

Aplicaciones para Sistemas de Información Sanitaria y Dispositivos Móviles

Aplicación Android de detección de hojas de plantas sanas y no saludables

Francisco Javier Tena Gómez

Contenido

Introducción	3
Objetivo y Funcionalidad de la Aplicación.....	3
Tecnologías y Herramientas Utilizadas	4
TensorFlow y TensorFlow Lite	4
Transfer Learning	4
Dataset	4
Android Studio y Java	4
Google Colab.....	5
GitHub	5
Google Drive	5
Desarrollo y Entrenamiento del Modelo	5
Selección del Dataset.....	5
Preprocesamiento de los Datos	6
Entrenamiento del Modelo con Transfer Learning	7
Evaluación del Modelo.....	9
Fine-Tuning del Modelo	10
Exportación y Conversión a TensorFlow Lite	10
Desarrollo de la Aplicación.....	11
Conclusiones.....	13
Bibliografía	14

Introducción

El desarrollo de aplicaciones móviles ha avanzado significativamente, con la incorporación de novedosas técnicas de inteligencia artificial, como el **Deep Learning**, que es una técnica que utiliza redes neuronales profundas para aprender automáticamente a través de un conjunto de datos.

En esta asignatura, hemos estudiado estas técnicas y su aplicación en dispositivos móviles. En concreto, hemos explorado el uso de TensorFlow, una biblioteca de código abierto para la creación y despliegue de modelos de aprendizaje automático, en este caso concreto se ha utilizado para detectar cuándo una hoja está sana o no, partiendo de un dataset de 4502 imágenes de hojas de plantas sanas y no saludables, divididas en 22 categorías por especie y estado de salud.

Además, en el tema hemos hablado de **Transfer Learning**, una técnica que permite aprovechar modelos pre entrenados para posteriormente resolver problemas específicos o más pequeños, como puede ser el caso de esta aplicación.

Para concluir, he utilizado todo este conocimiento adquirido y lo he puesto en práctica mediante el desarrollo de algunos codelabs propuestos y una aplicación final, desarrollada en Android, que hace uso de un modelo que ha sido entrenado con los datos mencionados anteriormente y ha utilizado Transfer Learning basándome en el modelo de Imagenet.

Objetivo y Funcionalidad de la Aplicación

El objetivo principal de esta aplicación es la clasificación de hojas de plantas en **sanas o no saludables**. Para ello he utilizado **Deep Learning** y la técnica **Transfer Learning**.

He desarrollado y creado un modelo, utilizando un dataset de 4502 imágenes cargado en la plataforma de código abierto **Mendeley Data**.

El modelo entrenado tiene la capacidad de determinar si una hoja está sana o si presenta una enfermedad.

La aplicación presenta una interfaz muy sencilla, en la cual el usuario podrá seleccionar una imagen y ver cómo la aplicación le muestra un mensaje diciéndole si la hoja seleccionada está sana o no.

Al iniciar la aplicación, el usuario podrá interactuar con un conjunto de 5 imágenes precargadas, escogidas del dataset. 2 imágenes se corresponden a hojas no saludables y 3 imágenes a hojas saludables. Además, el usuario tiene la posibilidad de acceder a su galería y poder importar más imágenes de este dataset.

Al hacer un clic sobre una imagen, el modelo realiza una clasificación y devuelve un resultado que se mostrará como un mensaje en la interfaz de usuario: **healthy** (hoja sana) o **diseased** (hoja no saludable).

Tecnologías y Herramientas Utilizadas

A continuación, describo las tecnologías y herramientas utilizadas en este proceso.

TensorFlow y TensorFlow Lite

TensorFlow es una biblioteca de código abierto, desarrollada por Google, usada en el desarrollo de modelos de Machine Learning y Deep Learning. En este proyecto se ha utilizado para entrenar un modelo de clasificación de imágenes, haciendo uso de transfer learning.

TensorFlow Lite es una versión optimizada de TensorFlow, que está diseñada para ejecutar modelos de aprendizaje automático en dispositivos móviles.

Transfer Learning

Técnica que permite la utilización de un modelo ya entrenado sobre un conjunto de datos (como Imagenet) y ajustarlo a un nuevo conjunto de datos. En este caso he utilizado Transfer Learning basándome en el modelo de Imagenet.

