MINIPROYECTO 2: Automatización del Riego de Cultivos Mediante Reconocimiento de Voz con TinyML

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 focuses on the implementation of a TinyML model to classify audio signals related to different crop types (tomato, onion, tangerine, carrot, banana and "stop") using a development kit such as the TinyML Kit from Arduino.***
The application is developed to control four servo motors based on the recognized class of audio, where each motor corresponds to a crop, and all motors stop functioning when the "stop" command is recognized.
Wireless communication is integrated using Bluetooth to allow remote control and monitoring.
The model was trained and tested using a 80-20 split on a custom dataset.
The project uses both Edge Impulse and Colab for model training, with deployment challenges encountered on Arduino IDE for inference.

Keywords: Arduino Uno, Arduino Nano 33 BLE Sense, Edge Impulse, artificial intelligence, inertial sensors, real-time monitoring, machine learning, motion detection.

Resumen- ***Este proyecto tiene como objetivo desarrollar un sistema automatizado de riego de cultivos basado en el reconocimiento de voz mediante TinyML y controlado de forma remota a través de Bluetooth. Se entrenó un modelo de aprendizaje automático utilizando la plataforma Edge Impulse, capaz de clasificar cinco comandos de voz: "cebolla", "mandarina", "zanahoria" “banano” y "apagar". Cada comando activa o desactiva un servomotor específico que controla el riego de los cultivos mencionados, mientras que el comando "apagar" detiene todos los servomotores.***

El sistema fue implementado en un TinyML Kit de Arduino para permitir la supervisión y el control remoto de los servomotores desde un dispositivo móvil o PC. El modelo fue cuantizado a TensorFlow Lite para optimizar su despliegue en el microcontrolador.

Palabras claves: Arduino Uno, Arduino Nano 33 BLE Sense, Edge Impulse, inteligencia artificial, monitoreo en tiempo real, detección de voz.

I. INTRODUCCIÓN

Este proyecto tiene como objetivo implementar un sistema inteligente de control de riego para cultivos utilizando modelos de TinyML. Mediante el reconocimiento de comandos de voz, el sistema es capaz de activar o desactivar servomotores que dirigen el riego a cultivos específicos (cebolla, mandarina, zanahoria, banano).

El sistema fue entrenado en plataformas como Edge Impulse y replicado en Google Colab, con el objetivo de lograr un reconocimiento preciso de las señales de audio.

II. OBJETIVOS

Objetivo General:
Desarrollar un sistema de riego automatizado basado en reconocimiento de voz que utilice TinyML para activar y desactivar servomotores dirigidos a diferentes cultivos (cebolla, mandarina, zanahoria y

banano) y detener el riego con el comando "apagar".

Objetivos Específicos:

1. Capturar señales de audio correspondientes a cinco clases: banano, cebolla, mandarina, zanahoria y apagar.
2. Entrenar un modelo de clasificación de audio utilizando una división de 80% para entrenamiento y 20% para prueba.
3. Implementar el modelo en un dispositivo embebido que controle cuatro servomotores.
4. Realizar la cuantización y conversión del modelo a TensorFlow Lite para su despliegue en el IDE de Arduino.
5. Resolver los problemas de inferencia y ejecución del modelo en Arduino para que el sistema funcione correctamente en tiempo real.

III. METODOLOGÍA

El sistema se compone de varios elementos clave que trabajan en conjunto para garantizar un monitoreo eficaz y una respuesta oportuna a eventos críticos.

Los componentes principales incluyen:

1. ***Arduino Nano 33 BLE Sense:*** El microcontrolador central que captura y procesa datos de movimiento a través de sus sensores integrados.

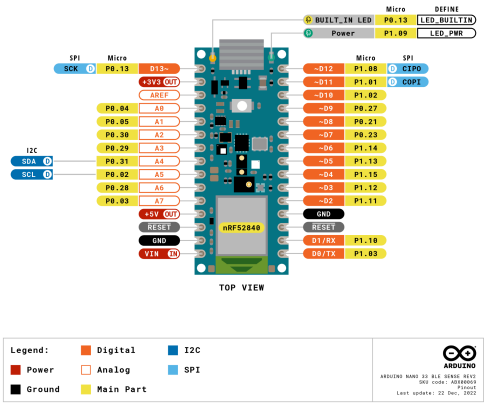


Imagen 1. Arduino nano 33 BLE sense

2. ***Micrófono integrado*:** Incorporado en el Arduino Nano 33 BLE Sense y se utiliza para registrar los comandos de voz del usuario en tiempo real.

3. ***Edge Impulse:*** La plataforma de inteligencia artificial utilizada para entrenar un modelo de aprendizaje automático. El modelo entrenado se despliega en el Arduino para la clasificación en tiempo real de los datos de voz.

