



UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE CIENCIAS EXACTAS Y NATURALES

Maestría en Explotación de Datos y Descubrimiento de Conocimiento

Módulo de detección de intrusiones para entornos rurales utilizando Edge Computer Vision en microcontroladores

Tesis presentada para optar al título de Magíster de la
Universidad de Buenos Aires en Explotación de Datos y
Descubrimiento de Conocimiento

Ing. Boris Dorian Da Silva

Director de Tesis: Dr. Marcelo A. Soria

Fecha de Defensa: 02/07/2025

Buenos Aires, 2025

Resumen

Este trabajo se centró en el desarrollo de un módulo de detección de intrusiones mediante la aplicación de *Edge Computer Vision*, especialmente diseñado para su implementación en microcontroladores de bajo costo, con aplicaciones específicas en entornos rurales con limitaciones de acceso a la red eléctrica y a la infraestructura de comunicación.

La solución propuesta comprende tres módulos esenciales: detección de personas, comunicación y suministro energético. Para el módulo de detección, se elaboró un clasificador utilizando una red neuronal convolucional basado en la arquitectura MobileNet V1, implementando una estrategia de transferencia de conocimiento para adaptar el modelo al contexto de detección de personas. El modelo resultante fue cuantizado para reducir su tamaño y mejorar la eficiencia en los tiempos de inferencia, para posteriormente ser implementado en un microcontrolador ESP32. El módulo de comunicación utiliza un transceptor LoRa para transmitir alertas generadas por la detección de personas, y el módulo de suministro energético incorpora un sistema con panel solar, batería y módulo de carga. Esta solución integral ofrece una respuesta para la detección de personas en áreas rurales con recursos limitados.

Palabras Claves: Redes Neuronales Artificiales, Redes Neuronales Convolucionales, Edge Computer Vision, Microcontroladores, ESP32, Lora

Abstract

Intrusion Detection Module for Rural Environments Using Edge Computer Vision on Microcontrollers

This work focused on the development of an intrusion detection module using Edge Computer Vision, specially designed for implementation in low-cost microcontrollers, with specific applications in rural environments with limited access to the electrical grid and communication infrastructure.

The proposed solution comprises three essential modules: person detection, communication, and power supply. For the detection module, a classifier was developed using a convolutional neural network based on the MobileNet V1 architecture, implementing a knowledge transfer strategy to adapt the model to the context of person detection. The resulting model was quantized to reduce its size and improve efficiency in inference times, and then implemented on an ESP32 microcontroller. The communication module uses a LoRa transceiver to transmit alerts generated by person detection, and the power supply module incorporates a system with solar panel, battery, and charging module. This comprehensive solution provides a response to person detection in rural areas with limited resources.

Keywords: Artificial Neural Networks, Convolutional Neural Networks, Edge Computer Vision, Microcontrollers, ESP32, Lora

Agradecimientos

A Marcelo Soria por la confianza, su buena predisposición y apoyo brindado para llevar a cabo este trabajo.

A todos los docentes de la maestría por su excelente enseñanza y por transmitirme su pasión por el conocimiento.

A las distintas autoridades de la maestría que siempre estuvieron a disposición para resolver mis dudas y brindarme la asistencia necesaria.

A mi familia por siempre apoyarme en cada etapa de mi vida. Les dedico este trabajo.

A los amigos que conocí en la cursada de la Maestría, les agradezco por su compañerismo y por los momentos compartidos.

Indice

Módulo de detección de intrusiones para entornos rurales utilizando Edge Computer Vision en microcontroladores.....	1
1. Introducción.....	7
1.1 Tecnologías para la seguridad y la vigilancia aplicables a producciones rurales.....	7
1.2 Alternativas de Detección de Intrusiones en Zonas Rurales en el Mercado Argentino.....	8
1.3 Sistemas de vigilancia basados en aplicaciones de modelos de computer vision en microcontroladores.....	9
1.4 Redes neuronales artificiales.....	10
1.4.1 Componentes de las redes neuronales.....	11
1.4.2 ¿Cómo funcionan las redes neuronales artificiales?.....	12
1.4.3 Tipos de redes neuronales artificiales.....	13
1.4.3.1 Redes Neuronales Convolucionales (CNN).....	15
1.4.4 Transferencia de aprendizaje.....	21
1.4.5 MobileNet.....	22
1.5 Edge Computer Vision.....	23
1.6 Microcontroladores.....	24
1.6.1 Elementos internos de un microcontrolador.....	25
1.6.2 Placas de desarrollo de microcontroladores.....	26
1.6.3 Placa de desarrollo ESP32-CAM.....	26
1.7 Cuantización de modelos.....	28
1.7.1 Antecedentes sobre la Representación de Punto Flotante.....	28
1.7.2 Cuantización de 8 bits.....	30
1.8 Tensorflow Lite.....	33
1.9 Lora y LoraWan.....	34
1.10 Visual Wake Words Dataset.....	38
1.11 Técnicas de preprocesamiento de datos.....	39
1.11.1 Conversión de imágenes RGB a escala de grises.....	39
1.11.2 Reducción de imágenes.....	43
1.11.3 Data augmentation.....	43
1.12 Objetivo.....	45
1.12.1 Objetivo general.....	45
1.12.2 Objetivos específicos.....	45
2. Materiales y métodos.....	46
2.1 Hardware.....	46
2.2 Software.....	50
2.3 Recursos Cloud.....	51
3. Desarrollo y resultados.....	52
3.1 Descripción general de la solución.....	52
3.1.1 Módulo de detección de personas.....	53
3.1.1.1 Lógica implementada en el módulo ESP32-CAM.....	54
3.1.2 Módulo de Comunicación.....	58
3.1.3 Módulo de suministro energético.....	61
3.1.4 Descripción de costos de construcción.....	62
3.2 Modelo de detección de personas.....	63
3.2.1 Adquisición y preprocesamiento de datos.....	63

3.2.1.1 Datasets.....	63
3.2.1.2 Preprocesamiento de datos.....	66
3.2.2 Diseño de la arquitectura.....	69
3.2.2.1 Configuración del modelo MobileNet.....	70
3.2.2.2 Capas de Preprocesamiento.....	72
3.2.3 Exploración de arquitecturas y estrategias de entrenamiento.....	72
3.2.4 Clasificadores.....	75
3.2.5 Cuantización del modelo posterior al entrenamiento.....	77
3.3 Análisis integrado.....	78
3.3.1 Análisis de performance y selección del modelo de detección de personas.....	78
3.3.1.1 Selección del modelo implementado.....	78
3.3.1.2 Análisis del modelo con la cuantización aplicada.....	81
3.3.2 Análisis de performance del módulo de comunicación.....	83
3.3.3 Análisis de performance del módulo de suministro energético.....	85
4. Conclusiones y trabajos futuros.....	85
4.1 Conclusiones.....	85
4.2 Trabajos futuros.....	87
5. Referencias bibliográficas.....	88
6. Anexo A: Funciones de activación.....	94
7. Anexo B: Arquitectura y estrategia de entrenamiento de los clasificadores.....	97
7.1 Clasificador A.....	97
7.2 Clasificador B.....	98
7.3 Clasificador C.....	99
7.4 Clasificador D.....	100
7.5 Clasificador E.....	102
7.6 Clasificador F.....	103
7.7 Clasificador G.....	104
7.8 Clasificador H.....	105
7.9 Clasificador I.....	106
8. Anexo C: Análisis de performance de los clasificadores.....	108
8.1 Análisis de performance: Clasificador A.....	108
8.2 Análisis de performance: Clasificador B.....	110
8.3 Análisis de performance: Clasificador C.....	112
8.4 Análisis de performance: Clasificador D.....	114
8.5 Análisis de performance: Clasificador E.....	116
8.6 Análisis de performance: Clasificador F.....	118
8.7 Análisis de performance: Clasificador G.....	121
8.8 Análisis de performance: Clasificador H.....	123
8.9 Análisis de performance: Clasificador I.....	125
9. Anexo D: Experimentos adicionales.....	127
9.1 Entrenamiento con diferentes valores de la capa DropOut.....	127
9.2 Entrenamiento con diferentes valores de tasa de aprendizaje.....	128

1. Introducción

1.1 Tecnologías para la seguridad y la vigilancia aplicables a producciones rurales

El sector agropecuario desempeña un papel crucial en la economía argentina, ya que genera aproximadamente el 24% de los empleos privados (Ariño et al., 2022). Sin embargo, al igual que en cualquier otro sector, se enfrenta a diversos desafíos en ciertas regiones del país, tales como condiciones climáticas extremas, falta de infraestructura e inseguridad.

Con respecto al último punto, la gran dificultad para mitigar la inseguridad está principalmente relacionado con las grandes extensiones de los campos, donde es difícil poder monitorear y detectar personas no autorizadas las cuales pueden tener intenciones de realizar algún hecho delictivo, por ejemplo, abigeato, robo de producción, sustracción de herramientas o maquinarias, toma de campos, vandalismo, rotura de silobolsas, etc. En áreas donde la infraestructura es escasa en términos de vías de acceso, comunicaciones, suministro eléctrico, entre otros, la complejidad se ve agravada.

Esta problemática tiene un impacto directo en las economías regionales. Además de las pérdidas generadas por los hechos delictivos en sí, también se observa un efecto desmotivador en las inversiones en estas zonas. En algunos casos, esta situación lleva a los productores a verse obligados a reducir o incluso interrumpir por completo su actividad productiva. Esta consecuencia adicional agrava aún más la situación, ya que se pierde no solo el potencial productivo, sino también la generación de empleo y el impulso económico que podría brindar a la región. Por lo tanto, es crucial encontrar soluciones efectivas y accesibles para mitigar esta problemática y brindar un entorno seguro y propicio para el desarrollo de las actividades rurales.

En términos de tecnología, hay diversas opciones disponibles en el mercado para mejorar la seguridad y detectar intrusiones en áreas restringidas. Estas soluciones incluyen sensores de movimiento, barreras infrarrojas y sistemas de cámaras que utilizan Inteligencia Artificial para identificar personas, entre otros. No obstante, en zonas con infraestructura de comunicaciones limitada o inexistente, y donde el acceso a la energía eléctrica no siempre es posible, la implementación de estas tecnologías resulta inviable o costosa.

Un ejemplo de tecnología que ha crecido significativamente en los últimos años en la detección de intrusos es la aplicación de modelos de inteligencia artificial en

sistemas de monitoreo mediante videocámaras. Estos sistemas suelen consistir en una red de cámaras que envían las imágenes captadas a una central de cómputo (con grandes capacidades de procesamiento y memoria), donde se procesan en busca de posibles intrusos. Sin embargo, la implementación de tales sistemas se vuelve económicamente inviable en entornos con grandes extensiones geográficas y limitaciones en la red eléctrica y el acceso a internet. Estas limitaciones obstaculizan tanto la transmisión de las imágenes como el suministro de energía necesario para el funcionamiento del sistema.

Por otro lado, tenemos los microcontroladores que desempeñan un papel fundamental en numerosos componentes de sistemas de seguridad, como centrales de alarma y sensores, entre otros. Estos dispositivos se destacan por su coste asequible, bajo consumo de energía y facilidad de integración, lo que los convierte en una elección popular en este campo. Pero al mismo tiempo, presentan una limitación en cuanto a su capacidad de memoria y procesamiento, lo que dificulta la implementación de modelos de *Machine Learning* (Aprendizaje Automático) en estos dispositivos. Sin embargo, existe un paradigma llamado *Tiny Machine Learning*, el cual se basa en el desarrollo de modelos de *Machine Learning* especialmente diseñados para operar en dispositivos con pocos recursos. Estos modelos resultan ideales para su implementación en microcontroladores, ya que aprovechan de manera óptima los recursos disponibles. De esta forma, se supera el desafío de utilizar *Machine Learning* en estos dispositivos. (Gordon et al., 2020)

1.2 Alternativas de Detección de Intrusiones en Zonas Rurales en el Mercado Argentino

Las soluciones más utilizadas para el monitoreo de intrusiones en zonas rurales consisten en la utilización de cámaras de seguridad IP con conexión 4G y alimentación energética por panel solar y batería.

Existen algunos modelos más nuevos que se ofrecen en el mercado argentino que aplican *Edge Computer Vision* en la cámara ofreciendo la capacidad de detectar automáticamente la presencia de personas en el mismo dispositivo y enviar la alerta.

Un ejemplo es la cámara marca “Hikvision” modelo “DS-2XS6A87G1-L/C32S80” el cual consta de esta funcionalidad (Fig. 1.1). (Hikvision ofrece cámaras solares para monitorear grandes extensiones sin necesidad de energía eléctrica, 2022)



Figura 1.1: Cámara marca “Hikvision” modelo “DS-2XS6A87G1-L/C32S80”. Tomado de Hikvision (sin fecha).

A pesar de que sus prestaciones son una muy buena alternativa para diferentes contextos, en situaciones donde se necesiten varios de estos dispositivos, su alto costo puede dificultar su implementación. Al día de hoy y en el mercado argentino el precio de este producto ronda los U\$S 3300. Y en situaciones donde no haya cobertura 4G, estos valores se pueden incrementar debido a la necesidad de sumar equipos de comunicación que permitan suplir esta limitación.

Existen otras alternativas que se basan en diferentes tipos de sensores clásicos, como barreras láser, sensores PIR (infrarrojo pasivos), sensores de microondas, entre otros. Sin embargo, estas tecnologías en ciertas situaciones tienden a generar falsas alarmas (Mechler, 2014), y en contextos donde un sensor se encuentre a varios kilómetros de distancia, el costo de una falsa alarma puede ser alto.

1.3 Sistemas de vigilancia basados en aplicaciones de modelos de computer vision en microcontroladores

En el campo de *Edge Computer Vision* en microcontroladores, se han realizado

diferentes investigaciones que abordan la detección de personas. Una de ellas es la descrita en el trabajo de Wulfert et al. (2022). En este trabajo se implementa un modelo de detección de personas en un microcontrolador ESP32, empleando una red neuronal artificial de tipo Feedforward. Canepa et al.(2022) presentan una solución similar de vigilancia basada en detección de personas, pero esta vez desplegada en un microcontrolador STM32. La arquitectura de la red neuronal desarrollada en esta investigación se basa en una red pre-entrenada MobileNet V3. Además, un equipo de investigación de Samsung ha contribuido otra aplicación de un controlador STM32 para la detección de personas (Bisulco et al., 2020)

Estas investigaciones representan avances significativos en el campo de *Edge Computer Vision*, ya que demuestran la viabilidad de implementar modelos de detección de personas en microcontroladores con recursos limitados. Sin embargo, es importante destacar que estas investigaciones se centran en la detección de personas en microcontroladores de manera general, sin abordar una aplicación específica ni considerar un contexto particular.

1.4 Redes neuronales artificiales

Una neurona artificial es un modelo computacional inspirado en las neuronas biológicas. En la naturaleza, las neuronas reciben señales a través de sinapsis ubicadas en las dendritas o en su membrana. Si estas señales superan un umbral determinado, la neurona se activa y emite un impulso a través del axón, el cual puede transmitirse a otras sinapsis y potencialmente activar más neuronas.

En el caso de las neuronas artificiales, esta complejidad biológica se simplifica significativamente. Estas unidades constan de entradas (análogas a las sinapsis), que se multiplican por pesos (que representan la intensidad de cada señal), y luego se procesan mediante una función matemática que determina su activación. Otra función (que puede ser la identidad) genera la salida de la neurona, a veces dependiendo de un umbral específico. Las redes neuronales artificiales combinan estas neuronas para procesar información de manera eficiente.

Cuanto mayor sea el peso de una neurona artificial, mayor será la influencia de la entrada asociada. Los pesos también pueden ser negativos, lo que inhibe la señal. Dependiendo de su configuración, el comportamiento de la neurona varía. Ajustando estos pesos, es posible obtener salidas deseadas para entradas específicas. (Gershenson, 2003)

Las redes neuronales se utilizan en una amplia variedad de tareas, como visión por computadora y reconocimiento de voz, que son difíciles de resolver mediante programación convencional basada en reglas.

1.4.1 Componentes de las redes neuronales

Neuronas

En una red neuronal, sus neuronas se distribuyen en tres grupos distintos (Fig. 1.2):

- Neuronas de Entrada: Estas neuronas son responsables de recibir información procedente del entorno externo.
- Neuronas Ocultas: El procesamiento de la información se lleva a cabo en este grupo de neuronas. Aquí, la información se somete a diversas transformaciones y cálculos.
- Neuronas de Salida: Las neuronas en esta capa generan las conclusiones o resultados de interés.

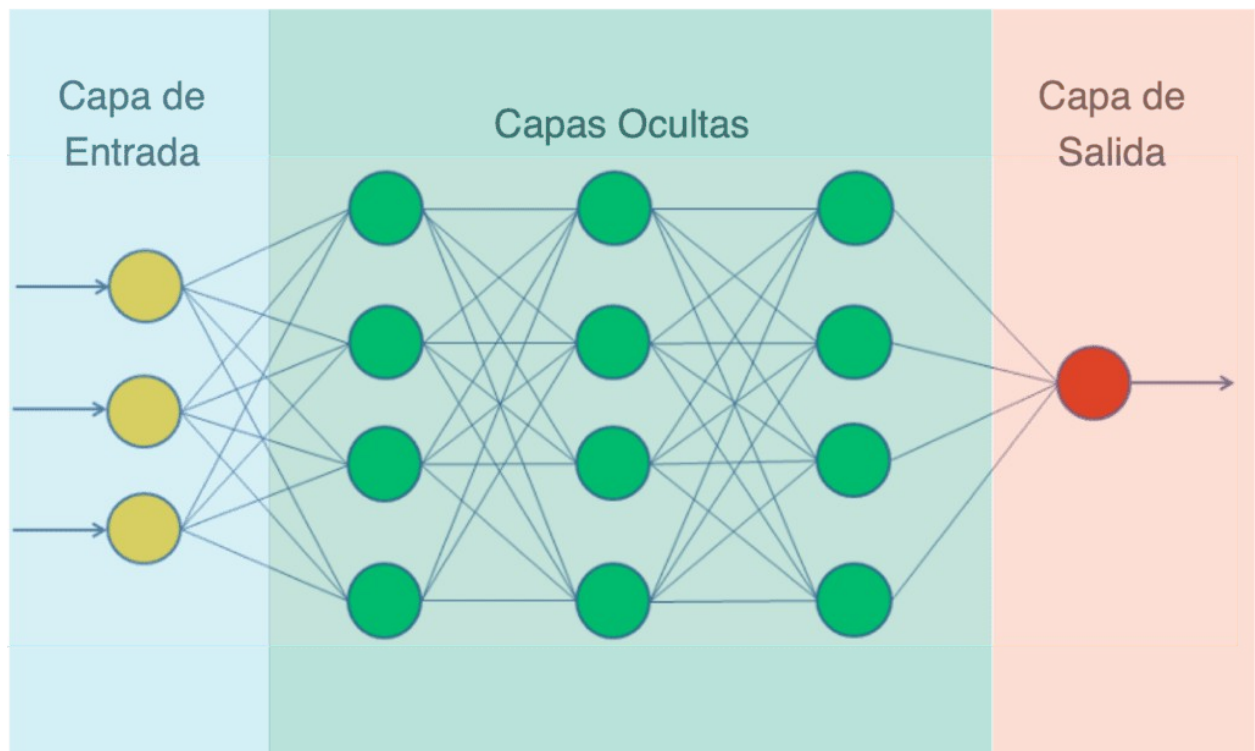


Figura 1.2: Capas de una red neuronal artificial. Tomado de González (2021).

Cuando se trata de redes neuronales más complejas con numerosas neuronas y conexiones entre ellas, estas neuronas se organizan en capas. La primera capa es la de entrada, que recibe la información. Luego, puede haber una o varias capas ocultas que realizan un procesamiento intermedio y finalmente, la capa de salida proporciona los resultados deseados. Cada neurona, sin importar su capa, desempeña una función particular en la transformación de la información de entrada. (González, 2021)

Pesos

Las sinapsis son las conexiones que enlazan neuronas. Cada una de estas conexiones tiene un peso asociado. Estos pesos contribuyen a modificar la información de entrada de las neuronas. En la capa de salida, las neuronas cuyas sinapsis tienen pesos más altos ejercen un mayor impacto en los resultados, mientras que aquellas con pesos más bajos tienen menos influencia. En otras palabras, podríamos afirmar que es la matriz de pesos que rige todo el sistema neuronal.

En el inicio, cuando la red neuronal se inicia por primera vez, los pesos se asignan de manera aleatoria. Sin embargo, posteriormente, es necesario ajustarlas para optimizar el rendimiento de la red (González, 2021).

Sesgo

El Sesgo (o *Bias*) es un parámetro adicional que se utiliza para ajustar y modular la salida de una neurona. Cada neurona en una red neuronal tiene sus conexiones con sus pesos asociados y, además, se agrega un valor de sesgo. El sesgo permite que la neurona haga una compensación o desplazamiento en la función de activación, lo que le permite aprender y modelar relaciones más complejas en los datos.

En términos sencillos, el sesgo permite a una neurona ajustar su salida incluso cuando todas las entradas son iguales a cero. Esto es importante porque puede permitir que la neurona aprenda a activarse incluso cuando no hay una señal evidente en las entradas. Por lo tanto, el sesgo es un componente crítico en una red neuronal que ayuda a modelar y aprender patrones en los datos al permitir desplazamientos y compensaciones en las salidas de las neuronas.

1.4.2 ¿Cómo funcionan las redes neuronales artificiales?

El núcleo fundamental de las redes neuronales artificiales reside en las propias neuronas artificiales. Cada neurona recopila entradas de múltiples neuronas, pondera esas entradas mediante asignación de pesos, realiza una suma ponderada y envía el resultado a una o más neuronas subsiguientes. Estas neuronas artificiales aplican una función de activación a la salida antes de transmitirla a la siguiente capa de neuronas (Fig. 1.3).

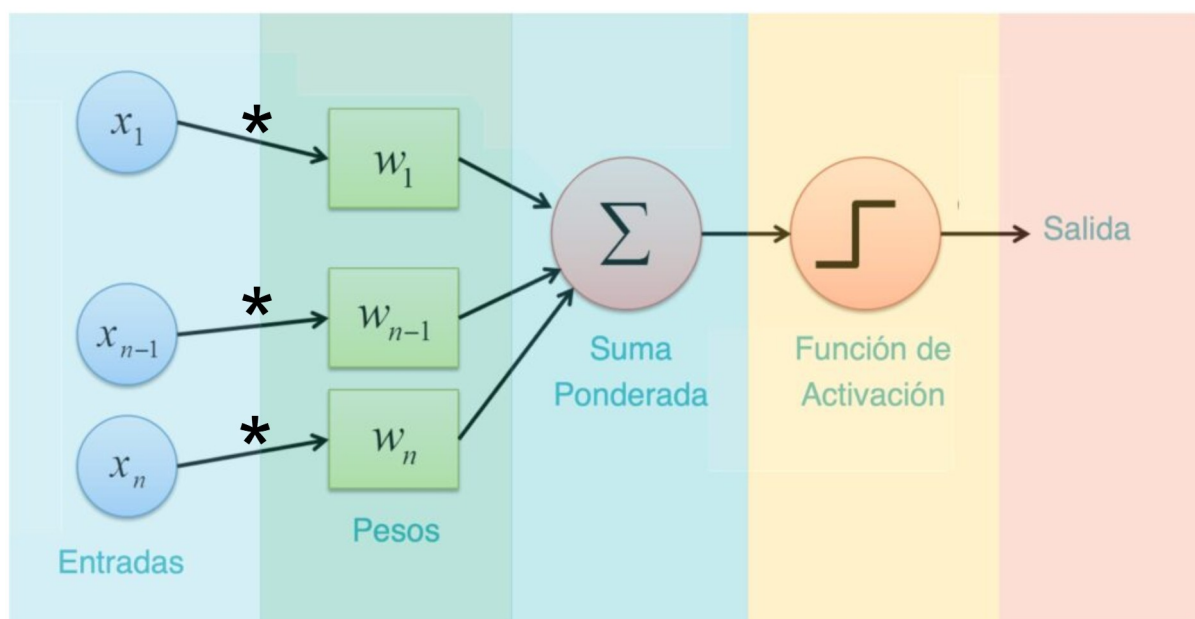


Figura 1.3: Funcionamiento general de una neurona. Modificado de González (2021).

En su esencia, esto puede parecer una operación matemática simple, pero cuando se reúnen cientos, miles e incluso millones de neuronas en diversas capas superpuestas, se forma una red neuronal artificial con la capacidad de realizar tareas extremadamente complejas, como la clasificación de imágenes o el reconocimiento del habla.

Las redes neuronales artificiales constan de varios componentes. En primer lugar, tienen una capa de entrada que recibe datos de diversas fuentes, como archivos, imágenes, sensores de hardware, micrófonos y más. Luego, incluyen una o varias capas ocultas que procesan estos datos y, finalmente, una capa de salida que proporciona uno o varios resultados basados en la función de la red. Por ejemplo, una red neuronal diseñada para detectar personas, automóviles y animales podría tener una capa de salida con tres nodos distintos. Por otro lado, una red que evalúa transacciones bancarias para determinar si son seguras o fraudulentas generalmente tendría una única salida (González, 2021).

1.4.3 Tipos de redes neuronales artificiales

Las redes neuronales artificiales presentan una amplia variedad de arquitecturas, cada una optimizada para resolver problemas particulares. A continuación, se describen algunos de los diseños más representativos. (Kurniawan, 2022)

- **Perceptrón Multicapa:** es una red neuronal artificial con una arquitectura capaz de aproximar cualquier función continua y resolver problemas no linealmente separables. Está compuesto por tres capas principales: una capa de entrada que recibe los datos, una o más capas ocultas que procesan la información (mediante operaciones no lineales y propagación hacia

adelante/atrás), y una capa de salida que genera predicciones o clasificaciones.

- Redes Neuronales Convolucionales (CNN): son arquitecturas especializadas en visión artificial, utilizadas para tareas como clasificación y detección de imágenes. Están compuestas por tres capas principales: convolucional (extrae características visuales básicas como bordes y colores en sus primeras capas, analizando solo regiones locales de la imagen), *pooling* (reduce dimensiones para optimizar el procesamiento) y capas densamente conectadas (conecta todas las neuronas para generar la salida final). Gracias a su eficacia, las CNN tienen aplicaciones cotidianas como reconocimiento facial en redes sociales, detección de objetos en vehículos autónomos y diagnóstico médico por imágenes.
- Redes Neuronales Recurrentes (RNN): son arquitecturas especializadas en procesar datos secuenciales como series de tiempo o lenguaje natural. A diferencia de un perceptrón multicapa convencional, incorporan capas recurrentes que reciclan su propia salida como entrada adicional, permitiendo analizar contextos temporales. Sin embargo, presentan inestabilidad en gradientes al manejar secuencias largas, problema resuelto con variantes avanzadas como LSTM y GRU, cuyas celdas regulan el flujo de información mediante mecanismos de memoria selectiva. Estas redes son fundamentales en aplicaciones como traducción automática, predicción financiera o generación de texto.
- Redes Neuronales Generativa Antagónica (GAN): es una arquitectura compuesta por dos redes neuronales enfrentadas: un generador que crea imágenes falsas a partir de ruido aleatorio, y un discriminador que intenta distinguir entre imágenes reales (del conjunto de entrenamiento) y falsas (generadas). Durante el entrenamiento, el generador mejora progresivamente hasta producir resultados indistinguibles de los reales, mientras el discriminador se vuelve más preciso. Esta técnica se usa ampliamente para generar imágenes, videos y otros datos sintéticos de alta calidad.

Cada arquitectura de red neuronal está diseñada para abordar distintos tipos de problemas. Las MLP (perceptrones multicapa) se utilizan para resolver problemas clásicos que no pueden ser abordados eficazmente mediante técnicas tradicionales de aprendizaje automático. Las CNN (redes neuronales convolucionales) son ideales para tareas de visión por computadora. Las RNN (redes neuronales recurrentes) se emplean en el procesamiento de datos secuenciales, como series temporales o lenguaje natural. Por su parte, las GAN (redes generativas antagónicas) se utilizan para generar imágenes realistas y de alta resolución.

1.4.3.1 Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales trabajan principalmente con imágenes, tomándolas como input, las cuales se le asignan pesos a ciertos elementos de la imagen, para poder diferenciar unos de otros. Este es uno de los principales tipos de redes neuronales que han contribuido considerablemente al perfeccionamiento de la visión por computadora (Silva & Freire, 2019).

Las redes convolucionales constan de capas ocultas donde van detectando patrones en las imágenes, así como líneas, curvas, hasta poder reconocer formas complejas como un rostro, silueta, etc. (Fig. 1.4). Las tareas generales de este tipo de redes son:

- Detección o categorización de objetos
- Clasificación de escenas
- Clasificación de imágenes en general

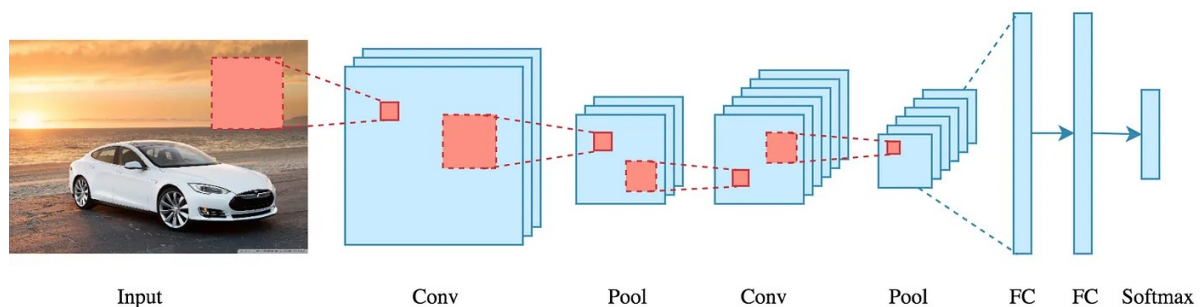


Figura 1.4: Arquitectura general de una Red Neuronal Convolutional para clasificación. Se muestran capas que secuencialmente van ejecutando diferentes funciones. Tomado de Silva & Freire (2019).

La red se alimenta de los píxeles de la imagen. Si la imagen es de tan solo 28x28 píxeles de tamaño, tendremos un total de 784 neuronas en la capa de entrada, suponiendo que se trate de una imagen en escala de grises. Si la imagen es a color, requerirá 3 canales (rojo, verde y azul), lo que nos lleva a utilizar 2352 neuronas en la capa de entrada al calcular $28 \times 28 \times 3$.

Kernel

En las redes convolucionales, el *kernel* se utiliza como un filtro para extraer características significativas o patrones de una imagen. Por ejemplo, consideremos la imagen de la figura 1.5. Aplicando el filtro o *kernel* se mostrará como se muestra en la figura 1.6.



Figura 1.5: Imagen sin aplicar el kernel. Tomado de Silva & Freire (2019).



Figura 1.6: Imagen con el kernel aplicado. Tomado de Silva & Freire (2019).

El *kernel* desempeña un papel crucial en la detección de características clave, como bordes, enfoque, y desenfoque, entre otros. Este proceso se logra mediante la convolución entre la imagen y el *kernel* (Silva & Freire, 2019).

Convolución

Uno de los procesos característicos de estas redes es la convolución, que implica tomar un conjunto de píxeles de la imagen de entrada y realizar productos escalares con un *kernel*. El *kernel* se desplaza por todas las neuronas de entrada, generando así una nueva matriz que se convierte en una de las capas ocultas (Fig. 1.7). Si la imagen es a color, se utilizan tres *kernels* del mismo tamaño que se suman para obtener una imagen de salida.

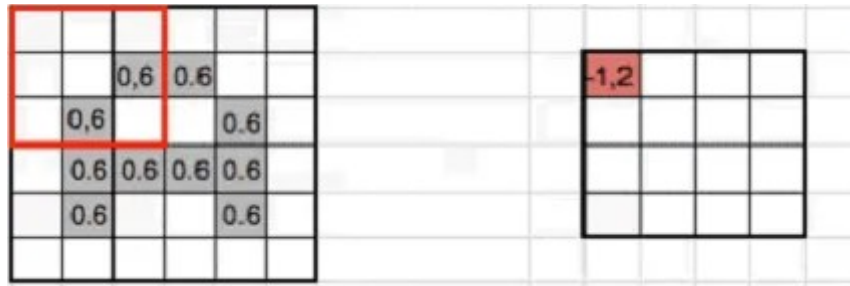


Figura 1.7: Ejemplo de convolución de una imagen. Tomado de Silva & Freire (2019)

Convolución con imágenes en escala de grises

En este proceso, contamos con una imagen y un *kernel*, y la intención es que el filtro o *kernel* se desplace por toda la imagen (píxel a píxel). Por lo general, el *kernel* es de menor tamaño que la imagen. La convolución implica multiplicar el *kernel* por la porción de la imagen seleccionada, siguiendo un procedimiento como se muestra en la figura 1.8. Luego, el *kernel* se desplaza, lo que hace que este proceso sea iterativo.

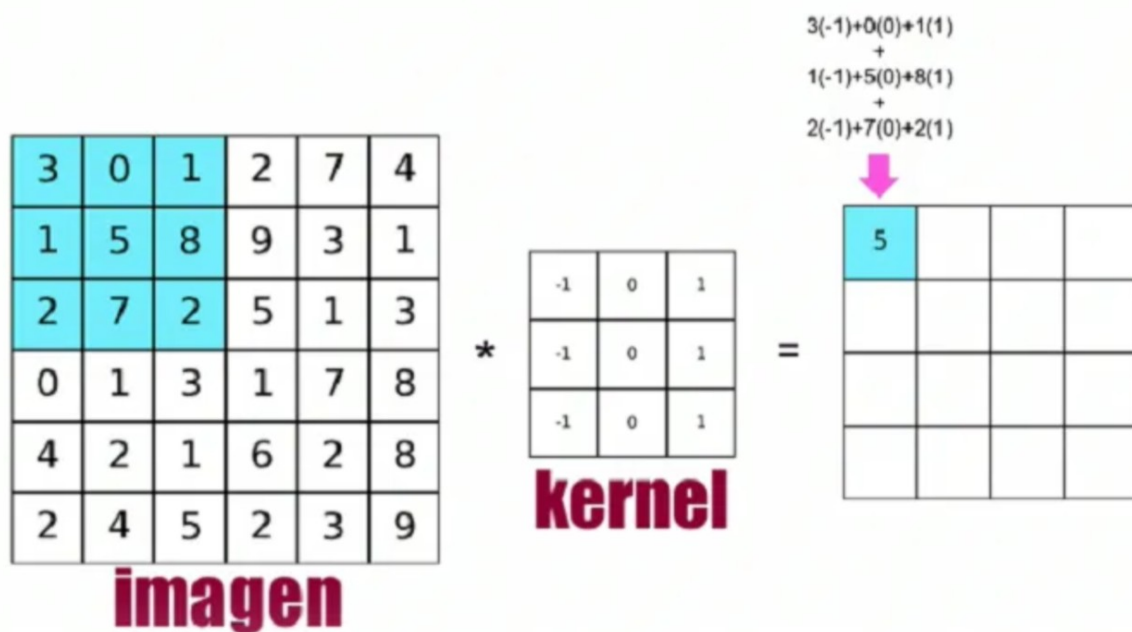


Figura 1.8: Cálculo de la primera celda de la matriz de salida en el proceso de convolución de una imagen. Tomado de Silva & Freire (2019).

Se procede a realizar un desplazamiento y se vuelve a multiplicar (Fig. 1.9).

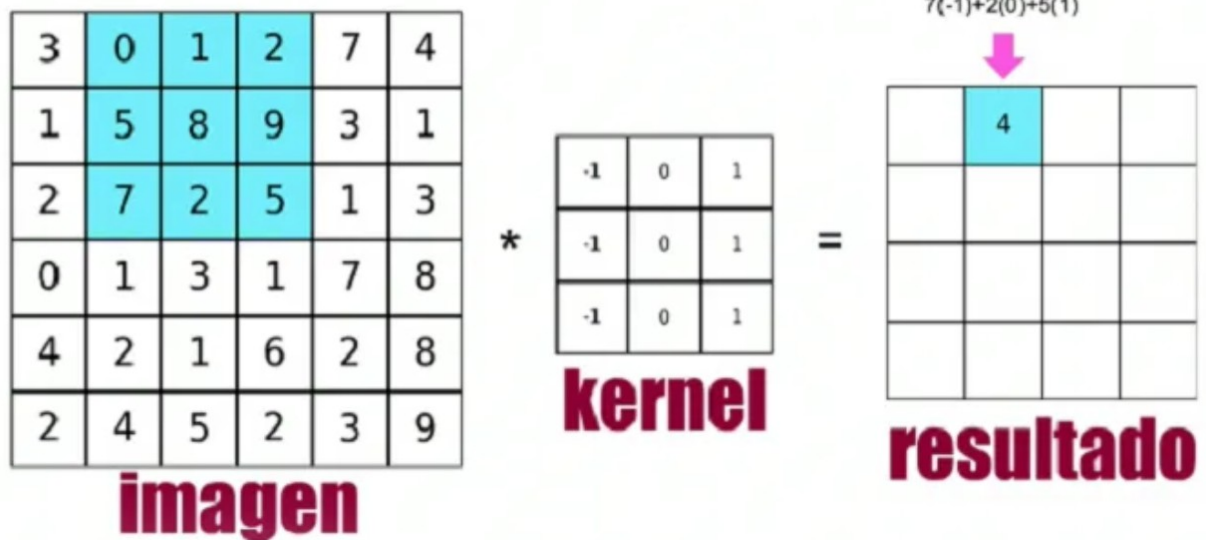


Figura 1.9: Cálculo de la segunda celda de la matriz de salida en el proceso de convolución de una imagen.. Tomado de Silva & Freire (2019).

Este proceso se repite sucesivamente hasta recorrer toda la imagen de entrada (Fig. 1.10).

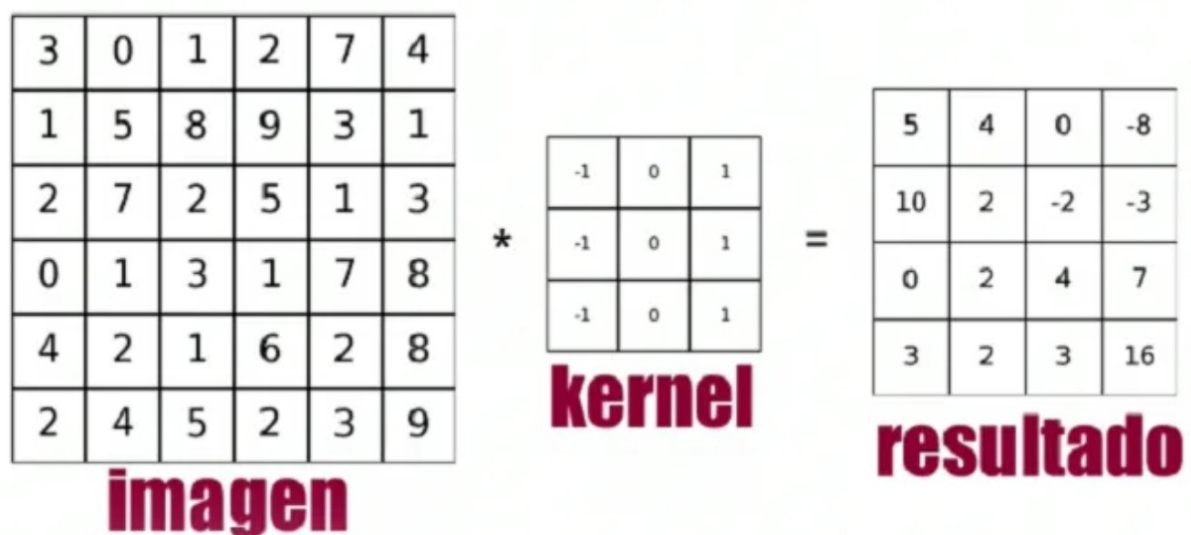


Figura 1.10: Resultado final de la matriz de salida del proceso de convolución de una imagen. Tomado de Silva & Freire (2019).

Convolución con imágenes RGB

Cuando se trabajan con imágenes RGB, las mismas son representadas mediante 3 canales: R (rojo), G (verde) y B (azul), que al combinarse generan todas las mezclas de colores. Por lo tanto, ahora el *kernel* también consta de 3 canales (Fig. 1.11).

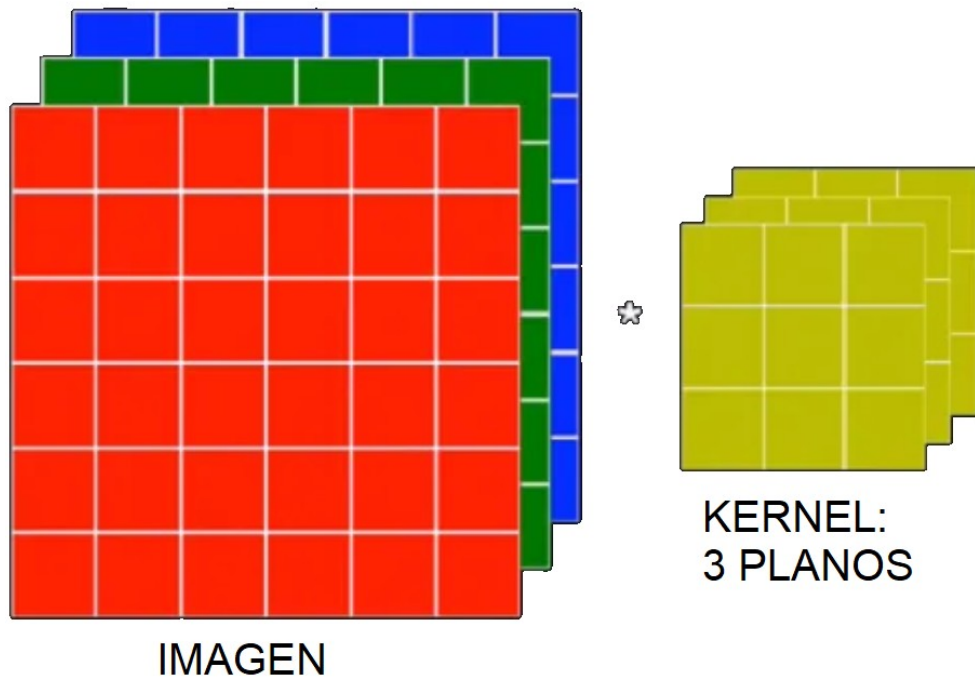


Figura 1.11: Elementos contemplados en el Proceso de convolución de una imagen RGB. Tomado de Silva & Freire (2019).

