

MINIPROYECTO 3: Reconocimiento de Patógenos en Plantas con TinyML en Raspberry Pi

Lopez, J.D. (2191160), Mendez, M.A (2215342), Castrillon J.D.(2226042)

Universidad Autónoma de Occidente

Cali, Colombia

miguel.mendez@uao.edu.co

julian_david.lopez@uao.edu.co

juan.castrillnn@uao.edu.co

Abstract- This project presents the development and implementation of a pathogen recognition system for plants using TinyML on a Raspberry Pi with a USB camera. The system utilizes deep learning to identify pathogens in plant images, enabling real-time disease detection directly on an edge device. By employing a pre-trained convolutional neural network (CNN) with a dataset specifically focused on plant pathogens, the solution supports sustainable agricultural practices by providing an accessible tool for early disease diagnosis. The model is trained on Google Colab and optimized with TFLite for efficient processing on the Raspberry Pi. The final setup includes real-time image capture, preprocessing, inference, and alert mechanisms for pathogen detection. This approach demonstrates the feasibility of implementing low-cost AI-powered plant health monitoring systems in resource-constrained environments, advancing efforts toward sustainable agriculture and food security.

Keywords: TinyML, Plant Pathogen Recognition, Edge Computing, Raspberry Pi, USB Camera, Pre-trained Convolutional Neural Network (CNN), Sustainable Agriculture, Real-Time Disease Detection, Image Classification, Deep Learning in Agriculture.

Resumen- Este proyecto presenta el desarrollo e implementación de un sistema de reconocimiento de patógenos en plantas mediante el uso de TinyML en una Raspberry Pi con una cámara USB. El sistema utiliza aprendizaje profundo para identificar patógenos en imágenes de plantas, permitiendo la

detección de enfermedades en tiempo real directamente en un dispositivo de borde. Al emplear una red neuronal convolucional (CNN) pre entrenada con un conjunto de datos específico de patógenos vegetales, la solución apoya prácticas agrícolas sostenibles al proporcionar una herramienta accesible para el diagnóstico temprano de enfermedades. El modelo se entrena en Google Colab y se optimiza con Tflite para su procesamiento eficiente en la Raspberry Pi. La configuración final incluye captura de imágenes en tiempo real, preprocessamiento, inferencia y mecanismos de alerta en caso de detectar un patógeno. Este enfoque demuestra la viabilidad de implementar sistemas de monitoreo de la salud de las plantas, impulsados por IA y de bajo costo, en entornos con limitaciones de recursos, avanzando en los esfuerzos hacia la agricultura sostenible y la seguridad alimentaria.

Palabras claves: TinyML, Reconocimiento de Patógenos en Plantas, Computación en el Borde, Raspberry Pi, Cámara USB, Red Neuronal Convolucional pre entrenada(CNN), Agricultura Sostenible, Detección de Enfermedades en Tiempo Real, Clasificación de Imágenes, Aprendizaje Profundo en Agricultura.

I. INTRODUCCIÓN

La agricultura enfrenta desafíos crecientes debido a factores como el cambio climático, la disminución de recursos naturales y el aumento de enfermedades en las plantas. Estos problemas afectan directamente la productividad agrícola y la seguridad alimentaria

global. En este contexto, las soluciones tecnológicas como el reconocimiento de patógenos en plantas juegan un papel crucial para identificar tempranamente enfermedades, lo que permite a los agricultores tomar decisiones rápidas y eficaces para mitigar su propagación.

El presente proyecto propone el desarrollo de un sistema de reconocimiento de patógenos en plantas mediante el uso de tecnologías de TinyML en un dispositivo de computación en el borde. A través de una Raspberry Pi conectada a una cámara USB, el sistema captura imágenes de las plantas y utiliza un modelo de red neuronal convolucional pre entrenado (CNN) para clasificar posibles enfermedades en tiempo real. Esta arquitectura permite la implementación de un sistema de monitoreo de bajo costo y portátil, capaz de realizar inferencias sin depender de conexiones a internet o sistemas en la nube. Esta independencia es particularmente beneficiosa en zonas rurales o de bajos recursos donde el acceso a infraestructura digital es limitado.

Para entrenar el modelo de reconocimiento, se utiliza un conjunto de datos específico de patógenos de plantas, obtenido a partir de una base de datos en Kaggle. El modelo se entrena en Google Colab, donde se ajustan los hiperparámetros y se realizan pruebas de rendimiento para lograr un balance entre precisión y eficiencia. Posteriormente, el modelo se optimiza en Edge Impulse, lo cual permite adaptar su tamaño y velocidad para una implementación eficiente en la Raspberry Pi. Esta combinación de herramientas no solo maximiza la precisión de detección, sino que también garantiza que el sistema pueda operar dentro de las limitaciones de hardware de los dispositivos de borde.

Además, el sistema está diseñado para integrarse con mecanismos de notificación y alerta, proporcionando una respuesta visual o sonora en caso de detectar un patógeno. Esta funcionalidad aumenta su valor práctico en el campo, permitiendo a los usuarios recibir alertas instantáneas de una posible enfermedad en sus cultivos.

Este proyecto contribuye al objetivo de promover una agricultura sostenible mediante el uso de tecnología de bajo costo y accesible para pequeños y medianos agricultores. Al reducir la dependencia de grandes infraestructuras digitales y facilitar el acceso a

herramientas de diagnóstico, esta solución impulsa prácticas agrícolas más sostenibles y eficientes, avanzando hacia los Objetivos de Desarrollo Sostenible (ODS), especialmente en los aspectos relacionados con la seguridad alimentaria y la gestión de recursos naturales.

II. OBJETIVOS

Objetivo General:

Desarrollar e implementar un sistema de reconocimiento de patógenos en plantas mediante el uso de TinyML en una Raspberry Pi con una cámara USB, capaz de identificar enfermedades en tiempo real, para apoyar prácticas agrícolas sostenibles y mejorar la gestión de la salud de los cultivos.

