

Tarea evaluable

CE_5073 5.1

Programación de inteligencia artificial



David Ramírez Ruiz

16/02/24

Índice

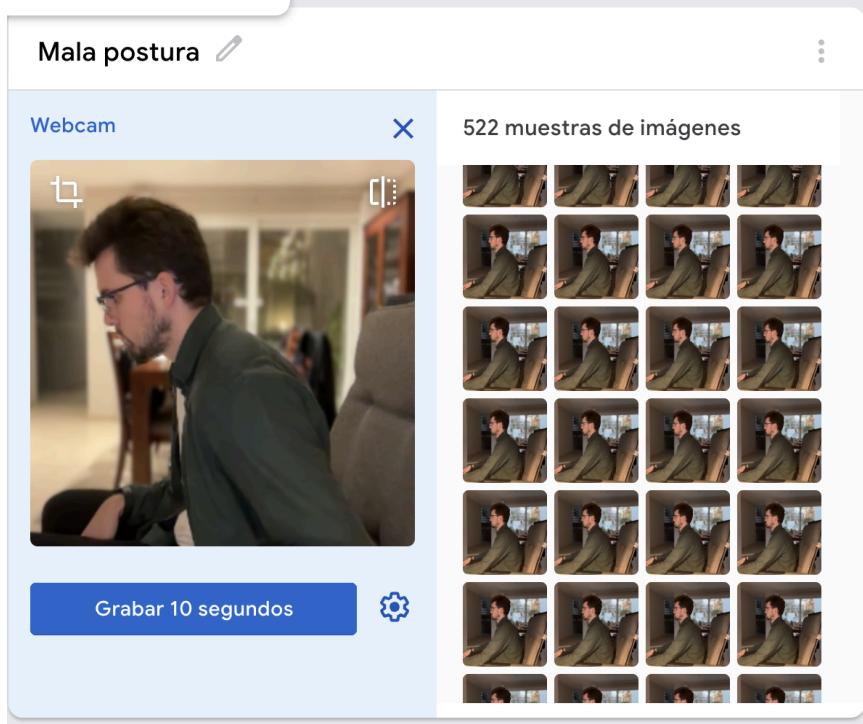
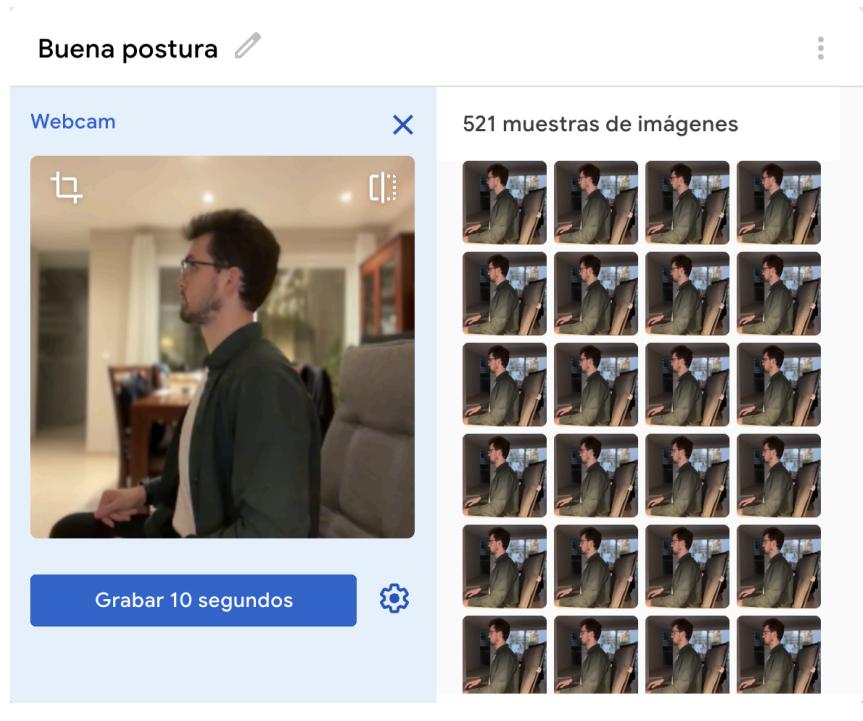
1 - PARTE A - TEACHABLE MACHINE	2
1.1 Preparación de archivos y subida a T.M.	2
1.2 Entrenamiento del modelo	3
1.3 Exportación del modelo	4
1.4 Publicación de la página web	4
2 - PARTE B - ASISTENTE DE PROGRAMACIÓN	6
2.1 Explicación del asistente	6

