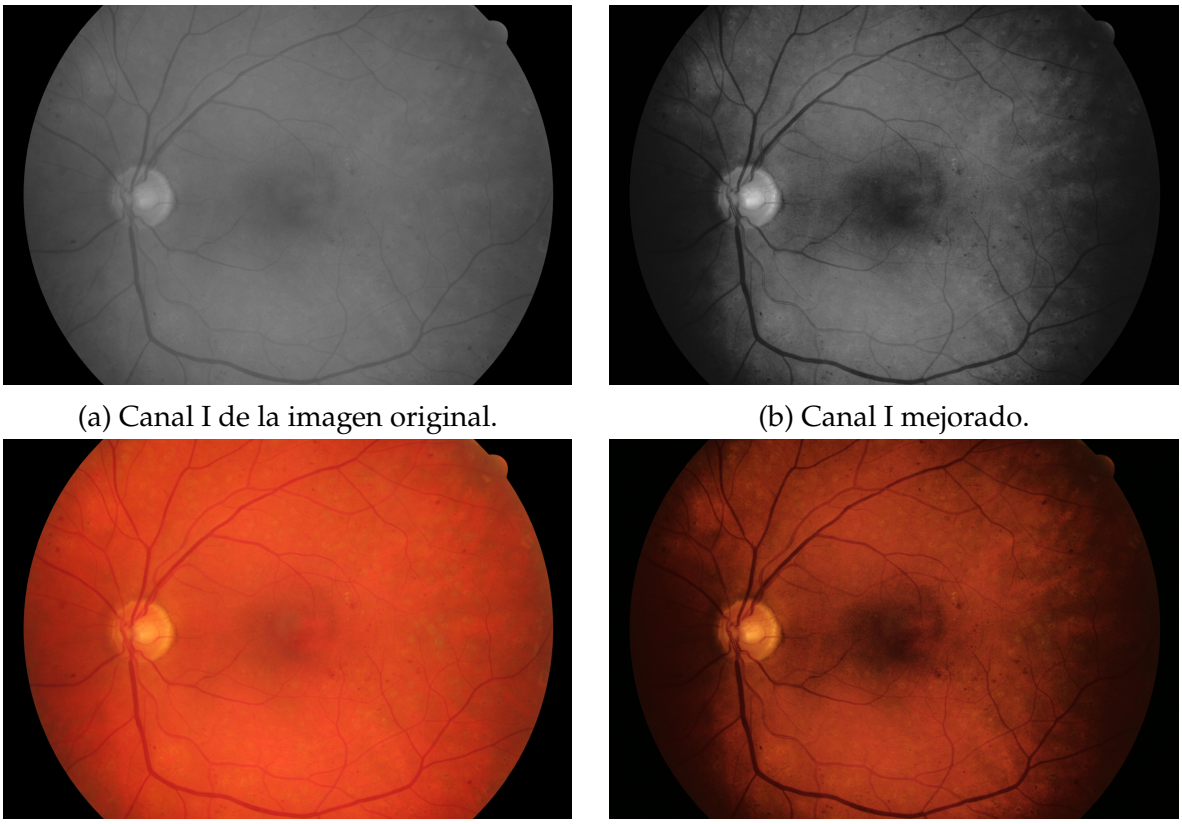
```
1  def HSI2RGB (HSI):
2      h = HSI[:,:,0]*(np.pi/180.0)
3      s = HSI[:,:,1]/100.0
4      i = HSI[:,:,2]/255.0
5      Comp = (2*np.pi)/3
6      Fil = HSI.shape[0]
7      Col = HSI.shape[1]
8      def Calc_X(s,i):
9          return i*(1-s)
10     def Calc_Y(h,s,i):
11         Num = s*np.cos(h)
12         Den = np.cos((np.pi/3)-h)
13         Den = np.where(Den==0, Den+0.000001, Den)
14         return i*(1+Num/Den)
15     def Calc_Z(i,x,y):
16         return 3*i-(x+y)
17     rgb = np.zeros((Fil,Col,3))
18     Comp_1 = h<Comp
19     Mat_Comp_1 = 1*Comp_1
20     Comp_2 = np.logical_and(Comp <= h, h < 2*Comp)
21     Mat_Comp_2 = 1*Comp_2
22     Comp_3 = np.logical_and(2*Comp <= h, h <3*Comp)
23     Mat_Comp_3 = 1*Comp_3
24     x1 = Calc_X(s,i)
25     y1 = Calc_Y(h,s,i)
26     z1 = Calc_Z(i,x1,y1)
```

```
    x2 = Calc_X(s,i)
    y2 = Calc_Y((h-Comp),s,i)
    z2 = Calc_Z(i,x2,y2)
    x3 = Calc_X(s,i)
    y3 = Calc_Y((h-2*Comp),s,i)
    z3 = Calc_Z(i,x3,y3)
    Canal_r = (Mat_Comp_1*y1)+(Mat_Comp_2*x2)+(Mat_Comp_3*z3)
    Canal_g = (Mat_Comp_1*z1)+(Mat_Comp_2*y2)+(Mat_Comp_3*x3)
    Canal_b = (Mat_Comp_1*x1)+(Mat_Comp_2*z2)+(Mat_Comp_3*y3)
    rgb[:,:,0] = Canal_r*255.0
    rgb[:,:,1] = Canal_g*255.0
    rgb[:,:,2] = Canal_b*255.0
    return rgb
```

Con las funciones optimizadas, se volvieron a procesar las 45 imágenes con la sección de código 3.9, obteniendo un tiempo de procesamiento poco mayor a 1 minuto por cada imagen, similar al tiempo de procesamiento utilizando el espacio YCbCr.

En la figura 3.9 se muestra la comparación entre la imagen original y la procesada en el espacio HSI. Al igual que en el espacio de color anterior, la figura 3.9a muestra el canal I original, la figura 3.9b el canal I mejorado, la figura 3.9c la imagen original y la figura 3.9d la imagen mejorada.

De las figuras anteriores se puede observar que, al igual que en el procesamiento anterior, el canal procesado presenta un mayor contraste, pero a comparación del procesamiento en el espacio YCbCr, en la imagen resultante de HSI son más evidentes los signos de la retinopatía y el cambio de tonalidad en los colores de la retina es muy leve, sin embargo, la imagen en general se oscurece, por lo que la tonalidad del color en los vasos sanguíneos, en mácula y en el nervio óptico se tornan considerablemente más oscuras. En consecuencia, si bien los signos de la retinopatía resaltan más que con el procesamiento anterior, no es completamente adecuado.



(a) Canal I de la imagen original.

(b) Canal I mejorado.

(c) Imagen original.

(d) Imagen mejorada.

Figura 3.9: Comparación entre la imagen original y la mejorada en el canal I del espacio HSI.

CIELAB

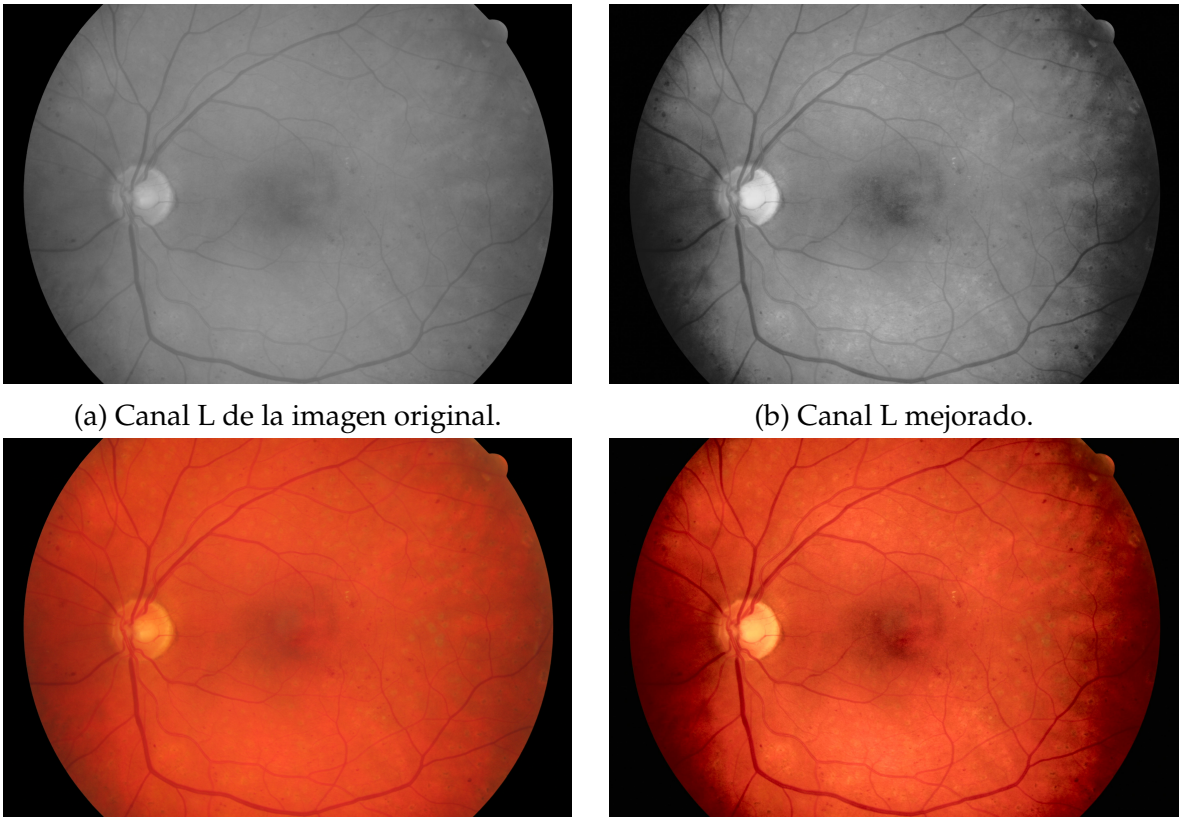
Para el caso del procesamiento en el espacio CIELAB, es posible utilizar las funciones optimizadas para la conversión entre RGB y CIELAB disponibles en la biblioteca skimage, de modo que los cambios con respecto a la sección de código 3.6, solamente serán nuevamente en las líneas 2, 3, 7 y 8, como se muestra en la siguiente sección de código 3.12.

Sección de código 3.12: Mejora de contraste con el espacio CIELAB.

```
1 Original = io.imread("/02_g.jpg")
2 lab = color.rgb2lab(Original)
3 Canal_L = lab[:,:,0]/100
4 std = np.std(Canal_L)
5 x = fisSD(std)
```

```
Canal_Mejorado = fisImg(X, Canal_L)
lab[:,:,0] = ilu_ace*100
rgb = color.lab2rgb(lab)
```

Los resultados de este procesamiento se muestran en la figura 3.10.



(a) Canal L de la imagen original.

(b) Canal L mejorado.

(c) Imagen original.

(d) Imagen mejorada.

Figura 3.10: Comparación entre la imagen original y la mejorada en el canal L del espacio CIELAB.

Al igual que en los dos espacios de color anteriores, visualmente el contraste del canal que contiene la información de la iluminación (canal L) es mayor en la imagen procesada (figura 3.10b), sin embargo, utilizando CIELAB, la imagen resultante (3.10d) no presenta cambios considerables de tonalidad en la retina, en los vasos sanguíneos, en mácula ni en el nervio óptico; aunado a ello, la mejora de contraste permite que los signos de la retinopatía diabética sean considerablemente más visibles, resaltando también las zonas de interés.

Por ello, al cumplir con las tres consideraciones anteriores y al presentar una mejoría visual de contraste, el procesamiento con CIELAB resulta adecuado para la metodología propuesta, coincidiendo con la opinión de los optometristas.

3.4 Evaluación de la metodología

Con la elección de procesar las imágenes con el espacio de color CIELAB, se obtuvo una mejora de contraste apta para las imágenes de fondo de ojo. Sin embargo, para tener la certeza de que el método propuesto de mejora de contraste usando lógica difusa es idóneo, se comparó con dos métodos básicos de mejora de contraste: la ecualización de histograma y una variante de ella.

Como se mencionó en el capítulo anterior, el histograma de una imagen digital muestra la cantidad de píxeles que tienen cada intensidad posible en la imagen. Por ello, para obtener dicho histograma en Python se creó una función que realiza un conteo del número de píxeles haciendo un barrido en toda la matriz del canal "L". Dicha función se muestra en la sección de código 3.13.

Sección de código 3.13: Histograma de un canal en Python.

```
1 def Histo (Canal):
2     Histograma = np.zeros(101)
3     for i in range (Canal.shape[0]):
4         for j in range (Canal.shape[1]):
5             Val = int(Canal[i,j])
6             Histograma[Val] += 1
7     return Histogram
```

Como se observa en la segunda línea de este código, primero se creó un vector con un tamaño de 101 debido a que, como se ha mencionado anteriormente, el canal L* de CIELAB contempla un rango de valores de 0 a 100, posteriormente se utilizaron dos ciclos `for` para hacer el barrido

del canal y obtener cada valor de los píxeles para hacer el conteo. Sin embargo, solo se tomó la parte entera de cada valor para obtener valores discretos.

El algoritmo para realizar la ecualización de histograma en Python se muestra en la sección de código 3.14, donde se siguen los 4 pasos mencionados en el capítulo anterior.

Sección de código 3.14: Ecualización de histograma en Python.

```
def Ecu_Hist (Histograma, Canal):
    Prob = Histograma/(Canal.shape[0]*Canal.shape[1])
    Suma = 0
    Acum = np.zeros(100)
    for k in range(100):
        Suma = Prob[k] + Suma
        Acum[k] = Suma
    Canal_Ecu = np.zeros((Canal.shape[0], Canal.shape[1]))
    for i in range (Canal.shape[0]):
        for j in range (Canal.shape[1]):
            Val = int(Canal[i,j])
            Canal_Ecu[i,j] = Acum[Val]
    return Canal_Ecu*100.0
```

Para el segundo método a comparar, antes de calcular la función de densidad de probabilidad, se divide la imagen en dos sub-imágenes. Este método pretende preservar cierta parte del brillo original de la imagen, por lo que la división depende de la cantidad de brillo que se requiera conservar. Para este trabajo se utilizó la técnica que preserva la media del brillo, conocida como BBHE (del inglés Brightness Preserving Bi-Histogram Equialization [52]).

La división de la imagen se realiza con respecto a su histograma. En BBHE la primera sub-imagen contempla los píxeles desde el valor mínimo hasta el valor promedio y la segunda imagen desde el promedio hasta el máximo. En este contexto, el algoritmo para procesar las imágenes con BBHE consiste en dividir la imagen y aplicar la ecualización de histograma a las dos sub-imágenes. En la sección de código 3.15 se muestra la creación de este procedimiento en Python.

Sección de código 3.15: BBHE en Python.

```
def Ecu_BiH (Histograma, Canal):
    Promedio = int(np.mean(Canal))
    H1 = Histograma[0:Promedio+1]
    H2 = Histograma[Promedio+1:100]
    Pro1 = H1/np.sum(H1)
    Pro2 = H2/np.sum(H2)
    Acum1 = np.cumsum(Pro1)
    Acum2 = np.cumsum(Pro2)
    f1 = Promedio*Acum1
    f2 = (Promedio+1)+(99-(Promedio+1))*Acum2
    Union = np.concatenate((f1,f2), axis=0)

    Canal_Ecu = np.zeros((Canal.shape[0], Canal.shape[1]))
    for i in range (Canal.shape[0]):
        for j in range (Canal.shape[1]):
            ValEq = int(Canal[i,j])
            Canal_Ecu[i,j] = Union[ValEq]
    return Canal_Ecu
```

Para procesar las imágenes con la ecualización de histograma se utilizó un algoritmo similar al mostrado en el diagrama de la figura 3.3, con la diferencia de que se utilizará la ecualización en lugar de los FIS.

En la sección de código 3.16 se muestra el algoritmo en Python para procesar las imágenes con la ecualización de histograma en el canal L de CIELAB, donde la función de la línea 5 es la que se mostró en la sección de código 3.14.

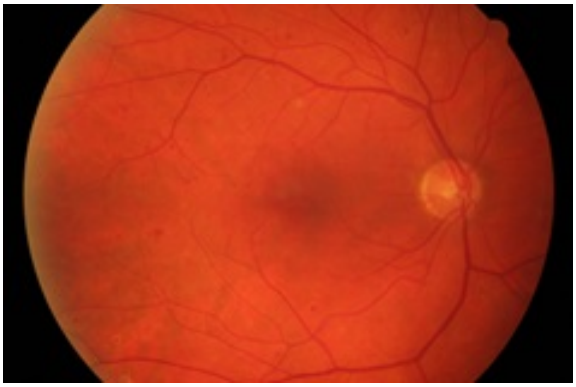
Sección de código 3.16: Ecualización de histograma en el canal L de CIELAB.

```
Original = io.imread("/01_dr.JPG")
lab = color.rgb2lab(Original)
Ilu = lab[:,:,0]
Hist_Ilu = Histo(Ilu)
```

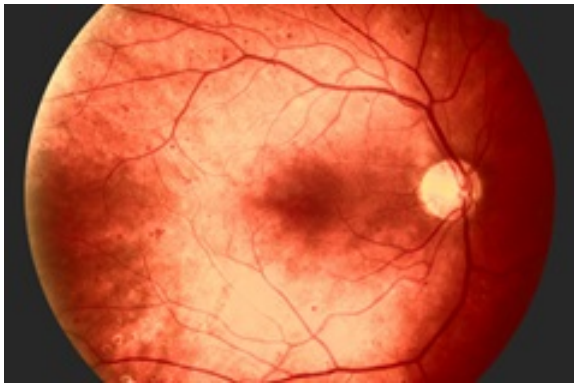
```
5 Ilu_Ecu = Ecu_Hist(Hist_Ilu, Ilu)
6 lab[:,:,0] = Ilu_Ecu
7 rgb = color.lab2rgb(lab)
```

El mismo algoritmo mostrado en la sección de código anterior (3.16) es aplicable para de BBHE, con el cambio de la función de la línea 5 por la función descrita en la sección de código 3.15.

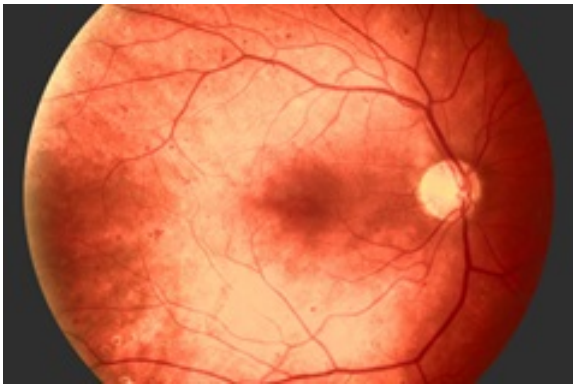
En la figura 3.11 se comparan los tres procedimientos, donde la figura 3.11a muestra la imagen original, la figura 3.11b la imagen resultante de la ecuación de histograma, la figura 3.11c la imagen resultante de BBHE y en la figura 3.11d la imagen resultante del método difuso propuesto.



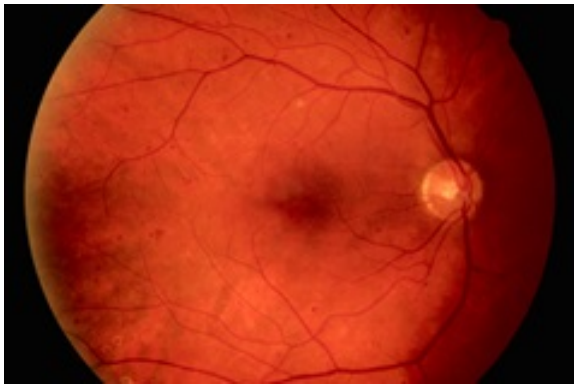
(a) Imagen original.



(b) Ecualización de histograma.



(c) BBHE.



(d) Método propuesto.

Figura 3.11: Comparación entre los tres métodos de mejora de contraste.

Visualmente, es notable que el contraste de las imágenes procesadas con la ecualización de

histograma y con BBHE es considerablemente mayor al del método propuesto, sin embargo, hay un cambio total en la tonalidad, tal como pasaba al procesar las imágenes en el espacio de color YCbCr. Por lo tanto, ninguno de los dos métodos de mejora de contraste modificando el histograma son adecuados para procesar imágenes de fondo de ojo.

En la figura 3.12 se muestran los histogramas del canal L* de las cuatro imágenes de la figura anterior (3.11), con el fin de complementar la información visual del contraste de dichas imágenes.

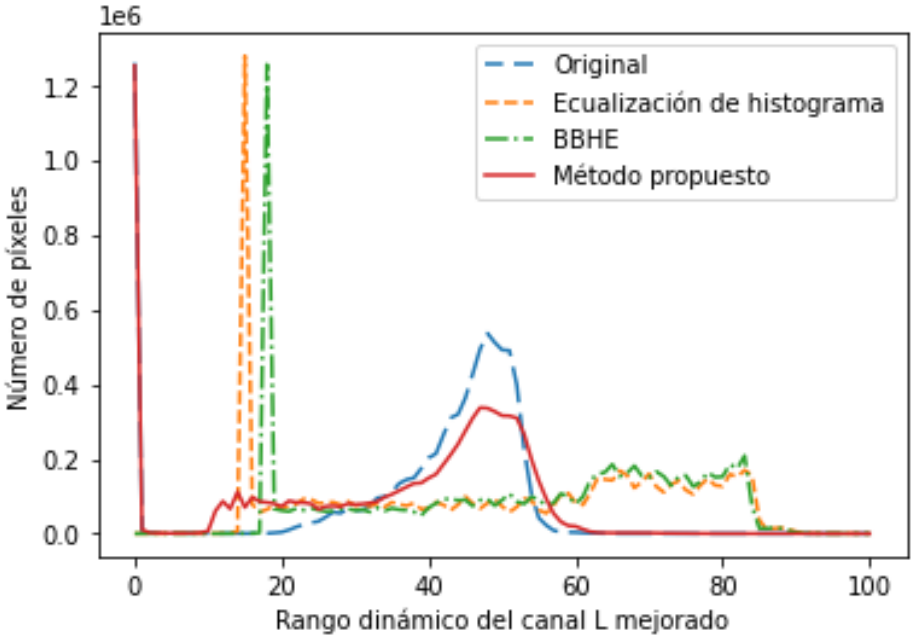


Figura 3.12: Histogramas.

En la figura 3.12 se puede observar que tanto la ecualización de histograma como BBHE aumentan en gran medida el rango dinámico de la distribución de densidad de los píxeles, lo que genera que se presenten las tonalidades claras observadas en la figura 3.11. En cambio, con el método propuesto igualmente se tiene un aumento en este rango dinámico, sin embargo, su histograma mantiene una forma similar a la original, lo que permite que el cambio de tonalidad sea mínimo, aumentando suavemente el contraste.

Visualmente es claro que el procesamiento con el método propuesto cumple con las con-

sideraciones planteadas por parte del personal médico, sin embargo, para poder realizar una evaluación numérica se procesaron las 45 imágenes de la base de datos HRF [65] con los tres métodos y se calcularon dos parámetros de medida de contraste a todas las imágenes procesadas.

El primer parámetro utilizado es conocido como medida del índice de contraste o MCI (del inglés *Measurement of Contrast Index*), el cual es una relación entre la desviación estándar (contraste RMS (2.33)) de la imagen procesada con la original. Dicho de otra manera, el MCI indica si la imagen procesada tiene un mayor o menor contraste que la original. Su cálculo se obtiene por medio de la ecuación (3.3).

$$MCI = \frac{SD_p}{SD_o} \tag{3.3}$$

Recordando del capítulo anterior que la desviación estándar SD (2.33) cuantifica el contraste de una imagen y es conocido como contraste RMS, en la ecuación (3.3) SD_p representa el contraste RMS de la imagen procesada y SD_o el de la original. De acuerdo con este cálculo, si la imagen procesada tiene un mayor contraste el valor del MCI será mayor a 1, en caso contrario su valor será menor a 1. Sin embargo, cuando este valor es muy alto, es un indicativo que la imagen tiene un sobre mejoramiento [78].

El segundo parámetro utilizado es la proporción máxima de señal a ruido conocido como PSNR (por sus siglas en inglés Peak Signal-to-Noise Ratio), el cual es una métrica comúnmente utilizada en telecomunicaciones, específicamente en la transmisión de señales analógicas donde comprimen dichas señales para enviarlas. El PSNR es la relación entre la energía de la señal original y el error cuadrático medio o MSE (por sus siglas en inglés Mean Squared Error) entre la señal original y la comprimida. De esta manera se mide la degradación o pérdida de calidad durante la compresión de la señal al compararlas con la original, evaluando de cierta manera su fidelidad o similitud.

En el contexto del procesamiento de imágenes digitales, el PSNR es utilizado como indicador de la calidad del procesamiento y se calcula comparando los píxeles de la imagen original con

los de la imagen procesada, donde el MSE está dado por la ecuación (3.4).

$$MSE = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} [I_{i,j} - J_{i,j}]^2 \tag{3.4}$$

Como la imagen está representada como matriz, $I_{i,j}$ y $J_{i,j}$ representan los valores del píxel (i, j) de la imagen original y procesada, respectivamente y M y N el tamaño de dicha matriz. Como este parámetro mide el error cuadrático medio, cuanto menor sea su valor se considerará que la calidad del procesamiento es mayor.

El PSNR se expresa en decibeles y se calcula por medio de la ecuación (3.5).

$$PSNR = 10 \log \left(\frac{MAX_I^2}{MSE} \right) \tag{3.5}$$

Donde MAX_I es el valor máximo que puede tomar un píxel. A diferencia del MSE, cuanto mayor sea el valor del PSNR menor será la degradación o pérdida de información, por lo tanto, la calidad del procesamiento se considerará mejor.

Como el procesamiento se realizó sobre el canal L* en el espacio de color CIELAB, ambos parámetros se calcularon sobre ese canal tanto de las imágenes originales como de las procesadas, donde el valor máximo que puede tomar un píxel en la ecuación (3.5) será 100.

Para el cálculo de estos parámetros se creó un programa en Python, donde se definieron las ecuaciones (3.3) y (3.5) como funciones para el cálculo de MCI y PSNR, respectivamente. Posteriormente se aplicaron al canal L de cada imagen original y cada imagen procesada y se graficaron los resultados.

En la sección de código 3.17 se muestra la función creada para el cálculo del MSI y en la figura 3.13 la gráfica resultante de las 45 imágenes de cada método.

Sección de código 3.17: Función para el cálculo del MSI en Python.

```
1 def calculate_mci(ori, pro):
2     mci = np.std(pro)/np.std(ori)
3     return mci
```

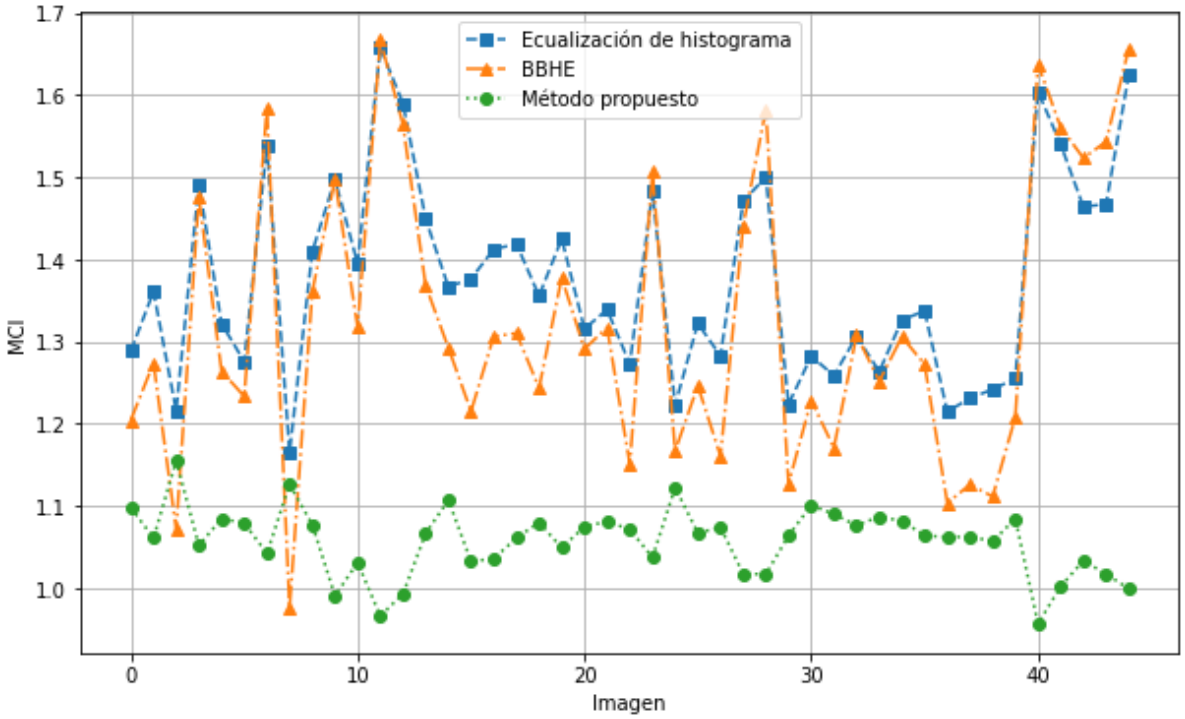


Figura 3.13: MCI de las imágenes procesadas.

