[1]

Tiny ML para la detección de vegetales.

Santiago Rivera Montoya, Leider Felipe Caicedo, Oswald Daniel Gutierrez

I. Introducción

El presente proyecto se enmarca en la implementación de soluciones basadas en Tiny Machine Learning (TinyML), cuyo propósito es llevar la inteligencia artificial al borde ("edge computing"), habilitando la ejecución de modelos de aprendizaje automático en dispositivos con recursos limitados, como microcontroladores. Esta propuesta responde a la creciente demanda de procesamiento local en tiempo real, una necesidad clave en aplicaciones como dispositivos portátiles, hogares inteligentes y sistemas de monitoreo industrial, donde la latencia y el consumo energético representan restricciones críticas.

La motivación principal del proyecto radica en las ventajas que ofrece TinyML para optimizar los tiempos de respuesta y la eficiencia operativa al prescindir de infraestructura remota, facilitando así la adopción de tecnologías de inteligencia artificial en escenarios de hardware asequible y ampliamente accesible. Además, el desafío técnico de implementar modelos ligeros y precisos bajo estrictas limitaciones de memoria y procesamiento fomenta la innovación en áreas como compresión de redes neuronales, cuantificación de pesos y poda de modelos.

La metodología del proyecto abarca desde el entrenamiento y optimización de un modelo de clasificación de imágenes mediante Edge Impulse, hasta su despliegue en un microcontrolador ESP32-Wrover-CAM. Este enfoque incluye el uso de herramientas específicas y pruebas rigurosas para evaluar la funcionalidad del sistema, como la clasificación en tiempo real de vegetales, el monitoreo del consumo de recursos, y la validación de su robustez bajo diferentes condiciones operativas.

Como resultado, se espera demostrar la efectividad de TinyML para abordar problemas del mundo real, generando un impacto positivo en el desarrollo de dispositivos inteligentes, contribuyendo así a la expansión de la inteligencia artificial hacia contextos de recursos limitados y necesidades específicas.

II. Marco teórico

[1]

Aprendizaje Automático en Dispositivos con Recursos Limitados

TinyML (Tiny Machine Learning) es una subárea del aprendizaje automático que se centra en el desarrollo e implementación de modelos en dispositivos con recursos extremadamente limitados, como microcontroladores y sensores. A diferencia de los sistemas convencionales de aprendizaje automático, estos dispositivos poseen restricciones significativas en términos de capacidad de procesamiento, memoria y energía. Por lo tanto, es necesario optimizar los modelos para que funcionen eficientemente en estos entornos restringidos.

Técnicas de Optimización de Modelos

Compresión de Redes Neuronales: La compresión de redes neuronales es una técnica utilizada para reducir el tamaño de los modelos sin sacrificar su precisión. Esto se logra mediante la eliminación de redundancias en la representación de los parámetros de la red.

Cuantificación de Pesos: La cuantificación de pesos implica reducir la precisión de los pesos del modelo, típicamente de 32 bits flotantes a 8 bits enteros. Este enfoque disminuye tanto el uso de memoria como el consumo energético, haciendo que los modelos sean más adecuados para dispositivos con recursos limitados.

Poda (Pruning): La poda es una técnica que elimina los parámetros menos importantes de la red neuronal, reduciendo así su tamaño y complejidad. Esto permite que el modelo sea más eficiente en términos de memoria y procesamiento sin perder significativamente su precisión.

Arquitecturas Ligera
Para facilitar la implementación de modelos en dispositivos con recursos limitados, se utilizan arquitecturas de redes neuronales ligeras. Algunas de las más comunes incluyen:

MobileNet: Diseñada para ser eficiente en términos de recursos computacionales, MobileNet es una red neuronal convolucional optimizada para dispositivos móviles y embebidos.

Redes Neuronales Convolucionales Reducidas: Estas redes están diseñadas específicamente para operar con menos recursos, proporcionando una alternativa viable a las arquitecturas más pesadas utilizadas en aplicaciones de alto rendimiento.

Sistemas Operativos en Tiempo Real (RTOS)
Para que TinyML funcione eficientemente en microcontroladores y otros dispositivos de baja potencia, es crucial un manejo eficiente de los recursos del sistema, como la memoria, los ciclos de CPU y el consumo energético. Los Sistemas Operativos en Tiempo Real (RTOS) proporcionan un entorno optimizado para la ejecución de procesos en tiempo real bajo restricciones estrictas.

[1]

Gestión de Memoria: Es esencial para garantizar que los procesos tengan suficiente memoria disponible sin causar fragmentación significativa.

