Implementación de un prototipo de bajo costo para la detección de personas desde vehículos en movimiento, mediante el uso de redes neuronales profundas

Álvaro Javier Vargas Serrano

Carlos Alberto Archila Vargas

Trabajo de Grado para Optar al título de Ingeniero Electrónico

Director

Jeyson Arley Castillo Bohorquez

Ingeniero electrónico

Codirector

Jaime Guillermo Barrero Pérez

Magíster en Potencia eléctrica

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2023

Dedicatoria

Este trabajo va dedicado a mis abuelos Belén Carreño y Hermes Serrano, Eugenia Forero y Álvaro Vargas por ayudar a forjar en mi la capacidad para cumplir mis sueños, a mis padres Belsy Serrano Carreño y Alvaro Vargas Forero por su apoyo económico y familiar parental que me dieron la vida y los primeros pasos. A mis hermanos John Alexander Vargas Serrano y Julieth Katherine Vargas Serrano que me acompañaron en este camino largo educativo complementando en mí el apoyo emocional y económico, a mis compañeros de la universidad que me apoyaron especialmente a los que me aportaron y motivaron en el desarrollo de este trabajo de grado a Melissa Alvarez Anaya y Sebastian Campo Cruz, a mi compañero de trabajo de grado Carlos Alberto Archila por acompañarme en el desarrollo de este proyecto y complementar con las cosas buenas que soy.

Álvaro Javier Vargas Serrano

Este proyecto va dedicado a mis padres, Amparo Vargas Acevedo y Carlos Alberto Archila, por su tiempo, dedicación y sacrificios en mi formación académica y personal, a mi hermano Javier Orlando Martínez por su apoyo; también a mis amigos David Espíndola, Edwar Sequeda, Ian Díaz, Sebastián González, Sebastian Meza, Horacio Guzmán y Juliana Porras, por su tiempo y ocurrencias vividas a lo largo de esta aventura; al entrenador Diego Córdoba y la selección de ultimate frisbee de la universidad industrial de Santander, por aportar siempre una meta y objetivos para estudiar y mejorar cada día.

Carlos Alberto Archila Vargas

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Lista de Apéndices

Glosario

Convolución: es una operación matemática que consiste en tomar ”los píxeles cercanos” contra una matriz llamada kernel. El kernel se desplaza a través de todas las entradas realizando operaciones de izquierda a derecha y arriba a abajo para generar una nueva matriz resultante.

CPU: unidad central de procesamiento (Central Processing Unit), es un componente vital de tipo hardware básico de todo dispositivo que procesa datos y realiza procesos matemáticos.

FCN: redes completamente convolucionales (Fully Convolutional Network), una arquitectura de red neuronal que se basa en evitar el uso de capas densas para reducir los parámetros del modelo, emplean capas como convolución, agrupación y muestreo.

GPIO: entrada-salida de propósito general (General Purpose Input Output), es un serie de pines o conexiones que se pueden usar como entrada o salida para diferentes usos.

GPU: unidad central de procesamiento (Graphics processing unit), es la tarjeta gráfica que se encarga de mejorar el rendimiento y realizar cálculos complejos del ordenador.

Google Drive: es un servicio que almacena datos desde internet que proporciona Google, su versión gratuita permite la capacidad de almacenar 15 GB.

IDE: un entorno de desarrollo integrado (IDE) es una herramienta de software que se le suministra al usuario un entorno completo de programación para desarrollar software.

Matriz de confusión: es una métrica fundamental en redes neuronales de tipo clasificación, que evalúa el desempeño de la red a partir de los aciertos o errores en cada una de las clases presentadas a la misma.

Memoria RAM: memoria de acceso aleatorio (Random Access Memory) es un tipo de memoria que se encarga de almacenar temporalmente programas y procesos de ejecución del dispositivo electrónico.

OpenCV: visión artificial abierta (Open Computer Vision), es una librería de código abierto basada en visión artificial. Esta librería permite extraer modelos en 3D, encontrar imagenes muy similares, clasificar acciones humanas vídeos, entre otras aplicaciones.

Python: es un lenguaje de programación versátil de multiplataforma, que cuenta con una licencia de código abierto y que se puede usar en cualquier dispositivo o escenario donde se pueda implementar.

Tensorflow: es una biblioteca de código abierto desarrollada por Google que se usa para construir y entrenar redes neuronales.

Zero Padding: es una técnica que rellena ceros a partir de agregar filas y columnas utilizada en redes neuronales profundas para el procesamiento de imágenes, que ayuda a aumentar la precisión de la red neuronal. Generalmente, utiliza la operación de convolución.

Resumen

Título: Implementación de un prototipo de bajo costo para la detección de personas desde vehículos en movimiento, mediante el uso de redes neuronales profundas. *

Autores: Álvaro Javier Vargas Serrano y Carlos Alberto Archila Vargas. **

Palabras Clave: Detección de objetos, Accidentes de tránsito, Peatones, Redes Neuronales Profundas, Vehículos en Movimiento.

Descripción: Los accidentes de tránsito en muchos países representan la primera tasa de mortalidad, lo que pone de manifiesto la urgente necesidad de abordar este grave problema de salud pública. En los últimos años, se han desarrollado sistemas electrónicos que apoyan a los conductores de vehículos en movimiento con el fin de evitar accidentes. Por esta razón, este proyecto está orientado al desarrollo de un prototipo autónomo de bajo costo para la detección de peatones mediante el uso de redes neuronales profundas. El modelo está compuesto por una Raspberry Pi 4 modelo B, una batería y una cámara web; en cuanto al funcionamiento, está dividido en dos etapas que se repiten de manera cíclica, la primera, es la captura de una imagen mediante la cámara que se encuentra ubicada en la parte frontal del vehículo para la detección de peatones en ese instante, este proceso mediante una red neuronal de detección de objetos, la imagen guardada con los peatones detectados entra en la segunda etapa, donde se clasificara para activar una salida acústica según las regiones de la imagen donde se encuentren personas detectadas, permitiendo al conductor tomar acciones de frenado para evitar accidentes cuando se encuentren peatones al frente del vehículo.

---

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Jeyson Arley Castillo Bohorquez, Ing. Electrónico. Codirector: Jaime Guillermo Barrero, Msc, en Potencia eléctrica

Abstract

Title: Implementation of a low-cost prototype for detecting people from moving vehicles using deep neural networks. *

Authors: Álvaro Javier Vargas Serrano y Carlos Alberto Archila Vargas. **

Keywords: Object Detection, Traffic Accidents, Pedestrians, Deep Neural Networks, Moving Vehicles

Description: Road accidents in many countries represent the number one fatality rate, highlighting the urgent need to address this serious public health problem. In recent years, electronic systems have been developed to support drivers of moving vehicles in order to avoid accidents. For this reason, this project is aimed at developing a low-cost autonomous prototype for pedestrian detection using deep neural networks. The model is composed of a Raspberry Pi 4 model B, a battery and a webcam; As for the operation, it is divided into two stages that are repeated cyclically, the first is the capture of an image by the camera that is located at the front of the vehicle for pedestrian detection at that moment, this process through a neural network object detection, the image saved with the detected pedestrians enters the second stage, where it is classified to activate an acoustic output according to the regions of the image where people are detected, allowing the driver to take braking actions to avoid accidents when pedestrians are in front of the vehicle.

---

* Bachelor Thesis

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Jeyson Arley Castillo Bohorquez, Ing. Electrónico. Codirector: Jaime Guillermo Barrero, Msc, en Potencia eléctrica.

Introducción

En las últimas décadas, el hombre ha experimentado un cambio profundo en el ámbito profesional y cotidiano debido al constante avance de la tecnología (Mazur, 2023). La inteligencia artificial (IA) es un pilar fundamental a estos cambios cotidianos debido a su implementación en múltiples áreas industriales (Xiong and Zou, 2022). Las técnicas más usadas dentro del tema de inteligencia artificial son las redes neuronales profundas. Estas redes neuronales profundas son un algoritmo computacional que consta de neuronas artificiales que se interconectan entre sí para transmitir y procesar la información, obteniendo una respuesta ante una entrada (ScienceDirect, sf). Entre las aplicaciones más destacadas se encuentran la detección y clasificación de imágenes. En el presente proyecto de grado se demostró como las redes neuronales específicamente las redes neuronales convolucionales, son eficaces a la hora de detectar y clasificar imágenes. Esto se evidenció en el caso de la detección de peatones para la precaución del conductor dentro de vehículos en movimiento con el fin de evitar accidentes de tránsito. Una red neuronal de clasificación requiere de una base de datos robusta que le permita diferenciar características de los datos. En este proyecto se construyó la base datos con dos directorios cada una, las cuales representan las clases para la red de clasificación. Para la tarea de entrenar la red de entrenar y validar la red neuronal de clasificación, se usó el entorno de ejecución llamado Google Colaboratory. Como resultado del proyecto se obtuvieron dos modelos de redes neuronales, los cuales se implementaron en una Raspberry Pi 4 Modelo B. El objetivo era crear un prototipo de bajo costo para peatones detección y clasificación de imágenes y para evaluar el rendimiento del dispositivo en tiempo real.

1. Objetivos

1.1. Objetivo general

Implementar un prototipo para la detección de personas desde vehículos de transporte mediante el uso de redes neuronales profundas.

1.2. Objetivos específicos

Seleccionar una base de datos con imágenes específicas de uso libre para la validación y entrenamiento.

Seleccionar un modelo de red neuronal profunda con respecto a las topologías disponibles para la detección de objetos y clasificación de personas basados en el estado del arte más reciente.

Diseño de prototipo de bajo costo teniendo en cuenta los requerimientos de memoria y procesamiento para la ejecución de la red neuronal.

Evaluar el desempeño del prototipo implementado.

2. Inteligencia artificial

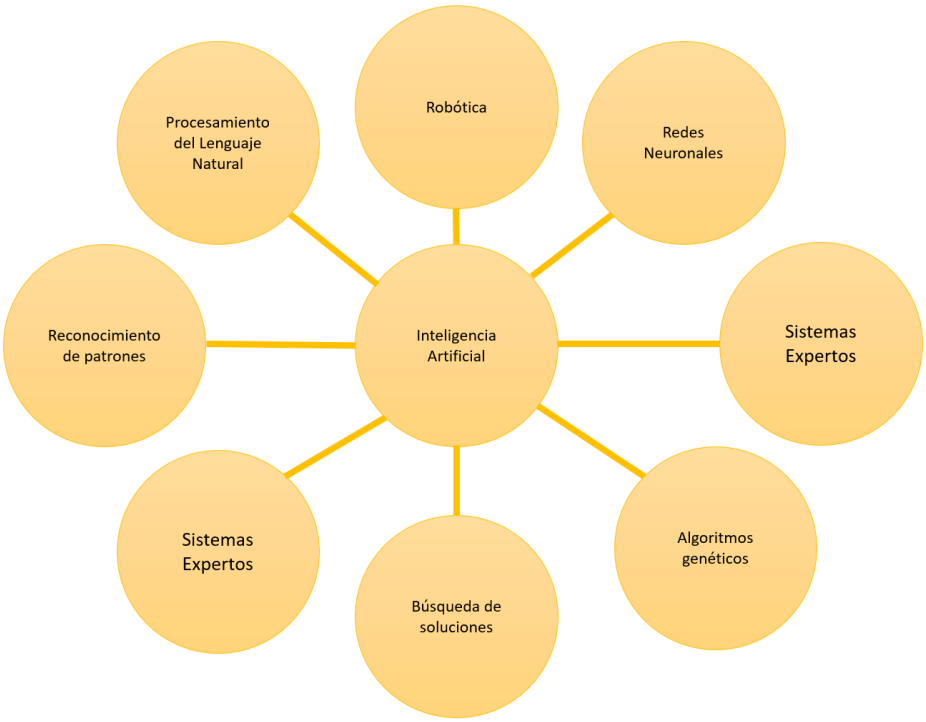
La inteligencia artificial es una disciplina que se encarga de simular el comportamiento humano en ámbitos de la toma de decisiones, comprensión, procesos de percepción y principalmente de las facultades intelectuales del ser humano, para realizar tareas similares mediante máquinas electrónicas (Benitez et al., 2014) (SGMA, 2017). La inteligencia artificial busca diversas formas de aportar soluciones a varios problemas, entre los que destacan la minería de datos, el diagnóstico médico, la robótica, entre otras aplicaciones (Benitez et al., 2014); usando técnicas que aportan elementos fundamentales en las áreas donde se utiliza la inteligencia artificial (ver Fig. 1). Actualmente, la inteligencia artificial ha tenido una estrecha relación en la investigación basada en temas de análisis de vídeo y imágenes para la detección de objetos (Zhao et al., 2019).

