MODELOS DE PYTHON A TENSORFLOW.JS



Adrián Yared Armas de la Nuez

**Contenido**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[**1. Temperaturas 2**](#_2c35568x6wg8)

[**1.1 Enunciado 2**](#_8r9fgvxupc1e)

[**1.2 Resolución 2**](#_i2e6wx48do58)

[**1.2.1 Requerimientos 2**](#_iy0wu44h2ohi)

[**1.2.2 Imports 2**](#_1t0m7ukb0q8u)

[**1.2.3 Dataset 2**](#_4motzp1dd7lj)

[**1.2.4 Division de los datos 3**](#_8pru1tnf02cr)

[**1.2.5 Modelo keras 3**](#_1h5ds63hjpz7)

[**1.2.6 Entrenamiento 4**](#_533gmcwgoz05)

[**1.2.7 Evaluación 4**](#_gq092lvnq8uv)

[**1.2.8 Guardado del modelo 4**](#_59mm6vvfm3xg)

[**1.2.9 comprobación de los resultados 4**](#_2ajpmiw61767)

[**1.2.10 Puesta en producción 5**](#_swx8etzfdk5g)

[**2. Flores 7**](#_kllkudm3uuo1)

[**2.1 Enunciado 7**](#_t8cwbo6n4466)

[**2.2 Resolución 7**](#_nzvpbwsratuo)

[**2.2.1 Requerimientos 7**](#_757oarzeqglk)

[**2.2.2 Imports 8**](#_wyztueup5ctk)

[**2.2.3 Dataset 8**](#_t9pu7x65712c)

[**2.2.4 Modelo redes convolucionales 9**](#_kdalvo4e07uf)

[**2.2.5 Limpieza de valores atípicos 10**](#_hzrfqvs03cf)

[**2.2.6 Entrenamiento 11**](#_szxkpr6ay8zz)

[**2.2.7 Guardado del modelo 12**](#_aporrt4ir8ly)

[**2.2.8 Puesta en producción 12**](#_g1fn06d5855n)

## 

## 

## 

## **1. Temperaturas**

### **1.1 Enunciado**

Realiza la tarea de implementar un modelo para convertir temperaturas de grados Fahrenheit a centígrados. Expórtalo a Tensorflow.js e implementa la aplicación web para que use el modelo.

Descarga la función de conversión y genera el dataset .csv con al menos 1000 temperaturas.

Divide los datos en 80% training y 20% test. Los datos de trainning reserva un 5% para

validación.

Muestra las gráficas de pérdida y precisión.

### **1.2 Resolución**

#### **1.2.1 Requerimientos**

!pip install scikit-learn # Instalar scikit-learn

#### **1.2.2 Imports**

import numpy as np # Manejo arrays

import pandas as pd # Manipular datos

import matplotlib.pyplot as plt # Graficar datos

import tensorflow as tf # Librería ML

from sklearn.model\_selection import train\_test\_split # Dividir datos

#### **1.2.3 Dataset**

Genera 1200 temperaturas aleatorias en Fahrenheit, las convierte a Celsius y guarda ambos datos en un archivo CSV. Luego muestra las primeras filas del dataset creado.

# Configurar semilla para reproducibilidad

np.random.seed(42)

# Parámetros

n\_samples = 1200 # Número de muestras

# Generar temperaturas en Fahrenheit aleatorias entre -100 y 200

temp\_fahrenheit = np.random.uniform(-100, 200, n\_samples)

# Convertir a Celsius

temp\_celsius = (temp\_fahrenheit - 32) \* 5 / 9

# Crear DataFrame con ambas columnas

temperatures\_df = pd.DataFrame({

'fahrenheit': temp\_fahrenheit,

'celsius': temp\_celsius

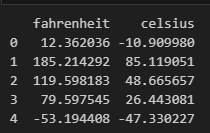
})

# Guardar DataFrame a CSV sin índice

temperatures\_df.to\_csv('temperature\_dataset.csv', index=False)

# Mostrar primeras filas

print(temperatures\_df.head())



#### **1.2.4 Division de los datos**

Se separan las temperaturas Fahrenheit y Celsius en arrays y luego se dividen en conjuntos de entrenamiento, validación y prueba para preparar el modelo.