Planificación de Procesos: En un RTOS, la planificación de procesos se realiza de manera que los procesos críticos reciban la prioridad necesaria para cumplir con los requisitos de tiempo real.

Manejo de Interrupciones: La capacidad de manejar interrupciones de manera eficiente es fundamental para aplicaciones de tiempo real, asegurando que los eventos importantes se procesen sin demoras significativas.

Conexión con la Inteligencia en el Borde (Edge Computing)
La necesidad de soluciones TinyML surge de la creciente demanda por inteligencia en el borde (edge computing), donde los datos se procesan en tiempo real sin depender de la infraestructura remota. Esto es crucial en aplicaciones como dispositivos portátiles, hogares inteligentes y monitoreo industrial, donde se requiere una respuesta inmediata, reducción en la latencia y menor consumo de ancho de banda.

TinyML permite la proliferación de la inteligencia artificial en nuevos escenarios al ejecutar tareas de IA de forma local en dispositivos con recursos limitados. Esto no solo mejora la eficiencia y la velocidad de respuesta, sino que también facilita el desarrollo de soluciones innovadoras en diversas industrias.

III. Metodología

En este proyecto se desarrolló un sistema de clasificación de vegetales utilizando TinyML con la ESP32-CAM, siguiendo un proceso estructurado que asegura tanto la precisión como la eficiencia del modelo. A continuación se detallan las herramientas y las actividades principales:

Herramientas:

ESP32-Wrover-CAM: Una pequeña cámara utilizada para capturar imágenes de vegetales y realizar inferencias en tiempo real.

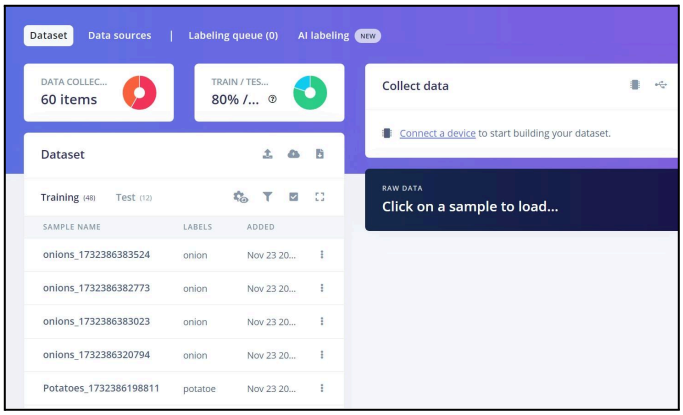
Cable USB: Herramienta para conectar y enviar datos al ESP32-Wrover-CAM desde una computadora.

Breadboard: Generalmente es utilizada para facilitar las conexiones entre los componentes, en nuestro caso solo se usó como base pero puede ser utilizada para futuras mejoras.

Vegetales para clasificar: Objetos físicos utilizados para entrenar y probar el modelo de clasificación.

1. Recopilación de datos
Se utilizaron las capacidades de captura de la ESP32-Wrover-CAM para obtener imágenes de vegetales, las cuales fueron procesadas y cargadas en la plataforma Edge Impulse. Estas imágenes sirvieron como conjunto de datos inicial para entrenar el modelo de clasificación.

2. Entrenamiento del modelo
En Edge Impulse se configuró un pipeline de aprendizaje automático con un enfoque en clasificación de imágenes. Las imágenes se etiquetaron para diferenciar entre las categorías definidas (vegetales específicos). Se emplearon redes neuronales (NN) como base del modelo, asegurando un balance entre precisión y eficiencia.



3. Optimización del modelo
Dado que la ESP32-Wrover-CAM es un dispositivo con recursos limitados, se realizaron optimizaciones utilizando TensorFlow Lite. Estas incluyeron técnicas de compresión y cuantificación, como la reducción de precisión de los pesos del modelo (de 32 bits a 8 bits), para minimizar el uso de memoria y CPU sin comprometer significativamente la precisión.

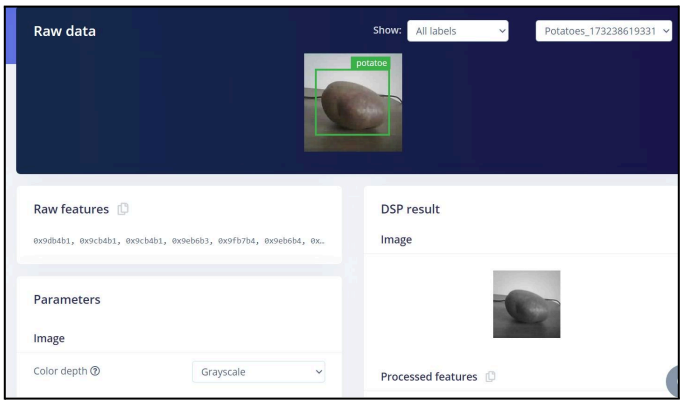
4. Implementación en hardware
El modelo optimizado fue integrado en la ESP32-Wrover-CAM mediante la programación con Arduino IDE. Se conectó la cámara al pc mediante el uso de cable USB.