4. ***Servomotores:*** Asignados a cada uno de los cultivos (tomate, cebolla, mandarina, zanahoria). El comando "apagar" detiene todos los servomotores.

Imagen 2. Servomotores

El proyecto se alinea con los siguientes Objetivos de Desarrollo Sostenible (ODS):

- **ODS 6: Agua limpia y saneamiento:** Al utilizar el sistema de riego automatizado basado en TinyML, se promueve un uso más eficiente del agua, dirigiéndose solo a los cultivos que lo necesitan, lo que contribuye a la conservación de los recursos hídricos.

- **ODS 12: Producción y consumo responsables:** Este sistema de riego controlado permite una gestión más sostenible

de los recursos, optimizando el consumo de agua y mejorando la eficiencia en las prácticas agrícolas, lo que se traduce en una producción más responsable y menos desperdicio.

El conjunto de datos fue dividido en dos partes: entrenamiento y prueba. Se recopilaron un total de 625 muestras para el entrenamiento y 158 muestras para el conjunto de prueba, distribuidas en cinco clases de comandos de voz:

Imagen 3. Datos de entrenamiento

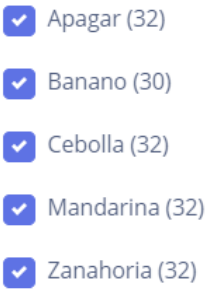


Imagen 4. Datos de validación

Cada muestra tiene una duración de entre 1.000 ms. Este rango de duración asegura que las señales de audio sean lo suficientemente largas para capturar las características acústicas de cada comando, permitiendo al modelo diferenciar de manera efectiva entre las diferentes clases.

Resultados del Entrenamiento en Edge Impulse:

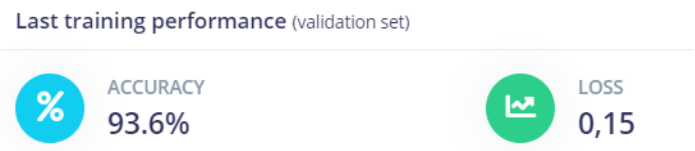
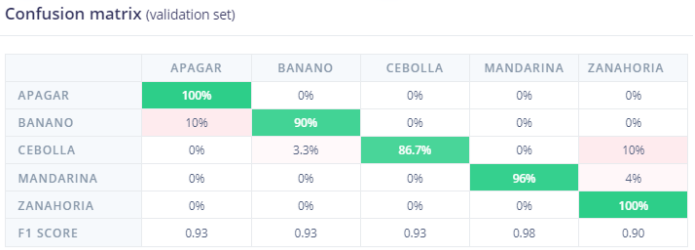


Imagen 5. Precisión y perdida

- Matriz de confusión: El modelo mostró un buen rendimiento en la mayoría de las clases, con pequeñas confusiones entre banano, cebolla y mandarina.



	APAGAR	BANANO	CEBOLLA	MANDARINA	ZANAHORIA
APAGAR	100%	0%	0%	0%	0%
BANANO	10%	90%	0%	0%	0%
CEBOLLA	0%	3.3%	86.7%	0%	10%
MANDARINA	0%	0%	0%	96%	4%
ZANAHORIA	0%	0%	0%	0%	100%
F1 SCORE	0.93	0.93	0.93	0.98	0.90

Imagen 6. Matriz de confusión

IV. IMPLEMENTACIÓN

La implementación del sistema se llevó a cabo en varias etapas, desde el entrenamiento del modelo de aprendizaje automático hasta el despliegue en el dispositivo embebido y la integración de los servomotores. A continuación, se detalla cada una de las fases de implementación:

Fase 1: Entrenamiento del Modelo

El modelo fue entrenado utilizando la plataforma Edge Impulse con un conjunto de datos que incluyó 625 muestras para el entrenamiento y 158 muestras para la prueba, distribuidas en cinco clases: "apagar", "banano", "cebolla", "mandarina" y "zanahoria". Sin embargo antes de realizar el modelo se verificó en el explorador de características que los datos entre las clases no estuvieran superpuestos, para evitar que la red no tuviera conflictos en el entrenamiento.

