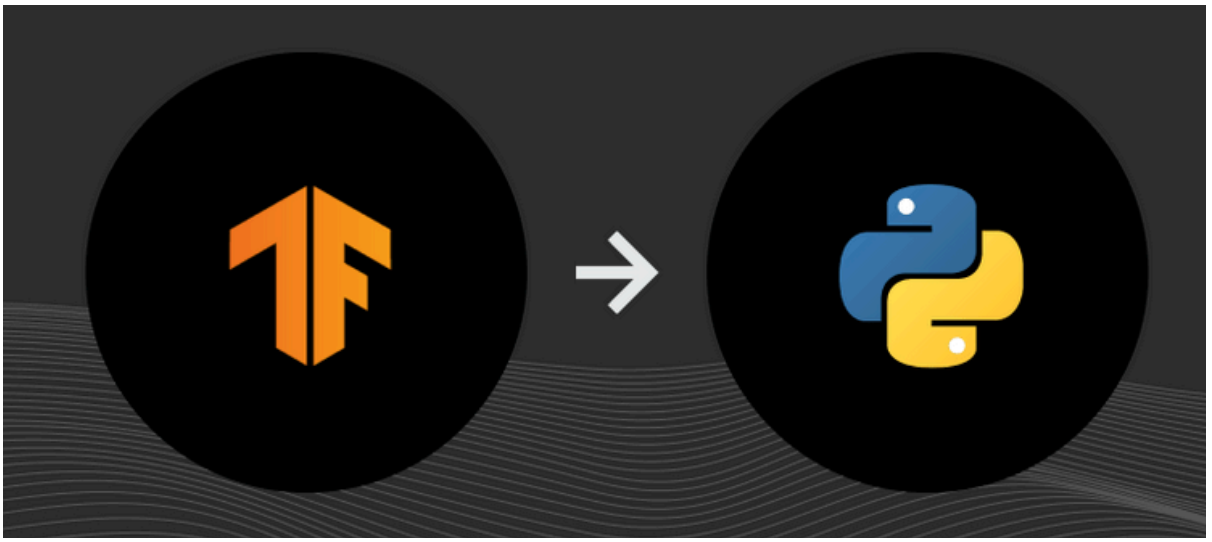
TAREA MODELOS
DE PYTHON A TENSORFLOW.JS



ÍNDICE

Requisitos previos (para los dos ejercicios)

Versiones de librerías y python:
→ tensorflow==2.11.0
→ tensorflowjs==3.18.0
→ numpy==1.23.5
→ ipykernel
→ Versión de python: 3.10.11

***Nota**: Las librerías se instalan automáticamente al ejecutar el archivo con el modelo, porque coje el archivo de requirements.txt donde están las librerías mencionadas con sus versiones.*

Ejercicio 1: Conversión de Temperatura con TensorFlow.js

1. Realiza la tarea de implementar un modelo para convertir temperaturas de grados Fahrenheit a centígrados. Exportarlo a Tensor Flow.js e implementar 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 training reservan un 5% para validación.
- ➔ Muestra las gráficas de pérdida y precisión.

Objetivo

Implementar un modelo de aprendizaje automático que convierte temperaturas de grados Fahrenheit a grados Celsius. Exportar el modelo a TensorFlow.js y desarrollar una aplicación web que utilice el modelo para hacer predicciones en tiempo real.

0. Estructura del Proyecto

```
conversor_tfjs/
│
├── conversion_modelo.py        # Script Python que genera dataset, entrena y exporta el modelo
├── dataset_temperaturas.csv    # Dataset generado
├── modelo_temperatura_js/      # Carpeta con el modelo exportado para TensorFlow.js
│   ├── model.json
│   └── groupX-shardXofX.bin
├── index.html                  # Página web
├── conversion.js               # Lógica JavaScript de predicción
├── estilos.css                 # (opcional) Estilo visual
├── requirements.txt            # (opcional) Dependencias necesarias
├── video1.mp4                  # (opcional) Vídeo del funcionamiento de la app
```

***Nota final**: los códigos completos del Ejercicio 1 están en el **Anexo 1 (adicional)**, también se pueden descargar del repositorio de github (está en el apartado de Enlaces y Referencias)*

También se ha creado un video para cada uno de los ejercicios comprobando el funcionamiento de la página web, está dentro de cada carpeta correspondiente al ejercicio.

1. Generación del Dataset

Se creó un script en Python que genera un conjunto de datos con **1000 temperaturas** en Fahrenheit, distribuidas uniformemente desde **-100°F** hasta **212°F**. Estas temperaturas se convierten a grados Celsius utilizando la fórmula clásica: **Celsius = (Fahrenheit - 32) × 5/9**

$$°C = (°F - 32) \times \frac{5}{9}$$

El resultado se guarda en un archivo CSV llamado dataset_temperaturas.csv.

→ Código relacionado:

```
38  # ------------------- Sección 3: Generación de datos -------------------
39
40  # Crear una serie de valores en Fahrenheit entre -100 y 212 grados
41  fahrenheit = np.linspace(-100, 212, 1000, dtype=np.float32)
42
43  # Convertir esos valores a Celsius usando la fórmula de conversión
44  celsius = (fahrenheit - 32) * 5 / 9
45
46  # Crear un DataFrame y guardarlo como archivo CSV
47  data = pd.DataFrame({'Fahrenheit': fahrenheit, 'Celsius': celsius})
48  data.to_csv('dataset_temperaturas.csv', index=False)
49  print("Dataset guardado como 'dataset_temperaturas.csv'")
50
```

2. División de Datos

El dataset generado se dividió en:

- ➔ 80% para entrenamiento, dentro del cual:
- ➔ 5% se usó como conjunto de validación.
- ➔ 20% restante para test.