1 - PARTE A - TEACHABLE MACHINE

1.1 Preparación de archivos y subida a T.M.

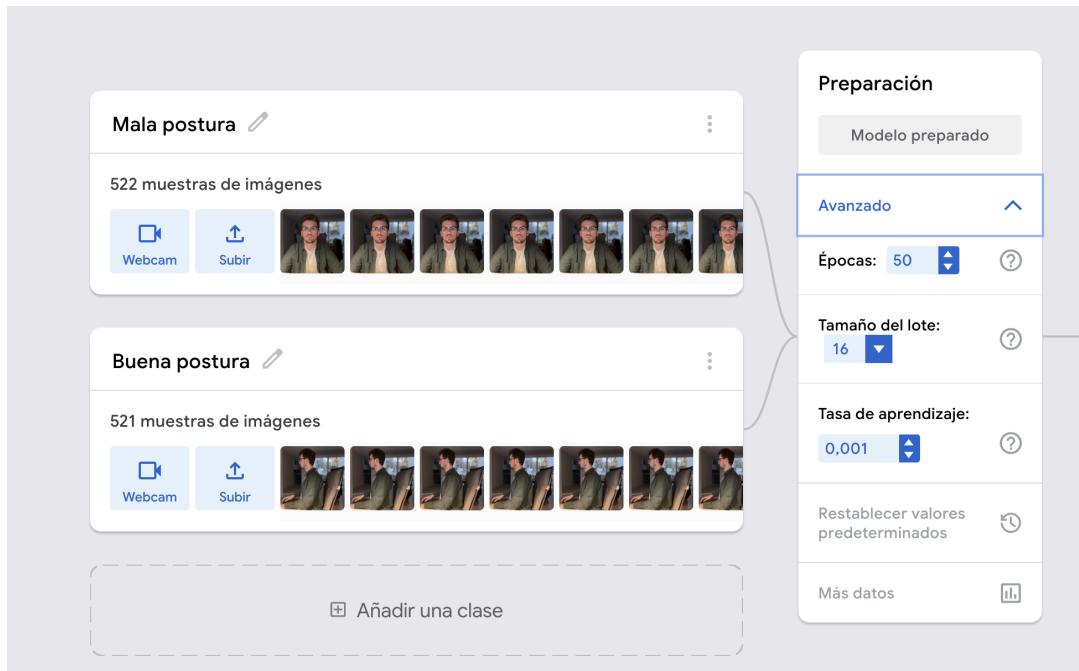
He elegido crear un modelo basado en imágenes, que a través de la webcam, identifica si tu postura es correcta o no.

El primer paso ha sido hacer las fotos de lo que es una buena postura y una mala postura.