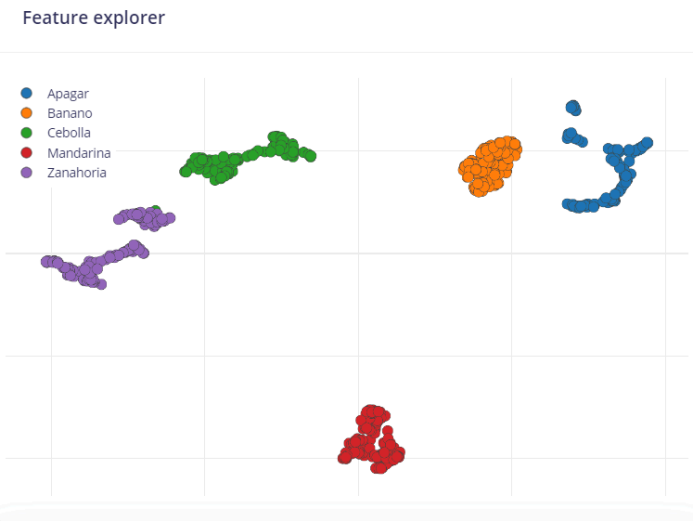


Imagen 7. Explorador de características

Ahora bien, Se utilizó una red neuronal convolucional (CNN) con la siguiente estructura:

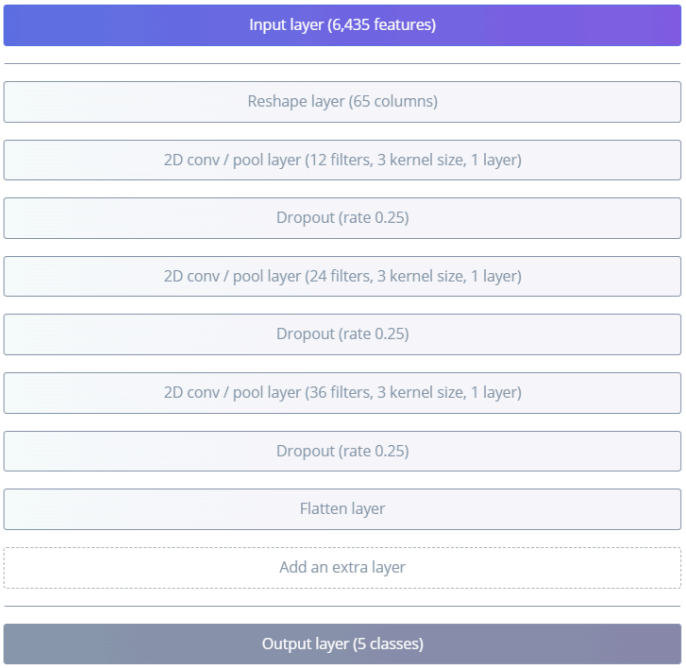


Imagen 8. Estructura de la red neuronal

De donde se resalta el uso de 3 capas convolucionales con 12, 24 y 36 filtros respectivamente en cada una de ellas, usando un kernel de 3.

Por otra parte se utilizó el data argumentation para aumentar la cantidad de ruido y con esto mismo aumentar la cantidad de los datos.

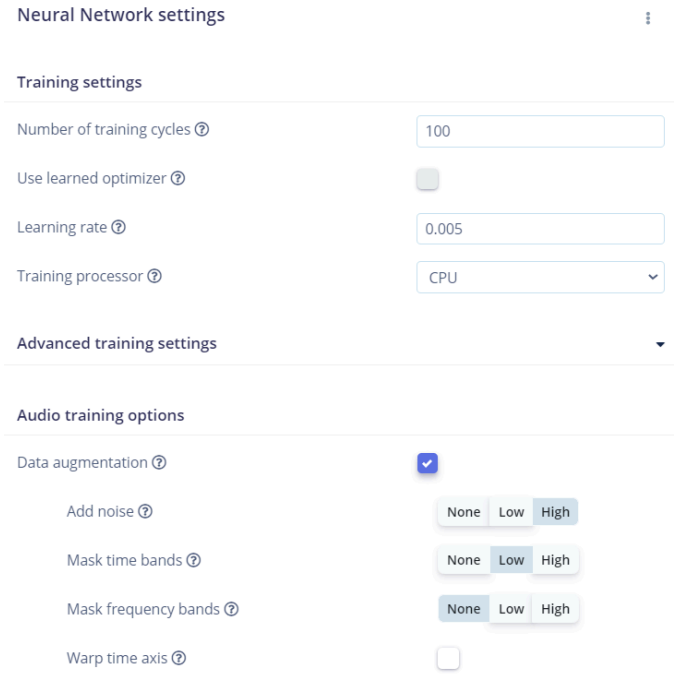


Imagen 9. Parámetros de entrenamiento

El proyecto del edge impulse se encuentra en el siguiente link: https://studio.edgeimpulse.com/public/535477/live

Fase 2: Despliegue en el TinyML Kit de Arduino

El modelo entrenado y cuantizado se desplegó en el TinyML Kit de Arduino, que cuenta con un micrófono integrado para la captura de los comandos de voz en tiempo real. Para lo cual se descargó la librería generada en el edge impulse y se agregó en el IDE de Arduino para ejecutar el modelo de inferencia y controlar los servomotores en función de los resultados de la clasificación.

