Trabajo de Investigación y Aplicación Práctica: Integración de Modelos de Machine Learning en Aplicaciones Web

Enunciado:

Tu objetivo es **investigar, entrenar, exportar e integrar un modelo de Machine Learning** dentro de una aplicación web funcional. Utilizando **Teachable Machine de Google**, entrenarás un modelo de clasificación (puede ser de imágenes, sonidos o poses). Luego deberás **exportarlo** y utilizarlo en una página web mediante **TensorflowJS** y **JavaScript**, simulando una aplicación real que pueda ejecutarse en el navegador.

Requisitos del trabajo:

1. **Investigación Documentada**:
   - ¿Qué es Teachable Machine y cómo se entrena un modelo ahí?

     Teachable Machine: Es una herramienta web de Google que permite crear modelos de Machine Learning (clasificación de imágenes, sonidos o poses) de forma sencilla y sin código.

     Entrenamiento:

     1. Elegir proyecto: Imagen, Sonido o Postura.
     2. Recolectar muestras: Subir o capturar ejemplos para cada clase (categoría) que el modelo debe reconocer (ej., fotos de "Piedra", "Papel", "Tijera").
     3. Entrenar: Con un clic, Teachable Machine usa transfer learning sobre modelos pre-entrenados (como MobileNet) para aprender de tus muestras. El entrenamiento se hace en el navegador.
     4. Probar y exportar: Se puede previsualizar el modelo en tiempo real y luego exportarlo.

   - ¿Cómo se exporta un modelo para uso en web (TensorflowJS)?

     1. Desde la pestaña "Exportar modelo" en Teachable Machine.
     2. Seleccionar la pestaña "Tensorflow.js".
     3. Elegir "Descargar" para obtener un archivo .zip que contiene:
        - model.json: La arquitectura del modelo.

- weights.bin: Los pesos entrenados.
- metadata.json: Nombres de las clases y otra información.

4. Estos tres archivos se usan para cargar el modelo en una página web.

- ¿Qué es TensorflowJS y cómo se utiliza para cargar un modelo?

1. TensorflowJS (TFJS): Es una biblioteca de JavaScript para definir, entrenar y ejecutar modelos de Machine Learning en el navegador (cliente) o en Node.js (servidor). Permite usar la GPU para acelerar cálculos.
2. Uso para cargar un modelo (de Teachable Machine):
   - Incluir bibliotecas:
     <script
     src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest/dist/tf.min.js">
     </script>
     <script
     src="https://cdn.jsdelivr.net/npm/@teachablemachine/image@latest/dist/teachablemachine-image.min.js">
     </script> ` ` `
   - Cargar en JavaScript: Se usa una función de la biblioteca @teachablemachine (ej., tmImage.load()) que internamente utiliza TFJS.
     - const modelURL = "./ruta_al_modelo/model.json";
     - const metadataURL = "./ruta_al_modelo/metadata.json";
     - model = await tmImage.load(modelURL, metadataURL);

       // Luego se usa model.predict()

- Breve explicación de qué es .h5 y cómo se exporta un modelo en Python (Keras).

1. .h5 (o .keras): Es un formato de archivo usado para guardar modelos de Machine Learning entrenados, especialmente con Keras (una API de alto nivel para TensorFlow, JAX, PyTorch). Almacena la arquitectura del modelo, los pesos entrenados, la configuración del entrenamiento y el estado del optimizador.
2. Exportar en Keras (Python):

```
import tensorflow as tf

from tensorflow import keras

# (Suponiendo que 'model' es un modelo Keras ya entrenado)

# model = keras.models.Sequential([...])
```

```
# model.compile(...)

# model.fit(...)

model.save('mi_modelo_entrenado.h5')

# O el formato más nuevo:
model.save('mi_modelo_entrenado.keras')
```

- ¿Qué es GitHub y cómo se puede versionar un proyecto allí?

1. GitHub: Es una plataforma web para alojar repositorios de Git, facilitando el control de versiones y la colaboración en proyectos de software.
2. Versionar con Git/GitHub (flujo básico):
   - Crear Repositorio: En GitHub o localmente (git init).
   - Clonar (si es remoto): git clone <url_del_repositorio>.
   - Trabajar: Modificar archivos.
   - Añadir Cambios (Staging): git add . (para todos los cambios) o git add <archivo>.
   - Confirmar Cambios (Commit): git commit -m "Descripción de los cambios". Guarda una instantánea local.
   - Subir Cambios (Push): git push origin main (o la rama correspondiente). Envía los commits locales a GitHub.
   - Descargar Cambios (Pull): git pull origin main. Trae cambios desde GitHub al repositorio local.
   - Ramas (Branches): git branch <nombre_rama>, git checkout <nombre_rama>. Permiten trabajar en características de forma aislada.
   - Fusiones (Merge): git merge <nombre_rama>. Integra cambios de una rama a otra.
3. **Aplicación Práctica**:
   a. Entrena un modelo con Teachable Machine (por ejemplo: reconocimiento de objetos, sonidos o poses).
   b. Exporta el modelo en formato compatible con TensorflowJS.
   c. Desarrolla una página web (HTML + JS) que cargue e interprete el modelo usando TensorflowJS.
   d. Sube el proyecto a GitHub.
   e. Incluye una sección explicativa del modelo en la interfaz (qué hace, cómo se entrena, etc.).

Sugerencias técnicas:

- Usa vectores o matrices simuladas en Python para entender el entrenamiento y funcionamiento de modelos.

```python
import numpy as np
# Simular una imagen pequeña (ej: 3x3 píxeles en escala de grises)
imagen_simple = np.array([[0.1, 0.2, 0.1],
                          [0.4, 0.5, 0.4],
                          [0.1, 0.2, 0.1]])
# Aplanar para convertirla en un vector de entrada
vector_entrada = imagen_simple.flatten()
print("Vector de entrada para el modelo:", vector_entrada)

# Simular una capa densa simple con 2 neuronas de salida
pesos = np.random.rand(len(vector_entrada), 2) # Pesos aleatorios
sesgos = np.random.rand(2) # Sesgos aleatorios
salida_capa = np.dot(vector_entrada, pesos) + sesgos
print("Salida (antes de activación):", salida_capa)
```

- Agrega estilos simples para mejorar la presentación (CSS).
- Bonus: realiza una versión local del modelo usando Python + Keras y guarda el .h5.

Entrega:

- Link del repositorio en GitHub con tu proyecto.
- Documento PDF (máx. 5 páginas) con tu investigación y explicación del desarrollo.
- Video corto (máx. 2 min) mostrando tu aplicación funcionando.