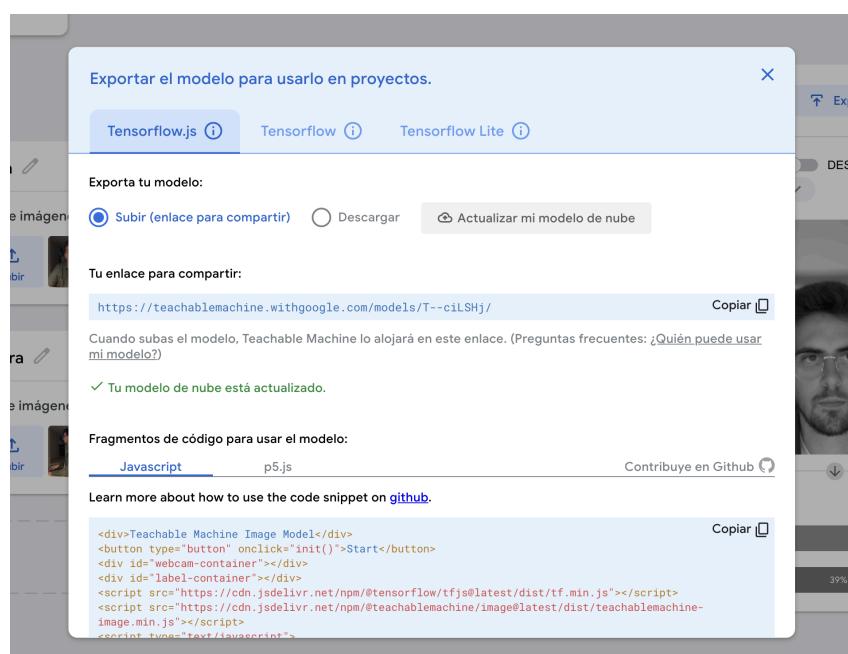
1.2 Entrenamiento del modelo

Lo siguiente ha sido entrenar al modelo a partir de los archivos generados.



1.3 Exportación del modelo

Una vez el modelo se ha exportado, se sube a la nube para que esté accesible desde el código. Una vez hecho esto se ha usado el código HTML generado por Teachable Machine.