Los servomotores se conectan físicamente a la placa de desarrollo y fueron asignados a cada cultivo, de manera que los comandos "banano", "cebolla", "mandarina" y "zanahoria" activaran el servomotor correspondiente para abrir el flujo de agua hacia ese cultivo. El comando "apagar" detiene todos los servomotores y, por lo tanto, interrumpe el riego en todas las áreas.

Fase 3: Comunicación entre arduinos

Se realizó la conexión mediante los puertos Tx y Rx, de un arduino uno y del arduino nano 33 BLE sense, debido a que hubieron complicaciones con la comunicación entre ellos con el bluetooth, debido a que no poseíamos el módulo bluetooth HC-05, solo HC-06, por ende no teníamos como dar la guia de que arduino iba a ser el maestro y cual el esclavo y esto nos limitó, por ende decidimos hacer la comunicación directa entre los arduinos. A continuación se muestra el código utilizado en el arduino uno para mover los servomotores.

```cpp
#include <Servo.h>

// Crear objetos Servo para cada servomotor
Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;

// Pines donde están conectados los servos
const int servoPin1 = 9;
const int servoPin2 = 10;
const int servoPin3 = 11;
const int servoPin4 = 12;

void setup() {
  // Inicializar la comunicación serial
  Serial.begin(9600);

  // Conectar los servos a sus respectivos pines
  servo1.attach(servoPin1);
  servo2.attach(servoPin2);
  servo3.attach(servoPin3);
  servo4.attach(servoPin4);

  // Colocar los servos en la posición inicial (0 grados)
  servo1.write(0);
  servo2.write(0);
  servo3.write(0);
```

```cpp
  Serial.println("Sistema listo. Esperando comandos...");
}

void loop() {
  // Verificar si hay datos disponibles en el puerto serie
  if (Serial.available() > 0) {
    String comando = Serial.readStringUntil('\n');  // Leer el comando enviado

    // Eliminar espacios en blanco si es necesario
    comando.trim();

    // Comparar el comando recibido con las palabras clave
    if (comando.equalsIgnoreCase("cebolla")) {
      moverServo(servo1, "Servo 1");
    } else if (comando.equalsIgnoreCase("banano")) {
      moverServo(servo2, "Servo 2");
    } else if (comando.equalsIgnoreCase("mandarina")) {
      moverServo(servo3, "Servo 3");
    } else if (comando.equalsIgnoreCase("zanahoria")) {
      moverServo(servo4, "Servo 4");
    } else if (comando.equalsIgnoreCase("apagado")) {
      apagarServos();  // Regresar todos los servos a 0 grados
    } else {
      Serial.println("Comando no reconocido. Prueba con: cebolla, banano, mandarina, zanahoria o apagado.");
    }
  }
```

```cpp
// Función para mover un servo a 180 grados
void moverServo(Servo &servo, const char* nombreServo) {
  // Mover el servo a 180 grados
  servo.write(180);
  Serial.print(nombreServo);
  Serial.println(" movido a 180 grados");
}

// Función para apagar (regresar) todos los servos a la posición 0 grados
void apagarServos() {
  servo1.write(0);
  servo2.write(0);
  servo3.write(0);
  servo4.write(0);

  Serial.println("Todos los servos regresados a 0 grados (Apagado).");
}
```

Fase 4: Replicación del modelo del edge impulse en un notebook de google colab

Posterior a realizar el despliegue con la red neuronal creada en edge impulse e implementada como librería de arduino se replicó la estructura de la red en un notebook de colab, en donde se realizaron los siguiente pasos:

```
[1] from google.colab import drive
    drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
# Se cargan las diferentes dependencias necesarias
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import matplotlib as mpl

from IPython import display
```

```
[3] def load_wav_16k_mono(filename):
        # Load encoded wav file
        file_contents = tf.io.read_file(filename)
        # Decode wav (tensors by channels)
        wav, sample_rate = tf.audio.decode_wav(file_contents, desired_channels=1)
        # Removes trailing axis
        wav = tf.squeeze(wav, axis=-1)
        return wav
```