De igual manera, en la sección de código 3.18 se muestra la función para el cálculo del PSNR y en la figura 3.14 la gráfica con los resultados.

Sección de código 3.18: Función para el cálculo del PSNR en Python.

```
1 def calculate_psnr(img1, img2, max_value=100):
2     mse = np.mean((np.array(img1, dtype=np.float32) - np.array(img2, dtype=np.
    float32)) ** 2)
3     if mse == 0:
4         return 100
5     return 20 * np.log10(max_value / (np.sqrt(mse)))
```

En la gráfica de la figura 3.13 se puede observar que, en general, el valor del MCI es mayor en los procesamientos donde se modifica el histograma, lo que concuerda con la percepción visual de que estas imágenes presentan un mayor contraste, sin embargo, se ha mencionado que para este tipo de imágenes no es deseable que esta mejora cambie la tonalidad del color ya que se puede llegar a perder información de las zonas de interés. Con el cálculo del PSNR de la

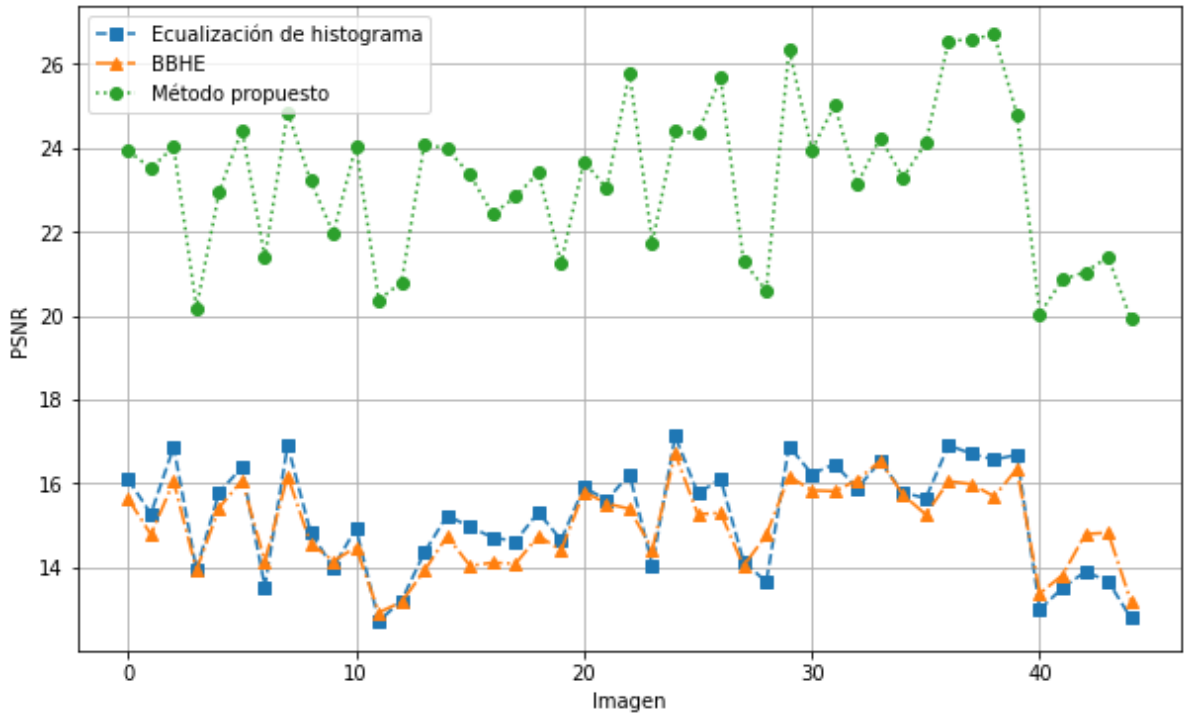


Figura 3.14: PSNR de las imágenes procesadas.

figura 3.14 se puede confirmar que el procesamiento utilizando estos dos métodos, genera esta pérdida de información al tener valores más bajos que con el método propuesto.

Es importante tener en cuenta que el MCI y el PSNR son dos métricas objetivas y no toma en cuenta aspectos perceptuales o subjetivos de la calidad visual. Por ello, aunque visualmente las imágenes procesadas con la ecualización de histograma y BBHE presentan un mayor contraste, no significa que la calidad del procesamiento sea adecuada.

Los resultados de las gráficas anteriores indican que los procesamientos con la ecualización de histograma y con BBHE generan una sobremejora de contraste ya que el MCI es considerablemente alto en ambos métodos y, por lo tanto, la calidad del procesamiento es bajo, ya que hay pérdida de información al tener un PSNR bajo. En cambio, las imágenes procesadas con el método propuesto presentan un mayor contraste que las originales sin tener valores tan altos de MCI, lo que indica que hay una mejora de contraste ya que la pérdida de información es baja, tal como se observa con los valores del PSNR, que son considerablemente más altos que con los

otros dos métodos.

Estos resultados concuerdan con los aspectos visuales que se puede observar en las imágenes procesadas, tomando en cuenta las tres consideraciones y los comentarios de los optometristas. Dicho de otra manera, los resultados objetivos coinciden con los subjetivos, por lo que es posible decir que el método propuesto es ideal para mejorar el contraste en las imágenes de fondo de ojo, ya que resalta los signos de la retinopatía diabética con una muy baja pérdida de información.

3.5 Conclusiones del capítulo

En este capítulo se presentó el desarrollo en Python de un sistema de procesamiento de imágenes para mejorar el contraste de las imágenes de fondo de ojo utilizando la lógica difusa.

Para realizar correctamente este procesamiento se tomaron en cuenta tres consideraciones realizadas por personal médico especializado, las cuales indican que se debe preservar las tonalidades del color y no deformar las zonas de mayor interés. Para lograr esto, la metodología propuesta consiste en utilizar dos sistemas de inferencia difusos, donde el primero tiene la tarea de evaluar su calidad para determinar la cantidad de mejora que el segundo aplicará.

Con el fin de evitar la alteración en los colores de las imágenes, se planteó utilizar un espacio de color donde la información de la luminosidad de la imagen esté separada de la tonalidad del color para poder modificar solo este canal y dejar intacta la información del color. De manera experimental, se determinó que el espacio CIELAB es idóneo, ya que con éste se logra mejorar el contraste considerablemente sin tener pérdidas significativas de información, a diferencia de los otros.

Para poder evaluar el desempeño del método propuesto de una manera objetiva, se calculó el MCI y el PSNR de las imágenes procesadas y se compararon con otros dos métodos de mejora de contraste, obteniendo como resultado que, si bien los otros dos métodos presentan visualmente

un mayor contraste, la calidad del procesamiento del método desarrollado es mayor, por lo que se concluye que la metodología propuesta es efectiva para mejorar el contraste de las imágenes de fondo de ojo.

Capítulo 4
CNN - Aprendizaje Profundo

Como se mencionó anteriormente, el desarrollo experimental de esta tesis consta principalmente de dos partes: la mejora de contraste usando la lógica difusa (presentada en el capítulo anterior) y la extracción y análisis de las características de las imágenes para implementar el sistema embebido, la cual se abordará en el presente capítulo.

Las redes neuronales, aparte de ser una técnica de *soft computing*, también forman parte de un campo de la inteligencia artificial conocida como aprendizaje máquina o *Machine Learning*, por su nombre en inglés. Básicamente el *Machine Learning* implica el desarrollo de modelos computacionales y algoritmos que permitan que una computadora pueda realizar tareas específicas sin ser programadas explícitamente. En otras palabras, en lugar de seguir instrucciones programadas como en la computación clásica, los sistemas de *Machine Learning* utilizan datos para "aprender" y mejorar, por ello, se considera que son capaces de aprender con base en la experiencia.

El objetivo de los modelos de *Machine Learning* es analizar y generalizar información a partir de un gran número de datos de entrada para realizar tareas de clasificación, regresión, agrupamiento, reconocimiento de patrones, etc. Por ello, las redes neuronales son una parte fundamental de estos modelos.

En el campo del procesamiento de imágenes, se han desarrollado modelos y algoritmos de *Machine Learning* para realizar tareas como reconocimiento facial, diagnóstico de enfermedades médicas a través de imágenes, visión artificial para control, inspección, seguridad o robots, entre

otras. Para ello, comúnmente se utilizan las redes neuronales convolucionales para reconocer objetos y patrones, detectar bordes, segmentar y clasificar imágenes, mejorar las imágenes, etc.

La red neuronal convolucional utilizada en esta tesis tiene el objetivo de analizar imágenes de fondo de ojo para discernir aquellas con retinopatía diabética de las saludables, para ello, se decidió utilizar un modelo de *Machine Learning* con una CNN como componente principal.

Actualmente, existe un gran número de modelos basados en CNN, donde muchos son de código abierto, es decir, que el acceso a su código fuente está disponible y se puede modificar y distribuir libremente, lo que fomenta la colaboración en su desarrollo, mejorándolo continuamente. Por lo general, estos modelos están pre-entrenados con grandes conjuntos de datos y pueden servir como punto de partida para adaptarlos con el entrenamiento para tareas específicas.

En Python, bibliotecas y frameworks como PyTorch y TensorFlow integran funciones que permiten cargar y utilizar algunos de estos modelos, facilitando la implementación y el desarrollo de aplicaciones, por ello para este trabajo de tesis se decidió utilizar uno de estos.

Entre los numerosos modelos para clasificación de imágenes que se pueden cargar con bibliotecas y frameworks se encuentran las ya mencionadas AlexNet y ResNet, pero también otros modelos como MobileNet [79], DenseNet [80] y YOLO (por su acrónimo en inglés You Only Look Once que significa "solo observas una vez") [81]. Todos ellos han demostrado tener una efectividad aceptable, como en [82] donde Jung y colaboradores comparan varias versiones de ResNet, AlexNet y YOLO, junto con otros modelos, para clasificar y localizar vehículos en imágenes, obteniendo como resultado una precisión un poco mayor con ResNet. Otro ejemplo se observa en [83] donde se compara el desempeño de AlexNet, Faster R-CNN y YOLOv4 al clasificar imágenes de resonancia magnética cerebral con Alzheimer, obteniendo una precisión del $99\,\%$ con YOLOv4 y AlexNet. Así como estos dos ejemplos, existen numerosos trabajos donde se comparan varios modelos y, en muchos de ellos, los desempeños son similares.

Para clasificar las imágenes de fondo de ojo, en este trabajo se decidió utilizar la quinta

versión del modelo YOLO, conocido como YOLOv5. En general YOLO, destaca por su rapidez, eficiencia y ligereza, a comparación de otros modelos de detección de objetos y clasificación, permitiendo la detección de múltiples objetos en tiempo real.

Actualmente, YOLOv9 es la versión más reciente de este modelo, sin embargo, al momento de realizar el desarrollo experimental lo era YOLOv5, cuyo algoritmo fue considerado por varios autores como el estado del arte en ese momento, por lo que este último fue utilizado para clasificar las imágenes de fondo de ojo.

Para la elección del modelo a utilizar se consideró tanto su adaptabilidad, eficiencia y velocidad, así como las mejoras y actualizaciones que han tenido, ya que cada nueva versión de YOLO es una evolución de la versión anterior, donde se modifican tanto su arquitectura neuronal como algunos algoritmos de optimización para mejorar la velocidad y precisión y, en algunas versiones, reduciendo también su tamaño, haciéndolas más ligeras y, por lo tanto, adaptables.

4.1 YOLO

YOLO es un modelo para la detección de objetos en tiempo real, fue desarrollado por Joseph Redmon et al. y presentado por primera vez en el 2015 con el artículo "You Only Look Once: Unified, Real-Time Object Detection"[81].

A diferencia de modelos anteriores como R-CNN [84], donde utilizan múltiples etapas de procesamiento, YOLO realiza la detección de objetos al pasar la imagen una sola vez a través de una red neuronal convolucional —de ahí su nombre—. Esto se refiere a que la mayoría de modelos para la detección de objetos abordaban la tarea de detección en dos pasos: clasificación y localización. Es decir, primero se clasificaba cada región de la imagen para detectar si había un objeto y, de ser así, determinar qué tipo de objeto era, para posteriormente predecir la ubicación del objeto en la región clasificada.

A diferencia de esos modelos, YOLO predice ubicación y clase de los objetos en la imagen directamente en un solo procesamiento, con una CNN, la cual predice tanto la probabilidad de que exista un objeto, la clase del objeto y las coordenadas de una caja delimitadora que resalta el objeto detectado, tal como se muestra en la figura 4.1.

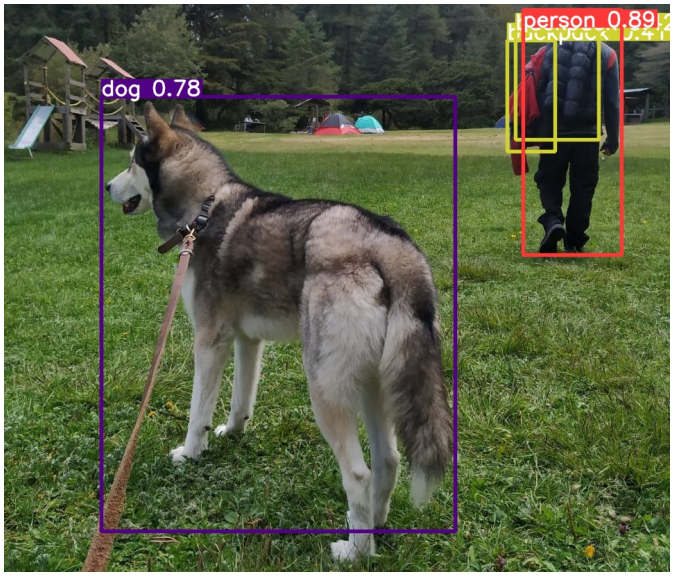


Figura 4.1: Detección de objetos con YOLO.

Procesar la imagen una sola vez, le permitió a YOLO tener una mayor eficiencia y, sobre todo, una velocidad considerablemente mayor a los modelos de su tiempo, logrando una latencia inferior a los 25 milisegundos (según los resultados reportados en [81]), logrando mantener una buena precisión procesando hasta 45 imágenes o fotogramas por segundo. Esto propició que YOLO pudiera ser utilizado tanto en el procesamiento de imágenes como de vídeos.

4.1.1 Funcionamiento de YOLO

Previo a la descripción de la arquitectura de YOLO, es conveniente revisar su funcionamiento, el cual puede describirse como un algoritmo que consta de 5 pasos.

Redimensión El primer paso del algoritmo de YOLO consiste en redimensionar las imágenes

de entrada. En la primera versión se utilizaban imágenes con un tamaño 256×256 píxeles para entrenar la red para clasificación y posteriormente se realizaba una redimensión a 448×448 píxeles para la detección. En las versiones posteriores fueron aumentando el tamaño de las imágenes de entrada hasta llegar a 1536×1536 pixeles, sin embargo, lo más común es utilizar imágenes de 640×640 píxeles, como es el caso de YOLOv5.

Bloques residuales Una vez redimensionada la imagen, se divide en una cuadrícula de $S \times S$ celdas, donde cada celda tiene el objetivo de detectar y localizar objetos dentro de ella. A manera de ejemplo, en la figura 4.2 se muestra una imagen dividida con una cuadrícula de 7×7 píxeles.

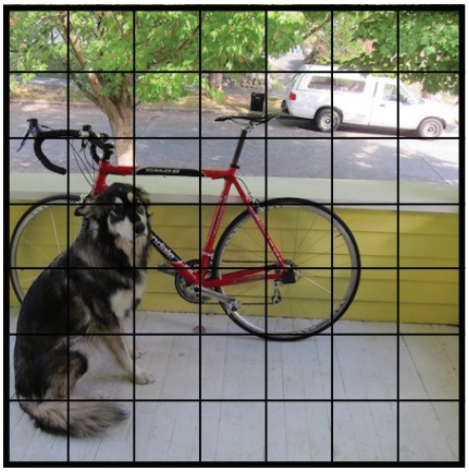


Figura 4.2: Bloques residuales de una imagen (tomada de [81]).

Regresión Cuando se detectan objetos en una celda, se predice un número B de cajas delimitadoras (mejor conocidas por su nombre en inglés *bounding boxes)* con su respectivo nivel de confianza y la probabilidad C de la o las clases detectadas. El número de cajas delimitadoras B por celda depende de la cantidad de objetos que se detecten, si no se detecta ningún objeto el nivel de confianza será cero.

Intersección sobre unión Por cada objeto detectado se generan varias cajas delimitadoras ya que el objeto puede abarcar más de una celda, tal como se observa en la figura 4.3. Para descartar que no son relevantes se aplica la intersección sobre unión o IoU (por sus siglas en inglés *Intersection Over Union*) que es la división entre el área de intersección y el área

de unión. Para ello se establece un valor de umbral y se calcula el IoU de cada celda, resultando solo las cajas delimitadoras que tengan un valor superior al umbral.

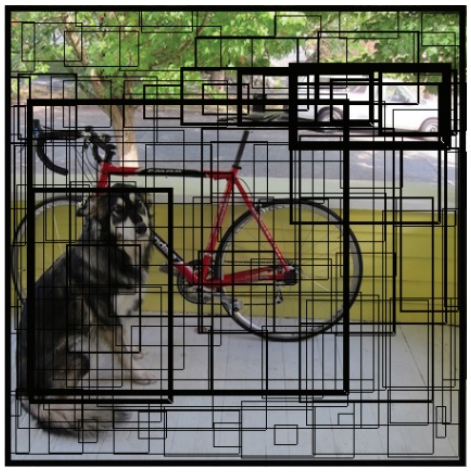


Figura 4.3: Cajas delimitadoras (tomada de [81]).

Supresión no máxima Finalmente se eliminan las múltiples detecciones de un mismo objeto aplicando la supresión no máxima o NMS (*Non-Max Suppression* en inglés), es decir, se eliminan todas excepto la que tenga el mayor nivel de confianza, obteniendo la detección final, como se observa en la figura 4.4.

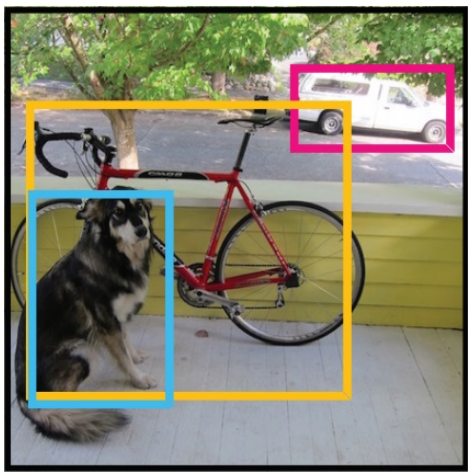


Figura 4.4: Detección final (tomada de [81]).

4.1.2 Arquitectura de YOLO

La arquitectura de la CNN de la primera versión de YOLO (figura 4.5) es similar a la de GoogleNet, la cual está basada en la Darknet [85], igualmente desarrollada por Joseph Redmon.

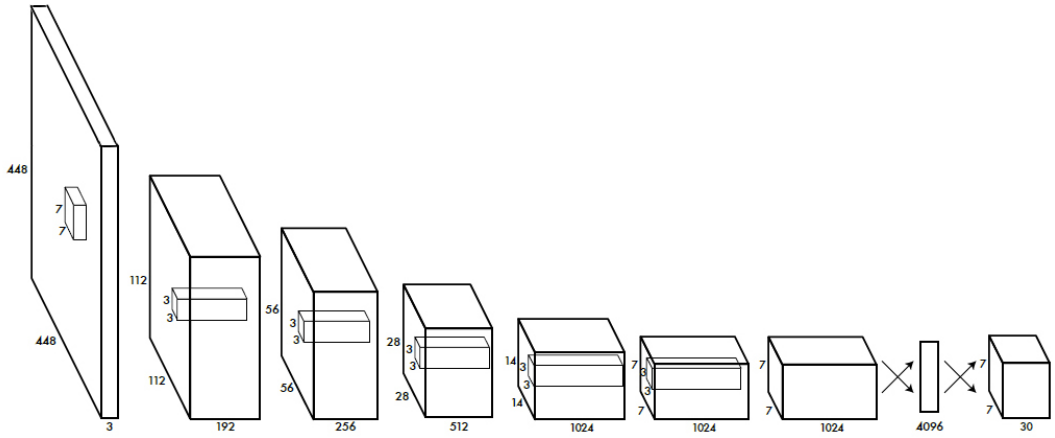


Figura 4.5: Arquitectura de YOLO (tomada de [81]).

Como se observa en la figura 4.5, esta arquitectura consta de 24 capas convolucionales activadas por la función *leaky ReLU* y utiliza 4 capas de agrupación máxima (max pooling), lo que permite reducir la resolución espacial de las características extraídas. Al final de la red se utilizan dos capas *fully connected* conectadas directamente a las características extraídas de las capas convolucionales anteriores; cabe mencionar que la última capa es la única de la red que utiliza una función de activación lineal.

A la salida de la red neuronal se obtendrá un tensor con la información de la o las cajas delimitadoras, la cual se muestra en la ecuación (4.1).

$$Y = \{P_0, T_x, T_y, T_w, T_h, C_1, C_2, \cdots, C_n\} \tag{4.1}$$

Donde el primer valor (P_0) es la probabilidad de que un objeto se detecte en la celda de los bloques residuales. Si en la celda se detectan objetos, P_0 será mayor a cero ($P_0 > 0$) y se crearán un número B de cajas delimitadoras, donde T_x y T_y serán las coordenadas relativas del centro de la posición donde se encuentra el objeto detectado y T_w y T_h representarán el ancho

y alto, respectivamente, de la caja delimitadora. Por otra parte, los valores $C_1, C_2, C_3, \cdots, C_n$ representan la probabilidad de que el objeto detectado pertenezca a la clase 1, 2, 3, etc. En el caso donde no se detecte ningún objeto P_0 será cero y los demás valores del tensor no tendrán importancia.

Para cada caja delimitadora se necesitará un vector de tamaño $(B * 5 + C)$, donde el 5 es debido a los valores requeridos por cada caja delimitadora: la probabilidad P_0, las coordenadas del centro T_x, T_y y el tamaño de la caja T_w, T_h; o sea, los cinco primeros datos de la ecuación (4.1). El tamaño final del tensor T obtenido a la salida de la red neuronal estará dado por la ecuación (4.2).

$$T = S \times S \times (B * 5 + C) \tag{4.2}$$

Función de error

En YOLO se utiliza la función de error de suma de cuadrados, mejor conocida por su nombre en inglés *sum-squared error*. La cual simplemente se calcula sumando los cuadrados de las diferencias entre las predicciones y los valores reales, tal como se muestra en la ecuación (4.3).

$$\mathcal{L}_{SSE} = \sum (y_i - \hat{y}_i)^2 \tag{4.3}$$

Donde y_i es el valor real y $\hat{y}_i$ el valor obtenido de la predicción del modelo.

Para ser más precisos, la función de error de YOLO está compuesta por la suma de varias funciones de error *sum-squared error*, correspondientes a cada parámetro del tensor, o sea, a cada

parámetro de la caja delimitadora. En la ecuación (4.4) se muestra la función de error de YOLO.

$$\begin{aligned}\mathcal{L} &= \lambda_{Coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{Obj}\left[(x_i-\hat{x_i})^2+(y_i-\hat{y_i})^2\right]\\ &+\lambda_{Coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{Obj}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]\\ &+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{Obj}\left(C_i-\hat{C}_i\right)^2+\lambda_{NoObj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{NoObj}\left(C_i-\hat{C}_i\right)^2\\ &+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{Obj}\sum_{c\in clases}\left(p_i(c)-\hat{p}_i(c)\right)^2\end{aligned} \tag{4.4}$$

En la ecuación (4.4), las sumatorias $\sum_{i=0}^{S^2}\sum_{j=0}^{B}$ indican que se realizará el cálculo de las funciones de pérdida de todas las cajas delimitadoras ($j=0$ hasta B) en cada celda ($i=0$ hasta $S\times S$) de los bloques residuales.

Los primeros dos términos de la ecuación (4.4) representan el error de los parámetros correspondientes a la ubicación de la caja delimitadora, donde x_i, y_i y w_i, h_i son las coordenadas y el tamaño, respectivamente, de los valores reales y $\hat{x_i}, \hat{y_i}$ y $\hat{w_i}, \hat{h_i}$ las predecidas por la red. $\mathbb{1}_{ij}^{Obj}$ es una condicional, la cual será 1 si se detecta un objeto en la j-ésima caja delimitadora de la i-ésima celda y será 0 si ningún objeto es detectado, esta condicional indica que solo se calculará el error si en la caja se detecta algún objeto. El parámetro λ_{Coord} es un factor de escala y se utiliza para darle un mayor o menor peso a las pérdidas cuando se detecte un objeto.

Las siguientes dos componentes de la ecuación (4.4) representan el error del nivel de confianza de la caja delimitadora para ambos casos: cuando se detecta un objeto $\mathbb{1}_{ij}^{Obj}$ y cuando no $\mathbb{1}_{ij}^{NoObj}$. El parámetro λ_{NoObj}, también es un factor de escala. El valor por defecto de los factores de escala son $\lambda_{Coord}=5$ y $\lambda_{NoObj}=0.5$, con los cuales se le da un mayor peso a las predicciones de las cajas delimitadoras cuando detectan objetos y poca importancia a las cajas que no contengan objetos.

El último término representa el error de la clasificación, donde la sumatoria $\sum_{c\in clases}$ indica

que se calculará el error de la probabilidad de cada clase si un objeto es detectado en la i-ésima celda.

4.1.3 YOLOv5

Como se mencionó anteriormente, existen varias versiones de YOLO y cada nueva versión introdujo mejoras en el modelo, agregando, en cada una, nuevas técnicas que se iban desarrollando en diversas investigaciones de la comunidad [86]. Existen numerosos trabajos formales como [86] y [87] donde se estudian y explican las diferentes versiones de YOLO. De igual manera se han creado diversos foros y *blogs* en internet como [88] donde se explican de una manera más superficial y, por lo tanto, más amigable.

En este trabajo no se pretende revisar detalladamente cada versión de YOLO, sin embargo, es conveniente mencionar, a grandes rasgos, las mejoras que tuvo cada nueva versión para describir la utilizada en la segunda parte del desarrollo experimental (YOLOv5).

YOLOv1 La descripción anterior del funcionamiento y arquitectura de YOLO corresponde a esta primera versión, la cual se puede resumir de la siguiente manera:

1. Arquitectura con 24 capas convolucionales seguida de dos capas completamente conectadas, todas con una función de activación *leaky ReLU* excepto la última, que utiliza una lineal.
2. Resolución de imágenes de entrada de 256×256 píxeles.
3. Para el *data augmentation* se utilizan técnicas de cambios aleatorios de escala y exposición y saturación en el espacio de color HSV.
4. Como función de error se utiliza *sum-squared error*.

YOLOv2 También conocido como YOLO9000, se presentó en [89] solamente 1 año después de la primera versión. Esta segunda versión introdujo cambios significativos en la arquitectura

de la red, ya que utilizaron una arquitectura completamente convolucional, compuesta por una "columna vertebral" y una "cabeza", donde la primera era la encargada de extraer las características de las imágenes y la segunda de realizar la detección de los objetos.

A diferencia de la primera versión, en YOLOv2 no se utilizaron capas *fully connected* para obtener los parámetros de las cajas delimitadoras, en cambio, la "cabeza", compuesta de 4 capas convolucionales, una capa de reorganización y una de concatenamiento, predice las cajas delimitadoras por medio de *Bounding Boxes* (cajas de anclaje), que son cajas con formas predefinidas. Cada celda se encarga de predecir 5 cajas delimitadoras basadas en 5 cajas de anclaje, las cuales son definidas automáticamente utilizando un algoritmo de agrupación conocido como *k-means* sobre todas las cajas delimitadoras reales del conjunto de entrenamiento. Por último, se asigna la caja delimitadora final a la celda correspondiente por medio de las coordenadas predichas pasándolas por una función *softmax*, obteniendo su tamaño por medio de la caja de anclaje con la que tenga la mayor IoU con la caja delimitadora real.

