Diseño de un sistema de medición de velocidad de vehículos con técnicas de procesamiento de imágenes e inteligencia artificial

Luis David Fontalvo Nuñez y Kevin Javier Sandoval Sandoval

Trabajo de Grado para Optar al Título de Ingeniero Electrónico

Director

Jaime Guillermo Barrero

Magister en Potencia Eléctrica

Universidad Industrial de Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Ingeniería Electrónica

Bucaramanga

2024

Dedicatoria

Dedicamos este proyecto a aquellos que han sido nuestra inspiración y sostén a lo largo de este camino:

A nuestros padres, cuyo amor incondicional y sacrificio han sido la fuerza que nos impulsa a alcanzar nuestras metas. Su constante apoyo y orientación han sido fundamentales en cada paso que damos.

A nuestros amigos y seres queridos, por su aliento, comprensión y ánimo en los momentos difíciles. Su presencia ha sido un alivio en los momentos de incertidumbre.

A nuestros mentores y profesores, cuya sabiduría y guía han guiado nuestro camino académico y profesional. Su dedicación y enseñanzas han dejado una huella imborrable en nuestro desarrollo.

A todas las personas que, de una forma u otra, han contribuido a nuestro crecimiento personal y profesional, les expresamos nuestra más sincera gratitud.

Que este proyecto sea un tributo a su confianza en nosotros y un testimonio de nuestro eterno agradecimiento.

Agradecimientos

Dedicamos este trabajo a aquellos que han sido parte fundamental de nuestro camino:

Agradecemos a Dios por otorgarnos la sabiduría y la perseverancia necesarias para alcanzar este logro.

A nuestras familias, por su constante apoyo, comprensión y amor incondicional a lo largo de esta travesía.

A nuestros amigos y compañeros de carrera, por su camaradería, ánimo y colaboración en cada etapa de este proyecto.

Al profesor Jaime Guillermo Barrero Pérez, por su invaluable orientación, paciencia y conocimientos compartidos durante el desarrollo de esta investigación.

Tabla de Contenidos

Pág.

Lista de tablas

Lista de figuras

Resumen

Título: Diseño de un sistema de medición de velocidad de vehículos con técnicas de procesamiento de imágenes e inteligencia artificial*

Autores: Luis David Fontalvo Nuñez, Kevin Javier Sandoval Sandoval**

Palabras Clave: Inteligencia artificial, aprendizaje automático, redes neuronales, YOLO, seguimiento de objetos

Descripción:

Este proyecto se enfoca en el diseño de un sistema económico y eficiente para medir la velocidad de vehículos utilizando procesamiento de imágenes e inteligencia artificial. Se investigaron redes neuronales y sistemas embebidos, eligiendo la AMB82 MINI por su bajo costo y compatibilidad con el entorno de Arduino, en comparación con la Raspberry Pi 4 B y la Jetson Nano. El sistema implementa detección de vehículos en tiempo real mediante el modelo YOLOv4 Tiny entrenado con un conjunto de datos personalizado. Se desarrolló un algoritmo de análisis de movimiento para estimar la velocidad de los vehículos detectados. El desarrollo del código incluyó la configuración inicial, el bucle principal y funciones de apoyo. Cuando se detecta un vehículo que excede el límite de velocidad, se captura una imagen que muestra el vehículo y su velocidad estimada. La imagen se envía a Google Drive mediante codificación base64 y URL. Utilizando Google Apps Script e IFTTT, se notifican los excesos de velocidad a través de Telegram. El sistema fue validado en pruebas de tráfico diurno en una zona de detección de quince metros, obteniendo un error promedio en las mediciones de 4.2945%.

***** Trabajo de Grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director Jaime Guillermo Barrero Perez

Abstract

Title: Design of a Vehicle Speed Measurement System Using Image Processing and Artificial Intelligence Techniques *

Authors: Luis David Fontalvo Nuñez, Kevin Javier Sandoval Sandoval **

Keywords: Artificial intelligence, machine learning, neural networks, YOLO, object tracking

Description:

This project focuses on designing an economical and efficient system to measure vehicle speed using image processing and artificial intelligence. Neural networks and embedded systems were investigated, choosing the AMB82 MINI for its low cost and compatibility with the Arduino environment, compared to the Raspberry Pi 4 B and the Jetson Nano. The system implements real-time vehicle detection using the YOLOv4 Tiny model trained with a custom dataset. A motion analysis algorithm was developed to estimate the speed of detected vehicles. The code development included initial setup, the main loop, and support functions. When a vehicle exceeding the speed limit is detected, an image is captured showing the vehicle and its estimated speed. The image is sent to Google Drive using base64 and URL encoding. Using Google Apps Script and IFTTT, speed violations are notified via Telegram.

The system was validated in daytime traffic tests in a detection zone of fifteen meters, obtaining an average error in measuremets of 4.2945%.

***** Bachelor Thesis

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director Jaime Guillermo Barrero Perez

1. Objetivos

1.1. Objetivos específicos

Diseñar un sistema de medición de velocidad vehicular basado en técnicas de procesamiento de imágenes e inteligencia artificial.

1.2. Objetivos específicos

- Seleccionar un sistema embebido de bajo costo con capacidad de adquisición de imágenes y procesamiento de video, considerando requisitos de rendimiento y procesamiento de imágenes.
- Entrenar una red neuronal en el ámbito de la inteligencia artificial para la identificación específica de vehículos, centrándose en la capacidad de reconocer automóviles.
- Desarrollar e implementar una técnica de procesamiento de imágenes para la medición de velocidad en tiempo real del vehículo identificado.

2. Medición de velocidad con inteligencia artificial

La medición de velocidad con inteligencia artificial es un área de investigación y aplicación que ha experimentado un crecimiento significativo en los últimos años. Utilizando algoritmos de visión por computadora y técnicas de aprendizaje automático, es posible estimar la velocidad de objetos en movimiento a partir de imágenes o secuencias de video. Esta capacidad tiene una amplia gama de aplicaciones en campos como la vigilancia de tráfico, la monitorización de peatones, la detección de vehículos en movimiento y la seguridad vial. A continuación, se presentan algunas consideraciones y herramientas para la medición de velocidad con inteligencia artificial:

2.1. Inteligencia artificial

La inteligencia artificial (*Artificial intelligence*) se constituye como un campo científico dedicado a concebir sistemas informáticos y máquinas capaces de realizar procesos de razonamiento y actuación, imitando habilidades que normalmente serían atribuidas a la inteligencia humana. Este campo también abarca situaciones en las que se manejan conjuntos de datos de una magnitud que sobrepasa la capacidad analítica de los seres humanos *¿Qué es la inteligencia artificial (IA)?* (s.f.).

2.2. Aprendizaje automático

Según se afirma en *¿Qué es machine learning?* (s.f.), el aprendizaje automático es un subconjunto de la inteligencia artificial, este implica el uso de redes neuronales por parte de los ordenadores para discernir patrones y potenciar progresivamente su habilidad en su identificación.

Mediante ajustes adecuados y la asimilación de datos suficientes, un algoritmo de aprendizaje automático puede anticipar y reconocer patrones e información. El aprendizaje automático está dividido en cuadro categorías:

- **Aprendizaje supervisado:** en donde se utiliza un conjunto de datos etiquetados para enseñar a los algoritmos o modelos a generar la salida deseada.
- **Aprendizaje no supervisado:** implica trabajar con un conjunto de datos no etiquetados, el algoritmo debe comprender los patrones por sí mismo.
- **Aprendizaje semi-supervisado:** utiliza algoritmos no supervisados para extraer características valiosas de datos no etiquetados con el fin de mejorar la calidad de un modelo de aprendizaje supervisado.
- **Aprendizaje de refuerzo:** permite que la inteligencia artificial desarrolle estrategias efectivas mediante la experimentación e interacción con el entorno mientras aprende de su propia experiencia, repitiendo y reforzando acciones de recompensas obtenidas durante la interacción.

2.3. Redes neuronales y aprendizaje profundo

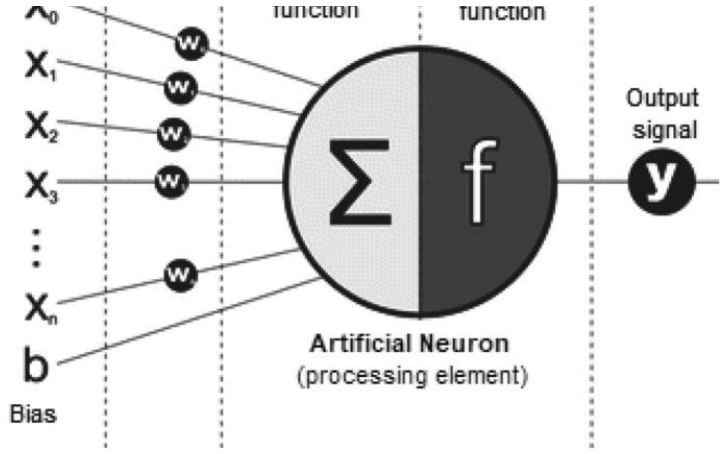
El aprendizaje profundo (*Deep learning*) se centra en el entrenamiento de algoritmos llamados redes neuronales artificiales para realizar tareas específicas. Estas redes neuronales están inspiradas en el funcionamiento y la estructura del cerebro humano, con capas y nodos interconectados que procesan y transforman la información recibida. Las redes neuronales profundas constan de múltiples capas interconectadas de nodos, cada capa se basa en la anterior para mejorar la predicción o clasificación. El flujo de cálculos a través de la red se llama

propagación directa. Las capas de entrada y salida se llaman capas visibles: la capa de entrada procesa los datos, y la capa de salida produce la predicción o clasificación.

La retropropagación (*Backpropagation*), mediante algoritmos como el descenso del gradiente, calcula errores en las predicciones y ajusta los pesos y sesgos de la red neuronal, moviéndose hacia atrás a través de las capas. Este proceso junto con la propagación directa permite que la red haga predicciones y corrija errores. Con el tiempo, el algoritmo se vuelve más preciso a medida que ajusta continuamente sus parámetros durante el entrenamiento. *What is deep learning?* (s.f.).

Figura 1

Ilustración del modelo matemático de una neurona artificial.



Nota. Tomado de: Sarker, I.H. (2021). *Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions*. SN Computer Science, 2, 420. https://doi.org/10.1007/s42979-021-00815-1 (Pag. 2)

Las redes neuronales tienen una amplia variedad de aplicaciones en diversos campos. Por ejemplo: la visión por computadora, problemas de reconocimiento, procesamiento de lenguaje natural para la traducción automática, predicción de eventos basado en datos de entrada, clasificación de objetos, sistemas de recomendación de productos en plataformas de comercio electrónico, etc. Los algoritmos de *Deep Learning* son complejos y hay diversos tipos de redes neuronales diseñadas para resolver tareas específicas, estas redes neuronales son: redes neuronales convolucionales y redes neuronales recurrentes.

2.4. Redes neuronales convolucionales (CNN)

Como se afirma en *¿Qué son las redes neuronales convolucionales?* (*s. f.*), son un subconjunto del *machine learning* y son fundamentales en el *deep learning.* Tienen capas de nodos que incluyen una capa de entrada, una o varias capas ocultas y una capa de salida. Estas redes neuronales son comúnmente usadas en clasificación y visión por computadora. Las CNN, más eficientes que métodos manuales, emplean álgebra lineal, especialmente en la multiplicación de matrices, para identificar patrones en imágenes. A pesar de su escalabilidad, las CNN pueden ser intensivas en recursos y requerir unidades de procesamiento gráfico (GPU) para entrenar modelos.

Las CNN se destacan por su buen rendimiento con datos de imagen, voz o audio. Están compuestas de tres tipos de capas: convolucional, de agrupación y totalmente conectada. La capa convolucional es la primera en la red, puede ser seguida por más capas convolucionales o de agrupación, siendo la capa totalmente conectada la última de la red.