Inicialmente se le dió permiso al almacenamiento de google drive para poder subir el dataset descargado del edge impulse. Se realizó la importación de las librerías y se realiza la función para cargar los audios que están en formato .wab.

```
[6] def preprocess(wave):
        wav = wave[:16000]
        spectrogram = tf.signal.stft(wav, frame_length=255, frame_step=128)
        spectrogram = tf.abs(spectrogram)
        return spectrogram

espectrograma= preprocess(wave)
print('Waveform shape:', wave.shape)
print('Spectrogram shape:', espectrograma.shape)
print('Audio playback')
display.display(display.Audio(wave, rate=16000))

Waveform shape: (16000,)
Spectrogram shape: (124, 129)
Audio playback
▶ 0:00 / 0:01

[8] def plot_spectrogram(espectrograma, ax):
      # Convert to frequencies to log scale and transpose so that the time is
      # represented in the x-axis (columns).
      log_spec = np.log(espectrograma.T)
      height = log_spec.shape[0]
      width = log_spec.shape[1]
      X = np.linspace(0, np.size(espectrograma), num=width, dtype=int)
      Y = range(height)
      ax.pcolormesh(X, Y, log_spec)


    fig, axes = plt.subplots(2, figsize=(12, 8))
    timescale = np.arange(wave.shape[0])
    axes[0].plot(timescale, wave.numpy())
    axes[0].set_title('Waveform')
    #axes[0].set_xlim([0, 32000])
    axes[0].set_xlim([0, 16000])
    plot_spectrogram(espectrograma.numpy(), axes[1])
    axes[1].set_title('Spectrogram')
    plt.show()
```

Se realizó el preprocesamiento del audio que inicialmente se subió solamente uno como ejemplo y se imprimió la forma del audio junto con su espectrograma.

```
[9] # Se cargan los diferentes archivos *.json que se van a usar en el proceso
    # de entrenamiento
    import os
    directory = '/content/gdrive/MyDrive/miniproyecto-2-elbueno-export.zip (Unzipped Files)/training'
    files = os.listdir(directory)
    files.sort()
    # Filtrar archivos que no sean WAV
    files = [f for f in files if f.endswith('.wav')]  # Solo procesar archivos que terminen con '.wav'

    cantidadFiles = len(files)
    # Variable donde se almacenaron los datos leidos de los archivos *.json
    #Datos=np.zeros((cantidadFiles*624,3))
    Datos=np.zeros((cantidadFiles,124,129))
    i = 0
    for file in files:
      RutaFile=os.path.join(directory, file)
      wave = load_wav_16k_mono(RutaFile)
      espectrograma= preprocess(wave)
      Datos[i,:,:]=espectrograma
      i+=1
    print(Datos.shape)
    Xtrain=tf.expand_dims(Datos, axis=3)
    print(Xtrain.shape)

(625, 124, 129)
(625, 124, 129, 1)

[10] YtrainIni = np.zeros((625, 1))
     for i in range(125):
         YtrainIni[i] = 0
     for i in range(125, 250):
         YtrainIni[i] = 1
     for i in range(250, 375):
         YtrainIni[i] = 2
     for i in range(375, 500):
         YtrainIni[i] = 3
     for i in range(500, 625):
         YtrainIni[i] = 4

     print(YtrainIni)
```

Después de esto se realizó la importación completa de los datos de entrenamiento junto con la carga y preprocesamiento de los mismos. Posterior a esto se dividieron los datos cargados en las diferentes clases de cada audio.

```
[ ] Ytrain= keras.utils.to_categorical(YtrainIni)
    print(Ytrain)

[[1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 ...
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]]

[ ] #Definición del modelo
    modelo = keras.models.Sequential()
    modelo.add(keras.layers.Conv2D(12, 3, activation="relu",padding="same", input_shape=(124,129,1)))
    modelo.add(keras.layers.Conv2D(24, 3, activation="relu",padding="same"))
    modelo.add(keras.layers.Conv2D(36, 3, activation="relu",padding="same"))
    #modelo.add(keras.layers.Conv2D(64, 3, activation="relu",padding="same"))
    modelo.add(keras.layers.MaxPooling2D(pool_size=2,strides=2, padding='same'))
    modelo.add(keras.layers.Flatten())
    modelo.add(keras.layers.Dense(5, activation = 'softmax'))
    modelo.summary()

    keras.utils.plot_model(modelo, to_file='model_plot3.png', show_shapes=True, show_layer_names=True)
```

Después de esto se definió la misma arquitectura realizada en el edge impulse. Y luego se entrenó la red.