2.1. Redes neuronales

Las redes neuronales artificiales son sistemas que intentan emular el comportamiento de las neuronas biológicas, como la capacidad de memorizar y asociar hechos, basados en representaciones matemáticas de la sinopsis que ocurre en el cerebro humano (Basogain, 2008), además están constituidas por grupos de neuronas interconectadas llamadas capas (Basogain, 2008).

Las redes neuronales están compuestas por unidades elementales llamadas neuronas, compuestas por la entrada, los pesos, el sesgo, la función suma, función de activación y la señal de salida correspondiente (ver Fig. 2) (Iqbal, 2021). La neurona artificial es un modelo binario determinado mediante la función de activación (ver Fig. 2) (Gomez et al., 1994). Para obtener la salida, se calcula la suma total de cada unidad de sus entradas para finalmente aplicar la función de acti-

Figura 1. Técnicas de la Inteligencia Artificial.

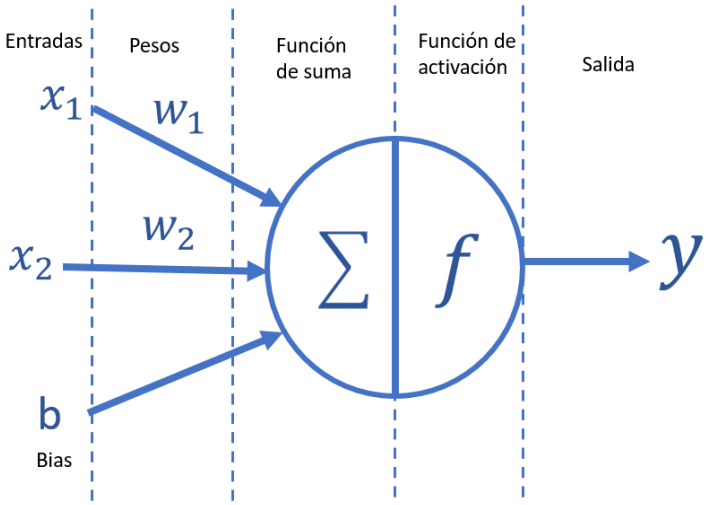


Nota: Adaptado de (SGMA, 2017).

vación y realizar una comparación a la suma desarrollada (Russell and Noving, 2004). Si la suma supera al valor del umbral de la función de activación la neurona toma la decisión de activarse en caso contrario se inactiva (Gomez et al., 1994).

La distribución de neuronas se realiza formando tres niveles (ver Fig. 3). En cada nivel se distribuyen las neuronas con el objetivo de proveer una organización jerárquica. En el primer nivel se encuentra la capa de entrada encargada de recibir la información proveniente de fuentes externas. En el segundo nivel se encuentra las capas ocultas que son internas a la estructura de la red, no tienen contacto con el exterior y el número de capas ocultas puede estar entre cero a un

Figura 2. Modelo de una neurona artificial.



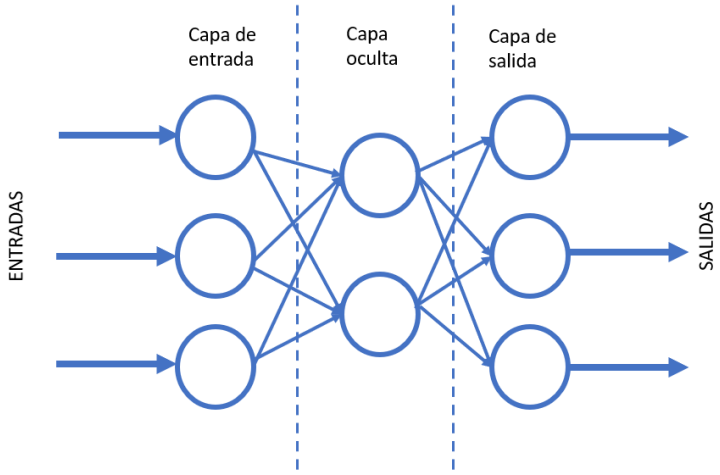
Nota: Adaptado de (Yang et al., 2020).

número elevado. Finalmente, la tercera capa llamada capa de salida (Caicedo and López, 2017). Esta información proveniente de fuentes externas siendo obtenida por la capa de entrada depende del tipo de aplicación que se desea realizar; en el caso de detección de objetos son imágenes (Zhao et al., 2019).

Las neuronas artificiales cuando son distribuidas por la estructura de la red interactúan entre sí para realizar el procesamiento necesario a la información (ver Fig. 2). Esta agrupación estructurada es lo que forma la red neuronal artificial (ver Fig. 3).

2.1.1. Aprendizaje de la red neuronal. El aprendizaje de la red neuronal es el proceso de ajustar valores de los parámetros de pesos y sesgos que están asociados con la interacción de cada neurona, creando, modificando y destruyendo las conexiones entre ellas en respuesta a la información recibida, con el objetivo de que la red neuronal adquiera conocimiento de alguna

Figura 3. Estructura de una red neuronal artificial.



Nota: Adaptado de (Alemán, 2017).

habilidad a partir del estudio o la experiencia (Basogain, 2008) (Caicedo and López, 2017). Existen dos mecanismos de aprendizaje utilizados en redes neuronales, conocidos como aprendizaje supervisado y aprendizaje no supervisado. (Ramón and José, 1995).

2.1.1.1. Aprendizaje supervisado. El aprendizaje supervisado consiste en realizar un proceso a la red presentándole una entrada determinada, calcular la salida de la red y compararla con la salida deseada, modificando los pesos por medio de un error resultante, si esta salida no es la deseada se modifican los pesos de las conexiones (Basogain, 2008). Esta comparación se realiza por la presencia de un agente externo que controla el proceso de entrenamiento a partir del error, con el fin de que la salida de la red neuronal se aproxime más a la salida deseada (Ramón and José, 1995). En el aprendizaje supervisado existen tres tipos, aprendizaje por corrección de error que consiste en ajustar los pesos de las conexiones de la red por medio de la diferencia de valores

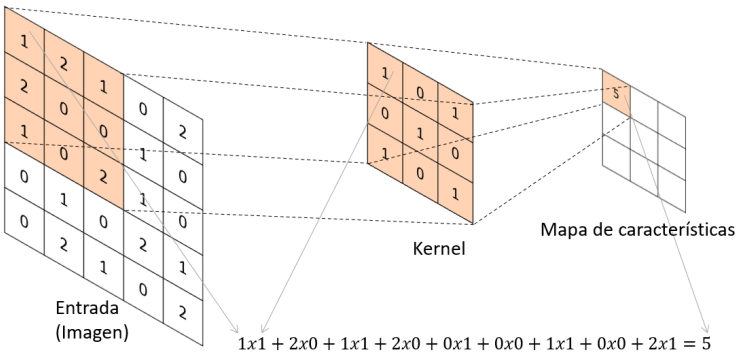
deseados y obtenidos en el resultado de la red (Ramón and José, 1995); aprendizaje por refuerzo el cual se trata de un aprendizaje más lento que el anterior, consiste en indicar por una "señal de refuerzo"si la salida obtenida en la red se ajusta a la deseada y en función de ello se ajustan los pesos (Ramón and José, 1995). Finalmente el tercer tipo de aprendizaje es el estocástico este realiza cambios aleatorios en los valores de pesos de las conexiones de la red y evaluar el efecto en la salida deseada (Ramón and José, 1995).

2.1.1.2. Aprendizaje no supervisado. El aprendizaje no supervisado no cuenta con un agente para ajustar los pesos de las conexiones de las neuronas a partir de los vectores de salidas únicamente consiste en vectores de entrada. Este ajusta los valores de los pesos obteniendo resultados consistentes en el vector de salida (Basogain, 2008). Existen algoritmos de aprendizaje no supervisado, considerando el algoritmo de Hebb (1994), este extrae las características de los datos de entrada y ajusta los pesos para obtener valores de salida coherentes (Basogain, 2008).

2.2. Redes neuronales convolucionales

Las redes neuronales convolucionales son capas de redes neuronales que procesan datos a partir de la operación matemática convolución (Goodfellow et al., 2016). Surgen de la función visual del cerebro humano y se usan para el reconocimiento de imagenes desde 1980 (Suárez et al., 2017) (Aurelien, 2019). Según (Yamashita et al., 2018) la operación convolución se realiza mediante el uso de dos matrices. La imagen se representa de manera numérica por medio de una matriz y se opera con la matriz kernel mediante la operación convolución (ver Fig. 4). El kernel es una matriz de menor dimensión que la imagen de entrada, el primer paso para realizar la convolución se muestra en (ver Fig. 4). Después de realizar la operación la matriz kernel recorre la entrada en

Figura 4. Ejemplo del proceso matemático de una convolución.

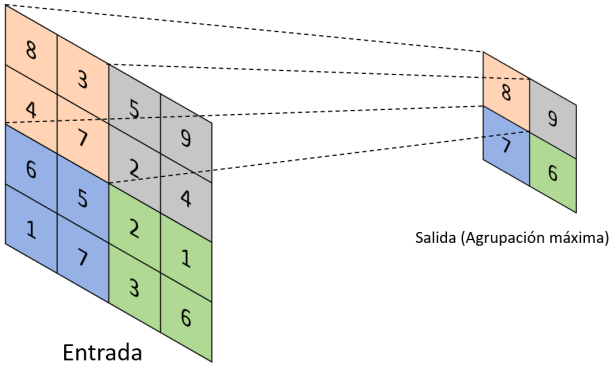


Nota: Adaptado de (Yamashita et al., 2018).

dirección izquierda a derecha y de arriba a abajo con un determinado paso, cuando la matriz kernel analiza el ancho de la entrada al final de la derecha esta salta a la izquierda hasta generar una matriz llamada mapa de características o matriz de convolución (Yamashita et al., 2018).

Las redes neuronales convolucionales vienen acompañadas, seguidamente con capas de agrupación. Estas capas se encargan de reducir el tamaño espacial de la matriz de convolución con la finalidad de extraer características dominantes de la imagen y disminuyen el costo computacional requerido para realizar las operaciones en un sistema embebido (Yamashita et al., 2018). Existen dos tipos de capas de agrupación que son agrupación máxima y agrupación promedio global; la agrupación máxima (ver Fig. 5) extrae el valor máximo de la matriz de convolución cubierta por el kernel y la agrupación promedio global extrae el valor promedio de todos los valores que cubren la matriz llamada kernel (Yamashita et al., 2018).

Figura 5. Ejemplo del proceso matemático de la agrupación máxima.



Nota: Adaptado de (Yamashita et al., 2018).

2.3. Arquitecturas de redes convolucionales

Para el desarrollo de este proyecto, utilizamos las arquitecturas de redes neuronales convolucionales YOLOv7-tiny y MobileNetV2, que se describen a continuación.

2.3.1. MobilenetV2. Según (Kumar, 2023), Mobilenet es de los modelos preentrenados más populares de última generación en clasificación de imagenes, se basa en redes convolucionales profundas. MobilenetV2 posee alrededor de 2 millones de parámetros en su modelo de red neuronal y es más ligera en comparación a MobileNet que cuenta con más de 3 millones de parámetros (Sandler et al., 2019). Esta red introduce conceptos como capas cuello de botella (*bottleneck*), comúnmente usadas para obtener una representación de la imagen de entrada de tamaño reducido (Sandler et al., 2019), además incluye capas de convolución, capas de *zero padding*, capas de convolución *depthwise*, una capa de *average pooling* y una capa *fully connected* (Chung et al., 2020). Su arquitectura se muestra en la tabla 1.