Otras mejoras con respecto a la primera versión se enlistan a continuación:

1. Aparte de las técnicas para el *data augmentation*, se utiliza una técnica conocida como *Batch Normalization* o normalización por lote en todas las capas convolucionales. Esta técnica normaliza (deja todos los datos a la misma escala) las características obtenidas a la salida de cada capa con el objetivo de mejorar la convergencia del modelo, permitiendo que el entrenamiento sea más rápido y estable.
2. Aumento de la resolución de las imágenes a 448×448 píxeles.
3. Mejoran la función de error ya que con imágenes etiquetadas para la detección se propaga el error completo, pero para la clasificación de imágenes solo se propaga el error de la parte de la arquitectura dedicada a la clasificación.
4. Se agregó una capa de paso para apilar las características extraídas en diferentes canales con el objetivo de poder detectar objetos más finos.

YOLOv3 La tercera versión de YOLO, conocida como YOLOv3 [90], introdujo una arquitectura

mucho más grande con respecto a la versión anterior. La "columna vertebral" (llamada *Darknet-53*) consta de 53 capas convolucionales con normalización de lote y funciones de activación *Leaky ReLU*. En esta arquitectura se reemplazaron las capas de *max pooling* por conectores residuales, los cuales, en pocas palabras, permiten entrenar redes neuronales más profundas al establecer "conexiones de salto" entre entradas de capas convolucionales y salidas de otras capas igualmente convolucionales.

Otra mejora significativa de YOLOv3 es que la "cabeza" permite realizar predicciones multiescala, es decir, predice tres cajas delimitadoras en tres escalas diferentes utilizando, al igual que en YOLOv2, cajas de anclaje previamente definidas con el algoritmo *k-means*.

Otros cambios con respecto a la versión anterior se enlistan a continuación:

1. En lugar de predecir las coordenadas del centro de la caja delimitadora, en YOLOv3 se predice una puntuación de objetivo (mejor conocido por su nombre en inglés *objectness score*) para cada caja delimitadora. Esta puntuación representa la confianza de que una región de la caja delimitadora contenga un objeto y se obtiene utilizando una regresión logística, donde la caja que tenga la mayor IoU con la caja delimitadora real tendrá un valor de 1 y las demás serán 0.
2. Para la clasificación se reemplaza la función $softmax$ por la función entropía cruzada binaria, la cual permite tener varias etiquetas en una sola caja delimitadora.
3. Se agregó a la "columna vertebral" un bloque de agrupamiento piramidal espacial (SPP por sus siglas en inglés de *Spatial Pyramid Pooling*) [91], la cual divide el mapa de características en varios niveles y, posteriormente, aplica *max pooling* a cada una.

YOLOv4 La cuarta versión de YOLO [92] introdujo principalmente cambios en las estrategias de entrenamiento para mejorar el rendimiento, aumentando un poco el coste computacional pero no el tiempo de la inferencia; y se agregaron métodos para mejorar significativamente la precisión, sacrificando un poco el costo de la inferencia.

En cuanto a la arquitectura, es considerablemente más compleja que la anterior. En la columna vertebral se mantuvo la *Darknet-53* con normalización por lotes, sin embargo, se

modificó al utilizar tanto funciones de activación *Leaky ReLU* como la función de activación *Mish*. También se agregaron conexiones parciales entre etapas, conocidas como CSP por sus siglas en inglés *Cross-Stage Partial Conections*, que tienen la función de conectar capas de diferentes etapas de la CNN mejorando el flujo de información para ayudar a reducir el coste computacional manteniendo la precisión.

Después de la columna vertebral, se agregaron varios módulos considerados como el "cuello"SPP de la red. Al igual que en la versión anterior, se agregaron módulos SPP, junto con una versión de la red de agregación de rutas (PANet por sus siglas en inglés) [93], que se implementa como una red paralela.

En la "cabeza" se utilizaron las mismas cajas de anclaje de YOLOv3, por lo tanto también se realizan predicciones multiescala. Por todos los módulos agregados, al modelo completo de YOLOv4 se le conoce como *CSPDarknet53-PANet-SPP*.

Entre otros cambios implementados en YOLOv4, se encuentran los siguientes:

1. La integración de métodos para un entrenamiento avanzado, conocidos como *bag of freebies*. Éstos principalmente incluyen técnicas más novedosas para el *data augmentation*, aparte de las clásicas, entre las cuales destaca el aumento por mosaico, que combina cuatro imágenes en una sola.
2. Con todos los módulos agregados, se genera un gran número de hiperparámetros, los cuales, como se mencionó anteriormente, permiten que el entrenamiento sea óptimo y, por lo tanto, el modelo tenga una mayor eficacia sin modificar la arquitectura de la red. Por ello, en YOLOv4, se utilizaron algoritmos genéticos para encontrar los hiperparámetros óptimos.

Arquitectura de YOLOv5

Actualmente, las arquitecturas de las redes para detección de objetos se componen por la columna vertebral, el cuello y la cabeza, como la arquitectura que se mencionó para YOLOv4.

Como se mencionó anteriormente, la columna vertebral es la encargada de extraer las características de las imágenes y la cabeza de realizar las predicciones. Por su parte, el cuello actúa como un conector entre los otros dos, su objetivo es optimizar las características obtenidas de la columna vertebral, para ello se suelen utilizar capas e incluso redes paralelas.

La quinta versión de YOLO [94] fue desarrollada por Glen Jorcher, fundador de una compañía dedicada al desarrollo de modelos de inteligencia artificial (específicamente de visión por computadora) de código abierto llamada Ultralytics. A diferencia de las versiones anteriores, YOLOv5 se implementa nativamente sobre Pytorch y se incorporan algunas herramientas de Ultralytics, como el algoritmo de preentrenamiento conocido como *AutoAnchor* que, a grandes rasgos, verifica que las cajas de anclaje estén adecuadamente ajustados al conjunto de datos y a la configuración y si no lo están los ajusta automáticamente.

A pesar de que YOLOv5 se lanzó solamente unos meses después de YOLOv4, se incluyeron varias mejoras de ésta última, como la generación de los mejores hiperparámetros utilizando algoritmos genéticos.

Con respecto a la arquitectura, se realizaron cambios considerables. Aunque se utiliza la misma *Darknet-53* como columna vertebral con normalización por lotes seguidas de las conexiones CSP, se utilizaron funciones de activación *SiLU* en lugar de *LeakyReLU*. Además, en la capa de entrada se utiliza una red conocida como *Stem*, que es un módulo de varias capas convolucionales agrupados, que tienen el objetivo de reducir la memoria y el coste computacional reduciendo la cantidad de parámetros y las operaciones sin perder calidad en las características extraídas.

Para el "cuello" de la red se sustituyeron los módulos SPP por SPPF (*Spatial Pyramid Pooling Fast Layer*), que igual son bloques de agrupamiento piramidal espacial pero más rápidos, lo que acelera aún más el tiempo del entrenamiento. Las capas SPPF también permiten procesar las predicciones en tres escalas, como en YOLOv3 y YOLOv4. De igual manera se utilizan redes de agregación de rutas, pero se modificaron las PANet agregando conexiones parciales entre

etapas, es decir CSP, por lo que se conocen como CSP-PAN.

Finalmente, en la "cabeza" de la red se utilizó una estructura similar a la de YOLOv3, por lo que también se realiza la predicción en tres escalas (pequeño, mediano y grande) al igual que en YOLOv3 y YOLOv4.

Debido a que la red de YOLOv5 es considerablemente grande, aún con un diagrama a bloques, no se mostrará en esta tesis, en embargo, se puede encontrar en la página oficial de Ultralytics [95], en donde también se describe la arquitectura completa y otros cambios significativos con respecto a sus antecesores.

Al igual que en su versión anterior, YOLOv5 también agregaron numerosas técnicas para el *data augmentation*, entre las cuales, aparte del aumento de mosaico y de las clásicas, destacan:

Copiar-Pegar Se copian partes aleatorias de varias imágenes y se pegan aleatoriamente en otras.

Mezcla Se crean imágenes nuevas compuestas de dos imágenes con sus respectivas cajas delimitadoras.

Volteado horizontal aleatorio Como su nombre lo indica, solamente se voltean horizontalmente imágenes de manera aleatoria.

Entre otras técnicas de entrenamiento que se agregaron a YOLOv5, aparte de *AutoAnchor* y la mejora de hiperparámetros, se encuentran:

Calentamiento y coseno programado Primero se incrementa gradualmente la taza de aprendizaje durante un determinado número de épocas, desde un valor muy bajo hasta el valor definido previo al entrenamiento. Después del "calentamiento", se reduce nuevamente la taza de aprendizaje, gradualmente por medio de una curva coseno.

Movimiento exponencial promedio Se utiliza el promedio de los parámetros de épocas anteriores para estabilizar el entrenamiento.

Precisión mixta Se utilizan operaciones con una menor precisión durante algunas épocas para reducir el uso de memoria y aumentar la velocidad.

Otra característica destacable de YOLOv5, es que de la misma quinta versión se desarrollaron hasta 10 versiones con respecto a la escala, en primera instancia estaban YOLOv5n (nano), YOLOv5s (pequeño), YOLOv5m (mediano), YOLOv5l (grande) y YOLOv5x (extra grande) —que eran las versiones disponibles al momento de realizar el desarrollo experimental de esta tesis— y, posteriormente se liberó la versión 6 de esos mismos modelos: YOLOv5n6, YOLOv5s6, YOLOv5m6, YOLOv5l6 y YOLOv5x6.

De acuerdo con el autor de YOLOv5, todos los modelos de YOLOv5 son variantes a escala compuesta de la misma arquitectura, la diferencia entre ellas varía en el número de capas dentro de cada componente de la arquitectura y el tamaño de los filtros, variando así la profundidad de la red.

Esta escalabilidad permite que el modelo pueda adaptarse al hardware de cada aplicación. Por ejemplo, en este trabajo, al utilizar una Raspberry Pi 4, los recursos de hardware son considerablemente limitados a comparación de computadoras personales con GPU dedicada, por ello lo conveniente sería utilizar alguno de los modelos más ligeros como YOLOv5n o YOLOv5s. Tomando en cuenta las características de la Raspberry utilizada en esta tesis, se optó por utilizar el modelo YOLOv5s.

La diferencia entre los modelos estándar con respecto a su versión 6, aparte de lo mencionado anteriormente, es que ésta última tiene 4 capas de salida y se entrena con imágenes de entrada con un tamaño de 1280×1280 píxeles, a diferencia la versión estándar que tiene 3 capas de salida y se entrena con imágenes de 640×640 píxeles.

Función de error

Otro punto importante, es que en YOLOv5 se utiliza una función de error total conocida como BCE (por sus siglas en inglés de *Binary Cross-Entropy*) o entropía cruzada binaria ((4.5)).

$$\mathcal{L}_{BCE}(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \tag{4.5}$$

La función de error total (4.6) está compuesta por la combinación de las pérdidas o errores individuales: error de clase, error de puntuación de objetivo (o simplemente error de objeto) y error de localización.

$$\mathcal{L} = \lambda_1 \mathcal{L}_{cls} + \lambda_2 \mathcal{L}_{obj} + \lambda_3 \mathcal{L}_{loc} \tag{4.6}$$

Donde λ_1, λ_2 y λ_3 son factores de escala al igual que en la ecuación (4.4)

El error de clase $\mathcal{L}_{cls}$ simplemente mide el error de la tarea de clasificación, el error de objeto $\mathcal{L}_{obj}$ mide el error de los objetos detectados (o no detectados) en las celdas de los bloques residuales y el error de localización $\mathcal{L}_{loc}$ calcula el error de la localización de los objetos dentro de las celdas de los bloques residuales.

Para el error de clase y de objetividad se utiliza la BCE, sin embargo, el error de objeto también utiliza factores de escala, dependiendo de la escala o tamaño de los objetos detectados: pequeño, mediano o grande. Por defecto, el balance de estos factores es de 4, 1 y 0.4, respectivamente, tal como se muestra en la ecuación (4.7).

$$\mathcal{L}_{obj} = 4.0\mathcal{L}^s_{obj} + 1.0\mathcal{L}^m_{obj} + 0.4\mathcal{L}^l_{obj} \tag{4.7}$$

El error de localización, a diferencia de los anteriores, no utiliza BCE, sino la IoU al igual que YOLOv3 y YOLOv4, la cual se muestra en la ecuación (4.8).

$$IoU(A, B) = \frac{A \cap B}{A \cup B}; \quad AoU(AB) \in [0, 1] \tag{4.8}$$

Donde A es la caja delimitadora obtenida con la red y B la real. Recordando que las cajas delimitadoras tiene los parámetros T_x, T_y, T_w y T_h.

4.2 Deep Learning con YOLOv5

Como se mencionó anteriormente, las redes neuronales artificiales forman parte dos campos de estudio: *soft computing* y *machine learning*. La relación con *soft computing* se basa en que son sistemas computacionales con inspiración biológica y son capaces de resolver problemas complejos adaptándose a datos cambiantes. Con respecto a su relación con *machine learning* se puede resumir afirmando que son su herramienta más importante.

A grandes rasgos, el *machine learning* se puede definir como modelos y algoritmos que permiten que un computador sea capaz aprender a detectar patrones de un conjunto de datos para realizar tareas o generar predicciones sin necesidad de ser programadas explícitamente. En palabras más sencillas, se busca que las computadoras sean capaces de realizar tareas con base en la experiencia. Por ello, es posible afirmar que las ANN son una herramienta fundamental, ya que son capaces de aprender a extraer características y detectar patrones y representaciones complejas a partir de su entrenamiento con un conjunto de datos.

Dentro del área de *machine learning* existe un concepto conocido como *deep learning* o aprendizaje profundo. Previamente se mencionó que una red neuronal se puede considerar "profunda" cuando tiene un gran número de capas en la capa oculta y que esto le permite aprender representaciones complejas y abstracta de los datos. Si bien este tipo de redes se relacionan con *deep learning* a tal grado que son su base, implica más que solo redes profundas y complejas. De manera resumida, *deep learning* implica el aprendizaje de patrones muy complejos y abstractos, por lo que se requiere de redes neuronales con una profundidad considerable, enormes conjuntos de datos y técnicas específicas para entrenar y optimizar este tipo de redes con la gran cantidad de datos, con el fin de lograr que un modelo sea más robusto, es decir que sea potente y flexible al aprender a representar el mundo como una jerarquía anidada de conceptos y representaciones con cada concepto definido en relación con conceptos más simples y representaciones más abstractas calculadas en términos de conceptos menos abstractos [96].

Bajo esta premisa, YOLOv5 es considerado como un modelo de *deep learning* no solo por su arquitectura, ya que es compleja y con un gran número de capas capaces de extraer características y patrones muy complejos y abstractos de las imágenes, sino también porque el modelo incluye un gran número de técnicas de entrenamiento por la gran cantidad de datos con la que se entrena, también incluye algoritmos de optimización para lograr un entrenamiento eficiente.

4.2.1 Metodología

Para el entrenamiento de YOLOv5 —y en general para todas las versiones de YOLO— se requiere de imágenes etiquetadas con los objetos y las ubicaciones de dichos objetos, con las cuales al pasarlas por el modelo se calculará la función de pérdida para medir la diferencia tanto de las predicciones de la red como de las ubicaciones de los objetos. De esta manera, en cada iteración del entrenamiento se actualizan los parámetros de la red para minimizar la función de pérdida.

En el caso particular de este trabajo, se prepararon las imágenes, con las cuales se entrenará el modelo, de forma manual siguiendo la guía proporcionada por Ultralytics para entrenar YOLOv5 [97]. Esta preparación consiste en realizar el etiquetado de los objetos en las imágenes y organizarlas en directorios para entrenamiento y validación. Sin embargo, previo a seguir dichos pasos se realizará el preprocesamiento descrito en el capítulo anterior a las imágenes con las que se entrenará el modelo.

La metodología para el entrenamiento de YOLOv5 de esta tesis, se muestra en la figura 4.6.

4.2.2 Preprocesamiento (mejora de contraste)

De acuerdo con la guía proporcionada por los autores de YOLOv5, se recomienda utilizar por lo menos 1500 imágenes por clase y que el etiquetado sea consistente y preciso. Por esta

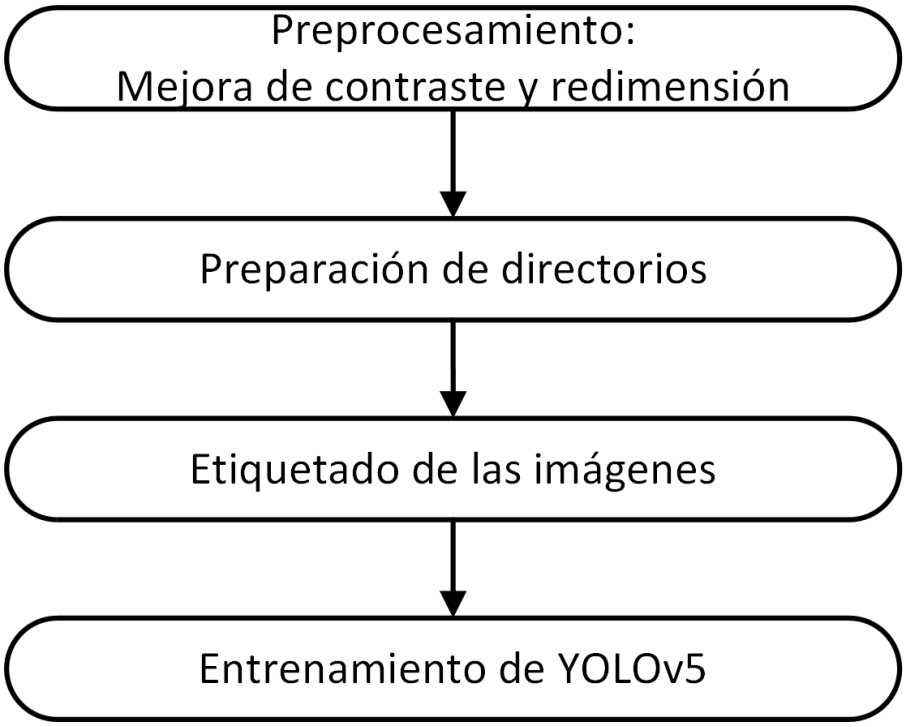


Figura 4.6: Metodología para el entrenamiento de YOLOv5.

razón y con el fin de organizar las imágenes obtenidas de las bases de datos, se dividieron en las dos clases: con retinopatía diabética (`dr`) y saludables (`he`).

Para ello se creó un programa en Python donde, de las bases de datos descargadas, se lee el archivo con extensión `.csv` para identificar la clase de las imágenes y separarlas en un directorio por clase. El archivo `.csv` es proporcionado por los creadores de las bases de datos y se utiliza para identificar la clase de las imágenes almacenando el nombre de la imagen y su clase de forma tabular, donde cada línea representa una fila de datos y los valores de cada columna están separados por comas, de ahí su nombre "Valores Separados por Coma" o *Comma-Separated Values* (CSV) en inglés.

Para el entrenamiento de la YOLOv5, se utilizaron las imágenes de la base de datos de Kaggle [67] ya que contiene más de 53,000 imágenes, donde el primer dato del archivo `.csv` corresponde al nombre de la imagen y el valor separado por coma la clase de la imagen, la cual

puede contener 5 posibles valores donde 0 indica que es una imagen saludable y los demás el grado de la retinopatía diabética.

Como la red se entrenará para detectar aquellas imágenes con retinopatía diabética, el grado de la retinopatía será indistinto, por lo tanto, aquellas con una clase distinta a cero estarán en el directorio retinopatía diabética `Dr`.

En la sección de código 4.1 se muestra el programa creado en Python donde se lee el archivo `csv` y se separan por clases, copiando las imágenes con retinopatía diabética en carpeta `Dr` y las saludables en la carpeta `He`.

Sección de código 4.1: Clasificación de imágenes en Python.

```
os.chdir("/Pictures/Prueba")
copy_dir1 = "/Pictures/He"
copy_dir2 = "/Pictures/Dr"
os.getcwd()
archivoCSV = open("trainLabels.csv")
tipoDato = np.dtype([("label", "U12"), ("value", int)])
data = np.loadtxt(raw_data, dtype = tipoDato, delimiter=",",skiprows=1)
for i in range(3500):
    if data[i][1] == 0:
        shutil.copy2(data[i][0]+".jpeg", copy_dir1)
    else:
        shutil.copy2(data[i][0]+".jpeg", copy_dir2)
```

En la primera parte del código anterior se definen tres directorios: el directorio donde se utilizará el programa, que es donde se encuentran todas las imágenes originales; el directorio en el cual se copiarán las imágenes saludables `He` y el directorio donde se copiarán las imágenes con retinopatía diabética `Dr`. En la segunda parte se lee al archivo `csv` llamado `trainLabels.csv` y en la última parte se copian las imágenes en los directorios correspondientes utilizando un ciclo `for`. Es importante señalar que en el ciclo `for` se utilizó un rango de 3500, con el objetivo de tener por lo menos 1500 imágenes por clase.

Como se mencionó anteriormente, el tamaño de las imágenes utilizadas por YOLOv5 para el entrenamiento es de 640×640 píxeles. Por ello, previo al preprocesamiento con lógica difusa, se redimensionaron las imágenes a utilizar con el objetivo de que el coste computacional del procesamiento sea menor, al tener imágenes de menor dimensión.

Tomando en cuenta que todas las imágenes de fondo de ojo obtenidas tienen una proporción de ancho mayor que la altura de la imagen, se redimensionaron conservando la relación de aspecto, utilizando nuevamente un programa creado en Python cuyo algoritmo se muestra en la sección de código 4.2.

Sección de código 4.2: Redimensión de imágenes en Python conservando la relación de aspecto.

```
data_path = "/Pictures/he"
data_dir_list = os.listdir(data_path)
data_dir_list.sort()
data_dir_list = data_dir_list[0:4000]
img_data_list=[]
for image_file in data_dir_list:
    img_data_list.append('/he/'+image_file)
    ima = io.imread(img_data_list[-1])
    scale = 640/ima.shape[1]
    w = 640
    h = int(ima.shape[0]*scale)
    ima = (transform.resize(ima,(h,w),anti_aliasing=(True))*255).astype('uint8')
    io.imsave(img_data_list[-1], ima)
dataset_df = pd.DataFrame(img_data_list)
dataset_df.head()
```

En la primera parte de este código se accede al directorio donde se encuentran las imágenes y se crea un vector con el nombre de cada una en orden alfabético, en la segunda parte se utiliza un ciclo `for` para redimensionar una por una, definiendo la escala a 640 sobre el ancho de cada una para que el lado más largo sea de dicho tamaño, finalmente se guardan las imágenes redimensionadas sobrescribiendo las que ya se tenían. Como el total de las imágenes a utilizar

están separadas en dos carpetas, el programa se utilizó dos veces, la primera para las imágenes con retinopatía diabética en el directorio `Dr` y la segunda para las saludables en el directorio `He`.

Una vez redimensionadas, se les aplicó el método de mejora con lógica difusa. Para automatizarlo, se creó nuevamente un programa en Python donde se accede a la carpeta donde se encuentran las imágenes y se aplica el algoritmo una por una, por lo tanto, de igual manera se utilizó el programa en ambas carpetas. En la sección de código 4.3 se muestra el fragmento del programa donde se aplica el procedimiento a cada imagen.

Sección de código 4.3: Preprocesamiento de las imágenes de fondo de ojo.

```
filenames = next(walk("/Imagenes/test_02_red_pro/he"), (None, None, []))[2]
filenames.sort()
for i in range(1500):
    ori = io.imread("/Imagenes/test_02_red_pro/he/" + filenames[i])
    lab = color.rgb2lab(ori)
    ilu = lab[:,:,0]/100
    stdI = np.std(ilu)
    stdO = fisSD(stdI)
    ilu_ace = fisImg(stdO, ilu)
    lab[:,:,0] = ilu_ace*100
    rgb = color.lab2rgb(lab)
    io.imsave("/Imagenes/test_02_red_pro/he/" + filenames[i], rgb)
```

En la primera parte de este código nuevamente se crea un vector ordenado alfabéticamente con los nombres de todas las imágenes de la carpeta y en la segunda parte se procesa una por una utilizando un ciclo `for` y se guarda sobrescribiendo la imagen sin procesar.

4.2.3 Preparación de directorios

Para el entrenamiento de YOLOv5 se requiere de un archivo con extensión `yaml` donde se definen tanto los directorios donde se encontrarán las imágenes que se utilizarán para el

entrenamiento y la validación del modelo, como las clases a detectar. En la sección de código 4.4 se muestra el contenido del archivo llamado `customdata.yaml` donde se definen dichos parámetros para esta tesis.

Sección de código 4.4: Parámetros del archivo yaml.

```yaml
path: ../train_data  # Directorio raiz del conjunto de datos
train: ../train_data/images/train/  # Directorio de las imagenes de entrenamiento
val: ../train_data/images/val/  # Directorio de las imagenes para validacion
test:  # Imagenes para test (opcional)
# Classes
names:
  0: dr # Retinopatia diabetica (Diabetic Retinopathy)
  1: he # Saludables (Healty)
```

En las primeras tres líneas de la sección de código anterior se puede observar la organización de los directorios para el entrenamiento y validación, los cuales están dentro del directorio raíz llamado `train_data`. También se puede observar que se entrenará para dos clases, donde la clase 0 serán aquellas imágenes con retinopatía diabética y se llamarán `dr` y la clase 1 serán las saludables llamadas `he`.

En la cuarta línea del archivo `yaml` se puede definir el directorio para imágenes de validación, sin embargo éste se realizará localmente con imágenes de las diferentes bases de datos descargadas, por lo tanto no se definió ningún directorio.

4.2.4 Etiquetado de las imágenes

Previo al etiquetado de las imágenes preprocesadas, se organizaron manualmente en las carpetas de entrenamiento y validación dentro del directorio llamado `images`, utilizando un 80 % de las imágenes para el entrenamiento y el 20 % restante para validación. Para las etiquetas se utilizó una organización similar, cambiando el nombre del directorio `images` por `labels`, ya

que el algoritmo de YOLOv5 localiza dichas etiquetas automáticamente de esa manera. En la figura 4.7 se muestra esta organización gráficamente.

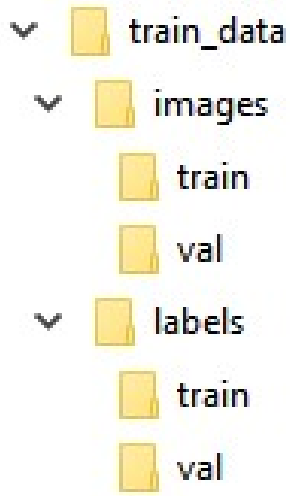


Figura 4.7: Organización de directorios de imágenes y etiquetas.

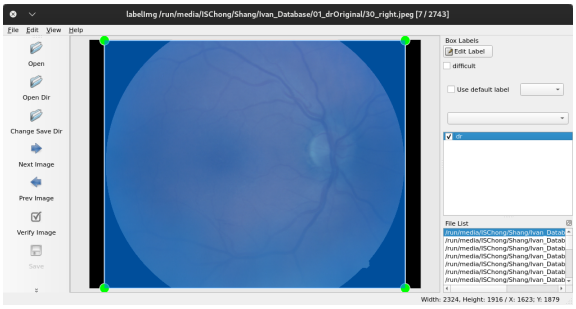
Como se observa en la figura 4.7, el nombre y organización de los directorios son los definidos en el archivo `yaml` mostrado en la sección de código 4.4.

Para el etiquetado se utilizó LabelImge, el cual es un programa de código abierto escrito en Python que permite etiquetar objetos en imágenes con el formato requerido por YOLO. LabelImage facilita la organización de los archivos ya que dentro de sus opciones permite seleccionar el directorio de donde se seleccionarán las imágenes a etiquetar y el directorio donde se guardarán las etiquetas.

Utilizando este programa se etiquetaron manualmente las 3000 imágenes, encerrando la retina completa dentro de una caja delimitadora clasificándola ya sea con la etiqueta de saludable `he` o con retinopatía diabética `dr`.