Código relacionado:

```
51  # ------------------- Sección 4: División de datos -------------------
52
53  # Dividir los datos en entrenamiento+validación (80%) y prueba (20%)
54  X_train_val, X_test, y_train_val, y_test = train_test_split(
55      fahrenheit, celsius, test_size=0.2, random_state=42
56  )
57
58  # Dividir el conjunto de entrenamiento+validación en entrenamiento (95%) y validación (5%)
59  X_train, X_val, y_train, y_val = train_test_split(
60      X_train_val, y_train_val, test_size=0.05, random_state=42
61  )
62
```

3. Creación y Entrenamiento del Modelo

Se construyó un modelo con **TensorFlow Keras**, con la siguiente arquitectura:

- ➔ Una capa de entrada (1 valor de entrada)
- ➔ Una capa oculta con 64 neuronas y activación ReLU
- ➔ Una capa de salida con 1 neurona (resultado en grados Celsius)

Se utilizó el optimizador **Adam** y la función de pérdida de **error cuadrático medio** (MSE).

Código relacionado:

```python
# ------------------- Sección 5: Creación y entrenamiento del modelo -------------------

# Crear un modelo secuencial con una capa oculta y una capa de salida
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(1,)),          # Entrada de un valor (temperatura)
    tf.keras.layers.Dense(64, activation='relu'), # Capa oculta con 64 neuronas
    tf.keras.layers.Dense(1)                     # Capa de salida con 1 neurona (temperatura en Celsius)
])

# Compilar el modelo especificando el optimizador y la función de pérdida
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Entrenar el modelo con los datos de entrenamiento y validación
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=200,
    verbose=1
)
```

4. Gráficas de Pérdida y Precisión

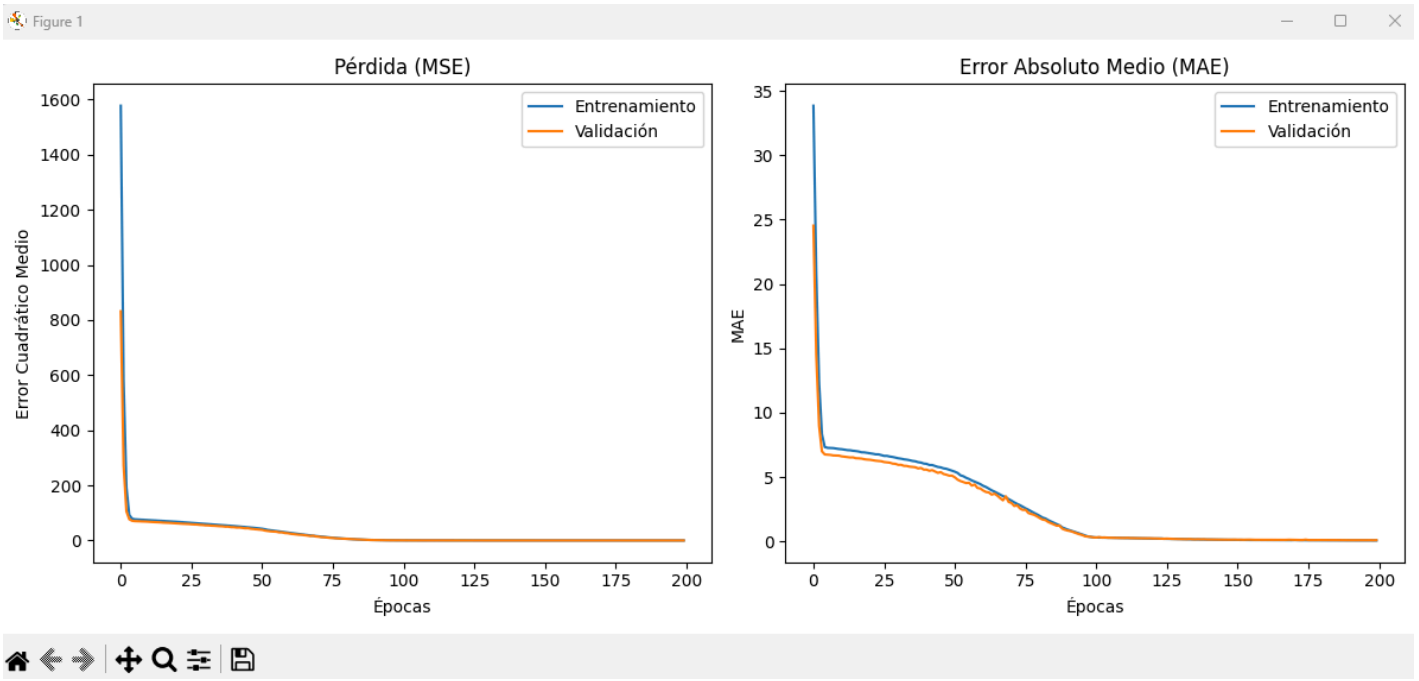
Se generaron dos gráficos para visualizar el entrenamiento del modelo:

- ➔ **Pérdida (MSE)**: muestra cómo el error disminuye con cada época.
- ➔ **Precisión (MAE)**: muestra cómo mejora el error absoluto medio.

Estas gráficas ayudan a evaluar el rendimiento del modelo y a detectar problemas como sobreajuste.

Código relacionado:

```
# ------------------- Sección 6: Visualización de entrenamiento ------------------

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

# Gráfico de la pérdida (MSE)
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title("Pérdida (MSE)")
plt.xlabel("Épocas")
plt.ylabel("Error Cuadrático Medio")
plt.legend()

# Gráfico del error absoluto medio (MAE)
plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Entrenamiento')
plt.plot(history.history['val_mae'], label='Validación')
plt.title("Error Absoluto Medio (MAE)")
plt.xlabel("Épocas")
plt.ylabel("MAE")
plt.legend()

# Mostrar los gráficos
plt.tight_layout()
plt.show()
```



Pérdida (MSE)

Se ve cómo tanto entrenamiento como validación disminuyen rápidamente y luego se estabilizan muy cerca de cero.