La convolución se asemeja a la de una imagen en escala de grises, pero en este caso, el *kernel* de 3 capas se aplica a un píxel correspondiente. A pesar de que la imagen de entrada tiene 3 canales, la imagen resultante tendrá sólo 1 canal (Fig. 1.12).

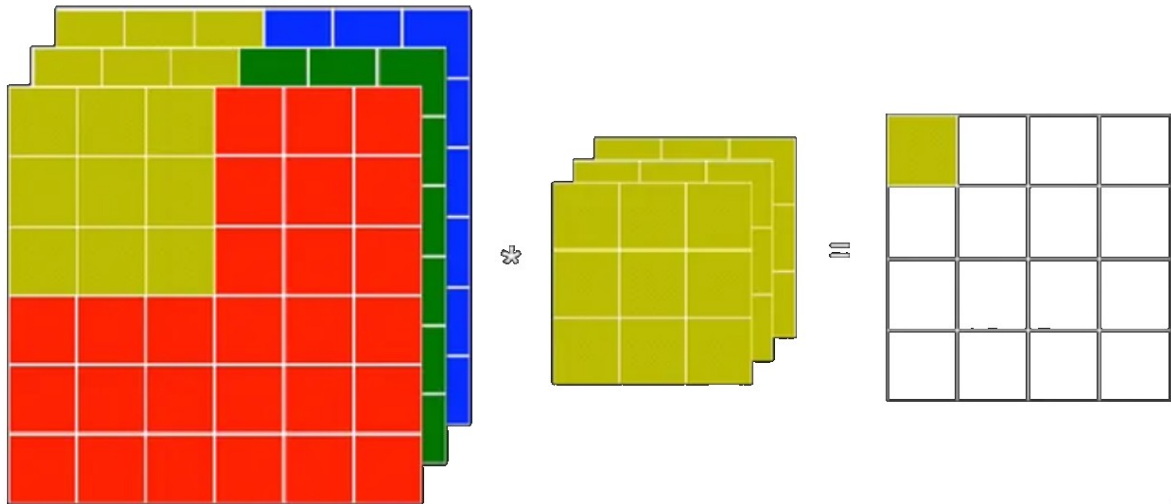


Figura 1.12: Cálculo de la primera celda de la matriz de salida en el proceso de convolución de una imagen RGB. Tomado de Silva & Freire (2019).

Este proceso se realiza iterativamente hasta cubrir toda la imagen de entrada.

Padding

Es una operación común en las redes convolucionales. El padding implica agregar píxeles de valor cero alrededor de la imagen original (Fig. 1.13) y se utiliza por dos razones principales. La primera razón es para que la imagen resultante de la convolución sea del mismo tamaño que la imagen original. La otra razón es cuando la información relevante se encuentra en las esquinas de la imagen. El padding se aplica de modo que el filtro convolucional se centre mejor en las regiones importantes de la imagen, en lugar de enfocarse principalmente en las esquinas. (Silva & Freire, 2019)

0	0	0	0	0	0	0	0
0	3	4	6	5	1	3	0
0	5	3	2	4	3	2	0
0	5	4	3	3	2	6	0
0	1	1	2	5	3	4	0
0	2	3	3	4	1	2	0
0	3	3	2	4	2	4	0
0	0	0	0	0	0	0	0

Figura 1.13: Ejemplo de una imagen después de aplicar una operación de Padding. Tomado de Silva & Freire (2019).

1.4.4 Transferencia de aprendizaje

La transferencia de aprendizaje es una técnica en el campo del *machine learning* que permite aprovechar conocimientos adquiridos en un contexto (dominio fuente) para mejorar el aprendizaje en otro contexto diferente (dominio objetivo), incluso cuando los datos o las distribuciones de probabilidad no son idénticos. A diferencia del aprendizaje tradicional, que asume que los datos de entrenamiento y operación siguen la misma distribución, la transferencia de aprendizaje aborda escenarios reales donde existen limitaciones de datos o cambios en las distribuciones a lo largo del tiempo. Esto se logra mediante la transferencia de conocimiento desde espacios muestrales y distribuciones fuente hacia un espacio objetivo, facilitando la adaptación y generalización de modelos en entornos dinámicos. (Cody & Beling, 2021)

Esta técnica mejora las capacidades de un algoritmo de *machine learning* al exponerlo previamente a un dataset distinto del objetivo, lo que permite utilizar redes preentrenadas. Esto elimina la necesidad de contar con grandes volúmenes de datos o invertir tiempo en entrenar una red desde cero. Cuando el conjunto de datos original utilizado para entrenar la red es lo suficientemente amplio y representativo, los atributos aprendidos permiten que el modelo funcione como una representación general del dominio, lo que lo hace aplicable en contextos diferentes al de su entrenamiento inicial.

Específicamente, en el ámbito de *computer vision*, muchos modelos pre-entrenados, como por ejemplo *MobileNet* (entrenado en el conjunto de datos *ImageNet*) están disponibles públicamente para su descarga y pueden ser utilizados para desarrollar modelos de visión altamente efectivos con un esfuerzo de recopilación de datos considerablemente reducido. (Silva & Freire, 2019)

1.4.5 MobileNet

MobileNet es una arquitectura de red neuronal diseñada específicamente para aplicaciones de visión por computadora en dispositivos móviles y embebidos. Fue desarrollada por Google y se destaca por su eficiencia y capacidad de ejecutarse en dispositivos con recursos limitados, como teléfonos inteligentes y sistemas embebidos.

La característica principal de MobileNet es su capacidad para realizar inferencias (tomar decisiones basadas en datos de entrada) de manera rápida y eficiente, lo que es esencial en dispositivos móviles donde los recursos computacionales, como la potencia de cálculo y la memoria, suelen ser limitados. Esto se logra mediante el uso de operaciones de convolución profunda, junto con técnicas como la factorización, que reducen significativamente la carga computacional. (Howard et al., 2017)

Por otro lado, MobileNet ofrece un modelo preentrenado con ImageNet, permitiendo implementar estrategias de transferencia de conocimiento para adaptar la red a nuevas tareas de visión artificial con menos datos y tiempo. Estos pesos preentrenados son especialmente útiles en entornos con limitaciones computacionales o datasets pequeños, ya que el modelo ya contiene características visuales básicas que pueden ajustarse para aplicaciones específicas sin entrenar desde cero, optimizando recursos y acelerando el desarrollo para dispositivos móviles.

MobileNet ha demostrado un rendimiento sólido en tareas de *computer vision*, como la clasificación de imágenes y la detección de objetos, y se ha convertido en una elección popular para aplicaciones de inteligencia artificial en dispositivos móviles. La versatilidad y eficiencia de MobileNet lo hacen ideal para aplicaciones de visión en tiempo real, como reconocimiento de objetos en imágenes o videos, incluso en dispositivos con recursos limitados.

1.5 Edge Computer Vision

La visión por computadora o Computer Vision es un campo de la informática que se enfoca en permitir que los equipos identifiquen y comprendan objetos y personas en imágenes y vídeos. El objetivo principal de esta disciplina es permitir a las computadoras “ver” y comprender el mundo visual de la misma manera que lo hacen los seres humanos.

Dentro de este campo existe una subcategoría denominada *Edge Computer Vision* el cual es un nuevo paradigma de *Computer Vision* que traslada el procesamiento de datos visuales desde la nube hasta el “borde” de la red (Fig. 1.14). Esto permite el procesamiento en tiempo real de los datos, lo cual es esencial para diferentes aplicaciones como la conducción autónoma, la realidad aumentada, la automatización industrial y sistemas de seguridad. (McEnroe et al., 2022; Earney, 2023)

Existen varias ventajas que trae *Edge Computer Vision*. En primer lugar, puede reducir significativamente la latencia, ya que los datos no necesitan ser enviados a la nube y luego devueltos, como sí sucede en *Cloud Computing*. En segundo lugar, puede mejorar la privacidad, ya que los datos no se transmiten por la red y tampoco se almacenan en la nube. En tercer lugar, puede mejorar la confiabilidad, ya que los dispositivos suelen ser más resistentes a las interrupciones o limitaciones en la red. (McEnroe et al., 2022; Earney, 2023)

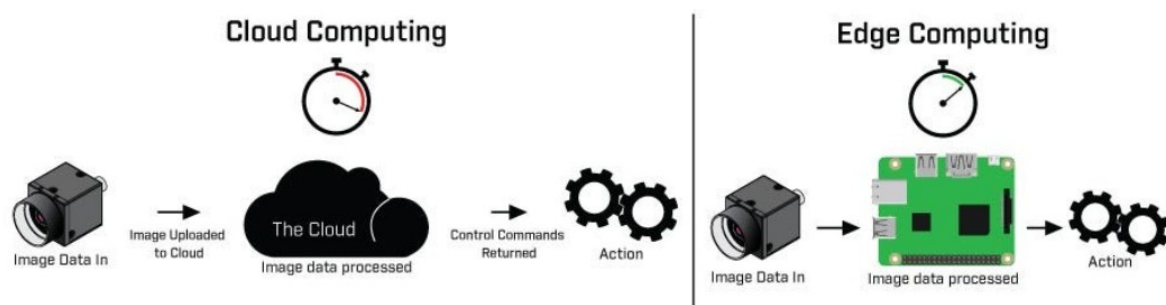


Figura 1.14: *Edge Computing vs Cloud Computing*. Modificado de Raj (2018).

También existen algunos desafíos sobre *Edge Computer Vision*. En primer lugar, requiere dispositivos más potentes. En segundo lugar, requiere algoritmos más eficientes. En tercer lugar, requiere una mayor coordinación entre los dispositivos de “borde” (que capturan las imágenes) y los servicios en la nube (McEnroe et al., 2022).

A pesar de los desafíos, es una tecnología prometedora con el potencial de revolucionar muchas industrias.

En el campo de *Edge Computer Vision*, se han desarrollado diversos proyectos y frameworks, destacando el impacto significativo de Tensorflow Lite (Khandelwal, 2020). Este framework ha generado aportes importantes en el ámbito, consolidándose como una de las más destacadas y cada vez más utilizadas.

1.6 Microcontroladores

Un microcontrolador es un circuito integrado compacto diseñado para ejecutar un conjunto de instrucciones almacenadas en su memoria. Un microcontrolador típico está equipado con memoria, temporizadores, pines de E/S y otros periféricos integrados en el chip. (Fig. 1.15). (Gridling & Weiss, 2007)

Estos dispositivos se encuentran en una amplia gama de aplicaciones, desde vehículos y robots hasta electrodomésticos y dispositivos médicos. Funcionan como pequeñas computadoras diseñadas para controlar funciones específicas de un sistema más grande, sin necesidad de un sistema operativo complejo. (Lutkevich, 2019)

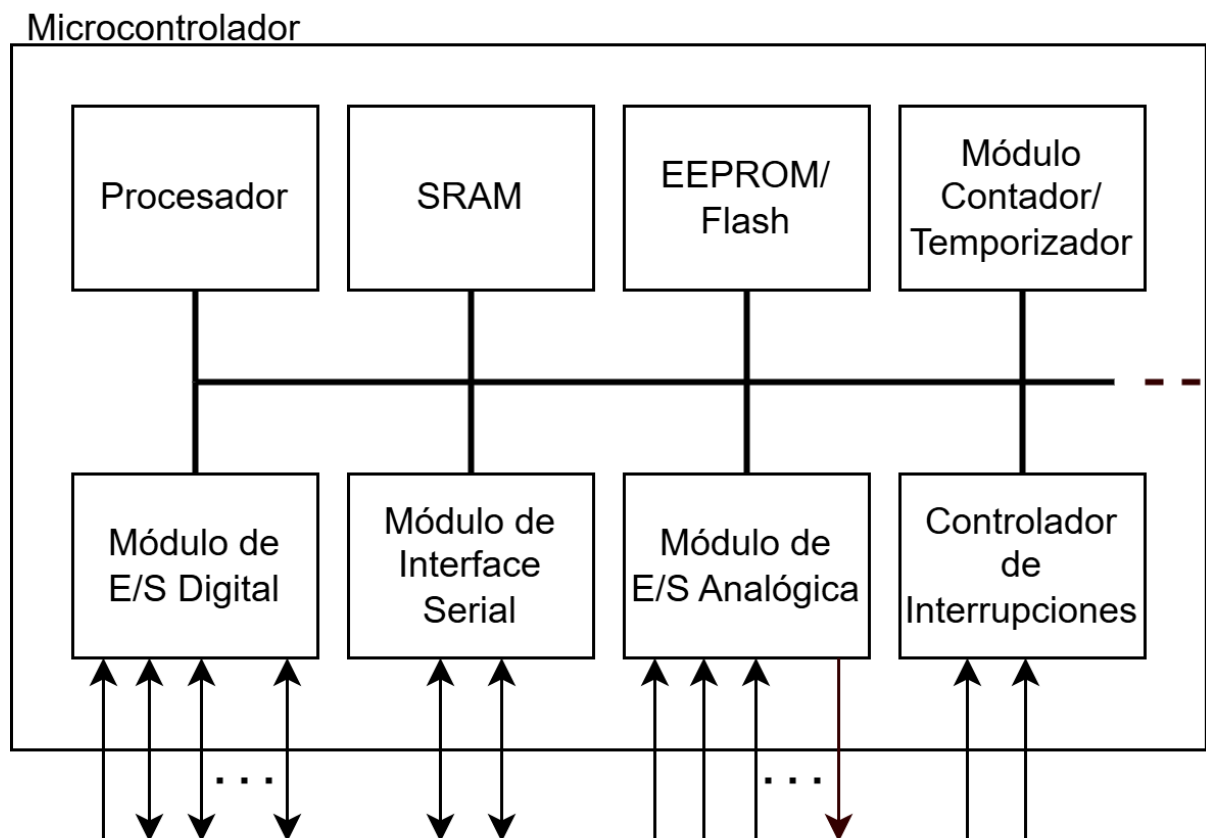


Figura 1.15: Esquema general de un Microcontrolador. Se muestran los elementos más representativos de los elementos internos de un Microcontrolador. Modificado de Gridling & Weiss (2007).

Los microcontroladores pueden tener palabras de cuatro bits y operar a velocidades

de reloj tan bajas como 4 kHz, con un consumo de energía muy reducido, en la escala de milivatios o microwatts. Pueden permanecer en un estado de espera para eventos, como la detección de una interrupción o la pulsación de un botón, lo que minimiza el consumo de energía cuando no están activos, a menudo alcanzando valores de nanowatts. Esto los hace adecuados para aplicaciones con baterías de larga duración.

Cuando se fabrica un microcontrolador, su memoria ROM generalmente está vacía. Para que pueda controlar un proceso, es necesario generar un programa y grabarlo en la EEPROM u otro tipo de memoria del microcontrolador. Este programa suele estar escrito en lenguaje ensamblador u otros lenguajes específicos para microcontroladores y se codifica en sistema hexadecimal. Cuando se alimenta con el voltaje adecuado y se asocia con componentes analógicos y discretos, el microcontrolador puede ejecutar el programa y realizar sus funciones.

(Microcontrolador, sin fecha)

1.6.1 Elementos internos de un microcontrolador

A continuación se enumeran los componentes típicos de un microcontrolador (Gridling & Weiss, 2007):

- **Procesador:** Es el núcleo de procesamiento del microcontrolador. Está compuesto por la unidad aritmético-lógica (ALU), la unidad de control y diversos registros como el contador de programa, puntero de pila, acumulador y banco de registros, entre otros.
- **Memoria:** A veces se divide en memoria de programa y memoria de datos. En modelos más complejos, se puede incluir un controlador de Acceso Directo a Memoria (DMA) que facilita el intercambio de datos entre periféricos y memoria sin intervención directa del procesador.
- **Controlador de Interrupciones:** Permite modificar la ejecución normal del programa cuando ocurren eventos internos o externos relevantes. Su uso, junto con modos de bajo consumo, contribuye a optimizar el consumo energético del sistema.
- **Temporizador/Contador:** La mayoría de los microcontroladores incorporan al menos uno, y comúnmente se encuentran varios. Estos módulos permiten medir tiempos, registrar sucesos o contar pulsos. Muchos también integran salidas PWM (modulación por ancho de pulso), útiles para tareas como el control de motores o funciones como el frenado seguro (por ejemplo, en sistemas ABS). Además, una señal PWM con un filtro adecuado puede funcionar como un DAC (convertidor digital a analógico) simple.
- **Entrada/Salida Digital:** Los puertos de E/S digital en paralelo son una de las características principales de los microcontroladores. Dependiendo del modelo y la familia, el número de pines de E/S puede variar desde unos pocos hasta más de 90.

- **Entrada/Salida Analógica:** Salvo en los modelos más simples, los microcontroladores suelen incluir convertidores analógico/digital (ADC) con distintos números de canales (entre 2 y 16) y resoluciones (entre 8 y 12 bits). También pueden disponer de comparadores analógicos y, en algunos casos, de convertidores digital/analógico (DAC).
- **Interfaces:** Casi todos los microcontroladores cuentan con al menos una interfaz serie, empleada tanto para la carga del programa como para la comunicación con otros dispositivos. Muchas veces también ofrecen interfaces como SPI o SCI para interactuar con periféricos externos. En microcontroladores pueden encontrarse interfaces como USB, Ethernet o PCI, así como controladores de bus integrados, especialmente útiles en entornos industriales.
- **Temporizador de vigilancia (Watchdog Timer):** Este componente resulta esencial en aplicaciones críticas para la seguridad. Su función es reiniciar el microcontrolador automáticamente si detecta que el sistema ha dejado de funcionar correctamente, ayudando a prevenir bloqueos por errores de software o fallos de hardware.
- **Unidad de depuración:** Algunos microcontroladores integran hardware específico para facilitar la depuración remota directamente desde una computadora. Esta característica permite realizar pruebas sin necesidad de cargar herramientas de depuración adicionales, evitando que errores en el código de aplicación interfieran con el proceso de depuración.

1.6.2 Placas de desarrollo de microcontroladores

Las placas de desarrollo de microcontroladores, conocidas también como kits de desarrollo, facilitan a los diseñadores de sistemas la creación y prueba ágil de proyectos. Estas placas incluyen el microcontrolador deseado junto con, generalmente, memoria adicional, puertos de entrada y salida, LEDs, interruptores, programador y otras interfaces periféricas como UART, I2C, SPI, entre otras. Debido a que el hardware de las placas de desarrollo ha sido pre-testeado por los fabricantes, el diseñador puede concentrarse en el software de la aplicación, sabiendo que el hardware está en funcionamiento. (Ibrahim, 2020)

1.6.3 Placa de desarrollo ESP32-CAM

ESP32-CAM es una placa de desarrollo de bajo costo y de reducido tamaño, basada en un microcontrolador ESP32. Es una solución ideal para aplicaciones de IoT, prototipos y proyectos de bricolaje.

La placa integra una cámara, WiFi, Bluetooth tradicional y BLE de bajo consumo, con 2 CPUs de 32 bits.

ESP32-CAM se puede utilizar ampliamente en diversas aplicaciones de IoT. Es adecuado para dispositivos inteligentes para el hogar, control inalámbrico industrial, monitoreo inalámbrico, identificación inalámbrica QR, señales de sistemas de posicionamiento inalámbrico y otras aplicaciones de IoT. En la figura 1.16 se describen los diferentes puertos que conforman el microcontrolador. (Ai-Thinker Technology, sin fecha)

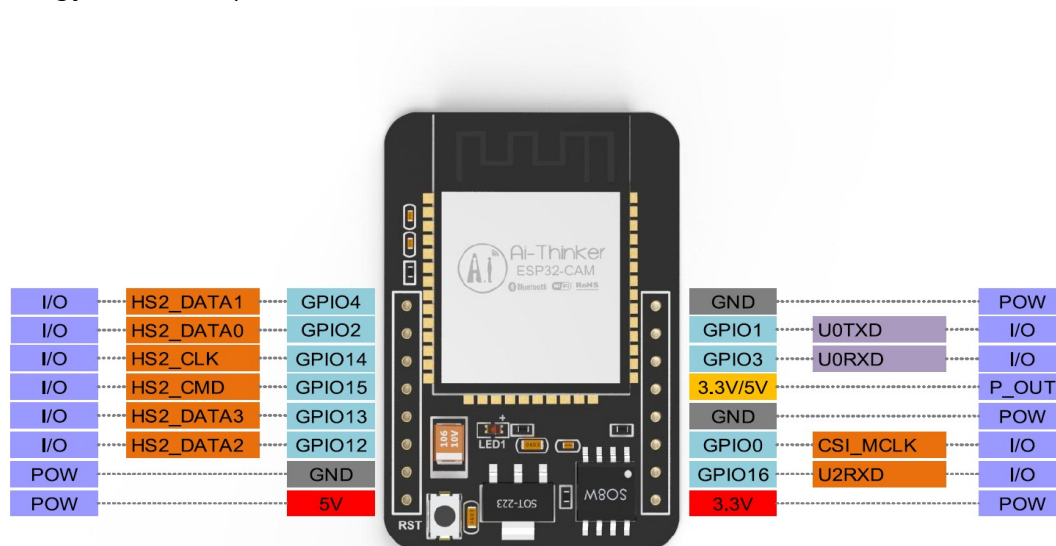


Figura 1.16: Puertos de la placa de desarrollo ESP32-CAM. Tomado del sitio Ai-Thinker Technology.

Esta placa de desarrollo es comercializada por distintos fabricantes, entre los que se destacan “Espressif Systems” (www.espressif.com) y “Ai-Thinker Technology” (www.ai-thinker.com).

Especificaciones generales:

- CPU de 32 bits de doble núcleo
- Velocidad de reloj de hasta 240 MHz
- Potencia de cálculo de hasta 600 DMIPS
- 520 KB de SRAM
- 8 MB PSRAM
- 802.11b/g/n Wi-Fi
- Bluetooth 4.2 con BLE
- Interfaces UART, SPI, I2C y PWM
- Admite carga de imágenes a través de Wi-Fi
- Múltiples modos de suspensión
- Posibilidad de actualizaciones de firmware por vía aérea (FOTA)
- 9 puertos GPIO
- LED Flash incorporado
- Cámara:

- Sensor de 2 megapíxeles
- Tamaño máximo soportado UXGA 1622×1200
- Los formatos de salida incluyen YUV422, YUV420, RGB565, RGB555 y datos comprimidos de 8 bits
- Tasa de transferencia de imágenes de 15 a 60 fps

1.7 Cuantización de modelos

La cuantización es una técnica para reducir los costos computacionales y de memoria de la ejecución de modelos de redes neuronales artificiales al representar los pesos y activaciones con tipos de datos de baja precisión, como enteros de 8 bits (int8) en lugar de los habituales 32 bits en punto flotante (float32).

Reducir el número de bits significa que el modelo resultante requiere menos almacenamiento en memoria, consume menos energía, y las operaciones como la multiplicación de matrices se pueden realizar mucho más rápido con aritmética de enteros. También permite ejecutar modelos en dispositivos integrados que a veces solo admiten tipos de datos enteros. (Opendragon, sin fecha)

En la literatura, distinguimos dos principales familias de técnicas de cuantización de pesos (Labonne, 2023):

- Cuantización Post-Entrenamiento (CPE): es una técnica en la que los pesos de un modelo ya entrenado se convierten a una menor precisión sin necesidad de un nuevo entrenamiento. Aunque fácil de implementar, la CPE se asocia con posibles degradaciones de rendimiento.
- Entrenamiento Consciente de Cuantización (ECC): incorpora el proceso de conversión de pesos durante la etapa de pre-entrenamiento o ajuste fino, lo que resulta en un mejor rendimiento del modelo. Sin embargo, el ECC es computacionalmente costoso y requiere datos de entrenamiento representativos.

1.7.1 Antecedentes sobre la Representación de Punto Flotante

La elección del tipo de dato dicta la cantidad de recursos computacionales necesarios, lo que afecta la velocidad y eficiencia del modelo. En aplicaciones de *deep learning*, lograr un equilibrio entre la precisión y el rendimiento computacional se convierte en un ejercicio vital, ya que una mayor precisión a menudo implica mayores demandas computacionales.

Entre varios tipos de datos, los números de punto flotante son predominantemente empleados en *deep learning* debido a su capacidad para representar una amplia gama de valores con alta precisión. Típicamente, un número de punto flotante utiliza

n bits para almacenar un valor numérico. Estos n bits se dividen en tres componentes distintos:

- **Signo:** El bit de signo indica la naturaleza positiva o negativa del número. Utiliza un bit donde 0 indica un número positivo y 1 señala un número negativo.
- **Exponente:** El exponente es un segmento de bits que representa la potencia a la cual se eleva la base (generalmente 2 en representación binaria). El exponente puede ser positivo o negativo, lo que permite al número representar valores muy grandes o muy pequeños.
- **Mantisa:** Los bits restantes se utilizan para almacenar la mantisa, también conocida como significando. Esto representa las cifras significativas del número. La precisión del número depende en gran medida de la longitud de la mantisa.

Esta estructura permite que los números de punto flotante abarquen una amplia gama de valores con diferentes niveles de precisión. La ecuación 1.1 es la utilizada para esta representación.

$$(-1)^{\text{signo}} \times \text{base}^{\text{exponente}} \times \text{mantisa}$$

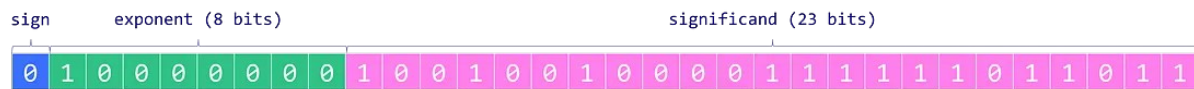
Ecuación 1.1: Representación de números de punto flotante.

Para tener una mayor comprensión de este punto, seguidamente profundizaremos en algunos de los tipos de datos más comúnmente utilizados en *deep learning*: float32 (FP32), float16 (FP16) y bfloat16 (BF16):

- FP32 utiliza 32 bits para representar un número: un bit para el signo, ocho para el exponente y los 23 restantes para la mantisa. Si bien proporciona un alto grado de precisión, la desventaja de FP32 es su alto consumo computacional y de memoria.
- FP16 utiliza 16 bits para almacenar un número: un bit para el signo, cinco para el exponente y diez para la mantisa. Aunque esto lo hace más eficiente en memoria y acelera los cálculos, el rango y la precisión reducidos pueden introducir inestabilidad numérica, lo que potencialmente afecta la precisión del modelo.
- BF16 es también un formato de 16 bits, pero con un bit para el signo, ocho para el exponente y siete para la mantisa. BF16 amplía el rango representable en comparación con FP16, disminuyendo así los riesgos de desbordamiento y subdesbordamiento. A pesar de una reducción en la precisión debido a menos bits en la mantisa, BF16 generalmente no afecta significativamente el rendimiento del modelo y es un compromiso útil para tareas de aprendizaje profundo.

La figura 1.17 muestra la diferencia de tamaño y composición de cada uno de los tipos de datos descritos anteriormente.

32-bit float (FP32)



$$(-1)^0 \times 2^{128-127} \times 1.5707964 = 3.1415927$$

16-bit float (FP16)



$$(-1)^0 \times 2^{128-127} \times 1.571 = 3.141$$

bfloat16 (BF16)



$$(-1)^0 \times 2^{128-127} \times 1.5703125 = 3.140625$$

Figura 1.17: Tipos de datos más comúnmente utilizados en deep learning. Tomado de Labonne (2023).

En la jerga del aprendizaje automático, a menudo se denomina a FP32 como "precisión completa" (4 bytes), mientras que BF16 y FP16 son "media precisión" (2 bytes). Pero es posible ir un paso más allá y almacenar pesos utilizando solo un byte. El tipo de dato INT8 proporciona una representación de 8 bits capaz de almacenar $2^8 = 256$ valores diferentes. En la siguiente sección, se explora cómo convertir pesos de FP32 a un formato INT8. (Labonne, 2023)

1.7.2 Cuantización de 8 bits

En esta sección, se muestran dos de las técnicas de cuantización más conocidas, valor absoluto máximo (absmax) la cual es una cuantización simétrica y cuantización de punto cero la cual es una cuantización asimétrica. En ambos casos, el objetivo es mapear un tensor (matriz multidimensional) FP32 X (pesos originales) a un tensor INT8 X_quant (pesos cuantizados). (Fig. 1.18)

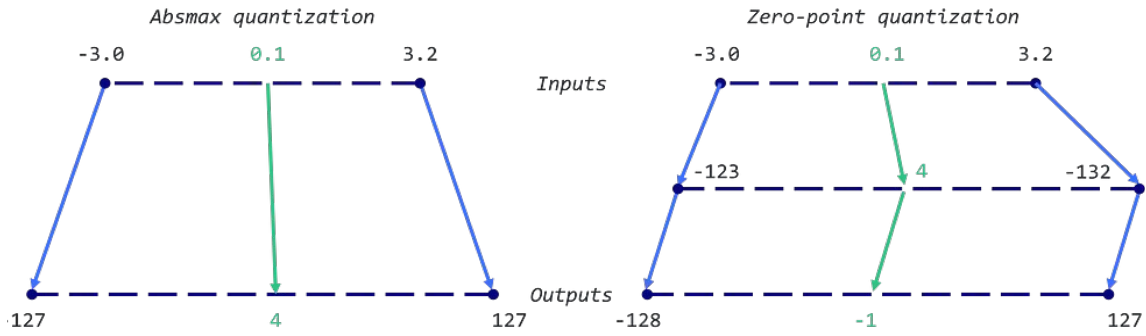


Figura 1.18: Cuantización Absmax vs cuantización punto cero. Tomado de Labonne (2023).

Con la cuantización absmax, el número original se divide por el valor absoluto máximo del tensor y se multiplica por un factor de escala (127) para mapear las entradas en el rango $[-127, 127]$ (ecuación 1.2). Para recuperar los valores FP16 originales, el número INT8 se multiplica por el valor absoluto máximo del tensor y se divide por el factor de cuantización (ecuación 1.3), reconociendo cierta pérdida de precisión debido al redondeo.

$$X_{\text{quant}} = \text{round} \left(\frac{127}{\max |X|} \cdot X \right)$$

Ecuación 1.2: Fórmula de cuantización.

$$X_{\text{dequant}} = \frac{\max |X|}{127} \cdot X_{\text{quant}}$$

Ecuación 1.3: Fórmula de des-cuantización.

Por ejemplo, supongamos que tenemos un valor máximo absoluto de 3.2. Un peso de 0.1 se cuantizaría como $\text{round}(0.1 \times 127/3.2) = 4$. Si deseamos des-cuantizarlo, obtendríamos $4 \times 3.2/127 = 0.1008$, lo que implica un error de 0.0008. (Labonne, 2023)

Por otro lado, con la cuantización de punto cero, podemos considerar distribuciones de entrada asimétricas, lo cual es útil cuando se tiene en cuenta la salida de una función ReLU (solo valores positivos), por ejemplo. Primero, escalamos los valores de entrada mediante el rango total de valores (255) dividido por la diferencia entre los valores máximos y mínimos. Luego, esta distribución se desplaza por el punto cero para mapearla al rango $[-128, 127]$ (consta de un valor adicional en comparación con absmax). En primer lugar, se calcula el factor de escala (ecuación 1.4) y el valor del punto cero (ecuación 1.5).

$$\text{scale} = \frac{255}{\max(\mathbf{X}) - \min(\mathbf{X})}$$

Ecuación 1.4: Cálculo de factor de escala.

$$\text{zeropoint} = -\text{round}(\text{scale} \cdot \min(\mathbf{X})) - 128$$

Ecuación 1.5: Cálculo de valor de punto cero.

Posteriormente, se usan las variables anteriores para cuantizar o des-cuantizar los pesos del modelo, de acuerdo con las ecuaciones 1.6 y 1.7 respectivamente.

$$\mathbf{X}_{\text{quant}} = \text{round}(\text{scale} \cdot \mathbf{X} + \text{zeropoint})$$

Ecuación 1.6: Fórmula de Cuantización.

$$\mathbf{X}_{\text{dequant}} = \frac{\mathbf{X}_{\text{quant}} - \text{zeropoint}}{\text{scale}}$$

Ecuación 1.7: Fórmula de des-cuantización.

Por ejemplo, si tenemos un valor máximo de 3.2 y un valor mínimo de -3.0. Podemos calcular que la escala es $255 / (3.2 + 3.0) = 41.13$ y el punto cero sería $-\text{round}(41.13 \times -3.0) - 128 = 123 - 128 = -5$, por lo que nuestro peso anterior de 0.1 sería cuantificado como redondeando $(41.13 \times 0.1 - 5) = -1$. Esto es muy diferente del valor anterior obtenido utilizando absmax (4 frente a -1). (Labonne, 2023)

Es importante recalcar que cuando aplicamos el proceso de des-cuantización, generamos un error por la pérdida de precisión que genera su previa transformación a INT8. En la figura 1.19 puede visualizarse más claro en un ejemplo.

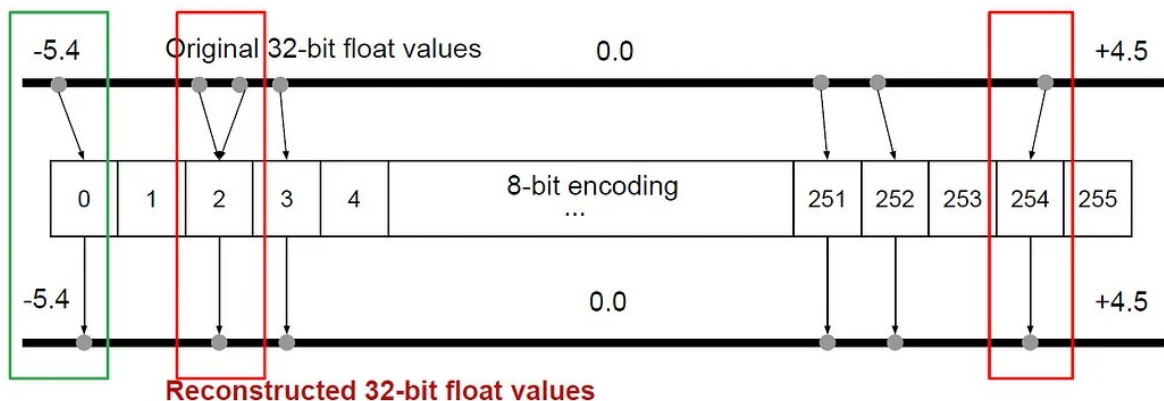


Figura 1.19: Pérdida de precisión en cuantización. Tomado de Braun (2021).

1.8 Tensorflow Lite

TensorFlow Lite (TFLite) es un framework de *Machine Learning* de código abierto, liviano y eficiente para dispositivos móviles y de borde (*Edge*). Es un subconjunto del framework TensorFlow y está diseñado para ser utilizado en dispositivos con recursos limitados, como teléfonos inteligentes, tabletas y microcontroladores. (Khandelwal, 2020)

TFLite se puede utilizar para ejecutar una variedad de modelos de *Machine Learning*, incluyendo clasificación de imágenes, detección de objetos, procesamiento de lenguaje natural y reconocimiento de voz. (Khandelwal, 2020)

Utilizar TFLite para la creación de modelos proporciona los siguientes beneficios principales:

- **Tamaño reducido:** los modelos de TFLite suelen ser mucho más pequeños que sus contrapartes de TensorFlow, lo que los hace ideales para implementar en dispositivos con recursos limitados. (Tensorflow Lite, sin fecha)
- **Aumento de la velocidad de ejecución del modelo:** los modelos de TFLite están optimizados para tener una mayor velocidad, por lo que pueden ejecutarse en dispositivos con capacidad de procesamiento limitada. (Tensorflow Lite, sin fecha)
- **Compatibilidad multiplataforma:** TFLite es compatible con una amplia gama de plataformas de hardware, lo que lo convierte en una opción versátil para su uso en diferentes contextos. (Tensorflow Lite, sin fecha)

Para la generación de un modelo TFLite se pueden seguir las siguientes alternativas:

- **Usar un modelo TFLite existente:** Se puede consultar los Ejemplos de TensorFlow Lite para seleccionar un modelo existente. (Tensorflow Lite, sin fecha)
- **Crear un modelo TFLite:** Utilizar el Generador de Modelos TFLite para crear un modelo con un conjunto de datos personalizado. (Tensorflow Lite, sin fecha)
- **Convertir un modelo TensorFlow en un modelo TFLite:** Emplear el Conversor TensorFlow Lite para transformar un modelo TensorFlow en un modelo TFLite. Durante la conversión, es posible aplicar optimizaciones como la cuantización para reducir el tamaño y la latencia del modelo con una pérdida mínima o nula de precisión. Esta estrategia fue empleada en este trabajo. (Tensorflow Lite, sin fecha)

1.9 Lora y LoraWan

En los últimos tiempos, una de las tecnologías de comunicación que ha experimentado un notable crecimiento es LoRa (Long Range), con una amplia aplicación en el Internet de las Cosas.

LoRa es un protocolo inalámbrico de largo alcance y baja velocidad diseñado para permitir la conexión de sensores y otros dispositivos con un consumo mínimo de energía, lo que puede llegar a prolongar la duración de las baterías por años. (LoRa Alliance, 2015, Ubierna, 2020 ; *LoRa*, sin fecha)

Para gestionar de manera eficiente la información proveniente de estos dispositivos, se emplea una arquitectura de red conocida como LoRaWAN. Esta arquitectura incluye gateways, que son transmisores de radio que se encargan de la comunicación directa con los sensores, así como varios servidores dedicados a funciones como la gestión de la red, la aplicación y la seguridad, entre otros. (LoRa Alliance, 2015, Ubierna, 2020 ; *LoRa*, sin fecha)

Las principales características de LoRa son su largo alcance, superando obstáculos y alcanzando distancias de hasta 3 km en áreas urbanas y de hasta 20 km en áreas abiertas; baja transmisión de datos, priorizando la transmisión de pequeñas cantidades de información; bajo consumo de energía; y conexión masiva de dispositivos. (Ubierna, 2020)

En la figura 1.20 se pueden observar las diferencias de consumo energético, velocidad y alcance, entre LoRa y otras tecnologías de red que normalmente se utilizan en IoT o soluciones tradicionales de conectividad entre dispositivos:



Figura 1.20: Diferencias entre tecnologías de red. Tomado de Ubierna (2020).

Como se aprecia en la imagen anterior, aunque la velocidad de transmisión es significativamente más lenta que otras tecnologías (con la excepción de Sigfox), LoRa posee una ventaja sustancial en términos de alcance y eficiencia energética en comparación con las demás tecnologías.

Modulación de radio y LoRa

LoRa, una técnica de modulación de espectro ensanchado propietaria derivada de la tecnología existente de Espectro Ensanchado Chirp (CSS), ofrece un equilibrio entre sensibilidad y velocidad de datos. Opera en un canal de ancho de banda fijo de 125 KHz o 500 KHz (para canales ascendentes) y 500 KHz (para canales descendentes). Además, LoRa utiliza factores de expansión ortogonales, lo que permite a la red preservar la vida útil de la batería de los nodos finales conectados mediante optimizaciones adaptativas de los niveles de potencia y tasas de datos de cada nodo final individual. Por ejemplo, un dispositivo final ubicado cerca de un *gateway* debería transmitir datos con un bajo factor de expansión, ya que se necesita muy poco margen de enlace. Sin embargo, un dispositivo final ubicado a varios kilómetros de un *gateway* necesitará transmitir con un factor de expansión mucho mayor. Este factor de expansión superior proporciona una mayor ganancia de procesamiento y una mayor sensibilidad de recepción, aunque la velocidad de datos será necesariamente más baja.

LoRa es puramente una implementación de capa física (PHY) o de "bits", según lo define el Modelo de Red de siete capas OSI, representado en la figura 1.21. En lugar de cables, se utiliza el aire como medio para transportar las ondas de radio LoRa desde un transmisor de RF en un dispositivo IoT hasta un receptor de RF en un *gateway*, y viceversa. (LoRa and LoRaWAN: Technical overview, sin fecha)

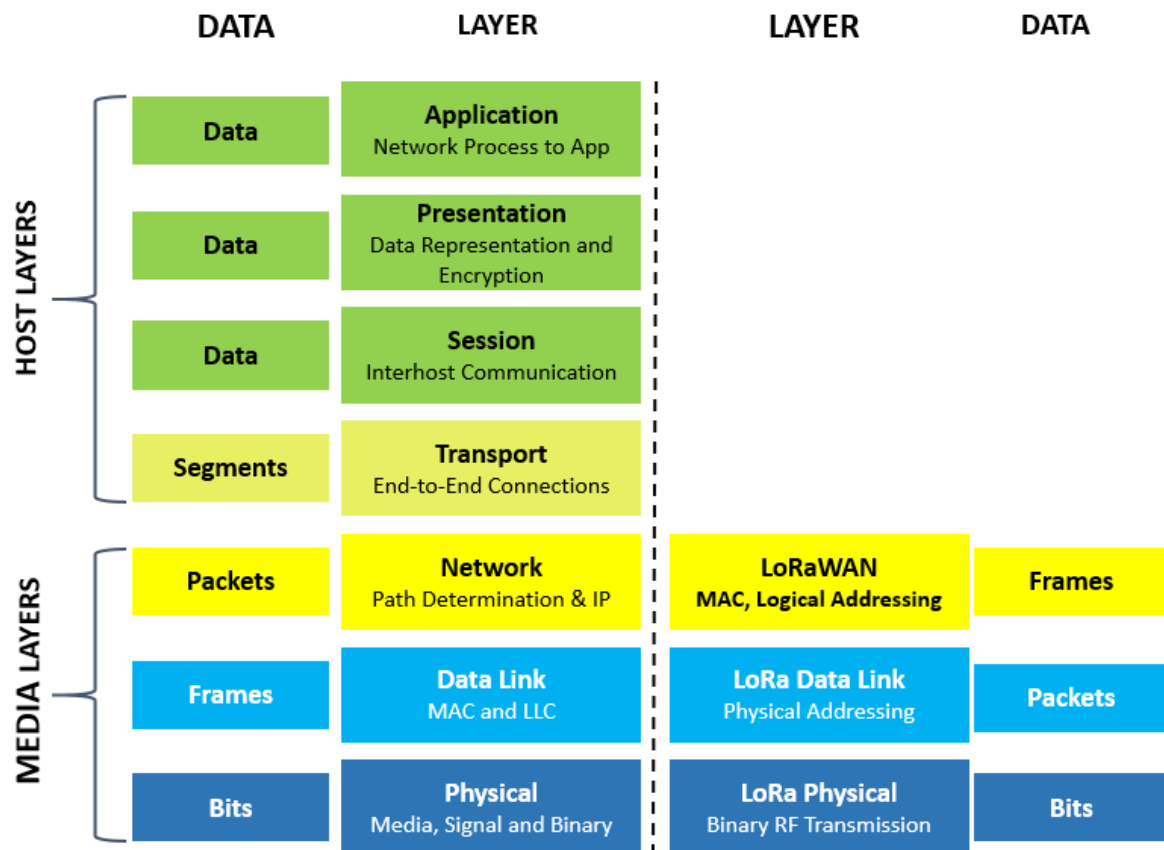


Figura 1.21: Modelo de Red de siete capas OSI destacando la posición que ocupan los protocolos LoRa. Tomado de “LoRa and LoRaWAN: Technical overview” (sin fecha).

