GUÍA PRÁCTICA – TENSORFLOWJS

1. Crear y entrenar una red convolucional en Google Colab

Paso 1.1: Abre Google Colab y prepara el entorno

python

CopiarEditar

```
!pip install tensorflow tensorflowjs
```

Paso 1.2: Importa librerías y carga los datos

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalizar
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

Paso 1.3: Crear un modelo convolucional simple

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])


model.summary()
```

Paso 1.4: Entrenar el modelo

```
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

2. Exportar el modelo para TensorFlow.js

Paso 2.1: Guardar el modelo Keras

```
model.save("modelo_mnist")
```

Paso 2.2: Convertir el modelo a formato TensorFlow.js

```
!tensorflowjs_converter --input_format=tf_saved_model --output_format=tfjs_graph_model
./modelo_mnist ./modelo_tfjs
```

Esto generará un directorio modelo_tfjs con dos archivos clave:

- model.json
- group1-shard1of1.bin

Paso 2.3: Descargar los archivos

```
from google.colab import files

import zipfile

import os


# Comprimir carpeta

def zip_folder(folder_path, zip_name):

    with zipfile.ZipFile(zip_name, 'w') as zipf:

        for root, dirs, files_in_dir in os.walk(folder_path):

            for file in files_in_dir:

                filepath = os.path.join(root, file)
```

```
            zipf.write(filepath, os.path.relpath(filepath, folder_path))

zip_folder("modelo_tfjs", "modelo_tfjs.zip")
files.download("modelo_tfjs.zip")
```

3. Crear una página web básica con el modelo exportado

Estructura de carpetas

```
/web_app
 ├── index.html
 ├── script.js
 └── modelo_tfjs/
     ├── model.json
     └── group1-shard1of1.bin
```

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Clasificador MNIST</title>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
</head>
<body>
  <h1>Clasificador de Dígitos</h1>
  <canvas id="canvas" width="280" height="280" style="border:1px solid
black;"></canvas><br>
  <button onclick="predecir()">Predecir</button>
  <p id="resultado">Dibuja un número y presiona predecir</p>

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

```
</html>
```

script.js

```
let model;
async function cargarModelo() {
  model = await tf.loadGraphModel('modelo_tfjs/model.json');
  console.log("Modelo cargado");
}
cargarModelo();

// Canvas
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');
ctx.fillStyle = 'black';
ctx.fillRect(0, 0, canvas.width, canvas.height);

let pintando = false;

canvas.addEventListener('mousedown', () => { pintando = true; });
canvas.addEventListener('mouseup', () => { pintando = false; ctx.beginPath(); });
canvas.addEventListener('mousemove', dibujar);

function dibujar(e) {
  if (!pintando) return;
  ctx.lineWidth = 15;
  ctx.lineCap = 'round';
  ctx.strokeStyle = 'white';
  ctx.lineTo(e.offsetX, e.offsetY);
  ctx.stroke();
  ctx.beginPath();
  ctx.moveTo(e.offsetX, e.offsetY);
```

```
}


function predecir() {
  let imgData = ctx.getImageData(0, 0, 280, 280);
  let tensor = tf.browser.fromPixels(imgData, 1)
    .resizeNearestNeighbor([28, 28])
    .mean(2)
    .toFloat()
    .div(255.0)
    .reshape([1, 28, 28, 1]);


  model.executeAsync(tensor).then(pred => {
    const prediccion = pred.arraySync()[0];
    const clase = prediccion.indexOf(Math.max(...prediccion));
    document.getElementById('resultado').innerText = "Predicción: " + clase;
  });
}
```

Resultado final

Tendrás una web donde puedes **dibujar un número**, presionar "Predecir" y ver la salida usando el modelo que entrenaste en Colab.

Crea una mini aplicación web que use un modelo de TensorFlow.js

Objetivo:

Desarrollar una página web interactiva que utilice un modelo de **TensorFlow.js** para hacer una predicción o clasificación en tiempo real (imagen, texto o datos simples).

Instrucciones generales:

1. **Elige una de estas 3 opciones:**

Opción	Proyecto sugerido
A	Clasificador de dibujos de números usando el elemento HTML canvas (modelo MNIST)
B	Clasificador de emojis entrenado con Teachable Machine (usando webcam)
C	Detector de estado de ánimo con texto (modelo entrenado con frases positivas y negativas)

2. **Requisitos del proyecto:**
    - Una página HTML (index.html)
    - Un archivo JS (script.js) que cargue el modelo con TensorFlow.js
    - Interfaz básica (canvas, webcam o input de texto)
    - Mostrar en pantalla la predicción del modelo
    - El modelo puede ser uno preentrenado (como los de Teachable Machine) o uno exportado desde Colab
3. **Documentación:**
    - Incluir un pequeño documento o archivo README:
        - ¿Qué hace tu app?
        - ¿Qué modelo usaste?
        - ¿Qué aprendiste?

Entrega esperada:

- Un archivo .zip con:
    - index.html
    - script.js
    - Carpeta del modelo (model.json, .bin)