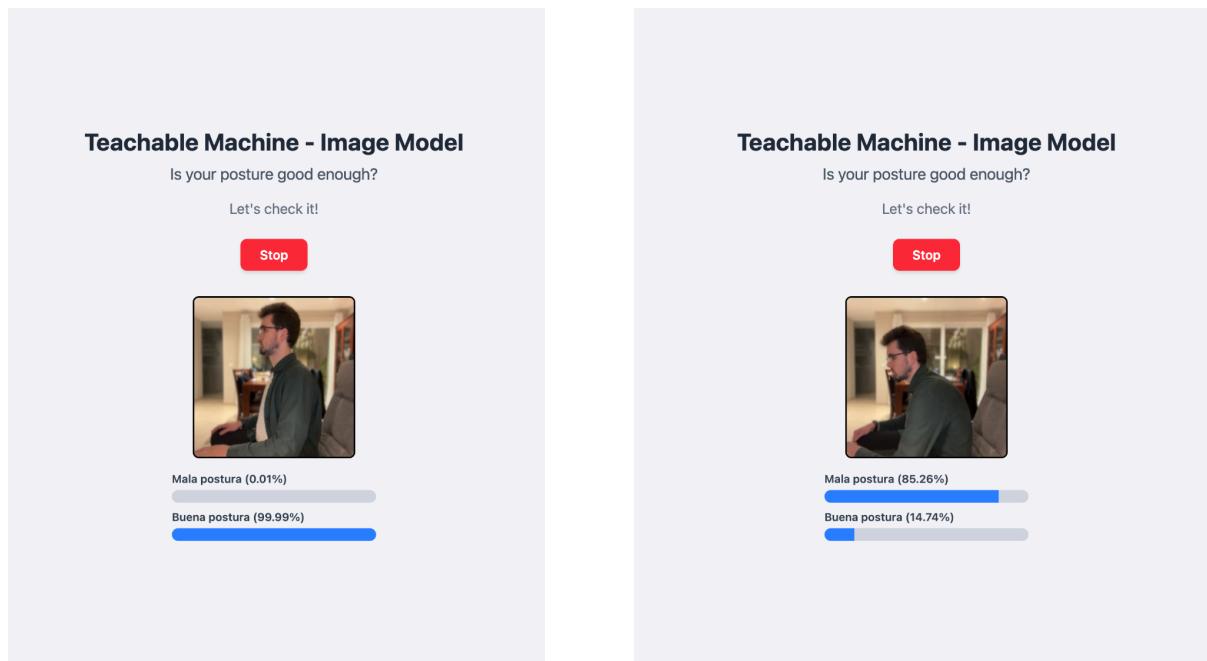
1.4 Publicación de la página web

La publicación ha sido a través de Netlify ya que tengo más experiencia en esta plataforma por subir anteriores proyectos.

```
codigo-teachable-machine > index.html > script
1  <h1>Teachable Machine - Image Model</h1>
2  <button type="button" onclick="init()">Start</button>
3  <div id="webcam-container"></div>
4  <div id="label-container"></div>
5  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest/dist/tf.min.js"></script>
6  <script src="https://cdn.jsdelivr.net/npm/@teachablemachine/image@latest/dist/teachablemachine-image.min.js"></script>
7  <script type="text/javascript">
8  // More API functions here:
9  // https://github.com/googlecreativelab/teachablemachine-community/tree/master/libraries/image
10 // the link to your model provided by Teachable Machine export panel
11 const URL = "https://teachablemachine.withgoogle.com/models/T--ciLSHj/";
12
13 let model, webcam, labelContainer, maxPredictions;
14
15 // Load the image model and setup the webcam
16 async function init() {
17   const modelURL = URL + "model.json";
18   const metadataURL = URL + "metadata.json";
19 }
```

Y aquí podemos ver la web en acción. Adjunto el link para poder acceder a ella:

<https://posture-correction-default-code.netlify.app/>



2 - PARTE B - ASISTENTE DE PROGRAMACIÓN

2.1 Explicación del asistente

El asistente de programación que he elegido es ChatGPT, aquí vemos su explicación del código.

Este código es una implementación de **Teachable Machine** de Google para reconocimiento de imágenes en tiempo real usando una webcam. Te explico parte por parte:

Estructura HTML

```
html Copiar
<h1>Teachable Machine - Image Model</h1>
<button type="button" onclick="init()">Start</button>
<div id="webcam-container"></div>
<div id="label-container"></div>
```

- **Encabezado (`<h1>`):** Título de la página.
- **Botón (`<button>`):** Al hacer clic en él, se ejecuta la función `init()`, que inicia la cámara y el modelo.
- **Divs (`<div>`):**
 - `webcam-container`: Donde se mostrará el video de la webcam.
 - `label-container`: Mostrará las predicciones del modelo.

Carga de Librerías

```
html Copiar
<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"><scr
```

- **TensorFlow.js:** Librería para trabajar con modelos de IA en el navegador.
- **Teachable Machine Image:** Extensión específica para modelos de clasificación de imágenes.

Código JavaScript

```
js Copiar
const URL = "https://teachablemachine.withgoogle.com/models/T--ciLSHj/";
let model, webcam, labelContainer, maxPredictions;
```

- **URL:** Dirección donde está alojado el modelo de Teachable Machine.
- **Variables:**
 - `model`: Almacena el modelo de IA.
 - `webcam`: Controla la webcam.
 - `labelContainer`: Contenedor donde se mostrarán los resultados.
 - `maxPredictions`: Número de categorías que el modelo puede predecir.

Función init() → Inicializa el modelo y la cámara

```
js Copiar
async function init() {
  const modelURL = URL + "model.json";
  const metadataURL = URL + "metadata.json";

  model = await tmImage.load(modelURL, metadataURL);
  maxPredictions = model.getTotalClasses();
```

- **Carga el modelo** desde la URL y obtiene cuántas clases puede predecir.

```
js Copiar
const flip = true;
webcam = new tmImage.Webcam(200, 200, flip);
await webcam.setup();
await webcam.play();
window.requestAnimationFrame(loop);
```

- Carga el modelo desde la URL y obtiene cuántas clases puede predecir.

js

 Copiar

```
const flip = true;
webcam = new tmImage.Webcam(200, 200, flip);
await webcam.setup();
await webcam.play();
window.requestAnimationFrame(loop);
```