En un sistema tradicional de Espectro Ensanchado por Secuencia Directa (DSSS), la fase del portador de la señal del transmisor cambia de acuerdo con una secuencia de códigos, como se muestra en la figura 1.22. Al multiplicar la señal de datos con un patrón de bits predefinido a una velocidad mucho más alta, también conocida como secuencia de expansión (o secuencia de chips), se crea una señal "más rápida" que tiene componentes de frecuencia más alta que la señal de datos original. Esto significa que el ancho de banda de la señal se expande más allá del ancho de banda de la señal original. En la terminología de RF, los bits de la secuencia de códigos se llaman chips (para distinguir entre los bits más largos y no codificados de la señal de datos original). Cuando la señal transmitida llega al receptor de RF, se multiplica con una copia idéntica del código de expansión utilizado en el transmisor de RF, lo que da como resultado una réplica de la señal de datos original. (LoRa and LoRaWAN: Technical overview, sin fecha)

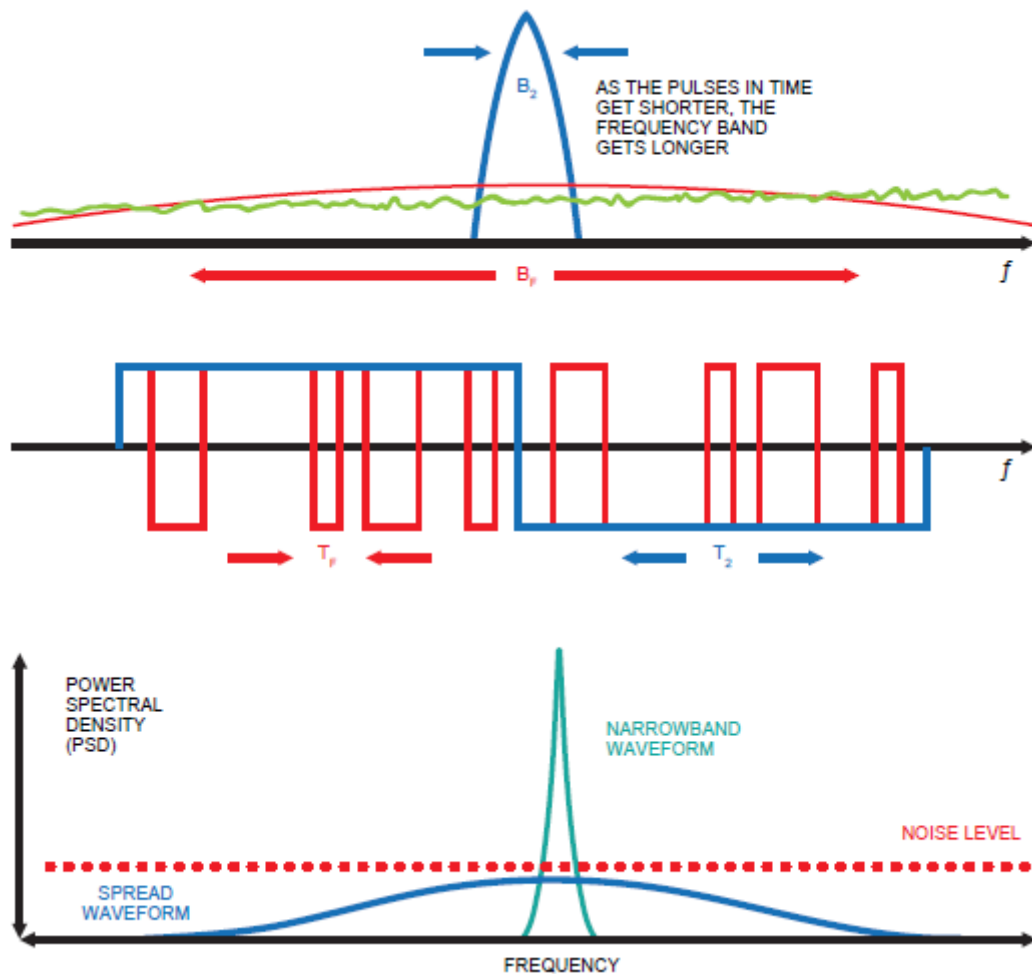


Figura 1.22: Cambios en la señal del transmisor de fase portadora del sistema DSSS. Tomado de “LoRa and LoRaWAN: Technical overview” (sin fecha).

Pasar por esta multiplicación de secuencia de códigos brinda un mayor presupuesto de enlace de RF, lo que te permite transmitir a una mayor distancia.

La relación Log10 entre la velocidad de chips de la secuencia de códigos y la velocidad de bits de la señal de datos se llama ganancia de procesamiento (G_p). Esta ganancia es lo que permite al receptor recuperar la señal de datos original, incluso si el canal tiene una relación señal-ruido (SNR) negativa. LoRa tiene una G_p superior en comparación con la modulación de cambio de frecuencia (FSK), lo que permite reducir el nivel de potencia de salida del transmisor mientras se mantiene la misma velocidad de datos de la señal y un presupuesto de enlace similar.

Una de las desventajas de un sistema DSSS es el hecho de que requiere un reloj de referencia altamente preciso (y costoso). La tecnología de espectro ensanchado por chirrido (CSS) de Semtech ofrece una alternativa DSSS de bajo costo y bajo consumo de energía, pero sólida, que no requiere un reloj de referencia altamente preciso. En la modulación LoRa, la expansión del espectro de la señal se logra generando una señal de chirrido que varía continuamente en frecuencia, como se

muestra en la figura 1.23.

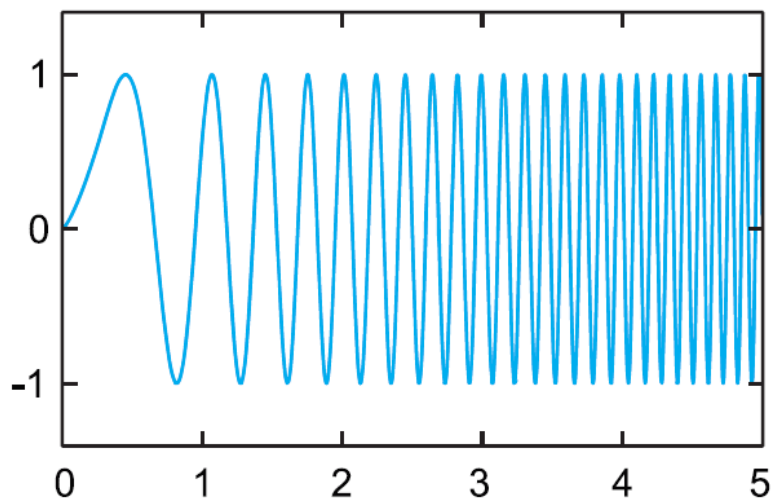


Figura 1.23: Ejemplo de variación continua de la frecuencia de la señal de chirrido. Tomado de “LoRa and LoRaWAN: Technical overview” (sin fecha).

Una ventaja de este método es que los desfases de tiempo y frecuencia entre el transmisor y el receptor son equivalentes, lo que reduce en gran medida la complejidad del diseño del receptor. El ancho de banda de frecuencia de este chirrido es equivalente al ancho de banda espectral de la señal. La señal de datos que lleva los datos de un dispositivo final a un *gateway* se divide en chips a una velocidad de datos más alta y se modula en la señal portadora de chirrido. La modulación LoRa también incluye un esquema de corrección de errores variable que mejora la robustez de la señal transmitida. Por cada cuatro bits de información enviados, se envía un quinto bit de información de paridad.

1.10 Visual Wake Words Dataset

El dataset Visual Wake Words (VWW) es un conjunto de datos públicos de imágenes etiquetadas con la presencia o ausencia de una persona. Fue creado por Google en 2019 para facilitar el desarrollo de modelos de *computer vision* pequeños para microcontroladores.

El Conjunto de Datos de VWW es un banco de pruebas realista para modelos de *computer vision* pequeños, ya que se centra en un caso de uso común de visión en microcontroladores: la detección de la presencia de una persona. El conjunto de datos contiene más de 115,000 imágenes del dataset público COCO, que ha sido filtrado para eliminar imágenes que no contienen personas.

El Conjunto de Datos de VWW es un recurso valioso para investigadores y desarrolladores interesados en desarrollar modelos de *computer vision* para microcontroladores. Puede utilizarse para entrenar y evaluar modelos para una variedad de aplicaciones, como:

- Detección de la presencia humana en dispositivos para el hogar inteligente.
- Seguimiento de personas en dispositivos ponibles y otros dispositivos de IoT.
- Reconocimiento de actividades para dispositivos de seguimiento de la actividad física y otros dispositivos ponibles.
- Detección de objetos para la automatización industrial y otras aplicaciones de IoT industrial.

1.11 Técnicas de preprocesamiento de datos

1.11.1 Conversión de imágenes RGB a escala de grises

La representación de colores como valores numéricos es un paso necesario en muchas aplicaciones. Para lograr esto, utilizamos modelos matemáticos que describen formas de asignar colores a un conjunto de números. Por lo general, un modelo de color define tres o cuatro componentes de color que se describen fácilmente a través de un sistema de coordenadas. Cada color que el modelo puede representar corresponde a un punto en este sistema de coordenadas. Además de estos valores, estos modelos también proporcionan una descripción de cómo interpretar estos componentes para generar un color. (Antoniadis, 2023)

El modelo de color más conocido es el RGB, que significa Rojo-Verde-Azul. Como su nombre indica, este modelo representa colores utilizando valores individuales para rojo, verde y azul. El modelo RGB se utiliza en casi todas las pantallas digitales en todo el mundo.

Específicamente, un color se define mediante tres valores enteros que van de 0 a 255 para rojo, verde y azul, donde un valor de cero significa oscuro y un valor de 255 significa brillante. Dados los valores, el color final se define cuando mezclamos estos tres colores básicos ponderados por sus valores.

Si mezclamos los tres colores por igual ($RGB = (255, 255, 255)$), obtendremos blanco, mientras que la ausencia de todos los colores ($RGB = (0, 0, 0)$) significa negro. En la figura 1.24 se muestra el sistema de coordenadas RGB donde podemos ver todos los diferentes colores que el modelo puede describir.

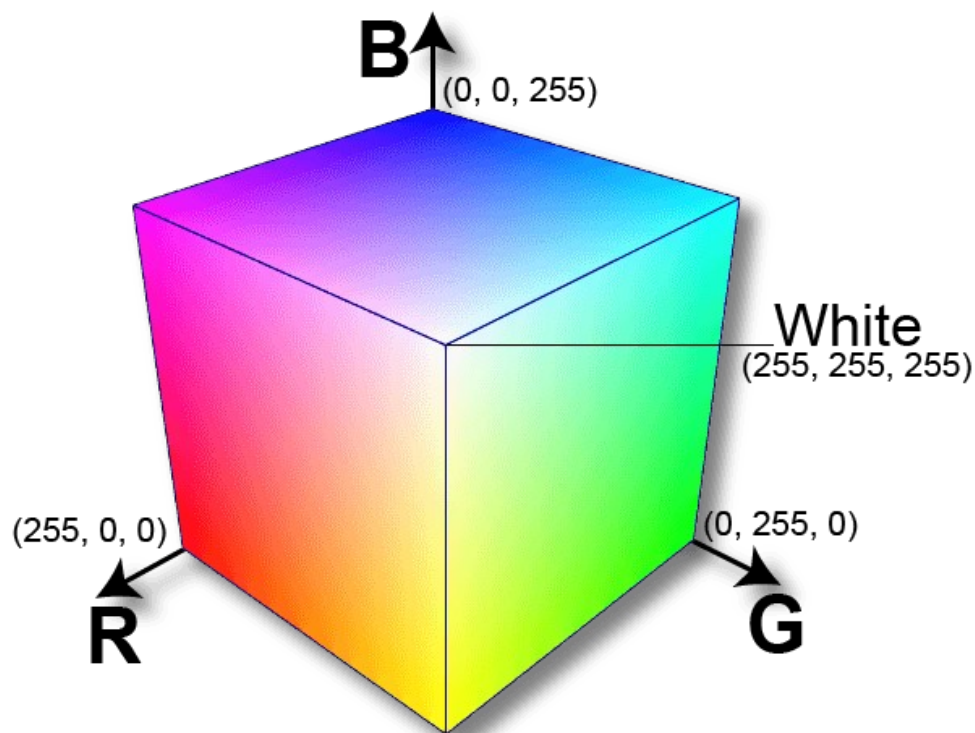


Figura 1.24: Sistema de coordenadas para imágenes RGB. Tomado de Antoniadis (2023).

Por otro lado, tenemos las imágenes en escala de grises, un modelo más sencillo que el RGB, ya que representa los colores mediante un único componente: la luminosidad. Esta se expresa con un valor numérico que va desde 0 (negro absoluto) hasta 255 (blanco puro).

Por un lado, las imágenes en escala de grises transmiten menos información que las RGB. Sin embargo, son comunes en el procesamiento de imágenes porque el uso de una imagen en escala de grises requiere menos espacio disponible y es más rápido, especialmente cuando se trata de cálculos complejos. (Antoniadis, 2023)

En la figura 1.25, podemos observar un ejemplo de la gama de tonos que el modelo de escala de grises es capaz de representar.

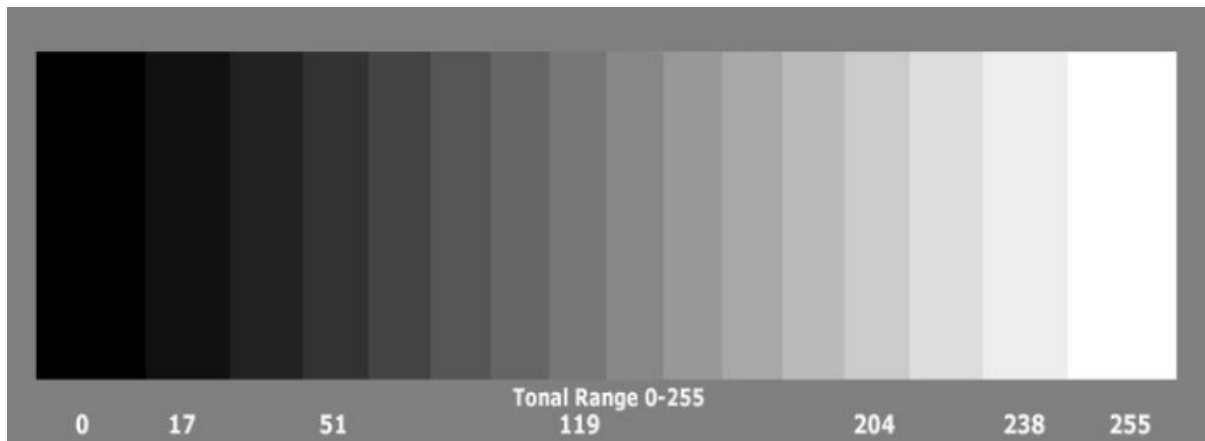


Figura 1.25: Gama de colores para imágenes en escala de grises. Tomado de Antoniadis (2023).

A la hora de realizar transformaciones de imágenes RGB a escala de grises, existen distintos métodos, los cuales tres de los más utilizados se muestran a continuación.

Método de Brillo

Como se muestra en la Ecuación 1.8, este método consiste en calcular el valor promedio entre el componente con el valor más alto y el componente con el valor más bajo dentro del canal RGB.

$$\text{grayscale} = \frac{\min(R,G,B) + \max(R,G,B)}{2}$$

Ecuación 1.8: Transformación a escala de grises mediante el método de brillo.

Podemos ver fácilmente que este método presenta una debilidad importante, ya que no se utiliza uno de los componentes RGB. Esto es definitivamente un problema porque la cantidad de luminosidad que percibe nuestro ojo depende de los tres colores básicos.

Método Promedio

Este método calcula el promedio aritmético de los tres canales de color (rojo, verde y azul) para determinar el valor en escala de grises, como se expresa en la Ecuación 1.9.

$$\text{grayscale} = \frac{R+G+B}{3}$$

Ecuación 1.9: Transformación a escala de grises mediante el método del promedio.

Aunque ahora tenemos en cuenta todos los componentes, el método del promedio también es problemático ya que asigna el mismo peso a cada componente. Según la investigación sobre la visión humana, sabemos que nuestros ojos reaccionan de

manera diferente a cada color. Específicamente, nuestros ojos son más sensibles al verde, luego al rojo y finalmente al azul. Por lo tanto, los pesos en la ecuación anterior deberían cambiar.

Método de Luminosidad

Este método soluciona las limitaciones de los anteriores aplicando un promedio ponderado a los componentes RGB. Basándose en el comportamiento del sistema visual humano, se asignan pesos distintos a cada canal: se reduce la influencia del azul y se incrementa la del verde. Tras diversos estudios y experimentos, se ha llegado a la fórmula que se muestra en la Ecuación 1.10.

$$\text{grayscale} = 0.3 * R + 0.59 * G + 0.11 * B$$

Ecuación 1.10: Transformación a escala de grises mediante el método de luminosidad.

En la figura 1.26 se puede ver un ejemplo de la aplicación de los 3 métodos en la misma imagen. (Antoniadis, 2023)

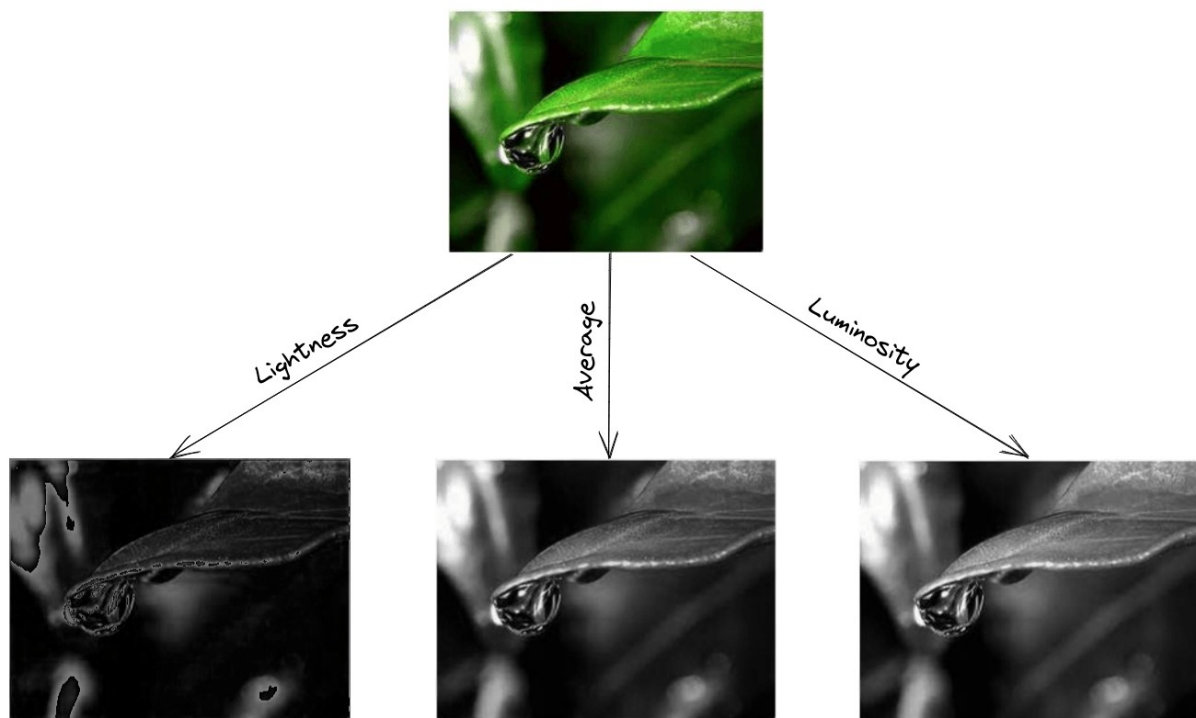


Figura 1.26: Ejemplo de aplicación de los 3 métodos de conversión. Tomado de Antoniadis (2023).

1.11.2 Reducción de imágenes

La "Reducción de imágenes" es una técnica fundamental en el preprocesamiento de imágenes que se utiliza para disminuir el tamaño de una imagen mientras se conserva la calidad visual. Una de las técnicas más comunes para lograr esto es la interpolación, y un método particularmente eficaz es el uso de filtros LANCZOS.

Cuando se reduce una imagen, es esencial mantener la claridad y los detalles para evitar la pérdida de información importante. Los filtros LANCZOS son una categoría de filtros que se utilizan en la interpolación, que es el proceso de estimar nuevos valores de píxeles en una imagen más pequeña a partir de los valores de los píxeles originales.

Los filtros LANCZOS se destacan por su capacidad para preservar los detalles finos y minimizar el efecto de aliasing, que es un fenómeno no deseado que puede ocurrir durante la reducción de imágenes. Estos filtros funcionan calculando una combinación ponderada de píxeles cercanos para determinar el valor de un nuevo píxel en la imagen reducida. Esto permite suavizar la imagen de manera controlada, manteniendo los bordes y detalles nítidos.

En resumen, la "Reducción de imágenes" es una parte esencial del preprocesamiento de imágenes, y los filtros LANCZOS son una técnica efectiva para lograr esta reducción sin sacrificar la calidad y los detalles de la imagen original. Esta técnica es particularmente útil en aplicaciones donde se necesita conservar la información visual crítica, como la visualización de imágenes en dispositivos móviles o la compresión de imágenes para su almacenamiento eficiente.

1.11.3 Data augmentation

La aumentación de datos es un conjunto de técnicas para aumentar artificialmente la cantidad de datos mediante la generación de nuevos puntos de datos a partir de datos existentes. Esto incluye realizar pequeños cambios en los datos o utilizar modelos de *deep learning* para generar nuevos puntos de datos.

La aumentación de datos es útil para mejorar el rendimiento y los resultados de los modelos de aprendizaje automático al crear nuevos y diferentes ejemplos para conjuntos de datos de entrenamiento. Si el conjunto de datos en un modelo de *machine learning* es rico y suficiente, el modelo funciona mejor y con mayor precisión.

Para los modelos de *machine learning*, la recopilación y etiquetado de datos pueden ser procesos agotadores y costosos. Las transformaciones en los conjuntos de

datos mediante técnicas de aumentación de datos permiten reducir estos costos operativos.

Uno de los pasos en un modelo de datos es la limpieza de datos, que es necesaria para modelos de alta precisión. Sin embargo, si la limpieza reduce la representatividad de los datos, el modelo no puede proporcionar buenas predicciones para entradas del mundo real. Las técnicas de aumentación de datos pueden permitir que los modelos de *machine learning* sean más robustos al crear variaciones que el modelo puede encontrar en el mundo real (Fig. 1.27). (Dilmegani, 2022)

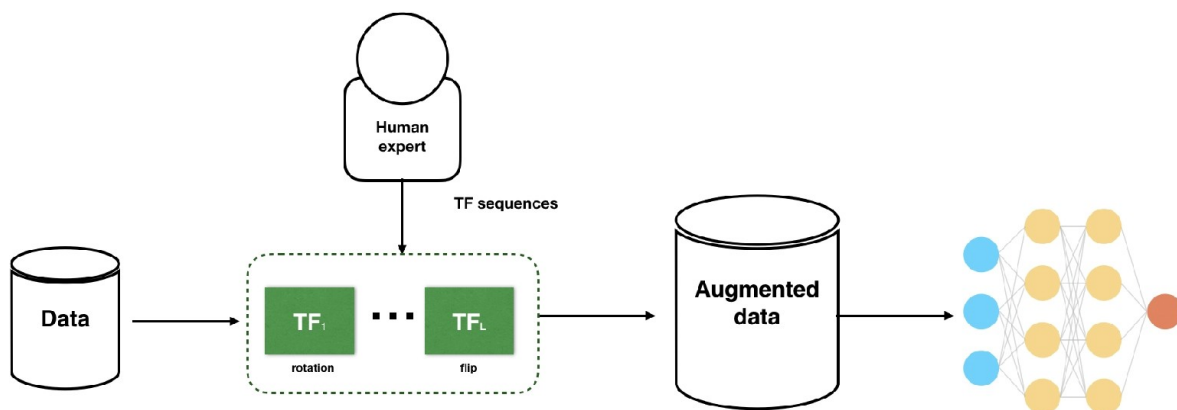


Figura 1.27: Proceso de Aumentación de datos para entrenar modelos de deep learning. Tomado de Dilmegani (2022).

Para la aumentación de datos, es común realizar simples alteraciones en los datos visuales (Fig. 1.28). Las actividades clásicas de procesamiento de imágenes para la aumentación de datos incluyen:

- Añadir margen
- Rotación aleatoria
- Reescalado
- Volteo vertical y horizontal
- Traslación (mover la imagen en dirección X e Y)
- Recorte
- Zoom
- Oscurecimiento y aclarado / modificación de colores
- Escala de grises
- Cambio de contraste
- Agregar ruido
- Borrado aleatorio

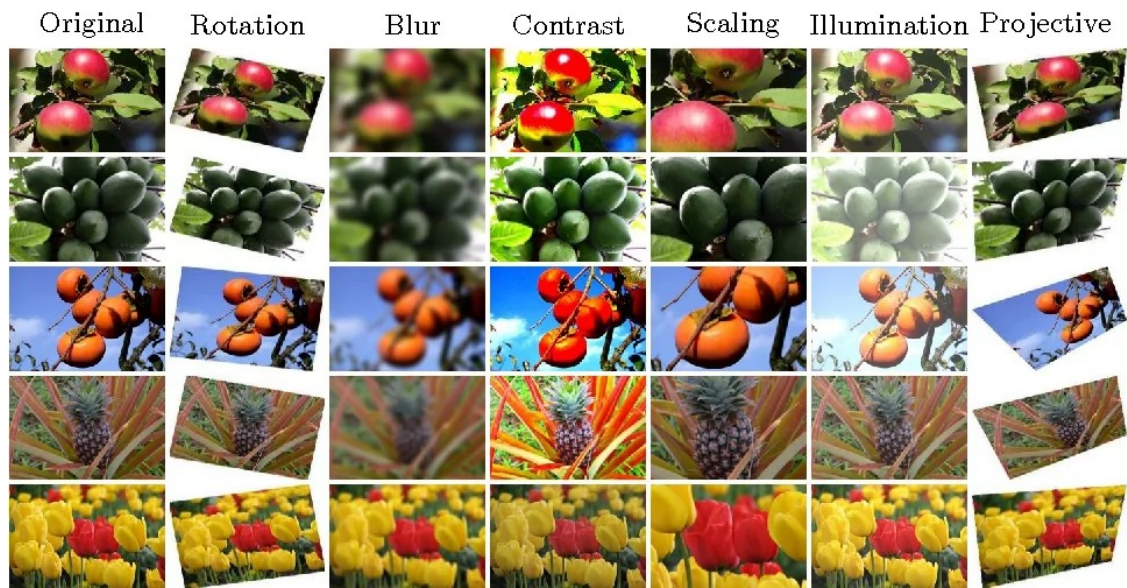


Figura 1.28: Ejemplos de Aumentación de datos sobre imágenes. Tomado de Dilmegani (2022).

1.12 Objetivo

1.12.1 Objetivo general

Teniendo en cuenta lo expuesto anteriormente, se propone desarrollar un módulo de detección de personas mediante el uso de *Edge Computer Vision*, diseñado específicamente para su implementación en microcontroladores de bajo costo y que pueda ser utilizado en zonas rurales donde el acceso a la red eléctrica y la infraestructura de comunicación sea limitada o inexistente.

1.12.2 Objetivos específicos

- Desarrollar y entrenar un modelo de *Machine Learning* basado en redes neuronales para la detección de personas adaptado para funcionar en microcontroladores.
- Desplegar el modelo de detección de personas en un microcontrolador, el cual este disponible en el mercado local y sea de bajo costo.
- La solución debe ser eficiente desde el punto de vista de consumo energético.
- El módulo debe autoabastecerse energéticamente
- Debe poseer un módulo de comunicación que pueda enviar alertas de la detección a grandes distancias.

2. Materiales y métodos

2.1 Hardware

- Notebook: Se utilizó para el desarrollo de la solución y además para el entrenamiento de los modelos. Características:
 - procesador Intel i7 (5° gen)
 - GPU NVIDIA GeForce 920M
 - 16GB de memoria RAM
 - 1TB de espacio de almacenamiento
 - Windows 10
- Placa de desarrollo “ESP32-CAM”. Este dispositivo es parte de la solución la cual se utiliza para detectar a las personas a través de su cámara integrada (Fig. 2.1). Características:
 - Marca: “AI Thinker”
 - Modelo: ESP32-CAM
 - Modelo del módulo: ESP32-S
 - Memoria SRAM: 520 KB
 - Memoria PSRAM: 8 MB
 - Voltaje de alimentación: 5V
 - Tipo de cámara: OV2640
 - Velocidad máxima de reloj: 240 MHZ
 - Módulo Wi-Fi BT 802.11b/g/n

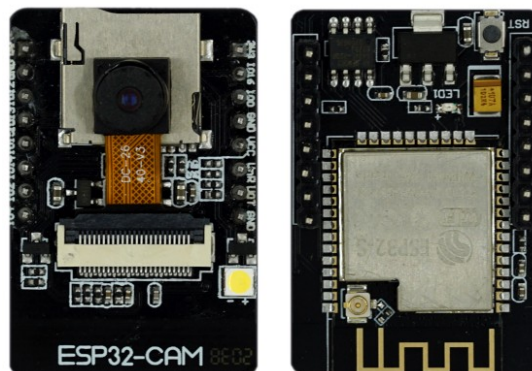


Figura 2.1: Placa de desarrollo ESP32-CAM.

- Módulo interfaz USB TTL. Este dispositivo se utilizó principalmente para desplegar en la placa de desarrollo ESP32-CAM los programas desarrollados en este trabajo. Además se lo utilizó como interfaz para alimentar con energía al ESP32-CAM desde el puerto USB (Fig. 2.2). Características:

- Marca: Funduino
- Modelo: FTDI FT232RL

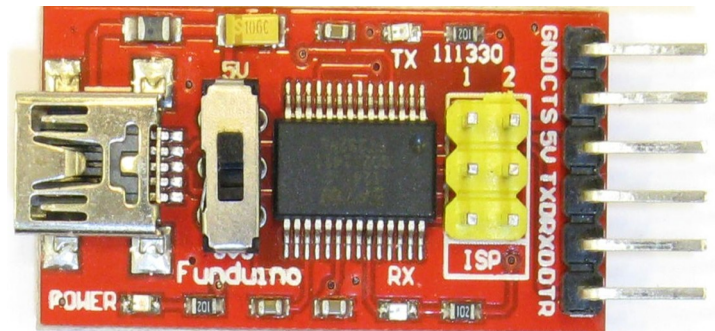


Figura 2.2: Módulo interfaz USB TTL.

- Placa de desarrollo “Arduino Uno”. Este dispositivo se utilizó para interactuar con un receptor Lora, el cual tuvo como objetivo testear que el transmisor de la solución esté funcionando correctamente (Fig. 2.3). Características:
 - Marca: Atmega
 - Modelo: Arduino UNO
 - Voltaje de alimentación: 5V
 - Pins digitales: 14
 - Pins Analógicos: 6
 - Memoria flash: 32 KB
 - SRAM 2 KB
 - Velocidad de reloj: 16 MHZ



Figura 2.3: Placa de desarrollo Arduino UNO.

- Transceptor Lora. Se utilizaron 2 transceptores (Fig. 2.4) para la comunicación entre los módulos ESP32-CAM y Arduino UNO. Características:
 - Marca: Generico

- Modelo: SX1278
- Voltaje de alimentación: 1.8V a 3.7V
- Potencia máxima de salida: 20 dBm
- Velocidad de transferencia de datos: 1.2-300kbps

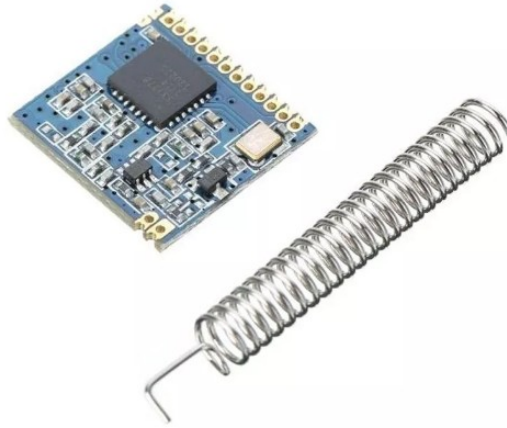


Figura 2.4: *Transceptor Lora Sx1278.*

- Panel Solar 5V 250 ma. Se utilizó un panel solar monocristalino (Fig. 2.5) para suministro energético de la solución. Características
 - Potencia Maxima en STC (P_{max}): 1.25W
 - Voltaje de funcionamiento optimo (V_{mp}): 5V
 - Corriente de funcionamiento optima (I_{mp}): 250mA
 - Tamaño: 130mm x 65mm



Figura 2.5: *Panel solar monocristalino.*

- Batería recargable (Fig. 2.6). Se la utilizó para el suministro energético de la solución. Características:
 - Marca: Samsung

- Modelo: ICR18650-26J
- Voltaje: 3.7V
- Capacidad: 2600 mAh



Figura 2.6: Batería recargable.

- Módulo de carga de batería. El módulo (Fig. 2.7) fue utilizado para gestionar la carga de la batería con la energía proveniente del panel solar.

Características:

- Modelo: TC4056A
- Voltaje nominal de entrada: 5V
- Corriente máxima de carga: 1000mA
- Voltaje de corte de carga (cut-off): $4.2V \pm 1\%$
- Protección de sobre-descarga de la batería

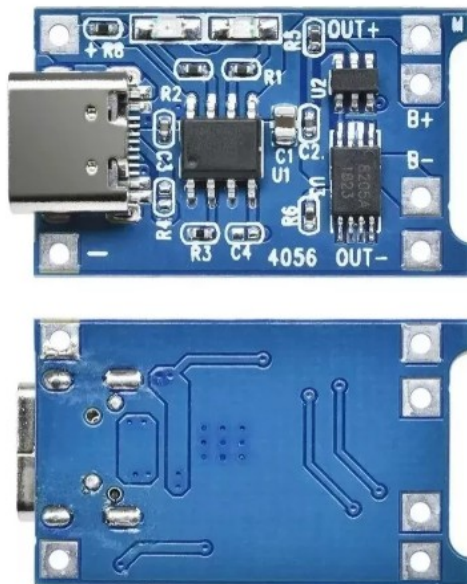


Figura 2.7: Módulo de carga de batería.

- Cargador portátil. Este cargador (Fig. 2.8) se utilizó para alimentar al módulo para colecta de imágenes.
 - Marca: Sony
 - Modelo: CycleEnergy

- Capacidad: 2000 mAh



Figura 2.8: Cargador portátil.

2.2 Software

- Lenguajes de programación:
 - Python: utilizado para el desarrollo del modelo y su respectivo entrenamiento y validación.
 - C++: utilizado para desarrollar las aplicaciones que son implementadas en los microcontroladores.
 - JavaScript: utilizado para desarrollo de la API que fue utilizada para la recolección de imágenes para el dataset.
- Principales Librerías:
 - Tensorflow
 - numpy
 - matplotlib
 - Pandas
 - csv
 - datetime
 - os
 - sys
 - zipfile
 - google.cloud
 - google.colab
 - PIL
- Visual Studio Code (IDE). Se lo utilizó para el desarrollo del módulo de entrenamiento (python) y del código que fue desplegado en el módulo ESP32-CAM (C++).

- ESP-IDF: Este es el entorno de desarrollo de software y despliegue para el hardware basado en el chip ESP32 de Espressif. Se utilizó la versión 4.4.
- Arduino IDE. Se utilizó para el desarrollo del código y despliegue sobre el módulo Arduino UNO.
- CUDA: esta plataforma permite utilizar las unidades de procesamiento gráfico (GPU) de NVIDIA para realizar tareas de entrenamiento del modelo en el cual se requiere de un cómputo intensivo de manera eficiente y paralela.
- WSL (Windows Subsystem for Linux): se utilizó esta herramienta de Windows para ejecutar comandos de Linux. Para este trabajo se la utilizó para transformar el modelo generado con la librería Tensorflow Lite a un archivo hexadecimal para ser incorporado el código del módulo del ESP32-CAM.

2.3 Recursos Cloud

- Google Colab: Se utilizó esta plataforma para ejecutar el módulo de entrenamiento del modelo de redes neuronales. Esta plataforma ofrece servicios de procesamiento en la nube que brindan capacidades técnicas favorables para acelerar el entrenamiento de los modelos. Su característica más destacada es la posibilidad de utilizar arquitecturas de procesamiento que incluyen GPUs (unidades de procesamiento gráficos), las cuales permiten paralelizar las operaciones necesarias para el entrenamiento de los modelos. (Tigreros, 2019)
- Bucket (Google Cloud Storage): es un servicio de almacenamiento de la nube de GCP el cual se utilizó en este trabajo para el almacenamiento de los datos de entrenamiento, los modelos entrenados, métricas de performance, etc.
- Google Drive: es un servicio de almacenamiento en la nube, el cual se empleó específicamente para guardar las imágenes capturadas por el módulo ESP32-CAM, las cuales posteriormente se utilizaron en el desarrollo del modelo.
- APP Script: es un servicio de Google que provee una plataforma de JavaScript basada en la nube que permite integrar y automatizar tareas con productos de Google. En este trabajo se lo utilizó para disponibilizar una API para recibir imágenes y almacenarlas en Google Drive.
- Repositorio GitHub: se utilizó para el almacenamiento del código generado y la gestión de versiones.

3. Desarrollo y resultados

3.1 Descripción general de la solución

La solución está compuesta por los siguientes módulos (Fig.3.1):

- Módulo de detección de personas
- Módulo de comunicación
- Módulo de suministro energético

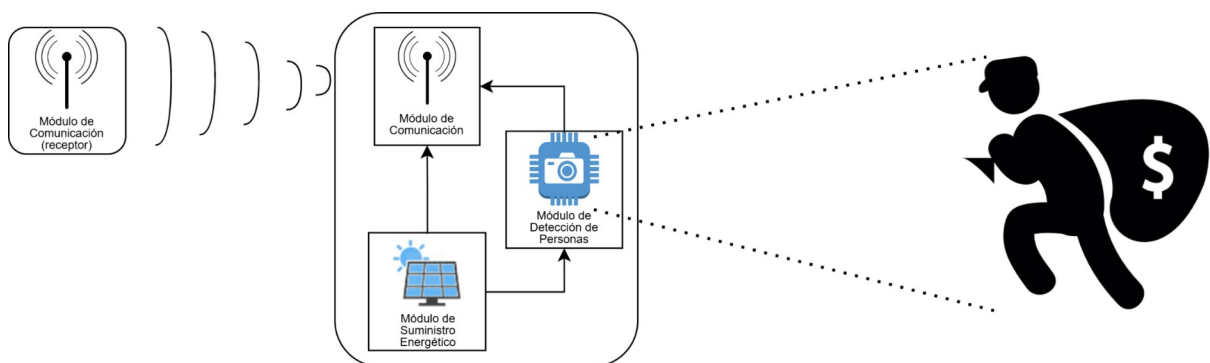


Figura 3.1: Diseño de la solución.

Cada uno de estos módulos trabaja de manera conjunta para ofrecer una solución completa y eficiente en la detección de personas en entornos rurales.

En la figura 3.2 pueden visualizarse imágenes del prototipo armado.

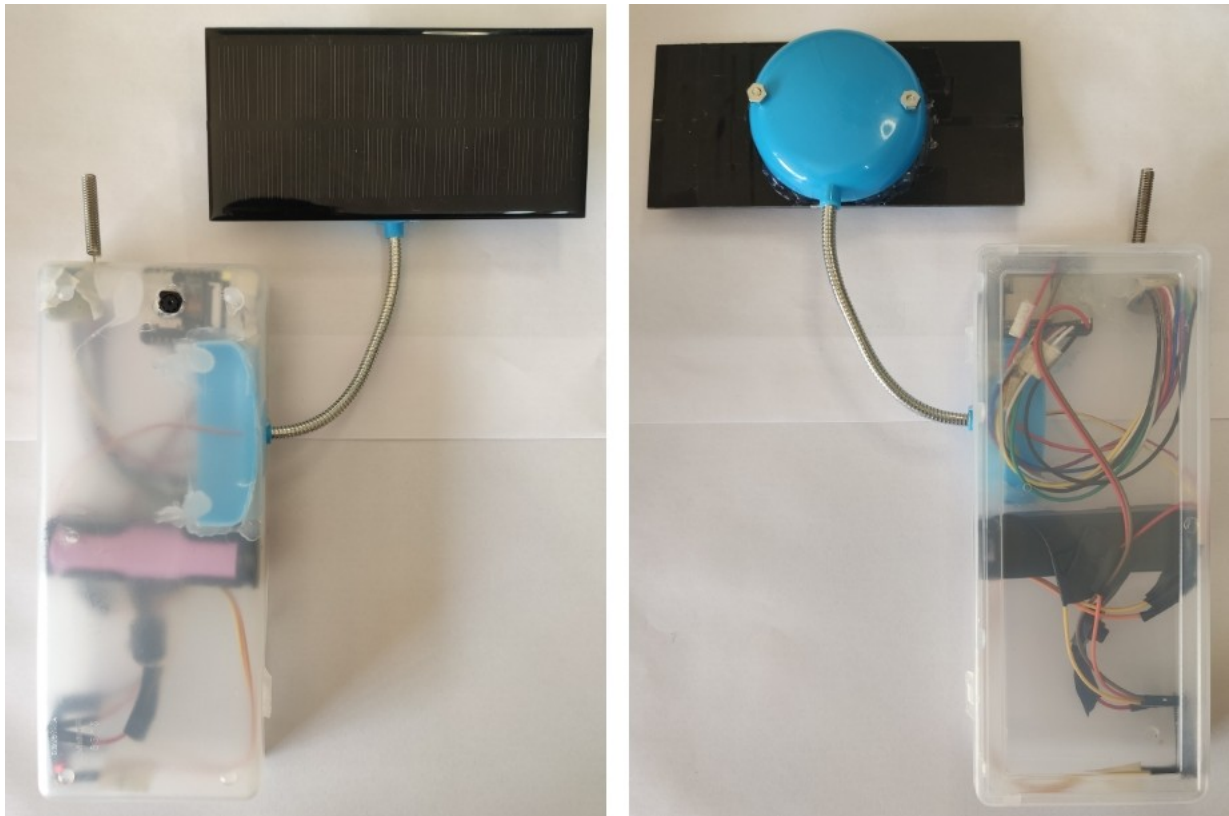


Figura 3.2: Prototipo - Vista frontal y trasera.