Las etiquetas requeridas por YOLO utilizan un formato con cinco datos por objeto, que corresponden a la clase y a los cuatro parámetros de la caja delimitadora: coordenadas del centro (T_x, T_y) y tamaño (T_w, T_h). Estos parámetros se guardan en un archivo de texto con extensión `.txt` con el nombre de la imagen. En la figura 4.8 se muestra el etiquetado de una

de las imágenes de este trabajo utilizando el programa LabelImg, donde la figura 4.8a muestra la representación visual de la etiqueta y la figura 4.8b muestra el archivo `*.txt` generado con dicha etiqueta.



(a) Representación visual de la etiqueta en la imagen.

(b) Formato de la etiqueta en el archivo txt generado.

Figura 4.8: Etiquetado de imágenes con LabelImg.

4.2.5 Entrenamiento de YOLOv5

En general, el entrenamiento de redes neuronales requiere de un costo computacional considerable, aunque es variable dependiendo de la configuración y el tamaño del modelo.

Para el caso de YOLOv5 al ser un modelo con una CNN de gran tamaño y complejidad, el entrenamiento requerirá de una gran cantidad de cálculos matriciales tanto en las operación de la convolución como en las demás etapas de la red. Por ello el uso de unidades de procesamiento gráfico (GPU) o de Tensores (TPU) resulta beneficioso.

Las GPU, como su nombre lo indica, están diseñadas para procesar gráficos ya que poseen un gran número de núcleos (hasta miles de núcleos) para realizar cálculos de manera paralela. Por su parte, las TPU se diseñaron específicamente para aplicaciones en el ámbito del *deep learning*, ya que su arquitectura está optimizada para realizar operaciones tensoriales, lo que las hace sumamente ideales para realizar los cálculos requeridos por YOLOv5.

Entre las ventajas de utilizar GPU o TPU destacan las dos siguientes:

Mayor velocidad de entrenamiento Las GPU y TPU son capaces de ejecutar operaciones de manera paralela, con lo cual pueden realizar cálculos con una gran cantidad de datos simultáneamente.

Mayor eficiencia Al estar diseñadas para procesar imágenes y tensores, las GPU y TPU suelen tener una mayor capacidad de memoria dedicada en comparación con las CPU, lo que permite entrenar modelos grandes y trabajar con lotes de datos más grandes sin saturar la capacidad de la CPU.

Aunque en trabajos formales como en [98] y otros informales se ha demostrado que es posible entrenar YOLOv5 en la propia Raspberry Pi 4, resulta poco práctico debido a las limitaciones de su propio hardware. Comúnmente, lo mismo sucede con una computadora personal promedio, ya que se requeriría de una GPU potente —y por lo tanto costosa— para un entrenamiento eficiente. Aparte, por la gran cantidad de imágenes requeridas, se requerirá de una capacidad de memoria RAM considerable.

Como alternativa, existen plataformas de computación en la nube, comúnmente conocidas por su nombre en ingñés *Cloud Computing*, que permiten el acceso a GPUs y TPUs. Entre ellos se encuentran Google Colab, Kaggle Notebooks y Azure Notebooks. Cada una tienen sus propias ventajas y limitaciones, sin embargo, ofrecen los mismos servicios con opciones gratuitas con recursos limitados o con planes de pago con los cuales se puede acceder a mayores recursos y capacidades adicionales.

Para el entrenamiento del modelo utilizado en esta tesis se eligió utilizar Google Colab, la cual utiliza un entorno de desarrollo basado en Jupyter Notebook para crear y ejecutar códigos de Python y R en una máquina virtual. En su versión gratuita los recursos ofrecidos son limitados y variados según la disponibilidad, aunque usualmente ofrecen 12 GB de memoria RAM del sistema, 4 GB de memoria RAM de GPU y 80 GB de memoria en el disco, pudiendo ejecutar los cuadernos por un máximo de 12 horas.

Parámetros de entrenamiento

Debido a que en el entorno de desarrollo en el que se basa Google Colab se ejecutan comandos y programas de Python y R en una máquina virtual, es necesario configurarla previamente. Para el caso de este trabajo basta con seleccionar Python 3 como el entorno de ejecución y utilizar una GPU como acelerador de hardware.

Una vez configurada la máquina virtual ya es posible comenzar con la ejecución de los comandos. El primer paso consiste en obtener YOLOv5, para ello se descargó en la máquina virtual y se instalaron los requerimientos descritos en un archivo proporcionado por los desarrolladores llamado `requirements.txt` con los comandos mostrados en la sección de código 4.5.

Sección de código 4.5: Instalación de YOLOv5 y sus requerimientos en la máquina virtual.

```
1 !git clone https://github.com/ultralytics/yolov5  # clone
2 %cd yolov5
3 %pip install -qr requirements.txt  # install
```

Con el primer comando mostrado en la primera linea de la sección de código 4.5, se clonó el repositorio YOLOv5 disponible en GitHub en directorio raíz de la máquina virtual. Este repositorio contiene el código fuente del modelo y todos los archivos de entrenamiento y prueba necesarios para instalar y usar YOLOv5.

Entre los requerimientos instalados con el comando de la tercera línea, se encuentran bibliotecas como numpy para los cálculos numéricos y de matrices, pillow, opencv y pandas para procesamiento de imágenes.

Con los siguientes comandos, mostrados en la sección de código 4.6, se importa el *framework* PyTorch y un módulo de utilidades para trabajar con YOLOv5. PyTorch proporciona herramientas de entrenamiento e implementación para redes neuronales, además permite poner en funcionamiento YOLOv5.

Sección de código 4.6: Importación del *framework* Pytorch.

```
1 import torch
2 import utils
3 display = utils.notebook_init()  # checks
```

El comando `notebook_init()` mostrado en la tercera línea de la sección de código 4.6, comprueba que se esté trabajando o ejecutando sobre un cuadernillo de Google Colab y realiza configuraciones adicionales, tal como la carga de las bibliotecas y módulos necesarios y la configuración del tamaño de las imágenes y la cantidad de GPU que se utilizarán para que el cuadernillo funcione correctamente, en otras palabras, este comando instala YOLOv5 y realiza las configuraciones necesarias. La respuesta al ejecutar este comando se muestra en la figura 4.9.

```
YOLOv5 🚀 v7.0-306-gb599ae42 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (NVIDIA L4, 22700MiB)
Setup complete ✅ (16 CPUs, 62.8 GB RAM, 28.9/201.2 GB disk)
```

Figura 4.9: Salida del comando notebook_init().

Una vez ejecutado ya es posible utilizar YOLOv5 para entrenar y evaluar modelos de detección de objetos.

Para poder realizar el entrenamiento con los datos personalizados en este cuadernillo, es necesario que se encuentren en la máquina virtual. Para ello se comprimió la carpeta `train_data`, que contiene las imágenes de entrenamiento y validación con sus respectivas etiquetas ya organizadas, en un tipo de archivo `.zip` y se cargó junto con el archivo de configuración `customdata.yaml` dentro del mismo directorio donde se guardaron los archivos de YOLOv5 en la máquina virtual.

Para descomprimir el `zip` cargado, se utilizó el comando mostrado en la sección de código 4.7.

Sección de código 4.7: Descompresión del archivo zip.

```
1 !unzip -q ../train_data.zip -d ../
```

Una vez que se ha configurado la máquina virtual y se han ejecutados los comandos anteriores, se dispone de todos los elementos necesarios para llevar a cabo el entrenamiento de la red. Por lo tanto, el paso inmediato siguiente consiste en ejecutar el programa de YOLOv5 encargado del entrenamiento mediante el comando `!python train.py`.

Este programa permite configurar un total de 38 opciones, utilizadas principalmente para definir las rutas de los archivos de configuración y personalizar los hiperparámetros, o sea los parámetros del proceso de entrenamiento. También es posible configurar un conjunto de opciones adicionales que influyen en diversos aspectos de YOLOv5, como la configuración de la inferencia posterior al entrenamiento o la selección de un dispositivo de cómputo específico.

Todas las opciones disponibles se pueden observar en el código fuente del programa encargado del entrenamiento mostrado [99]. Para el entrenamiento del modelo de este trabajo se configuraron solamente 6 opciones, las cuales se muestran en la sección de código 4.8, donde el comando `!python train.py` ejecuta el programa encargado de realizar el entrenamiento.

Sección de código 4.8: Comando para ejecutar el entrenamiento de YOLOv5.

```
1 !python train.py --img 640 --batch 4 --epochs 200 --data customdata.yaml --
    weights yolov5s.pt --cache
```

Las opciones de entrenamiento configuradas por medio de la sección de código 4.8 se describen a continuación:

img 640 Se indica que se utilizará la resolución nativa de 640 píxeles en las imágenes de entrada para el entrenamiento.

batch 4 Se indica que se utilizará un lote de 4 subconjuntos de imágenes y etiquetas en cada iteración del proceso de entrenamiento.

epoch 200 Se indica que se entrenará con un total de 200 épocas.

data customdata.yaml Se indica la ruta del archivo de configuración `customata.yaml`, que

contiene las rutas de los directorios donde se encuentran las imágenes destinadas para el entrenamiento y validación, así como el número y el nombre de las clases.

weights yolov5s.pt Se indica que se utilizará el modelo YOLOv5-Small con sus pesos pre-entrenados como punto de partida para el entrenamiento.

cache Se activa la opción de control de almacenamiento en caché para acelerar el proceso de lectura y preprocesamiento de datos.

Debido a que no se configuraron las opciones restantes, los demás parámetros de entrenamiento configurables como el optimizador y la paciencia se dejaron en sus valores por defecto.

Como optimizador (para minimizar la función de error), se utiliza el gradiente descendente estocástico o SGD (por sus siglas en inglés *Stochastic Gradient Descent*). La paciencia (conocido como parámetro *patience*) determina el número de épocas necesarias para detener el entrenamiento si no hay una mejora, cuyo valor por defecto es de 100 épocas.

El gradiente descendente estocástico es una variación del gradiente descendente descrito en el capítulo 2. En el algoritmo original se utilizan todos los datos del conjunto de entrenamiento en cada iteración para actualizar los parámetros de la red, por lo que es eficiente para conjuntos de datos relativamente pequeños. En un modelo de *deep learning* como lo es YOLOv5, se suelen utilizar un número muy elevado de datos, que van desde los cientos, miles o hasta millones de datos; por ello, el uso del gradiente descendente como optimizador no resulta práctico en este tipo de modelos, por lo que se han desarrollado varios algoritmos de optimización, entre ellos está el SGD.

La principal idea del SDG es muy similar al original, la principal diferencia es que se utilizan lotes pequeños del conjunto original aleatorios (conocidos por su nombre en inglés *mini-batch*) para actualizar los parámetros de la red, en lugar de utilizar todo el conjunto. Esto permite que el coste computacional del entrenamiento sea menor, es decir, es más eficiente computacionalmente.

El algoritmo del SDG es prácticamente el mismo del original, con su misma ecuación ((2.20)), solamente se requerirá definir un parámetro extra: el tamaño del lote. De esta manera, el primer paso del algoritmo es crear los lotes para que en cada lote, uno por uno, se aplique el algoritmo original del gradiente descendente.

Los hiperparámetros utilizados en YOLOv5, donde se pueden modificar, entre muchos otros, la tasa de aprendizaje, el tamaño del lote (*batch-size*) y las técnicas de aumento de datos y de regularización, se encuentran en archivos de configuración. Existen 6 de estos archivos [100] los cuales están optimizados para el entrenamiento de la red desde cero ya sea con bajas, medias, altas o nulas técnicas de aumento de datos, sin embargo, también es posible modificarlos. Dentro de las opciones configurables se puede seleccionar el que se desea utilizar.

Para el entrenamiento de este trabajo se utilizó el primero: para bajas técnicas de aumento de datos sin modificaciones, el cual se puede observar detalladamente en [101].

Al ejecutar el comando mostrado en la sección de código 4.8, se despliega una descripción del modelo (ver figura 4.10) y, posteriormente, comienza el entrenamiento de la red.

Como se observa en la figura 4.10, el modelo tiene un total de 214 capas en la red neuronal y $7,025,023$ parámetros entrenables y gradientes, los cuales se calcularán durante el entrenamiento. Los 16.0 GFLOPs (por sus siglas en inglés de *Giga Floating Point Operations*) representan una estimación del costo computacional del modelo.

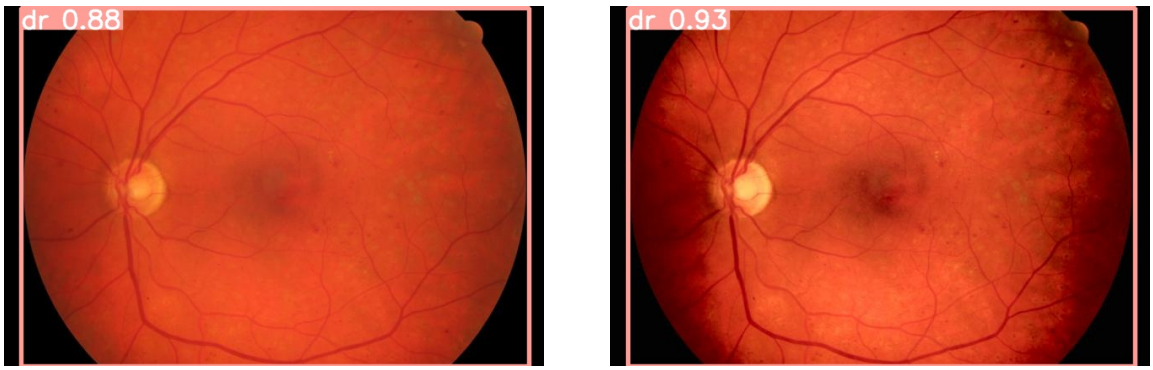
Al finalizar el entrenamiento se generan dos archivos, uno de ellos contiene los parámetros calculados que mejor rendimiento tuvieron y el otro los últimos parámetros calculados. Con cualquiera de ellos, ya es posible realizar la inferencia con el modelo entrenado, la cual permite detectar y clasificar imágenes de fondo de ojo.

Con el objetivo de probar modelo entrenado, se realizó la inferencia en algunas imágenes utilizadas previamente en la prueba del preprocesamiento con lógica difusa. En la figura 4.11 se muestra la inferencia de una imagen de fondo de ojo con retinopatía diabética tanto en su

```
                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12          [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    361984  models.common.C3                        [512, 256, 1, False]
 14                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16          [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  1     90880  models.common.C3                        [256, 128, 1, False]
 18                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  1    296448  models.common.C3                        [256, 256, 1, False]
 21                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  1   1182720  models.common.C3                        [512, 512, 1, False]
 24      [17, 20, 23]  1     18879  models.yolo.Detect                      [2, [[10, 13, 16, 30, 33, 23],
Model summary: 214 layers, 7025023 parameters, 7025023 gradients, 16.0 GFLOPs
```

Figura 4.10: Resumen del modelo YOLOv5.

versión original (4.11a) como en su versión preprocesada (4.11b).

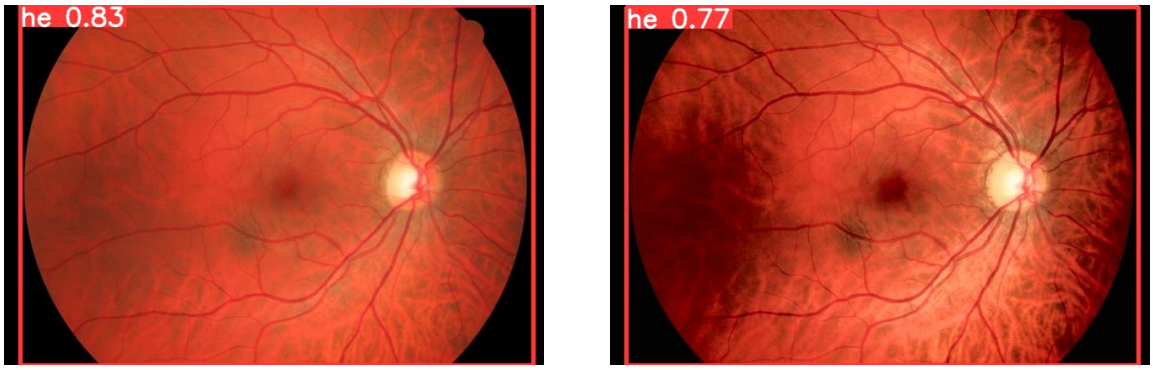


(a) Inferencia en la imagen original. (b) Inferencia en la imagen preprocesada.

Figura 4.11: Inferencia en imagen de fondo de ojo con retinopatía diabética.

De igual manera, en la figura 4.12 se muestra la inferencia en una imagen de fondo de ojo saludable en su versión original (4.12a) y preprocesada (4.12b).

Como se puede observar en la figura 4.11, el modelo detectó con una confianza del 93 %



(a) Inferencia en la imagen original. (b) Inferencia en la imagen preprocesada.

Figura 4.12: Inferencia en imagen de fondo de ojo saludable.

la retinopatía diabética en la imagen preprocesada y con una confianza del 88 % en la imagen original. Por otra parte, en la figura 4.12 se observa que el modelo clasifica con una mayor confianza las imágenes saludables en imágenes originales.

4.2.6 Resultados del entrenamiento

En general, los resultados del entrenamiento de una red neuronal son métricas de evaluación para medir su rendimiento durante y al final del proceso de entrenamiento. Con ellas es posible saber si el modelo, efectivamente, va mejorando a lo largo de cada época e, inclusive, es posible compararlo ya sea con otros modelos o con distintos parámetros de entrenamiento.

Al final del entrenamiento efectuado mediante la sección de código 4.8, se genera un resumen de los resultados, los cuales se muestran en la figura 4.13.

En la figura 4.13 se puede observar que el tiempo de entrenamiento para las 200 épocas fue de 2.134 horas, donde se entrenaron $7,015,519$ parámetros de 157 capas, con un coste computacional de de 15.8 GFLOPs.

Como parte importante de los resultados de entrenamiento se encuentra la evolución del error a lo largo del entrenamiento. Como se mencionó en el capítulo anterior, en YOLOv5 la

```
200 epochs completed in 2.134 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/exp/weights/best.pt, 14.4MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 7/7 [00:07<00:00,  1.05s/it]
                   all        800        800      0.581      0.883      0.711      0.711
                    he        800        400      0.587      0.922      0.679      0.679
                    dr        800        400      0.574      0.843      0.742      0.742
Results saved to runs/train/exp
```

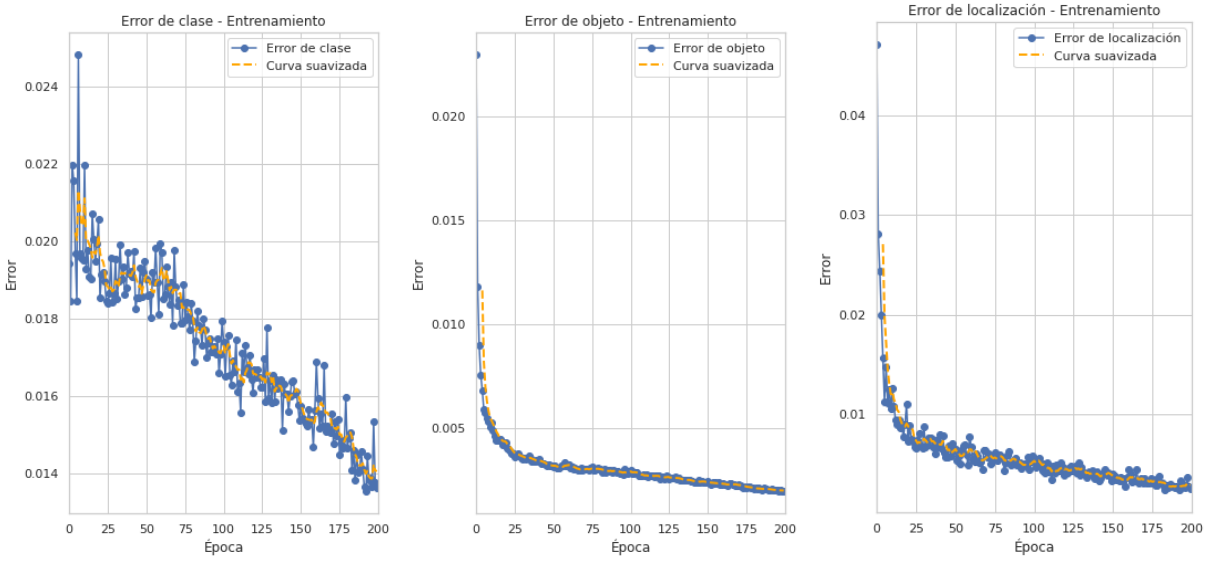
Figura 4.13: Resumen de resultados del entrenamiento.

función de error es la combinación de los errores de clase, objeto y localización, tal como se muestra en la ecuación (4.6). De este modo, en este caso particular, el error de objeto se refiere a la capacidad del modelo en detectar el fondo de ojo en una imagen, el error de localización a la capacidad de localizar adecuadamente el fondo de ojo detectado y el error de clase a la capacidad del modelo en clasificar correctamente la clase del fondo de ojo detectado, ya sea como saludable o con retinopatía diabética.

De acuerdo con lo anterior, es conveniente mostrar la evolución de los errores durante el entrenamiento en gráficas independientes, para observar el desempeño individual en las tres tareas. En la figura 4.14 se muestran las gráficas de los errores durante el entrenamiento con la metodología descrita en el presente capítulo, donde la figura 4.14a muestra el error de clase, la figura 4.14b el error de objeto y la figura 4.14c el error de localización.

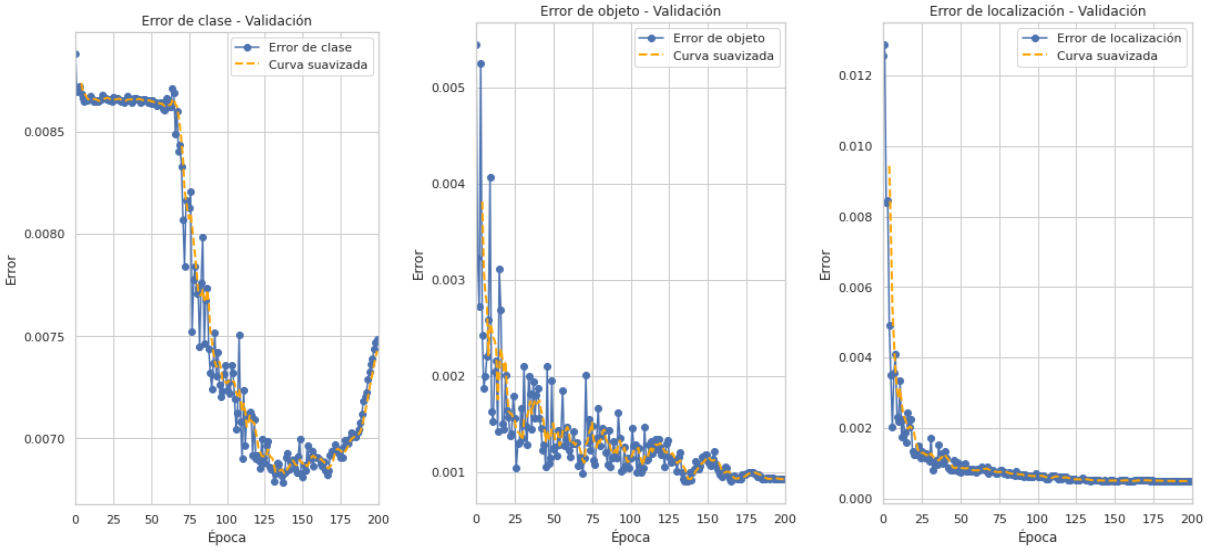
En YOLOv5, estos tres errores también se calculan en las imágenes de validación, los cuales se muestran en la figura 4.15, donde la figura 4.15a muestra el error de calse, la figura 4.15b el de objeto y la figura 4.15c el de localización.

Las gráficas mostradas en las figuras 4.14 y 4.15 muestran una tendencia descendente en los tres errores tanto del entrenamiento como en las imágenes de validación, lo cual indica que el modelo mejoró consecutivamente durante el entrenamiento en la detección, localización y clasificación de las imágenes. Sin embargo, es importante destacar que en el error de clase en las imágenes de entrenamiento (figura 4.14a), a pesar de que si tiene una tendencia descendente, se muestra una fluctuación considerable, donde cada punto por encima de la curva suavizada



(a) Error de clase. (b) Error de objeto. (c) Error de localización.

Figura 4.14: Evolución de los errores durante el entrenamiento.



(a) Error de clase (validación). (b) Error de objeto. (c) Error de localización.

Figura 4.15: Evolución de los errores de validación durante el entrenamiento.

indica que se clasificaron erróneamente más imágenes que en la época anterior. Esto sugiere que le es más complicado al modelo clasificar correctamente la clase que la detección y localización del fondo de ojo en las imágenes.

La fluctuación antes mencionada, se ve reflejada también en el error de clase en las imágenes de validación (figura 4.15a), donde también se observa una tendencia negativa solamente hasta la época 140, aproximadamente, y posteriormente comienza a subir. Esto se puede interpretar como un posible sobreajuste a partir de esa época, sin embargo, aparte de que el error es relativamente bajo, en YOLOv5 se guardan los pesos del entrenamiento tanto de la última época como de la época que mejores resultados tuvo, que en este caso fue de la época 137. Otro detalle a destacar es que en el error de objeto en la validación (figura 4.15b) también se observan fluctuaciones considerables, sin embargo, el valor del error es muy pequeño y la tendencia es descendente.

Criterios y métricas de evaluación

Si bien una tendencia descendiente del error sugiere un entrenamiento positivo, para poder realmente afirmarlo es necesario considerar más métricas de evaluación. Existe una gran variedad de éstas y su uso varía dependiendo del tipo de modelo.

Para modelos de detección de objetos (como YOLOv5) se utilizan principalmente 3 métricas: precisión, *recall* (cuya traducción al español se interpreta como "exhaustividad") y promedio de precisión media (mAP por sus siglas en inglés de *mean average precision*). La interpretación de cada una muestra una perspectiva diferente del rendimiento del modelo, y en conjunto (incluyendo la evolución de los errores) permiten obtener una comprensión general de su rendimiento.

Previo a la descripción de dichos parámetros, es necesario mencionar que su cálculo se obtiene a partir de la cantidad de resultados correctos e incorrectos que detecta el modelo, los

cuales se definen a partir de sus 4 posibles resultados: verdadero positivo (VP), falso positivo (FP), verdadero negativo (VN) y falso negativo (FN) [102].

VP Indica que el modelo realizó una detección correcta, es decir, detectó correctamente que un objeto se encontraba presente en la imagen.

FP Indica que se realizó una detección incorrecta en el sentido de que detectó un objeto cuando en realizad no existía en la imagen.

VN Indica que, correctamente, no se detectaron objetos en la imagen.

FN Indica que el modelo, incorrectamente, no detectó objetos en la imagen cuando si existía un objeto en ella.

Con esto, es posible describir las métricas de evaluación de la siguiente manera:

Precisión Mide la proporción de detecciones correctas realizadas por el modelo, dicho de otra manera, permite medir la cantidad de veces que se hace una predicción correcta. Matemáticamente se define como la relación entre los verdaderos positivos y el número total de predicciones positivas, tanto correctas como incorrectas, como se observa en la ecuación (4.9).

$$P = \frac{VP}{VP + FP} \tag{4.9}$$

Recall Mide la habilidad del modelo para detectar los objetos reales, ya que se calcula la proporción entre los verdaderos positivos y el número total de verdaderos positivos junto con los falsos negativos, tal como se muestra en la ecuación (4.10).

$$R = \frac{VP}{VP + FN} \tag{4.10}$$

Esta métrica es muy significativa en ciertas aplicaciones, por ejemplo, en el caso de la aplicación de un modelo de *machine learning* para el diagnóstico de imágenes médicas, como es el caso de esta tesis, es deseable un alto valor de este parámetro ya que se

minimizaría el número de FN. Los falsos negativos en este tipo de imágenes implicarían un posible mal diagnostico al clasificar como saludable a una imagen con alguna patología [103].

mAP Permite cuantificar la precisión de un detector. A diferencia del parámetro de precisión, el mAP calcula el promedio de la precisión de todas las clases a diferentes niveles de confianza. En palabras sencillas, mide la robustez del modelo, por lo que se puede utilizar para comparar diferentes modelos para una misma tarea. Matemáticamente, se define por medio de la ecuación (4.11).