Objetivos Específicos:

1. Entrenar un modelo de red neuronal convolucional (CNN) pre entrenado como mobilnetV2 utilizando un conjunto de datos de patógenos de plantas en Google Colab.
2. Optimizar el modelo con Tensor Flow Lite y cuantizar para su implementación en dispositivos de borde, adaptándolo a las limitaciones de procesamiento y memoria de la Raspberry Pi.
3. Integrar una cámara USB con la Raspberry Pi para la captura de imágenes en tiempo real, permitiendo la adquisición de datos en el entorno de campo.
4. Desarrollar un sistema de alertas que notifique al usuario sobre la presencia de patógenos, mediante mecanismos visuales o sonoros, facilitando la respuesta rápida ante posibles enfermedades en las plantas.
5. Validar el desempeño del sistema en condiciones reales de campo, evaluando su precisión, velocidad de inferencia y eficiencia energética, para asegurar su utilidad y practicidad en entornos agrícolas.

III. METODOLOGÍA

El desarrollo de este proyecto se estructuró en varias etapas clave para lograr un sistema eficiente de reconocimiento de patógenos en plantas mediante TinyML en una Raspberry Pi.

Además, el proyecto se alinea con los siguientes Objetivos de Desarrollo Sostenible (ODS):

- **ODS 12:** Producción y Consumo Responsables - El sistema permite un monitoreo efectivo y a bajo costo, optimizando los recursos agrícolas y minimizando el uso de insumos innecesarios como pesticidas.
- **ODS 15:** Vida de Ecosistemas Terrestres - Al reducir la propagación de enfermedades, el proyecto apoya la preservación de cultivos y promueve prácticas sostenibles en la gestión del ecosistema agrícola.

1. Definición y Preparación de los Datos

Componentes principales:

- Conjunto de datos de patógenos de plantas (Kaggle)
- Google Colab para procesamiento de datos y entrenamiento del modelo

Se seleccionó un conjunto de datos específico de imágenes de patógenos de plantas desde la plataforma Kaggle. Las imágenes fueron preprocesadas en Google Colab, donde se realizó la normalización y el escalado. Luego, se utilizó MobileNetV2 como modelo base, aprovechando sus pesos pre entrenados en ImageNet. La capa superior del modelo fue ajustada para adaptarse a la clasificación de patógenos, y el modelo fue afinado para mejorar su precisión en entornos variados.

2. Diseño y Entrenamiento del Modelo de Red Neuronal Convolutacional (CNN)

Componentes principales:

- Google Colab para entrenamiento del modelo
- Bibliotecas de Machine Learning (TensorFlow, Keras)

Se diseñó un modelo de red neuronal convolucional (CNN) en Google Colab, optimizado para tareas de

clasificación de imágenes. El modelo fue configurado con capas convolucionales y de pooling para extraer características significativas de las imágenes y permitir una clasificación precisa de los patógenos. Durante esta fase, se ajustaron los hiperparámetros clave (tasa de aprendizaje, número de filtros, tamaño de las capas) para lograr un equilibrio entre precisión y velocidad de inferencia.

3. Optimización del Modelo para Edge Computing

Componentes principales:

- Modelo MobilnetV2
- Conversión a TensorFlow Lite

El modelo fue convertido a TensorFlow Lite reduciendo así su tamaño y mejorando su velocidad de inferencia para poder ser implementado en la raspberry pi 4.

4. Configuración del Hardware y Captura de Imágenes

Componentes principales:

- Raspberry Pi (modelo 4 recomendado)
- Cámara USB para captura de imágenes
- Librerías de control de cámara y procesamiento de imágenes en Raspberry Pi (OpenCV)

Se integró una cámara USB con la Raspberry Pi para la captura de imágenes en tiempo real. La Raspberry Pi se configuró para recibir y procesar las imágenes capturadas por la cámara, utilizando la librería OpenCV para el manejo de imágenes. Esta configuración permite que el sistema obtenga datos de las plantas en campo y los procese en el dispositivo de borde sin necesidad de conectividad externa.

5. Implementación de Inferencia en Tiempo Real y Sistema de Alertas

Componentes principales:

- Raspberry Pi y sistema de notificación (buzzer o LEDs)
- Código en Python para la ejecución de inferencias y alertas

El modelo optimizado fue implementado en la Raspberry Pi para realizar inferencias en tiempo real sobre las imágenes capturadas. Cuando se detecta un patógeno, el sistema activa un mecanismo de alerta visual o sonora, notificando al usuario de una posible enfermedad en la planta. Este sistema de alertas facilita una respuesta rápida y permite al agricultor tomar acciones inmediatas.

6. Validación y Pruebas en Campo

Componentes principales:

- Raspberry Pi con cámara USB en condiciones de campo
- Procedimiento de validación (precisión, velocidad de inferencia, consumo energético)

El sistema completo fue probado en diferentes condiciones de campo para validar su precisión y eficiencia en la detección de patógenos. Se evaluaron métricas como la velocidad de inferencia, la precisión del modelo y el consumo energético del sistema. Estas pruebas permitieron identificar mejoras necesarias y confirmar la viabilidad del sistema para aplicaciones prácticas en entornos agrícolas.



Imagen 1. Diagrama de Bloques

IV. IMPLEMENTACIÓN

Carga del dataset

```

[ ] !mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

[ ] !kaggle datasets download -d sujallimje/plant-pathogens
Dataset URL: https://www.kaggle.com/datasets/sujallimje/plant-pathogens
License(s): CC-BY-NC-SA-4.0
Downloading plant-pathogens.zip to /content
100% 23.7G/23.7G [06:17<00:00, 96.4MB/s]
100% 23.7G/23.7G [06:17<00:00, 67.3MB/s]
  
```

Primero se carga el dataset de la página Kaggle, se descarga un dataset de 5 clases de imágenes de distintos tipos de hojas del usuario Sujal Limje.