3.1.1 Módulo de detección de personas

El módulo de detección de personas está conformado por una placa de desarrollo ESP32-CAM el cual consta de un microcontrolador ESP32 y una cámara que permite capturar imágenes del entorno.

Se desarrolló e implementó en dicho microcontrolador un modelo de redes neuronales para la detección de personas utilizando como entrada las imágenes capturadas por la cámara del módulo.

El modelo implementado fue desarrollado en Python y se utilizó la librería Tensorflow Lite para lograr que dicho modelo tenga un tamaño reducido que satisfaga las capacidades de almacenamiento limitadas del microcontrolador seleccionado.

Por otro lado, la lógica implementada en el microcontrolador, la cual se encarga de la captura de imágenes, la ejecución del modelo y la gestión de la salida en el microcontrolador se desarrolló utilizando el lenguaje C++.

3.1.1.1 Lógica implementada en el módulo ESP32-CAM

Como base para este trabajo, se tomó como referencia un código fuente disponible en el repositorio de TensorFlow Lite (TensorFlow Lite for Microcontrollers, sin fecha). Este código constituye un ejemplo específico de detección de personas en microcontroladores.

En dicho código se realizaron diferentes modificaciones, principalmente en la sección donde se inicializan los operadores de TensorFlow Lite que ejecutan la arquitectura del modelo implementado. Dado que la arquitectura en este trabajo es diferente, se incorporaron operadores adicionales para su ejecución

Por otro lado, se incorporó una sección para la gestión del módulo de comunicación (el cual consta del transceptor LoRa) para que este pueda transmitir el mensaje generado por el módulo.

Finalmente, se modificó la sección que gestiona la salida del modelo para que, al superar el umbral definido, el módulo active el módulo de comunicación y envíe la alerta.

La figura 3.3 muestra el diagrama de flujo que describe la lógica implementada en la aplicación desplegada en el módulo ESP32-CAM.

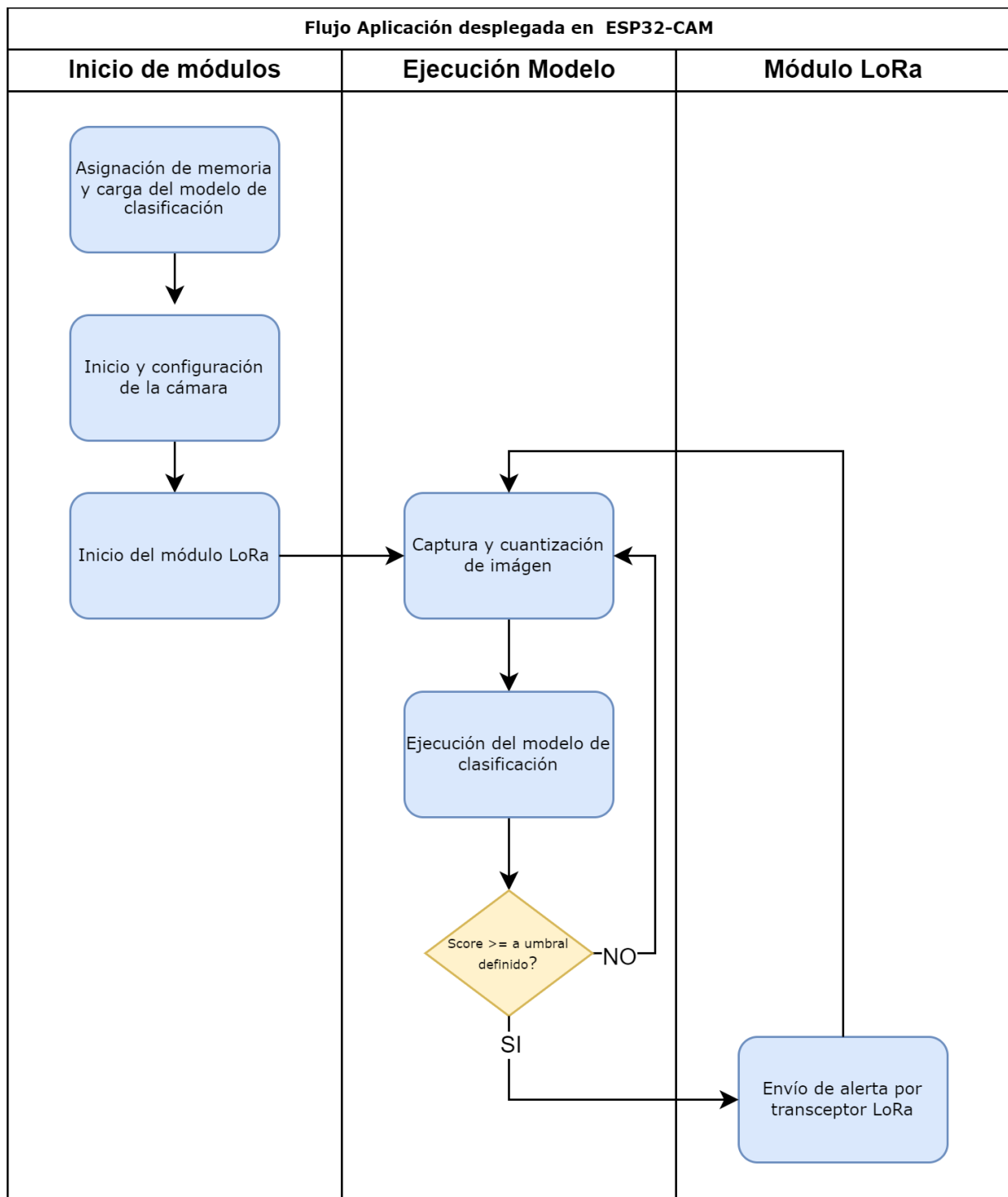


Figura 3.3: Diagrama de Flujo de la aplicación desplegada en módulo ESP32-CAM.

Es importante señalar que la lógica de funcionamiento se divide en tres etapas generales: la primera implica la inicialización de los diversos servicios que se utilizarán más adelante; a continuación, se lleva a cabo la ejecución del modelo; y finalmente, se emplea el módulo de comunicación LoRa. Seguidamente se describen cada una de las partes de este flujo.

Asignación de memoria y carga del modelo de clasificación.

En esta sección del código se realizan las configuraciones correspondientes a la asignación de memoria para alojar los tensores de entrada al modelo y también para montar el modelo.

Además, se inicializan únicamente los operadores de TensorFlow Lite que se utilizarán para ejecutar la arquitectura del modelo implementado. Es decir, en lugar de utilizar la función "AllOpsResolver" la cual inicia todos los operadores de la librería TensorFlow Lite (como suele hacerse al desarrollar un modelo para ser ejecutado en un dispositivo con más memoria), lo cual requiere más memoria al necesitar cargar en memoria también los operadores que no se utilizarán, aquí solo se inicializan los operadores que se utilizarán en el modelo. Esto permite utilizar exactamente la memoria requerida por dichos operadores.

Inicio y configuración de la cámara

En esta sección, se lleva a cabo la inicialización de la cámara junto con la configuración necesaria para su funcionamiento. En este proyecto, se ha considerado que las imágenes capturadas sean en escala de grises y que tengan un tamaño de 96 x 96 píxeles.

Inicio del módulo LoRa

En esta sección, se procede a la inicialización del transceptor LoRa, especificando la frecuencia de operación y configurando el algoritmo destinado a verificar la integridad de los datos transmitidos. En cuanto a esta última configuración, se ha optado por utilizar el algoritmo CRC (Cyclic Redundancy Check).

Captura y cuantización de la Imagen

En esta etapa, se realiza la captura de la imagen con la cámara del módulo, que se genera en formato uint8 (entero sin signo de 8 bits) con un rango de valores de 0 a 255. Como el modelo implementado ha sido cuantizado a int8 (entero de 8 bits en el rango -127 a 128), para principalmente reducir su tamaño, la imagen también debe cuantizarse a int8 para poder procesarla con el modelo implementado (Fig. 3.4). La lógica para transformar los valores de cada pixel expresados en formato uint8 (que toman valores de 0 a 255) a su equivalente en int8 (que toman valores de -127 a 128) se expresa en la ecuación 3.1.

$$int8_image = (uint8_image / scale) + zero_point$$

Ecuación 3.1: Cálculo del valor cuantizado.

Los parámetros *zero_point* y *scale* son parámetros que son provistos por el modelo

implementado.

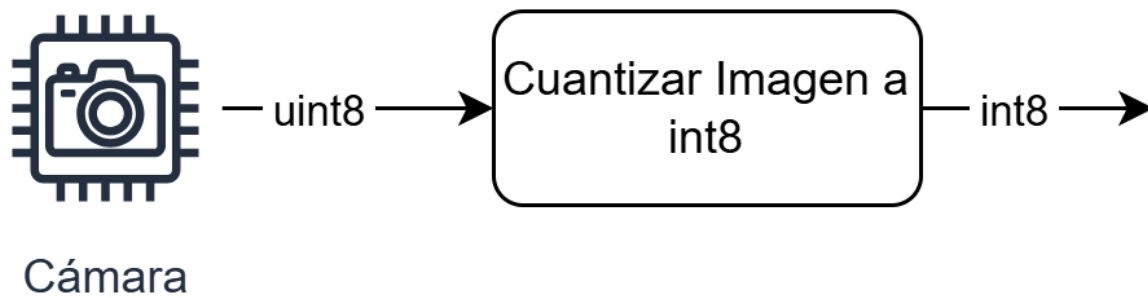


Figura 3.4: Cuantización de imagen tomada por la cámara.

Ejecución del modelo de clasificación

Una vez que se ha capturado la imagen y se ha aplicado la cuantización requerida, se procede a invocar el modelo y se captura su salida, que está representada por dos valores. Cada uno de estos valores corresponde al puntaje de cada categoría del modelo, es decir, "Persona detectada" y "Persona no detectada."

Como el modelo está cuantizado en formato int8, los valores de salida son en dicho formato, lo que implica que el valor se encuentra en una escala que va desde -128 hasta 127, en contraste con el rango de 0 a 1 que se encuentra en el modelo sin cuantizar.

Para obtener una aproximación al valor real original, se utiliza la fórmula presentada en la ecuación 3.2.

$$Real_value = (int8_value - zero_point) \times scale$$

Ecuación 3.2: Aproximación al valor des-cuantizado.

Esta fórmula transforma el valor int8 en un valor que varía entre 0 y 1 para cada una de las categorías.

Evaluación del umbral: ¿Score mayor o igual al umbral definido?

En esta fase se evalúa el puntaje de la categoría "Persona detectada," que se expresa como un porcentaje (multiplicado por 100), y se compara con un umbral o punto de corte previamente establecido.

Si el valor es igual o superior al umbral definido, se procede a avanzar con el módulo de comunicación para enviar la alerta. En caso contrario, se retorna a la fase de captura y procesamiento de imágenes.

Envío de alerta por transceptor LoRa

En esta instancia se arma el mensaje a enviar, se lo empaqueta y se lo envía al receptor.

El mensaje enviado tiene el formato general:

“Person detected: <value> %”

En <value> se adiciona el valor de salida de la categoría “Persona detectada” expresado en porcentaje (valor entre 0 y 100).

3.1.2 Módulo de Comunicación

Transceptor Emisor

Como se explicó en la sección anterior, cuando se detecta un objetivo mediante el módulo de detección de personas, se activa el módulo de comunicación para enviar una alerta utilizando el transceptor LoRa.

Para realizar la integración del transceptor con el módulo ESP32-CAM se siguieron las siguientes conexiones entre los pines de los correspondientes dispositivos (Fig. 3.5):

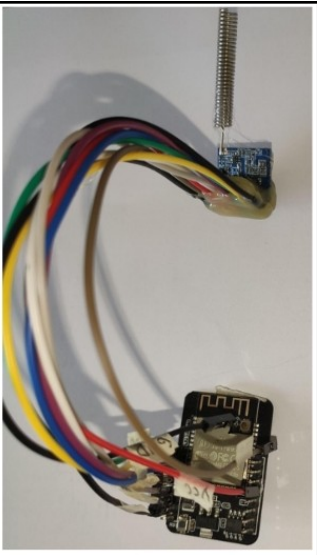
Transceptor LoRa	ESP32-CAM	
DI00	IO4	
Reset	IO2	
NSS	IO14	
SCK	IO15	
MOSI	IO13	
MISO	IO12	
VCC	VCC	
GND	GND	

Figura 3.5: Conexiones de los puertos del transceptor con el módulo ESP32-CAM.

Transceptor Receptor

La señal enviada por el transceptor emisor es recibida por otro módulo equipado también con un transceptor LoRa, el cual se puede utilizar para llevar a cabo diversas acciones, como activar una alarma, almacenar los datos recibidos, entre otras posibilidades.

Es importante destacar que este trabajo se enfoca exclusivamente en el proceso de envío de la señal y su confirmación de recepción por parte del transceptor receptor. Por esta razón, se diseñó y construyó este módulo con el único propósito de validar el correcto funcionamiento del emisor y asegurarse de que el mensaje de alerta se reciba correctamente.

Para la construcción de este módulo receptor se utilizó un microcontrolador Arduino UNO y un transceptor LoRa.

La figura 3.6 muestra una captura de pantalla del mensaje recibido, que se muestra en la consola del IDE de Arduino.

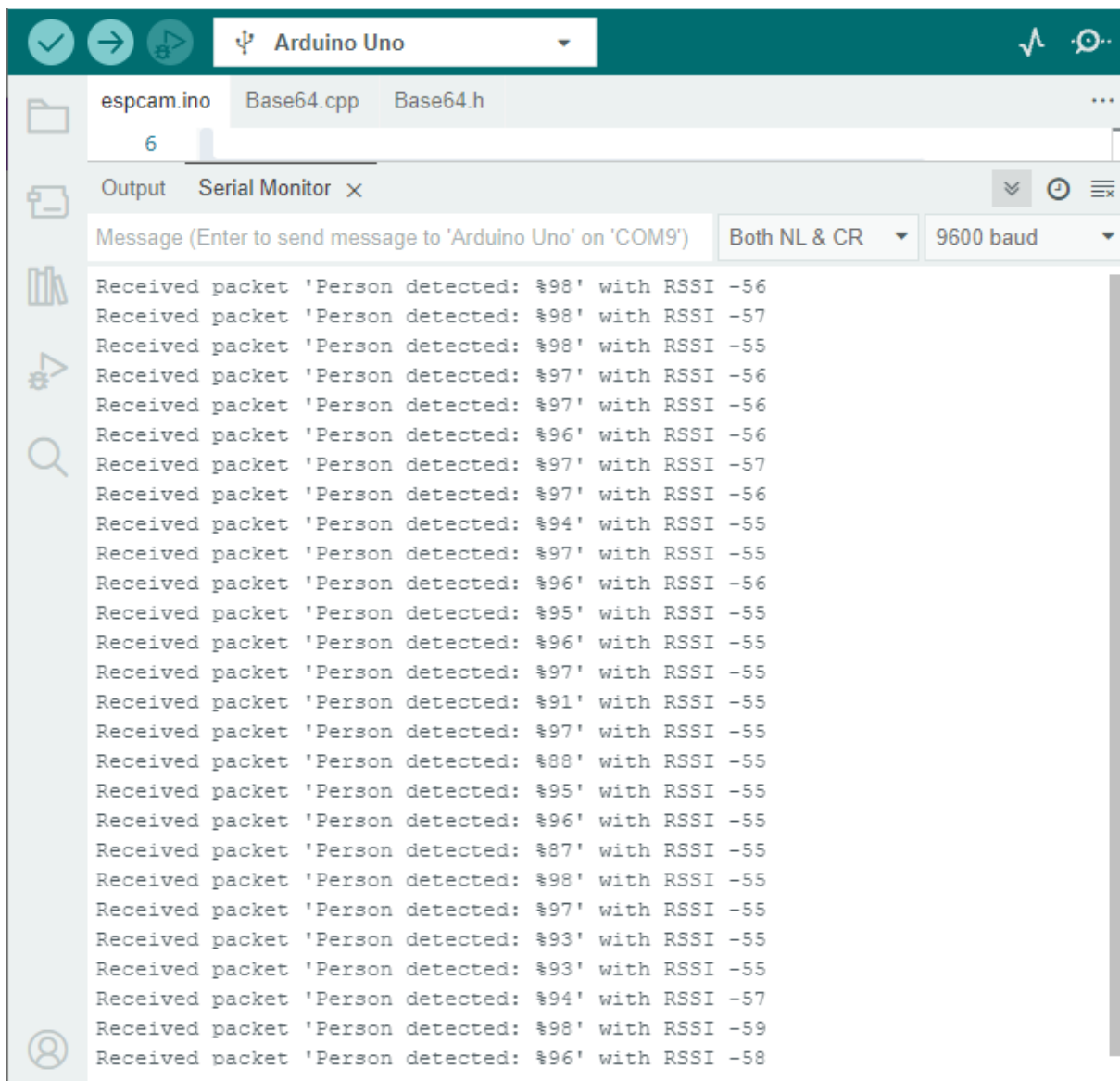


Figura 3.6: Mensaje recibido por el transceptor receptor visto desde la consola.

Para realizar la integración del transceptor con el módulo Arduino UNO se siguieron las siguientes conexiones entre los pines de los correspondientes dispositivos (Fig. 3.7):

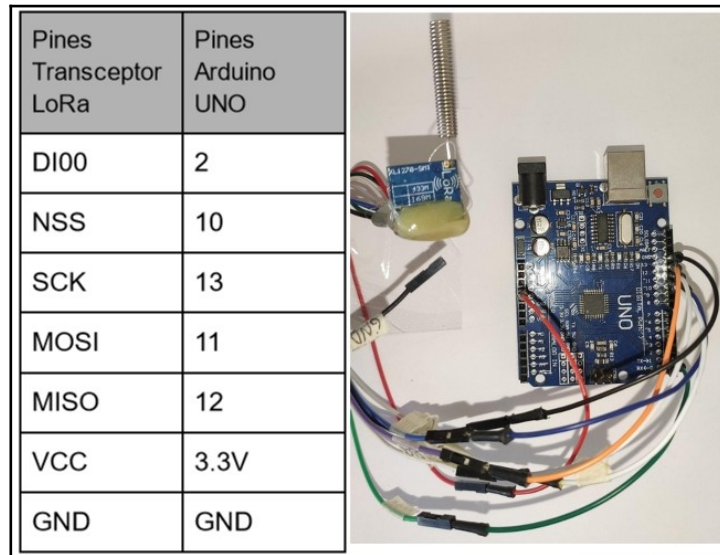


Figura 3.7: Conexiones de los puertos del transceptor con el microcontrolador Arduino UNO.

3.1.3 Módulo de suministro energético

Este módulo se basa en la utilización de energía solar para su alimentación. El mismo está conformado por tres componentes principales:

- Panel solar: Este dispositivo aprovecha la energía del sol y la convierte en electricidad para su uso.
- Batería: Permite almacenar la energía generada por el panel solar para utilizarla en momentos en los que este no esté generando energía suficiente, como en períodos nocturnos o días nublados.
- Módulo de Carga: Este dispositivo se encarga de recibir la energía del panel solar y cargar la batería para asegurar un suministro constante de energía. Además de asegurar que en momentos en los que el panel solar no esté generando energía, el mismo no sea dañado por la energía almacenada en la batería.

En la figura 3.8 se puede visualizar las conexiones realizadas entre los tres componentes y la conexión con el ESP32-CAM.

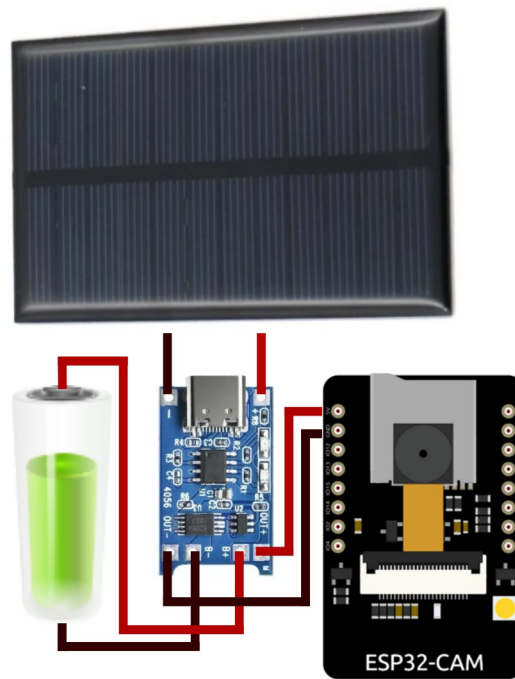


Figura 3.8: Diagrama con las conexiones del módulo de suministro energético.

3.1.4 Descripción de costos de construcción

En esta sección se detallan los costos asociados a los componentes que conforman la solución propuesta. Los valores expresados están en dólares y reflejan los precios vigentes al momento de la redacción de este trabajo.

- Placa de desarrollo “ESP32-CAM”: U\$S 10
- Transceptor Lora Sx1278: U\$S 10
- Panel solar: U\$S 8
- Módulo de carga de batería modelo TC4056A: U\$S 1.50
- Bateria “Samsung” modelo ICR18650-26J: U\$S 9.50
- Carcasa: U\$S 1.30

Costo total de construcción: U\$S 40.30

Los precios de referencia fueron obtenidos del sitio de comercio electrónico Mercado Libre Argentina (www.mercadolibre.com.ar) y convertidos a dólares según el tipo de cambio oficial del día de cotización.

3.2 Modelo de detección de personas

El desarrollo del modelo de detección de personas se dividió en dos etapas fundamentales: adquisición y preprocesamiento de datos, diseño de la arquitectura del modelo y su estrategia de entrenamiento. A continuación, se ofrece un desglose detallado de cada una de estas fases.

3.2.1 Adquisición y preprocesamiento de datos

3.2.1.1 Datasets

Los datos utilizados para la realización de este proyecto tienen dos fuentes principales: el dataset Visual Wake Words (123,287 imágenes) que está basado en el dataset público COCO, y las imágenes que se recolectaron utilizando el módulo ESP32-CAM (500 imágenes).

Adquisición de datos del dataset Visual Wake Words

En esta instancia se utilizó código del repositorio público de Tensorflow llamado "*Tensorflow Model Garden*" (Tensorflow Model Garden, sin fecha) en el cual se comparten diferentes implementaciones de modelos y soluciones de modelado de última generación para que puedan ser utilizados por los usuarios de Tensorflow.

Particularmente se utilizó el archivo "download_and_convert_visualwakewords.py" el cual realiza las siguientes acciones:

- Descarga los datasets de las imágenes y anotaciones del repositorio de "images.cocodataset.org". Los archivos descargados son:
 - train2014.zip: contiene las imágenes de entrenamiento.
 - val2014.zip: contiene las imágenes de validación.
 - annotations_trainval2014.zip: contiene las anotaciones referente a las imágenes descargadas.
- Descomprime los archivos descargados.
- Crea el archivo 'labels.txt' con las clases de interés. En este caso particular son 'person' (cuando hay al menos una persona en la imagen) y 'background' (cuando no hay personas en la imagen).
- Usa la anotaciones descargadas de COCO y genera unas nuevas anotaciones basados en la lógica de Visual Wake Words. Para crear estas anotaciones, el código recorre cada imagen y sus cuadros delimitadores, y si

una imagen tiene al menos un cuadro delimitador etiquetado como "persona" con un área mayor al 0,5% del área de la imagen, entonces la imagen se etiqueta como "persona", en caso contrario se etiqueta como "no persona". Cabe destacar que el valor 0.5% del área es un parámetro de la función, por lo que podría modificarse de ser necesario.

- Usa las anotaciones creadas para crear el dataset en formato TFRecords

Colecta de imágenes utilizando ESP32-CAM

Se efectuó una recopilación de imágenes utilizando la placa de desarrollo ESP32-CAM, con el propósito de integrarlas en el proceso de entrenamiento y validación de los modelos.

Esta recopilación se realizó en entornos que replicaban las condiciones de zonas rurales, caracterizadas por amplios espacios y cierta vegetación (Fig. 3.9).



Figura 3.9: Ejemplos de imágenes recolectadas por ESP32-CAM.

Para llevar a cabo esta colecta se desarrolló un módulo en el cual el ESP32-CAM toma las imágenes y las envía a una carpeta compartida de Google Drive. Para poder realizar dichos envíos, el ESP32-CAM se conecta a internet a través de una conexión WiFi compartida por un teléfono celular. En la figura 3.10 se puede visualizar un diagrama explicando su funcionamiento.

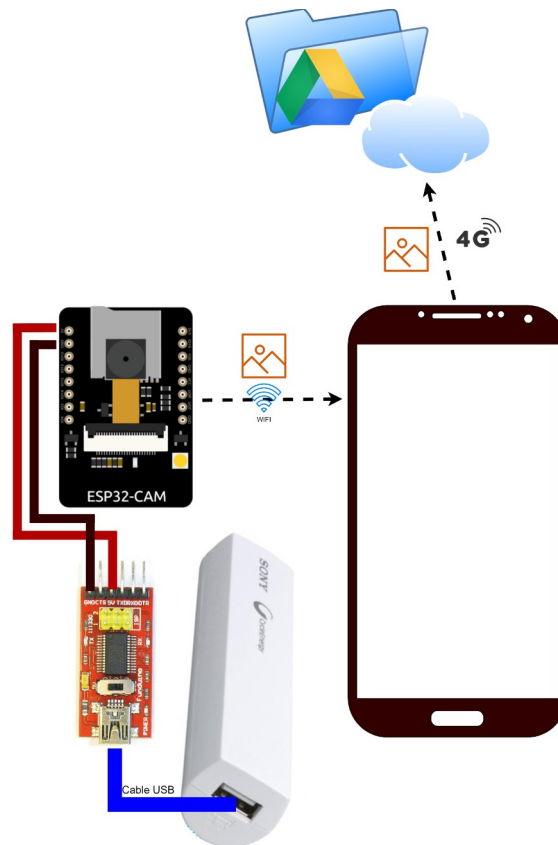


Figura 3.10: Diagrama de funcionamiento del módulo de colecta de imágenes.

El módulo ESP32-CAM es alimentado por un cargador portátil 2000 mAh, lo cual permite operar al módulo por un tiempo considerable.

Para la captura y almacenamiento de imágenes el módulo ESP32 CAM inicializa la cámara y se conecta a una red wifi provista por un celular. Luego se captura la imagen y se envía a una carpeta en el servicio de almacenamiento google Drive usando la API disponible en el servicio App Script de Google.

A pesar de disponer de esta solución, el proceso de recolección resultó ser considerablemente lento. Esto se debió, en gran medida, a la baja velocidad de transmisión de datos del operador de telefonía celular, lo que retrasó significativamente el envío de las imágenes a la carpeta compartida en Drive. Además, el módulo utilizado presentaba una velocidad de procesamiento limitada, lo que agravó el retraso general. Como consecuencia, sólo fue posible recolectar 500 imágenes.

Distribución del Dataset

Las 123,287 imágenes del dataset Visual Wake Words fueron distribuidas de la siguiente manera:

- Train: 82783 imágenes (67%)
 - Con Personas: 39926 imágenes

- Sin Personas: 42857 imágenes
- Validación: 32408 imágenes (26%)
 - Con Personas: 15340 imágenes
 - Sin Personas: 17068 imágenes
- Test: 8096 imágenes (7%)
 - Con Personas: 3773 imágenes
 - Sin Personas: 4323 imágenes

Por otro lado, las 500 imágenes recolectadas se añadieron al conjunto anterior con la siguiente distribución:

- Train: 336 imágenes (67%)
 - Con Personas: 168 imágenes
 - Sin Personas: 168 imágenes
- Validación: 131 imágenes (26%)
 - Con Personas: 65 imágenes
 - Sin Personas: 66 imágenes
- Test: 33 imágenes (7%)
 - Con Personas: 16 imágenes
 - Sin Personas: 17 imágenes

El dataset final quedó distribuido de la siguiente manera:

- Train: 83119 imágenes (67%)
 - Con Personas: 40094 imágenes
 - Sin Personas: 43025 imágenes
- Validación: 32539 imágenes (26%)
 - Con Personas: 15405 imágenes
 - Sin Personas: 17134 imágenes
- Test: 8129 imágenes (7%)
 - Con Personas: 3789 imágenes
 - Sin Personas: 4340 imágenes

3.2.1.2 Preprocesamiento de datos

Al conjunto de imágenes obtenido en la etapa anterior se le aplicaron las siguientes transformaciones: reducción, cambio de formato RGB a escala de grises, “data augmentation” y cambio de escala. A continuación se detallan los aspectos más importantes de cada una de estas etapas.

Reducción de imágenes

En una primera etapa, todas las imágenes se redimensionan a 96x96 píxeles. Si bien esta reducción de tamaño podría haberse realizado más tarde, durante el

proceso de entrenamiento, se optó por hacerlo previamente debido al incremento en los tiempos de procesamiento que habría supuesto de lo contrario.

Cabe destacar, que las imágenes del dataset original no tienen el mismo tamaño, generalmente alrededor de 640x480 píxeles. Esto significa que cada canal de la imagen consta de más o menos 307,200 píxeles. La transformación a 96x96 reduce esta cantidad a 9,216 píxeles, es decir, un tamaño aproximadamente 33 veces menor.

Para realizar esta transformación se utilizó la librería PIL de python, en el cual se aplicó la técnica de redimensionamiento basada en interpolación con uso de filtros LANCZOS.

Cambio de formato RGB a escala de grises

Dado que la configuración definida del módulo ESP32-CAM implica la captura de imágenes en escala de grises, el entrenamiento del modelo debe realizarse con imágenes en este formato. Por ello, se procedió a convertir las imágenes del conjunto de datos de RGB (formato original) a escala de grises (Fig. 3.11). Esta conversión redujo aún más el tamaño de las imágenes, pasando de 27.648 píxeles ($96 \times 96 \times 3$) por imagen a 9.216 píxeles, lo que representa un tamaño tres veces menor.

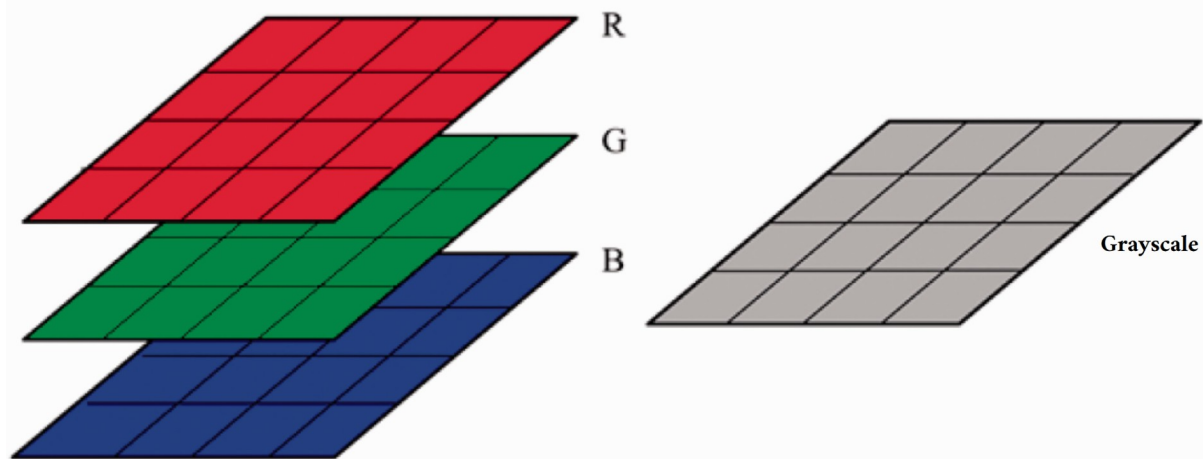


Figura 3.11: Conversión de imagen RGB a escala de grises.

Este proceso de conversión se realizó utilizando una función de la librería Tensorflow.

Data Augmentation

Con el propósito de mejorar el rendimiento de los modelos, se implementó la técnica de "data augmentation" para generar nuevas imágenes a partir de las existentes en

el conjunto de entrenamiento (Fig. 3.12). Estas imágenes generadas incorporaron dos modificaciones aleatorias, inversión horizontal de las imágenes y rotación de la imagen en un margen de hasta un 20%, que se realizaron utilizando funciones de la librería Tensorflow.

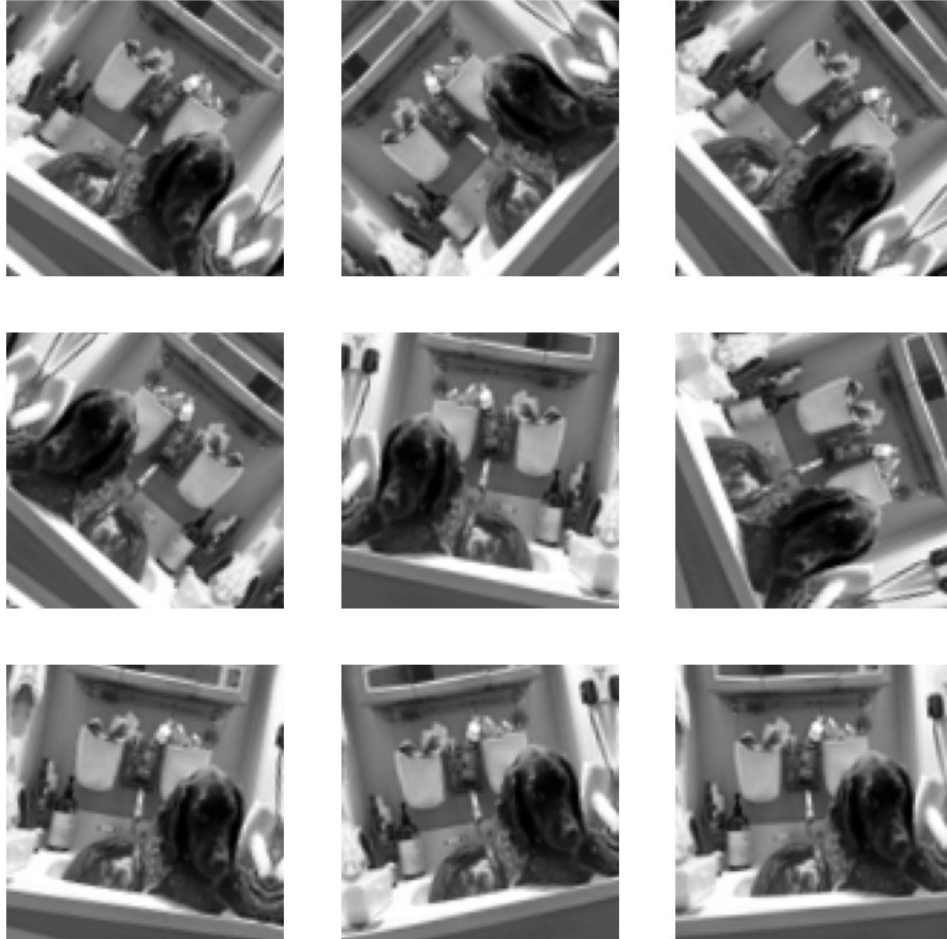


Figura 3.12: Ejemplos de data augmentation aplicado al dataset. Se muestran imágenes con inversión horizontal y rotación en un margen de hasta un 20%.

Cambio de escala de valores de los píxeles de las imágenes

En esta etapa, se procedió a cambiar la escala de los valores que cada píxel puede contener. Mientras que las imágenes originales tienen valores en un rango de 0 a 255, se realizó una transformación para que los valores estén en el intervalo de -1 a 1.

Comparación visual del preprocesamiento aplicado

La Figura 3.13 muestra comparativamente una imagen antes y después del preprocesamiento, ilustrando claramente las transformaciones aplicadas de redimensionamiento y conversión a escala de grises.

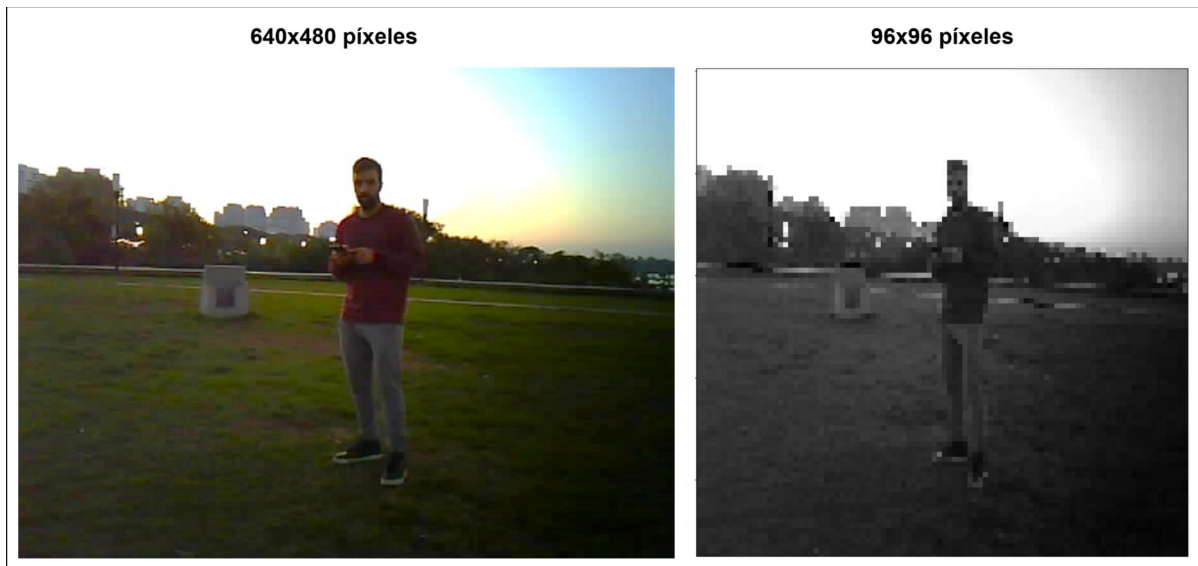


Figura 3.13: Ejemplo del antes y después del preprocesamiento aplicado a una imagen.

3.2.2 Diseño de la arquitectura

En la etapa de diseño de la arquitectura del modelo de clasificación, se optó por aplicar la técnica de transferencia de aprendizaje. Esto implicó aprovechar modelos pre-entrenados y ajustarlos para adaptarlos a los requisitos de este proyecto. En este contexto, se decidió utilizar el modelo de *computer vision* pre-entrenado MobileNet, desarrollado por Google. Este modelo se entrenó con el conjunto de datos públicos de ImageNet. Para este trabajo se utilizó la primera versión de MobileNet ya que es la de tamaño más pequeño.

En consecuencia, el modelo de redes neuronales consta de 3 secciones de capas principales (Fig. 3.14):

- Capas de preprocesamiento
- Capas del modelo pre-entrenado MobileNet V1
- Capas de clasificación

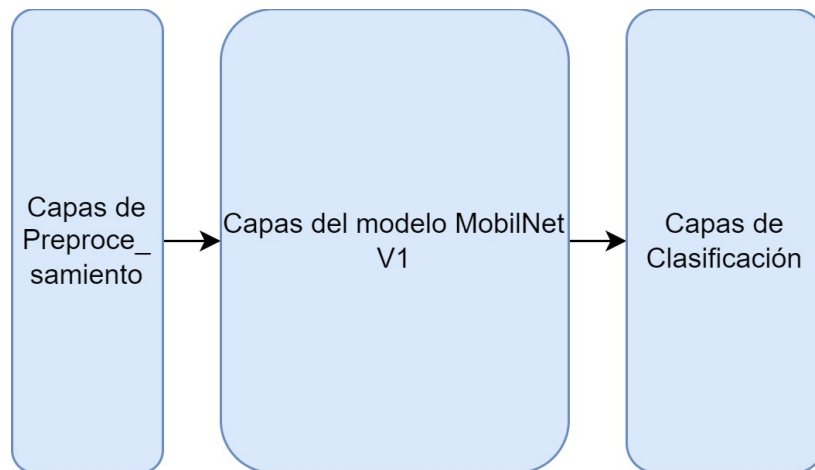


Figura 3.14: Detalle de las secciones principales del modelo implementado.

3.2.2.1 Configuración del modelo MobileNet

El principal desafío de este trabajo radica en la implementación de un modelo de *machine learning* en un dispositivo con recursos extremadamente limitados. El ESP32-CAM dispone de apenas 520 KB de memoria RAM, espacio que debe compartirse entre la ejecución de la aplicación, la carga del modelo y el almacenamiento temporal de las imágenes capturadas para su procesamiento. Para poner esto en perspectiva, el límite práctico para desplegar modelos en este entorno es de aproximadamente 360 KB. Por lo que intentar cargar un modelo de, por ejemplo, 400 KB, provoca fallos debido a desbordamiento de memoria.

Para adaptar MobileNet a estas restricciones, se procedió a reducir el ancho de su red mediante un factor $\alpha = 0.25$. Este parámetro α se conoce también como multiplicador de ancho de la red y controla el número de filtros en cada capa de la red. El valor por defecto de α es 1.0, y valores menores implican un número proporcionalmente menor de filtros en cada capa. Este ajuste reduce la red a su configuración más compacta, disminuyendo su tamaño de 3.2 millones de parámetros a tan solo 0.22 millones, es decir, a cerca de 1/14 de su tamaño original.

Si bien MobileNet fue diseñado específicamente para operar en dispositivos con recursos limitados, la configuración con $\alpha = 0.25$ representa el límite inferior de su escalabilidad. Esta reducción extrema del modelo conlleva inevitablemente una pérdida en capacidad predictiva, como contrapartida al ahorro en memoria.

En la tabla 3.1 se listan los detalles de las capas del modelo pre-entrenado Mobilenet.

