MODELOS DE PYTHON A TENSORFLOW.JS

Adrián Yared Armas de la Nuez

Contenido

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())
```

```
    fahrenheit    celsius
0    12.362036 -10.909980
1   185.214292  85.119051
2   119.598183  48.665657
3    79.597545  26.443081
4   -53.194408 -47.330227
```

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}")
```

```
Test Loss: 3.5790, Test MAE: 1.1204
```

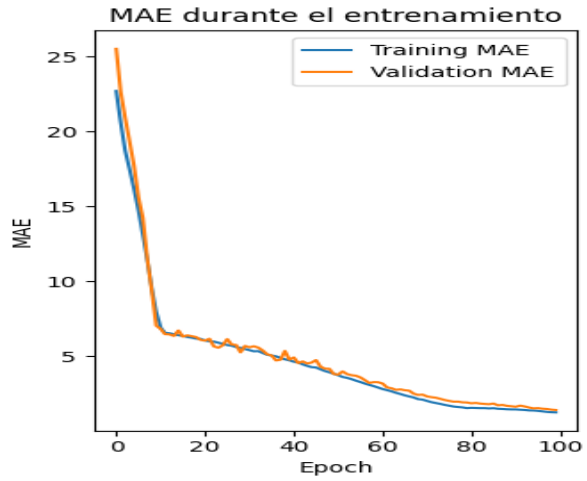
1.2.8 Guardado del modelo

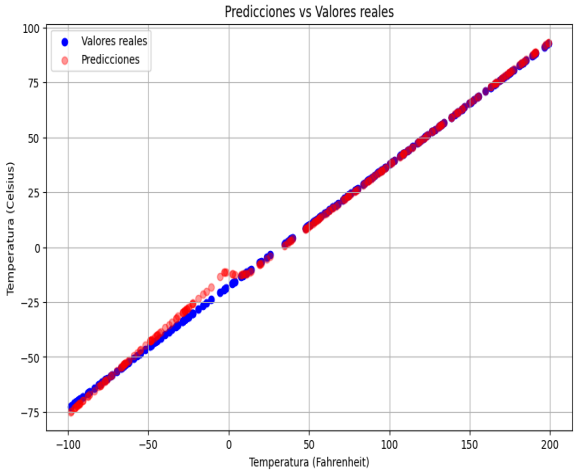
```
# Guardamos el modelo entrenado y visualizamos las curvas de pérdida y
MAE durante el entrenamiento
model.save('temperature_model.keras')
```

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"
/>
```

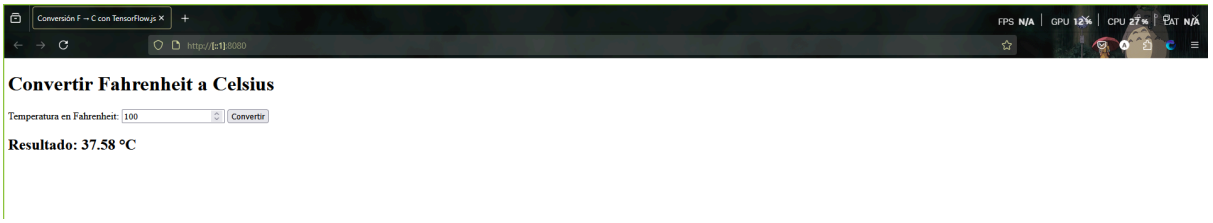
```
  <title>Conversión F → C con TensorFlow.js</title>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
</head>
<body>
  <main>
    <h1>Convertir Fahrenheit a Celsius</h1>

    <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:

```
PS C:\Users\adria\Desktop\Flores> cd .\temperatura\
PS C:\Users\adria\Desktop\Flores\temperatura> python -m http.server 8080
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
::1 - - [28/May/2025 01:41:03] "GET / HTTP/1.1" 200 -
::1 - - [28/May/2025 01:41:04] "GET /model.json HTTP/1.1" 200 -
::1 - - [28/May/2025 01:41:04] code 404, message File not found
::1 - - [28/May/2025 01:41:04] "GET /favicon.ico HTTP/1.1" 404 -
::1 - - [28/May/2025 01:41:04] "GET /group1-shard1of1.bin HTTP/1.1" 200 -
```

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")
```

```
Found 2746 files belonging to 5 classes.
Using 2197 files for training.
Found 2746 files belonging to 5 classes.
Using 549 files for 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}")
```

```
✅ Modelo guardado como modelo_flores.h5
```

2.2.8 Puesta en producción

Html:

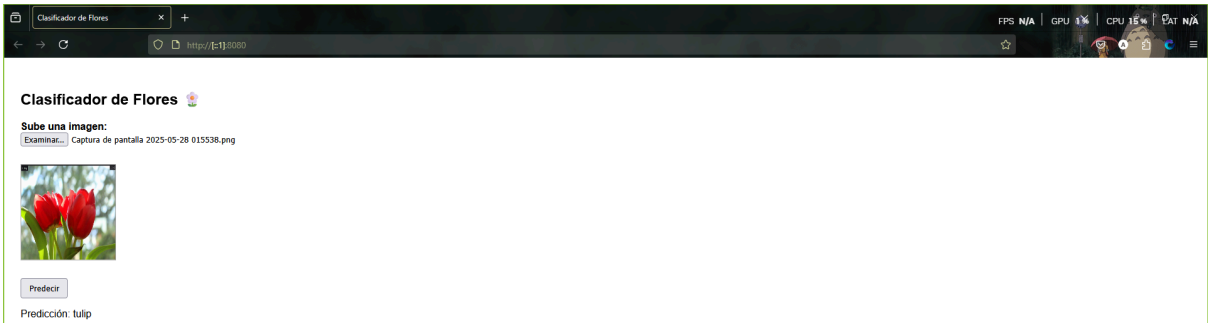
```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1"
/>
  <title>Clasificador de Flores</title>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
  <style>
    body {
      font-family: sans-serif;
      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:

```
PS C:\Users\adria\Desktop\Flores\flores> python -m http.server 8080
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
```

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();
```