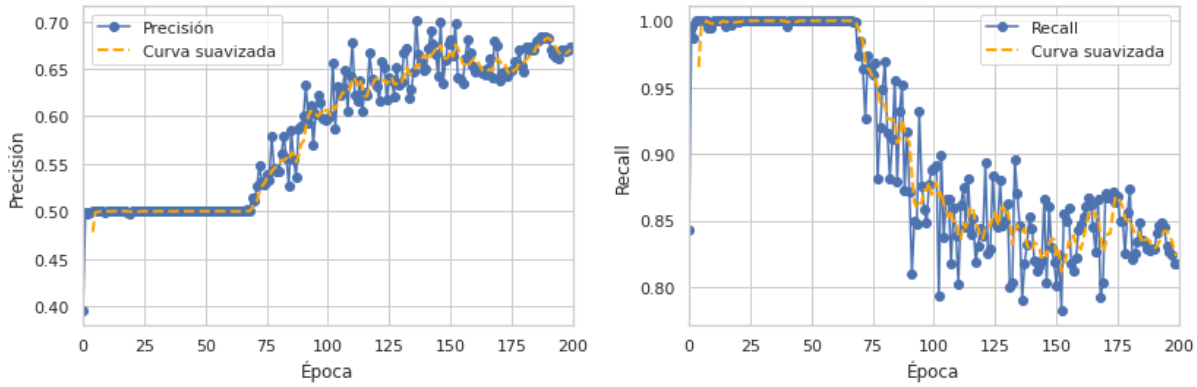
$$mAP@\alpha = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{4.11}$$

Donde N representa el número total de clases, α el nivel de confianza y AP_i la precisión promedio de cada clase. Por su parte, el AP se calcula por medio de la ecuación (4.12).

$$AP = \int_0^1 p(r)\mathrm{d}r \tag{4.12}$$

Donde el su valor representa el tamaño del área encerrada, es decir, la intersección sobre la unión, de la precisión como función del *recall* [102].

En la figura 4.16 se muestran las gráficas de las métricas de precisión (figura 4.16a) y *recall* (figura 4.16b) obtenidos en el entrenamiento de nuestro modelo.



(a) Precisión. (b) *Recall*.

Figura 4.16: Evolución de las métricas precisión y *recall* durante el entrenamiento.

Como se puede observar en la figura 4.16a, la gráfica de la precisión muestra una tendencia ascendente, lo que indica que en cada nueva época del entrenamiento el modelo mejora la proporción de predicciones correctas.

Al analizar la gráfica de *recall* (figura 4.16b) se observa una tendencia ascendente al inicio del entrenamiento, posteriormente se mantiene constante cerca del máximo valor (1.0) y, finalmente, alrededor de las 150 épocas se observan unas fluctuaciones hasta llegar al final del entrenamiento. Si bien estas fluctuaciones podrían indicar un sobreentrenamiento o que algunos parámetros o hiperparámetros de entrenamiento no son los correctos, los valores entre los que fluctúan están alrededor de 0.9, lo cual es un valor aceptable, por lo que es posible decir que durante el entrenamiento se mantuvo en un valor considerablemente bueno y, a pesar de las fluctuaciones, tendría un buen desempeño al detectar aproximadamente el 90 % de los objetos en las imágenes.

La precisión y el *recall* están relacionados en el sentido que mantienen una compensación, es decir, a medida que se aumenta el umbral del nivel de confianza comúnmente se aumenta la precisión pero disminuye el *recall* y viceversa, al bajar el umbral del nivel de confianza disminuye la precisión pero aumenta el *recall*, lo que es recomendable en cierto tipo de aplicaciones, como se mencionó anteriormente.

Existe una gráfica que muestra la relación entre estas dos métricas, conocida como curva P-R. En la figura 4.17 se muestra la curva P-R del modelo entrenado. Es importante mencionar que el área bajo la curva de este gráfico representa la precisión promedio, por ello en la ecuación (4.12) el AP se representa como una integral de la precisión en función del *recall*.

Por otra parte, en la figura 4.18 se muestran los resultados del mAP durante cada época a los niveles de confianza de 0.5 (4.18a) y de 0.5 a 0.95 (4.18b)

Las gráficas del mAP, tanto a un umbral de 0.5 (figura 4.18a) como a un umbral de 0.5 a 0.95 (figura 4.18b) muestran una tendencia positiva lo que indica que el modelo, efectivamente, mejoró a medida que avanzaba el entrenamiento. Los valores máximos en ambas gráficas se

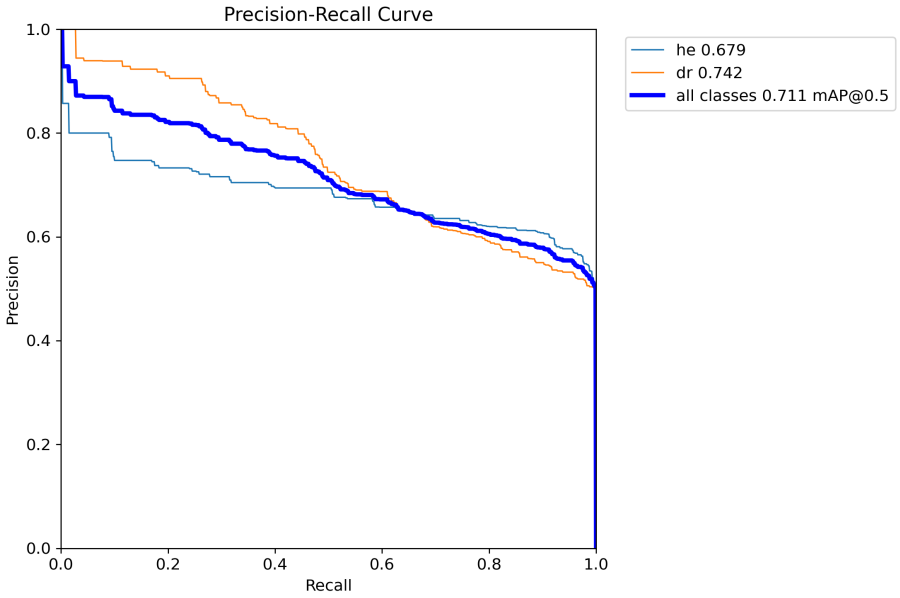
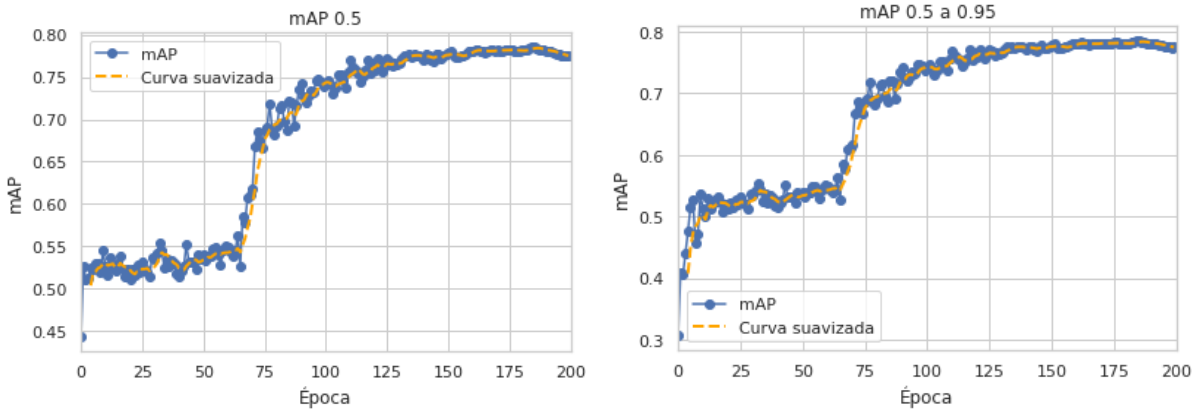


Figura 4.17: Curva P-R.



(a) mAP a niveles de confianza de 0.5. (b) mAP a niveles de confianza de 0.5 a 0.95.

Figura 4.18: Evolución del mAP durante el entrenamiento.

alcanzan alrededor de las 160 épocas, donde se observa que se llega a un máximo aproximado de 0.7, lo que indica que el modelo convergió. Este valor indica que el modelo es capaz de detectar y clasificar objetos con una precisión promedio del 70 % a deferentes niveles de confianza superiores a 0.5.

Existe una herramienta que permite medir el rendimiento de un modelo de clasificación como YOLOv5, ésta es conocida como matriz de confusión y es una tabla que muestra el número o porcentaje de predicciones correctas e incorrectas, es decir el número de las predicciones VP, FP, FN y VN de todas las clases.

En el formato de una matriz de confusión, las filas de la matriz representan la clase real de las imágenes y las columnas las clases predichas por el modelo.

Para interpretar una matriz de confusión es importante mencionar que la diagonal principal representan los valores verdaderos positivos por lo que valores altos en esta diagonal indican un mejor desempeño del modelo el ser capaz de identificar y clasificar correctamente los objetos. Los valores fuera de la diagonal corresponden a los falsos positivos y los valores en las celdas de las clases predichas, pero fuera de la diagonal, corresponden a los falsos negativos.

En la figura 4.19 se muestra la matriz de confusión generada durante el entrenamiento previamente descrito con las imágenes de validación.

Como se observa en la matriz de confusión de la figura 4.19, la diagonal principal presenta valores más altos que los valores fuera de ella, sin embargo el porcentaje de valores VP es relativamente bajo: 61 % para la clase de imágenes saludables y 62 % para la clase de las imágenes con retinopatía diabética.

Con el fin de subir el porcentaje de valores verdaderos positivos, se realizaron dos entrenamientos más: con imágenes originales, o sea sin el preprocesamiento (YOLOv5s_Originales) y con la combinación de imágenes procesadas y sin procesar (YOLOv5s_Combinadas). Además, de igual manera se entrenó el modelo mediano (YOLOv5m) con imágenes originales (YO-

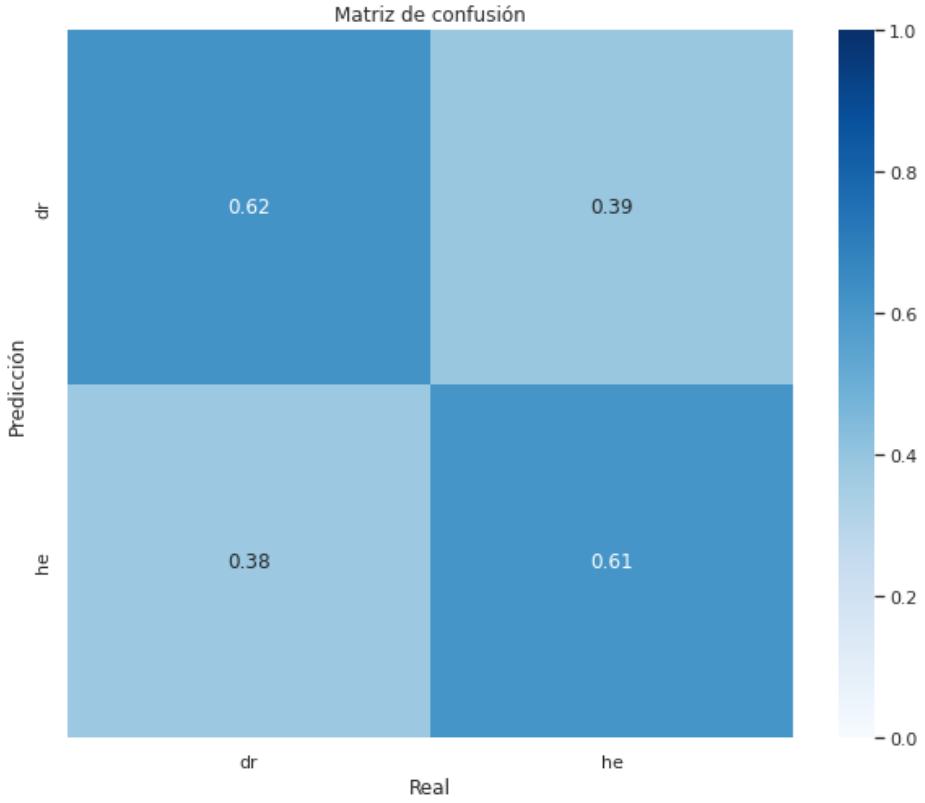


Figura 4.19: Matriz de confusión generada.

LOv5m_Originales), procesadas (YOLOv5s_Procesadas) y combinadas ((YOLOv5s_Combinadas) para comprobar su viabilidad al implementarlo en la Raspberry y comparar su rendimiento con YOLOv5s.

Para el entrenamiento de los otros modelos de YOLOv5s se utilizó el mismo comando mostrado en la sección de código 4.8, diferenciándose únicamente en las imágenes de entrenamiento y validación, donde, en lugar de utilizar imágenes preprocesadas, se utilizaron imágenes originales para el segundo entrenamiento y la combinación de ambas en el tercer entrenamiento.

Para los entrenamientos con el modelo mediano (YOLOv5m) solo bastó con modificar la opción `--weights yolov5s.pt` de la sección de código 4.8 por `--weights yolov5m.pt`.

En las figuras 4.20 y 4.21, se muestran gráficas comparativas de la evolución por época del mAP de cada entrenamiento, el primero con un umbral de confianza de 0.5 y e segundo dentro

del rango de 0.5 a 0.95.

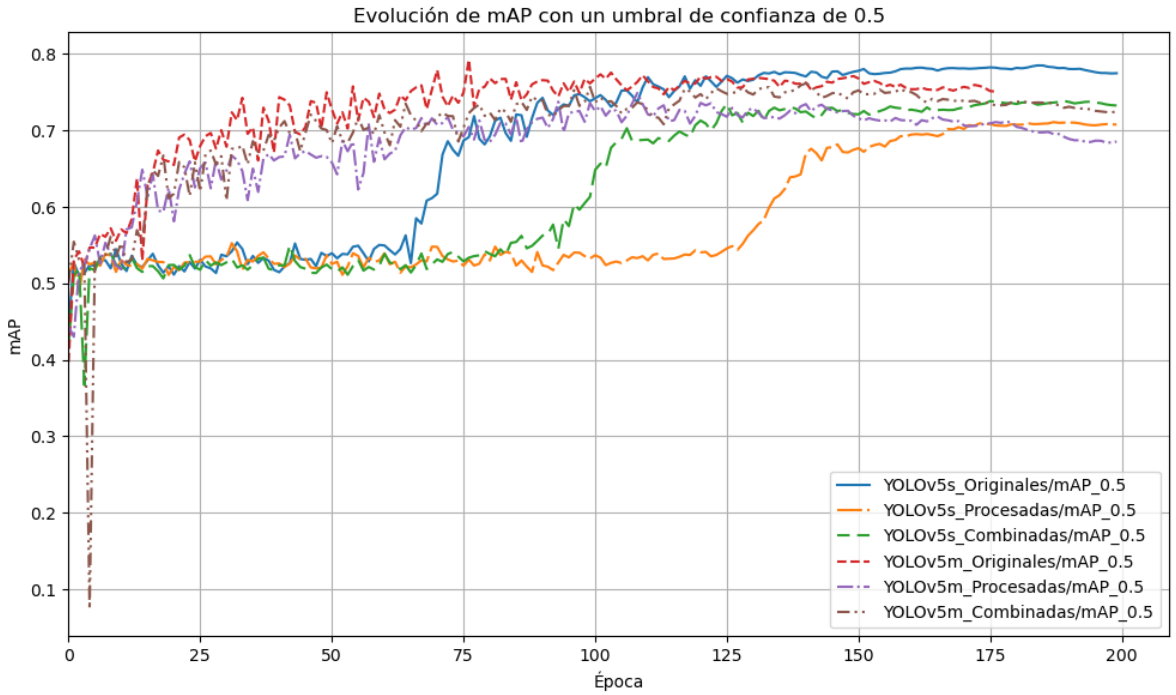


Figura 4.20: Comparativa mAP a nivel de confianza de 0.5.

Como se observa en ambas gráficas, al final de las 200 épocas el entrenamiento del modelo con las imágenes originales (YOLOv5s_Originales) presenta el valor más alto de mAP, seguido por YOLOv5m_Originales. De igual manera, se puede observar que el valor máximo de mAP fue alcanzado por éste último en la época 76, destacando que su entrenamiento interrumpió en la época 176 debido a la condición de paro establecida por un hiperparámetro.

Otro dato a destacar es que el modelo YOLOv5s_Procesadas (el resultante de la metodología propuesta) fue el que menor desempeño presentó, al obtener los valores máximos de mAP más bajos de todos los modelos entrenados. Sin embargo, con el fin de realizar una evaluación independiente de los datos arrojados con el entrenamiento, se realizará una matriz de confusión de cada modelo entrenado con otras imágenes de las bases de datos que no se utilizaron en para el entrenamiento, es decir, con imágenes que la red no ha "visto". No obstante, estas matrices de confusión se presentarán en la sección de resultados.

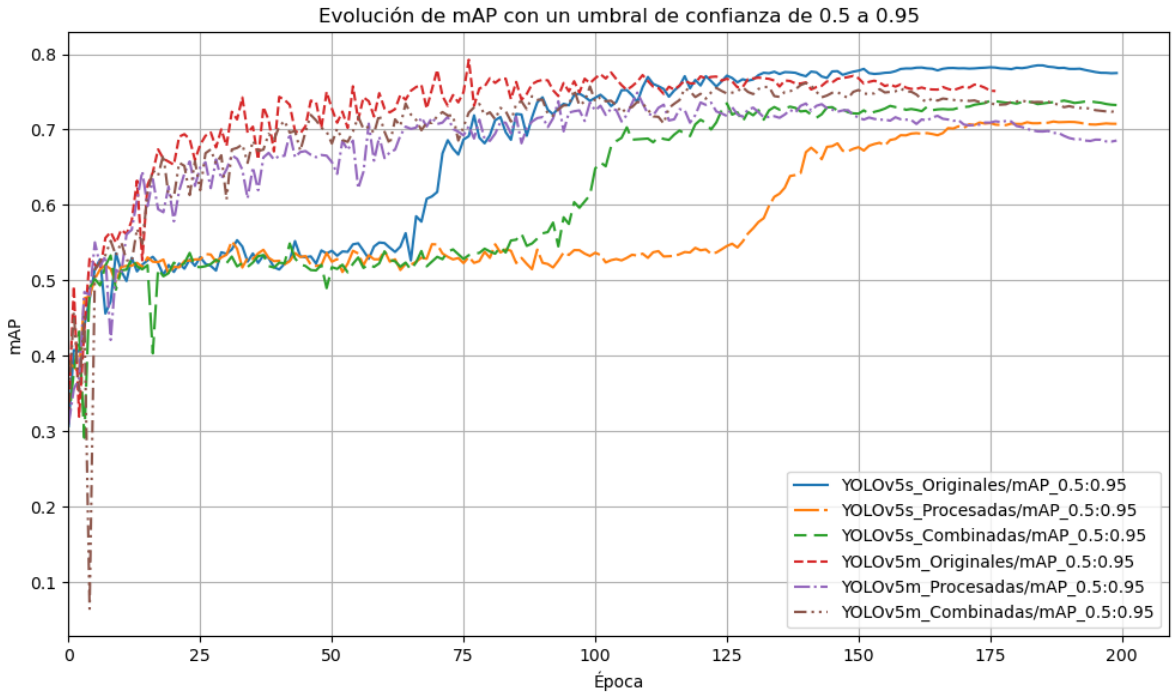


Figura 4.21: Comparativa mAP a niveles de confianza de 0.5 a 0.95.

4.3 Conclusiones del capítulo

En el presente capítulo se describió la segunda parte del desarrollo experimental de esta tesis, donde se utilizó un modelo de *deep learning* para detección de objetos llamado YOLO, específicamente la quinta versión conocida como YOLOv5. Esta versión cuenta con diferentes tamaños: nano, chico, mediano, grande y extra grande. En todos ellos la arquitectura es similar, pero cambia la cantidad de neuronas y filtros.

Debido a que el modelo que se utilizará en este trabajo se implementará en la Raspberry, se decidió utilizar el modelo pequeño conocido como YOLOv5s. Para su entrenamiento se siguió la metodología mostrada en la figura 4.6, donde se utilizaron imágenes con el preprocesamiento descrito en el capítulo anterior para el entrenamiento y la validación del modelo.

A pesar de que los resultados del entrenamiento demuestran un buen desempeño, la matriz

de confusión presenta un bajo porcentaje de valores verdaderos positivos. Por esta razón, se realizaron dos nuevos entrenamientos del mismo modelo pequeño, el primero utilizando solo imágenes originales (sin el preprocesamiento) y el segundo utilizando la combinación de imágenes preprocesadas y originales. De igual manera se entrenó el modelo mediano (YOLOv5m) con las 3 combinaciones de imágenes con el objetivo de comparar su rendimiento con el modelo pequeño y, posteriormente, evaluar su viabilidad y desempeño en la Raspberry Pi.

Aunque la evaluación todos los modelos con imágenes diferentes a las utilizadas en el entrenamiento y validación se presentará en la sección de resultados, la métrica de evaluación mAP mostrada en las figuras 4.20 y 4.21 sugieren un mejor rendimiento en los modelos entrenados con imágenes originales, ambos con un mAP máximo cercano a 0.8. También muestran que el modelo entrenado con la metodología propuesta presenta el menor desempeño.

Capítulo 5
Resultados

Si bien, en los dos capítulos anteriores se mostraron parte de los resultados, estos solamente corresponden a sus respectivas partes experimentales desarrolladas en la computadora personal (PC), como se mencionó anteriormente.

Por lo tanto, en la presente sección se mostrarán los resultados correspondientes a los objetivos establecidos en esta tesis, los cuales corresponden a la implementación de las dos partes experimentales en la Raspberry Pi, para obtener un sistema de procesamiento de imágenes de fondo de ojo para detectar aquellas con retinopatía diabética.

Parte importante de este trabajo es demostrar que el preprocesamiento propuesto mejora el desempeño del modelo de *machine learning* utilizado, por ello, como primer punto se evaluará el modelo entrenado mediante la metodología propuesta en el capítulo anterior (YOLOv5s_Procesadas) al compararla con los otros cinco modelos de YOLOv5 entrenados: YOLOv5s_Originales, YOLOv5s_Combinados, YOLOv5m_Procesados, YOLOv5m_Originales, YOLOv5m_Combinados.

Posteriormente, se mostrará la comparativa de los recursos utilizados por la PC y la Raspberry al realizar la inferencia en diversas imágenes, con el modelo pequeño y el modelo mediano que mejores resultados presenten. Esto con el objetivo de comprobar la viabilidad de implementación de ambos modelos en la Raspberry.

Finalmente, se presentará el sistema completo (preprocesamiento con lógica difusa e inferencia mediante la red neuronal) implementado en la Raspberry.

5.1 Evaluación de YOLOv5s y YOLOv5m

En los resultados del entrenamiento del capítulo anterior, se mencionó que las métricas sugerían un buen desempeño del modelo entrenado mediante la metodología propuesta, sin embargo, la matriz de confusión y los datos del mAP muestran un desempeño considerablemente bajo. Para comprobar su real desempeño, se evaluará con 1000 imágenes nuevas para el modelo, es decir imágenes que no fueron utilizadas para el entrenamiento ni para la validación. De estas 1000 imágenes 500 son con retinopatía diabética y 500 saludables.

Adicionalmente, con el objetivo de valorar el efecto del preprocesamiento con lógica difusa, se realizará la inferencia en las 1000 imágenes con y sin preprocesamiento y se compararán con los resultados de otros modelos entrenados.

Para efectuar la inferencia en las imágenes con los pesos obtenidos en los entrenamientos, se ejecutó el comando mostrado en la sección de código 5.1.

Sección de código 5.1: Comando para ejecutar la detección utilizando el modelo entrenado de YOLOv5.

```
1 python detect.py --weights runs/train/exp1_sProcess/weights/best.pt --source test
    /test_drOri/ --img 640 --conf 0.25 --agnostic-nms
```

El comando mostrado en la sección de código 5.1 ejecuta el programa de detección de YOLOv5 configurando cinco opciones. A diferencia de los comandos para el entrenamiento en la máquina virtual de Google Colab, éste se ejecuta localmente, es decir, en la PC o en la Raspberry. Por ello es necesario tener localmente tanto los pesos obtenidos mediante los entrenamientos como las imágenes a utilizar para la evaluación.

Las opciones configuradas con comando de la sección de código 5.1 se describen a continuación.

weights runs/train/exp1_sProcess/weights/best.pt Se especifica la dirección donde se encuentran los pesos del modelo entrenado a utilizar.

source test/test_drOri/ Se especifica la dirección donde se encuentran las imágenes a detectar.

img 640 Se indica que la inferencia se realizará con imágenes con resolución de 640 píxeles. En caso de que sean de otro tamaño, el mismo programa `detect.py` las redimensionará automáticamente.

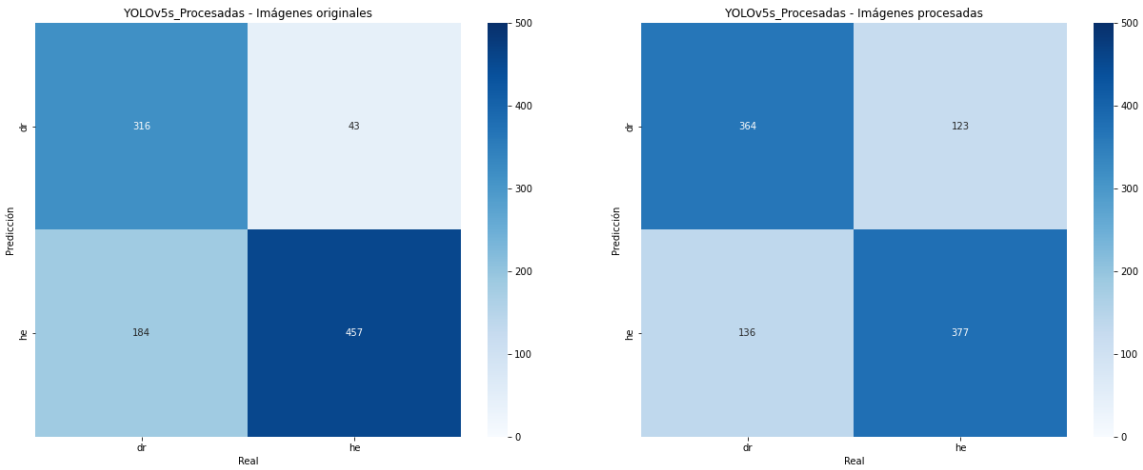
conf 0.25 Se establece el umbral de confianza a 0.25.

agnostic-nms Modifica el comportamiento de la supresión no máxima para que considere todas las cajas delimitadoras (sin importar su clase) para que se identifique una sola caja delimitadora por objeto. Esto con el objetivo de que solo se clasifique la imagen de fondo de ojo en la clase con la mayor probabilidad, ya sea saludable o con retinopatía diabética, pero no ambas.

Como se mencionó anteriormente, para el desarrollo de todos los programas de esta tesis se utilizó una PC con un sistema operativo basado en el kernel de Linux. Por lo tanto, para ejecutar el comando de la sección de código 5.1 en la PC, se utilizó la terminal o consola, la cual es básicamente una interfaz de usuario mediante la cual se interactúa con la computadora por medio de líneas de comandos. En los sistemas operativos basados en Linux, la consola permite ejecutar tanto líneas de código de Python como programas completos sin necesidad de una interfaz de usuario extra.

En la figura 5.1 se muestran las matrices de confusión obtenidas mediante las inferencias realizadas con el modelo YOLOv5s_Procesadas a las 1000 imágenes originales y a las 1000 procesadas, donde la figura 5.1a muestra la matriz con la inferencia en las imágenes originales y la figura 5.1b la matriz resultante con las imágenes procesadas.

Como se observa en la figura 5.1a, al presentar imágenes sin la mejora de contraste al modelo entrenado con imágenes preprocesadas, le es más fácil detectar imágenes saludables, con un



(a) YOLOv5s_Procesadas en imágenes originales.

(b) YOLOv5s_Procesadas en imágenes procesadas.