División del dataset en datos de entrenamiento y testeо

```

for file in all_files:
    for cls in classes:
        if f"/{cls}/" in file and file.lower().endswith('.jpg', '.jpeg', '.png'):
            class_files[cls].append(file)

# Asegurarse de que cada clase tenga exactamente 10,000 imágenes (o menos si no hay suficientes)
for cls in classes:
    if len(class_files[cls]) > max_images_per_class:
        class_files[cls] = random.sample(class_files[cls], max_images_per_class)

# Extraer y dividir las imágenes entre entrenamiento y prueba
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    for cls, files in class_files.items():
        print(f"Procesando clase '{cls}' con {len(files)} imágenes seleccionadas.")

        # comprobar si hay exactamente 10,000 imágenes en la clase
        if len(files) != max_images_per_class:
            print("Advertencia: La clase '{cls}' no tiene exactamente 10,000 imágenes (tiene {len(files)}).")

        # Dividir en 8,000 para train y 2,000 para test
        train_files = files[train_split:]
        test_files = files[:train_split + test_split]

        # Extraer y mover archivos a los directorios correspondientes
        print(f"Extrayendo {len(train_files)} imágenes de entrenamiento y {len(test_files)} de prueba para '{cls}'")
        for file in train_files:
            zip_ref.extract(file, extract_path)
            os.rename(os.path.join(extract_path, file), os.path.join(train_dir, cls, os.path.basename(file)))

        # Crear directorios de train y test en la raíz
        train_dir = os.path.join(extract_path, 'train')
        test_dir = os.path.join(extract_path, 'test')

        # Limpiar carpetas de train y test si ya existen para evitar conflictos
        if os.path.exists(train_dir):
            shutil.rmtree(train_dir)
        if os.path.exists(test_dir):
            shutil.rmtree(test_dir)

        # Ahora, crear carpetas vacías nuevamente
        for cls in classes:
            os.makedirs(os.path.join(train_dir, cls), exist_ok=True)
            os.makedirs(os.path.join(test_dir, cls), exist_ok=True)

        # Leer el contenido del ZIP y clasificar archivos por clase
        with zipfile.ZipFile(zip_path, 'r') as zip_ref:
            all_files = zip_ref.namelist()

            # Crear diccionario para almacenar las imágenes de cada clase
            class_files = {cls: [] for cls in classes}
  
```

```

# Extraer imágenes de entrenamiento
for file in tqdm(train_files, desc='Descomprimiendo {cls} - train'):
    zip_ref.extract(file, extract_path)
    os.rename(os.path.join(extract_path, file), os.path.join(train_dir, cls, os.path.basename(file)))

# Extraer imágenes de prueba
for file in tqdm(test_files, desc='Descomprimiendo {cls} - test'):
    zip_ref.extract(file, extract_path)
    os.rename(os.path.join(extract_path, file), os.path.join(test_dir, cls, os.path.basename(file)))

print("Organización y descompresión completadas.")

# Procesando clase 'bacteria' con 10000 imágenes seleccionadas.
Extrayendo 8000 imágenes de entrenamiento y 2000 de prueba para 'bacteria'...
Descomprimiendo bacteria - train: 100%|██████████| 8000/8000 [00:26<00:00, 299.52it/s]
Descomprimiendo bacteria - test: 100%|██████████| 2000/2000 [00:02<00:00, 771.63it/s]
Procesando clase 'fungus' con 10000 imágenes seleccionadas.
Extrayendo 8000 imágenes de entrenamiento y 2000 de prueba para 'fungus'...
Descomprimiendo fungus - train: 100%|██████████| 8000/8000 [00:32<00:00, 248.71it/s]
Descomprimiendo fungus - test: 100%|██████████| 2000/2000 [00:03<00:00, 307.06it/s]
Procesando clase 'healthy' con 10000 imágenes seleccionadas.
Extrayendo 8000 imágenes de entrenamiento y 2000 de prueba para 'healthy'...
Descomprimiendo healthy - train: 100%|██████████| 8000/8000 [00:27<00:00, 289.59it/s]
Descomprimiendo healthy - test: 100%|██████████| 2000/2000 [00:07<00:00, 258.11it/s]Organización y descompresión completadas.

```

Posteriormente se hizo la división del dataset en 3 clases con 8000 imágenes de entrenamiento y 2000 imágenes de testeo, esto debido a que el tamaño original del dataset era de aproximadamente 34000 imágenes por clase, y esto para la realización del modelo en colab, era muy pesado para trabajar, por ende se decidió trabajar con 3 clases y 10000 imágenes en cada una.

```

# Verificación de la cantidad de imágenes en cada clase
def count_images_in_dir(dir_path):
    return len([file for file in os.listdir(dir_path) if file.lower().endswith(('.jpg', '.jpeg', '.png'))])

# Validación de la estructura y conteo de imágenes
for cls in classes:
    train_class_dir = os.path.join(train_dir, cls)
    test_class_dir = os.path.join(test_dir, cls)

    if os.path.exists(train_class_dir) and os.path.exists(test_class_dir):
        print(f"Clase '{cls}': directorios de train y test encontrados.")

    # Contar imágenes en train y test
    train_count = count_images_in_dir(train_class_dir)
    test_count = count_images_in_dir(test_class_dir)

    # Validar la cantidad de imágenes
    total_count = train_count + test_count
    if total_count == 0:
        print(f" Advertencia: No se encontraron imágenes para la clase '{cls}'.")
    else:
        train_ratio = train_count / total_count
        test_ratio = test_count / total_count
        print(f" Total de imágenes: {total_count}")
        print(f" Imágenes en train: {train_count} ({train_ratio:.2%})")
        print(f" Imágenes en test: {test_count} ({test_ratio:.2%})")

    # Validar las proporciones
    if abs(train_ratio - 0.8) < 0.01 and abs(test_ratio - 0.2) < 0.01:
        print(" Las proporciones de train/test están dentro del rango esperado.")
    else:
        print(" Advertencia: Las proporciones de train/test no coinciden con el 80/20 esperado.")

    print(f"Advertencia: Faltan carpetas de train o test para la clase '{cls}'.")

Clase 'bacteria': directorios de train y test encontrados.
Total de imágenes: 10000
Imágenes en train: 8000 (80.00%)
Imágenes en test: 2000 (20.00%)
Las proporciones de train/test están dentro del rango esperado.
Clase 'fungus': directorios de train y test encontrados.
Total de imágenes: 10000
Imágenes en train: 8000 (80.00%)
Imágenes en test: 2000 (20.00%)
Las proporciones de train/test están dentro del rango esperado.
Clase 'healthy': directorios de train y test encontrados.
Total de imágenes: 10000
Imágenes en train: 8000 (80.00%)
Imágenes en test: 2000 (20.00%)
Las proporciones de train/test están dentro del rango esperado.