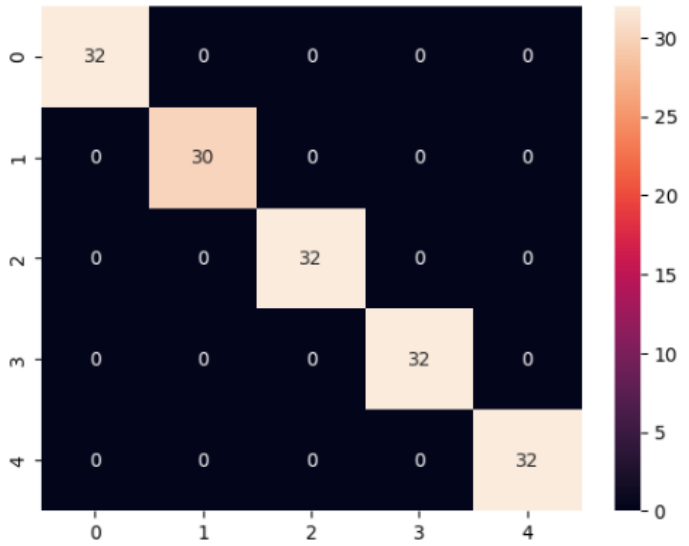

```
[ ] Ytrain= keras.utils.to_categorical(YtrainIni)
    print(Ytrain)

[[1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 ...
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]]

[ ] #Definición del modelo
    modelo = keras.models.Sequential()
    modelo.add(keras.layers.Conv2D(12, 3, activation="relu",padding="same", input_shape=(124,129,1)))
    modelo.add(keras.layers.Conv2D(24, 3, activation="relu",padding="same"))
    modelo.add(keras.layers.Conv2D(36, 3, activation="relu",padding="same"))
    #modelo.add(keras.layers.Conv2D(64, 3, activation="relu",padding="same"))
    modelo.add(keras.layers.MaxPooling2D(pool_size=2,strides=2, padding='same'))
    modelo.add(keras.layers.Flatten())
    modelo.add(keras.layers.Dense(5, activation = 'softmax'))
    modelo.summary()

    keras.utils.plot_model(modelo, to_file='model_plot3.png', show_shapes=True, show_layer_names=True)
```


Ahora se realizó la importación, carga, preprocesamiento y división de clases para los datos de entrenamiento.

Y se procedió a evaluar el modelo para ver su precisión y pérdida, así como analizar su matriz de confusión.

```
[ ] modelo.evaluate(XVal, YVal)

5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 319ms/step - accuracy: 1.0000 - loss: 3.4415e-07
[2.784053663162922e-07, 1.0]
```

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

ypredic=modelo.predict(XVal)

y_test_class = np.argmax(YVal,axis=1)
y_pred_class = np.argmax(ypredic,axis=1)

#Accuracy of the predicted values
print(classification_report(y_test_class, y_pred_class)) # Precision , Recall, F1-Score & Support
cm = confusion_matrix(y_test_class, y_pred_class)
print(cm)
# visualize the confusion matrix in a heat map
df_cm = pd.DataFrame(cm)
heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
```



Después de esto, se intentó hacer el despliegue del modelo desde colab, por lo cual se realizó la conversión a tf lite y se hizo una cuantización, en donde se intentó pasarlo a float16 y por el modelo ser muy pesado para cargarlo en el arduino nano se pasó a int8.

```
[ ] modelo.save('miniproyecto_model.h5')

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` o
```

```
[ ] modelo.save('miniproyecto_model.keras')
```

```
converter = tf.lite.TFLiteConverter.from_keras_model(modelo)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]
modelo_tflite = converter.convert()
open("/content/miniproyecto_model.tflite","wb").write(modelo_tflite)

Saved artifact at '/tmp/tmprvsduvps'. The following endpoints are available

  * Endpoint 'serve'
    args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 124, 129, 1), dtype=tf.
  Output Type:
    TensorSpec(shape=(None, 5), dtype=tf.float32, name=None)
  Captures:
    133215644445056: TensorSpec(shape=(), dtype=tf.resource, name=None)
    133215452917152: TensorSpec(shape=(), dtype=tf.resource, name=None)
    133213732607904: TensorSpec(shape=(), dtype=tf.resource, name=None)
    133213732612128: TensorSpec(shape=(), dtype=tf.resource, name=None)
    133213890305488: TensorSpec(shape=(), dtype=tf.resource, name=None)
    133213890306368: TensorSpec(shape=(), dtype=tf.resource, name=None)
    133214741182272: TensorSpec(shape=(), dtype=tf.resource, name=None)
    133214741184032: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1476092
```

```
[ ] def set_input_tensor(interpreter, Spectro_pru):
      tensor_index = interpreter.get_input_details()[0]['index']
      input_tensor = interpreter.tensor(tensor_index)()[0]
      input_tensor[:,:] = Spectro_pru
```

```
interpreter = tf.lite.Interpreter("/content/miniproyecto_model.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

```
[ ] input_details

[{'name': 'serving_default_keras_tensor_22:0',
  'index': 0,
  'shape': array([  1, 124, 129,   1], dtype=int32),
  'shape_signature': array([ -1, 124, 129,   1], dtype=int32),
  'dtype': numpy.float32,
  'quantization': (0.0, 0),
  'quantization_parameters': {'scales': array([], dtype=float32),
   'zero_points': array([], dtype=int32),
   'quantized_dimension': 0},
  'sparsity_parameters': {}}]
```

```
[ ] output_details

[{'name': 'StatefulPartitionedCall_1:0',
  'index': 24,
  'shape': array([1, 5], dtype=int32),
  'shape_signature': array([-1,  5], dtype=int32),
  'dtype': numpy.float32,
  'quantization': (0.0, 0),
  'quantization_parameters': {'scales': array([], dtype=float32),
   'zero_points': array([], dtype=int32),
   'quantized_dimension': 0},
  'sparsity_parameters': {}}]
```