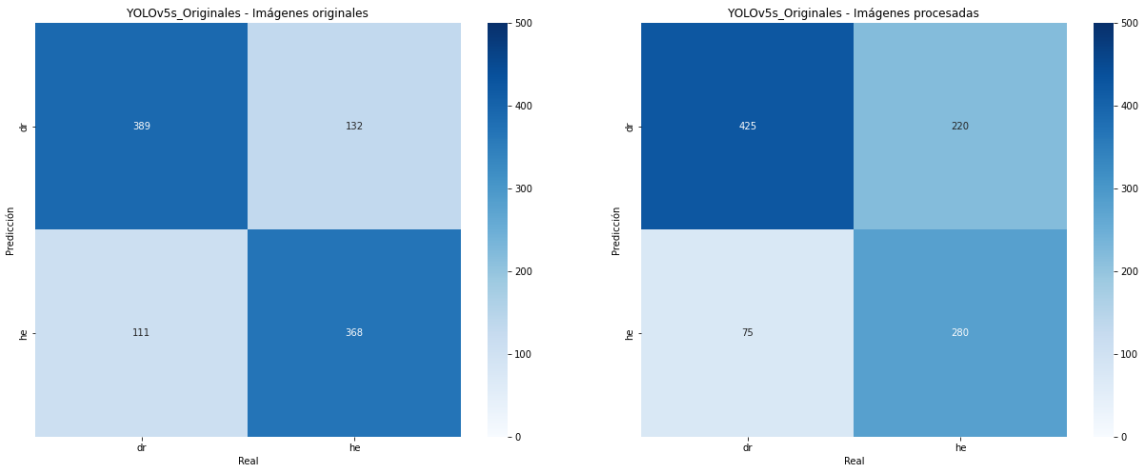
Figura 5.1: Matrices de confusión del modelo YOLOv5s_Procesadas en imágenes originales y procesadas.

número muy bajo de falsos negativos (FN) 43 que representan un porcentaje de error del 8.6 %. Por otra parte, el desempeño en la detección de la retinopatía diabética es más bajo, con más de 4 veces de FN, aproximadamente el 37 %.

En el caso de la inferencia a imágenes procesadas, la figura 5.1b muestra un desempeño más balanceado, subiendo el porcentaje de verdaderos positivos (VP) en la detección de la retinopatía diabética (aproximadamente al 73 % comparados con el 63 % de las imágenes originales), pero subiendo el porcentaje de FN al 24 %.

En la figura 5.2 se muestran las matrices de confusión del modelo YOLOv5s_Originales y en la figura 5.3 las matrices del modelo YOLOv5s_Combinadas.

La matriz de confusión mostrada en la figura 5.2a demuestra un desempeño considerablemente bueno y, a la vez, balanceado del modelo YOLOv5s_Originales al presentarle imágenes sin el procesamiento, con un numero de VP de 389 y 368 para las clases de retinopatía diabética y saludables, respectivamente; que en conjunto corresponden al 75.7 % de aciertos, lo que concuerda con sus valores de mAP obtenidos durante el entrenamiento mostrados en las figuras 4.20 y 4.21.



(a) YOLOv5s_Originales en imágenes originales.

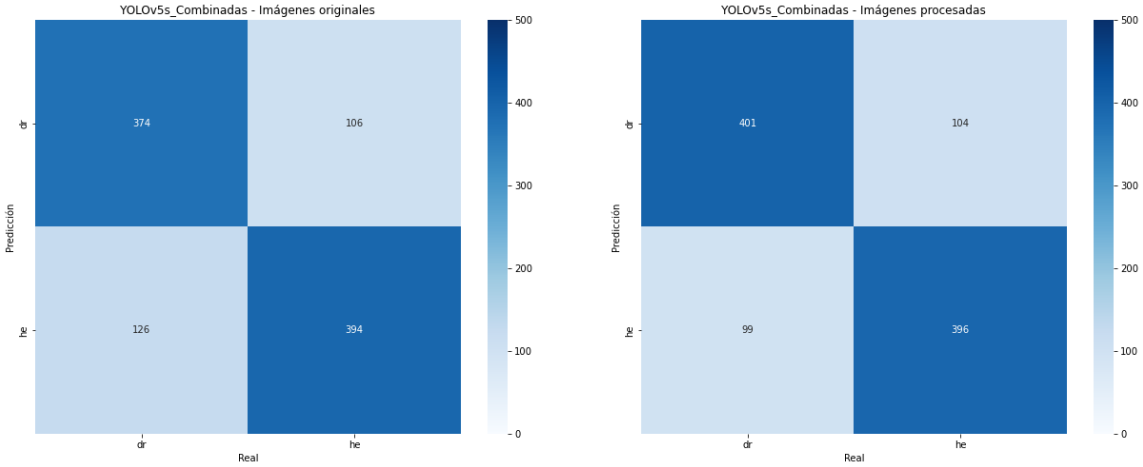
(b) YOLOv5s_Originales en imágenes procesadas.

Figura 5.2: Matrices de confusión del modelo YOLOv5s_Originales en imágenes originales y procesadas.

Al realizar la inferencia de éste mismo modelo con imágenes procesadas, se puede observar en su matriz de confusión (figura 5.2b) que la capacidad del modelo para detectar la retinopatía diabética sube considerablemente, con un número de VP de 425, que representan el 85 % de la clase, sin embargo, la detección de imágenes saludables disminuye hasta el 56 %, lo que representa una baja confianza del modelo, bajando su porcentaje total de VP hasta el 70 %.

Por otra parte, las dos matrices de confusión mostradas en la figura 5.3 demuestran un desempeño balanceado del modelo YOLOv5s_Combinadas en ambos tipos de imágenes, lo que es esperado al ser entrenado precisamente con la combinación de los dos tipos. Sin embargo, al realizar la inferencia con las imágenes procesadas (figura 5.3b) se obtiene el mejor desempeño, con un porcentaje aproximado del 80 % de VP para la clase de retinopatía diabética y un 79 % para la clase saludable, superando al modelo YOLOv5s_Originales.

Para el caso de los modelos de YOLOv5m entrenados, sus matrices de confusión se muestran en la figura 5.4, donde las figuras 5.4a y 5.4b muestran los resultados de la inferencia del modelo YOLOv5m_Procesada con imágenes originales y procesadas, respectivamente. De igual manera, las figuras 5.4c y 5.4d muestran los resultados del modelo YOLOv5m_Originales y las figuras



(a) YOLOv5s_Combinadas en imágenes originales.

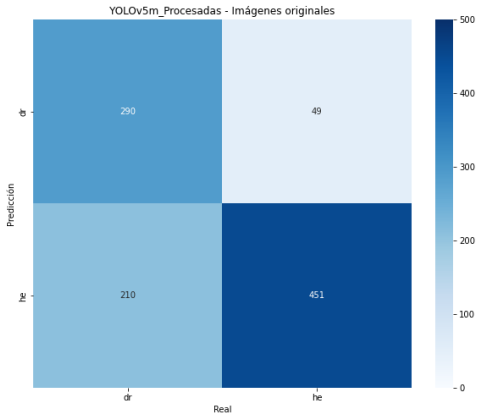
(b) YOLOv5s_Combinadas en imágenes procesadas.

Figura 5.3: Matrices de confusión del modelo YOLOv5s_Combinadas en imágenes originales y procesadas.

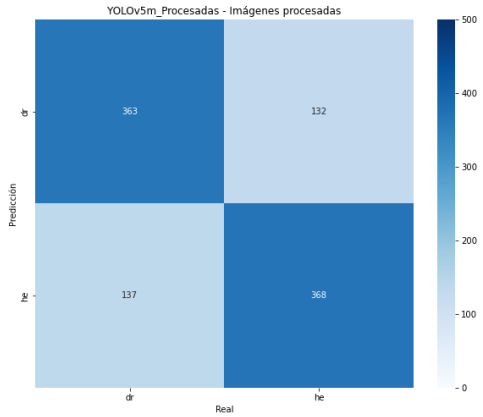
5.4e y 5.4f los del modelo YOLOv5m_Combinadas.

Para interpretar todas las matrices de confusión obtenidas, es importante recordar que en aplicaciones como la de este trabajo, es importante mantener un bajo número de FN, especialmente en imágenes con alguna patología, ya que estas representarían un diagnóstico saludable en imágenes que no lo son. Bajo esta premisa, los modelos que presentan un menor número de FN en la detección de retinopatía diabética son los modelos YOLOv5s_Originales, YOLOv5s_Combinados, YOLOv5m_Originales y YOLOv5m_Combinados, todos ellos al realizar la inferencia en imágenes con el contraste mejorado. Entre estos 4 modelos, los que presentan un mejor balance, es decir, un menor número de FN al clasificar imágenes saludables, son los modelos YOLOv5s_Combinados y YOLOv5m_Combinados.

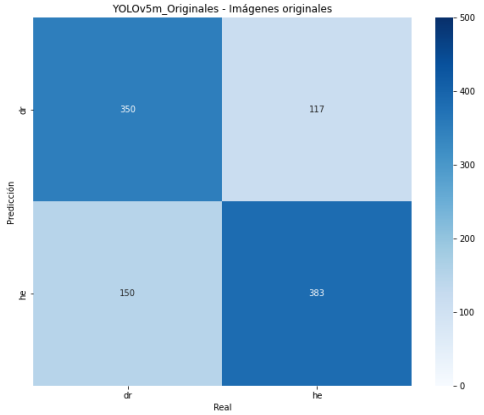
Al comparar estos dos modelos se observa que YOLOv5s_Combinados presenta un buen desempeño y más balanceado, con una confianza del 79.7 %. Por su parte, YOLOv5m_Combinados ofrece el mejor desempeño al detectar la retinopatía diabética, al tener el menor número de FN en esta clase, pero le cuesta clasificar las imágenes saludables, teniendo una confianza de acierto total del 80.3 %. Con estos resultados es posible afirmar que la mejora de contraste por medio



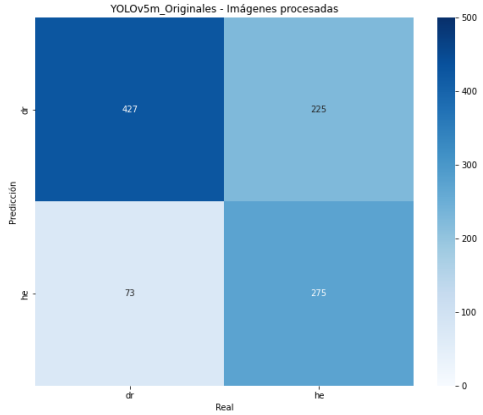
(a) YOLOv5m_Procesadas en imágenes originales.



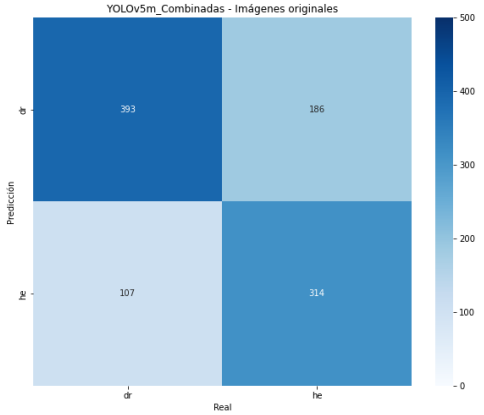
(b) YOLOv5m_Procesadas en imágenes procesadas.



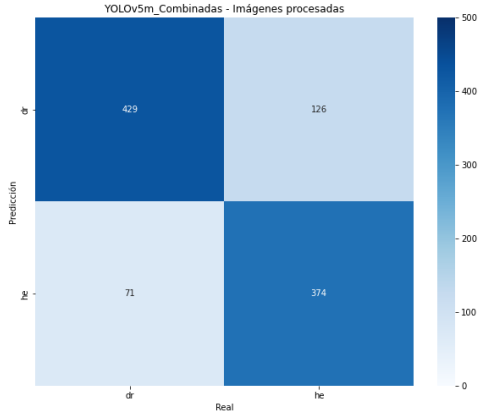
(c) YOLOv5m_Originales en imágenes originales.



(d) YOLOv5m_Originales en imágenes procesadas.



(e) YOLOv5m_Combinadas en imágenes originales.

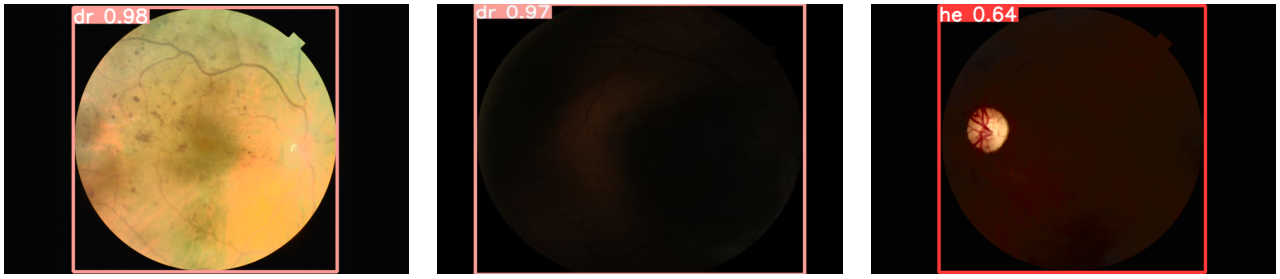


(f) YOLOv5m_Combinadas en imágenes procesadas.

Figura 5.4: Matrices de confusión de los tres modelos de YOLOv5m entrenados en imágenes originales y procesadas.

de la lógica difusa contribuye a un mejor desempeño de la red neuronal, con la combinación de imágenes originales y preprocesadas para el entrenamiento del modelo y con la utilización de imágenes mejoradas para la inferencia.

Las 1000 imágenes utilizadas en la inferencia se eligieron aleatoriamente de las bases de datos, por ello se incluyen imágenes con todo tipo de calidad y diferentes niveles de iluminación. La figura 5.5 muestra la inferencia de tres de éstas imágenes con retinopatía diabética y la figura 5.6 la inferencia de otras tres saludables.



(a) Inferencia de imagen con retinopatía diabética con buena calidad e iluminación.

(b) Inferencia de imagen con retinopatía diabética con baja iluminación.

(c) Inferencia de imagen con retinopatía diabética con baja calidad e iluminación.

Figura 5.5: Inferencia de tres imágenes con retinopatía diabética con diferentes calidades.

(a) Inferencia de imagen saludable con buena calidad e iluminación.

(b) Inferencia de imagen saludable con baja calidad e iluminación.

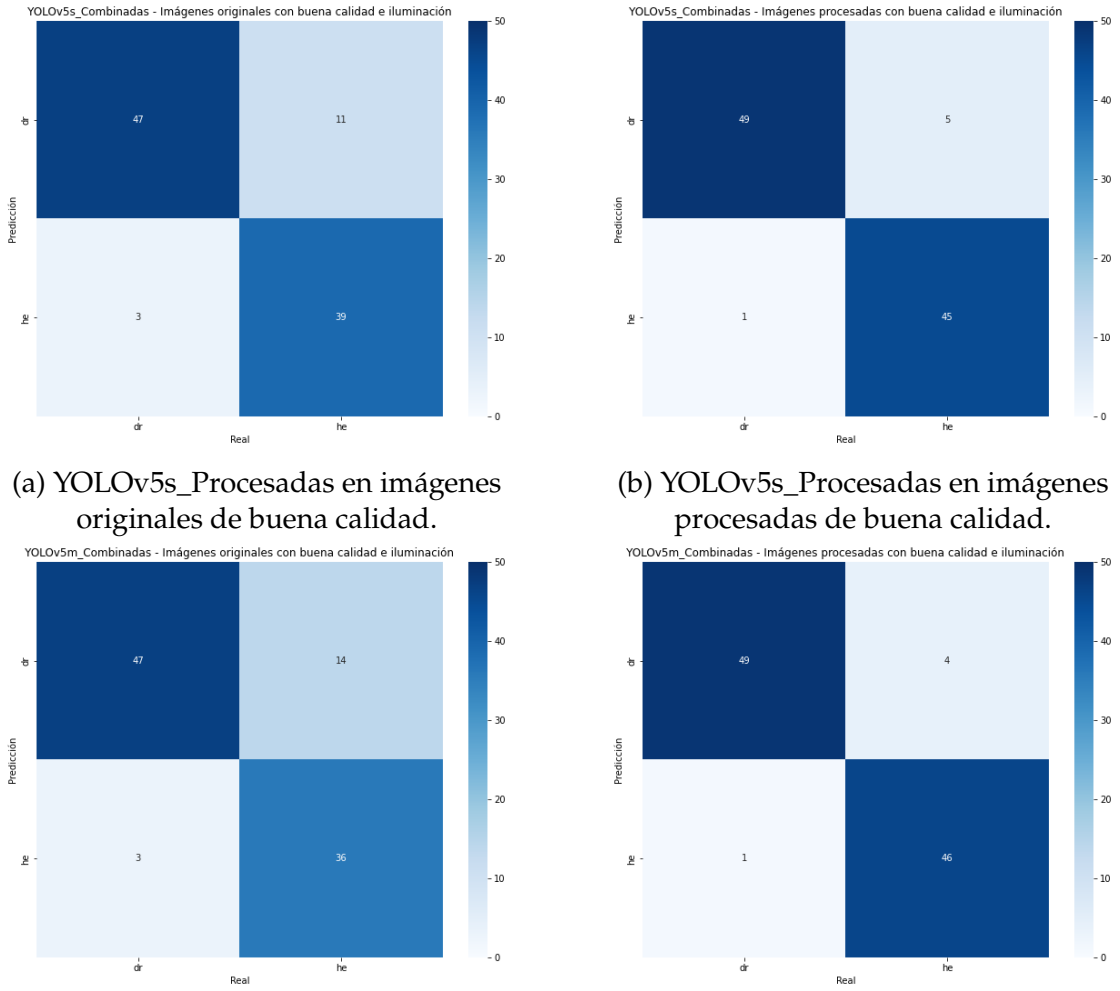
(c) Inferencia de imagen saludable con baja calidad e iluminación.

Figura 5.6: Inferencia de tres imágenes saludables con diferentes calidades.

Como se puede observar en las imágenes mostradas en las figuras 5.5 y 5.6, la inferencia en las imágenes con buena iluminación presentan una confianza mayor en la detección, en cambio las que tienen baja calidad e iluminación presentan una confianza menor al 60 % e inclusive

en la figura 5.5c se observa que el modelo falló al detectar una imagen saludable cuando en realidad es una con retinopatía diabética.

Con el objetivo de buscar un área de mejora para un trabajo a futuro, se realizó la inferencia de los dos modelos en 100 imágenes con buena iluminación. Los resultados se muestran en las matrices de confusión de la figura 5.7.



(a) YOLOv5s_Procesadas en imágenes originales de buena calidad.

(b) YOLOv5s_Procesadas en imágenes procesadas de buena calidad.

(c) YOLOv5m_Procesadas en imágenes originales de buena calidad.

(d) YOLOv5s_Procesadas en imágenes procesadas de buena calidad.

Figura 5.7: Matrices de confusión de YOLOv5s_Procesadas y YOLOv5m_Procesadas en la inferencia de imágenes originales y procesadas de buena calidad.

La figura 5.7a muestra los resultados del modelo YOLOv5s_Combinadas con las imágenes originales y la figura5.7b con imágenes procesadas, de igual manera la figura 5.7c muestra la

matriz de confusión del modelo YOLOv5m_Combinadas con imágenes originales y la figura 5.7d con imágenes procesadas.

Como se observa en las matrices de confusión de la figura 5.7, el desempeño de ambos modelos sube considerablemente al realizar la inferencia con ambos tipos de imágenes (originales y procesadas), sin embargo, el desempeño de la inferencia a imágenes procesadas es aún mayor, alcanzando el 99 % de VP en la detección de la retinopatía diabética y aproximadamente el 90 % en las saludables.

Al analizar estas matrices de confusión observa un desempeño similar en ambos, con la diferencia de que el modelo mediano es mejor en la detección de la retinopatía diabética y el modelo pequeño tiene un desempeño más balanceado.

5.2 Implementación de YOLOv5 en la Raspberry Pi 4 modelo B

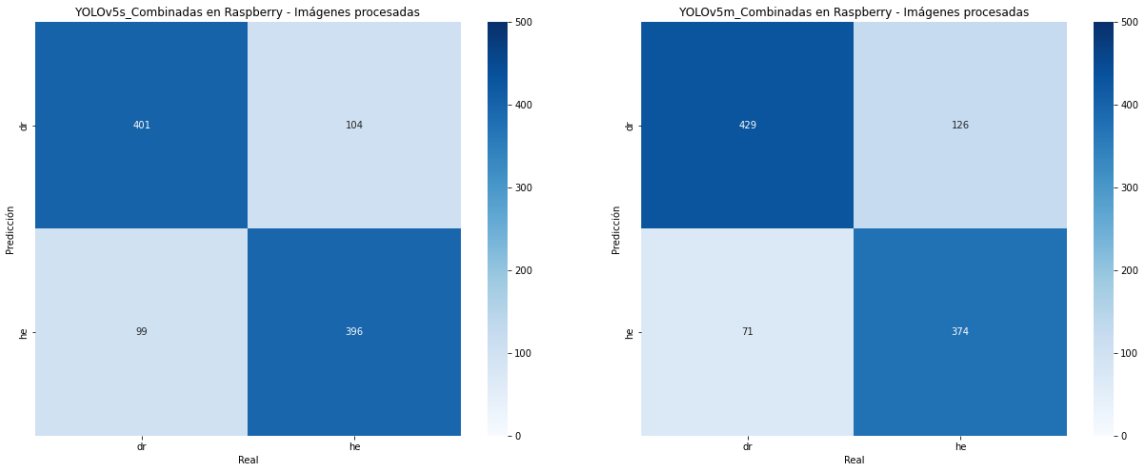
De acuerdo con los resultados presentados en la sección anterior, los modelos pequeño y mediano de YOLOv5 entrenados con la combinación de imágenes procesadas y originales presentaron los mejores desempeños al realizar la inferencia sobre imágenes procesadas, por ello, serán los que se implementarán en la Raspberry Pi 4.

Es importante mencionar que para poder implementar de YOLOv5 en la Raspberry, es necesario configurarla con ciertos parámetros en su sistema operativo. Esta configuración se detalla en la página oficial de Ultralytics [104].

Con el objetivo de comparar el desempeño de la inferencia en la Raspberry con el de la PC, se realizó la detección de las mismas 1000 imágenes procesadas. Como se mencionó anteriormente, el sistema operativo de la Raspberry está basado en el kernel de Linux, por lo tanto, para

realizar la inferencia se utilizó el mismo comando mostrado en la sección de código 5.1. Para ello, previamente se cargaron en la Raspberry los pesos entrenados de ambos modelos y las imágenes.

En la figura 5.8 se muestran las matrices de confusión obtenidas de la inferencia en la Raspberry de las 1000 imágenes procesadas.



(a) Inferencia en la Raspberry de YOLOv5s_Procesadas en imágenes procesadas.

(b) Inferencia en la Raspberry de YOLOv5s_Procesadas en imágenes procesadas.

Figura 5.8: Matrices de confusión de YOLOv5s_Procesadas y YOLOv5m_Procesadas en la inferencia en la Raspberry de imágenes procesadas.

Al comparar las matrices de la figura 5.8 con las figuras 5.3b y 5.4f, se observa que el resultado de la inferencia es exactamente el mismo que en la PC, inclusive el nivel de confianza en la detección es el mismo en todas las imágenes. Esto se debe a que, a pesar de que se carguen en diferentes dispositivos, el modelo es el mismo, mismo número de capas y parámetros: 157 capas y $7,015,519$ parámetros en YOLOv5s y 212 capas y $20,856,975$ parámetros en YOLOv5m. Por lo tanto, a pesar de que se cambie de dispositivo, mientras no se cambien los parámetros del modelo, el resultado de la inferencia será el mismo.

Pese a lo anterior, lo que sí cambia entre dispositivos es el tiempo de la inferencia, ya que este depende de la capacidad y velocidad de procesamiento. Entre los dos dispositivos utilizados en este trabajo la diferencia entre los procesadores es significativa, en el caso de la Raspberry se

utiliza el BCM2711 [23] y en la PC el AMD Ryzen™ 5 3500U [105]. Las diferencias entre estos dos procesadores se resume en la tabla 5.1.

Tabla 5.1: Comparación entre los procesadores de la Raspberry Pi 4 B y la PC.

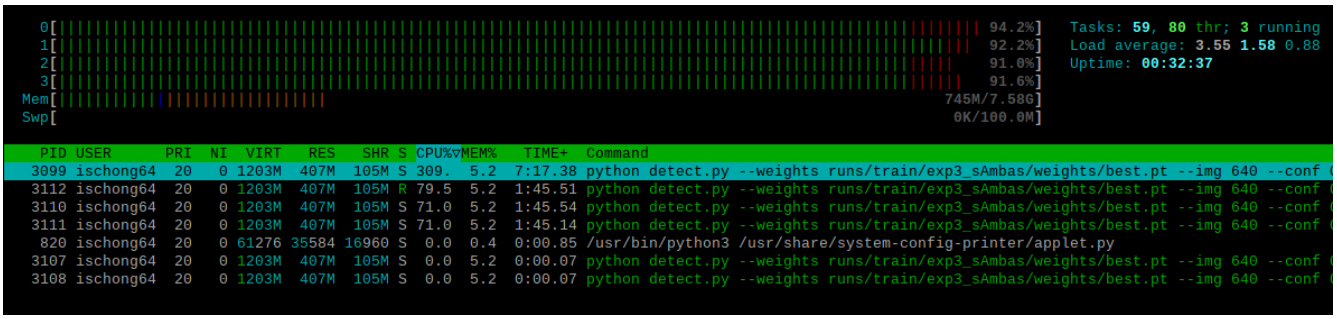
Procesador	**BCM2711**	**AMD Ryzen™ 5 3500U**
Arquitectura	ARMv8-A	ZEN+
Núcleos	4	4
Hilos	4	8
Frecuencia	1.4 GHz	2.1 GHz a 3.7 GHz
Tecnología	16 nm	12 nm
Caché Nivel 1	32 KB de datos 48 KB de instrucciones	128 KB de datos 256 KB de instrucciones
Caché Nivel 2	1 MB	2 MB
Caché Nivel 3	-	4 MB

A pesar de que en la tabla 5.1 se muestra que el número de núcleos es el mismo en ambos procesadores, la diferencia en la potencia es muy grande a favor del AMD Ryzen™ 5 3500U. Sin entrar en detalles, su arquitectura, a parte de ser más moderna, es más eficiente y permite trabajar a más del doble de frecuencia que en el BCM2711, además de que cuenta con el doble de hilos, que son unidades que realizan subprocesos paralelos. Aunado a eso, la capacidad de memoria caché es considerablemente más grande en los 3 niveles.

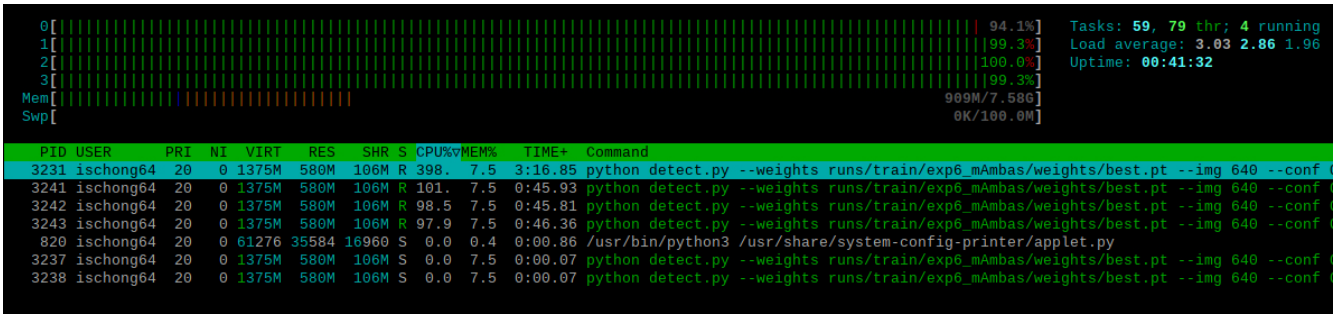
Con base en estos datos, se esperaría que los procesos realizados en la Raspberry Pi sean más lentos y consuman un mayor porcentaje de recursos de hardware que en una PC.

Al momento de realizar la inferencia de ambos modelos en ambos dispositivos, se realizó un monitoreo de los procesos para observar la cantidad de recursos utilizados durante la detección de las 1000 imágenes. En la figura 5.9 se muestra gráficamente el porcentaje de uso del procesador y memoria RAM en la Raspberry con YOLOv5s (figura 5.9a) y con YOLOv5m (figura 5.9b).

De la misma manera, en la figura 5.10 se muestra el monitoreo de la PC con YOLOv5s (figura 5.10a) y con YOLOv5m (figura 5.10b).



(a) Monitoreo al realizar inferencia con YOLOv5s en la Raspberry.



(b) Monitoreo al realizar inferencia con YOLOv5m en la Raspberry.

Figura 5.9: Monitoreo de recursos en la Raspberry durante la inferencia con YOLOv5s y YOLOv5m.

(a) Monitoreo al realizar inferencia con YOLOv5s en la PC.

(b) Monitoreo al realizar inferencia con YOLOv5m en la PC.

Figura 5.10: Monitoreo de recursos en la PC durante la inferencia con YOLOv5s y YOLOv5m.