- **Capa convolucional:** Se encarga de realizar los cálculos calve mediante la convolución. Utiliza datos de entrada, un filtro y un mapa de características. La entrada, puede ser una

imagen a color, se procesa con un filtro que se desplaza por la imagen, realizando convoluciones. Cada convolución produce un mapa de características. Tras la convolución, se aplica la función de ReLU para introducir no linealidad. Pueden agregarse capas de convolución adicionales, formando una estructura jerárquica en la CNN, permitiendo la detección de patrones más complejos.

- **Capa de agrupación:** También llamada submuestreo, reduce la dimensión de la entrada disminuyendo el número de parámetros. Al igual que la capa convolucional, utiliza un filtro para recorrer la entrada, pero sin pesos. En su lugar, aplica funciones de agregación como máxima o media. La agrupación máxima selecciona el valor más alto, mientras que la agrupación media calcula el promedio en el campo receptivo. Aunque se pierde información, la capa de agrupación beneficia a la CNN al reducir complejidad, mejorar eficiencia y limitar sobreajuste.
- **Capa totalmente conectada:** Tiene cada nodo conectado directamente a un nodo de la capa anterior. Esta capa clasifica basándose en las características extraídas de capas anteriores. Se utiliza funciones ReLU en capas convolucionales y de agrupación, mientras que en las capas totalmente conectadas se emplea la función de activación softmax para clasificar entradas y generar probabilidades entre 0 y 1.

Con cada capa, la CNN aumenta en complejidad, identificando partes más grandes de la imagen. las primeras capas se enfocan en características simples, como colores, y bordes, y a medida que avanzan las capas, la CNN reconoce elementos más grandes hasta identificar el objeto objetivo.

2.5. Funciones de activación.

En el aprendizaje sobre redes neuronales, se destaca que una función de activación decide la activación de una neurona. Las funciones no lineales comúnmente transforman la salida de una neurona a un valor entre 0 y 1 o -1 y 1. Entre estas se destacan la función sigmoide, ReLU, Softmax y tangente hiperbólica.

2.6. Framework

Es una estructura conceptual y tecnológica sobre la cual se puede construir y desarrollar software. Proporciona una serie de bibliotecas, herramientas y pautas que sirven como base para el desarrollo de aplicaciones de manera rápida y eficiente. Los *frameworks* suelen ser extensibles, lo que significa que se puede agregar o modificar su funcionalidad según tus necesidades específicas. Algunos ejemplos de *frameworks* son:

- **Tensorflow**: Es una plataforma de aprendizaje automático de extremo a extremo, de código abierto, desarrollada originalmente por Google Brain para investigar y desarrollar tecnologías de redes neuronales. Proporciona un ecosistema completo de herramientas y bibliotecas para investigadores y desarrolladores, permitiendo la creación, implementación y mantenimiento de aplicaciones basadas en aprendizaje automático de manera eficiente. Ofrece APIs estables en Python y C++, así como soporte para otros lenguajes, lo que lo hace versátil y adaptable a diferentes necesidades de desarrollo *Tensorflow* (s.f.).
- **PyTorch**: Desarrollado originalmente por Facebook AI Research y ahora gestionado por la Fundación PyTorch, es un marco de aprendizaje profundo de código abierto que utiliza

la biblioteca back-end de Torch y una API de alto nivel en Python. Permite crear una variedad de redes neuronales, desde simples hasta complejas, utilizadas en tareas como visión artificial y procesamiento de lenguaje natural. Es ampliamente utilizado en investigación y desarrollo de inteligencia artificial. *What is PyTorch?* (*s.f.*).

- **Keras**: Keras es una API de aprendizaje automático que ofrece una experiencia de desarrollo rápida, elegante y fácil de mantener. Se enfoca en la velocidad de depuración, la legibilidad del código y la facilidad de implementación, permitiendo a los desarrolladores crear modelos de aprendizaje profundo más eficientes y escalables. Keras está diseñado para ser accesible para los seres humanos, con una API simple y consistente, documentación detallada y mensajes de error claros, reduciendo la carga cognitiva y facilitando el proceso de desarrollo. *Keras, Simple. Flexible. Powerful* (s.f.).
- **Darknet**: Es un marco de red neuronal de código abierto desarrollado en C y CUDA, que ofrece velocidad y facilidad de instalación. Es compatible con el cálculo tanto en CPU como en GPU, lo que lo hace versátil para una variedad de aplicaciones de aprendizaje automático y visión por computadora. Puede acceder a su código fuente en GitHub y explorar sus capacidades adicionales para comprender mejor su potencial y aplicaciones. *Darknet: Open Source Neural Networks in C*. (2013-2016).

Darknet no es solo un framework de redes neuronales, sino también una red neuronal convolucional (CNN) en sí misma. Darknet tiene redes neuronales preentrenadas como Darknet19 y Darknet53 como se observa en la Tabla 1. Darknet19 es una versión más ligera de la red neuronal convolucional Darknet. Consiste en 19 capas de convolución, seguidas de capas de submuestreo y

algunas capas completamente conectadas. Darknet53 es una arquitectura más profunda y compleja que Darknet19. Consiste en 53 capas de convolución y se caracteriza por su mayor capacidad de representación y aprendizaje de características.

Tabla 1

Características de algunas redes neuronales preentrenadas.

Red neuronal	Profundidad	Tamaño	Parámetros (millones)	Tamaño de entrada de imagen
`squeezenet`	18	5,2 MB	1.24	227 por 227
`googlenet`	22	27 MB	7.0	224 por 224
`inceptionv3`	48	89 MB	23.9	299 por 299
`densenet201`	201	77 MB	20.0	224 por 224
`mobilenetv2`	53	13 MB	3.5	224 por 224
`resnet18`	18	44 MB	11.7	224 por 224
`resnet50`	50	96 MB	25.6	224 por 224
`resnet101`	101	167 MB	44.6	224 por 224
`xception`	71	85 MB	22.9	299 por 299
`inceptionresnetv2`	164	209 MB	55.9	299 por 299
`shufflenet`	50	5,4 MB	1.4	224 por 224
`nasnetmobile`	*	20 MB	5.3	224 por 224
`nasnetlarge`	*	332 MB	88.9	331 por 331
`darknet19`	19	78 MB	20.8	256 por 256
`darknet53`	53	155 MB	41.6	256 por 256
`efficientnetb0`	82	20 MB	5.3	224 por 224
`alexnet`	8	227 MB	61.0	227 por 227
`vgg16`	16	515 MB	138	224 por 224
`vgg19`	19	535 MB	144	224 por 224

Nota. Tomado de: *Pre-trained deep neural networks*. Mathworks (s.f.).

Recuperado de: https://es.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html

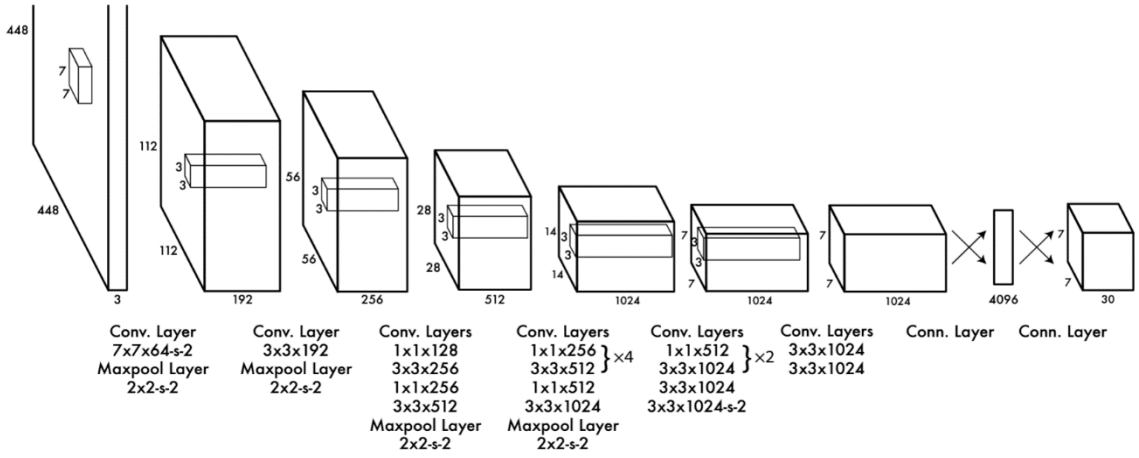
2.7. YOLO (You Only Look Once)

(Redmon et al., 2015) propone YOLO, fue desarrollado como un modelo de detección de objetos dentro del *framework* Darknet para la detección de objetos en tiempo real que utiliza una única red neuronal convolucional (CNN) para predecir las ubicaciones y las clases de los objetos en una imagen de manera eficiente. Este sistema divide la imagen en una cuadrícula y, de manera simultánea, predice los cuadros delimitadores y las probabilidades de clase para cada cuadro. Esta aproximación holística permite una detección rápida y precisa de objetos en tiempo real, eliminando la necesidad de pasos separados para la detección y la clasificación. Esto lo hace más rápido y preciso que otros métodos.

Este modelo está inspirado en la arquitectura GoogLeNet para la clasificación de imágenes.

Figura 2

Arquitectura de la red de detección YOLO.



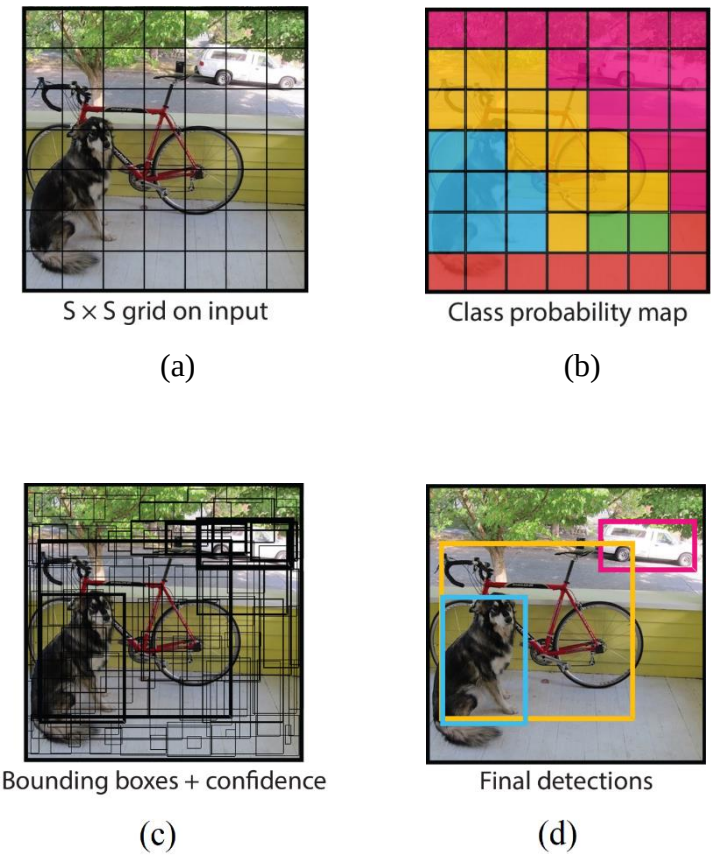
Nota. Tomado de: You Only Look Once: Unified, Real-Time Object Detection (2015). https://arxiv.org/pdf/1506.02640.pdf

Su funcionamiento se puede resumir de la siguiente manera:

- **División de la imagen**: YOLO divide la imagen en una cuadrícula de celdas. La imagen de entrada se redimensiona a un tamaño fijo (por ejemplo, 416x416 píxeles) y luego se divide en una cuadrícula de SxS celdas (por ejemplo, 13x13 celdas) Figura 3.a.
- **Predicción por celda**: Cada celda de la cuadrícula predice B cajas delimitadoras y sus correspondientes confianzas de clase. La confianza de clase indica la probabilidad de que un objeto de una clase particular esté presente en la celda como se observa en Figura 3.b.
- **Regresión**: YOLO utiliza una regresión lineal para ajustar las predicciones de las celdas. Las predicciones de las celdas incluyen las coordenadas del centro de la caja delimitadora, la anchura y la altura de la caja delimitadora, y la confianza de la clase y se observa en Figura 3.c.
- **Supresión de no máximos**: YOLO elimina las predicciones redundantes y superpuestas. Se utiliza un algoritmo de supresión de no máximos (NMS) para seleccionar las mejores predicciones para cada clase como se observa en Figura 3.d.