# Extraemos variables y dividimos en entrenamiento (70%), validación (10%) y prueba (20%)

X = temperatures\_df[['fahrenheit']].values

y = temperatures\_df[['celsius']].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.05, random\_state=42)

#### **1.2.5 Modelo keras**

Se crea una red neuronal con dos capas ocultas para aprender la relación entre Fahrenheit y Celsius, y una capa de salida para predecir la temperatura en Celsius.

# Definimos un modelo secuencial con dos capas ocultas y una de salida para regresión

model = tf.keras.Sequential([

tf.keras.layers.Dense(16, activation='relu', input\_shape=(1,)),

tf.keras.layers.Dense(8, activation='relu'),

tf.keras.layers.Dense(1)

])

#### **1.2.6 Entrenamiento**

Se configura el modelo para minimizar el error cuadrático medio y se entrena durante 100 épocas usando los datos de entrenamiento y validación para ajustar el modelo.

# Compilamos el modelo con optimizador Adam y error cuadrático medio, y entrenamos con validación

model.compile(optimizer='adam', loss='mean\_squared\_error', metrics=['mae'])

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_data=(X\_val, y\_val), verbose=1)

#### **1.2.7 Evaluación**

# Evaluamos el modelo con los datos de prueba y mostramos la pérdida y el error absoluto medio

test\_loss, test\_mae = model.evaluate(X\_test, y\_test)

print(f"Test Loss: {test\_loss:.4f}, Test MAE: {test\_mae:.4f}")



#### **1.2.8 Guardado del modelo**

# Guardamos el modelo entrenado y visualizamos las curvas de pérdida y MAE durante el entrenamiento

model.save('temperature\_model.keras')

#### **1.2.9 comprobación de los resultados**

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 2)

plt.plot(history.history['mae'], label='Training MAE') # Error absoluto medio en entrenamiento

plt.plot(history.history['val\_mae'], label='Validation MAE') # Error absoluto medio en validación

plt.title('MAE durante el entrenamiento')

plt.xlabel('Epoch')

plt.ylabel('MAE')

plt.legend()

plt.tight\_layout()

plt.show()

plt.figure(figsize=(10, 6))

y\_pred = model.predict(X\_test)

plt.scatter(X\_test, y\_test, color='blue', label='Valores reales') # Datos reales de prueba

plt.scatter(X\_test, y\_pred, color='red', alpha=0.4, label='Predicciones') # Predicciones del modelo

plt.title('Predicciones vs Valores reales')

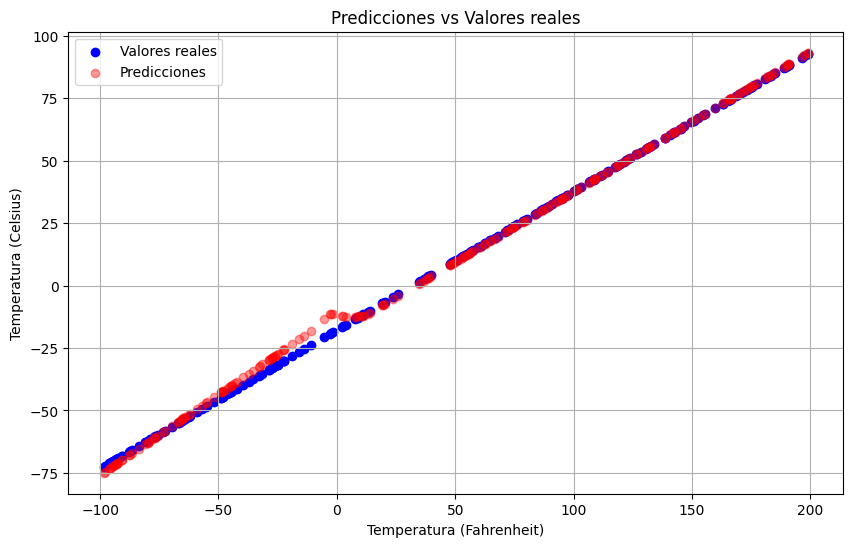
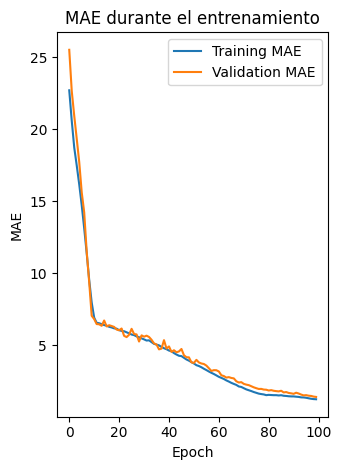
plt.xlabel('Temperatura (Fahrenheit)')