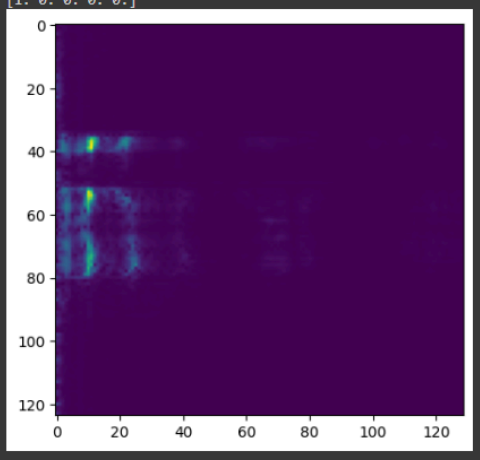
A continuación se muestra como se hizo para el int8:

```python
def representative_dataset_gen():
    for i in range(100):
        # Agrega la dimensión de batch (1) a Xtrain[i]
        input_data = np.expand_dims(Xtrain[i].numpy().astype(np.float32), axis=0)
        yield {
            "keras_tensor_202": input_data,
        }

converter = tf.lite.TFLiteConverter.from_keras_model(modelo)
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Establece los tipos de datos compatibles para la conversión.
# Aquí es donde especificamos la cuantificación a int8.
converter.target_spec.supported_types = [tf.int8]

converter.representative_dataset = representative_dataset_gen

modelo_tflite = converter.convert()
open("/content/miniproyecto_model.tflite","wb").write(modelo_tflite)
```

Después de esto se hizo el ejemplo con un archivo para rectificar la funcionalidad de la red.

```python
Spectro_pru = XVal[14]
plt.imshow(Spectro_pru)
print(YVal[14])
```

```
[1. 0. 0. 0. 0.]
```



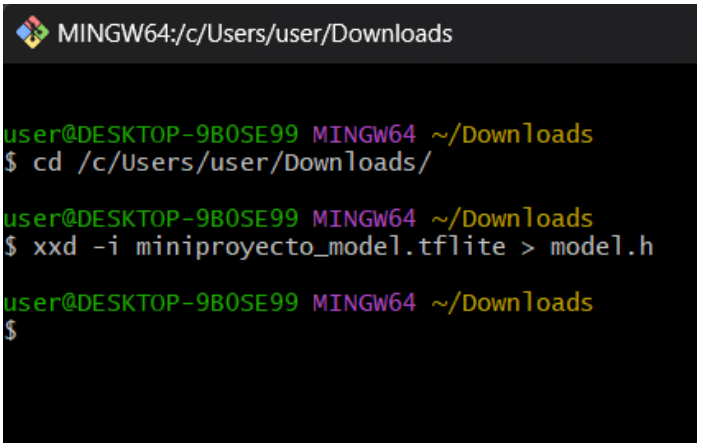

```python
set_input_tensor (interpreter, Spectro_pru)
interpreter. invoke()

output_details = interpreter.get_output_details()[0]

salida=interpreter.get_tensor(output_details['index'])
print(salida)

img_pred = np.argmax(salida)
print(img_pred)
```

```
[[1.0000000e+00 2.8407223e-14 1.4610296e-25 3.0601788e-22 1.9482786e-23]]
0
```

Por último se descargó el modelo para poder realizar el despliegue.

```python
[ ] from google.colab import files
    files.download('/content/miniproyecto_model.tflite')
```

A lo cual me descarga un archivo con el nombre especificado que posteriormente se convierte de tflite a un array C.



Lo que me genera un archivo llamado model.h que se deberá incluir en una carpeta donde vayamos a crear el sketch de arduino. Sin embargo hasta acá se logró llevar a cabo el intento de desplegar desde colab, debido a que no se supo bien cómo realizar la implementación, se intentó con el ejemplo del micrófono y con una programación inspirada en la librería que da el edge impulse pero aún así no se logró.

V. CONCLUSIONES

Este proyecto ha demostrado la capacidad de utilizar TinyML para la automatización del riego de cultivos mediante el reconocimiento de comandos de voz.
El proyecto está alineado con los ODS 6 y 12, ya que promueve el uso responsable del agua y fomenta prácticas agrícolas sostenibles. A pesar de los buenos resultados en la fase de entrenamiento y validación, se identificaron desafíos en la inferencia en tiempo real en Arduino. El sistema ofrece una solución prometedora para la agricultura inteligente, aunque requiere más optimizaciones para mejorar su precisión en escenarios de despliegue práctico.