De las figuras 5.9a y 5.10a, es posible notar que la inferencia de YOLOv5s exige de una gran capacidad de procesamiento, ocupando más del 90 % de la capacidad de los 4 núcleos del procesador de la Raspberry. En la PC también consume una cantidad relevante de recursos: un núcleo al 95 % y otros 4 núcleos arriba del 50 %. En cuanto a la memoria RAM no representa un consumo considerable, siendo solo el 5.2 % en la Raspberry y el 9.5 % en la PC.

Por su parte, como es de esperarse, la cantidad de recursos que consume YOLOv5m es mayor con respecto al modelo pequeño, ocupando el 100 % de un núcleo y solo un poco menos del 100 % de los otros tres en la Raspberry. En la PC también sube la cantidad de recursos utilizados, ocupando aproximadamente el 95 % de dos núcleos y el 80 % de otros dos, manteniendo un porcentaje bajo de uso en los 4 restantes. Al igual que con el modelo pequeño, la cantidad de memoria RAM utilizada por YOLOv5m puede considerarse baja, ya que en porcentaje utiliza solamente 7.9 % de la capacidad total de la Raspberry y el 11.9 % en la PC.

Con estos datos, es posible afirmar que si buen el modelo mediano si consume una mayor cantidad de recursos, su relevancia es poca en la Raspberry, ya que el porcentaje de uso de procesamiento en similar en ambos, cercanos al 100 %, dejando solo como diferencia relevante el tiempo de inferencia requerido por ambos modelos.

Es necesario aclarar que el monitoreo de la PC muestra un total de 8 núcleos a pesar de que el procesador Ryzen solo cuenta con 4, esto debido a que los 4 núcleos físicos pueden ejecutar 2 hilos a la vez y el sistema de monitoreo lo interpreta como si existieran 8 núcleos.

De igual manera, es importante mencionar que la cantidad de recursos varía entre cada imagen, pero, en general, el procesamiento de todas las imágenes de un mismo modelo consumen una cantidad de recursos similar. Donde sí existe una leve diferencia, pero lo suficiente para considerarse, es en la diferencia de recursos entre la Raspberry y la PC, teniendo esta última una mayor capacidad de procesamiento, lo que permite que no se sature al 100 %.

En el caso del tiempo de inferencia de YOLOv5, que como se mencionó fue el dato diferenciador más relevante, es posible obtenerlo con el mismo programa `detect.py`, ya que muestra

el tiempo de inferencia por cada imagen que procesa y al final muestra el tiempo promedio por imagen.

En la tabla 5.2 se muestra la comparativa del tiempo de inferencia por imagen de ambos modelos, tanto en la Raspberry como en la PC.

Tabla 5.2: Tiempos de inferencia por imagen de YOLOv5s_Combinadas y YOLOv5m_Combinadas en la Raspberry y en la PC.

Dispositivo	YOLOv5s_Combinadas	YOLOv5m_Combinadas
Raspberry	Preprocesamiento: 12.2 ms Inferencia: 1583.1 ms NMS: 3.7 ms	Preprocesamiento: 13.6 ms Inferencia: 4090.7 ms NMS: 4.1 ms
PC	Preprocesamiento: 0.6 ms Inferencia: 129.4 ms NMS: 0.6 ms	Preprocesamiento: 0.7 ms Inferencia: 326.9 ms NMS: 0.7 ms

Como se observa en la tabla 5.2 la diferencia de tiempos entre ambos dispositivos es significativa: aproximadamente 20 veces mayor en el preprocesamiento, 12 veces mayor en la inferencia y 6 veces mayor en la supresión no máxima; lo que deja en evidencia la menor velocidad de procesamiento de la Raspberry con respecto a la PC. Esta diferencia se hace más notoria en la detección de múltiples imágenes, ya que con los datos mostrados en dicha tabla, el tiempo total de inferencia en la 1000 imágenes en la PC fue de poco más de 2 minutos en YOLOv5s y de 5.47 minutos en YOLOv5m, en el caso de la Raspberry el tiempo total con modelo pequeño fue aproximadamente de 26.65 minutos y de 68.47 minutos para el mediano.

De igual manera, en la tabla 5.2 se puede notar que la diferencia de tiempos de preprocesamiento y NMS es relativamente pequeña entre el modelo pequeño y el mediano en ambos dispositivos, sin embargo, la diferencia en el tiempo de inferencia ya es considerable: aproximadamente 3 veces mayor en YOLOv5m en ambos dispositivos. Esta diferencia de tiempos corresponde a la misma deferencia en la cantidad de parámetros que tiene YOLOv5m (20,856,975) a comparación de YOLOv5s (7,015,519).

Si bien, los tiempos de inferencia en la Raspberry no son aptos para una detección en tiempo

real, para la aplicación que se propone en este trabajo se pueden considerar como buenos, ya que en cuestión de segundos se podría saber si una imagen de fondo de ojo presenta una retinopatía diabética o no.

5.3 Implementación del sistema completo de procesamiento de imágenes

En la sección anterior se presentaron y discutieron los resultados de la implementación de los modelos pequeño y mediano de YOLOv5 que mejor desempeño mostraron en la inferencia de 1000 imágenes, demostrando que la implementación en la Raspberry de ambos modelos es viable, con tiempos de inferencia no aptos para detección en tiempo real, pero factibles para la aplicación en imágenes médicas.

En esta sección se presentarán los resultados de la implementación del modelo completo, de igual manera comparando el desempeño en la PC con el de la Raspberry. Sin embargo, previamente se presentarán los resultados de la implementación individual del sistema de mejora de contraste con lógica difusa, descrito en el tercer capítulo de esta tesis.

5.3.1 Implementación del sistema de mejora de contraste en la Raspberry

El algoritmo desarrollado para la mejora de contraste se describió con el código mostrado en la sección de código 3.12, recordando que este algoritmo utiliza las funciones de los sistemas de inferencia difusos definidas en las secciones de código 3.3 y 3.5.

El tiempo requerido por el algoritmo depende principalmente del tamaño de la imagen a procesar, lo que se puede demostrar al procesar una misma imagen con dos tamaños distintos. En la tabla 5.3 se muestra la comparativa del tiempo de ejecución, del porcentaje de uso del CPU y de memoria RAM utilizado por los dispositivos al ejecutar el programa con el algoritmo de la sección de código 3.12 en la Raspberry y en la PC. La imagen original a procesar tiene un tamaño de 3504×2336 píxeles, mientras que su versión redimensionada tiene un tamaño de 640×426 píxeles (px).

Tabla 5.3: Comparativa del procesamiento de imagen en la Raspberry y la PC.

Dispositivo	Tamaño (px)	Tiempo (s)	CPU (%)	RAM (%)
Raspberry	$\mathbf{640 \times 426}$	7.37	25	1.2
	$\mathbf{3504 \times 2336}$	225.42	25	5.5
PC	$\mathbf{640 \times 426}$	2.21	7.6	1.9
	$\mathbf{3504 \times 2336}$	67.72	12.5	6.6

Al analizar los datos de rendimiento en la tabla 5.3 se puede observar una diferencia de tiempo importante al procesar las imágenes grandes con respecto a las redimensionadas en ambos dispositivos, requiriendo más de un minuto de procesamiento en la PC y poco más de 4 minutos en la Raspberry. Por su parte, para las imágenes pequeñas el tiempo baja de manera considerable, tomando solo poco más de 2 segundos en la PC y menos de 8 segundos en la Raspberry, tiempos de procesamiento aceptables tratándose de un método que utiliza dos sistemas de inferencia difusos para mejorar el contraste.

Con respecto a la diferencia entre dispositivos, es casi 4 veces mayor el tiempo de procesamiento en la Raspberry que en la computadora personal, sin embargo, con imágenes pequeñas el tiempo es aceptable para la aplicación propuesta.

Como se observa en la misma tabla 5.3, el consumo de recursos requerido para realizar el procesamiento es relativamente bajo en porcentaje en la PC, mientras que en la Raspberry es del 25 % en ambas imágenes, un porcentaje que se puede considerar bajo pero podría llegar a ser significativo. Finalmente, al observar el porcentaje de memoria RAM podría considerarse

irrelevante al ser un porcentaje muy bajo en ambos dispositivos.

Tomando en cuenta estos datos obtenidos, en el sistema completo que se implementará en la Raspberry, previo a la mejora de contraste, se redimensionará la imagen de entrada a manera que el lado de mayor tamaño tenga una dimensión de 640 píxeles, que es el tamaño utilizado por YOLOv5 para realizar la inferencia. Para ello, en el programa de mejora de contraste de la sección de código 3.12 se le agregaron las líneas 9, 10, 11 y 12 mostradas sección de código 4.2, que son las encargadas de redimensionar la imagen a un tamaño de largo de 640 píxeles conservando la relación de aspecto. Quedando el algoritmo de mejora de contraste como se muestra en la sección de código 5.2.

Sección de código 5.2: Algoritmo final para mejerar el contraste de imágenes de fondo de ojo mediante lógica difusa.

```
1  Original = io.imread ("original.jpg")
2  scale = 640/Original.shape[1]
3  w = 640
4  h = int(Original.shape[0]*scale)
5  Redim = (transform.resize(Original, (h, w), anti_aliasing=(True))*255.0).astype('
       uint8')
6  lab = color.rgb2lab(Redim)
7  Canal_L = lab[:,:,0]/100
8  std = np.std(Canal_L)
9  x = fisSD(std)
10 Canal_Mejorado = fisImg(X, Canal_L)
11 lab [:,:,0] = ilu_ace *100
12 rgb = color.lab2rgb(lab)
```

5.3.2 Implementación del sistema embebido

Como resultado final de esta tesis, se mostrará la implementación del sistema embebido capaz de discernir imágenes de fondo de ojo con retinopatía diabética de aquellas que sean saludables.

Para ello, se creó un interfaz de usuario sencilla, únicamente con 3 botones, donde el primero carga una imagen, el segundo aplica la mejora de contraste y guarda la imagen procesada y el tercero realiza la inferencia, mostrando el resultado con la clase a la cual pertenece.

Para que el sistema funcione en la Raspberry es necesario la instalación de YOLOv5 con sus requerimientos. Para ello basta con abrir la terminal en la Raspberry y ejecutar en ella, uno a uno, los comandos de la sección de código 4.5 omitiendo el primer símbolo en cada uno, es decir, sin el `!` en el primer comando ni el `%` en los otros dos. Su efecto será el mismo que en la máquina virtual: se descargará el repositorio de YOLOv5 en la Raspberry y se instalarán las bibliotecas de Python necesarias para su funcionamiento.

En seguida, es necesario guardar los pesos obtenidos con el entrenamiento, el programa de mejora de contraste y el de la interfaz gráfica dentro de la carpeta YOLOv5, que se crea al descargar el repositorio. Finalmente, se debe editar el programa `detect.py` de YOLOv5 para configurar la ruta de los pesos que utilizará y activar la visualización de la imagen detectada.

Las opciones a configurar se encuentran en la definición de la función `def parse_opt():` y basta con sustituir las líneas donde se encuentran las opciones `--weights` y `--view-img` por las mostradas en las secciones de código 5.3 y 5.4, respectivamente.

Sección de código 5.3: Declaración de pesos a utilizar en la inferencia.

```
1 parser.add_argument("--weights", nargs="+", type=str, default=ROOT / "s_best.pt",
      help="model path or triton URL")
```

Sección de código 5.4: Activación de opción para visualizar la imagen inferida.

```
1 parser.add_argument("--view-img", default=True, action="store_true", help="show
     results")
```

En la sección de código 5.3 se observa que el nombre del archivo que contiene los pesos entrenados es "`s_best.pt`", esto con el fin de diferenciarlo de los otros entrenamientos. En caso de requerir utilizar el modelo mediano, basta con guardar dentro del mismo directorio los pesos

del modelo mediano y sustituir "`s_best.pt`" por el nuevo nombre del archivo, que en este caso sería "`m_best.pt`".

Para abrir la interfaz de usuario y poder realizar la mejora de contraste y/o la inferencia, simplemente se debe abrir la terminal, acceder a la carpeta `/yolov5` por medio del comando `cd yolov5` y correr el programa de interfaz de usuario, al que se nombró "`interfaz.py`", por medio del comando `python interfaz.py`, tal como se muestra en la figura 5.11.

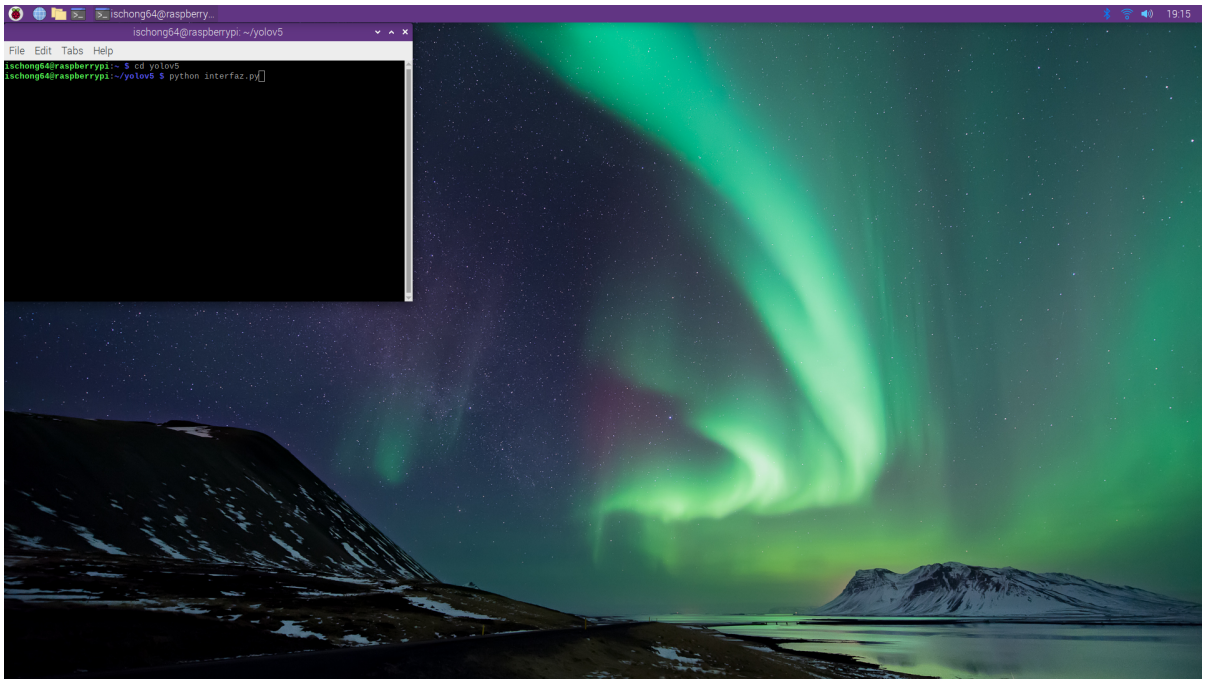


Figura 5.11: Comandos para la ejecución de la interfaz de usuario.

Al ejecutar el programa `interfaz.py` se abrirá la interfaz de usuario mostrada en la figura 5.12, donde se observan los 3 botones antes mencionados.

Al presionar el botón "`Cargar imagen`" se abrirá otra ventana (ver figura 5.13) que nos permitirá buscar y seleccionar la imagen a utilizar.

Una vez seleccionada la imagen, el programa guardará una copia en el directorio de `/yolov5`, para que al presionar el botón "`Procesar imagen`" se ejecute el programa de mejora de contraste, utilizando la copia de la imagen original. Al terminar el procesamiento, se guardará la imagen

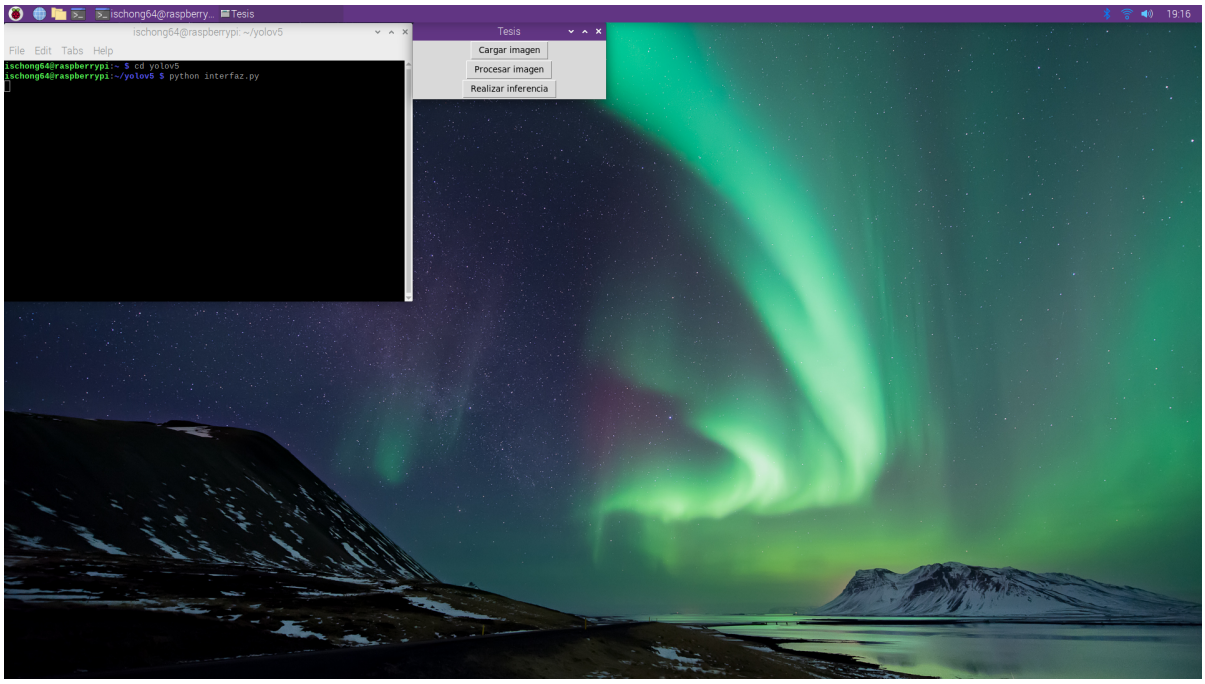


Figura 5.12: Interfaz de usuario.

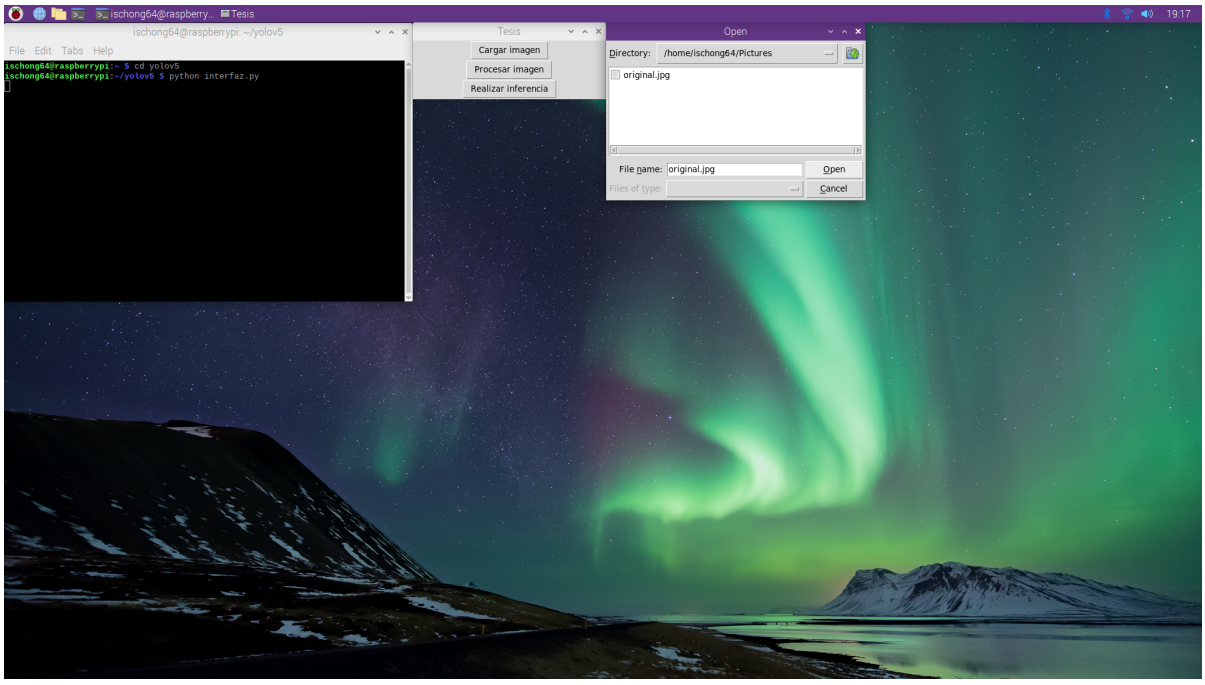


Figura 5.13: Cargar imagen.

procesada en el directorio `data/images`, en la figura 5.14 se muestra la imagen mejorada y el directorio donde se guarda al presionar el botón.

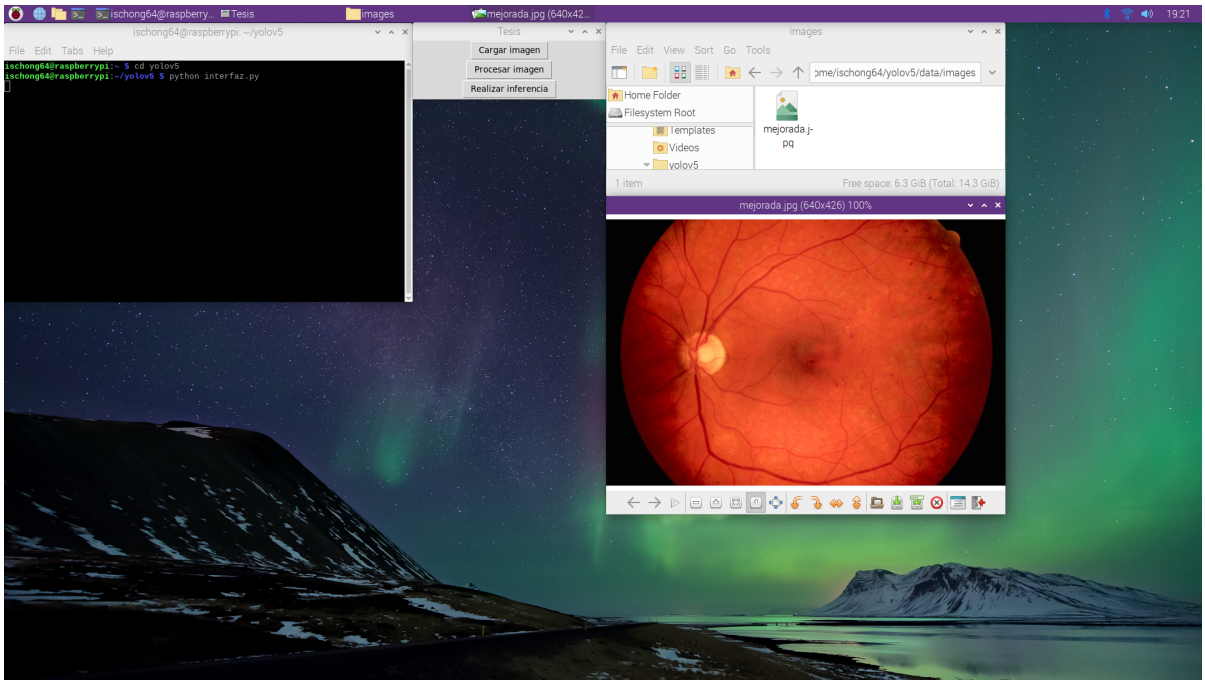


Figura 5.14: Imagen mejorada al presionar el botón.

Finalmente, al presionar el botón "`Realizar inferencia`", se ejecutará el programa de YOLO `detect.py`, el cual utilizará los pesos configurados (ya sean los del modelo pequeño "`s_best.pt`" o del mediano "`m_best.pt`"), accederá al directorio donde se guardó la imagen procesada y realizará la inferencia, mostrando la imagen resultante con la etiqueta (ver figura 5.15) y guardándola en el directorio `runs/detect/exp`.

Con esto queda implementado el sistema de procesamiento de imágenes digitales de fondo de ojo en la Raspberry Pi 4 modelo B, utilizando un sistema de inferencia difuso para mejorar el contraste y una red neuronal convolucional (implementada en un modelo de *deep learning*) para detectar aquellas imágenes que presenten una retinopatía diabética, con una precisión del 80.3 % con el modelo YOLOv5s y del 79.7 % con YOLOv5m, ambas entrenadas con una combinación al 50 % de imágenes procesadas y originales.

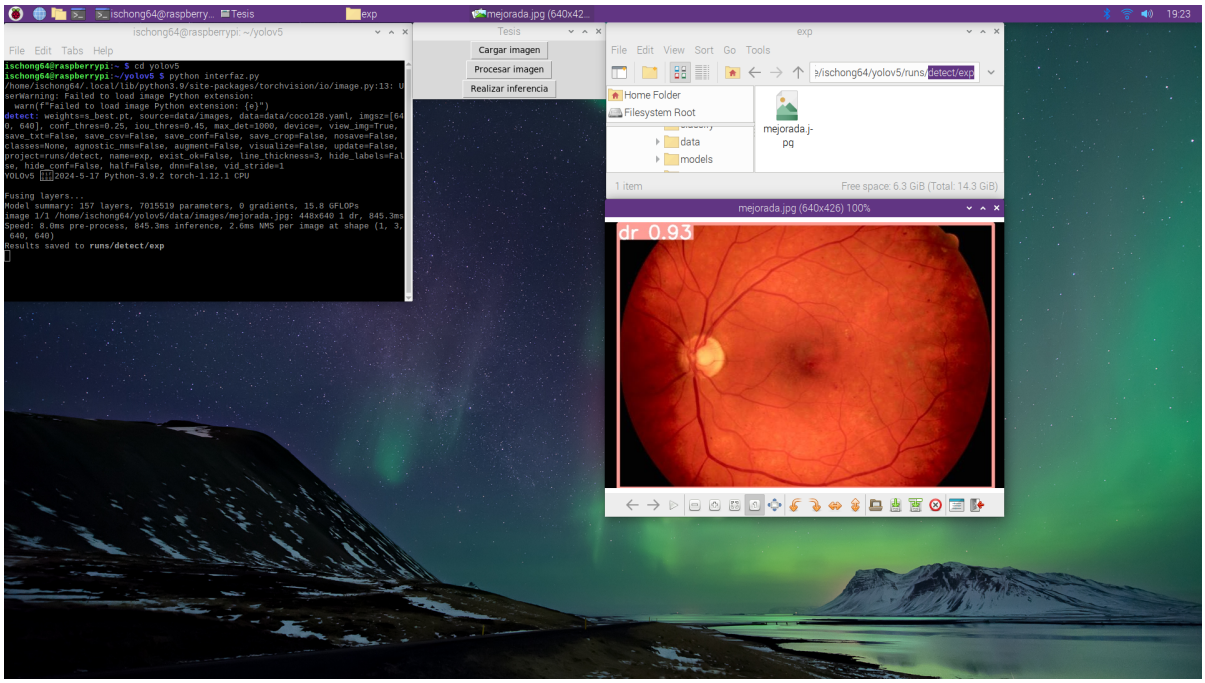


Figura 5.15: Inferencia de imagen mejorada al presionar el botón.

5.4 Conclusiones de capítulo

En este capitulo, se presentaron los resultados obtenidos al evaluar los modelos pequeño y mediano de YOLO, utilizando tres configuraciones de imágenes para su entrenamiento: originales, procesadas y combinadas. También se presentaron los resultados de la implementación en el sistema embebido Raspberry Pi, tanto del sistema de mejora de contraste como de los modelos de YOLOv5 que mejor desempeño exhibieron.

Como primer punto se demostró que el entrenamiento de los modelos YOLOv5s y YOLOv5m con la combinación de imágenes procesadas y originales presentaron el mejor rendimiento en la prueba de los modelos. De igual manera, se mostró que el rendimiento de ambos modelos es mejor al hacer la inferencia sobre imágenes procesadas.

Posteriormente se mostró el rendimiento de ambos modelos implementados en la Raspberry y se compararon los resultados con el rendimiento en la computadora personal. Con ello se com-

probó que la implementación YOLOv5s y YOLOv5s es viable en dicho dispositivo, resultando en una inferencia más lenta pero igual de precisa que en la PC.