Layer n°	Layer (type)	Output Shape	Param #
1	input_3 (InputLayer)	(None, 96, 96, 3)	0
2	conv1 (Conv2D)	(None, 48, 48, 8)	216
3	conv1_bn (BatchNormalization)	(None, 48, 48, 8)	32
4	conv1_relu (ReLU)	(None, 48, 48, 8)	0
5	conv_dw_1 (DepthwiseConv2D)	(None, 48, 48, 8)	72
6	conv_dw_1_bn (BatchNormalization)	(None, 48, 48, 8)	32
7	conv_dw_1_relu (ReLU)	(None, 48, 48, 8)	0
8	conv_pw_1 (Conv2D)	(None, 48, 48, 16)	128
9	conv_pw_1_bn (BatchNormalization)	(None, 48, 48, 16)	64
10	conv_pw_1_relu (ReLU)	(None, 48, 48, 16)	0
11	conv_pad_2 (ZeroPadding2D)	(None, 49, 49, 16)	0
12	conv_dw_2 (DepthwiseConv2D)	(None, 24, 24, 16)	144
13	conv_dw_2_bn (BatchNormalization)	(None, 24, 24, 16)	64
14	conv_dw_2_relu (ReLU)	(None, 24, 24, 16)	0
15	conv_pw_2 (Conv2D)	(None, 24, 24, 32)	512
16	conv_pw_2_bn (BatchNormalization)	(None, 24, 24, 32)	128
17	conv_pw_2_relu (ReLU)	(None, 24, 24, 32)	0
18	conv_dw_3 (DepthwiseConv2D)	(None, 24, 24, 32)	288
19	conv_dw_3_bn (BatchNormalization)	(None, 24, 24, 32)	128
20	conv_dw_3_relu (ReLU)	(None, 24, 24, 32)	0
21	conv_pw_3 (Conv2D)	(None, 24, 24, 32)	1024
22	conv_pw_3_bn (BatchNormalization)	(None, 24, 24, 32)	128
23	conv_pw_3_relu (ReLU)	(None, 24, 24, 32)	0
24	conv_pad_4 (ZeroPadding2D)	(None, 25, 25, 32)	0
25	conv_dw_4 (DepthwiseConv2D)	(None, 12, 12, 32)	288
26	conv_dw_4_bn (BatchNormalization)	(None, 12, 12, 32)	128
27	conv_dw_4_relu (ReLU)	(None, 12, 12, 32)	0
28	conv_pw_4 (Conv2D)	(None, 12, 12, 64)	2048
29	conv_pw_4_bn (BatchNormalization)	(None, 12, 12, 64)	256
30	conv_pw_4_relu (ReLU)	(None, 12, 12, 64)	0
31	conv_dw_5 (DepthwiseConv2D)	(None, 12, 12, 64)	576
32	conv_dw_5_bn (BatchNormalization)	(None, 12, 12, 64)	256
33	conv_dw_5_relu (ReLU)	(None, 12, 12, 64)	0
34	conv_pw_5 (Conv2D)	(None, 12, 12, 64)	4096
35	conv_pw_5_bn (BatchNormalization)	(None, 12, 12, 64)	256
36	conv_pw_5_relu (ReLU)	(None, 12, 12, 64)	0
37	conv_pad_6 (ZeroPadding2D)	(None, 13, 13, 64)	0
38	conv_dw_6 (DepthwiseConv2D)	(None, 6, 6, 64)	576
39	conv_dw_6_bn (BatchNormalization)	(None, 6, 6, 64)	256
40	conv_dw_6_relu (ReLU)	(None, 6, 6, 64)	0
41	conv_pw_6 (Conv2D)	(None, 6, 6, 128)	8192
42	conv_pw_6_bn (BatchNormalization)	(None, 6, 6, 128)	512
43	conv_pw_6_relu (ReLU)	(None, 6, 6, 128)	0
44	conv_dw_7 (DepthwiseConv2D)	(None, 6, 6, 128)	1152
45	conv_dw_7_bn (BatchNormalization)	(None, 6, 6, 128)	512
46	conv_dw_7_relu (ReLU)	(None, 6, 6, 128)	0
47	conv_pw_7 (Conv2D)	(None, 6, 6, 128)	16384
48	conv_pw_7_bn (BatchNormalization)	(None, 6, 6, 128)	512
49	conv_pw_7_relu (ReLU)	(None, 6, 6, 128)	0
50	conv_dw_8 (DepthwiseConv2D)	(None, 6, 6, 128)	1152
51	conv_dw_8_bn (BatchNormalization)	(None, 6, 6, 128)	512
52	conv_dw_8_relu (ReLU)	(None, 6, 6, 128)	0
53	conv_pw_8 (Conv2D)	(None, 6, 6, 128)	16384
54	conv_pw_8_bn (BatchNormalization)	(None, 6, 6, 128)	512
55	conv_pw_8_relu (ReLU)	(None, 6, 6, 128)	0
56	conv_dw_9 (DepthwiseConv2D)	(None, 6, 6, 128)	1152
57	conv_dw_9_bn (BatchNormalization)	(None, 6, 6, 128)	512
58	conv_dw_9_relu (ReLU)	(None, 6, 6, 128)	0
59	conv_pw_9 (Conv2D)	(None, 6, 6, 128)	16384
60	conv_pw_9_bn (BatchNormalization)	(None, 6, 6, 128)	512
61	conv_pw_9_relu (ReLU)	(None, 6, 6, 128)	0
62	conv_dw_10 (DepthwiseConv2D)	(None, 6, 6, 128)	1152
63	conv_dw_10_bn (BatchNormalization)	(None, 6, 6, 128)	512
64	conv_dw_10_relu (ReLU)	(None, 6, 6, 128)	0
65	conv_pw_10 (Conv2D)	(None, 6, 6, 128)	16384
66	conv_pw_10_bn (BatchNormalization)	(None, 6, 6, 128)	512
67	conv_pw_10_relu (ReLU)	(None, 6, 6, 128)	0
68	conv_dw_11 (DepthwiseConv2D)	(None, 6, 6, 128)	1152
69	conv_dw_11_bn (BatchNormalization)	(None, 6, 6, 128)	512
70	conv_dw_11_relu (ReLU)	(None, 6, 6, 128)	0
71	conv_pw_11 (Conv2D)	(None, 6, 6, 128)	16384
72	conv_pw_11_bn (BatchNormalization)	(None, 6, 6, 128)	512
73	conv_pw_11_relu (ReLU)	(None, 6, 6, 128)	0
74	conv_pad_12 (ZeroPadding2D)	(None, 7, 7, 128)	0
75	conv_dw_12 (DepthwiseConv2D)	(None, 3, 3, 128)	1152
76	conv_dw_12_bn (BatchNormalization)	(None, 3, 3, 128)	512
77	conv_dw_12_relu (ReLU)	(None, 3, 3, 128)	0
78	conv_pw_12 (Conv2D)	(None, 3, 3, 256)	32768
79	conv_pw_12_bn (BatchNormalization)	(None, 3, 3, 256)	1024
80	conv_pw_12_relu (ReLU)	(None, 3, 3, 256)	0
81	conv_dw_13 (DepthwiseConv2D)	(None, 3, 3, 256)	2304
82	conv_dw_13_bn (BatchNormalization)	(None, 3, 3, 256)	1024
83	conv_dw_13_relu (ReLU)	(None, 3, 3, 256)	0
84	conv_pw_13 (Conv2D)	(None, 3, 3, 256)	65536
85	conv_pw_13_bn (BatchNormalization)	(None, 3, 3, 256)	1024
86	conv_pw_13_relu (ReLU)	(None, 3, 3, 256)	0
Total of parameters			218544

Tabla 3.1: Detalle de las capas del modelo MobileNet implementado.

3.2.2.2 Capas de Preprocesamiento

En lo que respecta a la sección de preprocesamiento, la misma contempla las siguientes capas:

- *Data Augmentation*. Esta capa solo se activa en la etapa de entrenamiento, generando nuevas imágenes en base a la original con los cambios configurados (inversión y rotación aleatoria de la imagen).
- Transformar formato de entrada de 1 canal a 3 canales. Como el modelo pre-entrenado MobileNet recibe como entrada imágenes RGB (3 canales), y la entrada definida para este trabajo es en escala de grises (1 canal), esta capa replica los valores de la imagen en los otros 2 canales. De esta forma puede ser aceptado por el modelo MobileNet, sin la necesidad de modificar su arquitectura original.
- Cambio de escala de valores de los píxeles de las imágenes. Esta capa cambia los rangos de valores de $[0, 255]$ a $[-1, 1]$.

3.2.3 Exploración de arquitecturas y estrategias de entrenamiento

En este trabajo se presentan varias propuestas de modelos de clasificación que se distinguen principalmente por el diseño de la arquitectura de sus capas de clasificación y la estrategia de entrenamiento empleada. A continuación, se proporciona un desglose detallado de estos aspectos clave que varían de un modelo a otro.

Cambios en el diseño de las capas de clasificación

En este contexto, se realizaron variaciones en la cantidad de capas, la cantidad de neuronas y las funciones de activación presentes en cada capa del modelo. Estos ajustes permitieron explorar diferentes configuraciones y enfoques para la clasificación de imágenes (Fig. 3.15).

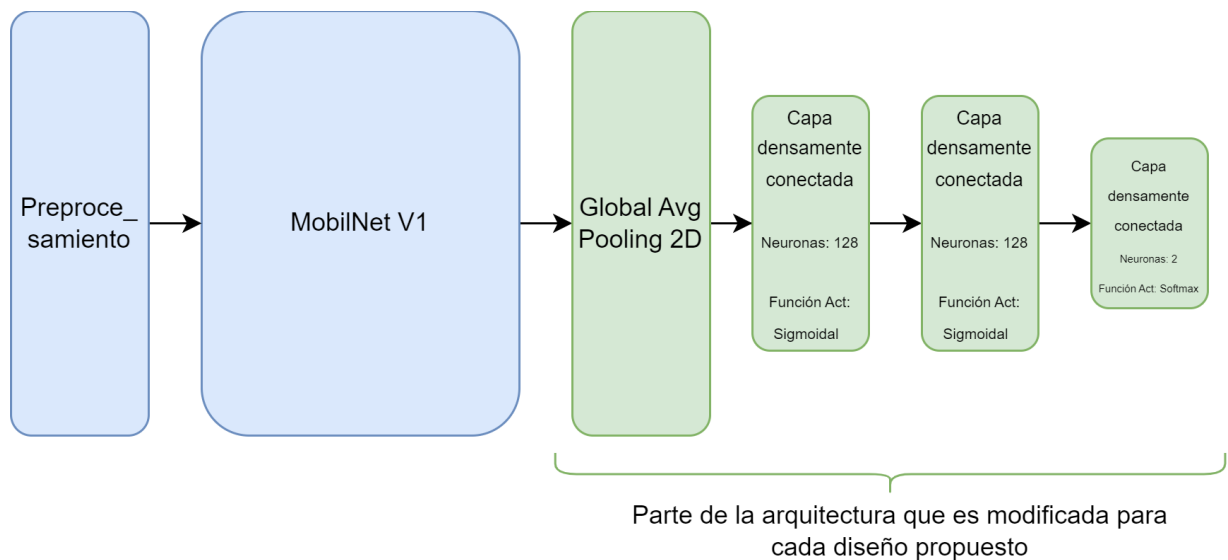


Figura 3.15: Sección de la arquitectura que es modificada para cada Clasificador propuesto.

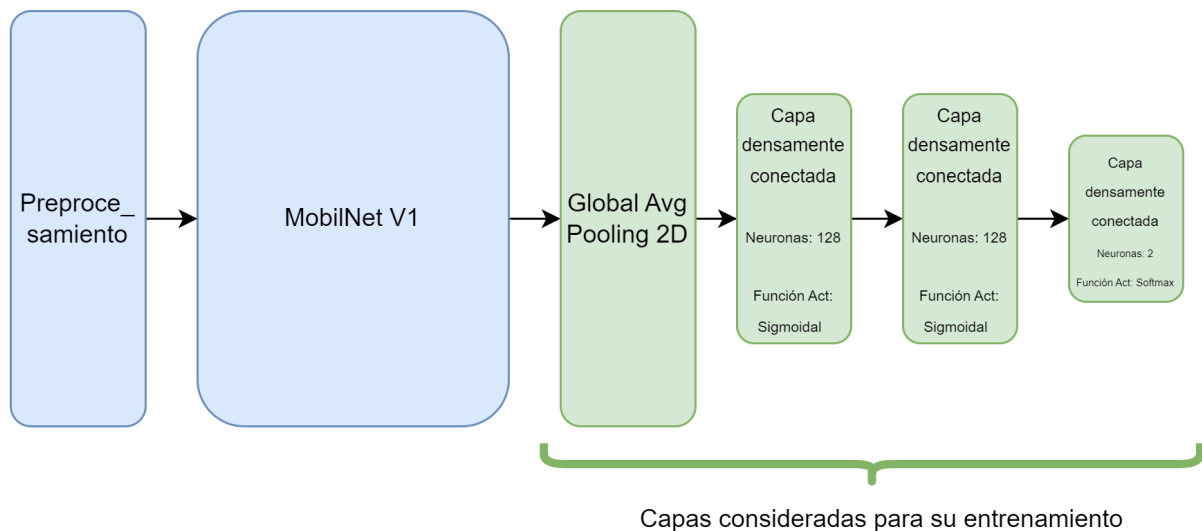
Cabe destacar que dado que el microcontrolador posee capacidades de almacenamiento limitadas, la definición de esta arquitectura se encontraba restringida por la capacidad del dispositivo. Por lo tanto, al añadir más capas y/o neuronas a esta sección de la arquitectura, se requería un mayor espacio para su despliegue en el microcontrolador.

Cantidad de capas entrenadas del modelo MobileNet

Otro aspecto que varía entre los clasificadores propuestos es la estrategia de entrenamiento en relación a la cantidad de capas del modelo base MobileNet que se entrenaron.

En este trabajo, se ha empleado una estrategia de entrenamiento que consta de dos etapas (Fig. 3.16). En la primera etapa, solo se entrenaron las capas de la sección de clasificación (posteriores al modelo base MobileNet). En la segunda etapa de entrenamiento, se añadieron las últimas capas del modelo base MobileNet al proceso de entrenamiento.

1° etapa de entrenamiento



2° etapa de entrenamiento

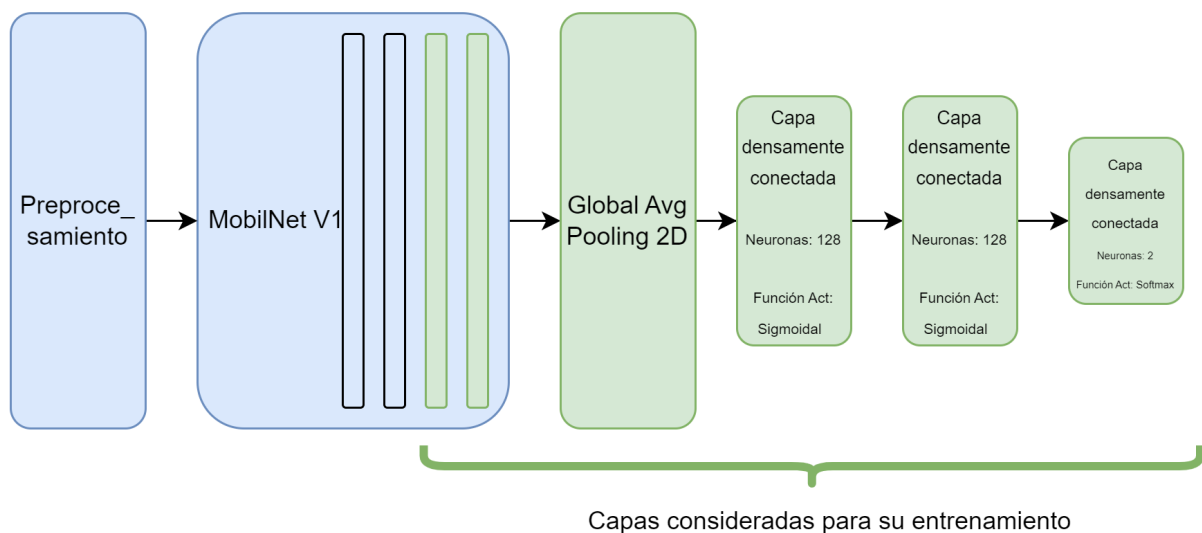


Figura 3.16: Etapas de entrenamiento de los modelos de Clasificación.

Es importante destacar que, para la segunda etapa, el parámetro de Tasa de Aprendizaje Inicial (Initial Learning Rate) se redujo significativamente. Esto se hizo con el fin de evitar cambios drásticos en los pesos de la red MobileNet que pudieran afectar negativamente el rendimiento del modelo durante el proceso de entrenamiento.

Parámetros de Optimización

Además de los factores previamente mencionados, se contemplaron modificaciones en los parámetros de optimización durante el proceso de entrenamiento de los modelos. Los parámetros de optimización desempeñan un papel esencial en la convergencia del modelo y su capacidad para aprender patrones en los datos de entrenamiento. Estos parámetros incluyeron la elección del algoritmo de

optimización, la tasa de aprendizaje, los momentos y otros hiperparámetros relevantes.

3.2.4 Clasificadores

Se diseñaron y evaluaron diferentes clasificadores, de los cuales se presentan en este trabajo los 9 clasificadores más representativos. Estos clasificadores se diferencian en su arquitectura y estrategia de entrenamiento de cada uno de ellos.

En cuanto a la arquitectura, en la figura 3.17 se aprecia que las variaciones de un diseño a otro radican en la cantidad de capas densamente conectadas en la sección de clasificación, en la cantidad de neuronas por capa y en la función de activación utilizada.

Todos los clasificadores contemplan una capa *Global Average Pooling 2D*. Esta capa calcula el promedio de los valores en cada canal de un mapa de características tridimensionales, generando un conjunto de valores unidimensionales que resumen la información clave de cada canal. Esto reduce la dimensionalidad de los datos, disminuyendo el número de parámetros en la red y promoviendo la generalización al enfocarse en características relevantes en lugar de detalles espaciales.

Además todos los clasificadores contienen una capa de regularización *DropOut*. Esta capa desactiva (pone su valor en cero) aleatoriamente un porcentaje de las neuronas, en este caso el 20%, durante el entrenamiento, lo que promueve la generalización al evitar que la red se ajuste en exceso a los datos de entrenamiento y al fomentar la diversidad en las combinaciones de características aprendidas. Esto mejora la capacidad de la red para funcionar bien en datos no vistos y previene el overfitting. Cabe mencionar que se realizó un experimento para evaluar si otros valores de *DropOut* podrían mejorar el rendimiento del modelo, sin embargo, no se observaron mejoras significativas. Los valores evaluados en dicho experimento fueron 10% y 5% (consultar el anexo D para más información).

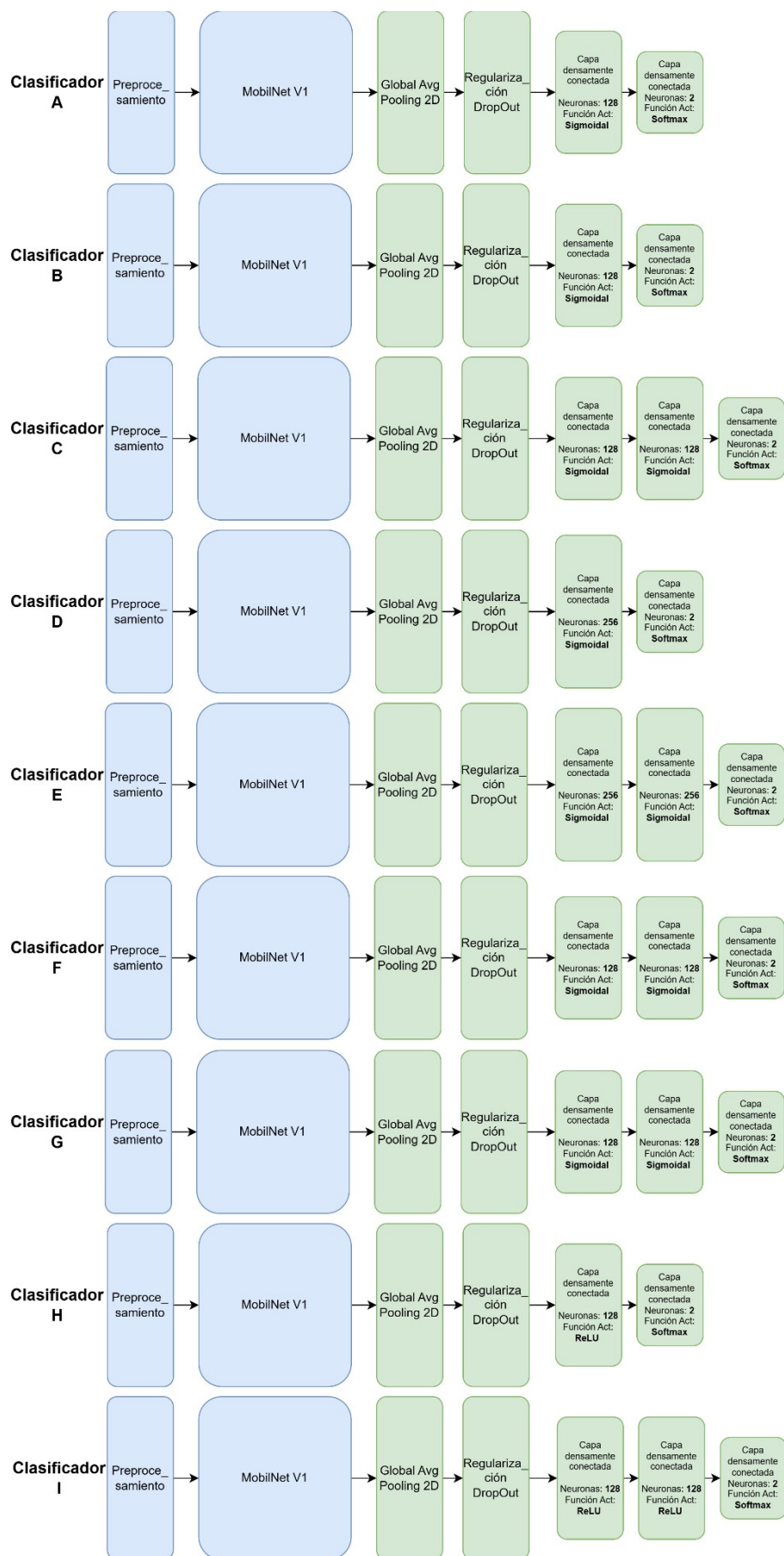


Figura 3.17: Diseños de la arquitectura de los clasificadores.

Como se visualiza en la Tabla 3.2, la estrategia de entrenamiento de cada uno de los clasificadores varían principalmente en la cantidad de capas consideradas para su entrenamiento en cada una de las dos etapas. Además, uno de los clasificadores se distingue por la función de optimización utilizada en la segunda etapa de entrenamiento.

Clasificador	1° Etapa de Entrenamiento						2° Etapa de Entrenamiento					
	Cant. de épocas	Cant. de capas a entrenar del modelo Mobile Net	Cant. de parámetros a entrenar	Parámetros de optimización			Cant. de épocas	Cant. de capas a entrenar del modelo Mobile Net	Cant. de parámetros a entrenar	Parámetros de optimización		
				Algoritmo de optimización	Tasa de aprendizaje inicial	Función de pérdida				Algoritmo de optimización	Tasa de aprendizaje inicial	Función de pérdida
A	60	0	33154	Adam	0.001	Sparse Categorical Cross Entropy	60	15	136962	RMSprop	0.0001	Sparse Categorical Cross Entropy
B	60	0	33154	Adam	0.001	Sparse Categorical Cross Entropy	60	15	136962	Adam	0.0001	Sparse Categorical Cross Entropy
C	60	0	49666	Adam	0.001	Sparse Categorical Cross Entropy	60	15	153474	Adam	0.0001	Sparse Categorical Cross Entropy
D	60	0	66306	Adam	0.001	Sparse Categorical Cross Entropy	60	15	170114	Adam	0.0001	Sparse Categorical Cross Entropy
E	60	0	132098	Adam	0.001	Sparse Categorical Cross Entropy	60	15	235906	Adam	0.0001	Sparse Categorical Cross Entropy
F	60	0	49666	Adam	0.001	Sparse Categorical Cross Entropy	60	30	206210	Adam	0.0001	Sparse Categorical Cross Entropy
G	60	0	49666	Adam	0.001	Sparse Categorical Cross Entropy	60	5	116226	Adam	0.0001	Sparse Categorical Cross Entropy
H	60	0	33154	Adam	0.001	Sparse Categorical Cross Entropy	60	30	189698	Adam	0.0001	Sparse Categorical Cross Entropy
I	60	0	49666	Adam	0.001	Sparse Categorical Cross Entropy	60	30	206210	Adam	0.0001	Sparse Categorical Cross Entropy

Tabla 3.2: *Parámetros de entrenamiento de los clasificadores.*

Respecto a la tasa de aprendizaje, no hubo diferencias entre los clasificadores. Se realizó un experimento para evaluar si ajustar esta tasa en la segunda etapa mejoraba el rendimiento, pero no se obtuvieron resultados significativos (consultar el anexo D para más información).

En el anexo B se muestran los detalles completos de cada clasificador.

3.2.5 Cuantización del modelo posterior al entrenamiento

Después de entrenar los diversos modelos, se aplicó la técnica de cuantización (Cuantización Post-Entrenamiento) con el propósito de reducir su tamaño y mejorar

la eficiencia computacional para su posterior despliegue en el microcontrolador. Esta fase se llevó a cabo utilizando la biblioteca TensorFlow Lite, que proporciona las funciones necesarias para aplicar la cuantización a los modelos.

La cuantización implementada transformó los pesos del modelo de números en punto flotante de 32 bits a números enteros de 8 bits, lo que redujo el tamaño del modelo en un factor de 4. Además, considerando que las operaciones aritméticas con números enteros son más rápidas que las operaciones con números en punto flotante, la cuantización mejoró el rendimiento del modelo durante la inferencia. En este trabajo se aplicó la técnica de cuantización de punto cero.

3.3 Análisis integrado

3.3.1 Análisis de performance y selección del modelo de detección de personas

Después de completar todos los entrenamientos de las configuraciones previamente mencionadas, esta sección presenta un análisis de los resultados obtenidos para cada una de las configuraciones. Finalmente, se revela el modelo elegido para su implementación en el microcontrolador, junto con los valores de performance tras aplicar la técnica de cuantización posterior al entrenamiento (*post-training quantization*).

3.3.1.1 Selección del modelo implementado

En la Figura 3.18 se presenta la curva comparativa de precisión-*recall* para los modelos evaluados, junto con métricas complementarias como el PR-AUC (área bajo la curva de precisión-*recall*) y el punto de máximo F1. Todos los indicadores de performance fueron calculados sobre los datos de test de los modelos entrenados. Para un análisis más detallado de cada clasificador, consultar el Anexo C.

Como puede observarse, los clasificadores F y H son los que presentan el mejor desempeño en términos de PR-AUC, con valores de 0.866 y 0.864, respectivamente. En cuanto al valor máximo de F1, el clasificador F alcanza el más alto (0.780), mientras que el clasificador H se posiciona en tercer lugar con 0.776.

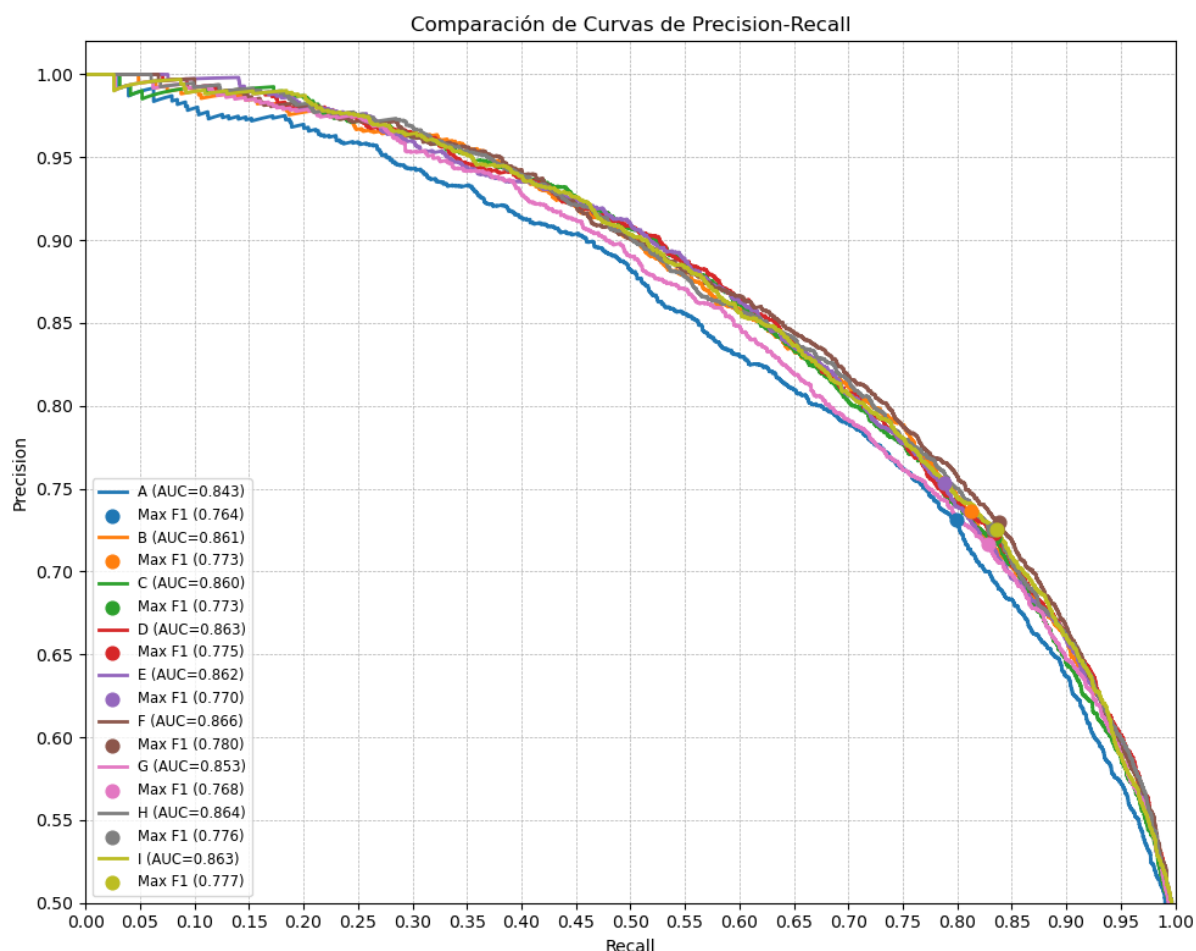


Figura 3.18: Comparación de curvas de Precisión-Recall de los clasificadores.

Por otro lado, en la Tabla 3.3 se muestran los indicadores de rendimiento para tres puntos de corte representativos (0.5, 0.75 y 0.9). En general, se observa que a medida que el umbral de decisión se incrementa, la precisión mejora considerablemente, aunque el *recall* disminuye de forma significativa.

Clasificador	Punto de corte = 0.5				Punto de corte = 0.75				Punto de corte = 0.9			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
A	76.69%	77.84%	70.73%	74.12%	72.52%	88.24%	48.20%	62.35%	65.71%	93.20%	29.49%	44.80%
B	77.73%	76.24%	76.45%	76.34%	75.66%	86.74%	56.99%	68.79%	69.30%	94.29%	36.99%	53.13%
C	77.55%	78.81%	71.84%	75.16%	74.14%	89.29%	51.55%	65.36%	67.06%	94.44%	32.29%	48.12%
D	78.10%	76.97%	76.60%	76.78%	75.42%	88.28%	55.35%	68.04%	68.10%	95.11%	34.29%	50.41%
E	78.15%	77.76%	75.34%	76.53%	75.25%	88.41%	54.81%	67.67%	68.08%	94.87%	34.31%	50.40%
F	77.79%	75.83%	77.56%	76.69%	75.90%	86.37%	57.99%	69.39%	69.36%	93.49%	37.58%	53.61%
G	77.31%	76.63%	74.79%	75.70%	74.31%	87.98%	52.81%	66.00%	66.91%	95.16%	31.53%	47.37%
H	78.24%	78.93%	73.74%	76.25%	74.63%	90.53%	51.86%	65.95%	67.05%	96.57%	31.55%	47.56%
I	77.88%	79.78%	70.33%	74.76%	73.67%	90.51%	48.55%	63.20%	66.25%	96.59%	28.56%	44.09%

Tabla 3.3: Indicadores de performance sobre datos de test.

Dado que los clasificadores F y H demostraron un rendimiento destacado tanto en PR-AUC como en el valor máximo de F1, se decidió realizar una tercera fase de entrenamiento para ambos, con el fin de explorar un posible margen de mejora. En esta nueva etapa, se añadieron 120 épocas adicionales de entrenamiento,

manteniendo las 30 capas finales de MobileNet como entrenables. Además, considerando que en experimentos previos se había observado un aumento del error al comenzar la segunda etapa, se optó por reducir la tasa de aprendizaje inicial de 0.0001 a 0.00001.

En la Figura 3.19 se presentan los resultados obtenidos tras esta fase adicional de entrenamiento. Ambos modelos muestran mejoras leves en los distintos indicadores, aunque el clasificador F mantiene su superioridad sobre el H en términos de PR-AUC y valor máximo de F1.

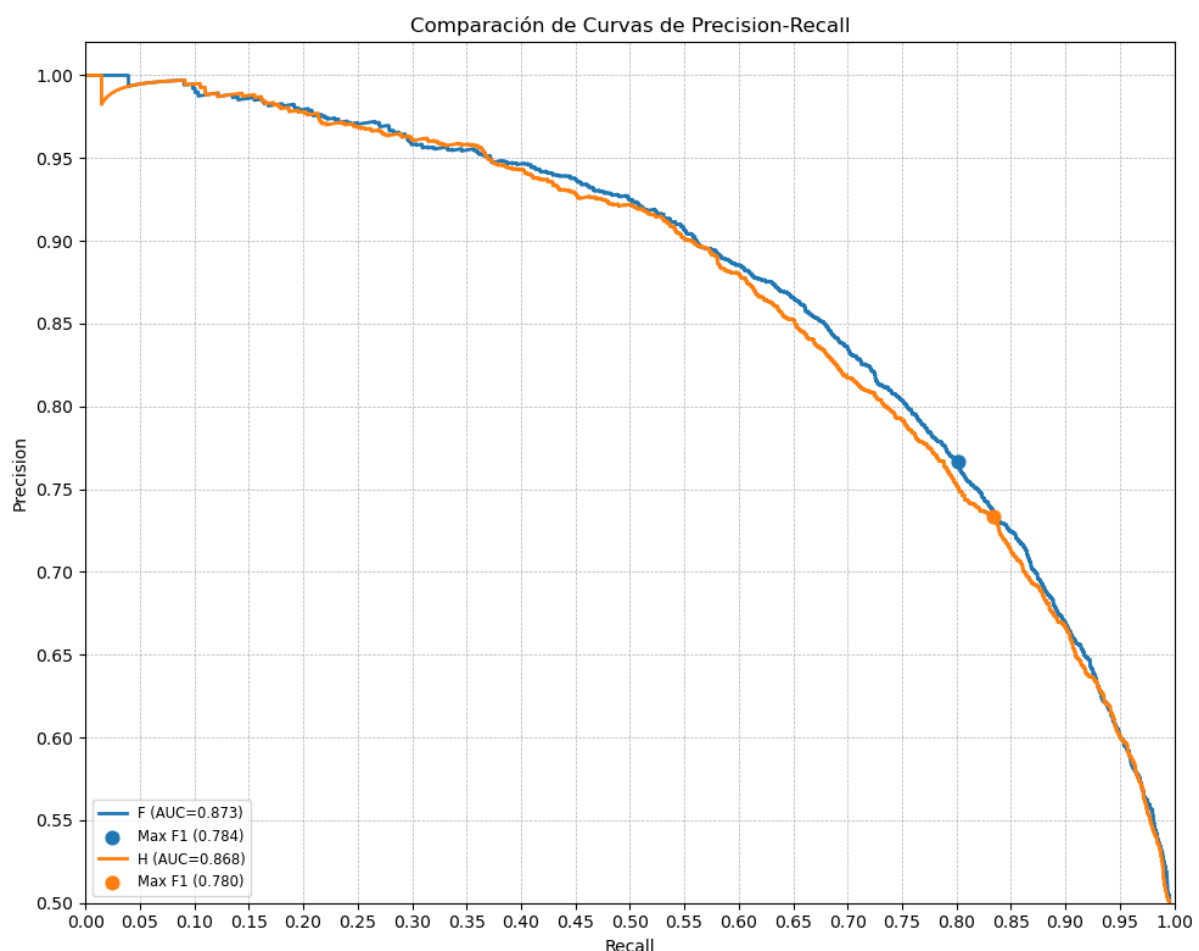


Figura 3.19: Comparación de curvas de Precisión-Recall de los clasificadores F y H después de la tercera etapa de entrenamiento.

La Tabla 3.4 muestra los indicadores de performance actualizados. Se observa una leve mejora en el *recall* para ambos clasificadores en el punto de corte 0.9. En particular, el clasificador F mejora también su precisión en ese punto. Además, el clasificador F logra mejores valores de F1 en los tres puntos de corte analizados, lo que indica un balance más favorable entre precisión y recall.

Clasificador	Punto de corte = 0.5				Punto de corte = 0.75				Punto de corte = 0.9			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
F	78.68%	79.45%	75.38%	77.36%	76.12%	89.25%	57.52%	69.95%	69.53%	93.81%	39.54%	55.64%
H	78.59%	81.36%	70.76%	75.69%	74.96%	91.50%	51.64%	66.02%	68.19%	96.26%	33.78%	50.01%

Tabla 3.4: Indicadores de performance sobre datos de test.

Considerando los resultados obtenidos, se concluye que el clasificador F presenta un rendimiento superior al clasificador H en términos de PR-AUC, valor máximo de F1 y valores de F1 en distintos puntos de corte. Este comportamiento indica un mejor equilibrio entre precisión y recall, lo cual es crucial en el contexto de implementación. Por lo tanto, se tomó la decisión de implementar el Clasificador F en el microcontrolador.

3.3.1.2 Análisis del modelo con la cuantización aplicada

Una vez seleccionado el modelo descrito en la etapa anterior, se procedió a aplicar la técnica de cuantización de punto cero sobre el modelo. Como se explicó anteriormente, esta transformación reduce el tamaño del modelo y aumenta la velocidad de inferencia. Pero como desventaja, reduce su precisión.

En esta sección analizamos la performance del modelo posterior a la implementación de esta técnica.

En la figura 3.20 puede visualizarse el umbral de discriminación para el Clasificador F con la cuantización aplicada.

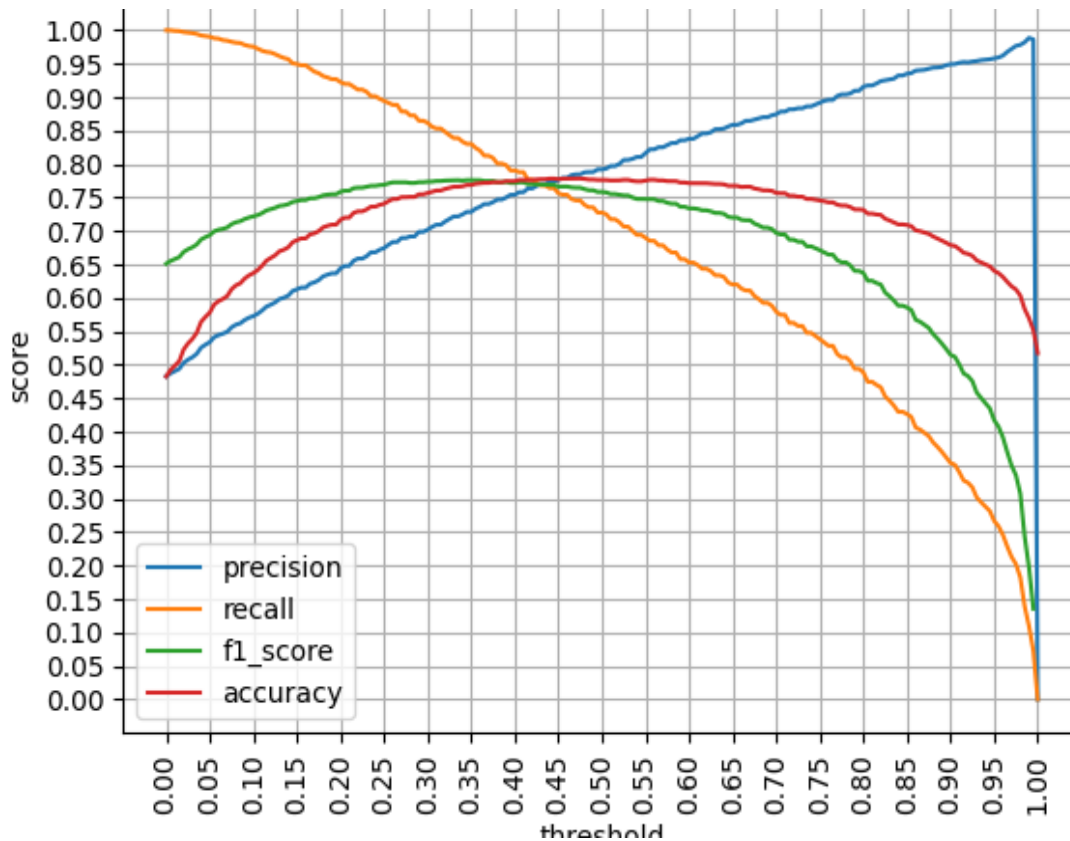


Figura 3.20: Umbral de discriminación Clasificador F con cuantización aplicada.

Como puede visualizarse en la tabla 3.5, la pérdida de performance del modelo fueron mínimas con respecto al modelo sin cuantizar. La mayor pérdida se dio en el *recall* para los puntos de cortes más altos.

Clasificador	Punto de corte = 0.5				Punto de corte = 0.75				Punto de corte = 0.9			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
F	78.68%	79.45%	75.38%	77.36%	76.12%	89.25%	57.52%	69.95%	69.53%	93.81%	39.54%	55.64%
F Cuantizado	77.64%	79.26%	72.74%	75.86%	74.56%	89.21%	53.71%	67.05%	67.82%	94.97%	35.24%	51.41%
Diferencia	-1.04%	-0.18%	-2.65%	-1.50%	-1.57%	-0.04%	-3.81%	-2.90%	-1.70%	1.15%	-4.30%	-4.23%

Tabla 3.5: Indicadores de performance sobre datos de test.