5. Validación y ajustes
El sistema se evaluó en tiempo real capturando imágenes y clasificando vegetales bajo diversas condiciones, como variaciones de iluminación y ángulos de captura. También se monitorean métricas clave, como la latencia de predicción, el consumo de memoria y la precisión general del modelo. Los parámetros se ajustaron iterativamente para optimizar el desempeño del sistema.

IV. Implementación

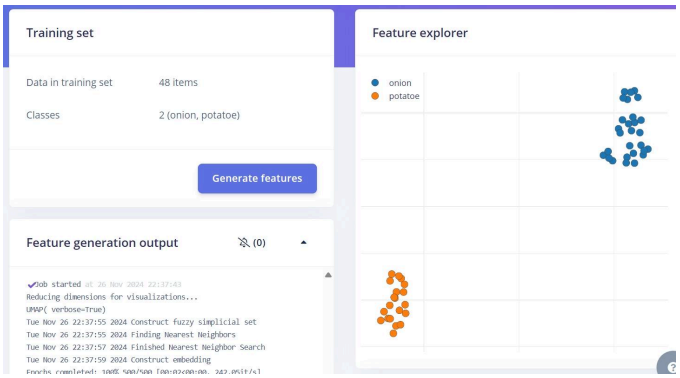
Se utilizó el firmware predeterminado de la ESP32-Wrover-CAM para capturar imágenes. Las mismas fueron procesadas y subidas a Edge Impulse para construir el

conjunto de datos inicial.



Entrenamiento y optimización del modelo:

En Edge Impulse, se configuró un pipeline para procesamiento de imágenes, seleccionando características relevantes como bordes y patrones.



El modelo entrenado fue exportado como un archivo Tensor Flow Lite, adaptado para microcontroladores.

Integración del modelo en la ESP32-Wrover-CAM:

Se utilizó el Arduino IDE para programar la ESP32-Wrover-CAM. La librería Tensor Flow Lite for Microcontrollers permitió cargar el modelo optimizado.
Se integraron funciones para capturar imágenes, ejecutar inferencias y mostrar resultados en el display OLED.
Validación del sistema:

Las pruebas se realizaron en tiempo real, evaluando la capacidad del modelo para clasificar vegetales con precisión en diversas condiciones de iluminación y ángulos.
Se midió la latencia desde la captura de una imagen hasta la visualización del resultado. Los valores promedio fueron inferiores a 1 segundo, cumpliendo con las restricciones del sistema.

V. Protocolo de experimentación

Estrategias para un Experimento Seguro y Eficiente.

1. Diseño Experimental Controlado:

- **Objetivo del proyecto:**

Identificar con precisión vegetales utilizando un sistema basado en soluciones TinyML.

- **Ambiente controlado:**

Se realizaron experimentos bajo condiciones aceptables (iluminación y posición de los objetos) para reducir las variables externas que pudieran afectar la precisión del modelo.

- **Uso de recursos limitados:**

Debido a que el proyecto se basa en la implementación de modelos de aprendizaje automático en dispositivos de recursos limitados, como la ESP32-Wrover-CAM, el proceso se diseñó centrado en la optimización para evitar la sobrecarga de los recursos del microcontrolador.

2. Optimización de modelos:

Se llevaron a cabo diversas técnicas de optimización de modelos como la cuantificación de los pesos del modelo y la poda (pruning) de la red neuronal para que fuera más fácil de implementar.

Además, se priorizó la reducción de la latencia y el consumo energético, ya que son aspectos críticos en dispositivos de bajo consumo como la ESP32-Wrover-CAM.

3. Validación del Sistema:

La validación se realizó en condiciones reales, teniendo en cuenta distintos ángulos y niveles de iluminación, para evaluar la precisión y robustez del modelo.

Se ajustaron parámetros como la precisión y la latencia del modelo para garantizar que funcionara correctamente en tiempo real.

Herramientas Utilizadas

1. Recopilación de Datos:

- **ESP32-Wrover-CAM:**

La cámara integrada en el ESP32-CAM se utilizó para capturar imágenes de vegetales en condiciones variadas. Este dispositivo permite obtener datos visuales que sirven como base para el entrenamiento del modelo de clasificación.