plt.ylabel('Temperatura (Celsius)')

plt.legend()

plt.grid()

plt.show()



#### **1.2.10 Puesta en producción**

Html:

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>Conversión F → C con TensorFlow.js</title>

<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>

</head>

<body>

<main>

<h1>Convertir Fahrenheit a Celsius</h1>

<label for="fahrenheitInput">Temperatura en Fahrenheit:</label>

<input type="number" id="fahrenheitInput" value="32" />

<button type="button" onclick="convert()">Convertir</button>

<h2>Resultado: <span id="result">...</span> °C</h2>

</main>

<script>

let model;

// Cargar el modelo al iniciar la página

async function loadModel() {

model = await tf.loadLayersModel('model.json');

console.log("Modelo cargado.");

}

// Función para convertir Fahrenheit a Celsius usando el modelo

async function convert() {

if (!model) {

alert("El modelo aún no ha sido cargado.");

return;

}

const f = parseFloat(document.getElementById('fahrenheitInput').value);

const input = tf.tensor2d([f], [1, 1]);

const output = model.predict(input);

output.array().then(result => {

document.getElementById('result').innerText = result[0][0].toFixed(2);

});

}

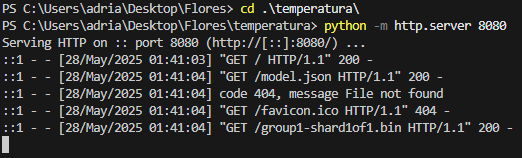
loadModel();

</script>

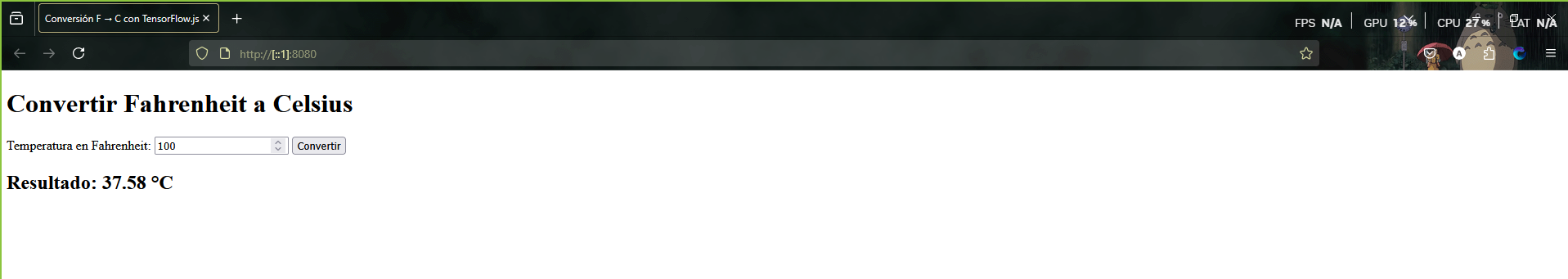
</body>

</html>

Inicio del servicio:



Prueba:



## **2. Flores**

### **2.1 Enunciado**

Descarga el data set de flores de: <https://www.kaggle.com/datasets/imsparsh/flowers-dataset?resource=download>

Implementa en Python el modelo de red convolucional que clasifique correctamente las flores, expórtalo y úsalo en una aplicación web en la que se seleccionará una imagen e indicará su nombre.