Dataset

He utilizado un dataset de 4502 imágenes en alta calidad. Estas imágenes han sido descargadas desde **Mendeley Data**. Este Dataset se compone de 11 especies diferentes, que a su vez 9 de esas 11 tienen dos divisiones **healthy** y **diseased**, las otras 2 especies solo tienen una de estas subcategorías, healthy o diseased.

Android Studio y Java

Para el desarrollo de la aplicación he utilizado el entorno de desarrollo Android Studio y como lenguaje de programación Java.

Google Colab

He usado Google Colab para la creación y entrenamiento del modelo Deep Learning, además de para hacer uso de Transfer Learning.

GitHub

He usado GitHub como repositorio de código.

Google Drive

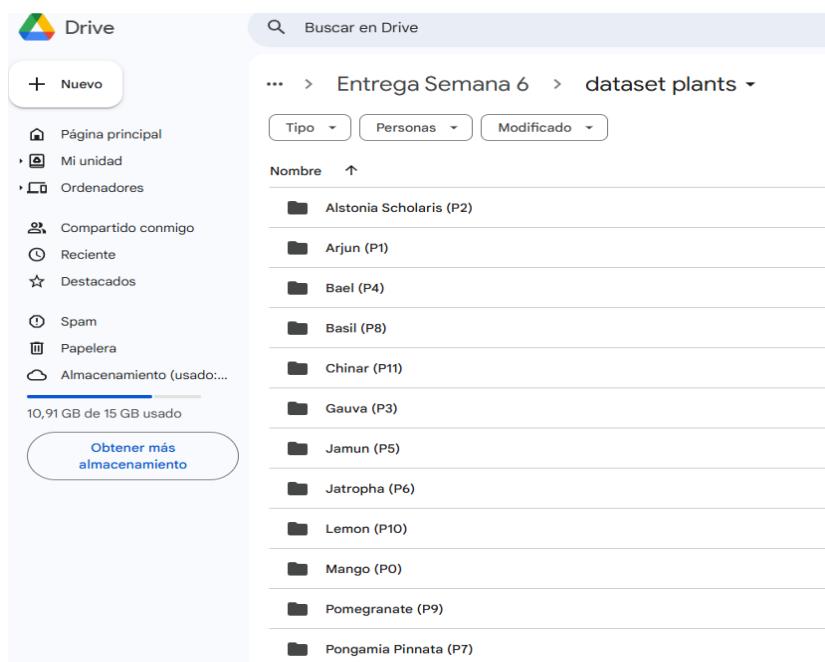
He hecho uso de Google Drive para almacenar el dataset descargado y posteriormente poder cargarlo en Google Colab.

Desarrollo y Entrenamiento del Modelo

Selección del Dataset

He seleccionado un dataset de entre los propuestos en el enunciado del ejercicio, concretamente el que hace referencia a la clasificación de hojas de plantas sanas y no saludables. Este dataset está disponible en la plataforma de código abierto de gestión de datos para la investigación, llamada Mendeley Data.

Debido al tamaño, 6GB aproximadamente, he utilizado Google Drive para realizar una primera carga del dataset y posteriormente, utilizando el cuaderno Google Colab, cargar los datos en memoria para su posterior procesamiento.