Para la definición del umbral utilizado en la lógica del microcontrolador, se priorizó la precisión, procurando al mismo tiempo no sacrificar en exceso el *recall*. En función de este criterio, se seleccionó un punto de corte de 0.53, el cual arrojó los siguientes valores de rendimiento sobre los datos de validación:

Punto de corte: 0.53

- **Precision:** 81%
- **Recall:** 70%
- **F1:** 75%
- **Accuracy:** 78%

Finalmente, se llevó a cabo un análisis de los tiempos de ejecución del módulo de detección de personas con la implementación de este modelo en el módulo ESP32-

CAM. El resultado de este análisis reveló que, en promedio, el módulo requiere aproximadamente 450 ms desde la captura de la imagen por la cámara hasta la generación de la predicción del modelo.

3.3.2 Análisis de performance del módulo de comunicación

Para evaluar el rendimiento del módulo de comunicación, se llevaron a cabo pruebas de alcance en diversas distancias. Estas pruebas se realizaron en una zona rural del departamento de General Alvear, en la provincia de Corrientes, Argentina. La elección de esta ubicación se basó en su topografía llana, lo que minimizó las interferencias debido a elevaciones en el terreno, y en la escasa presencia de vegetación que podría afectar significativamente la señal.

Tanto el transmisor como el receptor se colocaron a una altura de 2.5 metros sobre el suelo para reducir lo mayor posible las posibles interferencias de objetos o vegetación (Fig. 3.21).

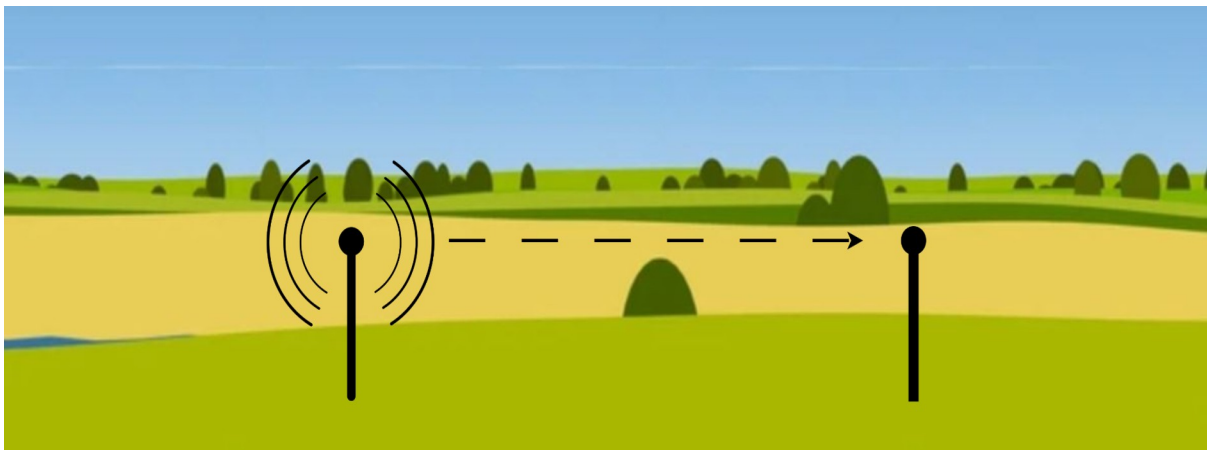


Figura 3.21: Esquema de prueba de distancias.

Se realizaron pruebas a diversas distancias. En la Tabla 3.6 se presentan los resultados obtenidos en estas evaluaciones.

N° de prueba	Coordenadas geográficas (Latitud y Longitud)		Distancia entre dispositivos (km)	Comunicación exitosa?
	Emisor	Receptor		
1	-28.843576205469994, -56.39139532339453	-28.852478849061047, -56.392430542422865	1	SI
2	-28.83461783824949, -56.39032873434346	-28.852478849061047, -56.392430542422865	2	SI
3	-28.825631216588295, -56.38932488582062	-28.852478849061047, -56.392430542422865	3	SI
4	-28.816726275861168, -56.38832103727759	-28.852478849061047, -56.392430542422865	4	NO
5	-28.807738109918457, -56.38725444824418	-28.852478849061047, -56.392430542422865	5	NO

Tabla 3.6: Resultados de pruebas de distancias.

Como se pudo observar en los resultados, este módulo es capaz de lograr una cobertura con un radio de al menos 3 kilómetros, un valor de gran relevancia. Esto es particularmente significativo al considerar un escenario en el cual el entorno alrededor de un dispositivo receptor presenta características geográficas similares al lugar de las pruebas realizadas. En tal caso, sería posible distribuir estos módulos de detección de personas en cualquier punto dentro de un área de cobertura de al menos 28.27 km² (aproximadamente 2827 hectáreas). (Fig. 3.22)

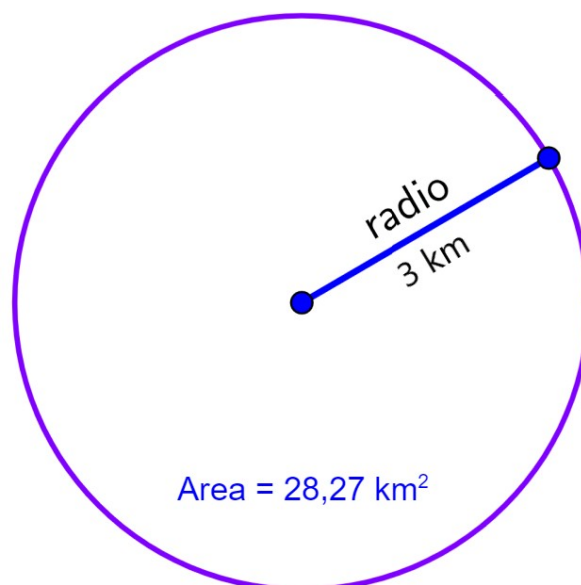


Figura 3.22: Área de cobertura.

3.3.3 Análisis de performance del módulo de suministro energético

Para evaluar el rendimiento del módulo de suministro de energía, se llevaron a cabo diversas pruebas. En primer lugar, se evaluó la duración máxima de la batería al anular el suministro de energía desde el panel solar. Además, se analizó el tiempo requerido para que el panel solar cargue completamente la batería mientras el módulo estaba en funcionamiento. Esta última prueba se realizó en un día despejado para minimizar la interferencia de las nubes y garantizar un rendimiento óptimo del panel solar.

Los resultados obtenidos son los siguientes:

- Duración máxima de la batería: 12 horas
- Tiempo de carga de la batería: 8 horas

Estos resultados nos permiten concluir que el módulo es capaz de funcionar de manera efectiva y su batería proporciona suficiente energía para operar en momentos donde no haya disponibilidad de luz solar. Esto le confiere autonomía y la capacidad de funcionar sin depender de fuentes de alimentación externas.

4. Conclusiones y trabajos futuros

4.1 Conclusiones

El objetivo principal de este trabajo consistió en desarrollar un módulo de detección de personas utilizando *Edge Computer Vision*, diseñado específicamente para su implementación en microcontroladores de bajo costo, con aplicaciones en zonas rurales donde el acceso a la red eléctrica y la infraestructura de comunicación son limitados o inexistentes. Para lograr estos objetivos se desarrolló una solución que consta de un componente de hardware, compuesta por tres módulos: detección de personas, comunicación y suministro energético. Y un módulo de procesamiento de datos para el que se implementaron diferentes metodologías de visión artificial empleando redes neuronales multicapas bajo restricciones severas de capacidad de memoria y procesamiento.

Para el módulo de detección de personas, se desarrolló un clasificador cuya arquitectura incluye tres secciones principales: una etapa de preprocesamiento, un modelo base MobileNet V1, y un bloque final de capas densamente conectadas para clasificación. Uno de los mayores desafíos fue la implementación del modelo en un entorno extremadamente limitado como el del microcontrolador ESP32-CAM, que cuenta con solo 520 KB de memoria RAM, espacio que debe ser compartido entre la aplicación, el modelo y el procesamiento de imágenes. El límite práctico de

uso para modelos en este entorno es de aproximadamente 360 KB, por lo que exceder este valor provoca fallos por desbordamiento de memoria.

Para adaptarse a las restricciones impuestas por el hardware, se implementó una versión reducida de MobileNet V1, utilizando un factor de ancho de red $\alpha = 0.25$. Esta optimización redujo la cantidad de parámetros de 3.2 millones a solo 0.22 millones, es decir, aproximadamente al 7 % del tamaño original, lo que inevitablemente implicó una pérdida de rendimiento respecto a la versión completa del modelo. Además, el procesamiento se realizó sobre imágenes en escala de grises de 96×96 píxeles, una representación considerablemente más limitada en comparación con las imágenes RGB de mayor resolución comúnmente utilizadas en conjuntos de datos como COCO (con resoluciones aproximadas de 640×420 píxeles).

El proceso de entrenamiento se estructuró en tres etapas. La primera etapa se centró exclusivamente en el entrenamiento de las capas situadas después del modelo base MobileNet. En las dos etapas posteriores, se incorporaron al entrenamiento las últimas 30 capas de MobileNet, lo que permitió una adaptación gradual del modelo al dominio específico de detección de personas. Finalmente, se implementó un proceso de cuantización post-entrenamiento que permitió optimizar tanto la eficiencia computacional durante la inferencia como la reducción del tamaño del modelo final.

Durante la exploración de distintos diseños, se observó que la incorporación de más capas del modelo MobileNet al proceso de entrenamiento producía mejoras en el rendimiento, aunque también requería un mayor número de épocas para estabilizar el error. Asimismo, se detectó que al iniciar la segunda etapa de entrenamiento (donde se incluyen capas de MobileNet), el error aumentaba en las primeras iteraciones. Esto llevó a ajustar la tasa de aprendizaje del algoritmo de optimización, disminuyéndola progresivamente en cada etapa para evitar dichos picos de error.

Por otro lado se observó que la arquitectura y cuantización implementada permitieron que el módulo procese, en promedio, una imagen cada 450 ms, lo que proporciona un tiempo de respuesta óptimo para aplicaciones en tiempo real. Estos resultados superan significativamente los obtenidos por Canepa et al., 2022, donde un modelo similar implementado en un microcontrolador STM32 requería 33 segundos por imagen.

El modelo final seleccionado, con un umbral de clasificación fijado en 0.53, alcanzó una precisión del 81% y un recall del 70%. Estos resultados se consideran satisfactorios, especialmente al compararlos con los obtenidos por Wulfert et al. (2022) en un experimento, en el que se utilizó una arquitectura similar basada en MobileNet V1 y dimensiones de entrada (96×96 píxeles). En ese caso, se reportó una precisión del 47% y un recall del 55%.

Con base en el análisis de rendimiento del módulo de suministro energético, se verificó que el módulo exhibe una autonomía y la capacidad de operar de manera autónoma, prescindiendo de fuentes de alimentación externas, lo cual se ajusta perfectamente a los requisitos de implementación deseados. Sin embargo, es importante tener en cuenta que el dispositivo propuesto carece de sistemas de visión nocturna o capacidad para operar en condiciones de baja luminosidad, lo que limita su utilidad en tales circunstancias.

En este trabajo no se realizó una comparación de performance directa con productos ofrecidos en el mercado Argentino que puedan tener una aplicación similar, como el caso presentado de la cámara “Hikvision” modelo “DS-2XS6A87G1-L/C32S80”. Sin embargo, al considerar que esta cámara opera con imágenes de mayor resolución (8MP) y ofrece visión nocturna, es plausible que ofrezca un rendimiento superior al de la solución propuesta. No obstante, desde una perspectiva económica, es importante tener en cuenta que el precio estimado de esta cámara es de U\$S 3300. En contextos donde se requiere cubrir áreas extensas, la implementación de la solución propuesta se vuelve altamente ventajosa, ya que el costo de una sola cámara equivale al precio de construir 81 de los módulos propuesto en este trabajo.

En cuanto al módulo de comunicación, se demostró su capacidad para proporcionar una cobertura efectiva con un radio mínimo de 3 kilómetros. Esta característica resulta crucial, especialmente en entornos geográficamente similares a los lugares de pruebas, permitiendo la dispersión estratégica de los módulos en cualquier punto dentro de una amplia área de cobertura. Esta flexibilidad y alcance amplio representan una ventaja significativa para la eficacia y versatilidad de la aplicación práctica del módulo de detección de intrusiones.

4.2 Trabajos futuros

En cuanto a posibles áreas de investigación futura, se vislumbran varios caminos para seguir explorando y mejorando el trabajo realizado. En primer lugar, dado el impacto positivo que tuvo a la performance la incorporación de más capas del modelo base MobileNet para su entrenamiento, se propone continuar en esta línea de investigación agregando aún más capas al proceso de entrenamiento. Esta estrategia podría profundizar aún más la capacidad del modelo para detectar y clasificar con mayor precisión los objetos de interés.

Además, se plantea la posibilidad de mejorar el enfoque actual mediante la inclusión de imágenes en formato RGB en lugar de escala de grises. Esta modificación podría enriquecer el conjunto de datos de entrenamiento y proporcionar al modelo una información visual más rica y detallada, lo que podría traducirse en una mejora en

su precisión.

Por otro lado, se considera fundamental la incorporación de un sistema de iluminación infrarroja para permitir el funcionamiento del dispositivo en condiciones de baja luminosidad. Esta adición sería especialmente relevante para entornos nocturnos o con poca luz, donde la detección precisa aún es crucial.

Por último, se destaca el potencial de la metodología desarrollada en este estudio para ser aplicada en otros contextos más allá de la seguridad. Por ejemplo, podría explorarse su utilidad en el control y monitoreo de diversas especies animales, abriendo nuevas perspectivas de investigación en áreas como la conservación y la ecología.

En resumen, estas sugerencias ofrecen diversas vías para continuar avanzando y mejorando la tecnología de detección de personas basada en *Edge Computer Vision* en microcontroladores de bajo costo, ampliando su alcance y utilidad en diferentes campos de aplicación.

5. Referencias bibliográficas

Activation function. Wikipedia. https://en.wikipedia.org/wiki/Activation_function

(verificado 31 oct. 2023)

Advantages of PIR sensor | disadvantages of PIR sensor. RF Wireless World.

<https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-PIR-sensor.html> (verificado 18 may. 2023)

Advantages of Microwave sensor | disadvantages of Microwave sensor. RF Wireless World. [https://www.rfwireless-world.com/Terminology/Advantages-and-](https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Microwave-sensor.html)

[Disadvantages-of-Microwave-sensor.html](https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Microwave-sensor.html) (verificado 18 may. 2023)

Ai-Thinker Technology. *ESP32-CAM camera development board*. Ai-Thinker

Technology. <https://docs.ai-thinker.com/en/esp32-cam> (verificado 26 feb. 2024)

Antoniadis, P. (2023, 19 de Junio). *How to Convert an RGB Image to a Grayscale*.

Baeldung. <https://www.baeldung.com/cs/convert-rgb-to-grayscale>

Ariño, N., Miazzo, D., & Pisani Claro, N. (2022). *Empleo en las cadenas agroindustriales*. Fundación Agropecuaria para el Desarrollo de Argentina. <https://fundacionfada.org/informes/el-24-del-empleo-nacional-viene-del-agro/>

Bisulco, A., Ojeda, F. C., Isler, V., & Lee, D. D. (2020, Julio). *Near-chip Dynamic Vision Filtering for Low-Bandwidth Pedestrian Detection*. In 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (pp. 234-239). IEEE. <https://ieeexplore.ieee.org/document/9155035>

Braun, A. (2021, 27 de agosto). *Quantization Tutorial in TensorFlow for ML model*. Medium. <https://medium.com/codex/quantization-tutorial-in-tensorflow-to-optimize-a-ml-model-like-a-pro-cadf811482d9>

Canepa, A., Ragusa, E., Zunino, R., & Gastaldo, P. (2022, octubre). *Detection-based video surveillance using Deep Neural Networks on STM32 microcontroller*. In 2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS) (pp. 1-4). IEEE. <https://ieeexplore.ieee.org/document/9970956>

Cody, T., & Beling, P. A. (2021, 2 de julio). *A Systems Theory of Transfer Learning*. arXiv. <https://arxiv.org/pdf/2107.01196>

Computer vision. Wikipedia. https://en.wikipedia.org/w/index.php?title=Computer_vision&oldid=1155325932 (verificado 18 may. 2023)

Dietz, H., Abney, D., Eberhart, P., Santini, N., Davis, W., Wilson, E., & McKenzie, M. (2022). *ESP32-CAM as a programmable camera research platform*. Imaging, 34(232). <https://scholar.archive.org/work/uiguk67mcndkfkvwltqnqn4wlu/access/wayback/https://library.imaging.org/admin/apis/public/api/sandbox/website/downloadArticle/ei/34/7/ISS-232>

Dilmegani, C. (2022, 26 de diciembre). *What is Data Augmentation? Techniques & Examples in 2023*. AIMultiple. <https://research.aimultiple.com/data-augmentation/>

Ibrahim, D. (2020). *ARM-Based Microcontroller Multitasking Projects: Using the FreeRTOS Multitasking Kernel*. Newnes. <https://doi.org/10.1016/C2019-0-03224-4>

Jain, V. (2019, 29 de diciembre). *Everything you need to know about “Activation Functions” in Deep learning models*. Towards Data Science.

<https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253>

Khandelwal, R. (2020, 14 de junio). A Basic Introduction to TensorFlow Lite. Towards Data Science. <https://towardsdatascience.com/a-basic-introduction-to-tensorflow-lite-59e480c57292>

Kurniawan, D. (2022, 13 de Mayo). *Several Types of Artificial Neural Networks Architecture that You Should Know*. <https://dedekurniawann.medium.com/several-types-of-artificial-neural-networks-architecture-that-you-should-know-c169a5e22ec7>

Labonne, M. (2023, 7 de julio). *Introduction to Weight Quantization*. Towards Data Science. <https://towardsdatascience.com/introduction-to-weight-quantization-2494701b9c0c>

LoRa. Wikipedia. <https://en.wikipedia.org/w/index.php?title=LoRa&oldid=1155520088> (verificado 19 may. 2023)

LoRa Alliance. (2015). *What is LoRaWAN®*. LoRa Alliance Resource Hub. <https://resources.lora-alliance.org/getting-started-with-lorawan/what-is-lorawan>

LoRa and LoRaWAN: Technical overview. Semtech. <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/> (verificado 30 oct. 2023)

Lutkevich, B. (2019, Noviembre). *What is a Microcontroller and How Does it Work?* TechTarget. <https://www.techtarget.com/iotagenda/definition/microcontroller> (verificado 26 feb. 2023)

Marmolejo, R. E. (s.f.). *Microcontrolador - qué es y para que sirve*. Hetpro.

<https://hetpro-store.com/TUTORIALES/microcontrolador/> (verificado 15 oct. 2023)

McEnroe, P., Wang, S., & Liyanage, M. "A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges". *IEEE Internet of Things, Journal* 9, no. 17 (2022): 15435-15459..

https://www.researchgate.net/publication/360594094_A_Survey_on_the_Convergence_of_Edge_Computing_and_AI_for_UAVs_Opportunities_and_Challenges

Mechler, T. (2014, 24 de abril). Intrusion detection technology innovations to reduce false security alarms. *SecurityInformed.com*.

[https://www.securityinformed.com/insights/co-289-ga.13346.html?](https://www.securityinformed.com/insights/co-289-ga.13346.html?utm_source=SSc%20International%20Edition&utm_medium=Redirect&utm_campaign=International%20Redirect%20Popup)

[utm_source=SSc%20International%20Edition&utm_medium=Redirect&utm_campaign=International%20Redirect%20Popup](https://www.securityinformed.com/insights/co-289-ga.13346.html?utm_source=SSc%20International%20Edition&utm_medium=Redirect&utm_campaign=International%20Redirect%20Popup) (verificado 18 may. 2023)

Microcontrolador. Wikipedia. <https://es.wikipedia.org/wiki/Microcontrolador> (verificado 28 oct. 2023)

Ming, Z. (2022, 28 de abril). *Google Colab for Machine Learning Projects*. Machine Learning Mastery. <https://machinelearningmastery.com/google-colab-for-machine-learning-projects/>

Opengenus. *Basics of Quantization in Machine Learning (ML) for Beginners*.

OpenGenus. <https://iq.opengenus.org/basics-of-quantization-in-ml/> (verificado 22 oct. 2023)

Ponce, J. (2021, 10 de julio). *Conoce qué son las funciones de activación y cómo puedes crear tu función de activación usando Python, R y Tensorflow*. Jahaziel

Ponce. <https://jahazielponce.com/funciones-de-activacion-y-como-puedes-crear-la-tuya-usando-python-r-y-tensorflow/>

Raj, B. (2018, 24 de junio). *Deep Learning on the Edge*. Towards Data Science.

<https://towardsdatascience.com/deep-learning-on-the-edge-9181693f466c>

Red neuronal artificial. Wikipedia.

https://es.wikipedia.org/wiki/Red_neuronal_artificial (verificado 31 oct. 2023)

Silva, a., & Freire, E. (2019, 23 de noviembre). *Intro a las redes neuronales convolucionales*. *Medium*. <https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8>

Softmax function. Wikipedia. https://en.wikipedia.org/wiki/Softmax_function (verificado 31 oct. 2023)

Tensorflow Model Garden. Github. <https://github.com/tensorflow/models> (verificado 15 sep. 2023)

TensorFlow Lite for Microcontrollers. Github. <https://github.com/tensorflow/tflite-micro> (verificado 10 sep. 2023)

TensorFlow Lite. TensorFlow. <https://www.tensorflow.org/lite/guide> (verificado 28 may. 2023)

Tigreros, J. (2019, 21 de septiembre). ¿Por qué utilizar una GPU sobre la CPU en Deep Learning?. LinkedIn. <https://www.linkedin.com/pulse/por-qu%C3%A9-utilizar-una-gpu-sobre-la-cpu-en-deep-tigreros-ni%C3%B1o/?originalSubdomain=es>

Toma, C., Popa, M., Iancu, B., Doinea, M., Pascu, A., & Ioan-Dutescu, F. (2022). Edge Machine Learning for the Automated Decision and Visual Computing of the Robots, IoT Embedded Devices or UAV-Drones. *Electronics*, 11(21).

<https://www.mdpi.com/2079-9292/11/21/3507>

Ubierna, O. (2020, 23 de noviembre). ¿Qué es LoRa y LoRaWan?. Comunicaciones inalámbricas hoy. <https://www.comunicacionesinalambricashoy.com/wireless/que-es-lora-y-lorawan/>

Wulfert, L., Wiede, C., Verbunt, M. H., Gembaczka, P., & Grabmaier, A. (2022,

septiembre). *Human Detection with A Feedforward Neural Network for Small Microcontrollers*. In 2022 7th International Conference on Frontiers of Signal Processing (ICFSP) (pp 14-22). IEEE.

<https://ieeexplore.ieee.org/abstract/document/9924667/authors#authors>

6. Anexo A: Funciones de activación

Una función de activación en una red neuronal artificial es una función matemática que determina la salida de una neurona o nodo en función de sus entradas. Esta función introduce no linealidad en la red, lo que permite que la red capture relaciones y patrones más complejos en los datos. (Jain, 2019)

Seguidamente se listan algunas de las funciones de activación más comunes:

Función Sigmoide:

La función de activación sigmoide es una de las funciones de activación más comunes utilizadas en redes neuronales artificiales. Su forma se asemeja a una curva en forma de "S" (Fig. 6.1) y se define matemáticamente en la ecuación 6.1, donde x es la entrada a la función de activación. (*Activation Function*, sin fecha)

$$\sigma(x) \doteq \frac{1}{1 + e^{-x}}$$

Ecuación 6.1: Función de activación sigmoide.

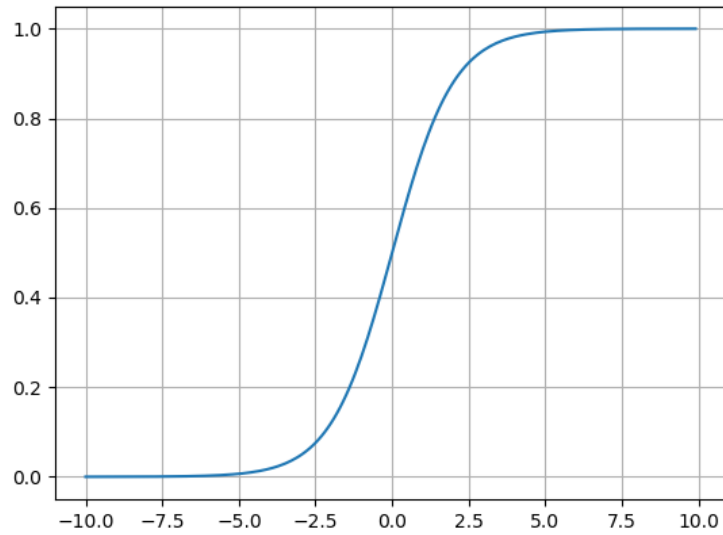


Figura 6.1: Función de activación Sigmoide. Tomado de Ponce (2021)

Función ReLU (Rectified Linear Unit)

La función de activación ReLU (Fig. 6.2) se define matemáticamente como en la ecuación 6.2.

$$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

$$= \max(0, x) = x \mathbf{1}_{x>0}$$

Ecuación 6.2: Función de activación ReLU.

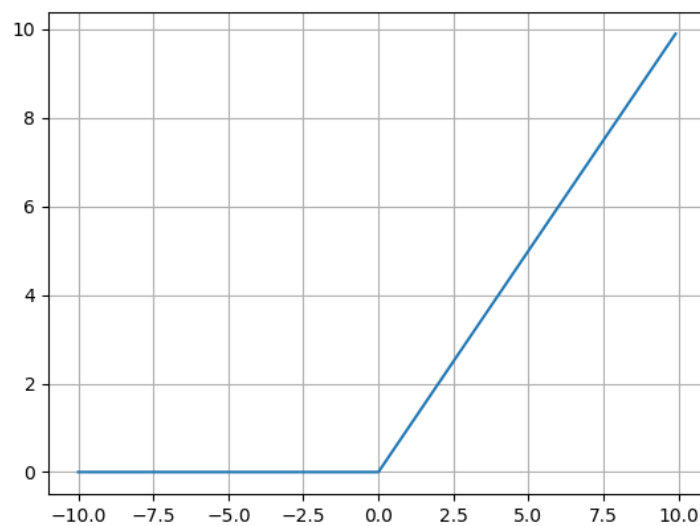


Figura 6.2: Función de activación ReLU. Tomado de Ponce (2021).

Función Tangente hiperbólica (Tanh)

La función de activación Tangente Hiperbólica (Fig. 6.3) se define matemáticamente como en la ecuación 6.3.

$$\tanh(x) \doteq \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Ecuación 6.3: Función de activación Tangente Hiperbólica.

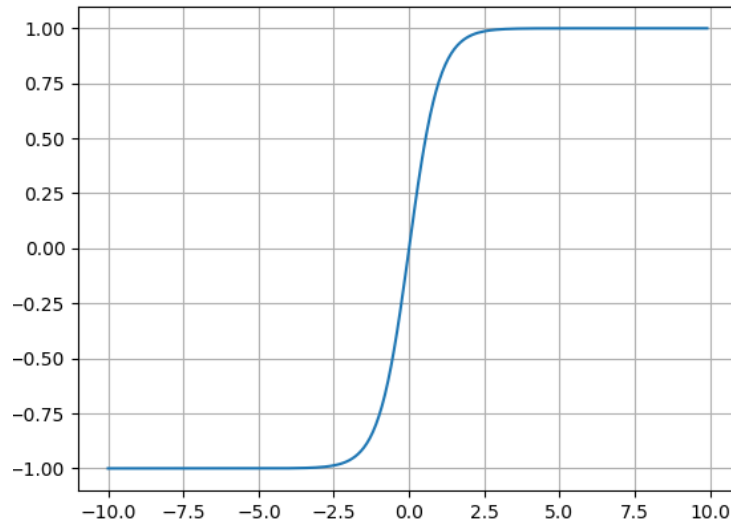


Figura 6.3: Función de activación Tangente hiperbólica. Tomado de Ponce (2021).

Función Softmax

La función de activación Softmax se utiliza comúnmente en la capa de salida de una red, cuando se trata de problemas de clasificación multiclase. La función Softmax toma un vector de números reales y los convierte en una distribución de probabilidad sobre varias clases diferentes.

La fórmula matemática de la función de activación Softmax es la descrita en la ecuación 6.4, donde x_i es la entrada de la neurona i y J corresponde al número de clases.

$$\frac{e^{x_i}}{\sum_{j=1}^J e^{x_j}}$$

Ecuación 6.4: Función de activación Softmax.

El propósito de la función Softmax es asignar probabilidades a cada clase, de modo que la clase con la probabilidad más alta sea la predicción final de la red. La función Softmax toma como entrada un vector de números (típicamente llamado logit) y produce una distribución de probabilidad donde la suma de todas las probabilidades es igual a 1. (*Softmax Function*, sin fecha)

7. Anexo B: Arquitectura y estrategia de entrenamiento de los clasificadores

7.1 Clasificador A

Arquitectura

Para este clasificador (Fig. 7.1) se contemplaron las siguientes capas en la sección de clasificación:

- *Global Average Pooling 2D*.
- Regularización *DropOut* (20%).
- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 2
 - Función de activación: SoftMax

Cantidad de parámetros de la red neuronal: 251698

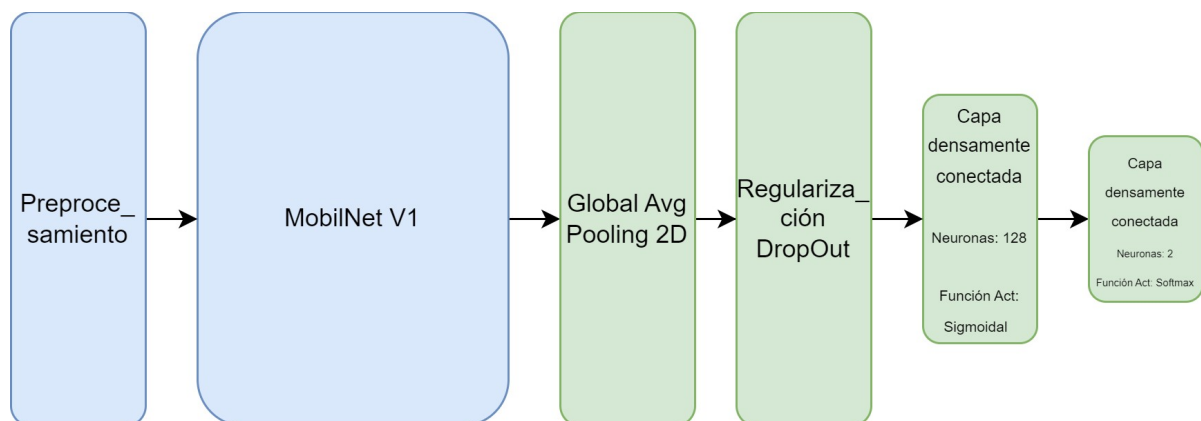


Figura 7.1: Diseño arquitectura Clasificador A.

Estrategia de entrenamiento

- 1° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 0
 - Cantidad de parámetros a entrenar: 33154
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.001

- Función de pérdida: Sparse Categorical Cross Entropy
- 2° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 15
 - Cantidad de parámetros a entrenar: 136962
 - Parámetros de optimización:
 - Función de optimización: RMSprop
 - Tasa de aprendizaje inicial: 0.0001
 - Función de pérdida: Sparse Categorical Cross Entropy

7.2 Clasificador B

Este clasificador mantiene la misma arquitectura que el Clasificador A, pero cambia en su estrategia de entrenamiento para su segunda etapa, cambiando la función de optimización a Adam.

Arquitectura

Para este clasificador se contemplaron las siguientes capas en la sección de clasificación:

- Global Average Pooling 2D.
- Regularización DropOut (20%).
- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 2
 - Función de activación: SoftMax

Cantidad de parámetros de la red neuronal: 251698

Estrategia de entrenamiento

- 1° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 0
 - Cantidad de parámetros a entrenar: 33154
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.001
 - Función de pérdida: Sparse Categorical Cross Entropy

- 2° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 15
 - Cantidad de parámetros a entrenar: 136962
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.0001
 - Función de pérdida: Sparse Categorical Cross Entropy

7.3 Clasificador C

Para este clasificador se adiciono una capa densamente conectada de 128 neuronas. Por otro lado se mantuvo el algoritmo ADAM como algoritmo de optimización en la segunda etapa de entrenamiento.

Arquitectura

Para este clasificador (Fig. 7.2) se contemplaron las siguientes capas en la sección de clasificación:

- Global Average Pooling 2D.
- Regularización DropOut (20%).
- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 2
 - Función de activación: SoftMax

Cantidad de parámetros de la red neuronal: 268210

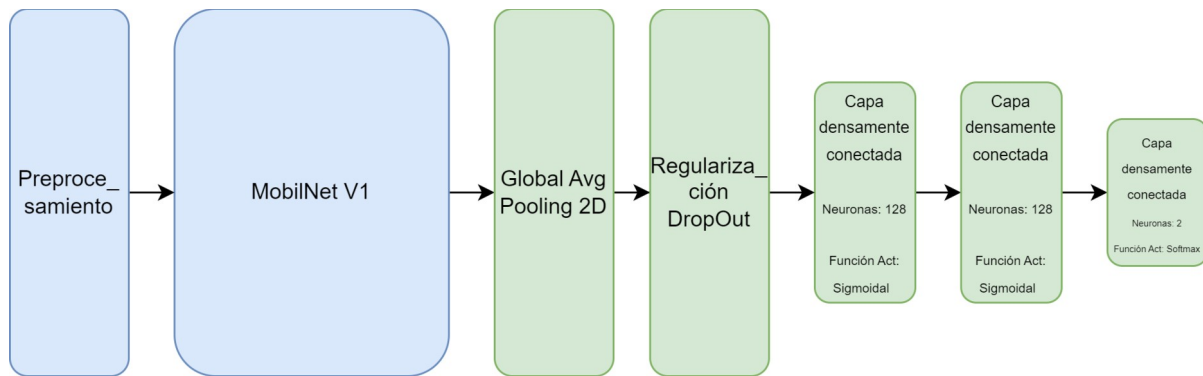


Figura 7.2: Diseño arquitectura Clasificador C.

Estrategia de entrenamiento

- 1° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 0
 - Cantidad de parámetros a entrenar: 49666
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.001
 - Función de pérdida: Sparse Categorical Cross Entropy
- 2° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 15
 - Cantidad de parámetros a entrenar: 153474
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.0001
 - Función de pérdida: Sparse Categorical Cross Entropy

7.4 Clasificador D

A diferencia del clasificador C, se reemplaza las 2 capas densamente conectadas de 128 neuronas por 1 de 256 neuronas.

Arquitectura

Para este clasificador (Fig. 7.3) se contemplaron las siguientes capas en la sección de clasificación:

- Global Average Pooling 2D.
- Regularización DropOut (20%).

- Capa densamente conectada:
 - Cantidad de Neuronas: 256
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 2
 - Función de activación: SoftMax

Cantidad de parámetros de la red neuronal: 284850

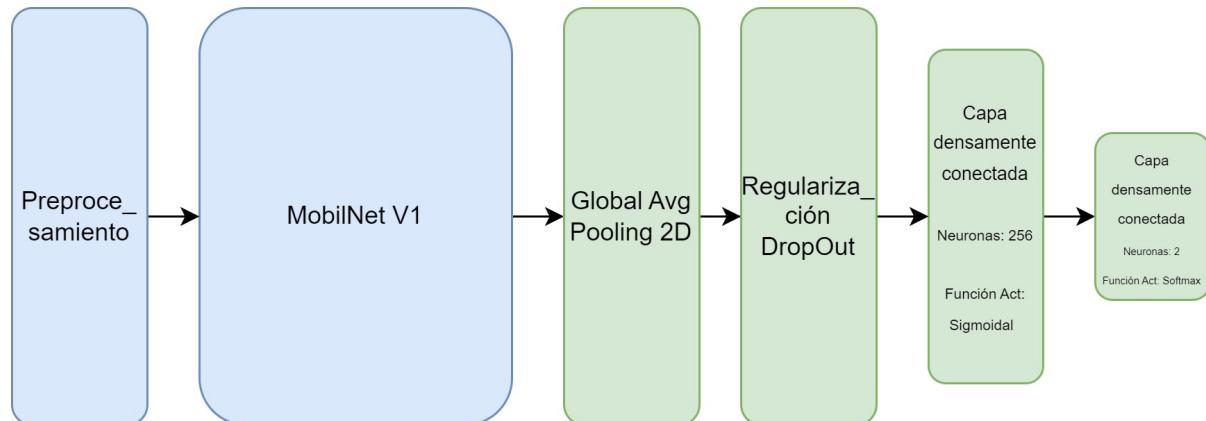


Figura 7.3: Diseño arquitectura Clasificador D.

Estrategia de entrenamiento

- 1° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 0
 - Cantidad de parámetros a entrenar: 66306
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.001
 - Función de pérdida: Sparse Categorical Cross Entropy
- 2° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 15
 - Cantidad de parámetros a entrenar: 170114
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.0001
 - Función de pérdida: Sparse Categorical Cross Entropy

7.5 Clasificador E

A diferencia del clasificador D, se adiciono una capa más de 256 neuronas.

Arquitectura

Para este clasificador (Fig. 7.4) se contemplaron las siguientes capas en la sección de clasificación:

- Global Average Pooling 2D.
- Regularización DropOut (20%).
- Capa densamente conectada:
 - Cantidad de Neuronas: 256
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 256
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 2
 - Función de activación: SoftMax

Cantidad de parámetros de la red neuronal: 350642

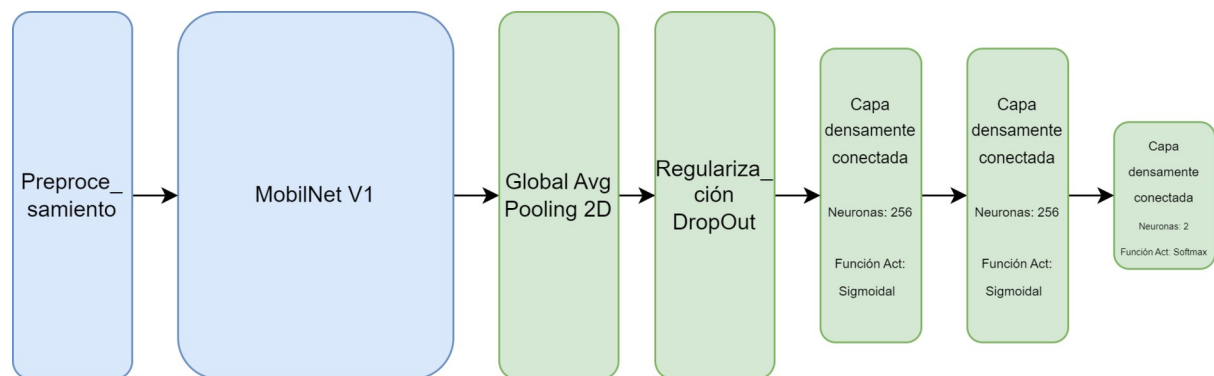


Figura 7.4: Diseño arquitectura Clasificador E.

Estrategia de entrenamiento

- 1° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 0
 - Cantidad de parámetros a entrenar: 132098
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.001
 - Función de pérdida: Sparse Categorical Cross Entropy

- 2° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 15
 - Cantidad de parámetros a entrenar: 235906
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.0001
 - Función de pérdida: Sparse Categorical Cross Entropy

7.6 Clasificador F

Este clasificador siguió la misma arquitectura que el Clasificador C, pero se distingue por la modificación en la cantidad de capas que se entrenaron en el modelo MobileNet, aumentando de 15 a 30 capas.

Arquitectura

Para este clasificador se contemplaron las siguientes capas en la sección de clasificación:

- Global Average Pooling 2D.
- Regularización DropOut (20%).
- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 2
 - Función de activación: SoftMax

Cantidad de parámetros de la red neuronal: 268210

Estrategia de entrenamiento

- 1° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 0
 - Cantidad de parámetros a entrenar: 49666
 - Parámetros de optimización:
 - Función de optimización: Adam

- Tasa de aprendizaje inicial: 0.001
 - Función de pérdida: Sparse Categorical Cross Entropy
- 2° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 30
 - Cantidad de parámetros a entrenar: 206210
 - Parámetros de optimización:
 - Función de optimización: ADAM
 - Tasa de aprendizaje inicial: 0.0001
 - Función de pérdida: Sparse Categorical Cross Entropy

7.7 Clasificador G

Por último, este clasificador siguió la misma arquitectura que el Clasificador C, pero se diferencia por una modificación en la cantidad de capas entrenadas en el modelo MobileNet, reduciendo de 15 a 5 capas.

Arquitectura

Para este clasificador se contemplaron las siguientes capas en la sección de clasificación:

- Global Average Pooling 2D.
- Regularización DropOut (20%).
- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: Sigmoidal
- Capa densamente conectada:
 - Cantidad de Neuronas: 2
 - Función de activación: SoftMax

Cantidad de parámetros de la red neuronal: 268210

Estrategia de entrenamiento

- 1° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 0
 - Cantidad de parámetros a entrenar: 49666

- Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.001
 - Función de pérdida: Sparse Categorical Cross Entropy
- 2° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 5
 - Cantidad de parámetros a entrenar: 116226
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.0001
 - Función de pérdida: Sparse Categorical Cross Entropy

7.8 Clasificador H

Este clasificador mantiene una arquitectura similar al del Clasificador B, pero cambia en su función de activación la cual utiliza ReLU en vez de Sigmoidal. Por otro lado, su estrategia de entrenamiento es la misma.

Arquitectura

Para este clasificador se contemplaron las siguientes capas en la sección de clasificación:

- Global Average Pooling 2D.
- Regularización DropOut (20%).
- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: ReLU
- Capa densamente conectada:
 - Cantidad de Neuronas: 2
 - Función de activación: SoftMax

Cantidad de parámetros de la red neuronal: 251698

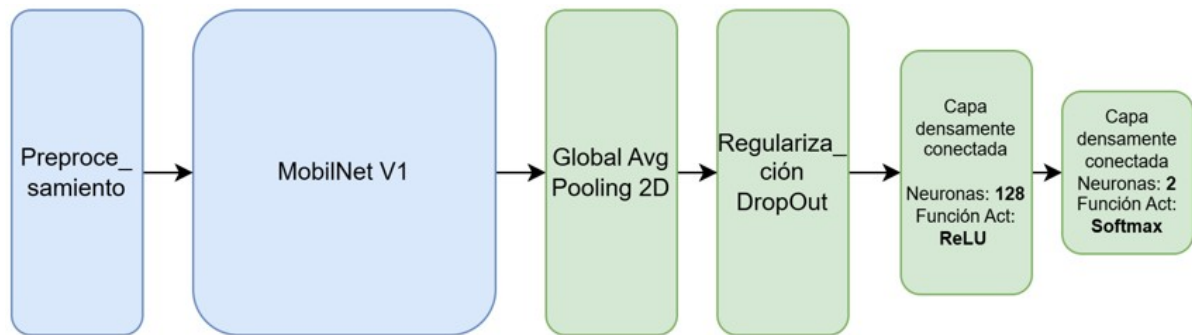


Figura 7.4: Diseño arquitectura Clasificador H.