Referencias:

<https://www.youtube.com/watch?v=JpE4bYyRADI><https://www.smashingmagazine.com/2019/09/machine-learning-front-end-developers-tensorflowjs/?utm_campaign=machine-learning-for-frontend-developers>

### **2.2 Resolución**

#### **2.2.1 Requerimientos**

Instalación de las librerías y dataset desde kaggle

%pip install kaggle

!kaggle datasets download -d imsparsh/flowers-dataset

Descomprimido del dataset

!tar -xf flowers-dataset.zip

Instalación Pillos y tensorflow

%pip install pillow

%pip install tensorflow

#### **2.2.2 Imports**

import os # Manejo archivos

import tensorflow as tf # Librería ML

from tensorflow.keras.preprocessing.image import ImageDataGenerator # Aumento imágenes

from tensorflow.keras.models import Sequential # Modelo secuencial

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout # Capas CNN

import subprocess # Ejecutar comandos

#### **2.2.3 Dataset**

El código define las rutas y parámetros necesarios para cargar imágenes desde carpetas, incluyendo el tamaño al que se redimensionarán y el tamaño de los lotes. Luego, normaliza las imágenes escalando sus píxeles para que estén entre 0 y 1. A continuación, crea una función que carga y normaliza los datos desde la carpeta de entrenamiento, separándolos automáticamente en conjuntos de entrenamiento y validación según un porcentaje. Finalmente, utiliza esta función para cargar ambos conjuntos y prepararlos para el entrenamiento del modelo.

# RUTAS

TRAIN\_DIR = "train" # Carpeta con datos de entrenamiento

TEST\_DIR = "test" # Carpeta con datos de prueba

MODEL\_FILE = "modelo\_flores.h5" # Archivo para guardar el modelo

# PARÁMETROS

img\_size = (150, 150) # Tamaño al que se redimensionan las imágenes

batch\_size = 32 # Tamaño del lote para entrenamiento

validation\_split = 0.2 # Porcentaje para validación

# NORMALIZACIÓN

normalization\_layer = tf.keras.layers.Rescaling(1./255) # Normaliza píxeles a rango [0,1]

# FUNCIÓN PARA CARGAR Y NORMALIZAR DATASET

def load\_dataset(subset):

return tf.keras.utils.image\_dataset\_from\_directory(

TRAIN\_DIR,

validation\_split=validation\_split,

subset=subset,

seed=123,

image\_size=img\_size,

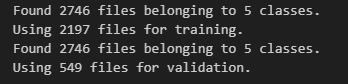
batch\_size=batch\_size

).map(lambda x, y: (normalization\_layer(x), y)) # Aplicar normalización

# Cargar datasets de entrenamiento y validación usando la función

train\_ds = load\_dataset("training")

val\_ds = load\_dataset("validation")



#### **2.2.4 Modelo redes convolucionales**

Este código define una red neuronal convolucional secuencial para clasificar imágenes en cinco categorías. Comienza con dos bloques de capas convolucionales y de pooling para extraer características espaciales relevantes. Luego aplana la salida para conectar con una capa densa, que incluye dropout para evitar el sobreajuste. Finalmente, la capa de salida usa softmax para predecir la probabilidad de cada una de las cinco clases.

# PARÁMETROS DEL MODELO

input\_shape = (150, 150, 3) # Tamaño y canales de la imagen

num\_classes = 5 # Número de clases a predecir

# DEFINICIÓN DEL MODELO

model = Sequential([

# Primera capa convolucional + pooling

Conv2D(32, kernel\_size=(3,3), activation='relu', input\_shape=input\_shape),

MaxPooling2D(pool\_size=(2,2)),

# Segunda capa convolucional + pooling

Conv2D(64, kernel\_size=(3,3), activation='relu'),

MaxPooling2D(pool\_size=(2,2)),

# Aplanar para conectar con capas densas

Flatten(),

# Capa densa con Dropout para regularización

Dense(128, activation='relu'),

Dropout(0.5),

# Capa de salida con activación softmax para clasificación multiclase

Dense(num\_classes, activation='softmax')

])