```

Después se realizó una verificación de que cada clase tenía los mismos datos en train y en test, que son 8000 y 2000 respectivamente.

```

train_path = os.path.join(base_path, 'train')
test_path = os.path.join(base_path, 'test')

# Tamaño de imagen y batch
image_size = (128, 128)
batch_size = 16

# Función para cargar y normalizar imágenes
def cargar_y_normalizar(imagen_path, etiqueta):
    imagen = tf.io.read_file(imagen_path) # Leer archivo de imagen
    imagen = tf.image.decode_jpeg(imagen, channels=3) # Decodificar imagen JPEG
    imagen = tf.image.resize(imagen, image_size) # Redimensionar a tamaño deseado
    imagen = tf.cast(imagen, tf.float32) / 255.0 # Normalizar de 0-255 a 0-1
    return imagen, etiqueta

# Función para preparar datasets desde un directorio
def crear_dataset(directorio):
    clases = os.listdir(directorio)
    imágenes = []
    etiquetas = []

    for etiqueta, clase in enumerate(clases):
        clase_dir = os.path.join(directorio, clase)
        if os.path.isdir(clase_dir):
            for imagen_nombre in os.listdir(clase_dir):
                imagen_path = os.path.join(clase_dir, imagen_nombre)
                imágenes.append(imagen_path)
                etiquetas.append(etiqueta)

# Crear y preprocesar los datasets de entrenamiento y prueba
train_dataset = crear_dataset(train_path)
test_dataset = crear_dataset(test_path)

# Configurar batch, cache, y prefetch para optimizar el rendimiento
train_dataset = train_dataset.cache().shuffle(5000).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test_dataset = test_dataset.cache().batch(batch_size).prefetch(tf.data.AUTOTUNE)

# Verificar el tamaño de un lote para entrenamiento y prueba
for image_batch, label_batch in train_dataset.take(1):
    print("Shape of training images:", image_batch.shape)
    print("Shape of training labels:", label_batch.shape)

for image_batch, label_batch in test_dataset.take(1):
    print("Shape of test images:", image_batch.shape)
    print("Shape of test labels:", label_batch.shape)

# `train_dataset` y `test_dataset` están listos y optimizados para entrenar en una red.

```

```

Shape of training images: (16, 128, 128, 3)
Shape of training labels: (16,)
Shape of test images: (16, 128, 128, 3)
Shape of test labels: (16,)


```

Primero, carga y normaliza las imágenes redimensionándolas a 128x128 píxeles, y normaliza los valores de los píxeles. Las imágenes y etiquetas se almacenan en objetos `tf.data.Dataset`, que se optimizan para el entrenamiento mediante operaciones de caché, shuffle, y batch para agrupar datos en lotes y mejorar la velocidad de acceso y procesamiento. Finalmente, se verifica la estructura de los datos, dejándolos listos para entrenar.

```
# Tomar algunos lotes y mostrar las etiquetas
for image_batch, label_batch in test_dataset.take(5):
    print("Etiquetas en el lote:", label_batch.numpy())
```

```
Etiquetas en el lote: [2 0 1 1 0 2 1 1 0 2 0 2 1 1 1 2]
Etiquetas en el lote: [1 1 0 2 0 0 1 2 2 2 0 1 1 1 2 2]
Etiquetas en el lote: [1 0 2 0 2 1 2 1 1 0 2 1 1 1 1 2]
Etiquetas en el lote: [1 2 2 2 1 2 2 1 2 2 1 0 1 1 2 0]
Etiquetas en el lote: [2 2 0 2 2 2 0 1 0 0 0 0 2 1 1 1]
```

Se recorren cinco lotes del conjunto de prueba y se muestran las etiquetas de cada lote, convirtiéndolas en arrays de NumPy para facilitar su lectura. Esto permite verificar las etiquetas de las imágenes en los primeros lotes del dataset de prueba.

```
# Cargar el modelo MobileNetV2 desde Tensorflow Hub
mobilenet_v2_url = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4" # Para feature extraction
IMAGE_SHAPE = (128, 128)
```

Se carga la red de movilnetV2 y se define la dimensión de las imágenes.

```
# Función que define el modelo para clasificación de tres clases basado en MobileNetV2
def hojas_model(image_shape=IMAGE_SHAPE, num_classes=3, data_augmentation=None):
    input_shape = image_shape + (3,) # Forma de entrada para imágenes RGB
    # Cargar el modelo base MobileNetV2 preentrenado en ImageNet y congelar sus pesos
    base_model = tf.keras.applications.MobileNetV2(input_shape=input_shape,
                                                    include_top=False, # Quitar la capa superior
                                                    weights='imagenet')
    base_model.trainable = True # Congela el modelo base para que sus capas no se entrenen

    # Capa de entrada
    inputs = tf.keras.Input(shape=input_shape)

    # Aumentación de datos, si se proporciona
    x = data_augmentation(inputs) if data_augmentation else inputs

    # Preprocesamiento específico de MobileNetV2
    x = preprocess_input(x)

    # Paso por el modelo base
    x = base_model(x, training=False)

    # Pooling global para reducir las dimensiones y regularización con Dropout
    x = tf.keras.layers.GlobalAveragePooling2D()(x)

    x = tf1.Dropout(0.5)(x) # Aumentar Dropout
    x = tf.keras.layers.Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))(x)
    x = tf1.Dropout(0.3)(x) # Añadir un segundo Dropout
    prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax', kernel_regularizer=tf.keras.regularizers.l2(0.01))

    # Definir la salida
    outputs = prediction_layer(x)

    # Crear el modelo
    model = tf.keras.Model(inputs, outputs)

    return model, base_model
```

Se crea un modelo de clasificación de tres clases basado en MobileNetV2. Después se carga MobileNetV2 con pesos preentrenados en ImageNet y ajusta su arquitectura añadiendo capas de regularización (Dropout) y una capa densa final con activación softmax para la clasificación y se optimiza el modelo para mejorar su generalización y reducir el

sobreajuste.

```
# Crear y compilar el modelo
model = hojas_model(image_shape=IMAGE_SHAPE, num_classes=3, data_augmentation=None)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(), # Cambiado para clasificación de 3 clases
              metrics=['accuracy'])

# Entrenamiento del modelo
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-6)
history = model.fit(train_dataset, validation_data=test_dataset, epochs=15, callbacks=[reduce_lr])
```