- Configura la webcam:

- flip = true: Invierte la imagen (efecto espejo).
- webcam.setup(): Sigue la configuración de la cámara.
- webcam.play(): Activa la cámara.
- window.requestAnimationFrame(loop): Comienza a ejecutar la función loop() en un bucle.

js

 Copiar

```
document.getElementById("webcam-container").appendChild(webcam.canvas);
labelContainer = document.getElementById("label-container");
for (let i = 0; i < maxPredictions; i++) {
    labelContainer.appendChild(document.createElement("div"));
}
```

- Agrega el video de la webcam al HTML y crea divs en label-container para cada predicción.

Función loop() → Se ejecuta constantemente para actualizar la imagen y predecir

js

 Copiar

```
async function loop() {
    webcam.update();
    await predict();
    window.requestAnimationFrame(loop);
```



Función loop() → Se ejecuta constantemente para actualizar la imagen y predecir

js

 Copiar

```
async function loop() {
    webcam.update();
    await predict();
    window.requestAnimationFrame(loop);
}
```

- Actualiza el video con webcam.update().
- Llama a predict() para hacer predicciones en cada fotograma.
- window.requestAnimationFrame(loop) mantiene el bucle en ejecución.

Función predict() → Hace las predicciones y las muestra en la pantalla

js

 Copiar

```
async function predict() {
    const prediction = await model.predict(webcam.canvas);
    for (let i = 0; i < maxPredictions; i++) {
        const classPrediction =
            prediction[i].className + ": " + prediction[i].probability.toFixed(2);
        labelContainer.childNodes[i].innerHTML = classPrediction;
    }
}
```

- Toma la imagen de la webcam y la pasa al modelo (model.predict(webcam.canvas)).
- Muestra la predicción en label-container, indicando el nombre de la clase y la probabilidad.

Resumen

 Este código:



Resumen

✍ Este código:

1. **Carga un modelo de IA** entrenado con **Teachable Machine**.
2. **Activa la webcam** cuando el usuario pulsa "Start".
3. **Captura la imagen en tiempo real** y la envía al modelo.
4. **Predice la categoría** de la imagen y muestra los resultados en la pantalla.

¿Te queda claro o quieres que te explique algo más en detalle? 😊



2.2 Cambio de funcionalidad

Lo primero que he hecho ha sido pedir un diseño más amigable visualmente usando TailwindCSS a partir del HTML puro que ofrece Teachable Machine.

ChatGPT >

dame un buen layout para esto

```
<body>
<h1 class="text-3xl">Teachable Machine - Image Model</h1>
<h2 class="text-xl">Is your posture good enough?</h2>
<p class="text-lg">Let's check it!</p>
<button class="" type="button" onclick="init()">Start</button>
<div id="webcam-container"></div>
<div id="label-container"></div>
</body>
```

Aquí tienes un **layout mejorado** con un diseño más limpio y atractivo usando **Tailwind CSS**.

◆ Mejoras aplicadas:

- ✓ Mejor estructura visual con flexbox para centrar contenido.
- ✓ Botón estilizado para mayor visibilidad.
- ✓ Secciones separadas con div para mejor organización.
- ✓ Webcam bien posicionada y con un marco sutil.