#### **2.2.5 Limpieza de valores atípicos**

Este código revisa todas las imágenes dentro de las carpetas de entrenamiento (o prueba) para detectar archivos corruptos. Usa PIL para abrir y verificar cada imagen sin cargarla completamente. Si encuentra alguna imagen dañada o ilegible, la elimina y registra su ruta. Al final, muestra un resumen de cuántas imágenes corruptas fueron eliminadas para mantener el dataset limpio y listo para usar.

from PIL import Image, UnidentifiedImageError

import os

base\_dir = "train" # Carpeta a limpiar ("train" o "test")

clases = os.listdir(base\_dir) # Lista de subcarpetas/clases

errores = [] # Lista para almacenar imágenes corruptas

for clase in clases:

ruta\_clase = os.path.join(base\_dir, clase)

if not os.path.isdir(ruta\_clase):

continue # Saltar si no es carpeta

for nombre\_archivo in os.listdir(ruta\_clase):

ruta\_imagen = os.path.join(ruta\_clase, nombre\_archivo)

try:

with Image.open(ruta\_imagen) as img:

img.verify() # Verifica si la imagen está corrupta sin cargarla completamente

except (UnidentifiedImageError, OSError):

errores.append(ruta\_imagen) # Guardar ruta de imagen corrupta

print(f"❌ Imagen corrupta eliminada: {ruta\_imagen}")

os.remove(ruta\_imagen) # Eliminar imagen corrupta

print(f"✔️ Validación completa. {len(errores)} imagen(es) eliminadas.")

#### **2.2.6 Entrenamiento**

Se crea y compila una red convolucional para clasificar imágenes en cinco clases, usando capas convolucionales, pooling y dropout para evitar sobreajuste. Luego, se entrena el modelo durante 10 épocas con los datos de entrenamiento y validación.

# DEFINICIÓN DEL MODELO CNN

model = Sequential([

Conv2D(32, (3,3), activation='relu', input\_shape=(150, 150, 3)), # Capa convolucional inicial

MaxPooling2D(2,2), # Reducción de dimensionalidad

Conv2D(64, (3,3), activation='relu'), # Segunda capa convolucional

MaxPooling2D(2,2), # Nueva reducción de tamaño

Flatten(), # Aplanar para conectar con capas densas

Dense(128, activation='relu'), # Capa densa para aprendizaje complejo

Dropout(0.5), # Regularización para evitar sobreajuste

Dense(5, activation='softmax') # Capa salida para 5 clases con softmax

])

# Compilar el modelo con optimizador Adam y función de pérdida para clasificación multiclase

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# ENTRENAMIENTO DEL MODELO

model.fit(train\_ds, epochs=10, validation\_data=val\_ds)

#### **2.2.7 Guardado del modelo**

# Guardar el modelo entrenado en un archivo

model.save(MODEL\_FILE)

print(f"✅ Modelo guardado correctamente en: {MODEL\_FILE}")



#### **2.2.8 Puesta en producción**

Html:  
<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>Clasificador de Flores</title>

<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>

<style>

body {

font-family: sans-serif;

padding: 20px;

}

canvas {

border: 1px solid #aaa;

margin-top: 10px;

display: block;

}

label {

font-weight: bold;

}

#predictBtn {

margin-top: 10px;

padding: 6px 12px;

cursor: pointer;

}

</style>

</head>

<body>

<main>

<h2>Clasificador de Flores 🌼</h2>

<label for="imageUpload">Sube una imagen:</label><br />

<input type="file" id="imageUpload" accept="image/\*" /><br /><br />

<canvas id="canvas" width="150" height="150" aria-label="Vista previa de la imagen"></canvas><br />