Esto indica que el modelo aprendió correctamente la relación lineal entre Fahrenheit y Celsius.

Error Absoluto Medio (MAE)

Bien, el error absoluto baja desde valores altos y llega casi a cero.

Las curvas de entrenamiento y validación se mantienen muy pegadas, lo que indica:

- ➔ Buena generalización.
- ➔ Nada de sobreajuste (overfitting).

5. Exportación a TensorFlow.js

El modelo entrenado se exportó a formato compatible con TensorFlow.js usando la herramienta tensorflowjs. El modelo resultante se guardó en la carpeta modelo_temperatura_js.

Código relacionado:

```python
# ------------------- Sección 7: Exportación del modelo a TensorFlow.js -------------------

# Ruta de salida para el modelo exportado
export_dir = 'modelo_temperatura_js'

# Eliminar carpeta existente si ya hay un modelo previo
if os.path.exists(export_dir):
    import shutil
    shutil.rmtree(export_dir)

# Exportar el modelo entrenado en formato compatible con TensorFlow.js
tfjs.converters.save_keras_model(model, export_dir)
print("Modelo exportado a TensorFlow.js en:", export_dir)
```

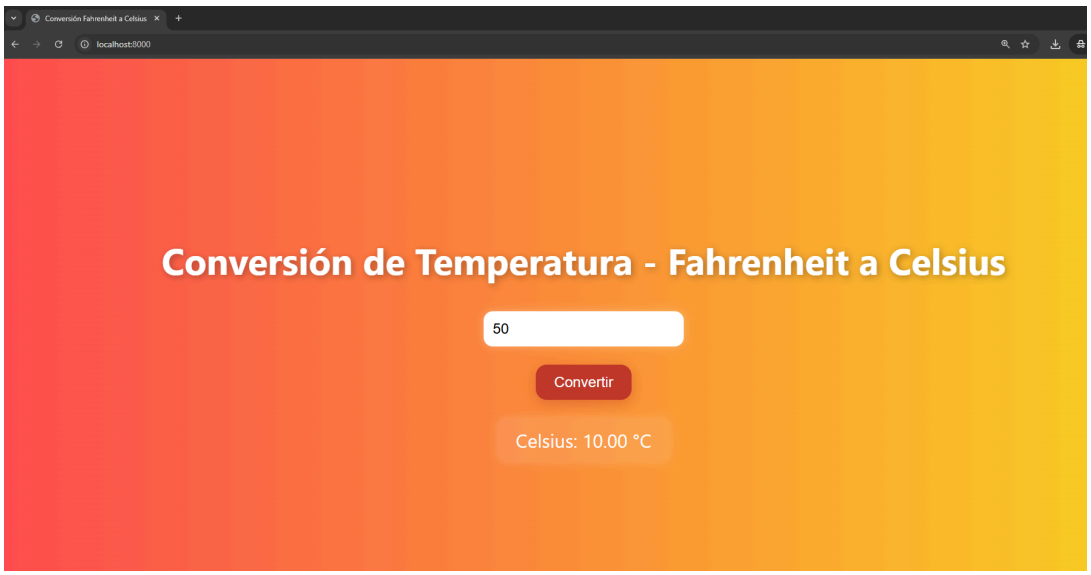
6. Implementación Web

Se desarrolló una aplicación web sencilla en HTML, que permite al usuario ingresar un valor en Fahrenheit y obtener la predicción en Celsius, utilizando el modelo exportado.

Archivos involucrados:

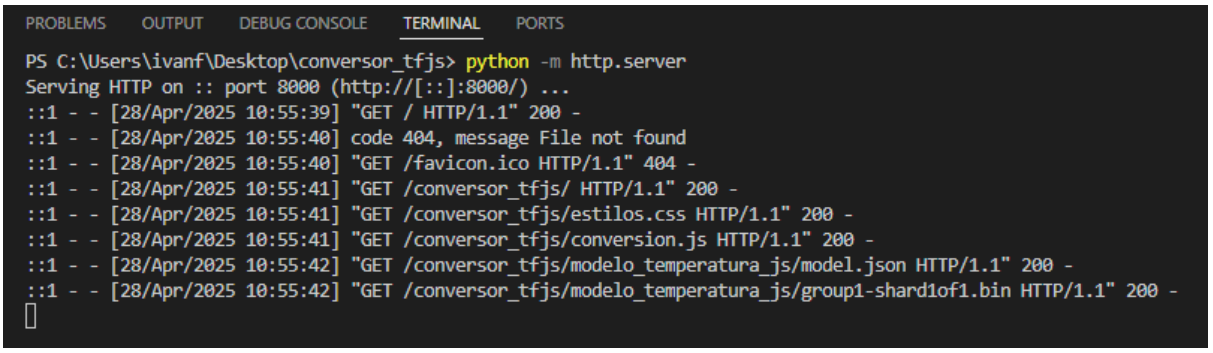
- ➔ **index.html**: Interfaz web
- ➔ **conversion.js**: Lógica de predicción usando el modelo cargado con TensorFlow.js
- ➔ **estilos.css**: (opcional) Hoja de estilo para presentación visual

El modelo se carga al inicio y se usa para hacer predicciones en tiempo real al pulsar un botón.



Para ejecutar el servidor HTTP local simple en python:

→ python -m http.server
→ http://localhost:8000/conversor_tfjs/



Ejercicio 2

Objetivo

Clasificación de flores mediante una **Red Neuronal Convolucional** (CNN) y despliegue en una aplicación web

Se descarga el dataset de flores desde el enlace:
https://www.kaggle.com/datasets/imsparsh/flowers-dataset?resource=download

Se desarrolla un modelo de Red Neuronal Convolucional (CNN) para clasificar las imágenes de flores correctamente.
Posteriormente, se exporta el modelo a TensorFlow.js y se implementa una aplicación web que permite cargar una imagen y mostrar el nombre de la flor reconocida.