Figura 3

Funcionamiento de la predicción por celda en YOLO.



Nota. Tomado de: *You Only Look Once: Unified, Real-Time Object Detection* (2015). https://arxiv.org/pdf/1506.02640.pdf

YOLO es comparable en precisión a los métodos de detección de objetos de dos pasos más lentos. La precisión de YOLO se puede mejorar utilizando una red neuronal más profunda o un conjunto de datos más grande para el entrenamiento.

2.8. Integración de IA en sistemas embebidos

La integración de IA en sistemas embebidos ha sido posible gracias al desarrollo de hardware especializado, como las Unidades de Procesamiento Neuronal (NPU), que están diseñadas para ejecutar eficientemente modelos de IA en dispositivos con recursos limitados. Estas NPUs permiten realizar tareas de inferencia de manera rápida y eficiente, lo que hace posible la implementación de aplicaciones de IA en dispositivos embebidos.

Esto habilita una amplia gama de aplicaciones prácticas en campos como la seguridad, la salud y la automatización industrial.

(Bamoumen et al., 2022) explora cómo la tecnología TinyML, una forma de aprendizaje automático integrado en dispositivos de baja potencia, puede contribuir significativamente a abordar problemas ambientales. Se destaca el papel de TinyML, señalando que esta tecnología permite la implementación de algoritmos complejos de aprendizaje automático en dispositivos de baja potencia. Esto abre la puerta a una amplia gama de nuevas aplicaciones que pueden extender el impacto de la tecnología en la sociedad. También describe los desafíos específicos asociados con la implementación de TinyML en dispositivos de borde de IoT, incluida la gestión de recursos limitados y la duración de la batería.

(Tan & Cao, 2022) expone que, en los últimos tiempos, diversas empresas como Huawei, Qualcomm y Samsung han incursionado en el desarrollo de Unidades de Procesamiento Neuronal (NPU) específicamente diseñadas para dispositivos móviles, capaces de manejar tareas de inteligencia artificial. Estas NPU han permitido una notable reducción en el tiempo de ejecución de las Redes Neuronales Profundas (DNN). Por ejemplo, en el caso del HUAWEI Mate 10 Pro, la

ejecución de AlexNet (una DNN) en la NPU es hasta 30 veces más rápida que en la CPU. Aunque las GPUs también pueden utilizarse para ejecutar DNN en dispositivos móviles, su capacidad no se compara con la de las NPU, las cuales ofrecen un rendimiento notablemente superior y son más eficientes energéticamente. Sin embargo, las NPU enfrentan ciertas limitaciones fundamentales, siendo la más destacada la precisión de los números de punto flotante. A diferencia de la CPU que utiliza 32 bits para representar estos números, las NPU utilizan solo 16 bits o 8 bits, lo que resulta en una ejecución más rápida pero menos precisa de las DNN.

La IA en los sistemas embebidos puede aplicarse en diversos campos, como la visión por computadora para el reconocimiento de objetos, el procesamiento del lenguaje natural para la interacción con el usuario, la toma de decisiones autónomas en tiempo real, entre otros. Estas capacidades inteligentes permiten a los sistemas embebidos ofrecer funciones avanzadas, mejorar la automatización y optimizar el rendimiento.

2.9. Seguimiento de objetos

El seguimiento de objetos es una técnica fundamental en visión por computadora que consiste en detectar y seguir el movimiento de objetos en secuencias de imágenes o videos. El seguimiento de objetos se centra en el seguimiento continuo del objeto a lo largo del tiempo.

1. **Detección Inicial**: El proceso comienza con la detección inicial del objeto en la primera imagen de la secuencia. Esto puede hacerse mediante técnicas de detección de objetos como regiones de interés (ROI) o redes neuronales convolucionales (CNN), que identifican la presencia y ubicación aproximada del objeto en la imagen.
2. **Inicialización del Seguimiento**: Una vez que el objeto ha sido detectado en la primera imagen, se inicializa el proceso de seguimiento. Esto implica asignar un identificador único

al objeto y crear una región de interés alrededor del mismo para seguir su movimiento en cuadros sucesivos.

3. **Actualización de la Posición**: A medida que la secuencia de imágenes avanza, el algoritmo de seguimiento actualiza continuamente la posición y la forma de la región de interés para mantenerse alineado con el objeto en movimiento. Esto se logra utilizando métodos de seguimiento que comparan las características del objeto en cuadros consecutivos.

2.10. Codificación en Base64

Base64 es un esquema de codificación que se utiliza para representar datos binarios en una forma legible por humanos mediante la conversión de los datos en una secuencia de caracteres ASCII. Este método de codificación es útil cuando se necesita transmitir datos binarios a través de canales que pueden no ser seguros para caracteres especiales o cuando se requiere una representación legible de datos binarios.

2.10.1. *Funcionamiento de Base64*

1. **División en bloques de bits**: Los datos binarios se dividen en bloques de bits. Cada bloque de bits se convierte en un valor decimal que corresponde a un carácter ASCII en una tabla de codificación Base64.
2. **Mapeo a caracteres ASCII**: Los valores decimales obtenidos en el paso anterior se mapean a caracteres ASCII según una tabla de codificación Base64. Esta tabla consta de 64 caracteres, que incluyen letras mayúsculas y minúsculas, dígitos y algunos caracteres especiales.

3. **Relleno de caracteres**: Si la longitud de los datos binarios no es un múltiplo de 3, se agrega un relleno de uno o dos caracteres "=" al final de la cadena codificada para indicar que se han agregado bytes adicionales para completar un bloque completo de 24 bits.
4. **Aplicaciones de Base64**
    - **Transmisión de datos binarios**: Se utiliza comúnmente en protocolos de comunicación en línea, como HTTP, SMTP y MIME, para transmitir datos binarios, como imágenes, archivos adjuntos de correo electrónico y datos de sesión, a través de canales de texto sin dañar la integridad de los datos.
    - **Almacenamiento de datos binarios en texto plano**: Es útil para almacenar datos binarios en formatos de texto plano, como archivos de configuración y bases de datos, sin corromper los datos o causar problemas de codificación.

2.11. Codificación URL

La codificación URL se utiliza para convertir caracteres especiales y no ASCII en una cadena de texto en su representación hexadecimal precedida por un signo de porcentaje ("%"). Esto se utiliza principalmente para asegurar que los caracteres especiales no afecten la estructura y la integridad de una URL cuando se transmiten a través de Internet.

2.11.1. *Funcionamiento de la codificación URL*

1. **Identificación de caracteres especiales**: Se identifican los caracteres especiales y no ASCII en la cadena de texto que necesitan ser codificados. Estos caracteres incluyen espacios, signos de puntuación, caracteres acentuados y otros caracteres especiales.

2. **Codificación en formato hexadecimal**: Cada carácter especial se convierte en su representación hexadecimal utilizando la tabla ASCII. Por ejemplo, el espacio se convierte en "%20" y el signo de interrogación se convierte en "%3F".
3. **Reemplazo en la URL original**: Los caracteres especiales codificados se reemplazan en la URL original con su representación hexadecimal.
4. **Aplicaciones de codificación URL**
    - **Transmisión de datos en URLs**: Se utiliza en navegadores web y aplicaciones de Internet para codificar parámetros de URL, datos de formularios y otros componentes de URL que contienen caracteres especiales, asegurando una transmisión segura y confiable de datos a través de Internet.
    - **Generación dinámica de URLs**: Es útil en aplicaciones web dinámicas y servicios API donde los datos de entrada pueden contener caracteres especiales, garantizando que las URLs generadas sean válidas y funcionales.
    - **Seguridad de la web**: Ayuda a prevenir ataques de inyección de código y manipulación de URL al garantizar que los datos ingresados por el usuario se codifiquen correctamente antes de ser transmitidos a través de Internet.

3. Metodología

Después de realizar la investigación respectiva sobre los conceptos y herramientas para la detección de objetos con redes neuronales, se llevó a cabo una serie de pasos para su desarrollo y puesta en funcionamiento. Los pasos de este proceso se describen a continuación:

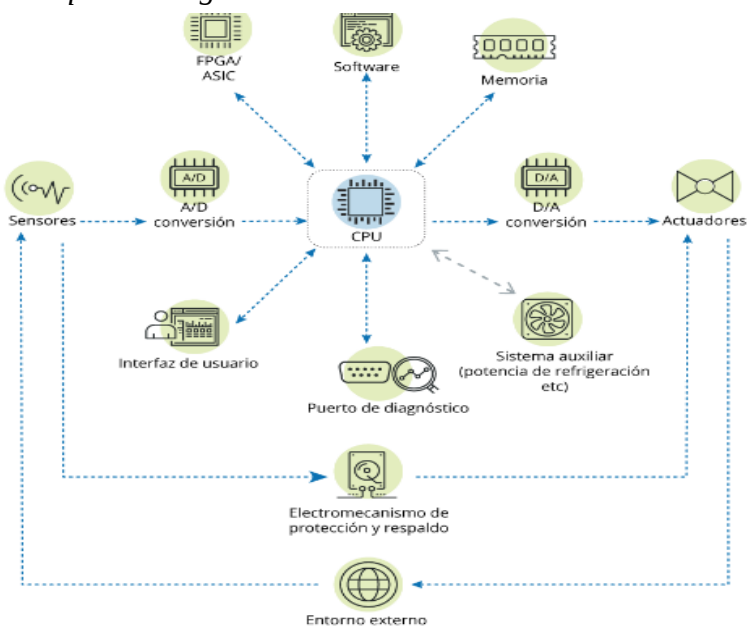
3.1. Selección del sistema embebido

Los sistemas embebidos, a diferencia de los sistemas de propósito general, están diseñados para realizar tareas específicas con recursos de hardware limitados. Esta característica les permite ser eficientes, confiables y económicos. Su diseño optimizado para una función específica requiere de un conocimiento profundo de la aplicación y de técnicas de programación especiales. Los sistemas embebidos se encuentran en una amplia gama de dispositivos y son una parte esencial de la vida moderna (Jaramillo & Potosí, 2017).

Con el tiempo, estos sistemas se han vuelto más sofisticados y han desempeñado un papel importante en la revolución del Internet de las Cosas (IoT). Los sistemas embebidos generalmente incluyen una CPU, memoria RAM y ROM, y se conectan mediante buses de cobre para intercambiar datos con el entorno a través de sensores y actuadores (*Innovación Digital 360*, 2023).

Figura 4

Ejemplo de la arquitectura general de un sistema embebido.



Nota. Tomado de: *Sistemas embebidos: qué son y para qué sirven*. Innovación digital 360. (2023, septiembre 22). https://www.innovaciondigital360.com/iot/sistemas-embebidos-que-son-y-para-que-se-utilizan/

Considerando las características principales de un sistema embebido, se evaluaron cuatro alternativas de implementación propuestas y se selecciona el modelo adecuado para su ejecución.

Las alternativas contempladas son:

- Maix-II-Dock, una plataforma de desarrollo asequible y versátil para el aprendizaje automático e IoT *Maix-II-Dock(M2dock) introduction.* (*s.f.*).
- Jetson Nano, una computadora de placa única de alto rendimiento para la inteligencia artificial (IA) *Jetson Nano* (s.f).
- Rasberry Pi 4 Model B, una excelente opción para una amplia gama de proyectos, desde el aprendizaje de programación hasta la creación de dispositivos IoT *Raspberry Pi 4* (s.f.).
- AMB82 MINI, una placa de desarrollo de bajo costo para el desarrollo de aplicaciones de Internet de las cosas (IoT) e Inteligencia Artificial (IA) *Ameba ARDUINO: Getting Started with AMB82 MINI (RTL8735B)* (s.f.).

Las características principales de los sistemas se han organizado en la Tabla 2, un cuadro comparativo que permite una mejor visualización de la información.

Tabla 2

Características de de Maix-II-Dock, Jetson Nano, Rasberry Pi 4 y AMB82 Mini.