<button id="predictBtn" type="button">Predecir</button>

<p id="prediction">Predicción: ...</p>

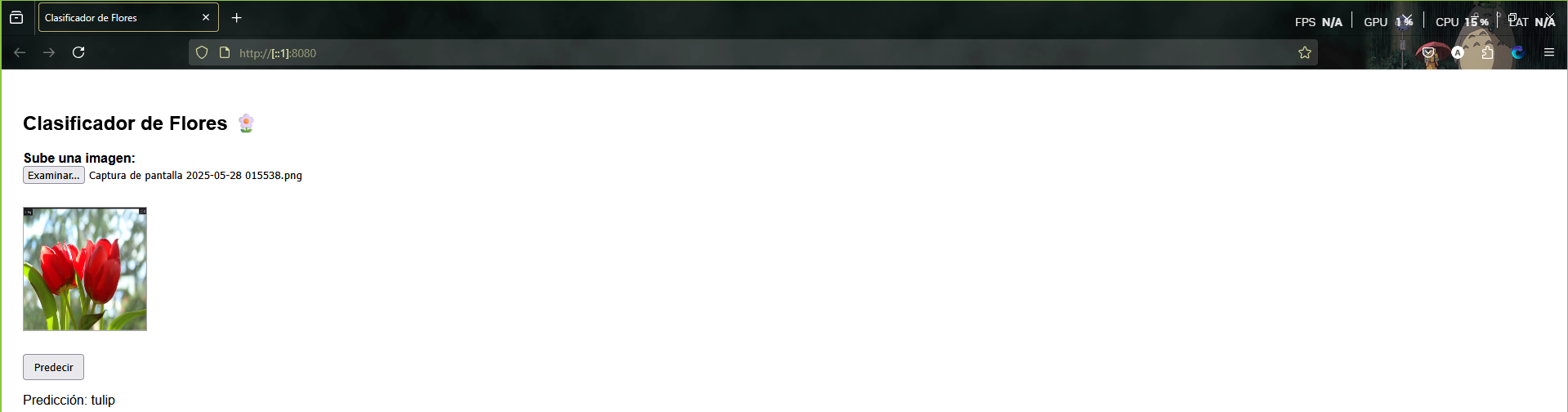
</main>

<script src="script.js"></script>

</body>

</html>

Ejecución:  


Prueba:  


Script:

Carga un modelo de TensorFlow.js para clasificar flores y permite al usuario subir una imagen.

Muestra la imagen en un canvas y realiza la predicción con el modelo cargado.

Finalmente, muestra la clase más probable al usuario, manejando la interacción completa.

const classNames = ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip'];

const imageInput = document.getElementById('imageUpload');

const canvas = document.getElementById('canvas');

const ctx = canvas.getContext('2d');

const predictBtn = document.getElementById('predictBtn');

const predictionText = document.getElementById('prediction');

let model = null;

let imageLoaded = false;

// Carga el modelo TensorFlow.js

async function loadModel() {

try {

model = await tf.loadLayersModel('tfjs\_model/model.json');

console.log("✅ Modelo cargado correctamente");

} catch (error) {

console.error("❌ Error al cargar el modelo:", error);

}

}

// Dibuja la imagen cargada en el canvas

function drawImageOnCanvas(img) {

ctx.clearRect(0, 0, canvas.width, canvas.height);

ctx.drawImage(img, 0, 0, 150, 150);

imageLoaded = true;

}

// Maneja la carga de imagen desde input

function handleImageUpload(event) {

const file = event.target.files[0];

if (!file) return;

const img = new Image();

const reader = new FileReader();

reader.onload = () => {

img.src = reader.result;

};

img.onload = () => {

drawImageOnCanvas(img);

};

reader.readAsDataURL(file);

}

// Realiza la predicción con el modelo cargado

async function predict() {

if (!model) {

alert("Modelo no cargado aún");

return;

}

if (!imageLoaded) {

alert("Primero sube una imagen");

return;

}

const tensor = tf.browser.fromPixels(canvas)

.resizeNearestNeighbor([150, 150]) // aseguramos tamaño esperado, opcional si el canvas ya tiene 150x150

.expandDims(0)

.div(255.0);

try {

const prediction = await model.predict(tensor).data();

const index = prediction.indexOf(Math.max(...prediction));

predictionText.innerText = `Predicción: ${classNames[index]}`;

} catch (error) {

console.error("Error durante la predicción:", error);

}

}

// Event listeners

imageInput.addEventListener('change', handleImageUpload);

predictBtn.addEventListener('click', predict);

// Inicialización

loadModel();