0. Estructura del Proyecto

```
CLASIFICADOR_FLORES/
│
├── data/                    # Carpeta principal de los datos
│   └── flowers/             # Dataset de flores organizado por clases
│       ├── daisy/           # Imágenes de margaritas
│       ├── dandelion/       # Imágenes de dientes de león
│       ├── rose/            # Imágenes de rosas
│       ├── sunflower/       # Imágenes de girasoles
│       └── tulip/           # Imágenes de tulipanes
│
├── modelo_flor_js/          # Modelo exportado para TensorFlow.js
│   ├── group1-shard1of4.bin   # Fragmento del modelo
│   ├── group1-shard2of4.bin   # Fragmento del modelo
│   ├── group1-shard3of4.bin   # Fragmento del modelo
│   ├── group1-shard4of4.bin   # Fragmento del modelo
│   └── model.json           # Configuración del modelo en JSON
│
├── test/                    # Carpeta opcional para imágenes de prueba
│
├── clasificador.js          # Lógica JavaScript para cargar y usar el modelo en la web
├── entrenar_modelo.py       # Script para crear, entrenar y exportar el modelo CNN
├── estilos.css              # Estilos visuales de la aplicación web
├── index.html               # Estructura de la página web
├── modelo_flor_cnn.h5       # Modelo entrenado guardado en formato .h5
├── preparar_dataset.py      # Script para preparar y cargar el dataset
├── requirements.txt         # Lista de librerías necesarias para el proyecto
├── Testing_set_flower.csv   # Archivo CSV posiblemente para pruebas adicionales (opcional)
└── video2.mp4               # Vídeo demostrativo del funcionamiento de la aplicación
```

***Nota**: los códigos completos del Ejercicio 2 están en su respectiva fase, si los quieres completos están en el repositorio de github.*

1. Preparar el dataset

Se crea un archivo llamado **preparar_dataset.py**, que realiza las siguientes tareas:

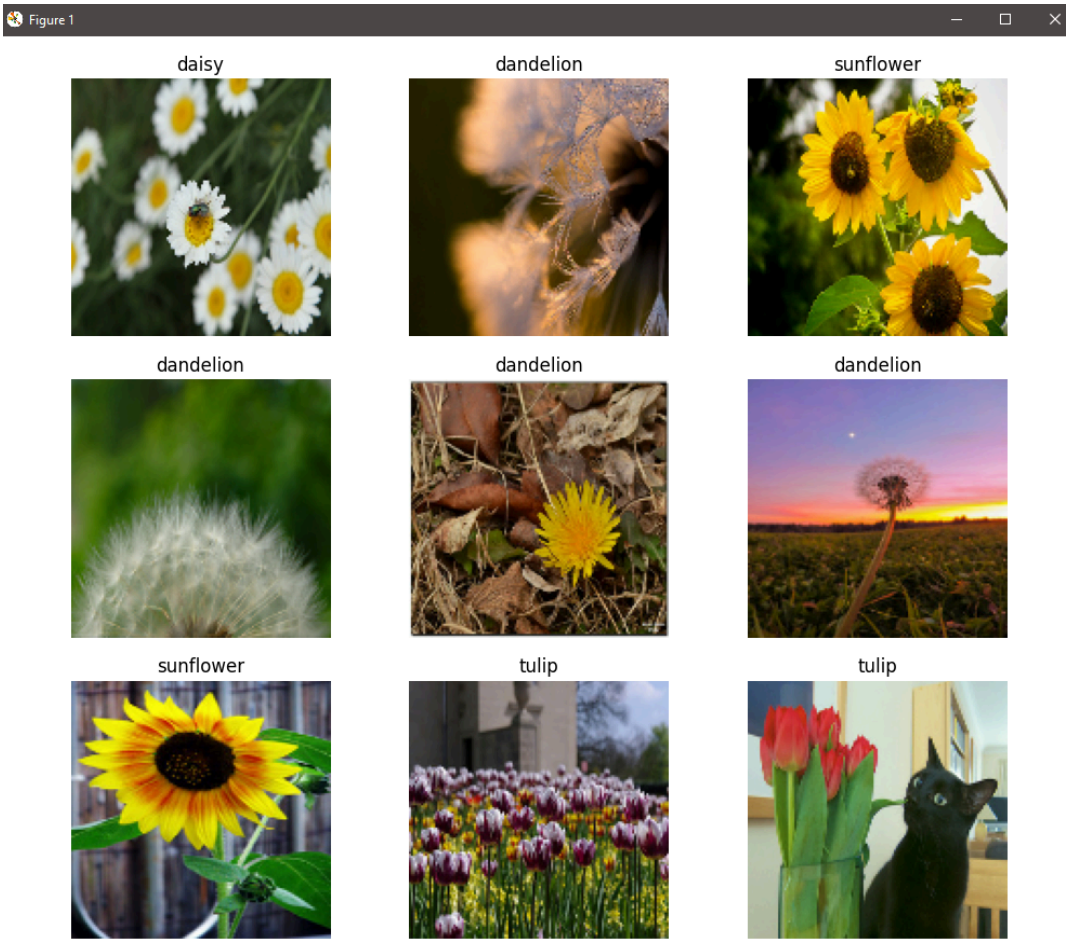
- ➔ Instala las dependencias necesarias.
- ➔ Carga las imágenes del dataset con un 80% de datos para entrenamiento y 20% para validación.
- ➔ Normaliza las imágenes (valores entre 0 y 1).
- ➔ Finalmente, se visualiza 9 imágenes de muestra del dataset.