	Sistemas embebidos			
Características	Maix-II-Dock	Jetson Nano	Rasberry Pi 4 Model B	AMB82 mini
Procesador	ARM Single-core Cortex-A7 800-1000MHz	Quad-core ARM® Cortex®-A57 MPCore processor	Quad core 64-bit ARM-Cortex A72 running at 1.5GHz	ARMv8M at 500MHz
NPU	0.2 TOPS	No	No	Intelligent Engine @ 0.4 TOPS
GPU	No	NVIDIA Maxwell™ architecture with 128 NVIDIA CUDA® cores 0.5 TFLOPs (FP16)	Broadcom BCM2711 VideoCore VI	No
Memoria	SIP 64MB DDR2	2GB LPDDR4	4GB LPDDR4-3200 SDRAM.	512KB RAM, support DDR2 128MB
Almacenanmie-nto	Choosable 16M flash(Blank default)	16 GB eMMC 5.1 Flash	Selectable micro SD card	Selectable micro SD card
Wi-Fi	Wi-Fi module*1	Wi-Fi requires external chip	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless	802.11 a/b/g/n 1×1, dual Wi-Fi 2.4GHz/5GHz Simple Wi-Fi Setup
Codificador de video	H.264, up to 1080p@30fps H265, up to 1080p@30fps JPEG, up to 1080p@30fps	250 MP/sec 1x 4K @ 30 (HEVC) 2x 1080p @ 60 (HEVC) 4x 1080p @ 30 (HEVC)	H.265 (4kp60 decode), H.264 (1080p60 decode, 1080p30 encode)	Max 5-megapixel resolution for H.264/H.265 encoding up to 1080p@30fps, 720p@30fps
Pantalla	8bit MCU LCD, can use other screen by convert board	HDMI 2.0 or DP1.2 | eDP 1.4 | DSI (1 x2) 2 simultaneous	2 micro-HDMI® ports (up to 4kp60 supported)	LCD interface | Web graphical user interface
Cámara	2lane MIPI, Up to 1080P@60fps	12 lanes (3x4 or 4x2) MIPI CSI-	5 Mpx 1080P@60fps	JXF37 1920×1080 CMOS Full HD

		2 DPHY 1.1 (18 Gbps)		image sensor with wide view angel FOV 128º optical lens
ADC	1-ch 6bit LRADC for key	No	No	Sí
Audio	LINEOUTP + MICIN1P/N	No	Sí	ADC/DAC/I2S
Ethernet	10/100 Mbit/s Ethernet port with RMII interface	10/100/1000 BASE-T Ethernet	Gigabit Ethernet	No
SPI	SPI x2 (SPI0, SPI1)	Sí	Sí	SÍ
Consumo de energía	5W 5V | 1A	5W, 10W 5V | 1A, 2A	6W 5V | 1.2A	5W 5V | 1A -
Precio	$ 215.961	$ 1’042.902	$ 400.000	$ 96.500

Nota. Tomado de: *Maix-II-Dock(M2dock) introduction.* (*s.f.*), *Jetson Nano* (s.f)., *Raspberry Pi 4* (s.f.) & *Ameba ARDUINO: Getting Started with AMB82 MINI (RTL8735B)* (s.f.).

Después de analizar las especificaciones de las diferentes placas, considerando la relación entre su precio y la disponibilidad en tiendas locales e internacionales, se optó por el sistema embebido AMB82 Mini. Debido a las siguientes características:

- La AMB82 MINI dispone de una NPU integrada para IoT especialmente diseñada para aplicaciones de inteligencia artificial, esta tiene una mayor capacidad que la Maix-II-Dock. Los demás sistemas carecen de esta característica.
- Su procesador de 500MHz le permite ejecutar instrucciones de software, realizar operaciones matemáticas y lógicas de datos. Además, tiene la capacidad de controlar dispositivos de hardware como sensores y actuadores.

- Incorpora una cámara con resolución 1920x1080 de 5 mega pixeles. Con decodificador de video para H.264/H.265 a 720p@30fps y 1080p@30fps.
- Bajo consumo energético, cerca de 1W.
- Su bajo precio en comparación con las demás alternativas de placas.

Esta decisión también se basó en que la placa ofrece una plataforma fácil de programar para el desarrollo de aplicaciones de IA y IoT, lo que facilita la implementación del proyecto. Además, está equipada con una variedad de interfaces periféricas, incluidas WiFi, BLE, GPIO INT, I2C, UART, SPI, PWM, ADC, que permiten la conexión con una amplia gama de componentes electrónicos, como varios tipos de sensores. Además, su capacidad para cargar datos a través de WiFi y su integración con el ecosistema de código abierto, como Arduino, brindan flexibilidad y facilidad de desarrollo (Ameba, s.f., "*Ameba ARDUINO: Getting Started with AMB82 MINI (RTL8735B)*").

3.2. Selección del modelo de aprendizaje automático

Revisando en la página oficial del sistema embebido, este es actualmente compatible con los modelos YOLOv3, v4 y v7 cada uno en su versión Tiny para la detección de objetos. Se puede revisar esta información en la página oficial: *AMB82 AI Model Conversion:* https://www.amebaiot.com/en/amebapro2-ai-convert-model/

Se optó por utilizar YOLOv4 Tiny debido a su mayor velocidad de inferencia en comparación con YOLOv7 Tiny, lo que se traduce en un procesamiento más rápido de los fotogramas capturadas en tiempo real y, pero un poco menos preciso. Así mismo, la versión

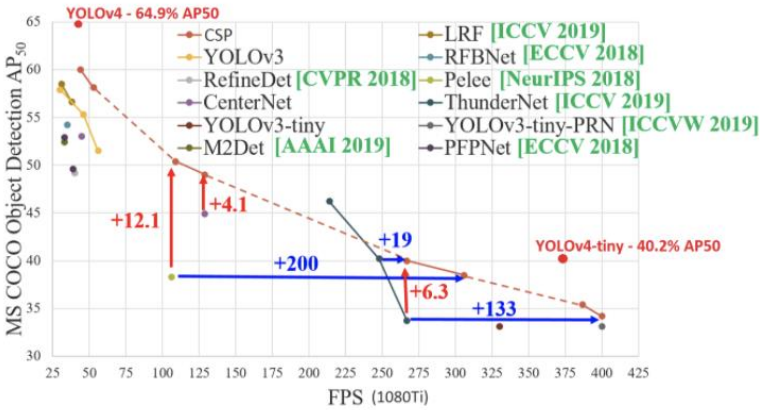
YOLOv3 Tiny es menos precisa que la versión 4 de YOLO Tiny. Esto junto con un factor muy importante como la cantidad de documentación disponible, fueron los aspectos que hicieron decantarse por utilizar el modelo YOLOv4-Tiny, el cual es una versión compacta de YOLOv4 diseñada para entrenar en máquinas con recursos computacionales limitados. Con un peso de alrededor de 16 megabytes, permite el entrenamiento con 350 imágenes en una hora utilizando una GPU Tesla P100. YOLOv4-Tiny logra una velocidad de inferencia de 3 ms en la Tesla P100, lo que lo convierte en uno de los modelos de detección de objetos más rápidos disponibles (Roboflow, 2020, "*YOLOv4 Tiny*").

La arquitectura de YOLOv4-Tiny se basa en modificaciones de la red YOLOv4 original para lograr estas velocidades. Además, se reduce el número de capas convolucionales en la columna vertebral de CSP a un total de 29 capas previamente entrenadas y se reduce el número de capas YOLO a dos en lugar de tres, con menos cuadros de anclaje para la predicción.

A pesar de su tamaño reducido, YOLOv4-Tiny ofrece resultados competitivos en comparación con YOLOv4, alcanzando una precisión media de 40 mAP @.5 en el conjunto de datos de MS COCO como se observa en la Figura 5.

Figura 5

Métricas de rendimiento de YOLOv4 Tiny.



Nota. Tomado de: Solawetz, J., & Sahoo, S. (2020, July 1). *Train YOLOv4-tiny on Custom Data – Lightning Fast Object Detection*. Roboflow Blog. https://blog.roboflow.com/train-yolov4-tiny-on-custom-data-lighting-fast-detection/

3.3. Entrenamiento del modelo de detección de objetos.

Se necesita tener en cuenta algunos conceptos para una mejor comprensión del proceso de entrenamiento:

- **TP (True Positive):** son los casos en los que el modelo clasifica correctamente una muestra como positiva cuando realmente es positiva. En otras palabras, el modelo predice correctamente la presencia de una clase.
- **FP (False Positive):** son los casos en los que el modelo clasifica incorrectamente una muestra como positiva cuando en realidad es negativa.
- **FN (False Negative):** son los casos en los que el modelo clasifica incorrectamente una muestra como negativa cuando en realidad es positiva.

- **TN (True Negative):** son los casos en los que el modelo clasifica correctamente una muestra como negativa cuando en realidad es negativa.
- **Recall:** Mide la capacidad del modelo para identificar correctamente todos los ejemplos positivos (TP) de un conjunto de datos. Este se calcula de la siguiente forma : TP / (TP + FN).
- **Precision:** Mide la capacidad del modelo para evitar identificar ejemplos negativos (FP) como positivos. Se calcula de la siguiente manera: TP / (TP + FP)
- **F1-Score:** es una medida de evaluación del rendimiento de un modelo de clasificación que combina la precisión (precision) y la sensibilidad (recall) en una sola métrica. Es particularmente útil cuando hay un desequilibrio entre las clases en los datos. Se calcula de la siguiente manera:

  2 * (Precisión * Recall) / (Precisión + Recall)
- **IoU (Intersection over Union):** mide la superposición entre la caja delimitadora (bounding box) predicha por el modelo y la caja delimitadora verdadera (ground truth) que rodea al objeto en la imagen. Se calcula como el área de la intersección de ambas cajas dividida por el área de su unión.
- **Dataset:** Es una colección de datos que se utiliza para entrenar y evaluar un modelo de red neuronal.
- **Labels:** Son etiquetas que se asocian a cada ejemplo del dataset para identificar su categoría o valor objetivo.
- **Learning rate:** Es un hiperparámetro que controla la velocidad a la que el modelo actualiza sus pesos durante el entrenamiento.

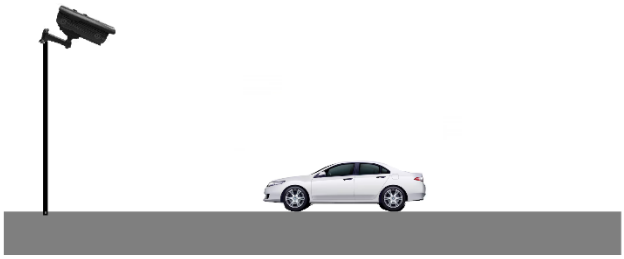
- **Epoch:** Es una iteración completa del proceso de entrenamiento, en la que el modelo se expone a todo el dataset.
- **Batch:** Es un subconjunto del dataset que se utiliza para actualizar los pesos del modelo durante el entrenamiento.
- **Trainig set:** Es la parte del dataset que se utiliza para entrenar el modelo.
- **Test set:** Es la parte del dataset que se utiliza para evaluar el rendimiento final del modelo después del entrenamiento.
- **Validation set:** Es la parte del dataset que se utiliza para ajustar los hiperparámetros del modelo durante el entrenamiento.
- **Lost function:** Es una medida de la discrepancia entre las predicciones del modelo y los valores objetivos.

3.3.1. *Recopilación de datos de entrenamiento.*

El conjunto de datos (*dataset*) fue construido mediante la captura de videos de automóviles en carreteras de un solo sentido, teniendo en cuanta que la cámara usada para la captura de video (Cámara de teléfono inteligente) debía estar posicionada a una altura aproximada de 6 a 9 metros de altura y apuntando hacia abajo como se observa en Figura 6.

Figura 6

Posición de la cámara para captura de video de automóviles en carretera.



Nota. Tomado de: Elaboración propia.

Se capturaron los videos en lugares con un flujo de tráfico moderado en condiciones de buena luminosidad y desde locales comerciales, puentes peatonales y el segundo piso de edificios locales.