La ESP32-Wrover-CAM es muy útil debido a su bajo costo y su capacidad de procesamiento. Además, su pequeño tamaño y bajo consumo energético la hacen perfecta para proyectos de TinyML.

- **Arduino IDE:**

Se utilizó este entorno de desarrollo para la programación de la ESP32-Wrover-CAM y la integración del modelo optimizado en el hardware.

2. Entrenamiento del Modelo:

- **Edge Impulse:**

Plataforma utilizada para el desarrollo del pipeline de aprendizaje automático, donde se cargaron las imágenes etiquetadas de los vegetales para entrenar el modelo de clasificación. Se utilizó una red neuronal (NN) como base.

Edge Impulse es una plataforma específicamente diseñada para dispositivos embebidos y microcontroladores, lo que la hace adecuada para este tipo de experimentos.

- **TensorFlow Lite:**

Esta herramienta permite optimizar el modelo para que se ejecute en microcontroladores como la ESP32-Wrover-CAM. Las técnicas de compresión y cuantificación de pesos fueron esenciales para la reducción del tamaño del modelo. Esta herramienta optimiza modelos para el hardware limitado sin afectar la precisión.

3. Procesamiento y Validación:

Los datos se observaron en un ordenador en tiempo real mientras se realizaban las pruebas. Esta observación permitió evaluar la precisión de la clasificación y verificar cómo el sistema manejaba las variaciones en las condiciones de prueba, como los cambios en la iluminación y los ángulos de captura.

Recopilación, Procesamiento y Validación de Datos

1. **Recopilación de Datos:** Se obtuvieron imágenes de vegetales utilizando la ESP32-Wrover-CAM, asegurándose de que las imágenes se obtuvieran en diferentes condiciones (distintos fondos, iluminación y ángulos). Las imágenes fueron etiquetadas en Edge Impulse para entrenar el modelo de clasificación.

2. Procesamiento de Datos: El modelo de clasificación se entrenó en Edge Impulse con las imágenes previamente etiquetadas. Posteriormente, se optimizó utilizando técnicas como la cuantificación y poda de redes neuronales para mejorar la eficiencia del modelo.

3. Validación de Datos: La validación se llevó a cabo con pruebas en tiempo real, evaluando la capacidad de la ESP32-Wrover-CAM para clasificar correctamente los objetos bajo diversas condiciones operativas. Se midieron parámetros como la latencia de predicción y el consumo de memoria, y se realizaron ajustes al modelo para optimizar su rendimiento.

Justificación de la Selección de Herramientas

1. **Edge Impulse:** Esta plataforma es ideal para proyectos de TinyML debido a su enfoque en dispositivos embebidos y su integración con TensorFlow Lite. Permite una fácil gestión del flujo de trabajo de aprendizaje automático, desde la recopilación de datos hasta la optimización del modelo para hardware limitado.

2. ESP32-Wrover-CAM: Se seleccionó este dispositivo por su capacidad para ejecutar modelos de TinyML, su bajo costo y su eficiente consumo energético. Es compatible con la plataforma Arduino, lo que facilita la programación y la integración con otros componentes.

3. TensorFlow Lite: Esta herramienta permite optimizar los modelos de aprendizaje automático para ejecutarse en dispositivos con recursos limitados sin sacrificar significativamente la precisión. La cuantificación y poda de los modelos son técnicas cruciales para cumplir con las restricciones de memoria y procesamiento.

VI. Resultados

Detección y clasificación de objetos:

- El sistema puede identificar correctamente los objetos como potato (papa) y onion (cebolla), con un nivel de precisión variable que generalmente (según las observaciones) es superior a 0.5 este resultado siendo afectado por la posición del vegetal o el entorno, como la iluminación o la posición de la cámara.
- Los *bounding boxes* (cajas delimitadoras) se generan de forma correcta y sus dimensiones se ajustan según los objetos detectados.