Código completo:

```
# ------------------- Sección 1: Instalación de dependencias -------------------
import subprocess
import sys

# Función para instalar las dependencias listadas en 'requirements.txt'
def instalar_dependencias():
    try:
        # Ejecutar el comando de instalación
            subprocess.check_call([sys.executable, "-m", "pip", "install", "-r",
"requirements.txt"])
        print("Dependencias instaladas correctamente.")
    except subprocess.CalledProcessError:
        print("Error al instalar las dependencias.")

# Llamar a la función para instalar dependencias automáticamente al inicio
instalar_dependencias()

# ------------------- Sección 2: Importación de librerías -------------------

import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.utils import image_dataset_from_directory

# ------------------- Sección 3: Configuración inicial -------------------

# Ruta donde se encuentra el dataset de flores
DATASET_DIR = 'data/flowers'

# Parámetros de configuración
IMG_SIZE = (128, 128)     # Tamaño al que se redimensionarán las imágenes
BATCH_SIZE = 32           # Número de imágenes por lote (batch)
SEED = 123                # Semilla para reproducibilidad

#  ------------------- Sección 4: Carga de los datasets de entrenamiento y
validación -------------------
```

```python
# Cargar el dataset de entrenamiento (80% del total)
dataset_entrenamiento = image_dataset_from_directory(
    DATASET_DIR,
    validation_split=0.2,    # Reservar el 20% para validación
    subset="training",       # Subconjunto de entrenamiento
    seed=SEED,               # Semilla para reproducibilidad
    image_size=IMG_SIZE,     # Redimensionar imágenes
    batch_size=BATCH_SIZE    # Tamaño de batch
)

# Cargar el dataset de validación (20% del total)
dataset_validacion = image_dataset_from_directory(
    DATASET_DIR,
    validation_split=0.2,    # Mismo porcentaje para validación
    subset="validation",     # Subconjunto de validación
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE
)

# ------------------- Sección 5: Visualización de clases -------------------

# Obtener y mostrar los nombres de las clases detectadas en el dataset
clases = dataset_entrenamiento.class_names
print(f"Clases detectadas: {clases}")

# ------------------- Sección 6: Normalización de imágenes -------------------

# Crear una capa de normalización para escalar los píxeles de [0,255] a [0,1]
normalizar = tf.keras.layers.Rescaling(1./255)

# Aplicar la normalización tanto al conjunto de entrenamiento como de validación
dataset_entrenamiento = dataset_entrenamiento.map(lambda x, y: (normalizar(x), y))
dataset_validacion = dataset_validacion.map(lambda x, y: (normalizar(x), y))

# ------------------- Sección 7: Visualización de algunas muestras del dataset
# Función para mostrar 9 imágenes de muestra del dataset junto a su etiqueta
def mostrar_muestras(dataset, clases):
    plt.figure(figsize=(10, 8))
    for images, labels in dataset.take(1): # Tomar un batch de imágenes
        for i in range(9): # Mostrar las primeras 9 imágenes
            ax = plt.subplot(3, 3, i + 1)
            plt.imshow(images[i].numpy())    # Mostrar la imagen
            plt.title(clases[labels[i]])     # Mostrar la etiqueta correspondiente
            plt.axis("off")                  # Ocultar ejes
    plt.tight_layout()
    plt.show()

# Llamar a la función para visualizar ejemplos
mostrar_muestras(dataset_entrenamiento, clases)
```

Resultado:



2. Entrenar el modelo CNN

Se crea el archivo **entrenar_modelo.py** que:

- ➔ Carga el dataset previamente preparado.
- ➔ Define una arquitectura CNN.
- ➔ **Entrena el modelo.**
- ➔ **Guarda el modelo en formato .h5.**
- ➔ **Exporta el modelo a TensorFlow.js.**

Código completo:

```
# -------------------- Sección 1: Instalación de dependencias --------------------

import subprocess
import sys
import tensorflow as tf
import matplotlib.pyplot as plt

# Función para instalar las dependencias listadas en 'requirements.txt'
def instalar_dependencias():
    try:
            subprocess.check_call([sys.executable, "-m", "pip", "install", "-r",
"requirements.txt"])
        print("Dependencias instaladas correctamente.")
    except subprocess.CalledProcessError:
        print("Error al instalar las dependencias.")

# Llamar a la función para instalar dependencias automáticamente al inicio
instalar_dependencias()

# -------------------- Sección 2: Importación de librerías adicionales
--------------------

from tensorflow.keras.utils import image_dataset_from_directory
import tensorflowjs as tfjs

# -------------------- Sección 3: Configuración del dataset --------------------

# Ruta donde se encuentra el dataset de flores
DATASET_DIR = 'data/flowers'

# Parámetros de configuración
IMG_SIZE = (128, 128)    # Tamaño de las imágenes
BATCH_SIZE = 32          # Tamaño del batch

# -------------------- Sección 4: Carga de datasets de entrenamiento y validación
--------------------

# Cargar el dataset de entrenamiento (80% del total)
dataset_entrenamiento = image_dataset_from_directory(
    DATASET_DIR,
    validation_split=0.2,    # Reservar 20% para validación
    subset="training",       # Subconjunto de entrenamiento
    seed=123,                # Semilla para reproducibilidad
    image_size=IMG_SIZE,     # Redimensionar imágenes
    batch_size=BATCH_SIZE
)

# Cargar el dataset de validación (20% del total)
```

```
dataset_validacion = image_dataset_from_directory(
    DATASET_DIR,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE
)

# ------------------- Sección 5: Normalización de imágenes -------------------

# Crear una capa de normalización para escalar los píxeles de [0,255] a [0,1]
normalizar = tf.keras.layers.Rescaling(1./255)

# Aplicar normalización a los conjuntos de datos
dataset_entrenamiento = dataset_entrenamiento.map(lambda x, y: (normalizar(x), y))
dataset_validacion = dataset_validacion.map(lambda x, y: (normalizar(x), y))

# ------------------- Sección 6: Definición del modelo CNN -------------------

# Crear el modelo de red neuronal convolucional
modelo = tf.keras.Sequential([
     tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128,
3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),     # Capa de dropout para reducir overfitting
    tf.keras.layers.Dense(5, activation='softmax')  # Capa de salida para 5 clases
])

# ------------------- Sección 7: Compilación del modelo -------------------

# Compilar el modelo especificando el optimizador y la función de pérdida
modelo.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# ------------------- Sección 8: Entrenamiento del modelo -------------------

# Entrenar el modelo utilizando los datasets de entrenamiento y validación
modelo.fit(
    dataset_entrenamiento,
    epochs=10,
```

```
    validation_data=dataset_validacion
)

# -------------------- Sección 9: Guardado del modelo entrenado --------------------

# Guardar el modelo entrenado en formato .h5
modelo.save('modelo_flor_cnn.h5')

#   -------------------   Sección   10:   Exportación   del   modelo   a   TensorFlow.js
-------------------

# Exportar el modelo en formato compatible con TensorFlow.js
tfjs.converters.save_keras_model(modelo, 'modelo_flor_js')

# -------------------- Sección 11: Mensaje final -------------------

# Mostrar mensaje de éxito
print("Modelo entrenado y exportado a TensorFlow.js correctamente.")
```