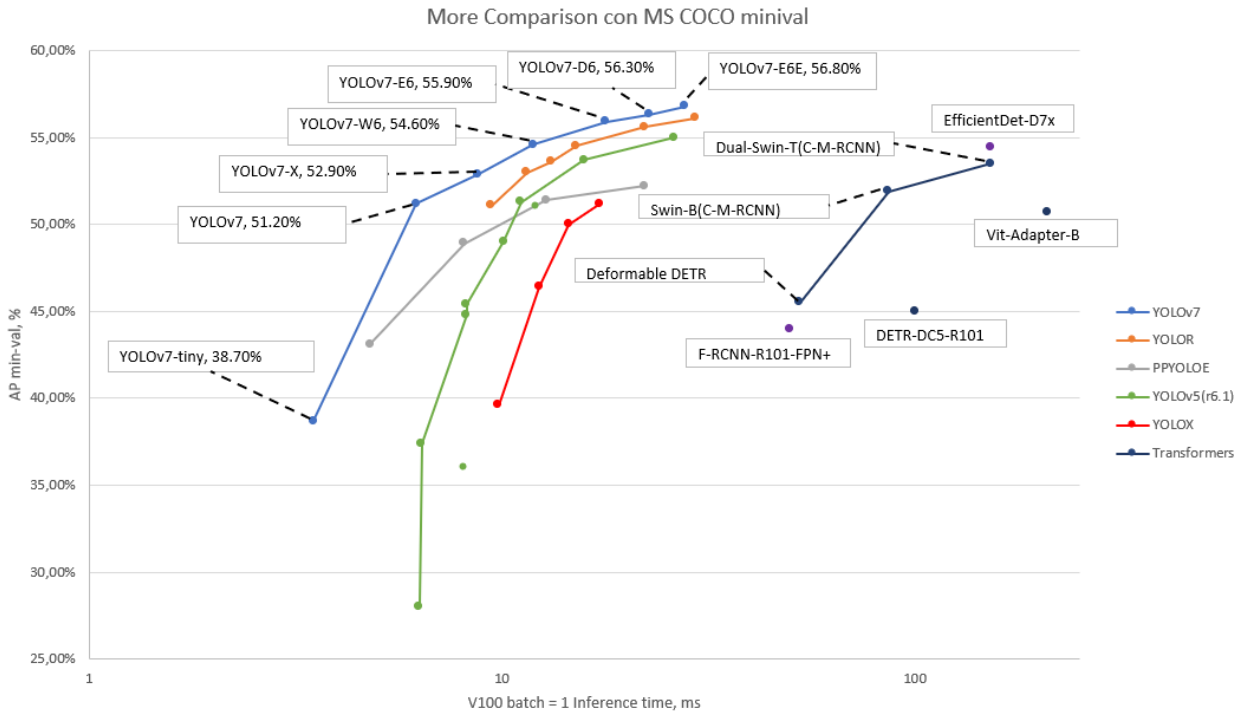
Tabla 1
Arquitectura de la red MobiletNetV2.

Entrada	Operador
224^2 x 3	conv2d
112^2 x 32	bottleneck
112^2 x 16	bottleneck
56^2 x 24	bottleneck
28^2 x 32	bottleneck
14^2 x 64	bottleneck
14^2 x 96	bottleneck
7^2 x 160	bottleneck
7^2 x 320	conv2d 1x1
7^2 x 1280	avgpool 7x7
1^2 x 1280	conv2d 1x1

Nota: Adaptado de (Sandler et al., 2019).

2.3.2. Detección de objetos. En la actualidad, la detección de objetos consiste en ubicar múltiples objetos en imágenes o vídeos y establecer etiquetas para cada uno de ellos (Zhao et al., 2019). Hace algunos años, un enfoque común para resolver el problema de detectar objetos mediante imágenes era tomar un modelo de redes convolucionales que estaba entrenada para clasificar y ubicar un solo objeto. Se hacía deslizar la red por medio de una ventana variable en la imagen para detectar múltiples objetos de la misma clase (Aurelien, 2019). Este proceso funcionaba bastante bien; sin embargo, el tiempo de inferencia era bastante largo, por lo que era poco eficiente. Recientemente, se ha logrado reducir este tiempo de inferencia usando redes completamente convolucionales (FCN). La arquitectura de estos modelos es que poseen capas convoluciones y son más eficientes que las redes convolucionales de clasificación, ya que el procesamiento realizado

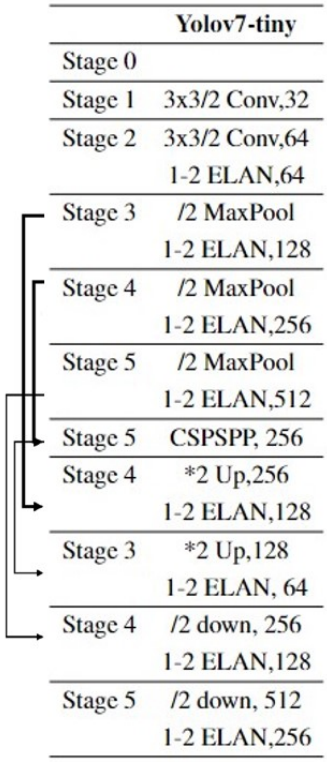
Figura 6. Comparación de los tiempos de inferencia de YOLOv7-tiny con otros detectores de objetos usando una GPU Tesla V100.



Nota: Adaptado de (Wang et al., 2022a).

para detectar objetos en una imagen lo pueden hacer una sola vez. De hecho YOLO (*You Only Look Once*) es de los modelos más populares por la alta eficiencia en sistemas computacionales surgido en el año 2015 (Aurelien, 2019) . Actualmente, se encuentran una enorme cantidad de detectores de objetos basados en YOLO de los cuales se encuentran YOLO, YOLOv2, YOLOv3, YOLOv4, PP-YOLO, PP-YOLOE, YOLOR, YOLOX, YOLOv5, YOLOv6, YOLOv7. Basado en el nuevo estado del arte se seleccionó el modelo YOLOv7-tiny en el desarrollo del proyecto principalmente por su alta rapidez y buena precisión para la detección de objetos como se presenta en la Figura 6.

Figura 7. Arquitectura de la red Yolov7-tiny.



	Yolov7-tiny
Stage 0	
Stage 1	3x3/2 Conv,32
Stage 2	3x3/2 Conv,64
	1-2 ELAN,64
Stage 3	/2 MaxPool
	1-2 ELAN,128
Stage 4	/2 MaxPool
	1-2 ELAN,256
Stage 5	/2 MaxPool
	1-2 ELAN,512
Stage 5	CSPSPP, 256
Stage 4	*2 Up,256
	1-2 ELAN,128
Stage 3	*2 Up,128
	1-2 ELAN, 64
Stage 4	/2 down, 256
	1-2 ELAN,128
Stage 5	/2 down, 512
	1-2 ELAN,256

Nota: Adaptado de (Wang et al., 2022a).

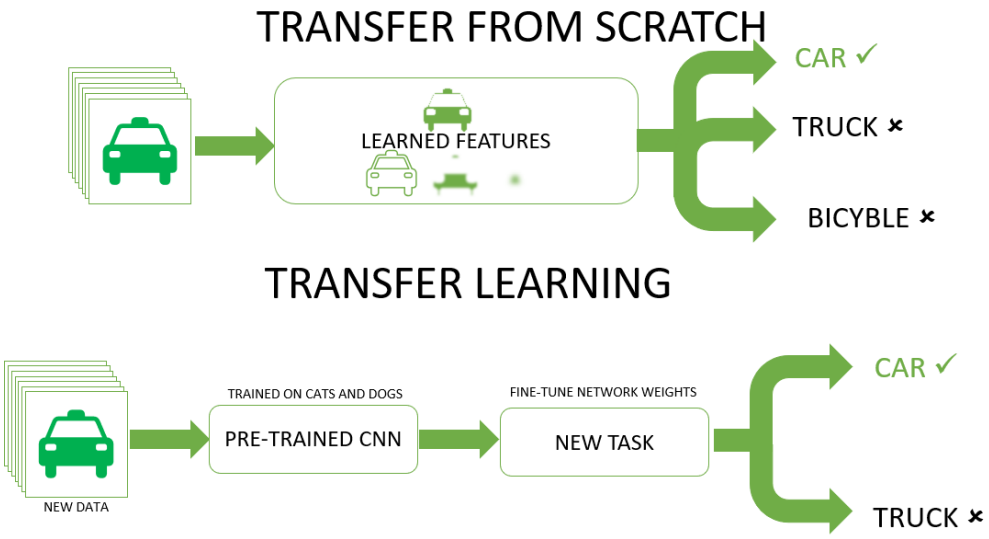
2.3.2.1. YOLOv7-tiny. Es una arquitectura de detección de objetos que contiene 6.2 millones de parámetros, lo que la hace más ligera en comparación con el modelo YOLOv7 que posee 36.9 millones de parámetros (Wang et al., 2022a). Esta arquitectura incluye una capa de red CSPSPP (*Cross stage partial–Spatial pyramid pooling*), capas de agrupación máxima, capas de red de agregación eficiente ELAN (*Efficient layer aggregation networks*), y capas de muestreo ascendente y descendente. El objetivo de la red ELAN es solucionar el problema de la perdida gradual de convergencia del modelo al realizar su escalado, diseñando así una arquitectura de agregación de capas con rutas de propagación de gradiente eficientes. ELAN está compuesta principalmen-

te de VoVNet combinado con CSPNet (Wang et al., 2022b) y la capa de CSPSPP, esta última se implementa para mejorar la capacidad de extraer y fusionar mejor la información deseada de la imagen (Ju et al., 2023). La arquitectura de YOLOv7-tiny se presenta en la Figura 7.

2.4. Transfer learning

El *transfer learning* es una técnica ampliamente utilizada en el campo de las redes neuronales, que involucra la adaptación y ajuste de una arquitectura de red neuronal preentrenada para abordar un problema específico, con el objetivo de aprovechar el conocimiento previo adquirido y aplicarlo a un nuevo problema relacionado. (Brownlee, 2019).

Figura 8. Transfer learning.



Nota: Adaptado de (Shah, 2019).

En la Figura 8, se ilustra claramente cómo el modelo que se entrena en *transfer learning* la cual comparte y transfiere conocimientos a través de los diferentes componentes, como los

pesos y las características, hacia el sistema de aprendizaje de la nueva tarea. Este intercambio de conocimientos permite una mejora sustancial en el rendimiento y la eficiencia del modelo en la nueva tarea específica. Esta técnica brinda la ventaja de disminuir el proceso de entrenamiento de la red neuronal y adaptarse al nuevo problema (Shah, 2019).

3. Sistemas de ejecución

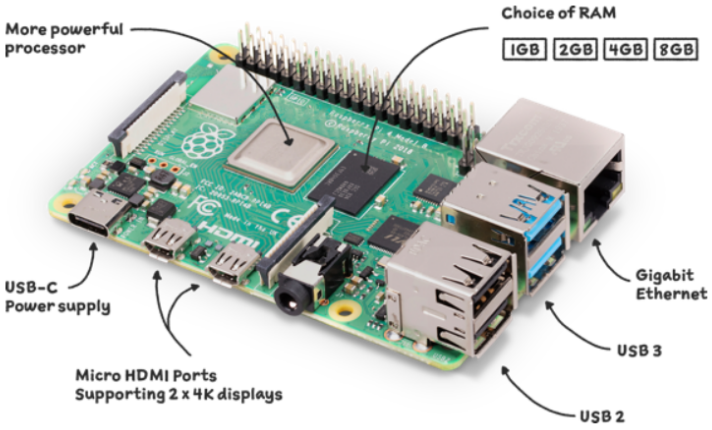
Para el desarrollo del prototipo se utilizó el lenguaje de programación Python, un lenguaje versátil, de alto nivel, de sintaxis clara y de uso libre; por otro lado, para el proceso de entrenamiento se usó Google Colaboratory, también conocido como Colab, es un servicio basado en la nube que facilita la investigación en aprendizaje automático, no necesita configuración y proporciona acceso gratuito a una potente GPU (Montoro, 2012) (Carneiro et al., 2018). A continuación, se muestran las características de los elementos que componen el prototipo.