Posteriormente se procedió a la extracción de los fotogramas con los vehículos. De esta forma se obtuvo un *dataset* de 2.051 imágenes. Este *dataset* puede visualizarse en el repositorio de Github: https://github.com/KevinSandoval02/Speed_Detection/blob/main/README.md#fotogramas-extra%C3%ADdos-2051

3.3.2. *Preparación de datos de entrenamiento.*

Para el etiquetado de las imágenes de entrenamiento se tuvieron en cuenta varias herramientas como:

- **labelImg**: Herramienta de anotación de imágenes gráficas en Python con interfaz Qt. Guarda anotaciones en XML en formato PASCAL VOC y admite exportación en formatos YOLO y CreateML.
- **LabelMe**: es una herramienta en línea para etiquetar imágenes y generar conjuntos de datos anotados. Proporciona una interfaz web interactiva que permite a los usuarios dibujar y editar etiquetas en imágenes.
- **CVAT (Computer Vision Annotation Tool)**: es una plataforma de código abierto para etiquetar imágenes y videos. Ofrece una amplia gama de funcionalidades, incluyendo la creación de etiquetas, el seguimiento de objetos y la segmentación semántica.
- **Roboflow**: es una plataforma en línea para trabajar con datos de imágenes en proyectos de visión por computadora y aprendizaje automático. Permite el trabajo en conjunto con varios colaboradores.

Se optó por usar la plataforma Roboflow para el etiquetado de las imágenes debido a su amplia gama de herramientas y funciones que facilitan el etiquetado. Las imágenes obtenidas fueron subidas a la plataforma de Roboflow y en esta plataforma se realizó la respectiva asignación de las etiquetas a las imágenes de automóviles. Las imágenes etiquetadas pueden evidenciarse en el repositorio en Github: https://github.com/KevinSandoval02/Speed_Detection/blob/main/README.md#im%C3%A1genes-etiquetadas-en-roboflow-2051

Asimismo, al conjunto de datos se aplicaron diferentes técnicas de procesamiento de imágenes como el redimensionamiento (*resize*) a un tamaño de imágen de 416x416, debido a que es la entrada que admite el modelo YOLOv4-Tiny, el volteo horizontal (*horizontal flip*) y rotación (*rotation*) entre -10º y 10º. Con éstas dos últimas técnicas (*horizontal flip* y *rotation*) se consigue el aumento de datos (*Data augmentation*) creando nuevas imágenes a partir de las imágenes existentes con el objetivo de aumentar la el tamaño y la diversidad del conjunto de datos para mejorar la generalización del modelo. Con todo lo anterior se obtuvieron un total de 4.929 imágenes etiquetadas en formato YOLO Darnet TXT, esto quiere decir que se generan archivos de texto individuales por cada imagen en el *dataset*, donde cada archivo contiene etiquetas detalladas de los objetos detectados. Estas etiquetas se presentan en un formato que incluye el índice de clase del objeto, las coordenadas normalizadas del centro de la caja delimitadora y las dimensiones de esta, todo separado por espacios. Las coordenadas se normalizan para que estén en el rango de 0 a 1, independientemente del tamaño de la imagen original. Además, los índices de clase comienzan desde cero (en este caso el índice 0 pertenece a la única clase *car*) y se asignan a cada clase de objeto detectado en la imagen, lo que permite una representación coherente y consistente de los objetos detectados en el conjunto de datos (Ultralytics, 2023). Se puede acceder al *dataset* de imágenes con aumentación de datos y redimensionamiento en el siguiente enlace en Github:

https://github.com/KevinSandoval02/Speed_Detection/blob/main/README.md#im%C3%A1genes-etiquetadas-en-roboflow-2051

3.3.3. *Entrenamiento de YOLOv4 Tiny*

Para el entrenamiento del modelo se trabajó en el entorno de Google Colab, ya que es una plataforma gratuita basada en la nube que proporciona un uso limitado, pero suficiente en este caso, de una GPU remota que tiene la capacidad de cómputo necesaria para entrenar el modelo usando el framework de darknet con el cual se entrena originalmente. El código para el entrenamiento se encuentra en el repositorio en Github: https://github.com/KevinSandoval02/Speed_Detection/blob/main/README.md#entrenamiento-de-yolov4-tiny

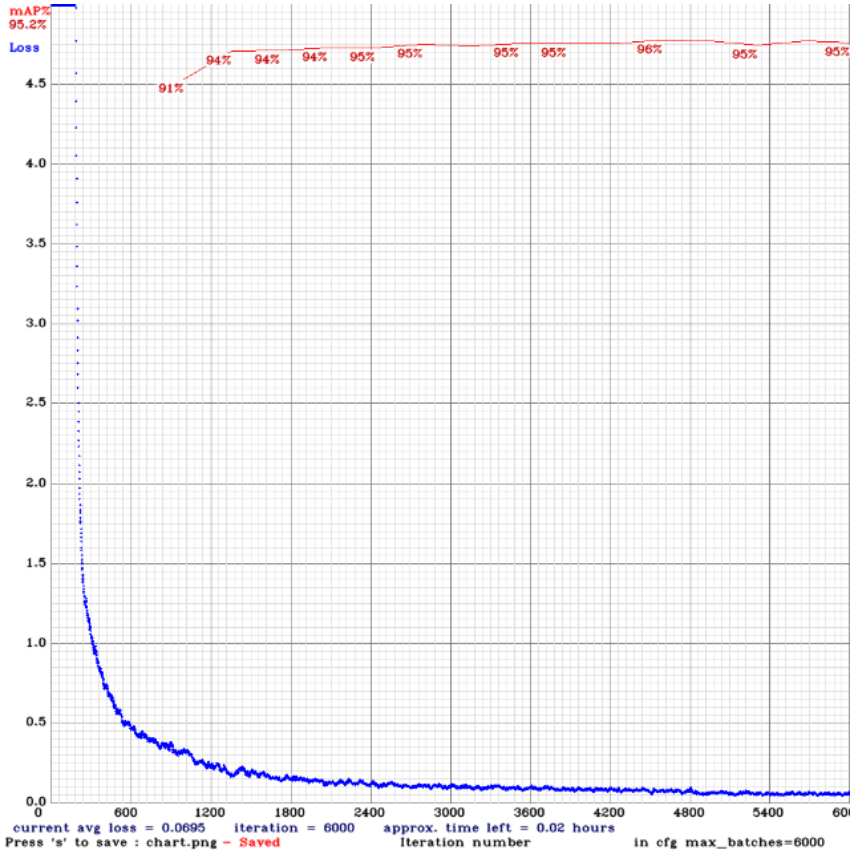
En el entrenamiento se modificó las subdivisiones de los lotes para que aumentara la cantidad de imágenes procesadas simultáneamente durante el entrenamiento y con esto el entrenamiento tenga una duración menor. Al terminar de entrenarse el modelo en Colab, se crean los archivos de configuración (.cfg) que contienen la configuración de la arquitectura de la red neuronal convolucional y los archivos de los pesos (.weights) los cuales contienen los parámetros aprendidos por la red neuronal convolucional. Estos archivos junto al código de entrenamiento en Colab pueden consultarse en el repositorio de Github en: https://github.com/KevinSandoval02/Speed_Detection/blob/main/README.md#entrenamiento-de-yolov4-tiny

Junto a estos archivos, se imprime en pantalla las siguientes métricas de evaluación de la red neuronal:

- Precisión = 0.90
- Recall = 0.95
- F1-Score = 0.92
- TP = 780
- FP = 88
- FN = 45
- Average IoU = 77.76 %
- Mean average precisión (mAp@0.50) = 0.951964

Figura 7

Curva de pérdida y precisión durante el entrenamiento del modelo



3.4. Desarrollo del sistema de medición de velocidad vehicular

Considerando los recursos computacionales de la AMB82-MINI, se desarrolló un algoritmo basado en el análisis de la variación temporal de los datos capturados por el modelo de aprendizaje automático para la detección de automóviles.

El código desarrollado en la IDE de Arduino se encarga de capturar los datos, procesarlos y calcular la velocidad del automóvil en función de la variación temporal de la posición de dicho vehículo. Este enfoque permite obtener mediciones de velocidad en tiempo real con un consumo de recursos adecuado para la plataforma de hardware disponible.

A continuación, se describe brevemente las secciones del código:

- **Librerías incluidas**: El código incluye varias librerías necesarias para diferentes funciones, como la conexión WiFi, la transmisión de video, la detección de objetos, la grabación de video en una tarjeta microSD, y la codificación de imágenes en Base64.
- **Definición de constantes y configuraciones**: Se definen varias constantes y configuraciones importantes, como los canales de video para la transmisión y detección de objetos, el nombre de archivos, las coordenadas de las zonas de detección de velocidad, la velocidad máxima permitida mediante la variable ***TopSpeed***, y otras configuraciones relacionadas con la cámara y la red WiFi.
- **Inicialización y configuración**: En la función ***setup()***, se inicializan los componentes necesarios, como la conexión WiFi, la configuración de la cámara y los canales de video, y se configura el protocolo RTSP para la transmisión de video en tiempo real.

- **Bucle principal**: En la función ***loop()***, se ejecuta el bucle principal del programa. En este caso, la función espera a que se active una variable booleana para capturar una imagen cuando se detecte un exceso de velocidad.
- **Funciones de detección de objetos y post-procesamiento**: La función ***ODPostProcess()*** se utiliza como un callback para el procesamiento de resultados de detección de objetos. Aquí se detectan los objetos en el video, se dibujan los cuadros delimitadores y se comprueba si un vehículo pasa por las líneas de detección de velocidad.
- **Funciones de captura y procesamiento de imágenes**: Las funciones ***CaptureImage()***, ***RenameVideo()*** *y* ***DeleteVideo()*** se encargan de capturar imágenes cuando se detectan excesos de velocidad, renombrar archivos de video y eliminar archivos de video en caso de no existir un exceso de velocidad, respectivamente.
- **Funciones auxiliares y de utilidad**: Se incluyen varias funciones auxiliares, como ***dist()*** para calcular la distancia entre dos puntos, ***isPointOnLine()*** para verificar si un punto está en una línea, ***touchesLineCenter()*** para verificar si el centro de un objeto toca una línea, ***SpeedHistory()*** para mostrar el historial de velocidades, y ***urlencode()*** para codificar cadenas de texto en formato URL.

El código descrito previamente se puede visualizar a detalle en el repositorio de GitHub: https://github.com/KevinSandoval02/Speed_Detection/tree/72812e4903fdf94eb77d3265bbef1adc75d64434/Speed_Detection_

3.4.1. *Implementación del sistema*

La implementación del sistema implica una serie de pasos que van desde la configuración del entorno de desarrollo hasta la calibración del sistema final. Este apartado detalla cada uno de estos pasos, proporcionando una visión completa del proceso de implementación.

3.4.2. *Configuración del entorno de desarrollo*

Desarrollar correctamente este paso garantiza una implementación exitosa del sistema en la placa AMB82 Mini. Esta placa ofrece una plataforma versátil e intuitiva de programar para desarrollar una amplia gama de aplicaciones de IoT, aprovechando sus múltiples interfaces periféricas y características clave.

La integración de la AMB82 Mini con el mundo del código abierto se facilita mediante el uso del entorno de desarrollo Arduino, que ofrece acceso a una amplia gama de recursos, incluidos kits de desarrollo de hardware (HDK), kits de desarrollo de software (SDK), documentos API y guías de ejemplo. Los pasos para su configuración se describen en el repositorio de GitHub:

https://github.com/KevinSandoval02/Speed_Detection/blob/main/README.md#pasos-para-configurar-el-entorno-de-desarrollo

3.4.3. *Integración del modelo de YOLO v4 Tiny en la placa AMB82 Mini*

La integración del modelo YOLOv4 Tiny en la placa AMB82 Mini implica varios pasos clave. En primer lugar, el modelo se entrena utilizando una base de imágenes y se generan los archivos .cfg y .weights.

Luego, se accede a la página de conversión de modelos de AMB82-MINI en AMB82 AI Model Conversion y se completa el proceso de conversión, proporcionando los archivos necesarios y especificando el tipo de modelo y cuantización. Una vez convertido, el archivo resultante se descarga y se renombra como yolov4_tiny.nb antes de ser reemplazado en la ruta del sistema operativo. Se ajusta el archivo ObjectClassList.h para reflejar las clases identificadas por el modelo. Finalmente, el modelo se prueba cargando el código ejecutable Arduino y el archivo de clases actualizado en la placa AMB82 Mini, lo que permite su implementación en aplicaciones de detección de objetos en tiempo real.