 Código Mejorado

```
html

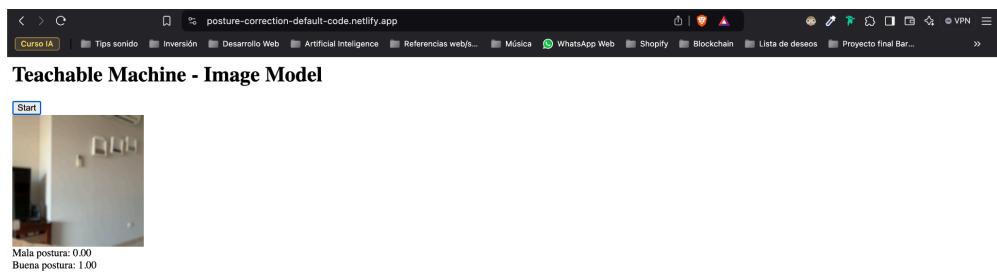
<body class="flex flex-col items-center justify-center min-h-screen bg-gray-100 p-6">
  <div class="text-center">
    <h1 class="text-3xl font-bold text-gray-800 mb-2">Teachable Machine - Image Model</h1>
    <h2 class="text-xl text-gray-600 mb-4">Is your posture good enough?</h2>
    <p class="text-lg text-gray-500 mb-6">Let's check it!</p>

    <button class="px-6 py-2 bg-blue-600 text-white font-semibold rounded-lg shadow-md hover:bg-blue-700 transition">
      Start
    </button>
  </div>

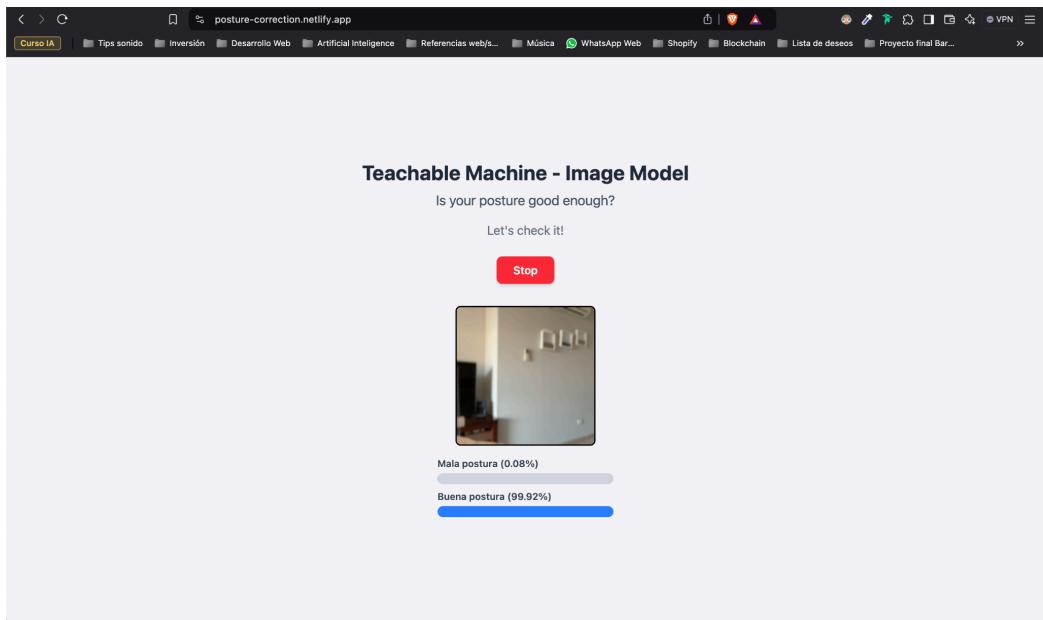
  <div id="webcam-container" class="mt-8 border-2 border-gray-300 rounded-lg shadow-md overflow-hidden"></div>
  <div id="label-container" class="mt-4 text-lg font-semibold text-gray-700"></div>
</body>
```

Aquí vemos los dos diseños uno al lado del otro:

Diseño original



Diseño modificado



Ahora sí, entrando en la funcionalidad. Inspirado en la web de **Teachable Machine**, quise replicar la opción de pausar la cámara y cambiar los colores a blanco y negro. Me pareció una funcionalidad interesante, así que pedí ayuda a ChatGPT para implementarla.

Primero, dividí la función init() en dos: **initStart()** y **initStop()**, con la intención de asignar distintas funcionalidades a cada botón.