3. Aplicación Web (Frontend)

La aplicación web consiste en:

- ➔ HTML para la estructura.
- ➔ CSS para el estilo con temática de flores (colores verde-amarillo).
- ➔ JavaScript para cargar el modelo y hacer predicciones.

Archivo: clasificador.js

```
// ------------------- Sección 1: Definición de variables -------------------

// Variable global para almacenar el modelo cargado
let modelo;

// ------------------- Sección 2: Función para cargar el modelo -------------------

async function cargarModelo() {
  // Cargar el modelo exportado en formato TensorFlow.js
  modelo = await tf.loadLayersModel('modelo_flor_js/model.json');
  console.log("Modelo cargado correctamente");
}

// ------------------- Sección 3: Función para predecir la clase de la imagen
-------------------

async function predecir() {
  // Verificar si el modelo ya está cargado
  if (!modelo) {
    document.getElementById("resultado").innerText = "Cargando el modelo, por favor
espere...";
    return;
  }

  // Mostrar mensaje de carga mientras se realiza la predicción
  document.getElementById("resultado").innerText = "Cargando...";

  // Obtener la imagen cargada por el usuario
  const imagen = document.getElementById("imagenSeleccionada");

  // Preprocesar la imagen para que tenga el formato adecuado para el modelo
  const tensorImagen = tf.browser.fromPixels(imagen)
    .resizeNearestNeighbor([128, 128]) // Redimensionar a 128x128 píxeles
    .toFloat()                         // Convertir a tipo float
    .expandDims(0)                     // Añadir dimensión extra para el batch
    .div(tf.scalar(255));              // Normalizar valores de píxel entre 0 y 1

  // Realizar la predicción
```

```
  const prediccion = modelo.predict(tensorImagen);
  const prediccionArray = await prediccion.data();

  // Determinar la clase con mayor probabilidad
  const clasePredicha = prediccionArray.indexOf(Math.max(...prediccionArray));

  // Definición de las clases en el mismo orden en que el modelo fue entrenado
  const clases = ["Daisy", "Dandelion", "Rose", "Sunflower", "Tulip"];

  // Mostrar el resultado de la predicción en la página
        document.getElementById("resultado").innerText = `La flor es:
${clases[clasePredicha]}`;
}

// ------------------- Sección 4: Eventos de interacción con el usuario
-------------------

// Asociar la función de predicción al botón
document.getElementById("predecirBtn").onclick = predecir;

// Manejar el evento de selección de imagen
document.getElementById("imagenInput").onchange = (e) => {
  const file = e.target.files[0];

  // Verificar que el archivo seleccionado sea una imagen
  const fileType = file.type;
  if (!fileType.startsWith("image/")) {
    alert("Por favor selecciona una imagen.");
    return;
  }

  // Cargar y mostrar la imagen seleccionada
  if (file) {
    const reader = new FileReader();
    reader.onload = (event) => {
      const img = document.getElementById("imagenSeleccionada");
      img.src = event.target.result;
    };
    reader.readAsDataURL(file);
  }
};

// ------------------- Sección 5: Llamada inicial para cargar el modelo
-------------------

// Cargar el modelo automáticamente cuando se carga la página
cargarModelo();
```

Archivo: estilos.css

```
/* ------------------- Sección 1: Estilos generales del cuerpo -------------------
*/

body {
  font-family: Arial, sans-serif;
  background-color: #f0fff0; /* Fondo verde muy claro (Honeydew) */
  color: #2e8b57; /* Texto verde (SeaGreen) */
  text-align: center;
  margin: 0;
  padding: 0;
}

/* ------------------- Sección 2: Contenedor principal ------------------- */

.container {
  max-width: 600px;
  margin: 50px auto;
  padding: 20px;
  background-color: #e6ffe6; /* Verde pastel */
  border-radius: 8px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}

/* ------------------- Sección 3: Títulos ------------------- */

h1 {
  color: #228b22; /* Verde Bosque */
  animation: fadeIn 2s ease-in;
}

/* ------------------- Sección 4: Botones ------------------- */

button {
  background-color: #9acd32; /* Amarillo verdoso (YellowGreen) */
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  font-size: 18px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

button:hover {
  background-color: #6b8e23; /* OliveDrab (verde más oscuro) */
}

/* ------------------- Sección 5: Input de archivos ------------------- */
```

```css
input[type="file"] {
  margin: 20px 0;
  padding: 10px;
  border-radius: 5px;
  border: 1px solid #228b22; /* Mismo verde bosque que el título */
}

/* ------------------- Sección 6: Imagen de la flor ------------------- */

.imagen-flor {
  width: 250px;
  height: 250px;
  margin-top: 20px;
  border-radius: 8px;
  animation: fadeIn 2s ease-in-out;
}

/* ------------------- Sección 7: Resultado de la predicción ------------------- */

#resultado {
  margin-top: 20px;
  font-size: 20px;
  font-weight: bold;
  color: #556b2f; /* DarkOliveGreen para resaltar */
  animation: slideIn 1s ease-in;
}

/* ------------------- Sección 8: Animaciones ------------------- */

/* Animación de aparición suave */
@keyframes fadeIn {
  0% { opacity: 0; }
  100% { opacity: 1; }
}

/* Animación de entrada desde abajo */
@keyframes slideIn {
  0% { transform: translateY(50px); opacity: 0; }
  100% { transform: translateY(0); opacity: 1; }
}
```