Este proceso se describe a detalle en el repositorio de GitHub:

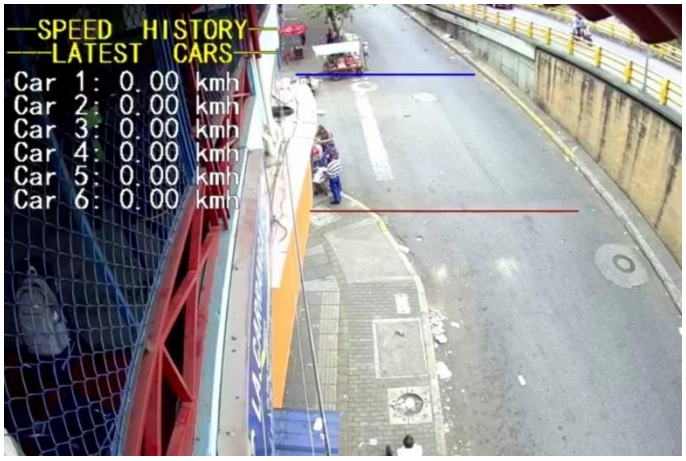
https://github.com/KevinSandoval02/Speed_Detection/blob/main/README.md#integraci%C3%B3n-del-modelo-de-yolo-v4-tiny-en-la-placa-amb82-mini

3.4.4. *Calibración del sistema*

Para la calibración se optó inicialmente por poner en marcha el sistema en un entorno controlado en el cual se instaló el dispositivo en el segundo piso de un local comercial a una altura de ocho metros, en donde se designó la zona de medición de velocidad sobre la carretera (líneas azul y roja) configurándose una distancia de quince metros previamente medidos entre las dos líneas. Esta distancia se asignó teniendo en cuenta la altura de la cámara y su ángulo de visión, con el fin de poder garantizar una identificación correcta de los vehículos considerando los datos con los que se entrenó el modelo de detección objetos.

Figura 8

Entorno de pruebas.



Posteriormente se grabaron y registraron todas las estimaciones que realizaba el sistema para seguidamente realizar un análisis de precisión.

Una vez teniendo las grabaciones se parte del algoritmo implementado en la AMB82-MINI para determinar la velocidad, en donde se registra el tiempo de entrada del carro en la zona de detección, se registra el tiempo de salida del carro en la zona de detección, y teniendo la distancia en este caso de quince metros, se realiza el cálculo de velocidad con la siguiente ecuación:

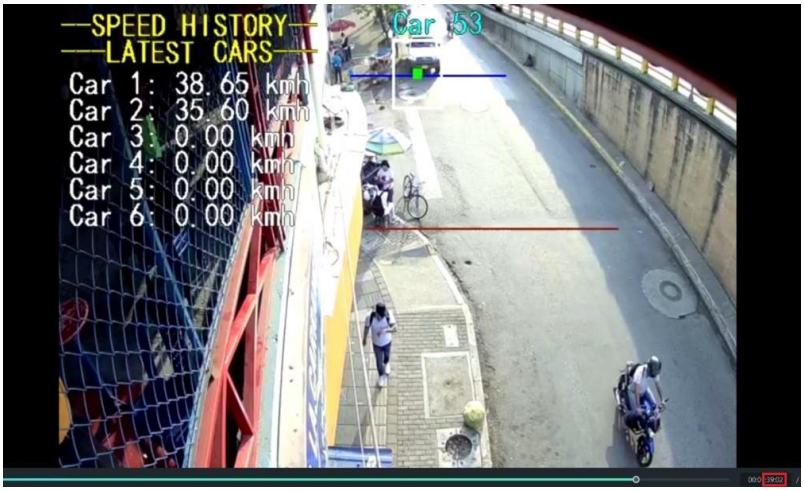
$$v = \frac{d}{t} = \frac{desplazamiento}{tiempo\ empleado}$$

En este caso el desplazamiento corresponde a quince metros y el tiempo empleado a la diferencia entre el tiempo registrado de salida y de entrada en la zona de detección.

Para verificar este tipo de pruebas se hizo uso de una aplicación de reproducción de video para tomar correctamente las lecturas de tiempo de la siguiente forma:

Figura 9

Ejemplo de entrada del automóvil en la zona de detección.

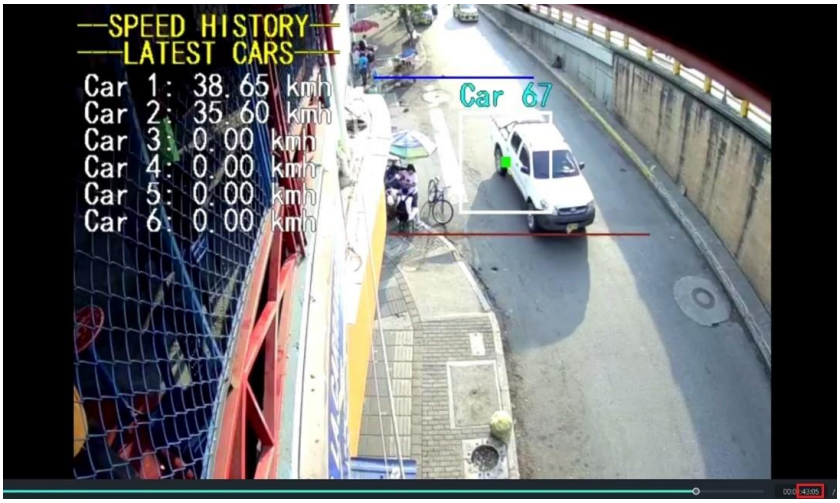


En la Figura 9 se observa un automóvil entrando a la zona de medición, usando sus llantas frontales y parachoques como referencia, y en la parte inferior derecha observamos el tiempo transcurrido hasta el momento en el tramo de video siendo de 39 segundos y $\frac{2}{22}$ fracciones de segundo, siendo 22 el total de fracciones de segundo con las que trabaja el reproductor de video. En el historial se reflejan las velocidades de los últimos seis automóviles en orden ascendente, una vez se estima la velocidad de un séptimo automóvil el historial se ajusta en ceros y se sitúa la velocidad del séptimo vehículo en la línea *Car 1* disponiendo el historial para los próximos vehículos.

Se destaca que, si bien el algoritmo implementado permite la estimación de la velocidad de varios vehículos al tiempo, se ajustó el código para grabar hasta dos tramos de video en paralelo teniendo en cuenta la capacidad de almacenamiento multimedia de la AMB82-MINI. Por lo cual la zona de detección deberá de tener distancia recomendada entre los diez a quince metros con el fin de garantizar que solo hasta dos vehículos entren en la zona de detección de manera simultánea.

Figura 10

Ejemplo de salida del automóvil de la zona de detección.

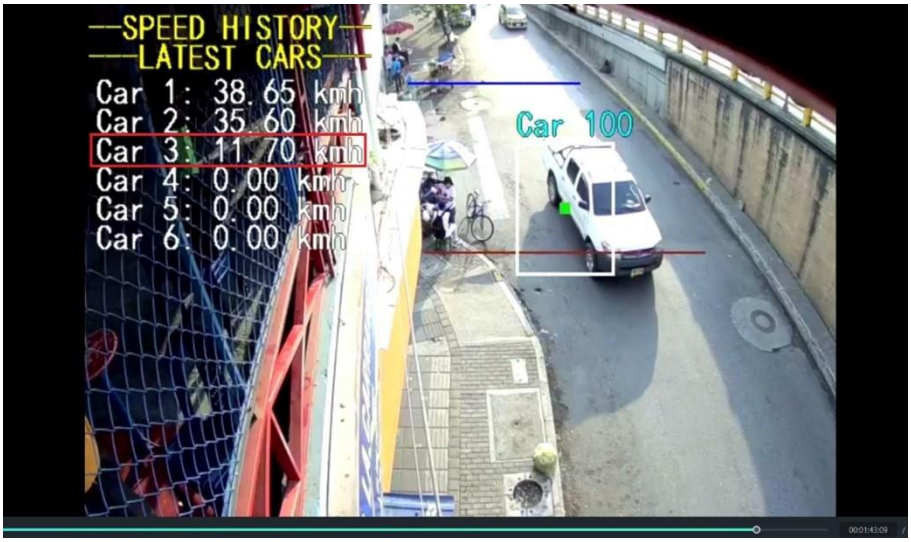


En la Figura 10 se observa el automóvil saliendo de la zona de medición y en la parte inferior derecha observamos el tiempo transcurrido hasta el momento en el tramo de video siendo de 43 segundos y $\frac{5}{22}$ fracciones de segundo.

Y un instante después el historial de velocidades se actualiza en pantalla con la estimación de la velocidad del vehículo como se observa en Figura 10.

Figura 11

Ejemplo de velocidad estimada por el sistema.



Teniendo los tiempos de entrada y salida ya registrados, se realizó el cálculo teórico de velocidad de la siguiente forma, siendo 3.6 el factor de conversión de m/s a km/h:

$$v = \frac{d}{t} = \frac{15}{\left(43 + \frac{5}{22}\right) - (39 + \frac{2}{22})} * 3.6 = 13.05\ km/h$$

De esta forma la velocidad teórica aproximada del vehículo sería de 13.05 km/h y la estimada por el sistema de 11.7 km/h.

Esta misma metodología se llevó a cabo con un total de veinticinco vehículos mientras se ajustaba el parámetro *Tolerance* con el fin de obtener la mejor estimación posible, este parámetro se utiliza para definir un rango de aceptación alrededor de las líneas guía de la zona de detección, este rango de aceptación determina cuán cerca debe estar el centro de masa del cuadro de detección, de la línea guía para considerarse “en la línea”, siendo el centro de masa el punto de color verde ubicado en el centro del recuadro que encierra el vehículo detectado por el modelo de detección de objetos.

Tras las pruebas se obtuvo la siguiente tabla de datos:

Tabla 3

Tabla de datos de calibración.

Carro	Velocidad teórica (km/h)	Velocidad estimada (km/h)	Discrepancia entre velocidades	%Error	Tolerance
1	36.0	29.74	6.26	17.38%	1
2	33.23	29.32	3.91	11.76%	1
3	43.2	38.65	4.55	10.53%	1
4	43.2	37.95	5.25	12.15%	1
5	39.27	44.33	5.06	12.88%	0.7
6	54.0	50.51	3.49	6.46%	0.7
7	58.9	62.28	3.38	5.73%	0.7
8	33.23	30.84	2.39	7.19%	0.5
9	34.92	32.39	2.53	7.24%	0.5
10	46.28	44.33	1.95	4.21%	0.5
11	36.0	33.71	2.29	6.36%	0.5
12	64.8	64.83	0.03	0.04%	0.5
13	35.02	33.71	1.31	3.74%	0.5
14	48.0	46.51	1.49	3.105%	0.3
15	36.24	34.95	1.29	3.55%	0.3
16	20.34	18.97	3.37	6.73%	0.3
17	39.27	38.63	0.64	1.62%	0.2
18	44.68	46.51	1.83	4.09%	0.2
19	13.05	11.7	1.98	15.17%	0.2
20	17.51	16.53	0.98	5.59%	0.2

Como resultado de las pruebas realizadas se optó por establecer el factor de tolerancia en 0.2 siendo el valor con el cual se obtuvo una menor discrepancia entre velocidades. Las pruebas realizadas se pueden visualizar en el enlace: pruebas de calibración.