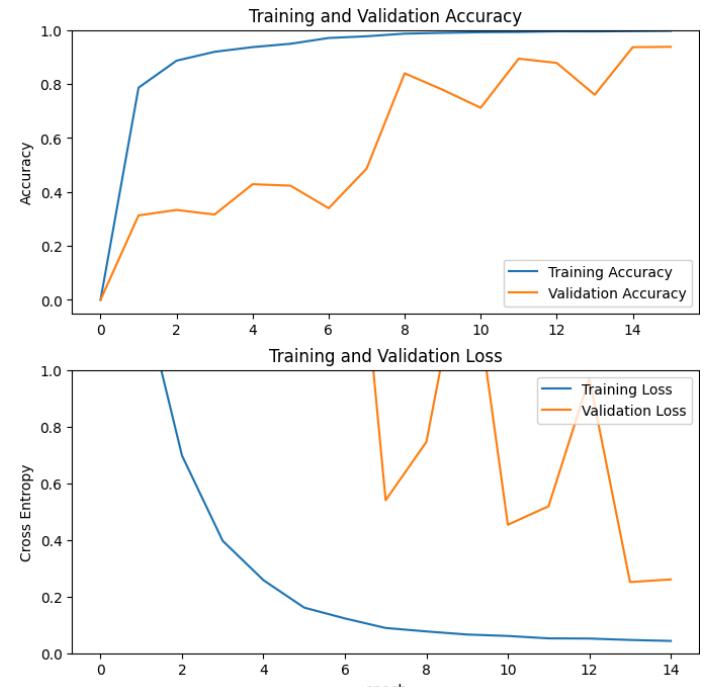
Después se crea y se compila el modelo.

```
acc = [0.] + history.history['accuracy']
val_acc = [0.] + history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

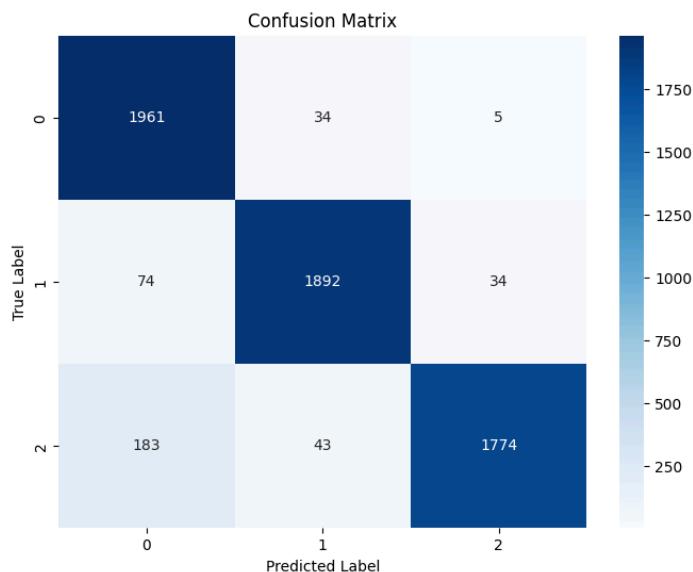
plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Después se grafica la pérdida y accuracy de los datos de entrenamiento y testeo con respecto al número de épocas establecido, que fue de 15.

```
model2.evaluate(test_dataset)
375/375 [0.2604384124279022, 0.937833309173584]
```

Se evalúa el modelo y se tiene un accuracy de aproximadamente 94%, lo cual es un valor aceptable para el análisis de imágenes.



Por último se saca su matriz de confusión y se verifican sus predicciones, como se puede ver el modelo tiene una buena respuesta y puede ser utilizado para la implementación.

Conversion a TensorFlow Lite

```
model2.save('miniproyecto3_model.h5')
WARNING:absl:You are saving your model as an HDF5 f
```

Primero se guarda el modelo como .h5

```
converter = tf.lite.TFLiteConverter.from_keras_model(model2)
model2_tflite = converter.convert()
open('miniproyecto3_model.tflite', 'wb').write(model2_tflite)
```

Después se convierte el modelo a tflite.

```
converter2 = tf.lite.TFLiteConverter.from_keras_model(model2)
converter2.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quant_model = converter2.convert()
open("./content/miniproyecto3_model.tflite", "wb").write(tflite_quant_model)
```

Después para mejorar la aplicación del modelo en la raspberry pi 4 se hace una cuantización para bajar su peso y poder usarlo en la tarjeta.

```
from google.colab import files
files.download('/content/miniproyecto3_model.tflite')
```

Por último se guarda el modelo en google drive como miniproyecto3_model_tflite, para después utilizar este archivo en el despliegue en la raspberry.

Despliegue en la raspberry Pi 4

Primero se instalan las siguientes dependencias:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install libhdf5-dev libc-ares-dev
libeigen3-dev -S
sudo apt-get install -y build-essential cmake
gfortran libjpeg-dev libtiff-dev libavcodec-dev
libavformat-dev libswscale-dev libv4l-dev
libxvidcore-dev libx264-dev libgtk-3-dev
libatlas-base-dev libblas-dev liblapack-dev
sudo apt-get install python3-venv
python3 -m venv tf-env
source tf-env/bin/activate
pip install --upgrade pip
pip install tensorflow
pip install opencv-python
pip install matplotlib
pip install RPi.GPIO
pip install qpiozero
```