Archivo: Index.html

```
<!-- ------------------- Sección 1: Definición de la estructura básica del documento ------------------- -->
<!DOCTYPE html>
<html lang="es">

<!-- ------------------- Sección 2: Encabezado de la página ------------------- -->
<head>
  <meta charset="UTF-8"> <!-- Establece la codificación de caracteres a UTF-8 -->
    <meta  name="viewport"  content="width=device-width,  initial-scale=1.0">  <!-- Permite una correcta visualización en móviles -->
  <title>Clasificación de Flores</title> <!-- Título de la página que aparece en la pestaña del navegador -->

  <!-- Importar TensorFlow.js desde CDN -->
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@4.22.0"></script>

  <!-- Vincular hoja de estilos externa -->
  <link rel="stylesheet" href="estilos.css">
</head>

<!-- ------------------- Sección 3: Cuerpo de la página ------------------- -->
<body>
  <div class="container"> <!-- Contenedor principal -->

    <h1>Clasificación de Flores</h1> <!-- Título principal en la página -->

    <!-- Entrada de archivo para que el usuario suba una imagen -->
    <input type="file" id="imagenInput" accept="image/*">

    <!-- Botón que activará la predicción -->
    <button id="predecirBtn">Predecir Flor</button>

    <!-- Párrafo donde se mostrará el resultado de la predicción -->
    <p id="resultado"></p>

    <!-- Imagen seleccionada por el usuario que será mostrada -->
      <img  id="imagenSeleccionada"  src=""  alt="Imagen  de  la  flor  seleccionada" class="imagen-flor">

  </div>

  <!-- Importar archivo JavaScript que contiene la lógica del clasificador -->
  <script src="clasificador.js"></script>
</body>
</html>
```

Resultado Web

*Para visualizar la página web en funcionamiento, esta el **video2.mp4** (dentro de la carpeta del ejercicio) donde se realizan varias pruebas del funcionamiento de la app.*

Conclusión

Se ha implementado un **modelo CNN** capaz de reconocer distintas clases de flores a partir de imágenes.

Se ha **exportado a TensorFlow.js** y se ha integrado en una **aplicación web interactiva** que permite al usuario seleccionar una imagen y obtener el nombre de la flor correspondiente.

Enlaces y referencias

Github: Archivos completos
https://github.com/IvanFalconMonzon/TA_MODELOS-DE-PYTHON-A-TENSORFLOW.JS_IVANFALCONMONZON.git

¿Cómo se configura un servidor de prueba local?
https://developer.mozilla.org/es/docs/Learn_web_development/Howto/Tools_and_setup/set_up_a_local_testing_server

Flowers Dataset:
https://www.kaggle.com/datasets/imsparsh/flowers-dataset?resource=download

Usa tus modelos de Tensorflow en páginas web | Exportación a Tensorflow.js:
https://youtu.be/JpE4bYyRADI?si=TneF91r7DnpuZYIM

Machine Learning For Front-End Developers With Tensorflow.js
https://www.smashingmagazine.com/2019/09/machine-learning-front-end-developers-tensorflowjs/?utm_campaign=machine-learning-for-frontend-developers

Anexo 1 (Adicional)

Index.html

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Conversión Fahrenheit a Celsius</title>

  <!-- Importa la librería de TensorFlow.js -->
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@4.22.0"></script>

  <!-- Enlace a archivo CSS externo -->
  <link rel="stylesheet" href="estilos.css">
</head>
<body>
  <h1>Conversión de Temperatura - Fahrenheit a Celsius</h1>

  <!-- Campo de entrada para el valor en Fahrenheit -->
  <input type="number" id="fahrenheitInput" placeholder="Fahrenheit °F">

  <!-- Botón que ejecuta la conversión al hacer clic -->
  <button onclick="predecir()">Convertir</button>

  <!-- Elemento donde se mostrará el resultado -->
  <p id="resultado"></p>
```

```
  <!-- Enlace al archivo JavaScript externo que contiene la lógica de conversión
-->
  <script src="conversion.js"></script>
</body>
</html>
```

estilos.css

```
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(to right, #ff4e50, #f9d423);
    color: #fff;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
    margin: 0;
    transition: background 0.5s ease-in-out;
  }

  h1 {
    font-size: 2.5rem;
    margin-bottom: 30px;
    text-shadow: 2px 2px 8px rgba(0,0,0,0.3);
  }

  input[type="number"] {
    padding: 10px;
    border: none;
    border-radius: 10px;
    width: 200px;
    font-size: 1rem;
    margin-bottom: 20px;
    box-shadow: 0 0 10px rgba(255, 255, 255, 0.3);
    transition: transform 0.3s ease;
  }

  input[type="number"]:focus {
    outline: none;
    transform: scale(1.05);
  }

  button {
    padding: 10px 20px;
    background-color: #c0392b;
    color: #fff;
    border: none;
    border-radius: 12px;
```

```
    font-size: 1rem;
    cursor: pointer;
    transition: all 0.3s ease;
    box-shadow: 0 4px 15px rgba(192, 57, 43, 0.4);
  }

  button:hover {
    background-color: #e74c3c;
    transform: scale(1.1);
    box-shadow: 0 6px 20px rgba(231, 76, 60, 0.6);
  }