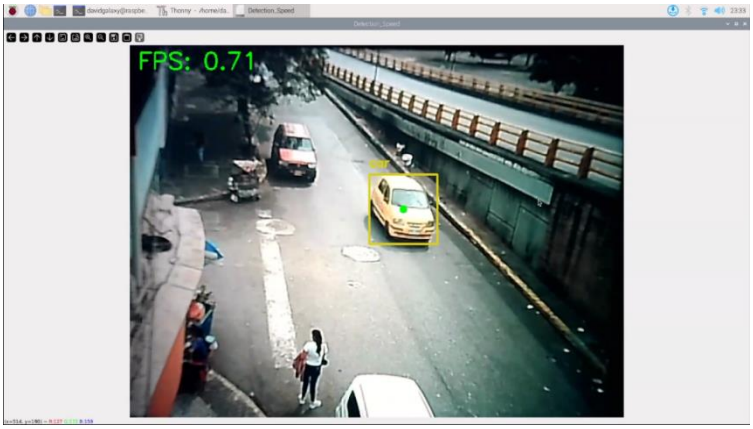
3.5. Detección de objetos Rasberry Pi 4

A modo de una pequeña comparación en rendimiento, durante el proceso de calibración del sistema de la AMB82 MINI, se utilizaron los mismos pesos y el archivo de configuración (.cfg)

del modelo de detección de objetos para ser ejecutados en la Rasberry Pi 4. Se elaboró un programa en Python el cual utiliza los parámetros del modelo para detectar los automóviles, obteniendo así las siguientes observaciones.

Figura 12

Detección de objetos con YOLOv4 Tiny en Rasberry Pi 4 Modelo B.



Como se observa en Figura 12, la detección de uno de los vehículos es precisa en cuanto a la caja delimitadora y se resalta que la velocidad de procesamiento fue un promedio de 0.71 FPS en contraste con la placa AMB82 MINI que muestra los cuadros de detección en un promedio de 5 FPS, durante la detección se observó que los vehículos que transitan por el sector fueron detectados correctamente, sin embargo, no fue el caso para uno de los vehículos que permanecía detenido. En general, la Rasberry fue capaz de ejecutar un modelo de detección de objetos bastante preciso, con la desventaja de que su velocidad de procesamiento es baja. La carencia de una NPU en la Rasberry es evidente en este caso específico en comparación con el rendimiento que otorga la placa AMB82 MINI.

El video del funcionamiento de la Rasberry Pi 4 se puede evidenciar en el siguiente enlace: Rasberry Pi 4 - Detección de objetos con Yolov4 Tiny.

3.6. Monitoreo de excesos de velocidad

El sistema de monitoreo de excesos de velocidad consta de varias etapas, desde la captura de video e imágenes hasta la gestión de excesos de velocidad capturados y notificaciones. La ejecución de cada apartado se puede visualizar en el enlace: monitoreo de excesos de velocidad.

3.6.1. *Captura de video de excesos de velocidad*

Cuando se detecta un automóvil en el área de interés, se inicia una grabación de video cuya duración está condicionada al tiempo que tarda el vehículo en atravesar la zona designada para la medición de velocidad. Si el automóvil sobrepasa el límite de velocidad permitido en el área de interés mediante la variable definida **TopSpeed**, se almacena la grabación del video que registra el exceso de velocidad y posteriormente se renombra con el formato **Speeding_Violation_#** mediante la función **RenameVideo()**. Este video se almacena en la tarjeta microSD extraíble de la AMB82-MINI para su posterior tratamiento.

En caso de que el automóvil no sobrepase el límite de velocidad definido, el video será eliminado haciendo uso de la función ***DeleteVideo()***, de esta forma solo se almacenarán los tramos de video correspondientes a excesos de velocidad.

3.6.2. *Captura de imágenes de excesos de velocidad*

Además de la grabación de video, con la función ***CaptureImage()*** se captura una imagen en el momento en que se identifica un exceso de velocidad. Esta imagen incluye la información del historial de velocidades con la velocidad estimada del vehículo infractor y se utiliza como evidencia complementaria.

3.6.3. *Codificación de imágenes capturadas*

Las imágenes capturadas se codifican en el formato Base64 y en codificación URL para facilitar su transferencia y almacenamiento en la nube. Este proceso de codificación convierte las imágenes en cadenas de texto que pueden ser fácilmente manejadas y enviadas a través de protocolos web. Una vez codificada se establece la conexión con un script de Google Apps Script para realizar el envío inmediato de la imagen a la nube en Google Drive.

3.6.4. *Transferencia de imágenes a Google Drive*

Las imágenes codificadas se transfieren a Google Drive utilizando un script de Google Apps. Este script recibe las imágenes y las almacena en una carpeta específica en la cuenta de Google Drive asociada al proyecto. El Script se puede encontrar en el repositorio de GitHub: https://github.com/KevinSandoval02/Speed_Detection/tree/main/Script

Figura 13

Script utilizado para decodificar y almacenar imágenes en Google Drive.

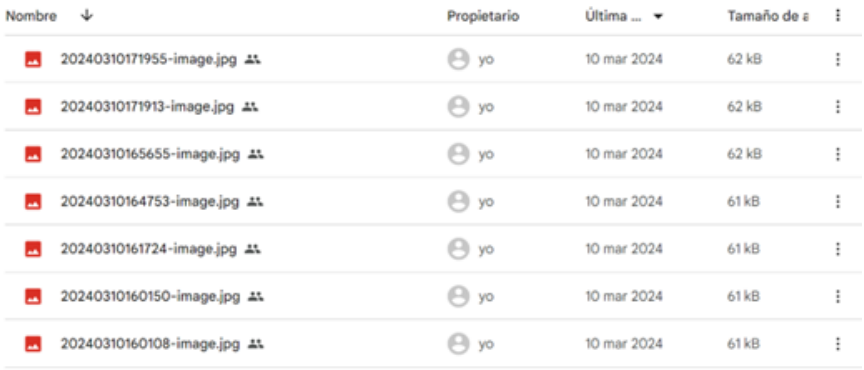
```
function doPost(e) {
// Retrieve variables from the POST requests
var myFoldername = e.parameter.myFoldername;
var myFile = e.parameter.myFile;
var myFilename = Utilities.formatDate(new Date(), "GMT-5", "yyyyMMddHHmmss")+"-"+e.parameter.myFilename;
var myToken = e.parameter.myToken;

// Store the file type and Base64 encoded data
var contentType = myFile.substring(myFile.indexOf(":")+1, myFile.indexOf(";"));
var data = myFile.substring(myFile.indexOf(",")+1);
data = Utilities.base64Decode(data);
var blob = Utilities.newBlob(data, contentType, myFilename);

// Save a captured image to Google Drive.
var folder, folders = DriveApp.getFoldersByName(myFoldername);
if (folders.hasNext()) {
folder = folders.next();
} else {
folder = DriveApp.createFolder(myFoldername);
}
var file = folder.createFile(blob);
file.setDescription("Uploaded by " + myFilename);

// Returning Results
return ContentService.createTextOutput(myFoldername+"/"+myFilename+"\n"+imageUrl+"\n"+res);
}
```

3.6.5. *Gestión de excesos de velocidad en Google Drive*

En Google Drive, las imágenes capturadas se organizan y gestionan automáticamente. Cada archivo se nombra de manera única y se categoriza según el siguiente formato *yyyyMMddHHmmss-image.jpg*. Esto facilita la identificación y el acceso a cada exceso de velocidad registrado.

Figura 14

Formato de designación en Google Drive.



3.6.6. *Integración de plataformas con IFTTT*

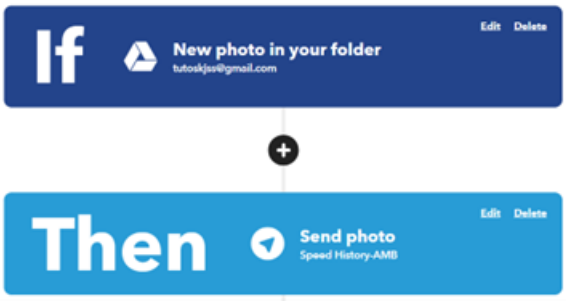
La gestión de excesos de velocidad capturados se integra con el servicio en línea IFTTT (If This Then That) el cual admite acciones automáticas en respuesta a eventos específicos. IFTTT permite crear condicionales simples conocidos como Applets, los cuales se pueden configurar para enviar notificaciones o alertas cuando se registra una acción.

En este proyecto, se utiliza IFTTT para conectar Google Drive con la aplicación de mensajería instantánea Telegram.

Disponible en: https://web.telegram.org/

Figura 15

Applet utilizado en IFTTT.



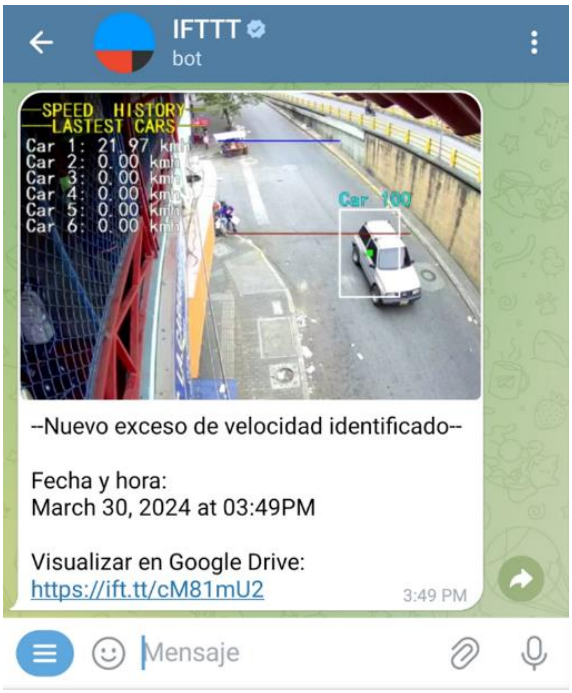
Disponible en: https://ifttt.com/

3.6.7. *Notificaciones vía Telegram*

La plataforma IFTTT envía un mensaje de texto a la cuenta de Telegram asociada cada vez que se carga una nueva imagen en la carpeta de Google Drive. En este mensaje se identifica la fecha y hora del exceso de velocidad identificado, y de igual forma se adjunta la imagen capturada.

Figura 16

Mensaje de notificación en Telegram.



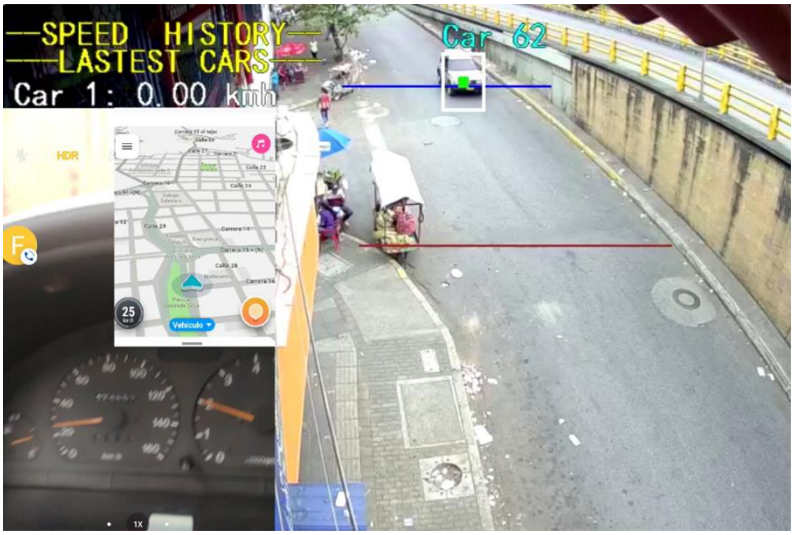
4. Pruebas y análisis de resultados

En esta sección se evaluó el rendimiento del modelo de detección de velocidad de automóviles implementado en la AMB82-MINI. Se realiza un análisis de la precisión utilizando datos recopilados durante unas pruebas controladas.

En este caso se dispone de la conducción de un automóvil con el objetivo de comparar la velocidad por GPS del vehículo con la estimada por el sistema implementado en la AMB82-MINI, para ello nos ubicamos nuevamente en el segundo piso del local comercial a una altura de ocho metros, en donde se designó la zona de medición de velocidad configurándose con una distancia de quince metros.

Figura 17

Ambiente de pruebas finales.



Nuevamente se grabaron y registraron las estimaciones que realizó el sistema, registrando un total de dieciséis pruebas con el automóvil, donde la velocidad real corresponde a la velocidad aproximada generada por GPS y la velocidad estimada corresponde a la velocidad obtenida por el sistema de detección de velocidad de vehículos.