3.1. Raspberry Pi 4 modelo B

La Raspberry Pi es una familia de computadoras de placa única (SBC) del tamaño de una tarjeta de crédito desarrollada en el Reino Unido por la Fundación Raspberry Pi con el objetivo de promover la enseñanza informática básica en los países en desarrollo proporcionando una plataforma de bajo costo (Ashwin, 2022), esta mini computadora de placa única proporciona unos pines abreviados como GPIO que se pueden usar para controlar componentes electrónicos para computación física (Raji et al., 2019). Existen dos tipos de modelo, el modelo A y el modelo B. La principal diferencia entre estos dos tipos de modelos es el puerto de bus serie universal (USB) y la ventaja del modelo B es que incluye puerto *ethernet* (Raji et al., 2019).

Para el desarrollo de este proyecto se utilizó la Raspberry Pi 4 modelo B de 4 GB de memoria RAM, con especificaciones que se muestran en la tabla 2. Además, se incorporó una cámara Webcam diseñada por *Drive Free Desing* (UVC) con una batería (*Power Bank*) diseñada por Kalley. Las especificaciones de los periféricos se muestran en las tablas 3 y 4 respectivamente.

Figura 9. Modelo de la Raspberry Pi.



Nota: Tomado de (Raspberry, sf).

Tabla 2
Información de la placa Raspberry Pi 4 modelo B.

Especificaciones	
Architecture	ARMv8
SoC broadcom	BCM2711
CPU	Quad core Cortex-A72 (ARM v8) 64-bit SoC @1.5GHz
GPU	Broadcom VideoCore IV
Memory	2GB, 4GB or 8GB LPDDR4-3200 SDRAM
USB	2 USB 3.0 ports, 2 USB 2.0 ports
Video output	2 × micro-HDMI ports (up to 4kp60 supported)
	2-lane MIPI DSI display port
On-board storage	Micro SDHC slot
On-board network	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless
	Bluetooth 5.0, BL
	Gigabit Ethernet
Power source	3A 5V via MicroUSB

Nota: Adaptado de (Ashwin, 2022).

Figura 10. Modelo de la Cámara.

Nota: Tomado de (Basery, sf).

Tabla 3
Información de la cámara Webcam.

Especificaciones	
Driver type	Driver-free (plug and play) USB
Line length	120 cm
SCINTILLATION control	50Hz, 60Hz
Photo Format	Bmp, jpg
Interface type	USB2.0
Sensor type	CMOS
Built-in	Sound-absorsing and noise-reducing microphone
Contrast optimization engine	Contrast Balance
Focusing range	20 mm extreme
VM engine	180M / 8MP high speed proccessor
Resolution	1280x720p
	640x480p

Nota: Elaboración propia.

Figura 11. Modelo de la Batería.

Nota: Tomado de (Kalley, sf)

Tabla 4
Información de la batería.

Especificaciones	
Maximum capacity	10000 mAh
Micro USB input	5V DC 2A
Input Type C	5V DC 2.6A
Output Type C	5V DC 3A
Output USB C	5V DC 3A

Nota: Elaboración propia.

Tabla 5
Costos del proyecto.

Costos del proyecto	
Raspberry Pi 4 mobelo B	$487.000
Cámara Webcam	$40.000
Batería	$90.000
Cable tipo C	$25.000
Buzzer	$3.500
Costos adicionales	$58.500
Total	**$704.000**

Nota: Elaboración propia.

4. Metodología de Desarrollo

Con el propósito de desarrollar la implementación de un prototipo de detección de peatones desde vehículos en movimiento utilizando redes neuronales profundas, se procedió a adquirir una base de datos de imágenes específicas de peatones. Estas imágenes se dividieron en conjuntos de datos para entrenamiento, validación y prueba respectivamente. Posteriormente, se adaptó un código de red neuronal diseñado para la detección de objetos, el cual se encarga de localizar y delinear a los peatones presentes en una imagen, para así proporcionar la entrada necesaria para la siguiente red neuronal, conocida como modelo de clasificación de imágenes.

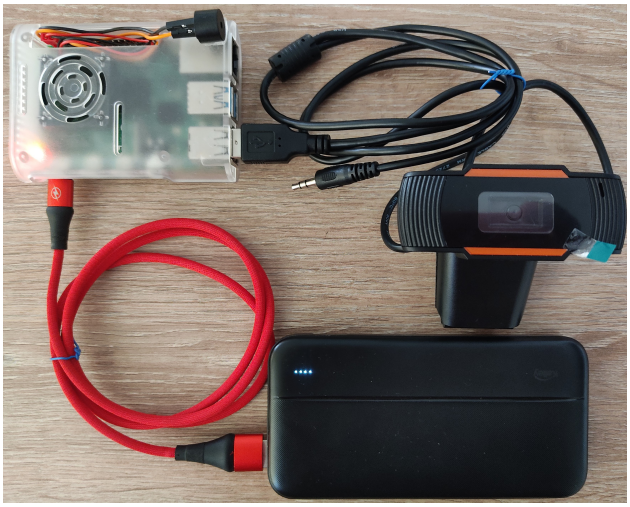
El modelo de clasificación se construyó utilizando el enfoque de *transfer learning*; este proceso permitió aprovechar el conocimiento adquirido de modelos previamente entrenados en tareas similares, lo que a su vez mejoró significativamente la eficiencia y precisión del modelo desarrollado.

Finalmente, se implementaron los modelos en un sistema embebido Raspberry Pi, permitiendo su funcionamiento autónomo en el contexto de un vehículo en movimiento. Además, se incorporó un dispositivo de alerta acústica, en este caso un *buzzer*, que tiene la finalidad de notificar al conductor la presencia de peatones frente al vehículo.

El prototipo final implementado se muestra en la Figura 12 y el diagrama de su funcionamiento se presenta en la Figura 13, donde se observa el ciclo y las transformaciones que se le realizan a la imagen para la detección, clasificación y activación de la salida acústica. Para una revisión detallada de los códigos fuente, programas y archivos utilizados en este capítulo, se en-

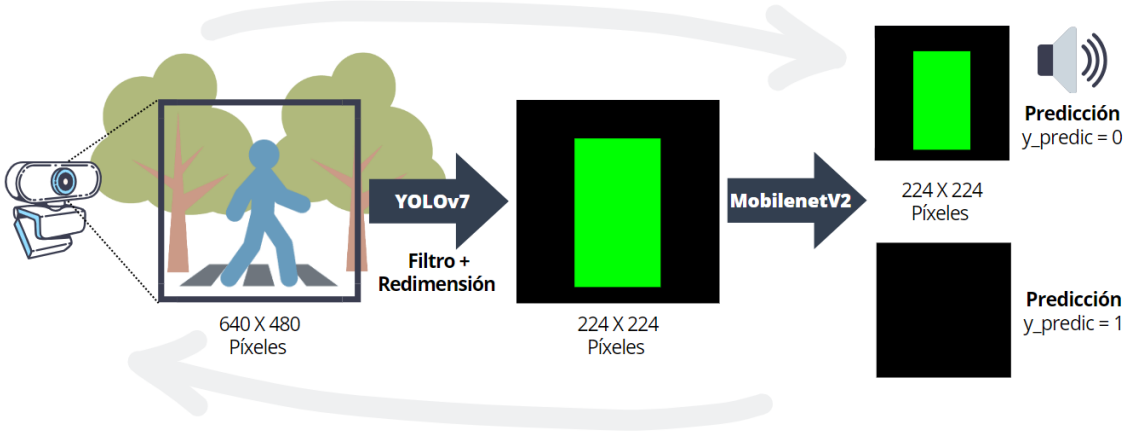
cuentran disponibles en los anexos adjuntos.

Figura 12. Prototipo final.



Nota: Elaboración propia.

Figura 13. Esquema del funcionamiento del modelo.



Nota: Elaboración propia.

Figura 14. Código para la creación de la base de datos.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random
cimg = 0

for imagenpixel in range(500):
  img1 = np.ones((480,640,3),np.uint8)
  v1=[]
  verde = random.randint(245,255)
  cantbox = random.randint(1,6)
  for arreglo in range(cantbox):
    cantbox2 = random.randint(1,6)
    v1.append(cantbox2)
  for col in range(214,439):
    for arreglo2 in range(cantbox):
      if 1 in v1:
        img1[col-15,range(257,342)] = (0,verde,0) #Izq peq
      if 2 in v1:
        img1[col-15,range(314,371)] = (0,verde,0) #Der peq
      if 3 in v1:
        img1[col,range(228,342)] = (0,verde,0) #Izq grande separados borde
      if 4 in v1:
        img1[col,range(285,400)] = (0,verde,0) #Der grande separados borde
      if 5 in v1:
        img1[col+19,range(200,314)] = (0,verde,0) #Izq grande
      if 6 in v1:
        img1[col+19,range(314,428)] = (0,verde,0) #Der grande
  cimg+=1
  direc = '/content/pppcrearimgdentro/'+str(cimg)+'.jpg'
  cv2.imwrite(direc,img1)

#plt.imshow(img1)
```

4.1. Adquisición de la base de datos

Se realizó una búsqueda de imágenes en diferentes repositorios, como Kaggle y Mendeley. Sin embargo, las imágenes obtenidas para el entrenamiento y validación para la red neuronal no eran adecuadas, ya que la distribución de personas en la imagen y el contexto de las imágenes no estaba basado en peatones. Por lo tanto, se escribió un código en lenguaje Python para la creación de una base de datos (ver Fig. 14), donde se generaron imágenes de color negro de 640x480 píxeles con cantidades y posiciones aleatorias de cajas verdes en los lugares que generalmente aparecen peatones en las imágenes tomadas por la cámara; la distribución de imágenes de la base de datos es presentada en la tabla 6.

Tabla 6
Distribución de la base de datos para el entrenamiento, validación y prueba de la red neuronal.

Subconjuntos	Muestras por clase	Total de muestras
Entrenamiento	800	1600
Validación	200	400
Prueba	150	300
Total	**1150**	**2300**

Figura 15. Código original para la comprobación de la red neuronal YOLOv7-tiny.

```
# Descargar codigo de Yolov7
!git clone https://github.com/WongKinYiu/yolov7.git
%cd yolov7
# Descargar pesos entrenados de Yolov7
!wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7-tiny.pt
!python detect.py --classes 0 --weights yolov7-tiny.pt  --conf 0.5 --img-size 640 --source IMAGENES
```

4.2. Adaptación de la red neuronal para la detección de peatones

Para realizar la adaptación de la red neuronal de la detección de personas, se obtuvo el código de YOLOv7 del repositorio de Github (Wong, 2022a). Mediante el uso de Google Colab fue cargado el modelo con la arquitectura de YOLOv7-tiny (ver Fig. 15), para luego utilizar esta red neuronal con el fin de detectar exclusivamente peatones (ver Fig. 16) (Wong, 2022a).

La modificación del modelo para nuestro proyecto se detalla en la Figura 17. Consistió en añadir líneas de código con el propósito de detectar peatones en frente del vehículo (ver Fig. 18). Una vez realizada la detección de personas se sobre escribe la imagen de color negro (en canales BGR (0,0,0)) y en el proceso que YOLOv7 reescribe la imagen, se añaden y rellenan los rectángulos que encierran cada una de las personas de color verde (en canales BGR (0,255,0)) (ver Fig. 18). Además, se agregó un filtro a la imagen para evitar que la detección de personas en los

Figura 16. Ejemplo de la detección de personas con el código original de YOLOv7.

Nota: Imagen tomada de (Karthika and Chandran, 2020).

laterales de la imagen confundan el proceso de clasificación, cubriéndolos con color negro con la misma tonalidad de la imagen sobrescrita (ver Fig. 18).

4.3. Entrenamiento y validación de la red neuronal para la clasificación

Para llevar a cabo el entrenamiento de la red neuronal, se realizó una cuidadosa identificación del tipo de arquitectura que se ajustara de manera óptima al objetivo principal del proyecto, teniendo en cuenta la naturaleza de la entrada requerida por la red neuronal.

Se procedió a implementar las bibliotecas y funciones necesarias para iniciar el entrenamiento de la red neuronal de clasificación, estas herramientas fundamentales se detallan en la Figura 19.

Una vez implementadas las librerías, es importante organizar la base de datos mediante carpetas identificadas con el nombre de las clases correspondientes.

Para entrenar esta red de clasificación, se utilizó una base de datos obtenida específicamente

Figura 17. Comparación del código modificado con el de código YOLOv7.