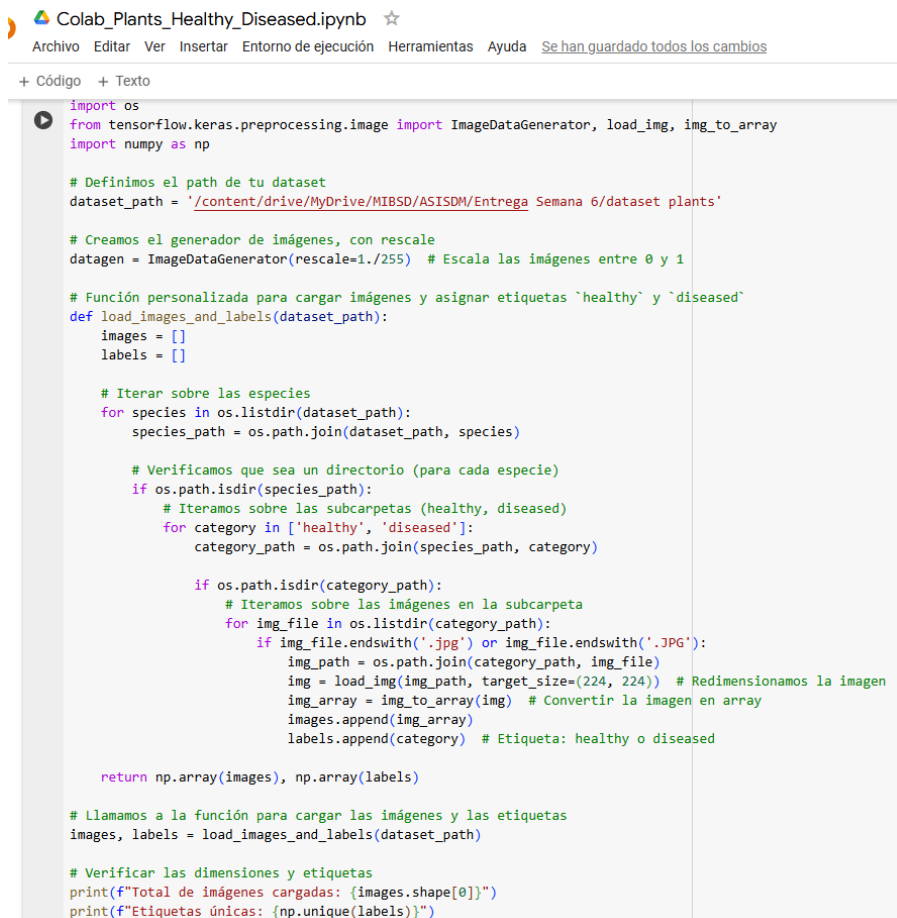


```
[ ] # Paso 1: Importar y montar Google Drive
from google.colab import drive

drive.mount('/content/drive')
```

Preprocesamiento de los Datos

Una vez cargadas las imágenes Las imágenes cargadas en Colab, ejecuté un script para procesarlas y normalizarlas, con el objetivo de que todas las imágenes tuvieran un tamaño adecuado y fácilmente procesable por el modelo. Este paso es bastante importante para que el modelo pueda realizar el aprendizaje de manera eficiente.



```
Colab_Plants_Healthy_Diseased.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios

+ Código + Texto

import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
import numpy as np

# Definimos el path de tu dataset
dataset_path = '/content/drive/MyDrive/MIBSD/ASISDM/Entrega Semana 6/dataset plants'

# Creamos el generador de imágenes, con rescale
datagen = ImageDataGenerator(rescale=1./255) # Escala las imágenes entre 0 y 1

# Función personalizada para cargar imágenes y asignar etiquetas 'healthy' y 'diseased'
def load_images_and_labels(dataset_path):
    images = []
    labels = []

    # Iterar sobre las especies
    for species in os.listdir(dataset_path):
        species_path = os.path.join(dataset_path, species)

        # Verificamos que sea un directorio (para cada especie)
        if os.path.isdir(species_path):
            # Iteramos sobre las subcarpetas (healthy, diseased)
            for category in ['healthy', 'diseased']:
                category_path = os.path.join(species_path, category)

                if os.path.isdir(category_path):
                    # Iteramos sobre las imágenes en la subcarpeta
                    for img_file in os.listdir(category_path):
                        if img_file.endswith('.jpg') or img_file.endswith('.JPG'):
                            img_path = os.path.join(category_path, img_file)
                            img = load_img(img_path, target_size=(224, 224)) # Redimensionamos la imagen
                            img_array = img_to_array(img) # Convertir la imagen en array
                            images.append(img_array)
                            labels.append(category) # Etiqueta: healthy o diseased

    return np.array(images), np.array(labels)

# Llamamos a la función para cargar las imágenes y las etiquetas
images, labels = load_images_and_labels(dataset_path)

# Verificar las dimensiones y etiquetas
print(f"Total de imágenes cargadas: {images.shape[0]}")
print(f"Etiquetas únicas: {np.unique(labels)}")
```

A continuación, he dividido las imágenes en dos conjuntos de datos. Un primer conjunto compuesto por el 80% de las imágenes del dataset, que se utilizarán para el entrenamiento del modelo. Y un segundo conjunto compuesto por el 20% de las imágenes, que lo utilizaremos para las pruebas. Esto permitirá evaluar el rendimiento del modelo.

He utilizado la función **train_test_split** de la biblioteca **scikit-learn** que permite dividir los datos en dos conjuntos, uno para entrenamiento y otro para evaluación.

El parámetro **images** contiene las imágenes procesadas anteriormente.

El parámetro **labels** contiene las etiquetas asociadas a cada imagen: **healthy** o **diseased**.