Tabla 4

Resultados prueba de precisión.

Carro	Velocidad real (km/h)	Velocidad estimada (km/h)	Discrepancia entre velocidades	%Error
1	23	24.6	1.6	6.95%

2	31	30.86	0.14	0.45%
3	37	35.62	1.38	3.72%
4	22	21.97	0.03	0.13%
5	38	38.68	0.68	1.78%
6	48	51.92	3.92	8.16%
7	30	30.82	0.82	2.73%
8	26	25.64	0.36	1.38%
9	24	24.56	0.56	2.33%
10	34	30.8	3.2	9.41%
11	23	25.6	2.6	11.30%
12	32	30.82	1.18	3.68%
13	30	30.87	0.87	2.9%
14	29	30.84	1.84	6.34%
15	30	30.82	0.82	2.73%
16	37	38.72	1.72	4.64%

Las pruebas realizadas se pueden visualizar en el enlace: pruebas de precisión.

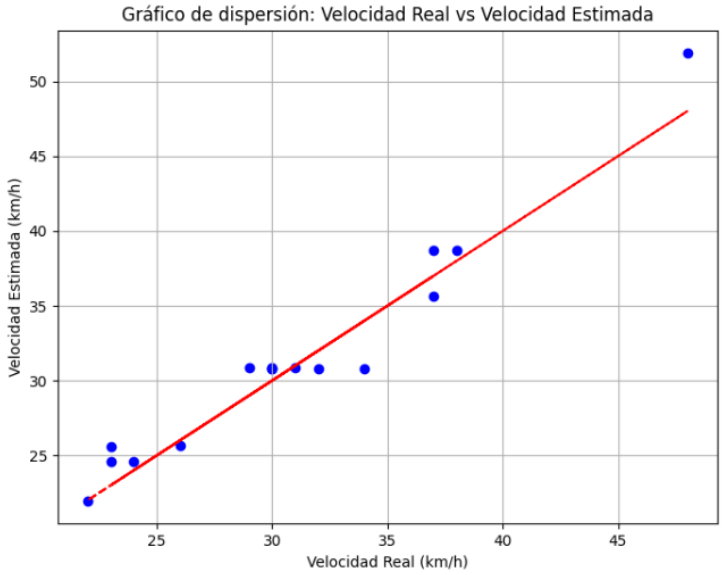
4.1. Análisis de la precisión

Al analizar los datos de la tabla 5, se obtuvieron las siguientes métricas:

- **Media de la diferencia**: Se calcula la diferencia entre las velocidades estimadas y las velocidades reales, obteniendo un valor promedio de 0.57125 km/h.
- **Desviación estándar de la diferencia**: La variabilidad de las diferencias entre las velocidades estimadas y las reales se cuantifica mediante una desviación estándar de 1.6273 km/h.

Figura 18

Gráfico de dispersión.



El gráfico de dispersión presenta la relación entre las velocidades reales y las velocidades estimadas con el trazo de una línea de referencia a 45°. Además, se calcula el porcentaje de error promedio obteniendo un valor del 4.2945%, lo que indica una precisión aceptable del sistema en la detección de velocidad en automóviles.

4.2. Beneficios y consideraciones

La implementación de este sistema de monitoreo de excesos de velocidad ofrece varios beneficios, como la contribución a mejorar la seguridad vial y una mejor aplicación de las leyes

de tránsito. Presentando una alternativa de bajo costo a la contribución de la problemática de accidentalidad producida por excesos de velocidad en zonas delimitadas.

El almacenamiento de información de video e imagen son de gran aporte para el monitoreo y soporte a los excesos de velocidad realizados. Sin embargo, es importante considerar aspectos como la privacidad de los datos recopilados y el cumplimiento de las regulaciones locales sobre el uso de sistemas de vigilancia.

El sistema fue creado para trabajar de manera adecuada en carreteras de un solo carril con vista frontal a los vehículos, por lo que se aconseja implementarse en este mismo tipo de situaciones y a una altura entre los siete a doce metros para designar una zona de detección suficientemente amplia. Así mismo, se recomienda implementarse en entornos con buena iluminación, puesto que el sistema no está diseñado para funcionar en ambientes con baja luminosidad y la detección de automóviles se verá muy limitada.

5. Conclusiones

El modelo YOLOv4 Tiny en este caso demostró ser una opción eficiente y precisa para la detección de objetos. Teniendo en cuenta su tamaño reducido, logró identificar los automóviles con una precisión satisfactoria, lo que lo hace adecuado para implementaciones en tiempo real con recursos limitados.

La integración del modelo en la placa AMB82 Mini se realizó de manera fluida siguiendo los pasos detallados en este proyecto. La arquitectura y los recursos de la placa fueron suficientes para ejecutar el modelo sin problemas, lo que demuestra su viabilidad para aplicaciones de detección de objetos embebidas.

La implementación del sistema de detección de velocidad de automóviles en la AMB82-MINI demostró una precisión aceptable en la estimación de velocidades, con una desviación estándar de 1.6273 km/h y un error promedio de 4.2945%.

El sistema es capaz de decodificar video y transmitirlo mediante el protocolo de transmisión en tiempo real RTSP (Real Time Streaming Protocol) a 30 FPS, sin embargo, la detección de objetos utilizando YOLOv4 Tiny se realizó a una velocidad promedio de 5 FPS, lo que indica una limitación en la capacidad de procesamiento del sistema para realizar esta tarea, aun así, se obtuvo un buen rendimiento del modelo implementado, logrando una precisión aceptable en las mediciones.

Tras las pruebas realizadas, con un valor de *Tolerance* de 0.2 se obtuvieron resultados aceptables en la detección de velocidad de los automóviles que no excedían los 65 km/h. Por lo tanto, si se desea tener un rango de operación más amplio, se debe aumentar el valor de esta variable teniendo en cuenta que el porcentaje de error de la medición también aumentará.

6. Trabajo futuro

Dando seguimiento a este proyecto de investigación, y considerando su naturaleza interdisciplinaria que abarca diversas áreas del conocimiento, existe la oportunidad de extender el estudio realizado como un área de investigación futura para otros investigadores. Este proyecto proporciona una base sólida para explorar nuevas perspectivas y enfoques en el campo, ofreciendo potenciales vías para la expansión y la profundización del conocimiento en el tema. Para futuras investigaciones, se podría explorar los siguientes aspectos:

- **Ampliación del conjunto de datos y clases**: Se puede recopilar y etiquetar un conjunto de datos más grande y diverso para entrenar el modelo, lo que permitirá la detección de una gama más amplia de vehículos. Además, se puede explorar la detección de objetos en tiempo real en entornos dinámicos y cambiantes.
- **Medición de velocidad en ambos sentidos**: Investigar y desarrollar técnicas para permitir que el sistema embebido mida la velocidad de los vehículos en ambos sentidos del flujo de tráfico. Esto podría requerir el uso de múltiples cámaras o técnicas de procesamiento de imágenes más sofisticadas para detectar la dirección del movimiento de los vehículos.
- **Mejora de la capacidad de medición de velocidad múltiple**: Explorar formas de aumentar la capacidad del sistema para medir la velocidad de más de dos vehículos simultáneamente. Esto podría implicar mejoras en el algoritmo de seguimiento de objetos o la utilización de cámaras adicionales para cubrir más áreas de la carretera.
- **Integración de reconocimiento de placas**: La incorporación de esta funcionalidad proporcionaría una valiosa capa adicional de información para el sistema, permitiendo la identificación única de cada vehículo y facilitando la gestión y seguimiento de vehículos en tiempo real.

En conclusión, el trabajo futuro en este proyecto tiene como objetivo continuar mejorando la precisión, eficiencia y robustez del sistema de detección de objetos en la placa AMB82 Mini, así como explorar nuevas aplicaciones, nuevas placas de desarrollo y funcionalidades para satisfacer las necesidades emergentes en el campo de la visión artificial.

Lista de referencias

Ameba. (s.f.). AmebaPro2 Datasheet Download. Recuperado el 12 de diciembre del 2023 de: https://www.amebaiot.com/zh/datasheet-download-amb82-mini/

Ameba. (s.f.). AMB82 AI Model Conversion. Recuperado el 12 de diciembre del 2023 de: https://www.amebaiot.com/en/amebapro2-ai-convert-model/

Bamoumen, H., Temouden, A., Benamar, N., & Chtouki, Y. (2022). How TinyML Can Be Leveraged to Solve Environmental Problems: A Survey. En 2022 3rd International Conference on Innovation and Intelligence for Computing, Communication and Technologies (3ICT) (pp. 338-343). IEEE. doi: 10.1109/3ICT56508.2022.9990661

IBM. (s.f.). What is PyTorch?. Recuperado el 2 de diciembre de 2023 de: https://www.ibm.com/topics/pytorch#:~:text=PyTorch%20is%20a%20software%2Dbased,for%20academic%20and%20research%20communities

IBM. (s.f.). ¿Qué es la inteligencia artificial (IA)? Recuperado el 1 de noviembre de 2023 de: https://www.ibm.com/mx-es/topics/artificial-intelligence

IBM. (s.f.). ¿Qué es machine learning?. Recuperado el 1 de noviembre de 2023 de: https://www.ibm.com/mx-es/topics/machine-learning

IBM. (s.f.). ¿Qué son las redes neuronales convolucionales? Recuperado el 1 de noviembre de 2023 de: https://www.ibm.com/es-es/topics/convolutional-neural-networks

Innovación Digital 360. (2023). Sistemas embebidos: qué son y para qué se utilizan. Recuperado de: https://www.innovaciondigital360.com/iot/sistemas-embebidos-que-son-y-para-que-se-utilizan/

Jaramillo, J. & Potosí, J. (2017). Diseño de un sistema embebido en una tarjeta de desarrollo Altera De2 para el conteo de ingreso de personas a través de detección y seguimiento facial. Recuperado de: https://sired.udenar.edu.co/9389/1/92379.pdf

Keras. (s.f). Keras, Simple. Flexible. Powerful. Recuperado el 2 de diciembre de 2023 de: https://keras.io/

NVIDIA. (s.f.). Kit para Desarrollador Jetson Nano. Recuperado de: https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano/product-development/

Raspberry Pi. (s.f.). Raspberry Pi 4 Model B Datasheet. Recuperado de: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. Recuperado de: https://arxiv.org/pdf/1506.02640.pdf

Redmon, J. (2013-2016). Darknet: Open Source Neural Networks in C. Recuperado de: http://pjreddie.com/darknet/

Roboflow. (2020). YOLOv4-tiny. Recuperado el 12 de diciembre del 2023 de: https://roboflow.com/model/yolov4-tiny

Solawetz, J., & Sahoo, S. (2020, julio 1). Train YOLOv4-tiny on Custom Data – Lightning Fast Object Detection. Roboflow Blog. Recuperado de: https://blog.roboflow.com/train-yolov4-tiny-on-custom-data-lighting-fast-detection/

Sarker, I.H. (2021). Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. SN Computer Science, 2(420). doi: 10.1007/s42979-021-00815-1

TensorFlow Developers. (s.f.). TensorFlow. GitHub. Recuperado el 2 de diciembre de 2023 de: https://github.com/tensorflow/tensorflow

Tan, T., & Cao, G. (2022). Deep Learning on Mobile Devices Through Neural Processing Units and Edge Computing. En IEEE INFOCOM 2022 – IEEE Conference on Computer Communications (pp. 1209-1218). doi: 10.1109/INFOCOM48880.2022.9796929

Ultralytics. (2023). Visión general de los conjuntos de datos de detección de objetos. Recuperado de: https://docs.ultralytics.com/es/datasets/detect/#ultralytics-yolo-format

Y. Liu, Y. Lu, Q. Shi y J. Ding. (2013). Optical Flow Based Urban Road Vehicle Tracking. En 2013 Ninth International Conference on Computational Intelligence and Security (pp. 391-395). doi: 10.1109/CIS.2013.89

Apéndice

Apéndice A. Repositorio en Github Speed_Detection

En el siguiente enlace público de la plataforma Github se podrá ver con más detalle los aspectos mencionados en este documento para el desarrollo del proyecto.

https://github.com/KevinSandoval02/Speed_Detection