Parámetros durante el procesamiento:

- El sistema realiza la predicción en tiempos consistentes (no hay variaciones en estos parámetros):
  - Procesamiento en DSP: 18 ms.
  - Clasificación: 713 ms.
  - Anomalía: 0 ms (indicando que no se detectaron anomalías en los resultados).

```
  potato (0.519531) [ x: 64, y: 40, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  potato (0.535156) [ x: 56, y: 40, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  potato (0.609375) [ x: 56, y: 40, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  onion (0.726562) [ x: 56, y: 32, width: 8, height: 8 ]
  potato (0.574219) [ x: 56, y: 40, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  onion (0.570312) [ x: 56, y: 32, width: 8, height: 8 ]
  potato (0.656250) [ x: 56, y: 40, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  potato (0.613281) [ x: 56, y: 40, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
```

Precisión en la detección de objetos:

- Los valores de precisión varían entre 0.5 y 0.99 dependiendo del objeto y las condiciones.
- Para las cebollas, se registraron niveles altos de precisión (hasta 0.99).

```
Object detection bounding boxes:
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  potato (0.527344) [ x: 48, y: 32, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  onion (0.957031) [ x: 16, y: 24, width: 24, height: 16 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  onion (0.726562) [ x: 40, y: 32, width: 16, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  onion (0.992188) [ x: 40, y: 32, width: 16, height: 24 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  onion (0.953125) [ x: 56, y: 32, width: 8, height: 16 ]
  onion (0.996094) [ x: 40, y: 40, width: 8, height: 16 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  onion (0.996094) [ x: 40, y: 32, width: 16, height: 24 ]
```

- Para las papas, la precisión es menor (rango de 0.5 a 0.66), Esto da a entender que hay una menor precisión o dificultad para identificarlas.

```
Object detection bounding boxes:
  onion (0.726562) [ x: 56, y: 32, width: 8, height: 8 ]
  potato (0.574219) [ x: 56, y: 40, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  onion (0.570312) [ x: 56, y: 32, width: 8, height: 8 ]
  potato (0.656250) [ x: 56, y: 40, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  potato (0.613281) [ x: 56, y: 40, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  potato (0.570312) [ x: 56, y: 32, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  potato (0.531250) [ x: 56, y: 32, width: 8, height: 8 ]
Predictions (DSP: 18 ms., Classification: 713 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
```

Escalado y adaptación de las cajas delimitadoras:

- Las dimensiones de las *bounding boxes* cambian conforme al tamaño y posición del objeto detectado en la imagen.
- Ejemplo: Para la cebolla (*onion*), las dimensiones van desde 8x8 hasta 16x24, mostrando que el sistema ajusta las cajas correctamente de acuerdo al tamaño del objeto.

Ausencia de anomalías:

- En todos los casos que se procesaron, no se registraron detecciones de anomalías, lo que indica que el sistema funcionó dentro de los parámetros esperados.

Resultados consistentes:

- Las detecciones se pueden replicar en condiciones similares, lo que indica estabilidad en el modelo. Sin embargo, los valores de precisión para la papa muestran cierta variabilidad.

Rendimiento del modelo:

- El modelo parece estar optimizado para detectar de forma más acertada las cebollas, ya que estas tienen niveles de precisión consistentemente altos.
- Para la papas, se podría mejorar la precisión ajustando el modelo o utilizando más datos específicos en el entrenamiento para este objeto.

VII. Conclusiones

Este proyecto nos permitió experimentar el desafío de implementar soluciones Tiny Machine Learning como lo es la Computación al Borde o IA en Edge (Edge computing or AI on Edge), llevando modelos de IA a dispositivos pequeños, lo que abre nuevas posibilidades para crear soluciones inteligentes sin depender de servidores remotos.

A través de la recolección de datos entendimos la importancia de recolectar datos de calidad y diversidad aceptables. La precisión del modelo depende en gran medida de la variedad y cantidad de las muestras, lo que aumenta la necesidad de realizar una correcta recolección y etiquetado de datos para poder obtener buenos resultados.

Ajustar el modelo es una tarea fundamental para equilibrar y mejorar la precisión y eficiencia en este tipo de proyectos. Al utilizar la plataforma Edge Impulse, pudimos optimizar el modelo, para conseguir predicciones aceptables que se adaptaran a un hardware limitado.

La Computación al Borde ofrece una importante ventaja en aplicaciones del mundo real, como el reconocimiento de objetos en tiempo real. Al procesar los datos de forma local, logramos reducir la latencia y mejorar la autonomía del dispositivo, lo que es muy efectivo en ambientes donde la conectividad a una red de internet es limitada o costosa.

VIII. Referencias

[1] DataScientest. (n.d.). TinyML: Todo lo que necesitas saber. DataScientest. https://datascientest.com/es/tinyml-todo-sobre

[2] Mjrovai. (n.d.). ESP32-TinyML: Exploring TinyML with ESP32 MCUs [Repository]. GitHub. https://github.com/Mjrovai/ESP32-TinyML

[3] Hackster.io. (n.d.). ESP32-CAM: TinyML Image Classification - Fruits vs Veggies. Hackster.io. https://www.hackster.io/mjrobot/esp32-cam-tinyml-image-classification-fruits-vs-veggies-4ab970