```
# Write results
im0 = np.zeros((im0.shape[0],im0.shape[1],3),np.uint8) ####Imagen negra####
imgfiltro = np.zeros((im0.shape[0],im0.shape[1],3),np.uint8) ####Imagen negra filtro####
for ifiltro in range(120,480):
    imgfiltro[ifiltro,range(213,426)] = (1,1,1)
for *xyxy, conf, cls in reversed(det):
    if save_txt:  # Write to file
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
        line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh)  # label format
        with open(txt_path + '.txt', 'a') as f:
            f.write(('%g ' * len(line)).rstrip() % line + '\n')

    if save_img or view_img:  # Add bbox to image
        label = f'{names[int(cls)]} {conf:.2f}'
        plot_one_box(xyxy, im0, color=colors[int(cls)], line_thickness=-1)
im0 *= imgfiltro
```

```
# Write results
for *xyxy, conf, cls in reversed(det):
    if save_txt:  # Write to file
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
        line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh)  # label format
        with open(txt_path + '.txt', 'a') as f:
            f.write(('%g ' * len(line)).rstrip() % line + '\n')

    if save_img or view_img:  # Add bbox to image
        label = f'{names[int(cls)]} {conf:.2f}'
        plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=1)
```

Nota: Códigos de Fuente:
`https://github.com/Carlos99archila/yolov7/blob/main/detect.py` y
`https://github.com/WongKinYiu/yolov7/blob/main/detect.py` respectivamente.

Figura 18. Comparación de una imagen de prueba vs la detección de personas con YOLOv7 original, YOLOv7 modificado y YOLOv7 modificado con filtro.

Nota: Imagen tomada de (Karthika and Chandran, 2020).

Figura 19. Librerías usadas para la red de clasificación.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random
import shutil
import sys
import torch
import torchvision
import os
import matplotlib.image as mpimg
import tensorflow as tf
import tensorflow_hub as hub
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, f1_score, roc_curve, precision_score, recall_score, accuracy_score, roc_auc_score
from sklearn import metrics
from mlxtend.plotting import plot_confusion_matrix
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
from google.colab import drive
```

con el código de Python (ver Fig. 14), Esta base de datos se guardó en Google Drive para entrenar la red fácilmente sin tener que cargar siempre los datos gracias a la comunicación entre Drive y Colab.

La separación de la base datos se realizó mediante líneas de código, que se destinaron el 80 % de los datos para entrenamiento y el 20 % para validación (ver Fig. 20), en este caso las imágenes de la base de datos tenían un tamaño de 640x480 píxeles. Se realizo un preprocesamiento cambiando el tamaño de las imagenes de la base de datos a 224x224 píxeles ya que la arquitectura

Figura 20. Distribución de la base de datos para validación y entrenamiento.

```
#Crear el dataset generador
datagen = ImageDataGenerator(
    rescale = 1. / 255, #Normalizar datos para tener valores entre 0 y 1
    #rotation_range = 5, #Rango que permite rotar las imagenes
    #width_shift_range = 0.10, #Rango de desplazamiento del 25 % de ancho
    #height_shift_range = 0.10, #Rango de desplazamiento del 25 % de alto
    #shear_range = 10, #Angulo de corte en sentido antihorario
    #zoom_range = [0.5, 1.5], #Rango de zoom
    validation_split = 0.2 #20% para pruebas del set
)

#Generadores para sets de entrenamiento y pruebas
#tamaño de las imagenes 224x224
data_gen_entrenamiento = datagen.flow_from_directory('/content/dataset', target_size=(224,224),
                                                     batch_size=16, shuffle=True, subset='training')
data_gen_pruebas = datagen.flow_from_directory('/content/dataset', target_size=(224,224),
                                               batch_size=16, shuffle=True, subset='validation')
```

Figura 21. Implementación de la red neuronal mobilenetV2 en Google Colab.

```
[ ]  #Importar tensorflow y tensorflow_hub donde esta el modelo
     import tensorflow as tf
     import tensorflow_hub as hub

     url = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
     mobilenetv2 = hub.KerasLayer(url, input_shape=(224,224,3))

     #imagenes de entrada de 224x224 y 3 canales de color

[ ]  #Congelar el modelo descargado para no perder el entrenamiento que le han hecho al modelo
     mobilenetv2.trainable = False

[ ]  #Entrenar al modelo con 2 salidas en la capa densa y activacion sigmoid
     modelo = tf.keras.Sequential([
         mobilenetv2,
         tf.keras.layers.Dense(2, activation='sigmoid')
     ])
```

de MobileNetV2 trabaja con este tamaño de entrada como se presenta en la Figura 20.

4.3.1. Transfer learning con MobilenetV2. Para la adaptación del modelo MobileNetV2 se utilizó la técnica llamada *transfer learning*. Primero se importó el tensor del modelo sin su capa de salida, luego se congelaron los parámetros con los que viene entrenado el modelo para no modificarlos y se agregó una capa densa de salida especificando el número de clases y la función de activación, como se presenta a continuación en la Figura 21.

Finalmente se compila el modelo y se hace uso de los *callbacks*; un *callback* es un objeto

Figura 22. Código implementando del modelo con los Callbacks.

```
#Compilar modelo
modelo.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

my_callbacks = [
    tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    patience=10,
    verbose=1,
    restore_best_weights=True),
    tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-{val_loss:.2f}.h5'),
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor="val_loss",
        factor=0.1,
        patience=4,
        verbose=1,
        min_lr=1e-12,
    )
]
```

Figura 23. Código para realizar el entrenamiento.

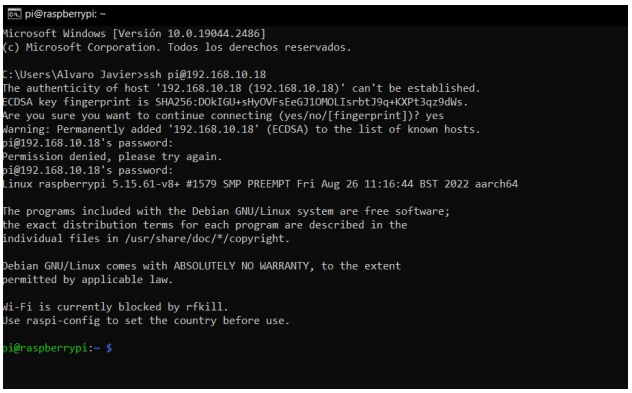
```
#Entrenar el modelo
EPOCAS = 100

historial = modelo.fit(
    data_gen_entrenamiento, epochs=EPOCAS, batch_size=16,
    validation_data=data_gen_pruebas,callbacks=my_callbacks
)
modelo.save('Final_Model_Filter_.h5')
```

que puede realizar acciones en varias etapas de entrenamiento, ya sea al comienzo o final de una época o un lote, en el entrenamiento se utilizaron tres (ver Fig. 22). *ReduceLROnPlateau* que permitió reducir la tasa de aprendizaje cuando las perdidas en la validación dejaron de mejorar, *ModelCheckpoint* que nos ayudó guardar el mejor modelo en cada época y *EarlyStopping* con el que pudimos dejar de entrenar el modelo cuando dejó de aprender.

Después de terminar el modelo de MobileNetV2 se especifican las variables que llevara el proceso de entrenamiento, como el número de épocas, *batch size* y los *datasets* de entrenamiento y validación, este proceso se presenta en la Figura 23.

Figura 24. Configuración de la raspberry pi por el servicio ssh mediante Command Prompt.



4.4. Ejecución de los dos modelos en la Raspberry Pi

Para hacer uso de la tarjeta Raspberry Pi, se comenzó instalando el sistema operativo Raspbian de 64-bits en una memoria SD-Card desde una computadora, este software esta optimizado para el hardware de la tarjeta; la implementación de los modelos de YOLOv7 y MobileNetV2 en el sistema embebido se realizó de forma remota utilizando el programa llamado VNC Viewer mediante conexiones Wi-Fi y Ethernet.

Para el uso de la Raspberry Pi se realizó una conexión vía Wi-Fi por el servicio SSH, este servicio se encuentra incluido en las últimas versiones de Raspbian y se configuró desde un computador portátil por el intérprete de comandos de *windows Command Prompt* como se presenta en la Figura 24.

Después de completar la configuración y verificación del correcto funcionamiento de la Raspberry Pi de forma remota, se instalaron las librerías y funciones necesarias para poder ejecutar

Figura 25. Código implementado en la Raspberry pi.

```
import time
import os
import cv2
camera = cv2.VideoCapture(0)
w=camera.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
h=camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
return_value, image = camera.read()
cv2.imwrite(f'yolov7/TESIS/exp/imagetesis.png', image)
del(camera)
os.system('cd /home/pi1/yolov7')
os.system(f'python yolov7/detect.py --classes 0 --weights yolov7-tiny.pt --conf 0.5 --img-size 640 --exist-ok --source yolov7/TESIS/exp/imagetesis.png')
```

los modelos de YOLOv7 y MobileNetV2. La lista de bibliotecas y funciones implementadas se puede encontrar en la siguiente tabla (7).

Tabla 7
Librerías y funciones implementadas en la Raspberry pi.

Librerías y funciones
OpenCV
Torch
Torchvision
Tqdm
Pyymal
Matplotlib
Seaborn
Spicy
Libpng-dev
Libjpeg-dev
Tensorflow
Tensorflow-hub

Mediante el uso de una conexión a internet estable fueron descargados los modelos desde Google Drive en la tarjeta, además, se escribió el siguiente código que se muestra en la Figura 25 en la Raspberry Pi desde el nuevo IDLE de Python, llamado “Thonny” el cual se encuentra incluido en Raspbian (Aivar, 2015).

Antes de usar la arquitectura Yolov7-tiny, se realizaron pruebas de rendimiento en la Raspberry Pi para diferentes modelos disponibles en el Github de Yolov (Wong, 2022b), Los tiempos de inferencia aproximados obtenidos en la Raspberry Pi se presentan en la siguiente tabla 8.

Tabla 8
Benchmark de los diferentes modelos de Yolov7 en la Raspberry Pi.

Modelos	**Tiempos de inferencia**
Yolov7_tiny.pt	747.5 ms
Yolov7.pt	4.74 s
Yolov7x.pt	7.68 s
Yolov7-w6.pt	4.26 s
Yolov7-e6.pt	5.86 s
Yolov7-d6.pt	7.70 s
Yolov7-e6e.pt	9.78 s

Nota: Elaboración propia.

Para la selección de la arquitectura de pesos que se usó en el sistema embebido, se tuvo como principal métrica la velocidad de ejecución, donde YOLOv7-tiny gana en comparación con las demás arquitecturas de pesos para YOLOv7.

Una vez seleccionada la arquitectura, se implementó el código que se presenta en la Figura 25, donde se realizó la captura de la imagen gracias a las librerías de OpenCV (Bradski, 2000). Esta imagen se introdujo en la red neuronal de detección de objetos debido a que YOLOv7 puede realizar detecciones con resoluciones estándar, luego en el código detect.py (archivo que realiza la detección de objetos) se carga y guarda el modelo entrenado de MobileNetV2 (ver Fig. 26).