El parámetro **test_size** indica que el 20% de los datos se utilizarán para pruebas y el 80% restante para entrenamiento.

El parámetro **random_state** utiliza este valor para generar siempre los mismos datos en cada uno de los conjuntos.

```
[ ] from sklearn.model_selection import train_test_split

# Dividimos los datos en dos conjuntos, entrenamiento y testeo
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
```

Como último paso de este proceso, previo a la compilación y entrenamiento del modelo, utilicé el siguiente código para normalizar los datos y que los datos estén en un rango más pequeño, más exactamente entre **[0-1]**. Con esto mejoramos el rendimiento en el entrenamiento, ya que las redes neuronales generalmente convergen más rápido cuando los valores están normalizados en valores pequeños.

Entrenamiento del Modelo con Transfer Learning

En este proyecto, he utilizado **MobileNetV2**, un modelo ya entrenado en el dataset **ImageNet**, como base para la clasificación de imágenes de hojas en sanas o no saludables. **MobileNetV2** fue elegido por su eficiencia y bajo costo computacional, características que lo hacen ideal para aplicaciones en dispositivos móviles.

Para ajustar el modelo al caso específico de esta aplicación, he añadido capas personalizadas, para la clasificación final.

Se han excluido las capas superiores originales (`include_top = false`) para permitir añadir capas personalizadas.

Sobre esta base, añadí:

Un nuevo "head" de clasificación que incluye:

Una capa convolucional con 32 filtros y activación ReLU, para extraer patrones específicos adicionales de las características ya procesadas.

Una capa de Dropout (0.2), para regularizar y prevenir el sobreajuste al eliminar aleatoriamente algunas conexiones durante el entrenamiento.

Una capa de Global Average Pooling que reduce las dimensiones de las características generadas por la convolución, manteniendo la esencia de la información relevante.

Una capa densa con activación **Sigmoid**, que permite clasificar las imágenes en una salida binaria: healthy (sana) o diseased (no saludable).

Finalmente, el modelo fue compilado utilizando el optimizador Adam y la función de pérdida **binary_crossentropy**, adaptada para clasificaciones binarias, y se configuraron métricas de precisión para evaluar el desempeño durante el entrenamiento.

```
import tensorflow as tf

# Definimos el tamaño de la imagen
IMAGE_SIZE = 224 # Tamaño estándar
IMG_SHAPE = (IMAGE_SIZE, IMAGE_SIZE, 3)

# 1. Creamos el modelo base desde MobileNet V2
base_model = tf.keras.applications.MobileNetV2(
    input_shape=IMG_SHAPE,
    include_top=False, # Excluimos las capas de clasificación originales
    weights='imagenet' # Usamos los pesos preentrenados en ImageNet
)

# 2. Congelamos los pesos del modelo base
base_model.trainable = False

# 3. Creamos un nuevo "head" de clasificación
model = tf.keras.Sequential([
    base_model, # Modelo base como extractor de características
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'), # Convolución adicional
    tf.keras.layers.Dropout(0.2), # Regularización para evitar sobreajuste
    tf.keras.layers.GlobalAveragePooling2D(), # Pooling global
    tf.keras.layers.Dense(1, activation='sigmoid') # Clasificación binaria en una sola clase (healthy o diseased)
])

# 4. Compilamos el modelo
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss='binary_crossentropy', # Usamos 'sparse' si las etiquetas no están codificadas como one-hot
    metrics=['accuracy']
)

# Mostramos resumen del modelo
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
mobilenetv2_1_00_224 (Functional)	(None, 7, 7, 1280)	2,257,984
conv2d (Conv2D)	(None, 5, 5, 32)	368,672
dropout (Dropout)	(None, 5, 5, 32)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 32)	0
dense (Dense)	(None, 1)	33

Total params: 2,626,689 (10.02 MB)
Trainable params: 368,705 (1.41 MB)
Non-trainable params: 2,257,984 (8.61 MB)

Además, antes de entrenar el modelo, realicé un mapeo de los valores originales de las etiquetas **“healthy”** y **“diseased”** a números 0 y 1. Esta conversión es bastante importante ya que el modelo utiliza la función de pérdida **binary_crossentropy**, que asume que las etiquetas son binarias y numéricas.