Estrategia de entrenamiento

- 1° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 0
 - Cantidad de parámetros a entrenar: 33154
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.001
 - Función de pérdida: Sparse Categorical Cross Entropy
- 2° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 30
 - Cantidad de parámetros a entrenar: 189698
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.0001
 - Función de pérdida: Sparse Categorical Cross Entropy

7.9 Clasificador I

Con respecto al clasificador H, este clasificador se adiciono una capa densamente conectada de 128 neuronas. Por otro lado se mantuvo el algoritmo ADAM como algoritmo de optimización en la segunda etapa de entrenamiento.

Arquitectura

Para este clasificador (Fig. 7.2) se contemplaron las siguientes capas en la sección de clasificación:

- Global Average Pooling 2D.
- Regularización DropOut (20%).

- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: ReLU
- Capa densamente conectada:
 - Cantidad de Neuronas: 128
 - Función de activación: ReLU
- Capa densamente conectada:
 - Cantidad de Neuronas: 2
 - Función de activación: SoftMax

Cantidad de parámetros de la red neuronal: 268210

Figura 7.2: *Diseño arquitectura Clasificador I.*

Estrategia de entrenamiento

- 1° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 0
 - Cantidad de parámetros a entrenar: 49666
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.001
 - Función de pérdida: Sparse Categorical Cross Entropy
- 2° etapa de entrenamiento:
 - Cantidad de épocas: 60
 - Cantidad de capas a entrenar del modelo MobileNet: 30
 - Cantidad de parámetros a entrenar: 206210
 - Parámetros de optimización:
 - Función de optimización: Adam
 - Tasa de aprendizaje inicial: 0.0001
 - Función de pérdida: Sparse Categorical Cross Entropy

8. Anexo C: Análisis de performance de los clasificadores

En esta sección se detalla la información relativa al análisis de performance de cada uno de los clasificadores entrenados.

8.1 Análisis de performance: Clasificador A

Seguidamente se muestra un resumen de distintos indicadores alcanzados en las diferentes etapas del entrenamiento del Clasificador A:

- Indicadores al finalizar 1° etapa de entrenamiento:
 - Error - Datos de entrenamiento: 0.5463
 - Error - Datos de validación: 0.4907
 - Indicadores de performance - Punto de corte: 0.5
 - *Accuracy* - Datos de entrenamiento: 0.7184
 - *Accuracy* - Datos de validación: 0.7595
- Indicadores al finalizar el entrenamiento del modelo:
 - Error - Datos de entrenamiento: 0.5329
 - Error - Datos de validación: 0.4947
 - Indicadores de performance - Punto de corte: 0.5
 - *Accuracy* - Datos de entrenamiento: 0.7342
 - *Accuracy* - Datos de validación: 0.7669
 - *Precision* - Datos de validación: 0.7784
 - *Recall* - Datos de validación: 0.7073
 - F1 - Datos de validación: 0.7412
 - Indicadores de performance - Punto de corte: 0.75
 - *Accuracy* - Datos de validación: 0.7252
 - *Precision* - Datos de validación: 0.8824
 - *Recall* - Datos de validación: 0.4820
 - F1 - Datos de validación: 0.6235
 - Indicadores de performance - Punto de corte: 0.90
 - *Accuracy* - Datos de validación: 0.6571
 - *Precision* - Datos de validación: 0.9320
 - *Recall* - Datos de validación: 0.2949
 - F1 - Datos de validación: 0.4480

En la figura 8.1, se presenta la evolución del error del modelo a lo largo de las iteraciones, considerando tanto los datos de entrenamiento como los datos de validación.

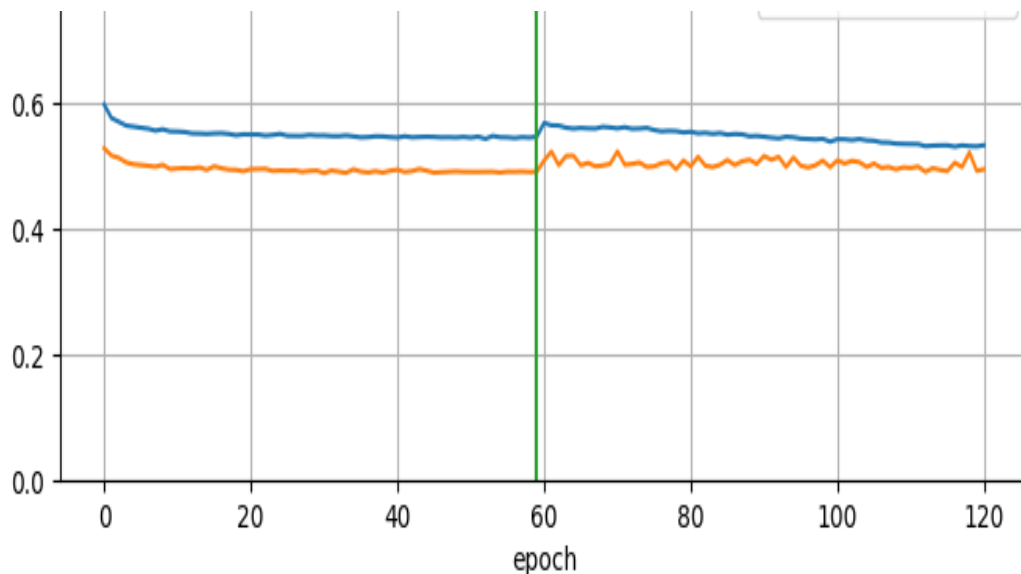


Figura 8.1: Evolución del error en entrenamiento del Clasificador A.

La línea verde, señalada como *Start Fine Tuning*, marca el inicio de la segunda etapa de entrenamiento del clasificador. En esta fase, se adicionan al entrenamiento las últimas capas del modelo base MobileNet.

En la figura 8.2 se ve la evolución del *Accuracy* del modelo considerando un punto de corte de 0.5 para la salida del modelo.

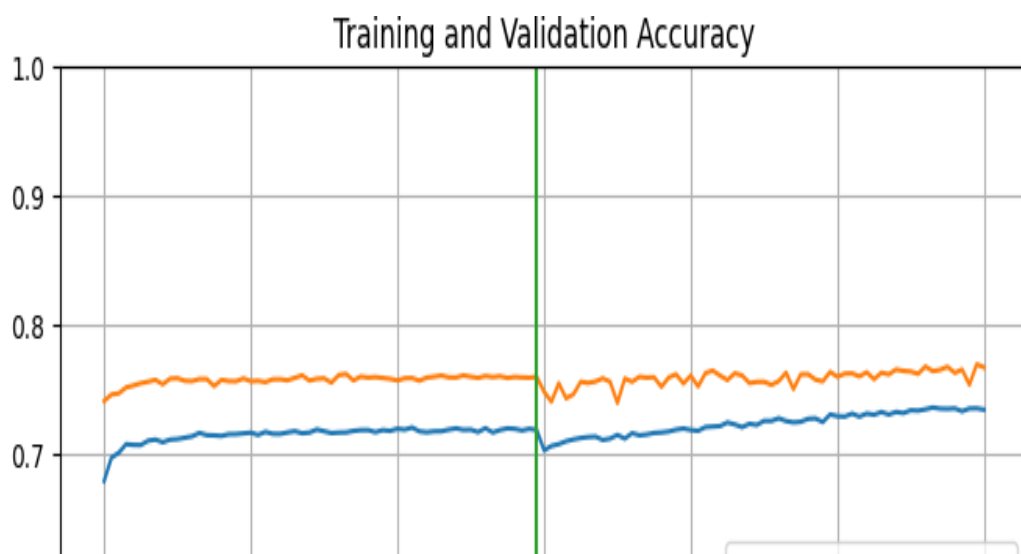


Figura 8.2: Evolución del accuracy en el entrenamiento del Clasificador A.

La siguiente representación gráfica (Fig. 8.3) muestra cómo variaron las métricas *precision*, *recall*, F1 y accuracy a medida que se ajustaba el punto de corte del modelo de clasificación ya entrenado. Los indicadores se obtuvieron con los datos de validación.

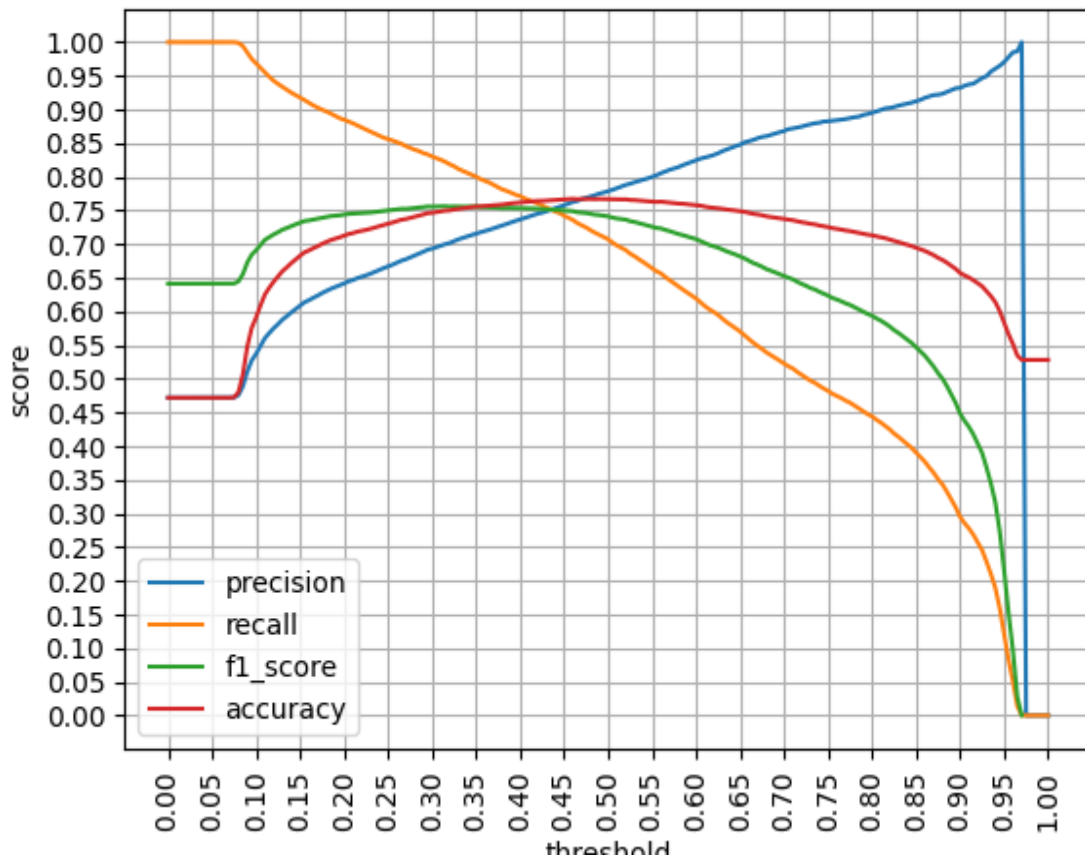


Figura 8.3: Umbral de discriminación del Clasificador A.

8.2 Análisis de performance: Clasificador B

- Indicadores al finalizar 1° etapa de entrenamiento:
 - Error - Datos de entrenamiento: 0.5432
 - Error - Datos de validación: 0.5005
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7206
 - Accuracy - Datos de validación: 0.7525
- Indicadores al finalizar el entrenamiento del modelo:
 - Error - Datos de entrenamiento: 0.4971
 - Error - Datos de validación: 0.4669
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7554
 - Accuracy - Datos de validación: 0.7773
 - Precision - Datos de validación: 0.7624
 - Recall - Datos de validación: 0.7645
 - F1 - Datos de validación: 0.7634
 - Indicadores de performance - Punto de corte: 0.75
 - Accuracy - Datos de validación: 0.7566
 - Precision - Datos de validación: 0.8674

- *Recall* - Datos de validación: 0.5699
- F1 - Datos de validación: 0.6879
- Indicadores de performance - Punto de corte: 0.90
 - *Accuracy* - Datos de validación: 0.6930
 - *Precision* - Datos de validación: 0.9429
 - *Recall* - Datos de validación: 0.3699
 - F1 - Datos de validación: 0.5313

En la figura 8.4 se puede visualizar la evolución del error en la etapa de entrenamiento de este modelo de clasificación.

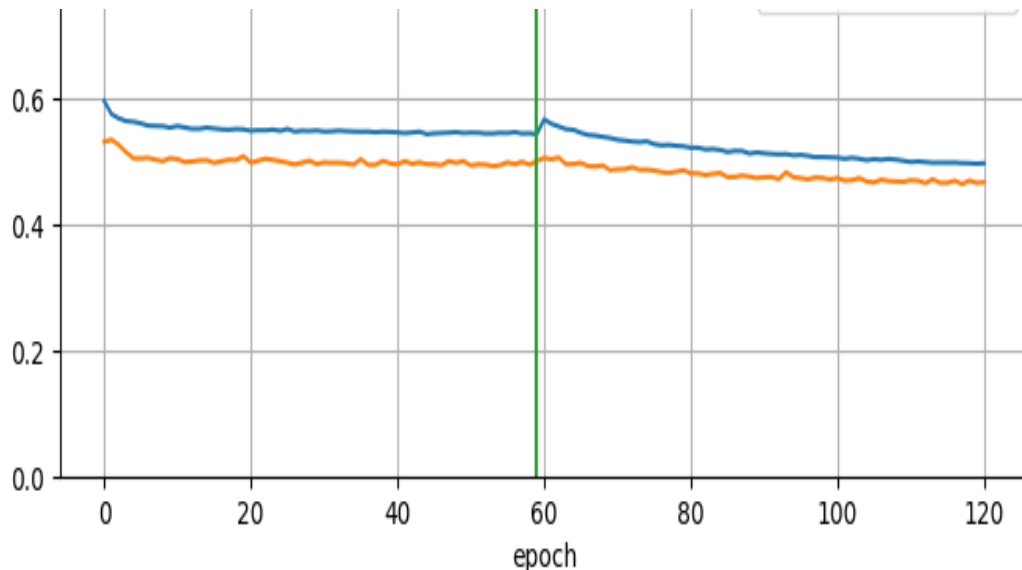


Figura 8.4: Evolución del error en entrenamiento del Clasificador B.

En la figura 8.5 se ve la evolución del Accuracy del modelo considerando un punto de corte de 0.5 para la salida del modelo.

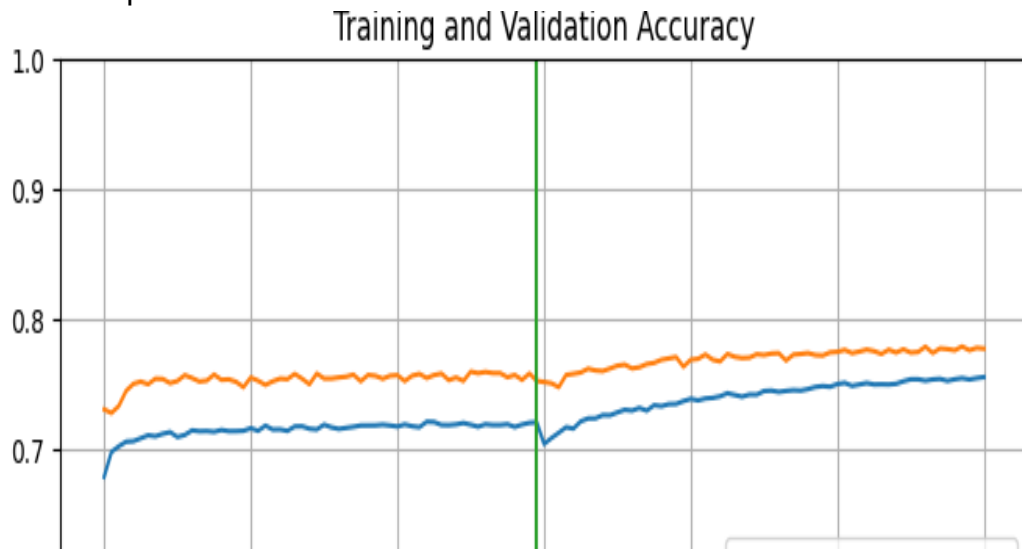


Figura 8.5: Evolución del accuracy en el entrenamiento del Clasificador B.

En la figura 8.6 se describe el umbral de discriminación del modelo.

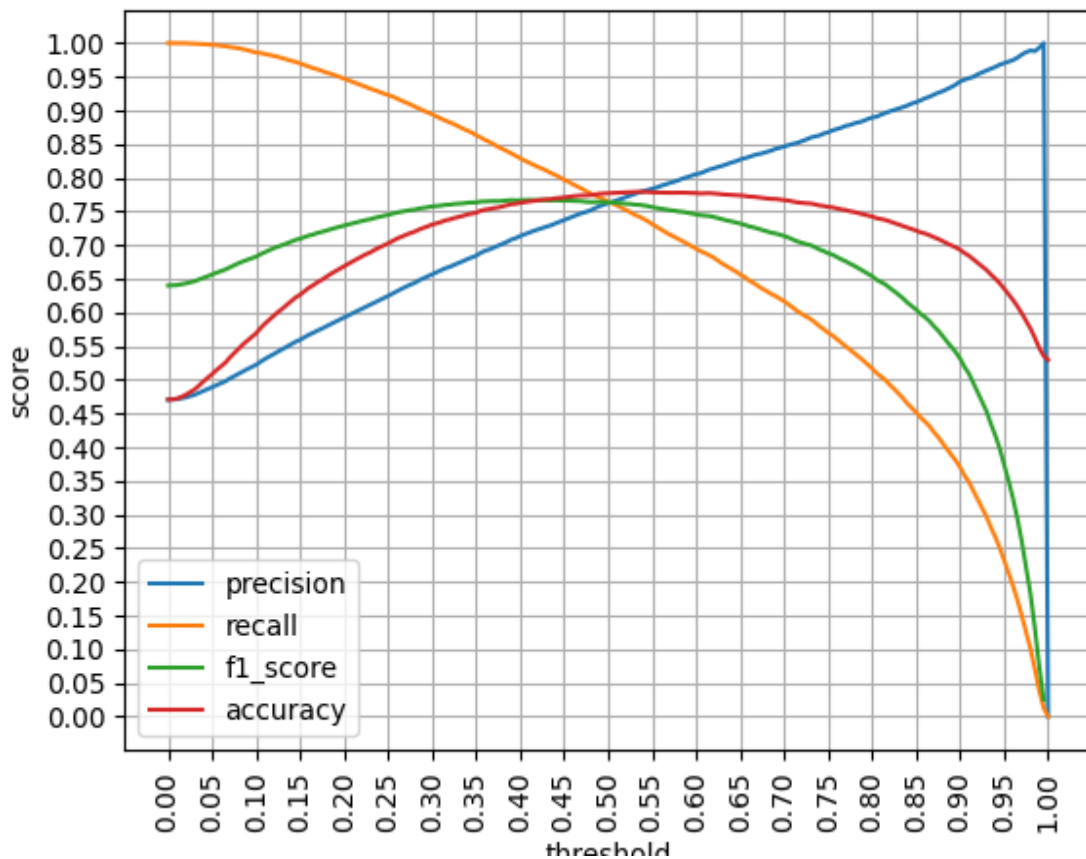


Figura 8.6: Umbral de discriminación Clasificador B.

8.3 Análisis de performance: Clasificador C

- Indicadores al finalizar 1° etapa de entrenamiento:
 - Error - Datos de entrenamiento: 0.5394
 - Error - Datos de validación: 0.4882
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7214
 - Accuracy - Datos de validación: 0.7602
- Indicadores al finalizar el entrenamiento del modelo:
 - Error - Datos de entrenamiento: 0.4960
 - Error - Datos de validación: 0.4693
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7562
 - Accuracy - Datos de validación: 0.7755
 - Precision - Datos de validación: 0.7881
 - Recall - Datos de validación: 0.7184
 - F1 - Datos de validación: 0.7516
 - Indicadores de performance - Punto de corte: 0.75
 - Accuracy - Datos de validación: 0.7414

- *Precision* - Datos de validación: 0.8929
- *Recall* - Datos de validación: 0.5155
- *F1* - Datos de validación: 0.6536
- Indicadores de performance - Punto de corte: 0.90
 - *Accuracy* - Datos de validación: 0.6706
 - *Precision* - Datos de validación: 0.9444
 - *Recall* - Datos de validación: 0.3229
 - *F1* - Datos de validación: 0.4812

En la figura 8.7 se visualiza la evolución del error en la etapa de entrenamiento del modelo de clasificación.

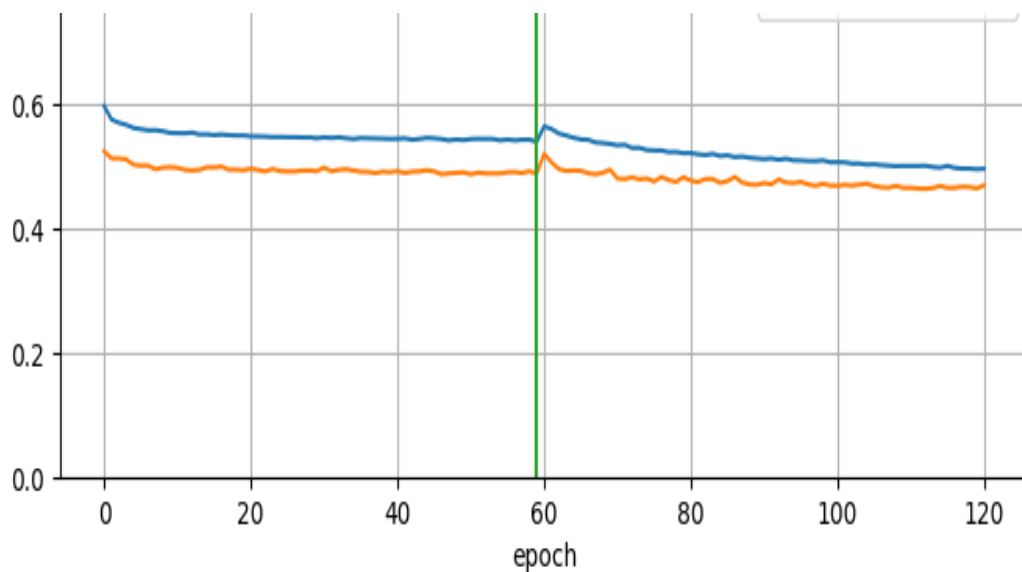


Figura 8.7: Evolución del error en entrenamiento del Clasificador C.

En la figura 8.8 se ve la evolución del Accuracy del modelo considerando un punto de corte de 0.5 para la salida del modelo.

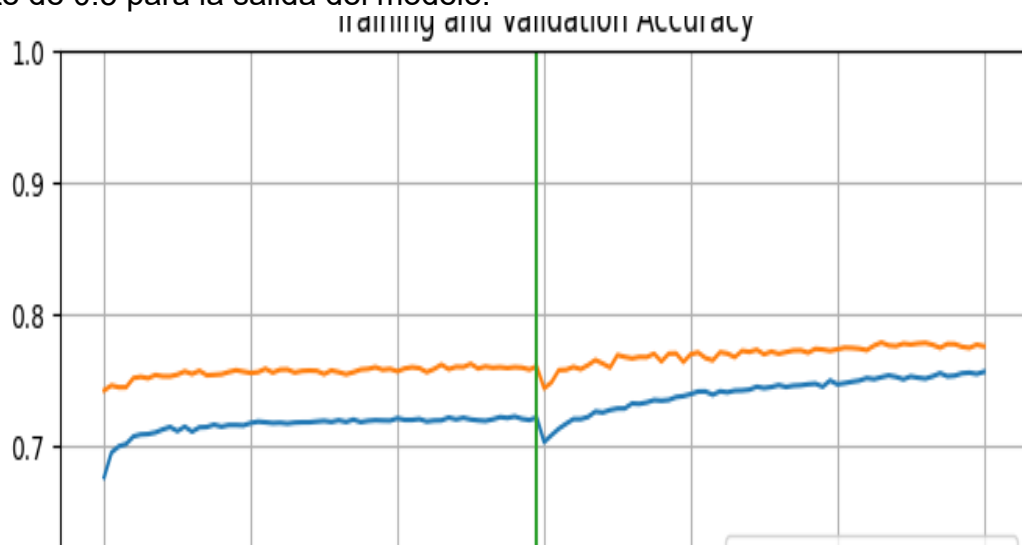


Figura 8.8: Evolución del accuracy en el entrenamiento del Clasificador C.

En la figura 8.9 se describe el umbral de discriminación del modelo.

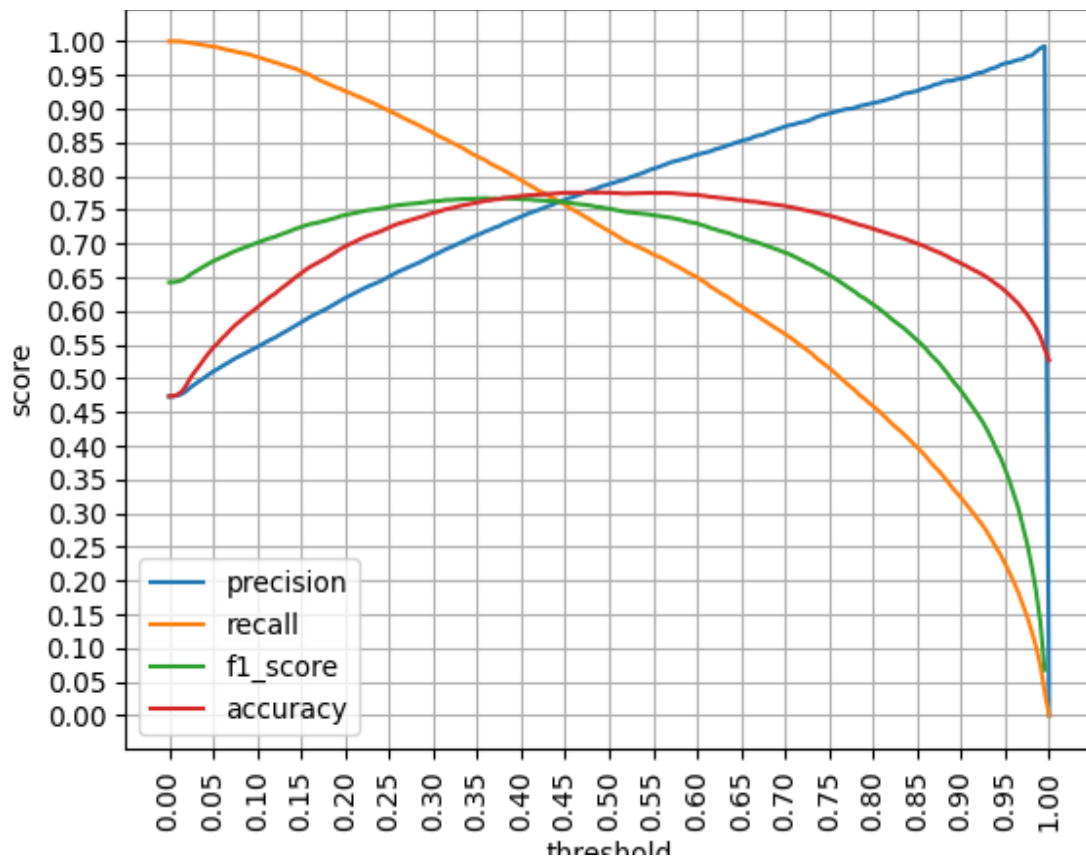


Figura 8.9: Umbral de discriminación Clasificador C.

8.4 Análisis de performance: Clasificador D

- Indicadores al finalizar 1° etapa de entrenamiento:
 - Error - Datos de entrenamiento: 0.5397
 - Error - Datos de validación: 0.4923
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7217
 - Accuracy - Datos de validación: 0.7547
- Indicadores al finalizar el entrenamiento del modelo:
 - Error - Datos de entrenamiento: 0.4935
 - Error - Datos de validación: 0.4571
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7560
 - Accuracy - Datos de validación: 0.7810
 - Precision - Datos de validación: 0.7697
 - Recall - Datos de validación: 0.7660
 - F1 - Datos de validación: 0.7678
 - Indicadores de performance - Punto de corte: 0.75

- *Accuracy* - Datos de validación: 0.7542
- *Precision* - Datos de validación: 0.8828
- *Recall* - Datos de validación: 0.5535
- F1 - Datos de validación: 0.6804
- Indicadores de performance - Punto de corte: 0.90
 - *Accuracy* - Datos de validación: 0.6810
 - *Precision* - Datos de validación: 0.9511
 - *Recall* - Datos de validación: 0.3429
 - F1 - Datos de validación: 0.5041

En la figura 8.10 se visualiza la evolución del error en la etapa de entrenamiento del modelo de clasificación.

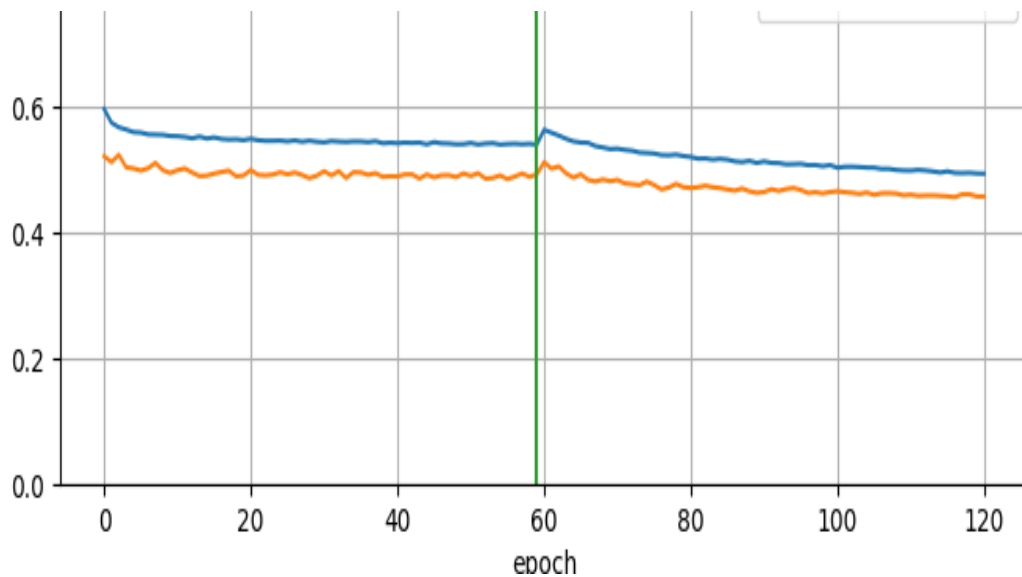


Figura 8.10: Evolución del error en entrenamiento del Clasificador D.

En la figura 8.11 se ve la evolución del *Accuracy* del modelo considerando un punto de corte de 0.5 para la salida del modelo.

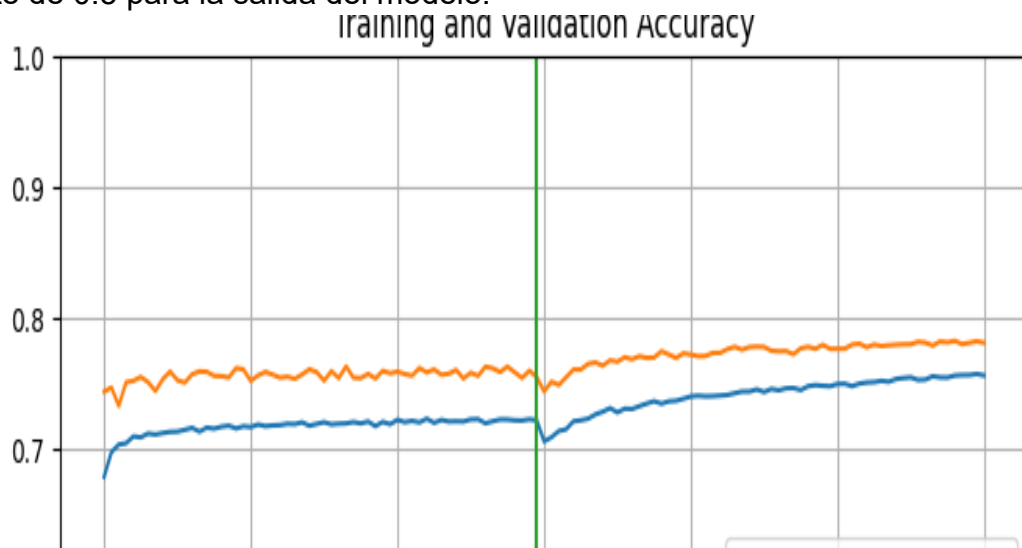


Figura 8.11: Evolución del accuracy en el entrenamiento del Clasificador D.

En la figura 8.12 se describe el umbral de discriminación del modelo.

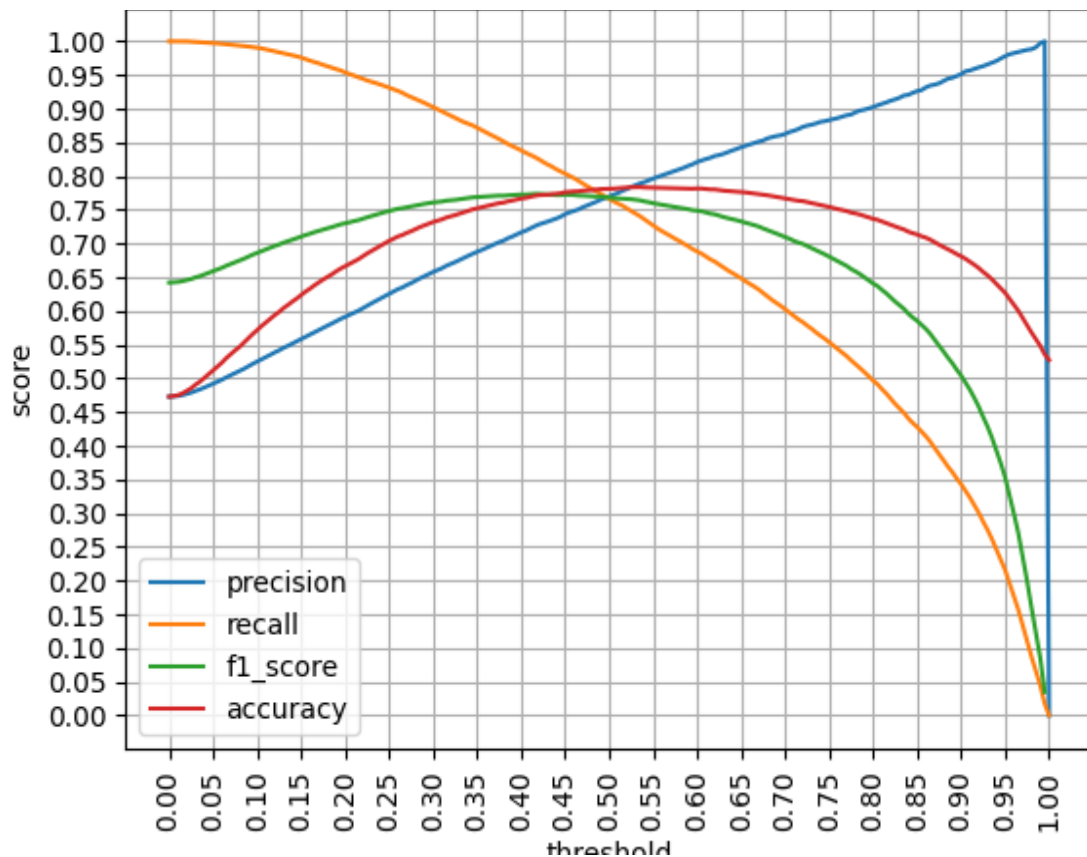


Figura 8.12: Umbral de discriminación Clasificador D.

8.5 Análisis de performance: Clasificador E

- Indicadores al finalizar 1° etapa de entrenamiento:
 - Error - Datos de entrenamiento: 0.5369
 - Error - Datos de validación: 0.4876
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7245
 - Accuracy - Datos de validación: 0.7602
- Indicadores al finalizar el entrenamiento del modelo:
 - Error - Datos de entrenamiento: 0.4942
 - Error - Datos de validación: 0.4587
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7550
 - Accuracy - Datos de validación: 0.7815
 - Precision - Datos de validación: 0.7776
 - Recall - Datos de validación: 0.7534
 - F1 - Datos de validación: 0.7653
 - Indicadores de performance - Punto de corte: 0.75

- *Accuracy* - Datos de validación: 0.7525
- *Precision* - Datos de validación: 0.8841
- *Recall* - Datos de validación: 0.5481
- F1 - Datos de validación: 0.6767
- Indicadores de performance - Punto de corte: 0.90
 - *Accuracy* - Datos de validación: 0.6808
 - *Precision* - Datos de validación: 0.9487
 - *Recall* - Datos de validación: 0.3431
 - F1 - Datos de validación: 0.5040

En la figura 8.13 se visualiza la evolución del error en la etapa de entrenamiento del modelo de clasificación.

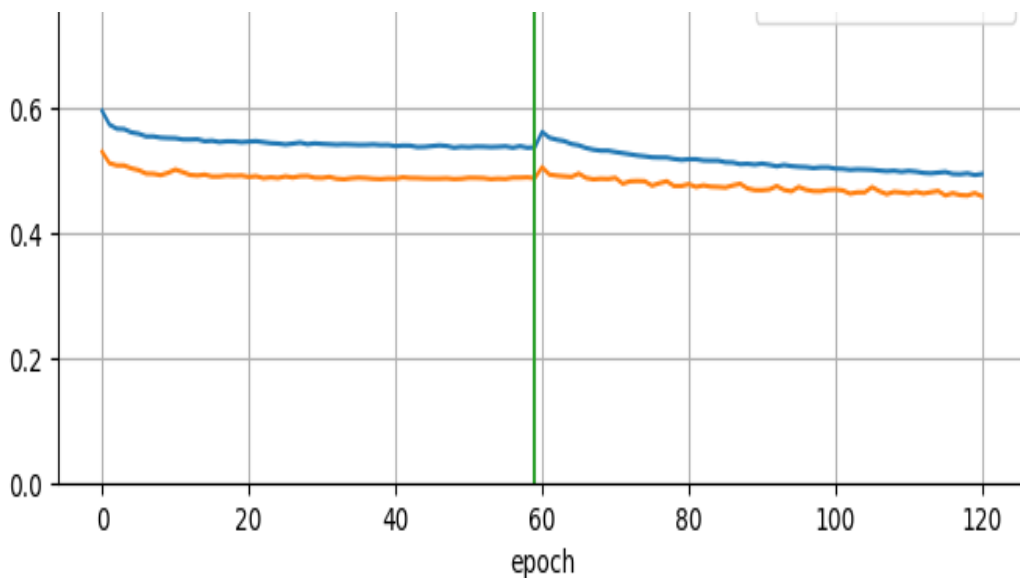


Figura 8.13: Evolución del error en entrenamiento del Clasificador E.

En la figura 8.14 se ve la evolución del *Accuracy* del modelo considerando un punto de corte de 0.5 para la salida del modelo.

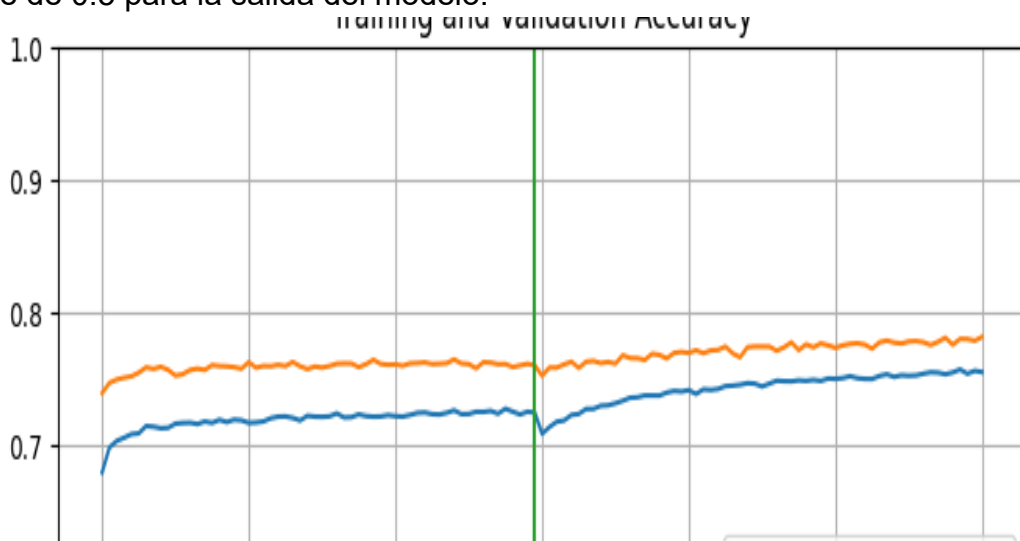


Figura 8.14: Evolución del accuracy en el entrenamiento del Clasificador E.

La figura 8.15 describe el umbral de discriminación del modelo.