Finalmente se mostró la implementación del sistema de procesamiento de imágenes de fondo de ojo en la Raspberry, demostrando su funcionalidad y viabilidad.

Conclusiones

En esta tesis se planteó el objetivo de implementar en software y hardware un sistema para el procesamiento de imágenes digitales utilizando las técnicas de lógica difusa y redes neuronales. Para ello se propuso utilizar imágenes médicas, específicamente imágenes de fondo de ojo, considerando un posible apoyo al personal médico en la detección de retinopatía diabética.

El trabajo experimental realizado, se dividió en dos partes, en la primera se propuso realizar una mejora de contraste en las imágenes de fondo de ojo utilizando dos sistemas de inferencia difusos (FIS) y en la segunda se propuso utilizar una red neuronal convolucional implementada sobre un modelo de aprendizaje profundo (mejor conocido por su nombre en inglés *deep learning*) llamado YOLOv5. Ambas partes se desarrollaron y optimizaron en una computadora personal (PC) y, posteriormente, se implementaron en una Raspberry Pi 4 modelo B de 8 GB.

Con base en el desarrollo de esta tesis, que incluye la descripción del marco teórico, las dos partes experimentales y los resultados, se concluye lo siguiente:

- En este trabajo se contó con el apoyo de personal médico especializado, ya que la correcta interpretación de este tipo de imágenes depende características particulares. Por ello, el diagnóstico de las imágenes se realizó bajo la supervisión del personal médico. De igual manera, se tomó en cuenta su criterio para diseñar el sistema de mejora de contraste, donde, con su opinión, se determinó que era necesario mantener la tonalidad de los colores y la forma en las regiones de mácula y nervio óptico en las imágenes de fondo de ojo mejoradas en contraste, ya que una alteración significativa en ellas podría resultar en un diagnóstico erróneo.

- El uso de la lógica difusa posibilitó realizar una mejora de contraste adecuada para las

imágenes de fondo de ojo, debido a que esta técnica permite interpretar los conocimientos del personal médico y el lenguaje natural para modelar los FIS. De igual manera, permitió crear un sistema de contraste adaptativo, es decir, que la cantidad de ajuste dependa del contraste inicial de las imágenes.

- Con la metodología propuesta mostrada en la figura 3.3, se logró realizar una mejora de contraste adecuada para las imágenes de fondo de ojo, de acuerdo con los comentarios del personal médico. Esto con la utilización del espacio de color CIELAB, modificando únicamente el contraste en canal L, el cual contiene la información de la iluminación de la imagen.

- La efectividad de la mejora de contraste utilizando la lógica difusa se puede observar en las gráficas del MCI y PSNR mostradas en las figuras 3.13 y 3.14, respectivamente. Las cuales indican un menor ajuste de contraste pero una pérdida de información considerablemente más baja a comparación de otros métodos básicos de mejora de contraste, lo cual coincide con los comentarios del personal médico.

- Con respecto al entrenamiento del modelo de *deep learning* YOLOv5, al utilizar únicamente las imágenes preprocesadas (con la mejora de contraste) para el entrenamiento del modelo, tal como se muestra en el diagrama de la figura 4.6, se obtuvo un bajo rendimiento en la tarea de clasificación, lo que se puede observar en la matriz de confusión de la figura 4.19, donde se muestra un porcentaje apenas arriba del 60 % en la detección de valores verdaderos positivos. A pesar de que la evolución del error de clasificación durante el entrenamiento (figura 4.14a) muestra una tendencia descendente, las constantes fluctuaciones por encima de la curva suavizada manifiestan ese bajo rendimiento en la clasificación.

- Por lo concluido en el punto anterior, se probó el entrenamiento con imágenes sin el preprocesamiento y con la combinación de ambos, tanto con el modelo pequeño como con el modelo mediano de YOLOv5, conocidos como YOLOv5s y YOLOv5m, respectivamente. Los resultados del entrenamiento, mostrados en las figuras 4.20 y 4.21 muestran que los

mejores desempeños son de los modelos entrenados con imágenes sin preprocesamiento. Sin embargo, al realizar la prueba con imágenes nuevas para el modelo, tanto originales como procesadas, se obtuvo un mejor desempeño en los modelos entrenados con la combinación de ambas imágenes (YOLOv5s_Combinadas y YOLOv5m_Combinadas), realizando la inferencia en imágenes con la mejora de contraste, obteniendo una confianza aproximada del 80 %, que se muestran en las figuras 5.3b y 5.4f.

- Al probar los modelos YOLOv5s_Combinadas y YOLOv5m_Combinadas en imágenes con una adecuada iluminación, el desempeño sube considerablemente, llegando a detectar el 99 % de imágenes con retinopatía, tal como se muestra en las matrices de confusión de la figura 5.7.

- Con respecto a la Raspberry, se demostró la viabilidad de implementación tanto del sistema de mejora de contraste utilizando lógica difusa como del modelo de *deep learning* para detectar imágenes de fondo de ojo y clasificarlas en saludables o con retinopatía diabética, diferenciándose de la PC en los tiempos de procesamiento, siendo poco más de 3 veces mayor en la Raspberry, tal como se muestra en las tablas 5.2 y 5.3. Esta menor velocidad se debe a la menor cantidad de recursos del procesador BCM2711 de la Raspberry en comparación con el procesador de la PC, principalmente en su velocidad y capacidad de procesamiento, contrastados en la tabla 5.1.

- Aunque la diferencia de tiempos entre ambos dispositivos es entre 3 y 4 veces mayor en la Raspberry, el tiempo total no representa un problema, ya que con el modelo YOLOv5s en un tiempo aproximado de 8 segundos se puede obtener la imagen con mejora de contraste y en un lapso entre 2 y 3 segundos se obtiene el resultado de la inferencia. Con esto, es posible obtener el resultado de la mejora de contraste de la imagen de fondo de ojo y su clasificación en saludable o con retinopatía diabética en un tiempo aproximado de 10 segundos con este modelo.

- La implementación en la Raspberry del modelo mediano de YOLO también es viable, sin embargo, el tiempo de inferencia es aproximadamente tres veces mayor que en YOLOv5s,

como se muestra en la tabla 5.2. Lo que supone un tiempo aproximado de 20 segundos de procesamiento total.

- En la figura 5.9 se observa que al implementar ambos modelos de YOLO en la Raspberry, el consumo de recursos es casi del 100 % con respecto al CPU y del 7 % en memoria RAM, sin embargo, debido a que está pensado en implementarse sobre un dispositivo compacto para una única la aplicación, se puede afirmar que su implementación en la Raspberry Pi es viable como sistema embebido.
- Como punto final, se concluye la viabilidad de implementar sobre un sistema embebido, un sistema de procesamiento de imágenes utilizando la lógica difusa y redes neuronales convolucionales.

Trabajo a futuro

Durante el desarrollo de esta tesis y con los resultados experimentales obtenidos, se identificaron varias oportunidades para futuras investigaciones, las cuales se mencionan a continuación:

- Validar el sistema buscando colaboración con instituciones médicas, con el fin de probar el modelo en escenarios reales y obtener retroalimentación.
- Optimizar el sistema mejora de contraste propuesto, ajustando la iluminación de las imágenes previo a la aplicación de los FIS de mejora el contraste. Ya que, como se mencionó en la sección de resultados, al comprar los datos obtenidos en la matriz de confusión de la figura 5.7 se observa un aumento representativo en el desempeño de los modelos entrenados al realizar la inferencia sobre imágenes con una adecuada iluminación.
- Explorar alternativas de hardware para superar las limitaciones actuales del entrenamiento, permitiendo, entre otras cosas, utilizar un mayor número de imágenes e incrementar el número de épocas de entrenamiento.

- Adaptar el sistema para la detección de otras patologías, no solo retinopatía diabética.
- Explorar el entrenamiento con la versión más reciente de YOLO y con otras arquitecturas para verificar su viabilidad de implementación sobre un sistema embebido.
- Mejorar la interfaz de usuario para que ésta pueda ejecutarse de una manera más sencilla, mostrar el resultado de ambos procedimientos dentro de la misma interfaz y permita guardar los resultados en cualquier directorio del dispositivo. En otras palabras, que sea más amigable para facilitar al personal médico el uso del sistema.

Productos de la tesis

Con parte del trabajo realizado en esta tesis, se desarrolló el artículo titulado *Eye Fundus Image Processing Using Fuzzy Logic* [106], el cual fue presentado en el congreso 19th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), 2022.

Bibliografía

[1] Xilinx. «What is an FPGA?» Accedido el 02 de Marzo 2022. (2022), dirección: `https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html`.

[2] S. Carey y R. Diamond, «From piecemeal to configurational representation of faces,» *Science*, vol. 195, n.º 4275, págs. 312-314, 1977.

[3] K. Zhang, W. Zuo, S. Gu y L. Zhang, «Learning deep CNN denoiser prior for image restoration,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, págs. 3929-3938.

[4] C. Singh y J. Singh, «Geometrically invariant color, shape and texture features for object recognition using multiple kernel learning classification approach,» *Information Sciences*, vol. 484, págs. 135-152, 2019.

[5] H. Hagiwara, K. Asami y M. Komori, «Real-time image processing system by using FPGA for service robots,» en *The 1st IEEE Global Conference on Consumer Electronics 2012*, IEEE, 2012, págs. 720-723.

[6] Y. Zhang, J. Yu, Y. Chen, W. Yang, W. Zhang e Y. He, «Real-time strawberry detection using deep neural networks on embedded system (rtsd-net): An edge AI application,» *Computers and Electronics in Agriculture*, vol. 192, pág. 106 586, 2022.

[7] A. Bochkovskiy, C.-Y. Wang y H.-Y. M. Liao, *YOLOv4: Optimal Speed and Accuracy of Object Detection*, 2020. arXiv: `2004.10934 [cs.CV]`.

[8] J. C. Vieira, A. Sartori, S. F. Stefenon, F. L. Perez, G. S. De Jesus y V. R. Q. Leithardt, «Low-cost CNN for automatic violence recognition on embedded system,» *IEEE Access*, vol. 10, págs. 25 190-25 202, 2022.

[9] U. Dirección General de Comunicación Social. «Boletín de Prensa: Publicación Especial Núm. 966.» Accedido el 22-12-05. (2021), dirección: https://www.dgcs.unam.mx/boletin/bdboletin/2021_966.html.

[10] P. d. l. S. Gobierno de México Secretaría de Salud. «Diabetes en México.» Accedido el 22-12-05. (2021), dirección: https://www.gob.mx/promosalud/acciones-y-programas/diabetes-en-mexico-284509.

[11] S. d. S. Gobierno de México. «Retinopatía Diabética o Ceguera Irreversible por Inadecuado Control de la Diabetes.» Accedido el 22-12-05. (2018), dirección: https://www.gob.mx/salud/documentos/retinopatia-diabetica-o-ceguera-irreversible-por-inadecuado-control-de-la-diabetes.

[12] D. M. Doblado. «Los 10 lenguajes de programación más populares para 2024.» (dic. de 2024), dirección: https://www.cio.mx/noticio/articulo.php?se=1571256&s=21823&gh=1178442.

[13] B. McCormick, R. Borovec, J. Read y R. Amendola, «Image processor for biomedical research,» en *Computers in Life Science Research*, Springer, 1974, págs. 129-135.

[14] E. Peli y T. Peli, «Restoration of retinal images obtained through cataracts,» *IEEE transactions on medical imaging*, vol. 8, n.º 4, págs. 401-406, 1989.

[15] A. W. Setiawan, T. R. Mengko, O. S. Santoso y A. B. Suksmono, «Color retinal image enhancement using CLAHE,» en *International conference on ICT for smart society*, IEEE, 2013, págs. 1-3.

[16] M. Zhou, K. Jin, S. Wang, J. Ye y D. Qian, «Color retinal image enhancement based on luminosity and contrast adjustment,» *IEEE Transactions on Biomedical engineering*, vol. 65, n.º 3, págs. 521-527, 2017.

[17] G. D. Joshi y J. Sivaswamy, «Colour retinal image enhancement based on domain knowledge,» en *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, IEEE, 2008, págs. 591-598.

[18] D. Almeida-Galárraga, K. Benavides-Montenegro, E. Insuasti-Cruz et al., «Glaucoma detection through digital processing from fundus images using MATLAB,» en *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)*, IEEE, 2021, págs. 39-45.

[19] M. J. Van Grinsven, B. van Ginneken, C. B. Hoyng, T. Theelen y C. I. Sánchez, «Fast convolutional neural network training using selective data sampling: Application to hemorrhage detection in color fundus images,» *IEEE transactions on medical imaging*, vol. 35, n.º 5, págs. 1273-1284, 2016.

[20] L.-P. Cen, J. Ji, J.-W. Lin et al., «Automatic detection of 39 fundus diseases and conditions in retinal photographs using deep neural networks,» *Nature communications*, vol. 12, n.º 1, pág. 4828, 2021.

[21] R. Afrin y P. C. Shill, «Automatic lesions detection and classification of diabetic retinopathy using fuzzy logic,» en *2019 International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*, IEEE, 2019, págs. 527-532.

[22] L. Upton, *The Raspberry Pi Launch - Raspberry Pi*, mar. de 2012. dirección: `https://www.raspberrypi.com/news/the-raspberry-pi-launch/`.

[23] R. P. Foundation. «Raspberry Pi Documentation: Processors.» (feb. de 2022), dirección: `https://www.raspberrypi.com/documentation/computers/processors.html`.

[24] C. J. Swinney y J. C. Woods, «Low-Cost Raspberry-Pi-Based UAS Detection and Classification System Using Machine Learning,» *Aerospace*, vol. 9, n.º 12, pág. 738, 2022.

[25] L. A. Zadeh, «Soft computing and fuzzy logic,» *IEEE software*, vol. 11, n.º 6, págs. 48-56, 1994.

[26] L. A. Zadeh, «Fuzzy sets,» *Information and control*, vol. 8, n.º 3, págs. 338-353, 1965.

[27] S. Roy y U. Chakraborty, *Introduction to soft computing: neuro-fuzzy and genetic algorithms*. Pearson, 2013, pág. 5.

[28] O. A. M. Ali, A. Y. Ali y B. S. Sumait, «Comparison between the effects of different types of membership functions on fuzzy logic controller performance,» *International Journal*, vol. 76, págs. 76-83, 2015.

[29] S. A. Loan, A. M. Murshid, A. C. Shakir, A. R. Alamoud y S. A. Abbasi, «A Novel VLSI Architecture of a Weighted Average Method based Defuzzifier Unit,» en *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 2, 2014, págs. 12-14.

[30] W. S. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity,» *The bulletin of mathematical biophysics*, vol. 5, págs. 115-133, 1943.

[31] F. Rosenblatt, «The perceptron: a probabilistic model for information storage and organization in the brain.,» *Psychological review*, vol. 65, n.º 6, pág. 386, 1958.

[32] D. E. Rumelhart, G. E. Hinton, R. J. Williams et al., *Learning internal representations by error propagation*, 1985.

[33] M. T. Hagan, H. B. Demuth y M. Beale, *Neural network design*. PWS Publishing Co., 1997.

[34] J.-W. Lin, «Artificial neural network related to biological neuron network: a review,» *Advanced Studies in Medical Sciences*, vol. 5, n.º 1, págs. 55-62, 2017.

[35] S. KILIÇARSLAN, A. Kemal y M. Çelik, «An overview of the activation functions used in deep learning algorithms,» *Journal of New Results in Science*, vol. 10, n.º 3, págs. 75-88, 2021.

[36] M. L. Minsky y S. A. Papert, «An Introduction to Computational Geometry,» 1969.

[37] P. Werbos, «Beyond regression: New tools for prediction and analysis in the behavioral sciences,» *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.

[38] R. M. Cichy y D. Kaiser, «Deep neural networks as scientific models,» *Trends in cognitive sciences*, vol. 23, n.º 4, págs. 305-317, 2019.

[39] C. G. Rafael y E. W. Richard, *Digital image processing*. Pearson education, 2018, pág. 945.

[40] K. R. Gegenfurtner y D. C. Kiper, «Color vision,» *Annual review of neuroscience*, vol. 26, n.º 1, págs. 181-206, 2003.

[41] 2024. dirección: http://hyperphysics.phy-astr.gsu.edu/hbasees/vision/cie.html#:~:text=Diagrama%20de%20Cromaticidad%20C.I.E.,-Sobre%20el%20diagrama&text=El%20borde%20representa%20la%20m%C3%A1xima,de%20todos%20los%20tonos%20perceptibles.&text=Se%20pueden%20asignar%20colores%20aproximados%20a%20%C3%A1reas%20del%20diagrama%20de%20cromaticidad%20CIE..

[42] S. Süsstrunk, R. Buckley y S. Swen, «Standard RGB color spaces,» en *Color and imaging conference*, Society of Imaging Science y Technology, vol. 7, 1999, págs. 127-134.

[43] C. A. Poynton, *A technical introduction to digital video*. USA: John Wiley & Sons, Inc., 1996, ISBN: 047112253X.

[44] G. A. Agoston, *Color theory and its application in art and design*. Springer, 2013, vol. 19.

[45] B. Payette, «Color Space Converter: R'G'B'to Y'CbCr,» *Xilinx Application Note*, 2002.

[46] E. Welch, R. Moorhead y J. Owens, «Image processing using the HSI color space,» en *IEEE Proceedings of the SOUTHEASTCON'91*, IEEE, 1991, págs. 722-725.

[47] R. Alvarado-Cervantes, «Segmentación de imágenes digitales sobre la base de la información integral del color,» Tesis doct., feb. de 2017. DOI: 10.13140/RG.2.2.21869.28640.

[48] C. G. Rafael y E. W. Richard, *Digital image processing*. Pearson education, 2018, pág. 419.

[49] W. contributors, *CIELAB color space*, Accedido el 22-12-05. dirección: https://en.wikipedia.org/wiki/CIELAB_color_space.

[50] G. Wilfrido y Flores, *Análisis de Imágenes Digitales Transformaciones de intensidad*. dirección: https://www.tamps.cinvestav.mx/~wgomez/diapositivas/AID/Clase03.pdf.

[51] K. G. Dhal, A. Das, S. Ray, J. Gálvez y S. Das, «Histogram equalization variants as optimization problems: a review,» *Archives of Computational Methods in Engineering*, vol. 28, págs. 1471-1496, 2021.

[52] Y.-T. Kim, «Contrast enhancement using brightness preserving bi-histogram equalization,» *IEEE transactions on Consumer Electronics*, vol. 43, n.º 1, págs. 1-8, 1997.

[53] L. Lu, Y. Zhou, K. Panetta y S. Agaian, «Comparative study of histogram equalization algorithms for image enhancement,» *Mobile Multimedia/Image Processing, Security, and Applications 2010*, vol. 7708, págs. 337-347, 2010.

[54] J. V. Haxby, L. G. Ungerleider, B. Horwitz, J. M. Maisog, S. I. Rapoport y C. L. Grady, «Face encoding and recognition in the human brain.,» *Proceedings of the National Academy of Sciences*, vol. 93, n.º 2, págs. 922-927, 1996.

[55] C. G. Rafael y E. W. Richard, *Digital image processing*. Pearson education, 2018, pág. 966.

[56] K. Fukushima, S. Miyake y T. Ito, «Neocognitron: A neural network model for a mechanism of visual pattern recognition,» *IEEE transactions on systems, man, and cybernetics*, n.º 5, págs. 826-834, 1983.

[57] Y. LeCun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE*, vol. 86, n.º 11, págs. 2278-2324, 1998.

[58] A. Krizhevsky, I. Sutskever y G. E. Hinton, «Imagenet classification with deep convolutional neural networks,» *Advances in neural information processing systems*, vol. 25, 2012.

[59] K. Simonyan y A. Zisserman, «Very deep convolutional networks for large-scale image recognition,» *arXiv preprint arXiv:1409.1556*, 2014.

[60] K. He, X. Zhang, S. Ren y J. Sun, «Deep residual learning for image recognition,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, págs. 770-778.

[61] J. Jiang, L. Zheng, F. Luo y Z. Zhang, «Rednet: Residual encoder-decoder network for indoor rgb-d semantic segmentation,» *arXiv preprint arXiv:1806.01054*, 2018.

[62] C. Szegedy, W. Liu, Y. Jia et al., «Going deeper with convolutions,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, págs. 1-9.

[63] Y. Ibrahim, H. Wang y K. Adam, «Analyzing the reliability of convolutional neural networks on gpus: Googlenet as a case study,» en *2020 International Conference on Computing and Information Technology (ICCIT-1441)*, IEEE, 2020, págs. 1-6.

[64] D. Aliseda y L. Berástegui, «Retinopatía diabética,» en *Anales del sistema sanitario de Navarra*, SciELO Espana, vol. 31, 2008, págs. 23-34.

[65] A. Budai, *High-Resolution Fundus (HRF) Image Database*, Available online: https://www5.cs.fau.de/research/data/fundus-images/ (accessed on 29 March 2022).

[66] MIRS, *MESSIDOR Image Database*, Available online: https://www.adcis.net/en/third-party/messidor/, 2004.

[67] J. Emma Dugas Jared, *Diabetic Retinopathy Detection*, Available online: https://www.kaggle.com/c/diabetic-retinopathy-detection/data, 2015. dirección: https://kaggle.com/competitions/diabetic-retinopathy-detection.

[68] S. Sharma y A. Bhatia, «Contrast enhancement of an image using fuzzy logic,» *International Journal of Computer Applications*, vol. 111, n.º 17, 2015.

[69] S. Joshi y S. Kumar, «Image contrast enhancement using fuzzy logic,» *arXiv preprint arXiv:1809.04529*, 2018.

[70] S. Fernandes, H. Vashi, A. Shetty y V. Kelkar, «Adaptive contrast enhancement using fuzzy logic,» en *2019 International Conference on Advances in Computing, Communication and Control (ICAC3)*, IEEE, 2019, págs. 1-6.

[71] A. W. Setiawan, T. R. Mengko, O. S. Santoso y A. B. Suksmono, «Color retinal image enhancement using CLAHE,» en *International conference on ICT for smart society*, IEEE, 2013, págs. 1-3.

[72] M. Zhou, K. Jin, S. Wang, J. Ye y D. Qian, «Color retinal image enhancement based on luminosity and contrast adjustment,» *IEEE Transactions on Biomedical engineering*, vol. 65, n.º 3, págs. 521-527, 2017.

[73] L. Cao, H. Li e Y. Zhang, «Retinal image enhancement using low-pass filtering and α-rooting,» *Signal Processing*, vol. 170, pág. 107 445, 2020.

[74] S. V. Paranjape, S. Ghosh, A. Ray, J. Chatterjee y A. R. Lande, «A Modified Fuzzy Contrast Enhancement Technique for Retinal Images,» en *2015 International Conference on Computing Communication Control and Automation*, IEEE, 2015, págs. 892-896.

[75] N. S. Datta, H. S. Dutta y K. Majumder, «Brightness-preserving fuzzy contrast enhancement scheme for the detection and classification of diabetic retinopathy disease,» *Journal of medical imaging*, vol. 3, n.º 1, págs. 014 502-014 502, 2016.

[76] B. M. K. Younis y D. B. Younis, «Fuzzy Image Processing Based Architecture for Contrast Enhancement in Diabetic Retinopathy Images,» *International Journal of Computer Engineering and Information Technology*, vol. 12, n.º 4, págs. 26-30, 2020.

[77] scikit-image contributors. «scikit-image: Image processing in Python.» Consultado el 31 de marzo de 2023. (), dirección: `https://scikit-image.org/docs/stable/api/skimage.color.html#skimage.color.rgb2ycbcr`.

[78] C. Reshmalakshmi y M. Sasikumar, «Image contrast enhancement using fuzzy technique,» en *2013 International Conference on Circuits, Power and Computing Technologies (ICCPCT)*, IEEE, 2013, págs. 861-865.

[79] A. G. Howard, M. Zhu, B. Chen et al., «Mobilenets: Efficient convolutional neural networks for mobile vision applications,» *arXiv preprint arXiv:1704.04861*, 2017.

[80] G. Huang, Z. Liu, L. Van Der Maaten y K. Q. Weinberger, «Densely connected convolutional networks,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, págs. 4700-4708.

[81] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, «You only look once: Unified, real-time object detection,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, págs. 779-788.

[82] H. Jung, M.-K. Choi, J. Jung, J.-H. Lee, S. Kwon y W. Young Jung, «ResNet-based vehicle classification and localization in traffic surveillance systems,» en *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, págs. 61-67.

[83] R. Mirchandani, C. Yoon, S. Prakash et al., «Comparing the Architecture and Performance of AlexNet Faster R-CNN and YOLOv4 in the Multiclass Classification of Alzheimer Brain MRI Scans,» 2021.

[84] R. Girshick, J. Donahue, T. Darrell y J. Malik, «Rich feature hierarchies for accurate object detection and semantic segmentation,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, págs. 580-587.

[85] J. Redmon, *Darknet: Open Source Neural Networks in C*, `http://pjreddie.com/darknet/`, 2016.

[86] D. Thuan, «Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detention algorithm,» 2021.

[87] J. Terven, D.-M. Córdova-Esparza y J.-A. Romero-González, «A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas,» *Machine Learning and Knowledge Extraction*, vol. 5, n.º 4, págs. 1680-1716, 2023.

[88] Z. Keita, *YOLO Object Detection Explained*, Consultado el 31 de marzo de 2023, sep. de 2022. dirección: `https://www.datacamp.com/blog/yolo-object-detection-explained`.

[89] J. Redmon y A. Farhadi, «YOLO9000: better, faster, stronger,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, págs. 7263-7271.

[90] J. Redmon y A. Farhadi, «Yolov3: An incremental improvement,» *arXiv preprint arXiv:1804.02767*, 2018.

[91] K. He, X. Zhang, S. Ren y J. Sun, «Spatial pyramid pooling in deep convolutional networks for visual recognition,» *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, n.º 9, págs. 1904-1916, 2015.

[92] A. Bochkovskiy, C.-Y. Wang y H.-Y. M. Liao, «Yolov4: Optimal speed and accuracy of object detection,» *arXiv preprint arXiv:2004.10934*, 2020.

[93] S. Liu, L. Qi, H. Qin, J. Shi y J. Jia, «Path aggregation network for instance segmentation,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, págs. 8759-8768.

[94] G. Jocher, *Ultralytics YOLOv5*, ver. 7.0, 2020. DOI: `10.5281/zenodo.3908559`. dirección: `https://github.com/ultralytics/yolov5`.

[95] S. W. Glenn Jocher, *Architecture Summary*, 2023. dirección: https://docs.ultralytics.com/yolov5/tutorials/architecture_description/.

[96] Y. Bengio, I. Goodfellow y A. Courville, *Deep learning*. MIT press Cambridge, MA, USA, 2017, vol. 1.

[97] G. Jocher, *Train Custom Data*, 2023. dirección: https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/.

[98] L. Muñiz García, «Viabilidad y rendimiento de YOLOv5 en Raspberry Pi,» 2021.

[99] G. Jocher, *yolov5/data/hyps/hyp.scratch.yaml*, 2020. dirección: https://github.com/ultralytics/yolov5/blob/ed887b5976d94dc61fa3f7e8e07170623dc7d6ee/data/hyps/hyp.scratch.yaml.

[100] G. J. Jirka Borovec, *YOLOv5 Hyps*, 2021. dirección: https://github.com/ultralytics/yolov5/tree/ed887b5976d94dc61fa3f7e8e07170623dc7d6ee/data/hyps.

[101] G. J. Jirka Borovec, *YOLOv5 Hyperparameter Low*, 2021. dirección: https://github.com/ultralytics/yolov5/blob/master/data/hyps/hyp.scratch-low.yaml.

[102] M. Qiu, L. Huang y B.-H. Tang, «ASFF-YOLOv5: Multielement detection method for road traffic in UAV images based on multiscale feature fusion,» *Remote Sensing*, vol. 14, n.º 14, pág. 3498, 2022.

[103] T. Lewick, M. Kumar, R. Hong y W. Wu, «Intracranial hemorrhage detection in CT scans using deep learning,» en *2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService)*, IEEE, 2020, págs. 169-172.

[104] L. D. Glenn Jocher, *Raspberry Pi*, 2023. dirección: https://docs.ultralytics.com/es/guides/raspberry-pi/.

[105] 2024. dirección: https://www.amd.com/es/products/specifications/processors.html.

[106] I. Chong-Cervantes, A. Anzueto-Ríos, J. A. Moreno-Cadenas, M. A. Reyes-Barranca y L. M. Flores-Nava, «Eye Fundus Image Processing Using Fuzzy Logic,» en *2022 19th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, IEEE, 2022, págs. 1-6.