Estas dependencias permiten el uso de opencv para el reconocimiento de imágenes, también permite el uso de tensor flow y otras librerías que serán necesarias para el despliegue del modelo.

```
import numpy as np
import cv2
import tensorflow as tf
import matplotlib.pyplot as plt
import RPi.GPIO as GPIO
from gpiozero import AngularServo
import time

# Configuración de GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

# Pines para actuadores
BUZZER_PIN = 17
RELAY_PIN_7 = 19
RELAY_PIN_8 = 26
SERVO_PIN = 18

# Configurar pines como salida
GPIO.setup(BUZZER_PIN, GPIO.OUT)
GPIO.setup(RELAY_PIN_7, GPIO.OUT)
GPIO.setup(RELAY_PIN_8, GPIO.OUT)

GPIO.output(RELAY_PIN_7, GPIO.HIGH)
GPIO.output(RELAY_PIN_8, GPIO.HIGH)

# configuración del servomotor
servo = AngularServo(SERVO_PIN, min_pulse_width=0.0006, max_pulse_width=0.0023)

# cargar el modelo de TensorFlow Lite
interpreter = tf.lite.Interpreter(model_path='/home/juandipro/Desktop/Miniproyecto3/miniproyecto3_model.tflite')
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Inicializar cámara
cam = cv2.VideoCapture(0)
cv2.namedWindow("test")

# Clases del modelo
class_names = ['bacteria', 'fungus', 'healthy']

def activate_buzzer(duration=5):
    """Función para activar el buzzer por un tiempo prolongado."""
    for _ in range(duration * 2):
        GPIO.output(BUZZER_PIN, GPIO.HIGH)
        time.sleep(0.5)
```

```
def activate_servo(duration=10):
    """Función para mover el servomotor por un tiempo prolongado."""
    end_time = time.time() + duration
    while time.time() < end_time:
        servo.angle = 90
        time.sleep(1)
        servo.angle = 0
        time.sleep(1)
        servo.angle = -90
        time.sleep(1)

def close_camera():
    """Cierra la cámara y libera los recursos."""
    cam.release()
    cv2.destroyAllWindows()

while True:
    # Captura de imagen en tiempo real
    ret, frame = cam.read()
    if not ret:
        print("Error al capturar el fotograma")
        break
    cv2.imshow("test", frame)

    # Espera a la tecla para capturar o cerrar
    k = cv2.waitKey(1)
    if k % 256 == 27: # Tecla ESC para salir y cerrar cámara
        print("Escape presionado, cerrando...")
        close_camera()
        GPIO.cleanup()
        break
    elif k % 256 == 32: # Tecla SPACE para capturar imagen y clasificar
        img_name = "opencv_frame_0.png"
        cv2.imwrite(img_name, frame)
        print(f"Imagen {img_name} guardada!")

    # Preprocesar imagen
    img = cv2.imread(img_name)
    img_resized = cv2.resize(img, (128, 128))
    img_resized = np.reshape(img_resized, [1, 128, 128, 3]) / 255.0
    test_imgs_numpy = np.array(img_resized, dtype=np.float32)

    # Realizar la predicción
    interpreter.allocate_tensors()
    interpreter.set_tensor(input_details[0]['index'], test_imgs_numpy)
    interpreter.invoke()
    predictions = interpreter.get_tensor(output_details[0]['index'])

    # Obtener clase predicha
    prediction_class = np.argmax(predictions, axis=1)[0]
    predicción = class_names[prediction_class]
    print(f"Predicción: {predicción}")

    # Activar actuadores según predicción
    if predicción == "bacteria":
        GPIO.output(RELAY_PIN_7, GPIO.LOW) # Activa el relé 7 de inmediato
        GPIO.output(RELAY_PIN_8, GPIO.HIGH) # Apaga el relé 8 si estaba encendido
        # Ejecuta el buzzer sin apagar el relé 7
        activate_buzzer(duration=5)

    elif predicción == "fungus":
        GPIO.output(RELAY_PIN_7, GPIO.LOW) # Activa el relé 7 de inmediato
        GPIO.output(RELAY_PIN_8, GPIO.HIGH) # Apaga el relé 8 si estaba encendido
        # Ejecuta el servomotor sin apagar el relé 7
        activate_servo(duration=10)

    elif predicción == "healthy":
        GPIO.output(RELAY_PIN_7, GPIO.HIGH) # Apaga el relé 7 si estaba encendido
        GPIO.output(RELAY_PIN_8, GPIO.LOW) # Activa el relé 8 de inmediato para la clase healthy
        # Mantiene el relé 8 activado hasta la siguiente clasificación

    # Ejecuta el servomotor sin apagar el relé 7
    activate_servo(duration=10)

    elif predicción == "healthy":
        GPIO.output(RELAY_PIN_7, GPIO.HIGH) # Apaga el relé 7 si estaba encendido
        GPIO.output(RELAY_PIN_8, GPIO.LOW) # Activa el relé 8 de inmediato para la clase healthy
        # Mantiene el relé 8 activado hasta la siguiente clasificación

    # Mostrar la imagen con la predicción
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    plt.title(predicción)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

    # Cierra recursos GPIO en caso de terminar el script
    GPIO.cleanup()
```

Por último se hace este código en python, el cual consta de lo siguiente:

1. **Configuración de GPIO y pines:**
 - Se configuran los pines de la Raspberry Pi para controlar un zumbador (BUZZER_PIN), dos relés (RELAY_PIN_7 y RELAY_PIN_8), y un servomotor (SERVO_PIN).
 - Los pines se configuran como salida, y los relés se inicializan en un estado "apagado" (lógico alto).
2. **Inicialización del servomotor:**
 - Se configura un servomotor con la librería gpiodriver, y se especifican los límites del pulso para controlar su movimiento.
3. **Carga del modelo TensorFlow Lite:**
 - Se carga el modelo preentrenado de TensorFlow Lite (miniproyecto3_model.tflite)
4. **Inicialización de la cámara:**
 - Se configura la cámara de la Raspberry Pi con OpenCV para capturar imágenes en tiempo real.
5. **Clases del modelo:**
 - Se define un listado class_names que contiene los nombres de las clases posibles: "bacteria", "fungus" y "healthy".
6. **Funciones para activar actuadores:**
 - activate_buzzer(): Activa el zumbador por un tiempo determinado, encendiendo y apagando el pin correspondiente.
 - activate_servo(): Mueve el servomotor en un ciclo de 90, 0 y -90 grados repetidamente durante un tiempo especificado.
 - close_camera(): Cierra la cámara y libera los recursos de OpenCV.
7. **Bucle principal:**
 - En un bucle infinito, el sistema captura imágenes en tiempo real con OpenCV.
 - Si el usuario presiona **ESC**, el programa cierra la cámara y limpia los pines GPIO.
 - Si el usuario presiona **ESPACIO**, el programa guarda la imagen capturada, la preprocesa (redimensionándola a 128x128 píxeles y normalizándola), y luego la pasa al modelo TensorFlow Lite para hacer la predicción.
8. **Predictión y control de actuadores:**
 - Despues de realizar la predicción, el modelo devuelve una de las tres clases posibles. Dependiendo de la clase predicha, el programa:
 - "bacteria": Activa el relé 7 y el zumbador.
 - "fungus": Activa el relé 7 y mueve el servomotor.
 - "healthy": Apaga el relé 7 y activa el relé 8, manteniendo este último activado.
9. **Mostrar la imagen con la predicción:**
 - Despues de realizar la predicción, la imagen capturada se convierte a escala de grises, se muestra con el título de la predicción y se presenta usando matplotlib.