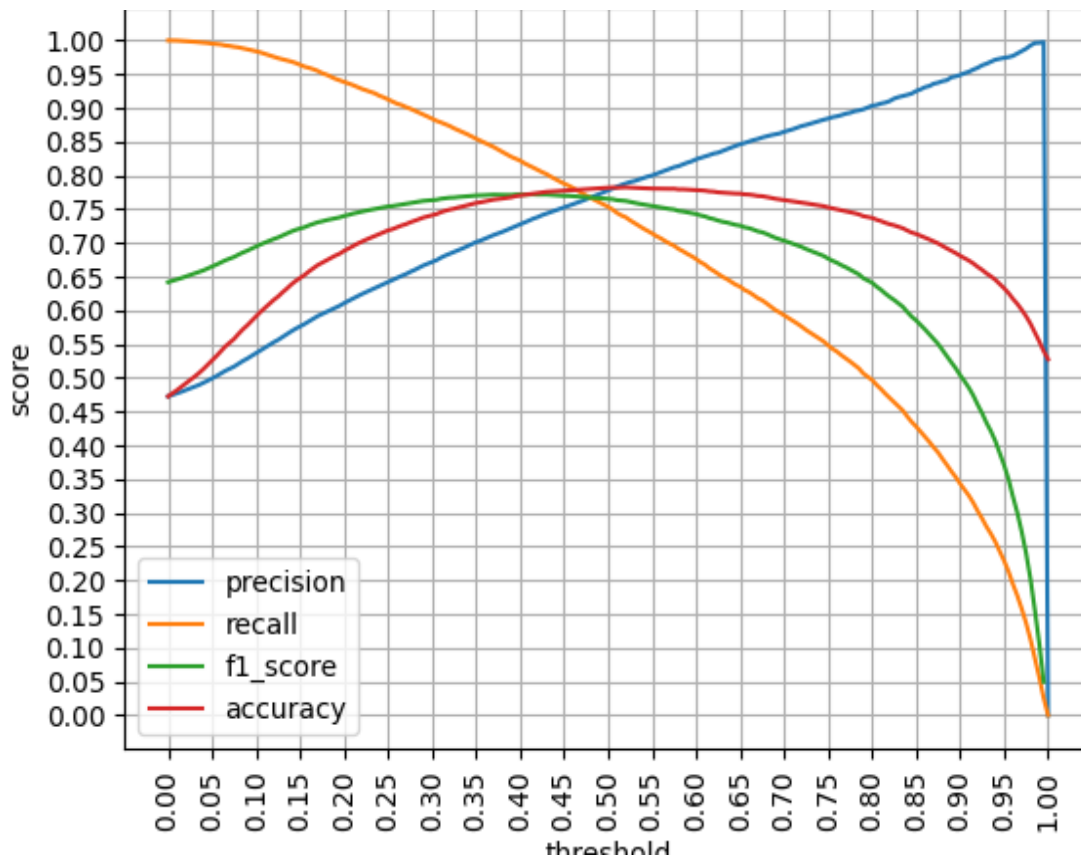


Figura 8.15: Umbral de discriminación Clasificador E.

8.6 Análisis de performance: Clasificador F

- Indicadores al finalizar 1° etapa de entrenamiento:
 - Error - Datos de entrenamiento: 0.5424
 - Error - Datos de validación: 0.4892
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7215
 - Accuracy - Datos de validación: 0.7590
- Indicadores al finalizar el entrenamiento del modelo:
 - Error - Datos de entrenamiento: 0.4737
 - Error - Datos de validación: 0.4668
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7711
 - Accuracy - Datos de validación: 0.7779
 - Precision - Datos de validación: 0.7583
 - Recall - Datos de validación: 0.7756
 - F1 - Datos de validación: 0.7669

- Indicadores de performance - Punto de corte: 0.75
 - *Accuracy* - Datos de validación: 0.7590
 - *Precision* - Datos de validación: 0.8637
 - *Recall* - Datos de validación: 0.5799
 - F1 - Datos de validación: 0.6939
- Indicadores de performance - Punto de corte: 0.90
 - *Accuracy* - Datos de validación: 0.6936
 - *Precision* - Datos de validación: 0.9349
 - *Recall* - Datos de validación: 0.3758
 - F1 - Datos de validación: 0.5361

En la figura 8.16 se visualiza la evolución del error en la etapa de entrenamiento del modelo de clasificación.

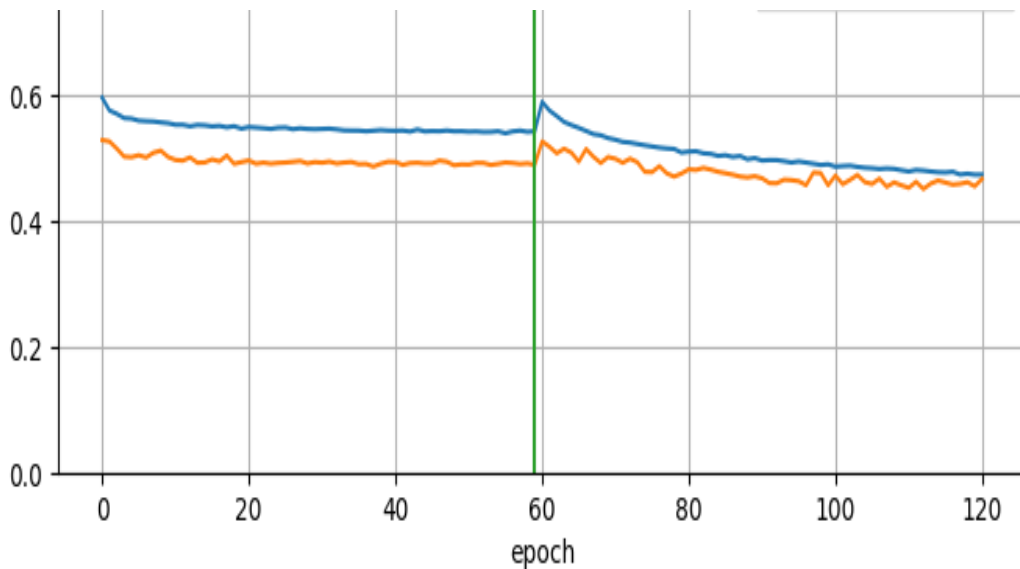


Figura 8.16: Evolución del error en entrenamiento del Clasificador F.

En la figura 8.17 se ve la evolución del *Accuracy* del modelo considerando un punto de corte de 0.5 para la salida del modelo.

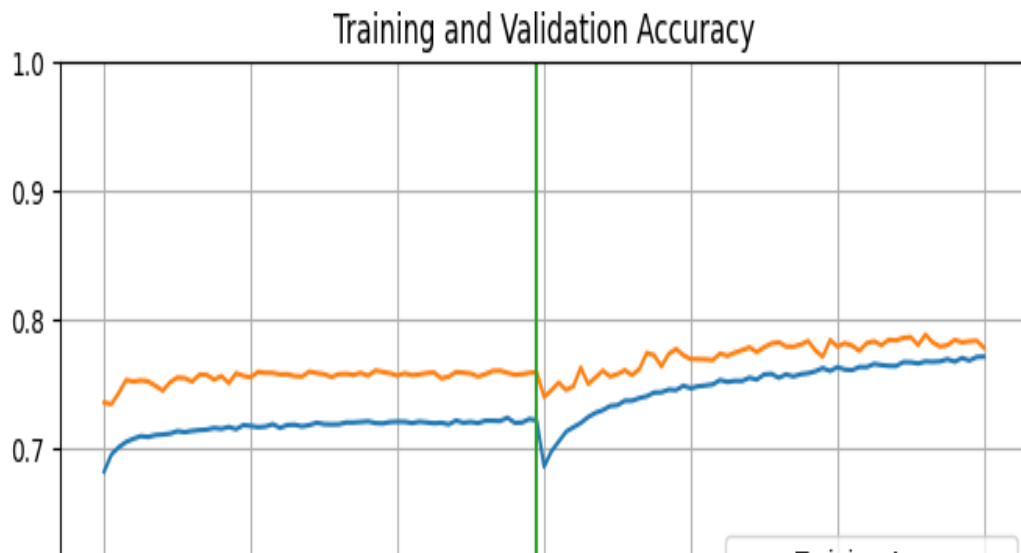


Figura 8.17: Evolución del accuracy en el entrenamiento del Clasificador F.

En la figura 8.18 se describe el umbral de discriminación del modelo.

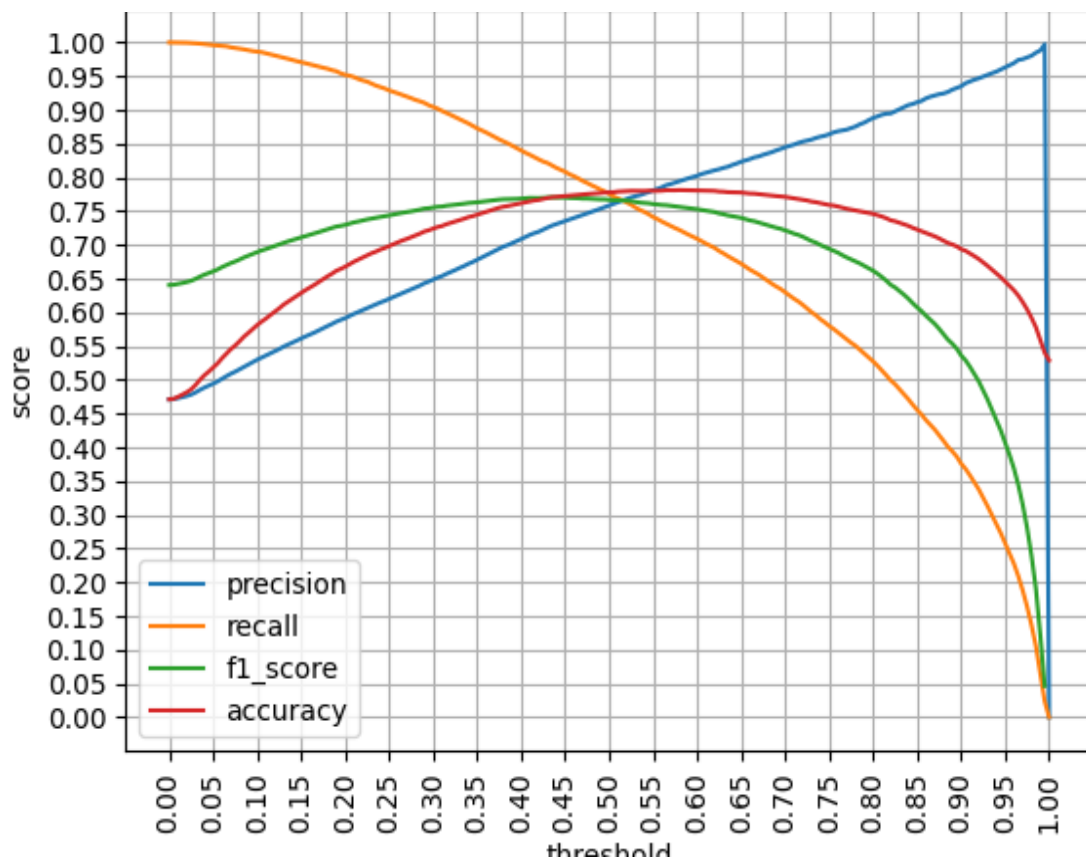


Figura 8.18: Umbral de discriminación Clasificador F.

8.7 Análisis de performance: Clasificador G

- Indicadores al finalizar 1º etapa de entrenamiento:
 - Error - Datos de entrenamiento: 0.5414
 - Error - Datos de validación: 0.4901
 - Indicadores de performance - Punto de corte: 0.5
 - *Accuracy* - Datos de entrenamiento: 0.7229
 - *Accuracy* - Datos de validación: 0.7590
- Indicadores al finalizar el entrenamiento del modelo:
 - Error - Datos de entrenamiento: 0.5143
 - Error - Datos de validación: 0.4715
 - Indicadores de performance - Punto de corte: 0.5
 - *Accuracy* - Datos de entrenamiento: 0.7413
 - *Accuracy* - Datos de validación: 0.7731
 - *Precision* - Datos de validación: 0.7663
 - *Recall* - Datos de validación: 0.7479
 - F1 - Datos de validación: 0.7570
 - Indicadores de performance - Punto de corte: 0.75
 - *Accuracy* - Datos de validación: 0.7431
 - *Precision* - Datos de validación: 0.8798
 - *Recall* - Datos de validación: 0.5281
 - F1 - Datos de validación: 0.6600
 - Indicadores de performance - Punto de corte: 0.90
 - *Accuracy* - Datos de validación: 0.6691
 - *Precision* - Datos de validación: 0.9516
 - *Recall* - Datos de validación: 0.3153
 - F1 - Datos de validación: 0.4737

En la figura 8.19 se visualiza la evolución del error en la etapa de entrenamiento del modelo de clasificación.

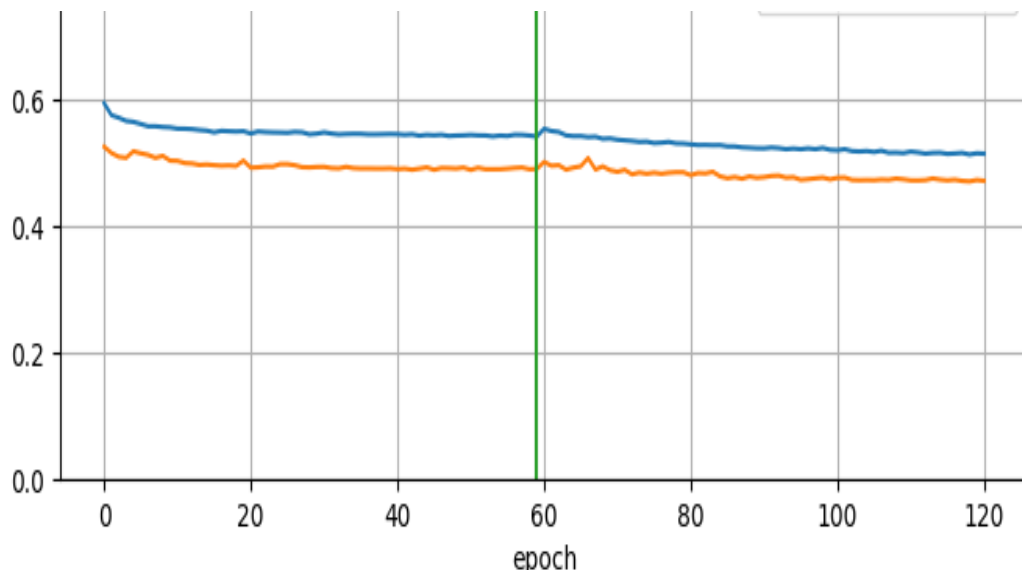


Figura 8.19: Evolución del error en entrenamiento del Clasificador G.

La figura 8.20 muestra la evolución del Accuracy del modelo considerando un punto de corte de 0.5 para la salida del modelo.

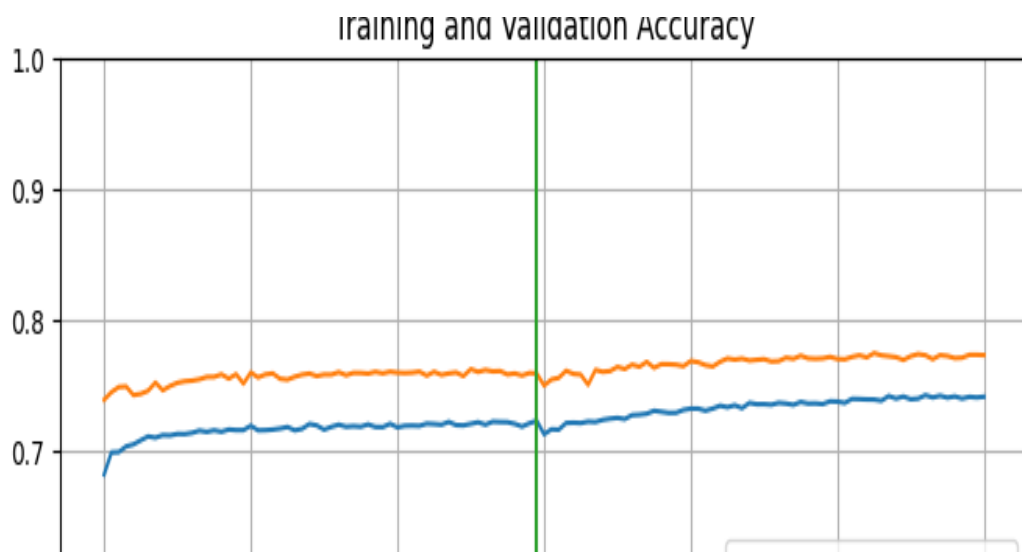


Figura 8.20: Evolución del accuracy en el entrenamiento del Clasificador G.

La figura 8.21 describe el umbral de discriminación del modelo.

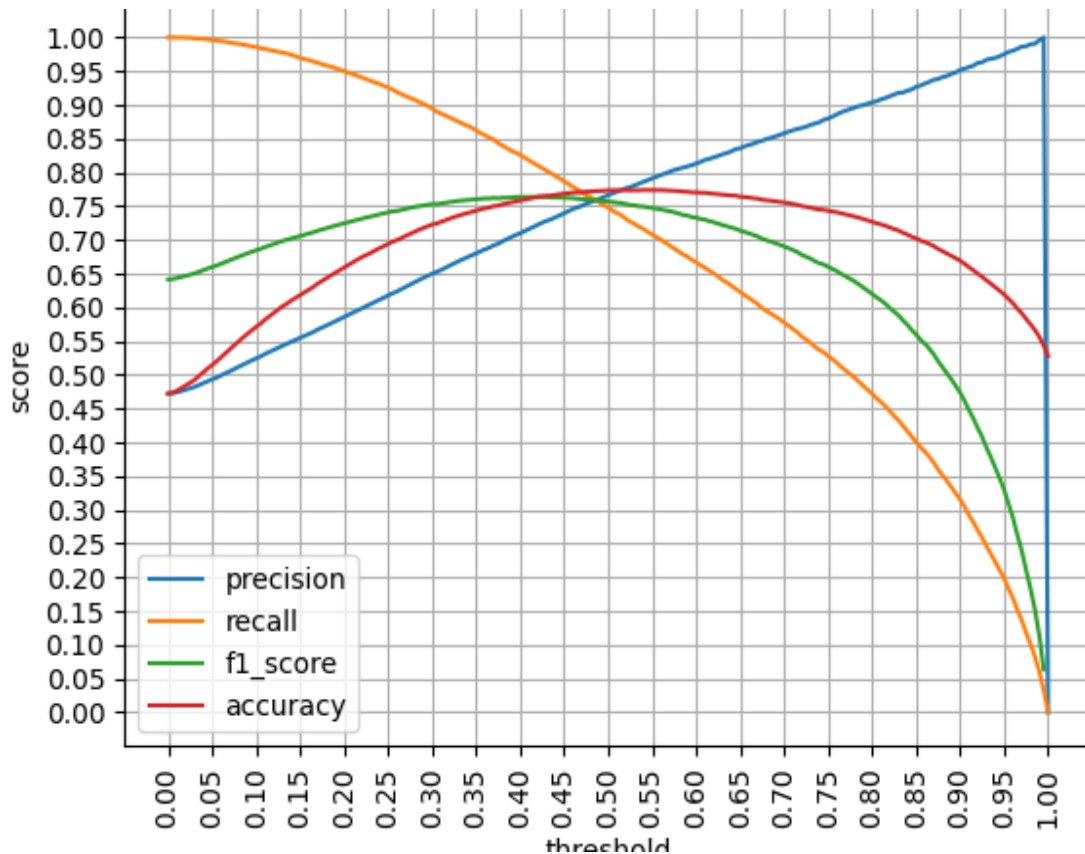


Figura 8.21: Umbral de discriminación Clasificador G.

8.8 Análisis de performance: Clasificador H

- Indicadores al finalizar 1° etapa de entrenamiento:
 - Error - Datos de entrenamiento: 0.5461
 - Error - Datos de validación: 0.4976
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7174
 - Accuracy - Datos de validación: 0.7557
- Indicadores al finalizar el entrenamiento del modelo:
 - Error - Datos de entrenamiento: 0.4867
 - Error - Datos de validación: 0.4621
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7413
 - Accuracy - Datos de validación: 0.7795
 - Precision - Datos de validación: 0.7848
 - Recall - Datos de validación: 0.7332
 - F1 - Datos de validación: 0.7581
 - Indicadores de performance - Punto de corte: 0.75
 - Accuracy - Datos de validación: 0.7411
 - Precision - Datos de validación: 0.8912

- *Recall* - Datos de validación: 0.5134
- F1 - Datos de validación: 0.6515
- Indicadores de performance - Punto de corte: 0.90
 - *Accuracy* - Datos de validación: 0.6669
 - *Precision* - Datos de validación: 0.9562
 - *Recall* - Datos de validación: 0.3074
 - F1 - Datos de validación: 0.4652

En la figura 8.22 se visualiza la evolución del error en la etapa de entrenamiento del modelo de clasificación.

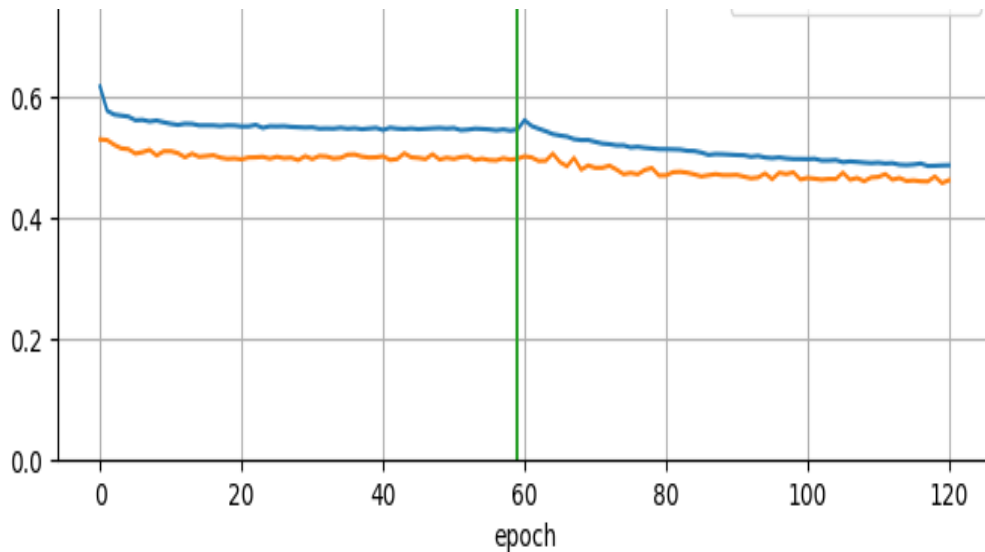


Figura 8.22: Evolución del error en entrenamiento del Clasificador H.

La figura 8.23 muestra la evolución del *Accuracy* del modelo considerando un punto de corte de 0.5 para la salida del modelo.

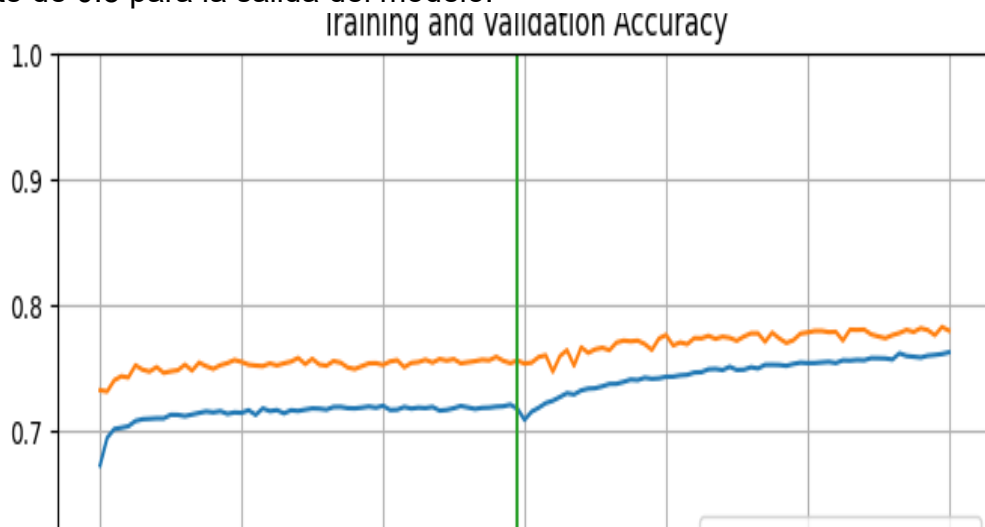


Figura 8.23: Evolución del accuracy en el entrenamiento del Clasificador H.

La figura 8.21 describe el umbral de discriminación del modelo.

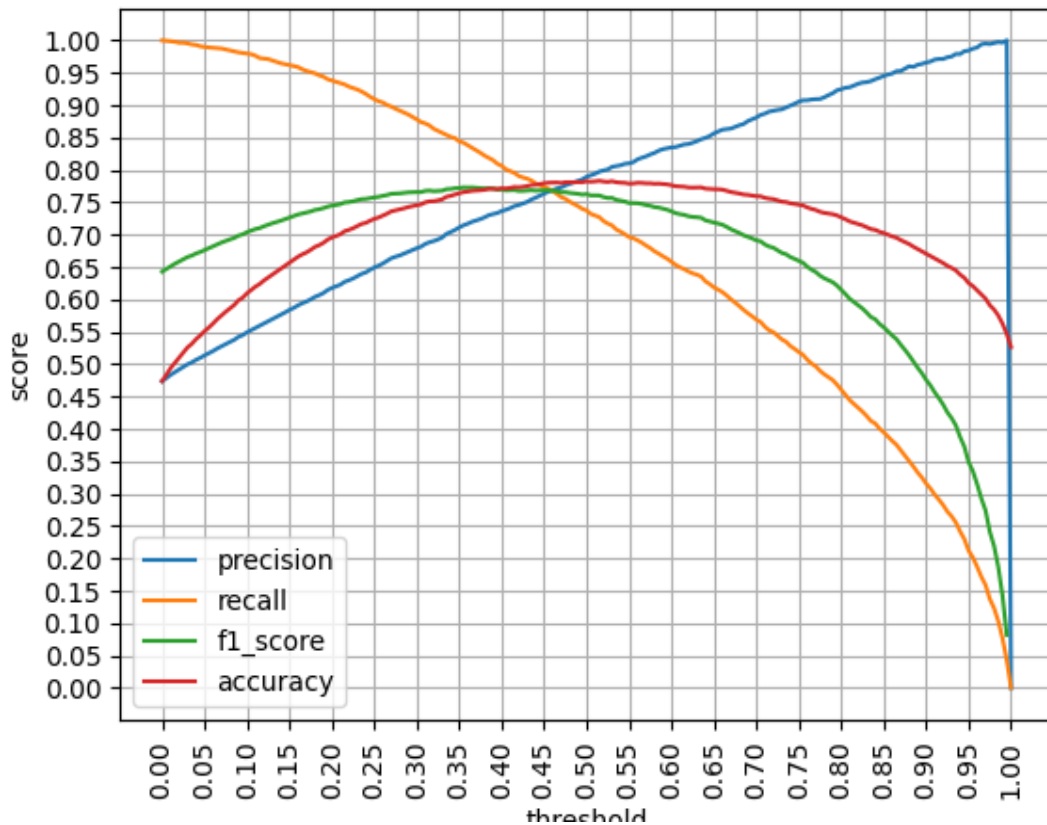


Figura 8.24: Umbral de discriminación Clasificador H.

8.9 Análisis de performance: Clasificador I

- Indicadores al finalizar 1° etapa de entrenamiento:
 - Error - Datos de entrenamiento: 0.5424
 - Error - Datos de validación: 0.4915
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7239
 - Accuracy - Datos de validación: 0.7580
- Indicadores al finalizar el entrenamiento del modelo:
 - Error - Datos de entrenamiento: 0.4840
 - Error - Datos de validación: 0.4636
 - Indicadores de performance - Punto de corte: 0.5
 - Accuracy - Datos de entrenamiento: 0.7621
 - Accuracy - Datos de validación: 0.7781
 - Precision - Datos de validación: 0.8002
 - Recall - Datos de validación: 0.7084
 - F1 - Datos de validación: 0.7515
 - Indicadores de performance - Punto de corte: 0.75
 - Accuracy - Datos de validación: 0.7325
 - Precision - Datos de validación: 0.9043

- *Recall* - Datos de validación: 0.4861
- F1 - Datos de validación: 0.6323
- Indicadores de performance - Punto de corte: 0.90
 - *Accuracy* - Datos de validación: 0.6561
 - *Precision* - Datos de validación: 0.9612
 - *Recall* - Datos de validación: 0.2848
 - F1 - Datos de validación: 0.4394

En la figura 8.25 se visualiza la evolución del error en la etapa de entrenamiento del modelo de clasificación.

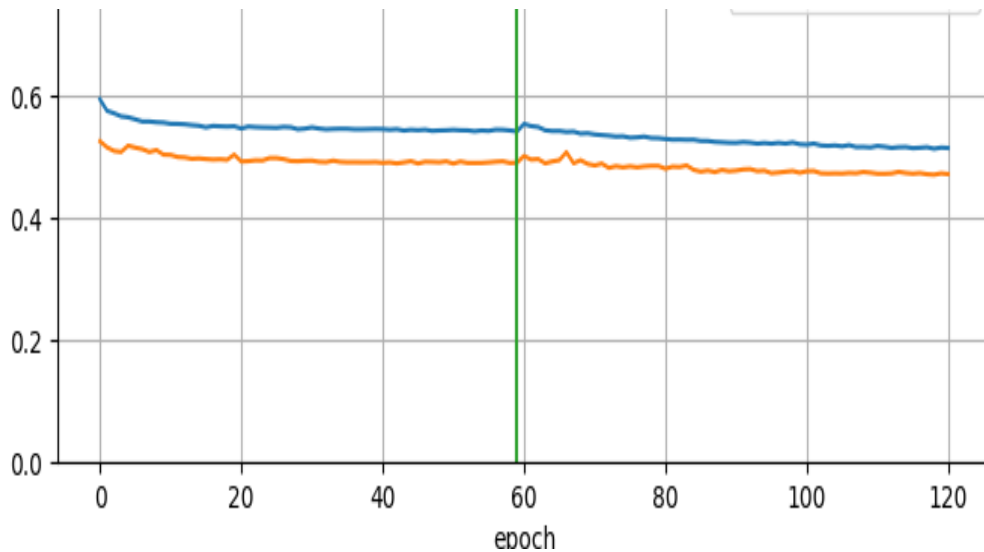


Figura 8.25: Evolución del error en entrenamiento del Clasificador I.

La figura 8.26 muestra la evolución del *Accuracy* del modelo considerando un punto de corte de 0.5 para la salida del modelo.

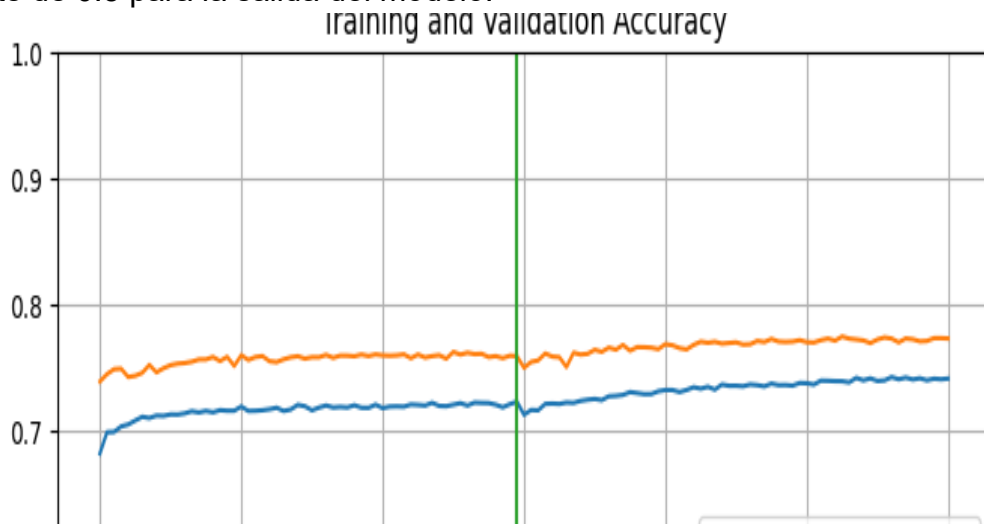


Figura 8.26: Evolución del accuracy en el entrenamiento del Clasificador I.

La figura 8.27 describe el umbral de discriminación del modelo.

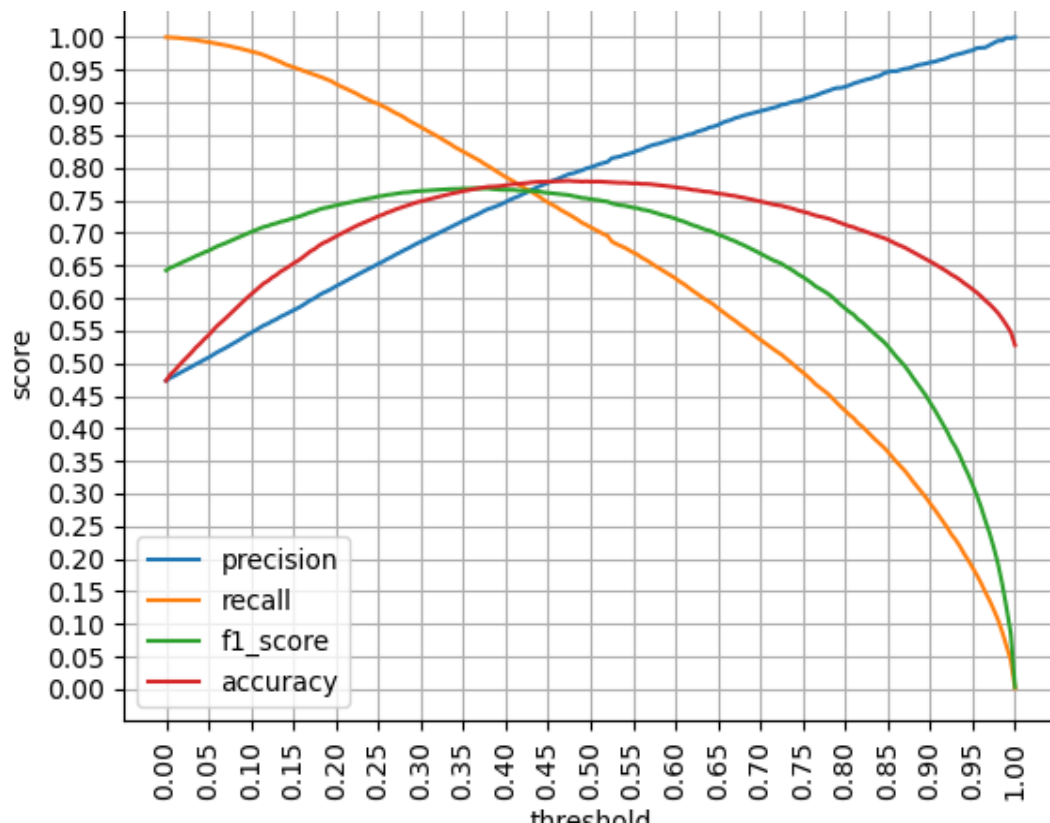


Figura 8.27: Umbral de discriminación Clasificador I.

9. Anexo D: Experimentos adicionales

9.1 Entrenamiento con diferentes valores de la capa *DropOut*

Se realizó un experimento con el objetivo de analizar si el uso de diferentes valores en la capa *DropOut* podía mejorar los indicadores de rendimiento del modelo. Para ello, se tomó el Clasificador I y se llevaron a cabo dos entrenamientos adicionales: uno con un *DropOut* del 10% y otro con un 5%.

Como puede observarse en la Figura 9.1, la modificación del porcentaje de *DropOut* a 10% y 5% resultó en una disminución de la performance, tanto en términos de PR-AUC como del valor máximo de F1.

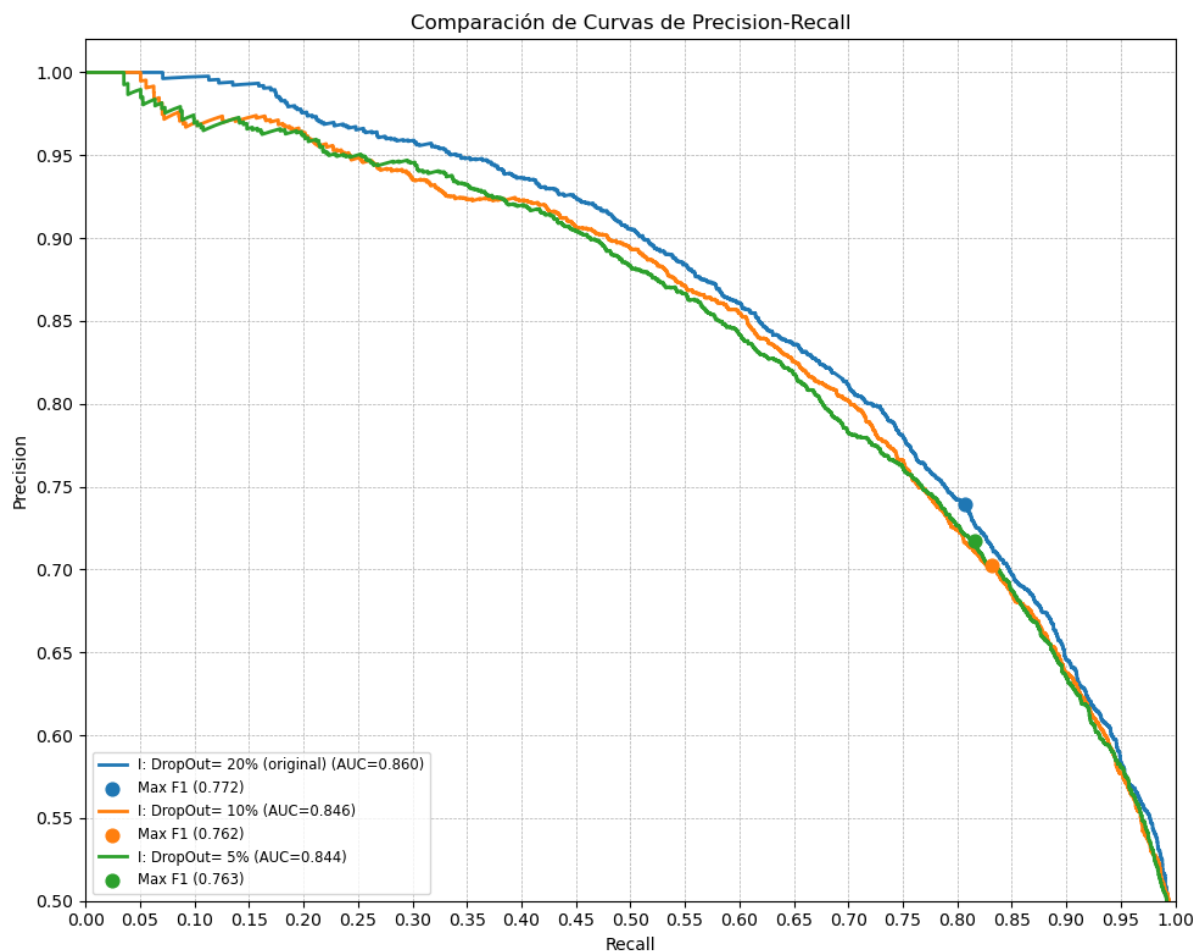


Figura 9.1: Comparación de curvas de Precision-Recall para el clasificador I con diferentes porcentajes de la capa Dropout.

9.2 Entrenamiento con diferentes valores de tasa de aprendizaje

Se tomó como referencia el Clasificador I para realizar un experimento que consistió en modificar la tasa de aprendizaje durante la segunda etapa de entrenamiento. Se evaluaron los valores 0.001, 0.00001 y 0.000001.

Como se puede observar en la Figura 9.2, la modificación de la tasa de aprendizaje no generó mejoras en la performance en comparación con el Clasificador I original, que utiliza una tasa de 0.0001. No se evidenciaron mejoras ni en el PR-AUC ni en el valor máximo de F1.

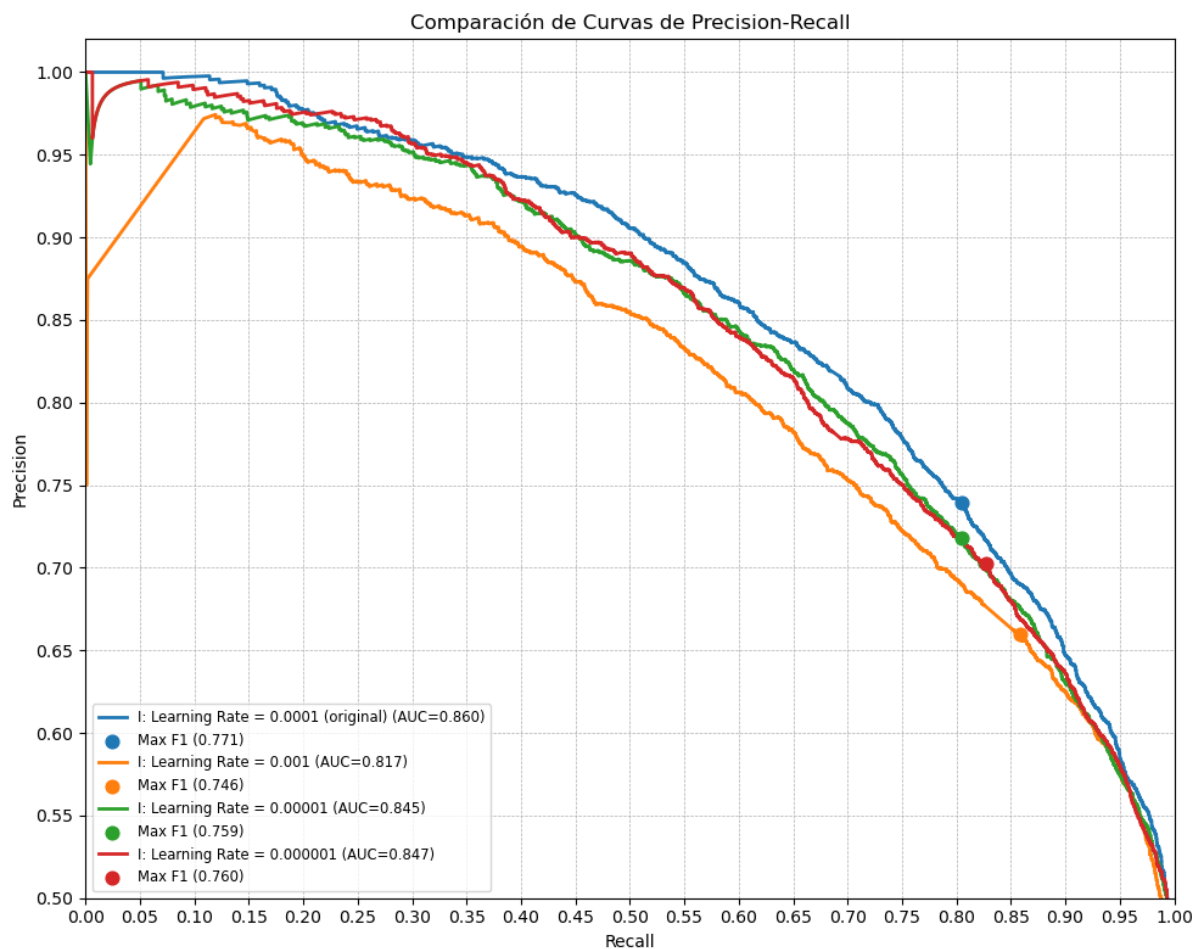


Figura 9.2: Comparación de curvas de Precision-Recall para el Clasificador I con diferentes tasas de aprendizaje.