```
# Mapeo manual de las etiquetas de texto a números
label_map = {'healthy': 0, 'diseased': 1}
y_train = np.array([label_map[label] for label in y_train])
y_test = np.array([label_map[label] for label in y_test])
```


Finalmente, para el entrenamiento del modelo utilicé el siguiente script:

```
# Entrenamiento del modelo
history = model.fit(
    X_train, y_train, # Datos de entrada (imágenes) y etiquetas de salida
    epochs=10,        # Número de épocas. Número de veces que el modelo ve los datos
    batch_size=32,     # Tamaño del batch. Número de lotes que el modelo procesa antes de actualizar sus pesos
    validation_data=(X_test, y_test) # Usamos los datos de test para validación durante el entrenamiento
)
```

En este paso, se entrena el modelo, utilizando el conjunto de datos de entrenamiento (80% del dataset), durante 10 épocas, con un tamaño de batch de 32 imágenes. Este proceso va ajustando los pesos del modelo buscando mejorar la capacidad de clasificación del modelo.

Evaluación del Modelo

El entrenamiento anterior mostró una mejora constante a lo largo de las 10 épocas. En la primera, el modelo comenzó con una precisión del 66% en entrenamiento y 82% en validación con una pérdida de 0.89 en entrenamiento y 0.49 en validación. En la última época, la precisión del modelo alcanzó el 96%, mientras que en validación llegó al 93%, con las pérdidas reduciéndose a 0.24 en entrenamiento y 0.30 en validación.



Fine-Tuning del Modelo

He intentado utilizar la técnica Fine-Tuning en este proceso, pero no ha sido posible, porque el entorno Google Colab, en el punto del entrenamiento, se quedaba sin memoria RAM.

En este caso, con ayuda siempre de documentación sobre todo este proceso, intenté configurar el modelo para que las primeras 100 capas del modelo base MobilNetV2 permanecieran congeladas y se entrenaran las últimas capas. He utilizado una tasa de aprendizaje baja ($1e-5$) para evitar que los pesos de las capas ya entrenados se actualicen demasiado. Después, recompilé el modelo e intenté el entrenamiento. Fue en este paso donde no he conseguido seguir y por tanto he seguido con mi modelo entrenado previamente.

```
[ ] # Descongelamos las últimas capas del modelo base
    base_model.trainable = True

    # Especificamos las capas que queremos afinar (por ejemplo, las últimas 4 capas del modelo base)
    fine_tune_at = 100 # Número de capas que queremos descongelar

    # Congelamos las primeras 100 capas y entrenar las últimas
    for layer in base_model.layers[:fine_tune_at]:
        layer.trainable = False

# Volvemos a compilar el modelo (necesario después de hacer cambios en las capas)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5), # Usamos una tasa de aprendizaje baja
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()

# Entrenamos el modelo con las capas descongeladas
history_finetune = model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=16,
    validation_data=(X_test, y_test)
)
```

Exportación y Conversión a TensorFlow Lite

Para el proceso de exportación y conversión a TensorFlow Lite, he desarrollado el siguiente código:

Guardar el modelo en formato SavedModel

```
import tensorflow as tf

# Ruta donde guardaste el modelo
saved_model_dir = 'save/fine_tuning'

# Cargar el modelo desde el directorio guardado
#model = tf.saved_model.load(saved_model_dir)

tf.saved_model.save(model, saved_model_dir) # Guardamos el modelo
```

Convertir a TensorFlow Lite:

```
[ ] saved_model_dir = 'save/fine_tuning' # Directorio donde se guardó el modelo
    converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
    tflite_model = converter.convert()
```

Guardar el modelo .tflite en un archivo:

```
[ ] # Guardar el modelo convertido en formato .tflite
with open('model.tflite', 'wb') as f:
    f.write(tflite_model)
```

En este proceso, primero he guardado el modelo entrenado en el formato SavedModel. Posteriormente, he convertido el modelo a TensorFlow Lite, utilizando TFLiteConverter, que optimiza el modelo para ser ejecutado en dispositivos móviles y finalmente el modelo se exportó en formato “.tflite”.

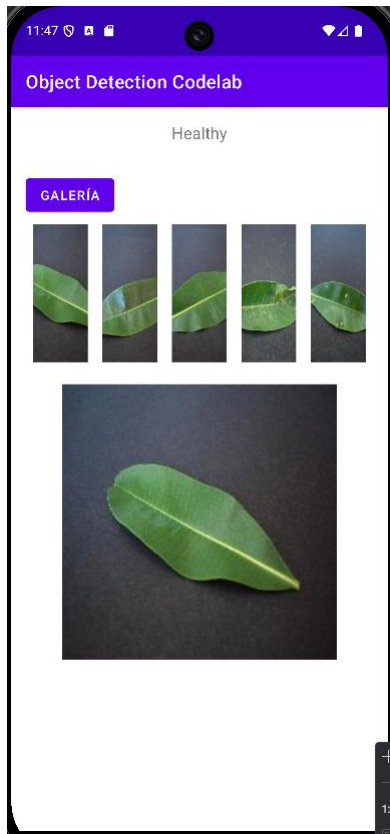
También guardé un fichero “.txt” con las etiquetas “healthy” y “diseased” por si fuera necesario utilizarlo, ya que vi que recomendaban hacer esta práctica.

Desarrollo de la Aplicación

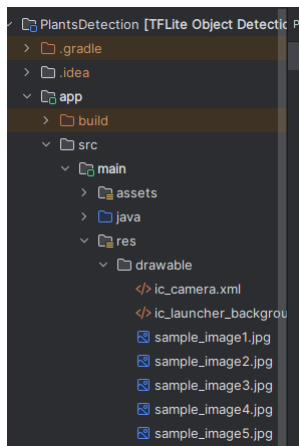
Para el desarrollo de la aplicación, inicialmente intenté tomar como base un proyecto codelab realizado en el proceso de este tema, indicado en la bibliografía en el punto [3] sobre detección de objetos. Después de importar mi modelo .tflite, tuve problemas para iniciar la aplicación y, tras investigar bastante, decidí que la mejor opción era crear un proyecto desde cero e implementar una lógica más personalizada.

A continuación, detallo los componentes de la interfaz gráfica de la aplicación Android desarrollada:

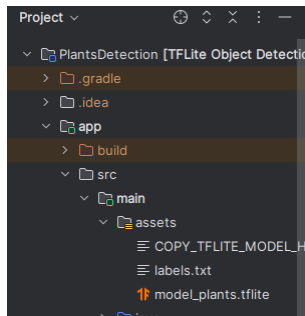
- Un espacio en el centro reservado para mostrar la imagen que será analizada por el modelo.
- Un TextView para mostrar el resultado de la predicción.
- Cinco imágenes precargadas que el usuario puede seleccionar para hacer la predicción.
- Un botón que permite al usuario seleccionar una imagen desde la galería del dispositivo, garantizando que las imágenes sean del mismo tipo que el dataset utilizado para entrenar el modelo.



Comentar, que las imágenes las he guardado en la carpeta **src/main/res/drawable**



El modelo está en la carpeta **src/main/assets**



Comentar, que todo este código está accesible en un repositorio que he creado:
https://github.com/fraaanci28/plants_detection

Conclusiones

En este proyecto de aplicación Android, se ha implementado un modelo de detección de imágenes de hojas de plantas, capaz de reconocer hojas saludables y no saludables, utilizando un modelo TensorFlow entrenado y convertido a TensorFlow Lite para su uso en dispositivos Android. El modelo ha mostrado buenos resultados en el entrenamiento, alcanzando alta precisión en las predicciones.

La aplicación Android complementa este modelo al permitir seleccionar imágenes precargadas del dataset inicial o acceder a la galería del teléfono para elegir nuevas imágenes. Este proyecto demuestra el gran potencial de combinar modelos de aprendizaje automático con aplicaciones móviles, ofreciendo soluciones prácticas en diversos ámbitos, especialmente en el sector de la salud.

Además, este proceso me ha proporcionado una visión más completa de los componentes de un proyecto como este. A través de nuevas técnicas aprendidas y la superación de retos técnicos, he adquirido una perspectiva más amplia que sin duda me ayudará a mejorar en futuros proyectos.

Bibliografía

- [1] *TensorFlow.org*. (s.f.). Obtenido de TensorFlow:
https://www.tensorflow.org/datasets/catalog/plant_leaves?hl=es
- [2] Siddharth Singh Chouhan, A. K. (06 de 06 de 2019). *Mendeley Data*. Obtenido de A Database of Leaf Images: Practice towards Plant Conservation with Plant Pathology: <https://data.mendeley.com/datasets/hb74ynkjc/1>
- [3] *Developers Google*. (s.f.). Obtenido de Compila e implementa un modelo de detección de objetos personalizado con TensorFlow Lite (Android):
<https://developers.google.com/codelabs/tflite-object-detection-android?hl=es-419#0>