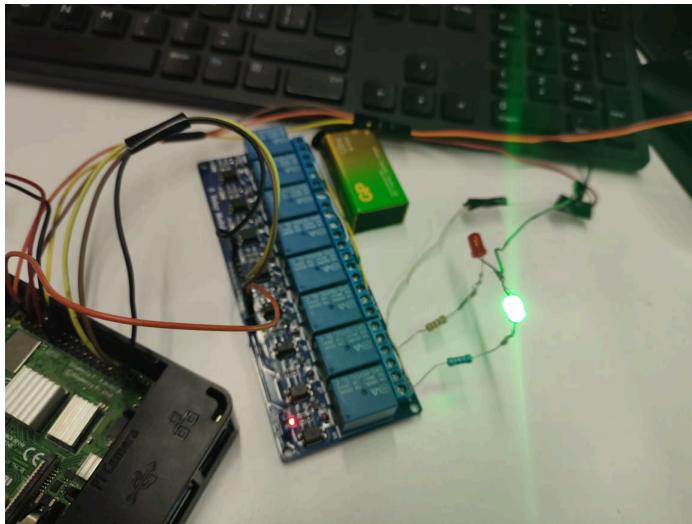
V. RESULTADOS

La idea del proyecto es poder identificar el tipo de hoja basado en 3 clases (bacteria, fungus and healthy), teniendo esto en cuenta cuando el modelo detecta que la hoja es bacteria, activa un buzzer y un led rojo, indicando esta clase, cuando detecta fungus, acciona un servomotor simulando el riego de la hoja por algún componente que puede sanar, y por ultimo cuando detecta la clase healthy, activa un led verde, indicando esta clase.

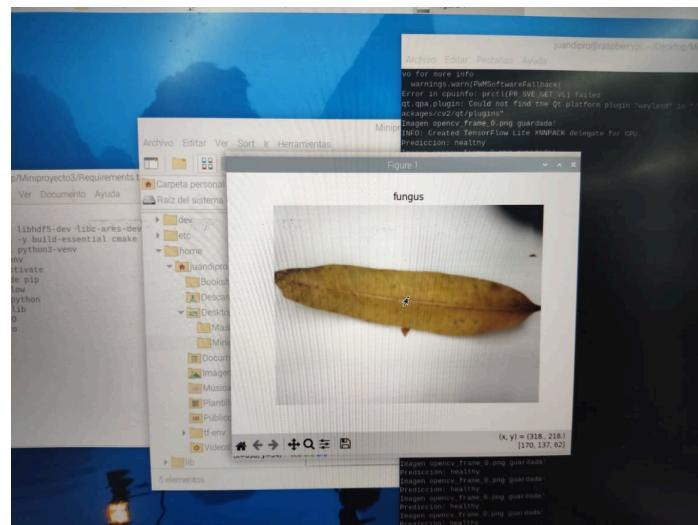
Clasificación en healthy:



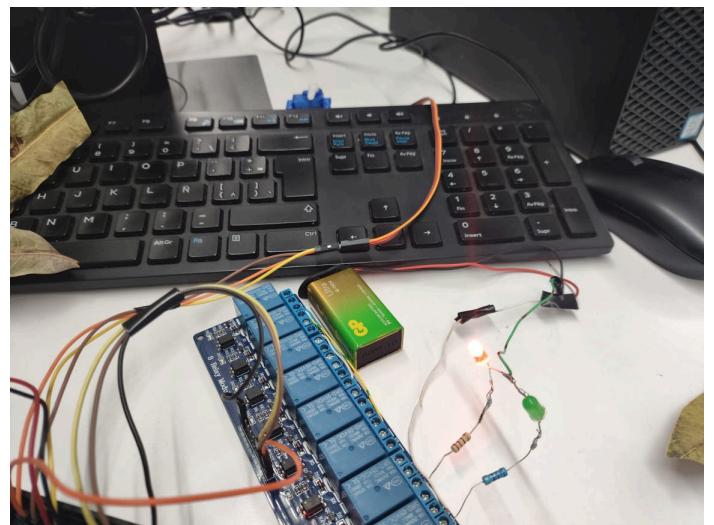
Respuesta de actuadores(Led verde) en healthy



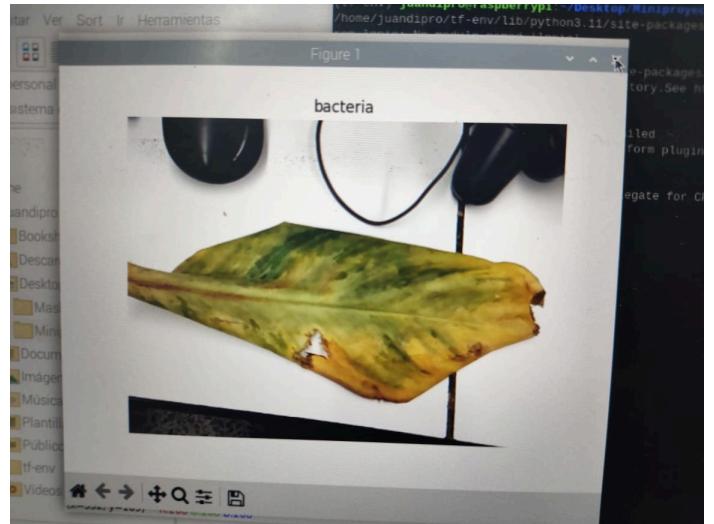
Clasificación en fungus



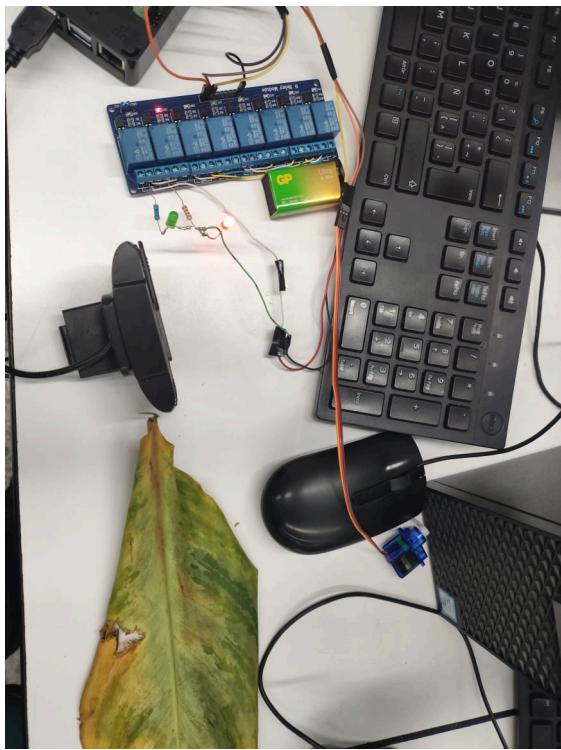
Respuesta en actuadores(led rojo y servomotor)



Clasificación en bacteria:



Respuesta de actuadores(led rojo y buzzer):



Optimizar los modelos de IA mediante herramientas como TensorFlow Lite y la cuantización resulta fundamental para lograr que el sistema funcione adecuadamente en dispositivos de bajo consumo y recursos limitados, manteniendo una precisión aceptable en la clasificación de imágenes.