Mejoras Futuras:

1. Optimización del Preprocesamiento de Audio

Durante las pruebas en tiempo real, se observó que la precisión del modelo disminuye ligeramente al ejecutar la inferencia en el dispositivo embebido. Una posible mejora sería implementar un preprocesamiento adicional en tiempo real para mejorar la calidad del audio antes de ser clasificado. Esto podría incluir técnicas de reducción de ruido, normalización de volumen y filtrado de señales no deseadas.

2. Mejoras en la Inferencia en Tiempo Real

Se pueden explorar técnicas de optimización de inferencia para mejorar la velocidad y la precisión del sistema al ejecutar el modelo en el microcontrolador. Esto podría implicar ajustes en el código de inferencia de Arduino o el uso de bibliotecas especializadas en la aceleración de redes neuronales en hardware embebido, como TensorFlow Lite Micro.

3. Ampliación de la Base de Datos de Audio

Si bien el modelo fue entrenado con un conjunto de datos bien definido, sería beneficioso expandir la base de datos de audio para incluir más clases y comandos. Además, se podrían recopilar más muestras de voz de diferentes usuarios para hacer que el modelo sea más robusto y capaz de manejar una mayor variedad de entonaciones y acentos.

4. Integración de Sensores Adicionales

Se podrían añadir más sensores para mejorar el sistema de riego, como sensores de humedad del suelo, temperatura y luz ambiental. Esto permitiría que el sistema tome decisiones más informadas sobre cuándo activar el riego, basándose no solo en comandos de voz, sino también en las condiciones ambientales en tiempo real.

5. Implementación de una Interfaz Gráfica de Usuario (GUI)

Se podría desarrollar una aplicación móvil o una interfaz web más completa para facilitar el control remoto del sistema. Esta interfaz podría mostrar datos históricos sobre el riego, el estado actual de los cultivos, e incluso ofrecer recomendaciones sobre cuándo es el mejor momento para regar en función de las condiciones climáticas locales.

VI. REFERENCIAS

- Arduino. "Arduino Nano 33 BLE Sense." [Online]. Available: https://store.arduino.cc/nano-33-ble-sense. [Accessed: 09-Sep-2024].

- A. He and M. Wimmer, "Fall Detection Algorithms for Elderly Care: A Review," IEEE Access, vol. 7, pp. 77695-77705, Jun. 2019, doi: 10.1109/ACCESS.2019.2917607.

- Edge Impulse. "Getting started with TinyML on Arduino." [Online]. Available: https://docs.edgeimpulse.com/docs/tutorials/arduino-nano-33-ble-sense. [Accessed: 09-Sep-2024].

- A. Cavallaro, R. Gaddam, and S. Mukhopadhyay, "A Survey on Fall Detection Systems With a Focus on Wearable and Unobtrusive Sensing Solutions," IEEE Sensors Journal, vol. 17, no. 21, pp. 7728-7735, Nov. 2017, doi: 10.1109/JSEN.2017.2742905.

- B. Xie, L. Hu, and H. Zhang, "Design and Implementation of a Mobile Elderly Monitoring System," in Proc. of the 2018 IEEE Int. Conf. on Artificial Intelligence and Big Data (ICAIBD), Chengdu, China, 2018, pp. 125-129, doi: 10.1109/ICAIBD.2018.8396186.

- P. K. Agarwal, M. F. Alhaddad, and A. S. Khamis, "Evaluation of Bluetooth Low Energy for Wearable Fall Detection Systems," in Proc. of the 2019 IEEE 16th Int. Conf. on Wearable and Implantable Body Sensor Networks (BSN), Chicago, IL, USA, 2019, pp. 1-4, doi: 10.1109/BSN.2019.8771027.

- S. Liu and D. Zhou, "A Smart Monitoring System for Senior Citizens Based on IoT Technologies," in IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4981-4989, Jun. 2019, doi: 10.1109/JIOT.2019.2902816.

- J. Smith and R. Liu, "Fall Detection Systems: A Comprehensive Review," IEEE Rev. Biomed. Eng.,

vol. 15, pp. 116-130, Mar. 2022, doi: 10.1109/RBME.2022.3143275.

- A. Bagheri, M. Y. Naderi, and S. Khorasani, "Bluetooth Low Energy: A Survey of Modulation Techniques and Protocols," IEEE Commun. Surveys Tuts., vol. 23, no. 4, pp. 2558-2582, Fourthquarter 2021, doi: 10.1109/COMST.2021.3109124.