Ya implementados los dos modelos, se usó un ciclo while para realizar continuamente el proceso de capturar una imagen de la cámara con un tamaño de 640x480 píxeles, ingresarla a

Figura 26. Lineas de código implementando la segunda red neuronal para la clasificación.

```
import tensorflow.keras
import tensorflow_hub as hub
os.environ["TFHUB_CACHE_DIR"] = "/home/pi1/model_mobile2"

modelo2 = tensorflow.keras.models.load_model(
    ('/home/pi1/Final_Model_filter.h5'),
    custom_objects={'KerasLayer':hub.KerasLayer}
)
```

Figura 27. Líneas de código para realizar la predicción la red neuronal de clasificación.

```
#MODELO DE MOBILENET PARA LA DETECCION
im0 = cv2.resize(im0, (224,224))
prediccion = modelo2.predict(im0.reshape(-1, 224, 224, 3))
salida = np.argmax(prediccion[0], axis=-1)
```

YOLOv7 para detección de personas y su procesamiento, luego reducir su tamaño a 224x224 píxeles para finalmente ser clasificada por MobileNetV2 (ver Fig. 27).

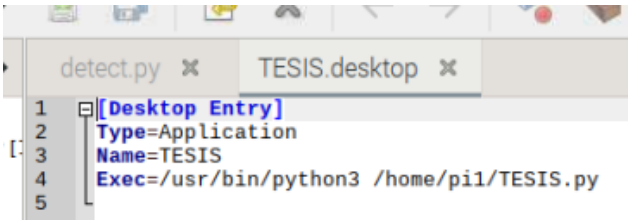
Después de implementar los dos modelos, se conecta un *buzzer* el ejecuta un sonido cuando la salida de predicción de la red de clasificación es nula indicando que hay una o más personas frente a la cámara. Cuando la predicción no es nula, el *buzzer* deja de emitir sonido, el código implementado para el *buzzer* con sus respectivas librerías y funciones se presentan en la Figura 28.

Finalmente se realiza el código de ejecución automática para los dos modelos al sistema embebido para que sea completamente autónomo, esta ejecución como se presenta en la Figura 29.

Figura 28. Código del *buzzer* implementado en la Raspberry Pi.

```
from gpiozero import Buzzer
from time import sleep
buzzer = Buzzer(17) #BUZZER EN EL PIN GPI017
    if   salida == 0:#SI
        buzzer.on()
    elif salida == 1:#NO
        buzzer.off()
```

Figura 29. Código para la ejecución automática.



5. Resultados

En este capítulo se exponen los resultados obtenidos a través de los procesos de entrenamiento, validación y prueba del modelo de la red neuronal de clasificación. Estos resultados se respaldan con curvas que ilustran la precisión y las pérdidas durante el entrenamiento y la validación, así como con la matriz de confusión de la red neuronal de clasificación. Esta matriz se utiliza para evaluar el rendimiento del modelo, donde se espera que la diagonal principal contenga la mayor cantidad de aciertos y las demás posiciones muestren los errores o confusiones del sistema.

Adicionalmente, se incluyen pruebas de distancia y velocidad realizadas con un vehículo. Estos ensayos proporcionan información relevante y se acompañan de imágenes que demuestran la capacidad del sistema para detectar peatones durante dichas pruebas.

5.1. Modelo de la red neuronal de clasificación

En la Figura 6 se observan las precisiones promedio de las detecciones de cada variante de los modelos de YOLOv7, donde YOLOv7-e6e es la topología de red más precisa en la detección de objetos con un 56.80 %, sin embargo, también es la más pesada para implementar. En el desarrollo del prototipo se priorizó la velocidad de ejecución utilizando YOLOv7-tiny, donde los tiempos de inferencia bajaron significativamente teniendo en promedio 747.5 milisegundos (Ver Tabla 8), pero también teniendo una reducción de la precisión del prototipo a 38.7 %, por esta razón se apoyó el proceso de detección con la red de clasificación MobileNetV2 para mejorar el funcionamiento del prototipo en tiempo real.

El comportamiento numérico de las métricas de rendimiento de la red neuronal de clasifica-

ción durante el entrenamiento y validación en la tabla 9. Además, en la Figura 30 se muestran las curvas de precisión del aprendizaje y pérdidas del modelo durante el entrenamiento y validación; donde se pudo observar el óptimo funcionamiento del modelo, desde la primera época las pruebas en el entrenamiento no tienen error y las pérdidas son menores al 0.5 %, esto debido a que el proceso de clasificación es muy sencillo, donde las imágenes a ser clasificadas se diferencian fácilmente y no necesitan un análisis matemático complejo.

Figura 30. Curvas de precisión del aprendizaje y pérdidas del modelo de clasificación durante el entrenamiento y validación.

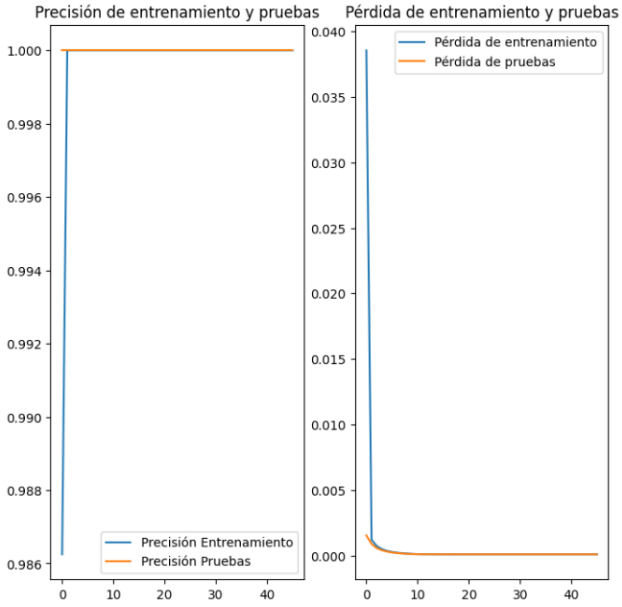
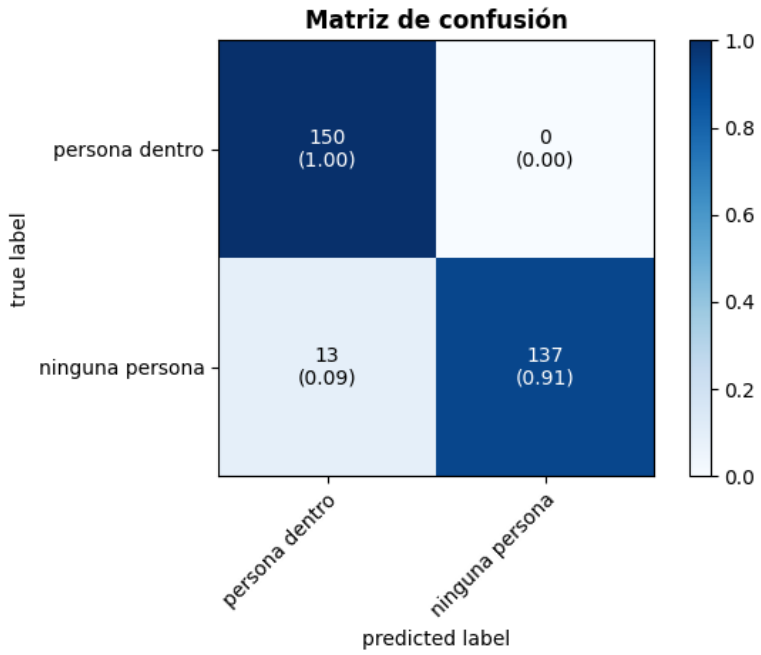


Tabla 9
Resultados del entrenamiento y validación para la red neuronal de clasificación.

Parámetro	**Entrenamiento**	**Validación**
Perdida	8.49210^{-5}	8.46410^{-5}
Precisión	1.0	1.0

Figura 31. Matriz de confusión.



En la Figura 31 se muestra la matriz de confusión, para realizarla se seleccionaron 300 imágenes, 150 por cada categoría de clasificación, para la clase donde tenemos peatones en las imágenes se tuvo una sensibilidad del 100 %, lo que indica que no se produjeron errores en la clasificación. Para las imágenes que no contienen peatones se lograron 137 imágenes clasificadas correctamente, es decir, un 91 % de precisión y 13 falsos negativos.

Los resultados observados en la matriz de confusión (ver Fig. 31) y la tabla 9 demuestran que el modelo de clasificación MobilenetV2 usando la técnica de *transfer learning* presenta un desempeño óptimo.

Tabla 10
Tiempo de inferencia final para cada modelo en la Raspberry pi 4 modelo B después de 10 ciclos de compilación.

YOLOV7-tiny	**MobilenetV2**
745 ms	345 ms
Total	1.09 s

5.2. Resultados en tiempo real para el sistema embebido

Luego de verificar el correcto funcionamiento del modelo de clasificación, se realizaron las pruebas dentro del sistema embebido, el tiempo de inferencia de cada modelo y la duración final se muestran en la tabla 10.

Una vez obtenido los tiempos de inferencia se llevaron a cabo pruebas de distancia para revisar el desempeño del modelo en la Raspberry Pi en tiempo real, usando la cámara suministrada con especificaciones que se muestran en la tabla 3. Las pruebas se realizaron en distancias aproximadas desde de 10 metros hasta 50 metros con toma de imágenes cada 10 metros; las distancias fueron establecidas con el uso de un decámetro, los resultados se pueden visualizar entre las Figuras 32 y 38; estas pruebas se realizaron durante condiciones diurnas y nocturnas.

Figura 32. Distancia de 10 metros aproximadamente.

Figura 33. Distancia de 20 metros aproximadamente.

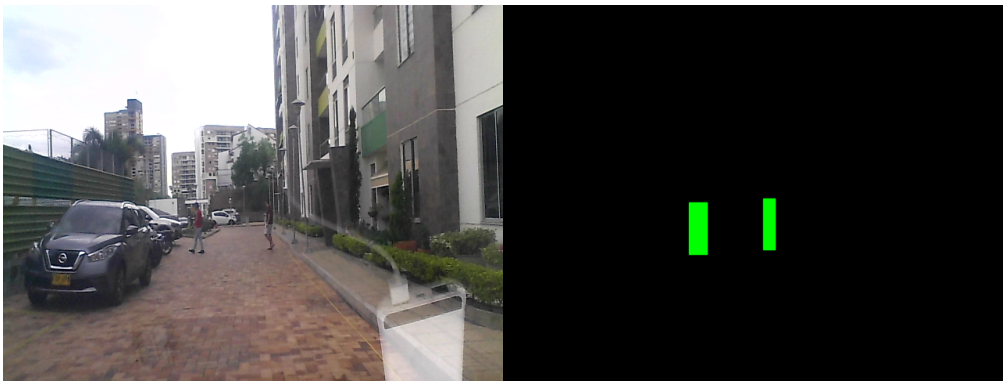


Figura 34. Distancia de 30 metros aproximadamente.

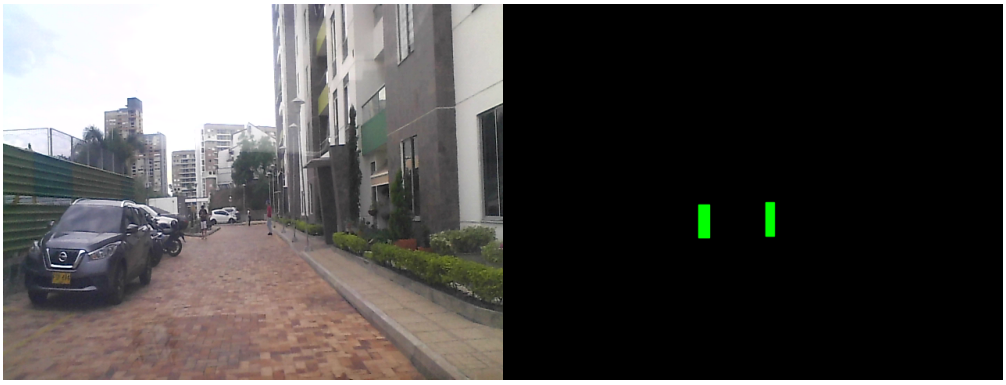


Figura 35. Distancia de 40 metros aproximadamente.