Implementar un sistema de alertas accesible y visual utilizando componentes simples como un LED o buzzer, facilita la notificación de posibles enfermedades en plantas de forma inmediata, contribuyendo a una respuesta oportuna por parte de los agricultores.

Evaluar el rendimiento del sistema en condiciones reales de campo permite identificar áreas de mejora en términos de precisión y velocidad de inferencia, y asegura que el sistema cumpla con los requisitos prácticos del entorno agrícola.

Contribuir a los Objetivos de Desarrollo Sostenible (ODS) mediante esta solución tecnológica, ayuda a avanzar hacia una agricultura más sostenible, apoyando la seguridad alimentaria, reduciendo pérdidas de cultivo y optimizando el uso de recursos en la gestión de la salud de las plantas.

Demostrar el potencial de TinyML en aplicaciones agrícolas subraya la importancia de las tecnologías de inteligencia artificial en el borde para el desarrollo de herramientas accesibles y de bajo costo, útiles en sectores rurales y de bajos recursos.

Mejoras Futuras:

Incrementar el tamaño y la diversidad del conjunto de datos: Ampliar el conjunto de datos utilizado en el entrenamiento del modelo con imágenes adicionales de distintos tipos de patógenos y en diferentes condiciones de luz y ambiente. Esto permitiría mejorar la precisión y robustez del modelo, especialmente en entornos reales de campo.

VI. CONCLUSIONES

Desarrollar un sistema eficiente de reconocimiento de patógenos en plantas utilizando TinyML en una Raspberry Pi demuestra la viabilidad de aplicar inteligencia artificial en dispositivos de borde, lo cual permite la identificación en tiempo real sin depender de infraestructura en la nube.

El uso de modelos preentrenados como MobileNetV2 facilita un entrenamiento más preciso y eficiente en tareas de clasificación de imágenes, ya que aprovecha el conocimiento adquirido en grandes volúmenes de datos. Esto no solo mejora el rendimiento, sino que también simplifica el diseño del modelo, haciendo que el proceso de desarrollo sea más efectivo y accesible.

Implementar técnicas avanzadas de procesamiento de imágenes: Integrar técnicas como la detección de bordes o filtros de realce para mejorar la calidad de las imágenes capturadas y facilitar la identificación de características específicas de los patógenos, lo cual podría optimizar la precisión del modelo en escenarios más desafiantes.

Integrar sensores adicionales: Incorporar sensores de temperatura, humedad o pH, lo que permitiría correlacionar la aparición de ciertos patógenos con condiciones ambientales específicas y mejorar la precisión del sistema en contextos cambiantes.

Optimizar el modelo para un menor consumo energético: Evaluar técnicas de compresión y optimización avanzada para reducir aún más el consumo de energía en la Raspberry Pi, permitiendo que el sistema funcione por períodos prolongados en ubicaciones remotas con fuentes de energía limitadas, como paneles solares.

Desarrollar una interfaz de usuario: Crear una aplicación móvil o interfaz web para monitorear el estado de las plantas en tiempo real y visualizar las alertas y datos históricos. Esto facilita el uso del sistema y permitiría a los agricultores revisar información crítica de manera remota.

Explorar la integración de redes neuronales más avanzadas: Investigar arquitecturas de redes neuronales más sofisticadas, como MobileNet o EfficientNet, que están diseñadas para dispositivos de borde y pueden ofrecer una mejor relación entre precisión y eficiencia en comparación con modelos estándar.

Realizar pruebas extensivas en distintos entornos agrícolas: Ampliar las pruebas en diferentes ubicaciones geográficas y tipos de cultivos para validar la eficacia del sistema en diversas condiciones, adaptando el modelo a las particularidades de cada región agrícola.

Automatizar la actualización del modelo: Desarrollar un sistema que permita actualizar y mejorar el modelo de manera automática o remota, en función de nuevas imágenes y datos recolectados en campo, para mantener la precisión y adaptabilidad del sistema a nuevos desafíos.

VI. REFERENCIAS

- Limje, S. (2024). Plant Pathogens Dataset. Kaggle. Recuperado de <https://www.kaggle.com/datasets/sujallimje/plant-pathogens>.
- TensorFlow. (2024). TensorFlow Lite Documentation. TensorFlow. Recuperado de <https://www.tensorflow.org/lite>.
- Edge Impulse. (2024). Edge Impulse for TinyML. Edge Impulse. Recuperado de <https://www.edgeimpulse.com>.
- Raspberry Pi Foundation. (2024). Getting Started with Raspberry Pi and Camera Modules. Recuperado de <https://www.raspberrypi.org>.
- OpenCV. (2024). OpenCV Documentation. OpenCV. Recuperado de <https://opencv.org>.
- World Health Organization. (2023). Sustainable Development Goals (SDGs). United Nations. Recuperado de <https://www.un.org/sustainabledevelopment/sustainable-development-goals/>.
- Chollet, F. (2017). Deep Learning with Python. Manning Publications.
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4510–4520.
- Wang, S., Ji, Y., & Lu, H. (2020). Review of Image Processing Techniques for Plant Disease Detection Using Machine Learning and Deep Learning. Applied Sciences, 10(19), 6514. doi:10.3390/app10196514.
- Brownlee, J. (2019). How to Develop a Convolutional Neural Network from Scratch for CIFAR-10 Photo

Classification. Machine Learning Mastery. Recuperado
de <https://machinelearningmastery.com>.