  #resultado {
    margin-top: 20px;
    font-size: 1.3rem;
    padding: 10px 20px;
    border-radius: 10px;
    background: rgba(255, 255, 255, 0.1);
    box-shadow: 0 0 10px rgba(255,255,255,0.2);
    transition: all 0.3s ease;
  }
```

entrenar_modelo.py

```
import os
import subprocess
import sys

#    --------------------    Sección    1:    Instalación    de    paquetes    requeridos
--------------------

# Lista de paquetes necesarios para ejecutar el script
requerimientos = [
    "tensorflow==2.11.0",
    "tensorflowjs==3.18.0",
    "numpy==1.23.5",
    "ipykernel",
    "matplotlib",
    "pandas",
    "scikit-learn"
]

# Función para instalar los paquetes que no estén disponibles en el entorno
def instalar_paquetes():
    for paquete in requerimientos:
        try:
                subprocess.run([sys.executable, "-m", "pip", "install", paquete],
check=True)
        except subprocess.CalledProcessError:
            print(f"Error instalando {paquete}")
```

```
# Ejecutar la función para instalar los paquetes
instalar_paquetes()

# ------------------- Sección 2: Importación de librerías -------------------

import tensorflow as tf
import numpy as np
import tensorflowjs as tfjs
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# ------------------- Sección 3: Generación de datos -------------------

# Crear una serie de valores en Fahrenheit entre -100 y 212 grados
fahrenheit = np.linspace(-100, 212, 1000, dtype=np.float32)

# Convertir esos valores a Celsius usando la fórmula de conversión
celsius = (fahrenheit - 32) * 5 / 9

# Crear un DataFrame y guardarlo como archivo CSV
data = pd.DataFrame({'Fahrenheit': fahrenheit, 'Celsius': celsius})
data.to_csv('dataset_temperaturas.csv', index=False)
print("Dataset guardado como 'dataset_temperaturas.csv'")

# ------------------- Sección 4: División de datos -------------------

# Dividir los datos en entrenamiento+validación (80%) y prueba (20%)
X_train_val, X_test, y_train_val, y_test = train_test_split(
    fahrenheit, celsius, test_size=0.2, random_state=42
)

# Dividir el conjunto de entrenamiento+validación en entrenamiento (95%) y
validación (5%)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.05, random_state=42
)

# ------------------- Sección 5: Creación y entrenamiento del modelo
-------------------

# Crear un modelo secuencial con una capa oculta y una capa de salida
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(1,)),          # Entrada de un valor (temperatura)
    tf.keras.layers.Dense(64, activation='relu'), # Capa oculta con 64 neuronas
     tf.keras.layers.Dense(1)                        # Capa de salida con 1 neurona
(temperatura en Celsius)
])
```

```
# Compilar el modelo especificando el optimizador y la función de pérdida
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Entrenar el modelo con los datos de entrenamiento y validación
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=200,
    verbose=1
)

# ------------------- Sección 6: Visualización de entrenamiento -------------------

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

# Gráfico de la pérdida (MSE)
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title("Pérdida (MSE)")
plt.xlabel("Épocas")
plt.ylabel("Error Cuadrático Medio")
plt.legend()

# Gráfico del error absoluto medio (MAE)
plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Entrenamiento')
plt.plot(history.history['val_mae'], label='Validación')
plt.title("Error Absoluto Medio (MAE)")
plt.xlabel("Épocas")
plt.ylabel("MAE")
plt.legend()

# Mostrar los gráficos
plt.tight_layout()
plt.show()

# ------------------- Sección 7: Exportación del modelo a TensorFlow.js
-------------------
# Ruta de salida para el modelo exportado
export_dir = 'modelo_temperatura_js'

# Eliminar carpeta existente si ya hay un modelo previo
if os.path.exists(export_dir):
    import shutil
    shutil.rmtree(export_dir)

# Exportar el modelo entrenado en formato compatible con TensorFlow.js
tfjs.converters.save_keras_model(model, export_dir)
print("Modelo exportado a TensorFlow.js en:", export_dir)
```

requirements.txt

```
tensorflow==2.11.0
tensorflowjs==3.18.0
numpy==1.23.5
ipykernel
matplotlib
pandas
scikit-learn
```

conversion.js

```
// conversion.js

// Variable global que almacenará el modelo cargado
let modelo;

// Carga el modelo de TensorFlow.js desde una URL local (carpeta modelo_temperatura_js)
async function cargarModelo() {
  try {
    modelo = await tf.loadLayersModel('modelo_temperatura_js/model.json');
    console.log("Modelo cargado correctamente");
  } catch (error) {
    console.error("Error cargando modelo:", error);
    // Mostrar error en el elemento HTML con id 'resultado'
      document.getElementById("resultado").innerText = "Error cargando modelo: " + error.message;
  }
}

// Función que se ejecuta al hacer clic en el botón "Convertir"
async function predecir() {
  if (!modelo) {
    // Si el modelo aún no se ha cargado, mostrar mensaje de advertencia
    document.getElementById("resultado").innerText = "Modelo no cargado aún.";
    return;
  }

  // Obtener el valor introducido por el usuario
  const input = parseFloat(document.getElementById("fahrenheitInput").value);

  // Validar que el valor ingresado sea un número
  if (isNaN(input)) {
      document.getElementById("resultado").innerText = "Ingrese un valor numérico válido.";
    return;
  }
```

```
  // Crear un tensor a partir del valor ingresado
  const tensorEntrada = tf.tensor2d([input], [1, 1]);

  // Usar el modelo para hacer una predicción
  const tensorSalida = modelo.predict(tensorEntrada);

  // Extraer el resultado del tensor
  const resultado = (await tensorSalida.data())[0];

  // Mostrar el resultado en la página
              document.getElementById("resultado").innerText       =       `Celsius:
${resultado.toFixed(2)} °C`;
}

// Llamar automáticamente a la función para cargar el modelo al inicio
cargarModelo();
```