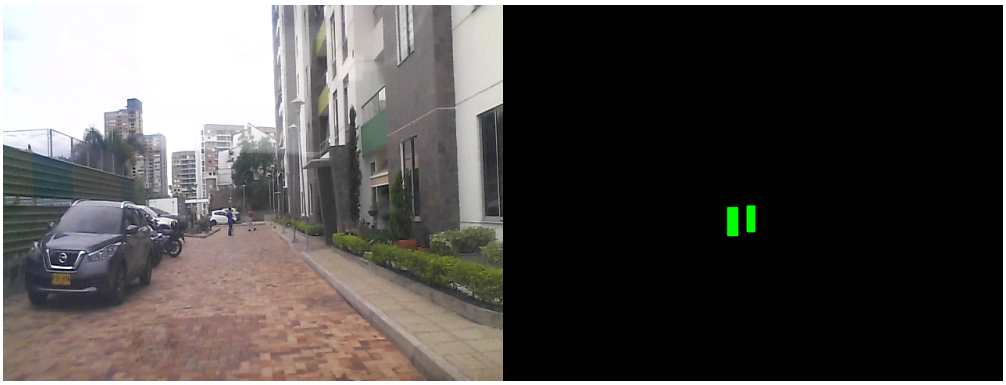


Figura 36. Distancia de 50 metros aproximadamente.

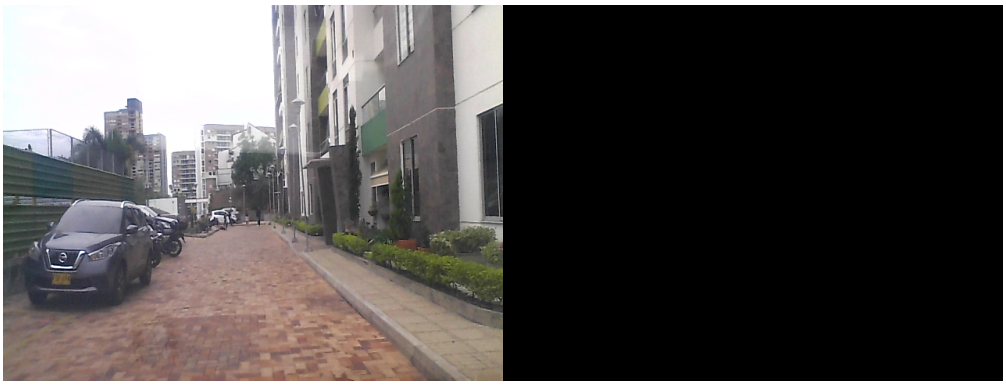
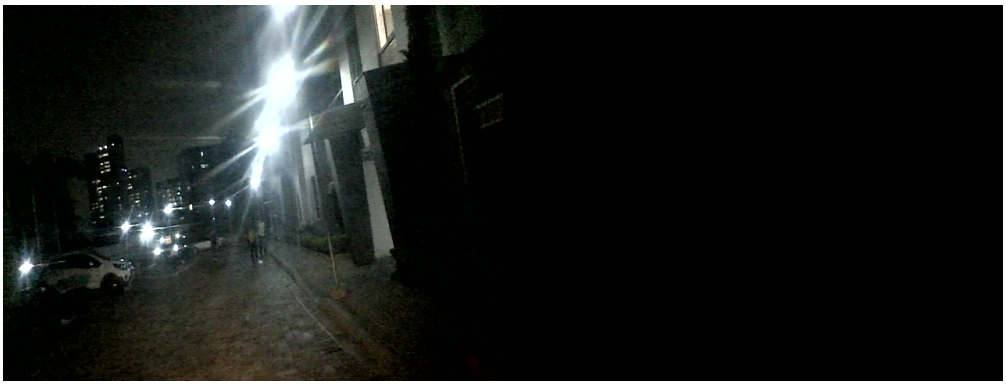


Figura 37. Distancia de 10 metros aproximadamente.

Figura 38. Distancia de 20 metros aproximadamente.



Los resultados obtenidos durante la prueba de distancia en el día muestran que la red neuronal de detección de objetos tiene limitaciones a partir de los 50 metros, al no detectar ninguna de las dos personas en la imagen (ver Fig. 36). En ambientes nocturnos la detección de peatones de YOLOv7 empieza a tener problemas en distancias cercanas a los 20 metros aproximadamente; la eficiencia de la red de detección de personas disminuye significativamente durante la noche y depende de la iluminación que pueda suministrar la ciudad y el vehículo. Finalmente, se realizaron comprobaciones tomando imágenes consecutivas desde un vehículo en movimiento a velocidades aproximadas desde $10 km/h$ hasta $50 km/h$.

Los resultados obtenidos muestran un correcto funcionamiento del prototipo para detectar peatones a determinadas velocidades y distancias. Cabe destacar que las pruebas se realizaron con la cámara fija en la posición del espejo retrovisor con vista frontal del vehículo.

Figura 39. Distancia máxima de detección de la persona con velocidad aproximada de $10 km/h$.

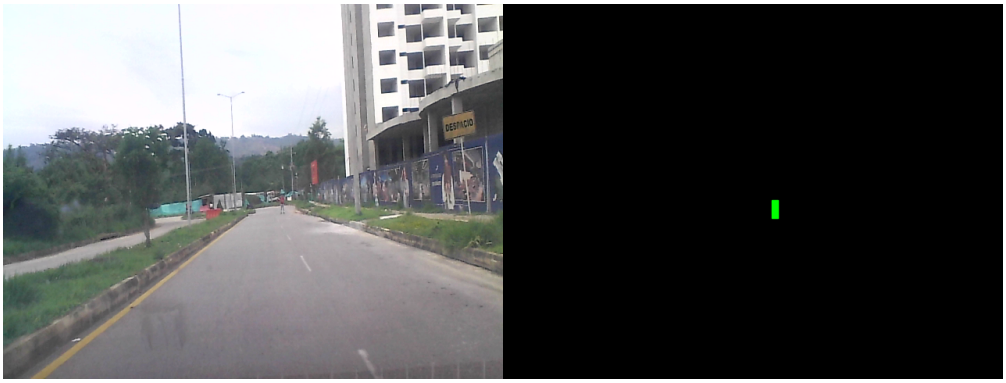


Figura 40. Distancia máxima de detección de la persona con velocidad aproximada de $20km/h$.

Figura 41. Distancia máxima de detección de la persona con velocidad aproximada de $30km/h$.

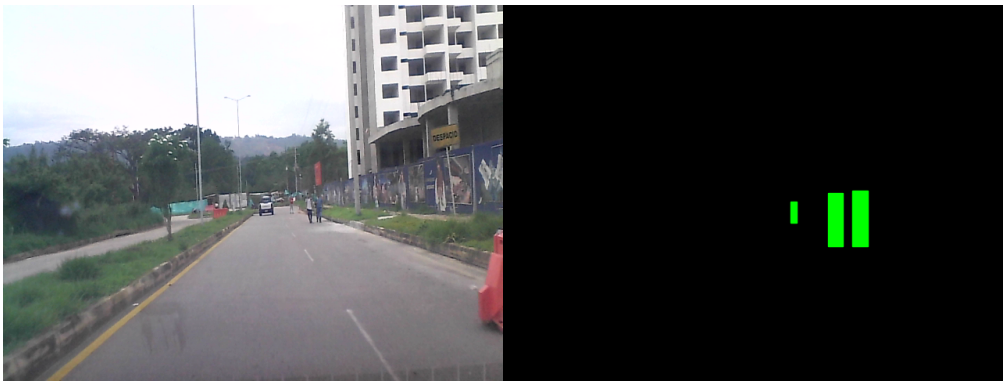
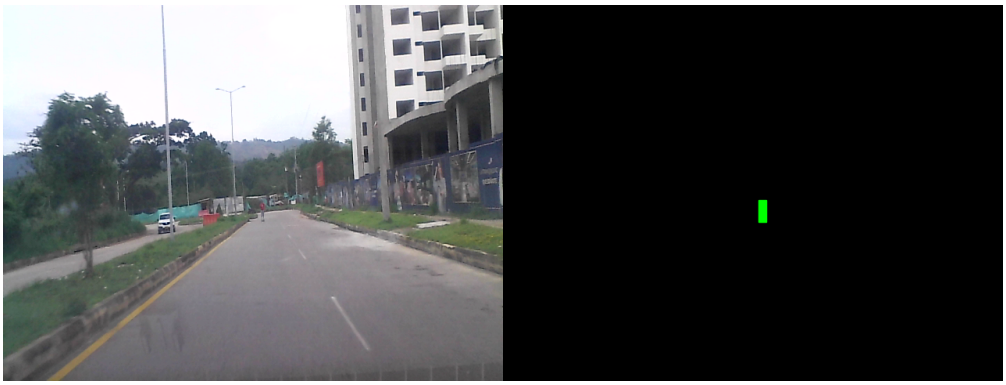


Figura 42. Distancia máxima de detección de la persona con velocidad aproximada de $40km/h$.

Figura 43. Distancia máxima de detección de la persona con velocidad aproximada de $50 km/h$.



6. Conclusiones y recomendaciones

Hemos seleccionado dos modelos de redes neuronales profundas para la detección y clasificación de personas. Como primera red seleccionamos a Yolov7 por su velocidad en la detección de objetos, ya que gracias a su arquitectura puede detectar los objetos analizando una sola vez la imagen, además, de permitir el uso de diferentes distribuciones de arquitecturas que optimizan parámetros como velocidad y exactitud. Para la segunda red neuronal escogimos mobileNetv2 por su bajo tiempo de inferencia con relación a su compatibilidad con sistemas computacionales de bajo procesamiento. Realizamos una búsqueda de bases de datos de libre acceso que contenían imágenes que cumplían con criterios específicos, como ser capturadas desde el interior de un vehículo o desde cierta altura, también pusimos especial énfasis en la distribución de los peatones dentro de las imágenes.

Para el diseño del prototipo se optó utilizar la placa Raspberry Pi 4 debido a su bajo costo y facilidad de adquisición. En términos de rendimiento, este dispositivo demostró resultados favorables al implementar las dos redes neuronales seleccionadas, con tiempos de inferencia aceptables a pesar de las limitaciones de procesamiento que presenta. Además, se complementó el prototipo con una batería portátil que permite su uso de manera independiente, así como un *buzzer* que actúa como salida para verificar su correcto funcionamiento.

Finalmente, la implementación del prototipo obtuvo resultados aceptables, logrando tiempos promedio de respuesta de 1,09 segundos, incluyendo la detección precisa de peatones sin importar su raza a distancias de hasta 40 metros en condiciones ambientales claras. Sin embar-

go, se observó una disminución de hasta el 50 % en el rango de detección en horas nocturnas, dependiendo de las condiciones de iluminación disponibles.

Digitalmente, se creó una matriz de confusión para medir la eficiencia del prototipo. En las pruebas realizadas se consiguió un índice de detección del 100 % al detectar peatones delante del vehículo. La clasificación de imágenes en las que no había peatones tuvo un 91 % de precisión y un 9 % de falsos negativos; este porcentaje de error se ve incrementado debido a la gran capacidad de detección que tiene YOLOv7 donde detecta personas en bicicletas y motocicletas.

Durante las pruebas físicas del prototipo se observó que, a velocidades cercanas y superiores a los 50 kilómetros por hora, el tiempo de respuesta del sistema no era lo suficientemente corto para tomar varias muestras sin avanzar demasiada distancia, lo que limitaba el tiempo de reacción que tendría el conductor del vehículo para tomar acciones. Por esta razón, para mejorar el funcionamiento del dispositivo se sugiere cambiar la Raspberry Pi 4 por un hardware más potente, además de incluir una etapa de detección para la carretera con el fin de mejorar la detección y evitar predicciones erróneas en curvas cerradas, cuando no hay peatones en la vía, pero si al frente del vehículo en un rango amplio.

Referencias Bibliográficas

Aivar, A. (2015). Introducing thonny, a python ide for learning programming.

Alemán, D. (2017). Técnicas de inteligencia artificial aplicadas a problemas de ingeniería civil. https://www.redalyc.org/articulo.oa?id=193955164005. *Revista de Arquitectura e Ingeniería*.

Ashwin, P. (2022). *Raspberry Pi Image Processing Programming*. Second edition.

Aurelien, G. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2nd edition.

Basery (s.f). Basery consumer electronics. https://baserytech.com.

Basogain, X. (2008). *Redes neuronales artificiales y sus Aplicaciones*.

Benitez, R., Escudero, G., Kanaan, S., and Masip, D. (2014). *Inteligencia artificial avanzada*.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Brownlee, J. (2019). A gentle introduction to transfer learning for deep learning https://machinelearningmastery.com/transfer-learning-for-deep-learning/.

Caicedo, F. and López, A. (2017). *Una aproximación practica a las redes neuronales artificiales*.

Carneiro, T., Medeiros, R., Nepomuceno, T., Bian, G., Albuquerque, V., and Filho, P. (2018). Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access*, 6:61677–61685.

Chung, C., Chen, W., and Chang, Y. (2020). Using quantization-aware training technique with post-training fine-tuning quantization to implement a mobilenet hardware accelerator. In *2020 Indo – Taiwan 2nd International Conference on Computing, Analytics and Networks (Indo-Taiwan ICAN)*.

Gomez, F., Fernandez, M., López, M., and Alonso, M. (1994). *Aprendizaje con redes neuronales artificiales*. MIT Press. https://dialnet.unirioja.es/servlet/articulo?codigo=2281678.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Iqbal, S. (2021). Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2.

Ju, L., Zou, X., Zhang, X., Xiong, X., Liu, X., and Zhou, L. (2023). An infusion containers detection method based on yolov4 with enhanced image feature fusion. *Entropy*, 25:275.

Kalley (s.f). K-pb10n. https://www.serviciokalley.com/documentos-k-pb10n.

Karthika, N. and Chandran, S. (2020). Addressing false positives in pedestrian detection. https://www.kaggle.com/datasets/karthika95/pedestrian-detection.

Kumar, S. (2023). Top 10 pre-trained models for image embedding every data scientist.

Mazur, B. (2023). 10 major technological advances in the last decade https://www.ignitec.com/insights/10-major-technological-advances-in-the-last-decade/.

Montoro, A. (2012). *Python 3 al descubierto*.

Raji, G., Vinish, A., Gopakumar, G., and Shahil, K. (2019). Implementation of bitcoin mining using raspberry pi. In *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 1087–1092.

Ramón, J. and José, V. (1995). *Redes neuronales artificiales : fundamentos, modelos y aplicaciones / José Ramón Hilera González, Victor José Martínez Hernando*.

Raspberry (s.f). Raspberry pi 4. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

Russell, S. and Noving, P. (2004). *INTELIGENCIA ARTIFICIAL. UN ENFOQUE MODERNO Segunda edición*. PEARSON EDUCACIÓN, S.A., 2 edition.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L. (2019). Mobilenetv2: Inverted residuals and linear bottlenecks.

ScienceDirect (s.f). Deep neural network https://www.sciencedirect.com/topics/computer-science/deep-neural-network.

SGMA (2017). Inteligencia artificial. *Capitulo 4*.

Shah, K. (2019). A quick overview to the transfer learning and it's significance in real world applications.

Suárez, A., Jiménez, A., Castro, M., and Cruz, A. (2017). Clasificación y mapeo automático de coberturas del suelo en imágenes satelitales utilizando redes neuronales convolucionales. *Orinoquia*.

Wang, C., Bochkovskiy, A., and Mark, H. (2022a). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.

Wang, C., Mark, H., and Yeh, I. (2022b). Designing network design strategies through gradient path analysis.

Wong, K. (2022a). *https://github.com/WongKinYiu/yolov7*.

Wong, K. (2022b). *https://github.com/WongKinYiu/yolov7/releases/tag/v0.1*.

Xiong, Y. and Zou, W. (2022). Does artificial intelligence promote industrial upgrading? evidence from china.

Yamashita, R., Nishio, M., Kinh, R., and Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9.

Yang, S., Zhu, X., Zhang, L., Wang, L., and Wang, X. (2020). Classification and prediction of tibetan medical syndrome based on the improved bp neural network. *IEEE Access*, 8:31114–31125.

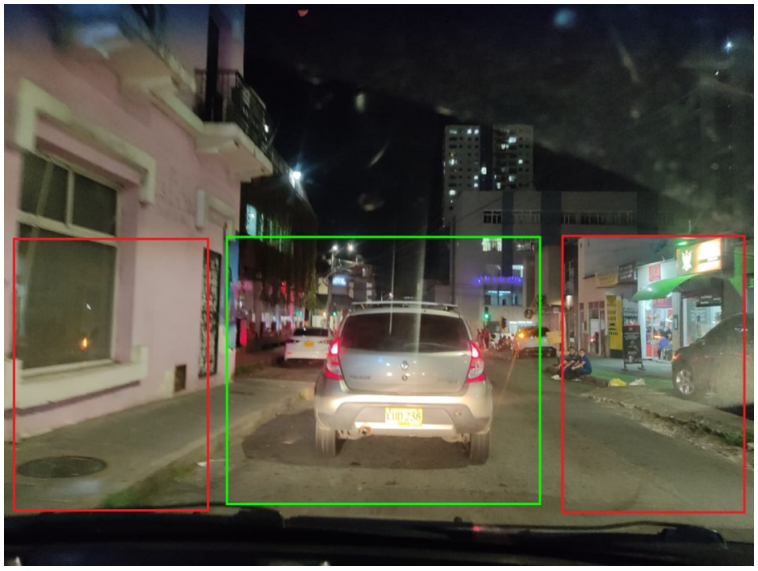
Zhao, Q., Zheng, P., Shou, X., and Wu, X. (2019). Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–21.

Anexos

Apéndice A. Modelos

Se realizaron varios procesos de entrenamiento para obtener los mejores resultados con respecto al clasificador de imágenes MobileNetV2, en los dos primeros modelos se usó una capa densa de salida con cuatro clases y una función de activación softmax, en cuanto a la base de datos, fue utilizada una distribución de personas en la imagen por cada clase, siendo las clases usadas “dentro”, “fuera”, “ambas” y “ninguna”, haciendo referencia a si se encontraban personas al frente del vehículo, la distribución de las clases en una imagen se encuentra en la Figura 44, donde dentro es cuando únicamente hay personas en la zona verde, fuera en el momento en el que hay personas solamente en las zonas rojas, ambas en las que hay personas dentro del cuadro verde y algún cuadro rojo y ninguna en el instante en el que no se detectan personas.

Figura 44. Distribución de las zonas de detección por cada clase.

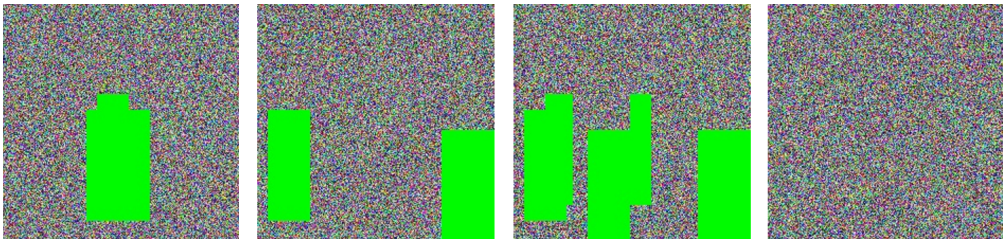


El primer modelo realizado utilizó una modificación de YOLOv7 donde la imagen que se guardaba después de detectar a las personas era la misma que se le ingresaba, pero rellenando de color verde (en canales BGR (0,255,0)) la caja que delimitaba los peatones, como se presenta en la Figura 45.

Figura 45. Ejemplo de las imágenes obtenidas con YOLOv7 para el entrenamiento del primer modelo.

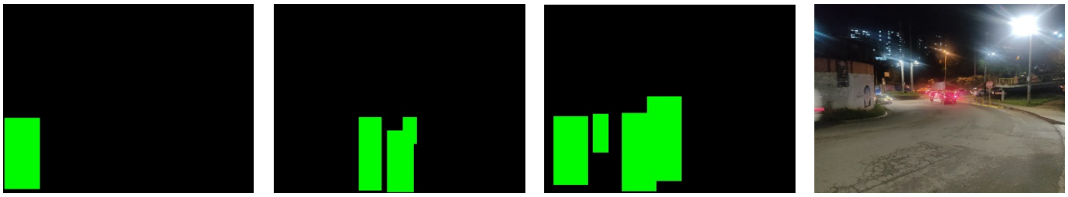
Con el conjunto de datos de esa forma se finalizó el primer entrenamiento donde se observó un resultado no tan favorable en la clasificación de las imágenes, donde era evidente la confusión del modelo, sin embargo, se amplió el dataset con imágenes BGR aleatorias generadas píxel por píxel en Python, cuidando que los píxeles verdes no tuvieran nunca el mismo color usado por YOLOv7 para generar y colorear las cajas, algunos ejemplos de las imágenes creadas se muestran en la Figura 46.

Figura 46. Imágenes de cada clase generadas aleatoriamente píxel por pixel para ampliar la base de datos del primer modelo.



El segundo modelo tuvo como punto de partida la hipótesis de que la confusión del modelo anterior se debía a la cantidad de colores diferentes en la imagen y que la imagen de salida de YOLOv7 (resolución de la imagen tomada por la cámara 640x480 píxeles) tiene unas dimensiones diferentes a las de la entrada de MobileNetV2 (224x224 píxeles) haciendo que se pierda información al redimensionar la imagen, de esta forma se buscó eliminar el fondo de la imagen al detectar personas, generando un fondo unicolor negro y manteniendo la imagen original si no se detectaban peatones, a continuación, se muestra un ejemplo de imagen por cada clase para esta versión en la Figura 47.

Figura 47. Imágenes de cada clase para la base de datos del segundo modelo.



El segundo modelo arrojo resultados alentadores donde las imágenes con peatones detectados eran clasificadas de forma correcta para la mayoría de las pruebas de las clases dentro y ninguna; la confusión se presentaba de forma mayoritaria entre las clases ambas, fuera y dentro (en algunos casos), si bien el dispositivo arrojaba respuesta cuando había personas situadas en las zonas dentro y ambas, también emitía la señal de aviso cuando se encontraban peatones en la regiones de la clase fuera, teniendo falsos negativos que afectaban la utilidad del prototipo.

El modelo final corrigió los problemas de confusión al reducir las clases a dos, “dentro” y “ninguna”, siguiendo la lógica de la distribución de personas ya mencionada, para garantizar las dos clases de salida se aplicó un filtro sobre la imagen donde YOLOv7 detecto personas, elimi-

nando los píxeles verdes de las personas detectados a los lados y en el caso de la clase ninguna se obtiene una imagen completamente negra, facilitando el proceso de clasificación.

Para realizar este modelo se editó YOLOv7 como se mostró en este documento, en el proceso de entrenamiento se utilizó una capa densa de salida con dos clases y la función de activación sigmoide como se pudo observar en la Figura 21 del proceso de entrenamiento, a continuación, se muestran las imágenes que representan el filtro y un ejemplo de cada clase obtenida en la Figura 48.

Figura 48. Visualización del filtro usado en el modelo final, donde la zona blanca es la parte que se mantiene donde hay detección de peatones con YOLOv7 y la región negra es la parte que se elimina de la imagen por estar fuera el rango central del vehículo.

Apéndice B. Repositorio de toda la metodología y resultados del proyecto.

Todo lo realizado en metodología y pruebas se suministra en los repositorios de los siguientes links:

1. https://github.com/Carlos99archila/yolov7

2. https://drive.google.com/drive/folders/1wmX8S1e1hNlcYijiZOTl8J7KdfGgtzAr?usp=sharing

3. https://drive.google.com/drive/folders/1LBrsaCfCzKjYcw0GQ_f_4gsRsrX6iY1H?usp=sharing