Luego, le pedí a ChatGPT que añadiera algunas mejoras a la función **initStart()**, que se ejecuta al pulsar “Start”:

- Restaurar los colores originales de la imagen tras aplicar la escala de grises.
- Eliminar el contenido de webcam-container para evitar duplicados (un problema en el código original).
- Mostrar y ocultar los botones “Start” y “Stop” según corresponda.

De esta manera, evito la duplicación de la cámara, aseguro que los colores vuelvan a la normalidad al detener el modo en escala de grises y controlo la visibilidad de los botones según el estado de la cámara.

ChatGPT >

Modifica este código para que haga varias cosas:
- Se quite la clase "grayscale" del web-container
- Se oculte el botón de "start" cuando se pulsa y viceversa con el de "stop"
- Y una parte importante, que que resetee el web-container. Si ya tiene contenido y se vuelve a pulsar "start" quiero que se resetee para evitar duplicados

```
async function init() {
  const modelURL = URL + "model.json";
  const metadataURL = URL + "metadata.json";

  // load the model and metadata
  // Refer to tmlImage.loadFromFiles() in the API to support files from a file picker
  // or files from your local hard drive
  // Note: the pose library adds "tmlImage" object to your window (window.tmlImage)
  model = await tmlImage.load(modelURL, metadataURL);
  maxPredictions = model.getTotalClasses(); // get total number of classes predicted by the
  model

  // Convenience function to setup a webcam
  const flip = true; // whether to flip the webcam
  webcam = new tmlImage.Webcam(200, 200, flip); // width, height, flip
  await webcam.setup(); // request access to the webcam
  await webcam.play();
  window.requestAnimationFrame(loop);

  // append elements to the DOM
  document.getElementById("webcam-container").appendChild(webcam.canvas);
  labelContainer = document.getElementById("label-container");
  for (let i = 0; i < maxPredictions; i++) { // and class labels
    labelContainer.appendChild(document.createElement("div"));
  }
}
```

Vale, aquí tienes el código modificado según lo que pediste:

```
javascript
async function init() {
    const modelURL = URL + "model.json";
    const metadataURL = URL + "metadata.json";

    // Cargar el modelo y los metadatos
    model = await tmImage.load(modelURL, metadataURL);
    maxPredictions = model.getTotalClasses();

    // Obtener referencias a los botones y contenedor
    const btnStart = document.getElementById("btn-start");
    const btnStop = document.getElementById("btn-stop");
    const webcamContainer = document.getElementById("webcam-container");

    // Eliminar el contenido anterior para evitar duplicados
    webcamContainer.innerHTML = "";

    // Configurar la webcam
    const flip = true;
    webcam = new tmImage.Webcam(200, 200, flip);
    await webcam.setup();
    await webcam.play();
    window.requestAnimationFrame(loop);

    // Agregar elementos al DOM
    webcamContainer.appendChild(webcam.canvas);
    labelContainer = document.getElementById("label-container");
    for (let i = 0; i < maxPredictions; i++) {
        labelContainer.appendChild(document.createElement("div"));
    }

    // Eliminar la clase "grayscale" si está presente
    if (webcamContainer.classList.contains("grayscale")) {
        webcamContainer.classList.remove("grayscale");
    }

    // Alternar botones
    btnStart.classList.add("hidden");
    btnStop.classList.remove("hidden");
}
```

Copiar

Después de comprobar el código generado por ChatGPT he visto que tanto la funcionalidad del color como la de ocultar y mostrar los botones funcionan correctamente. Sin embargo, definir “`webcamContainer.innerHTML`” como un string vacío no ha sido una buena solución.

La webcam se sigue duplicando. Veo que se usa `appendChild` para añadir el `webcam.canvas`, prueba a modificar algo relacionado con eso para poder evitar duplicados en el `webcamContainer`

Claro, aquí tienes una posible respuesta al código que mencionas:

Lo que pasa es que, para evitar duplicados, puedes usar `removeChild` de la siguiente manera:

```
javascript
if (webcamContainer.childNodes.length > 0) {
    webcamContainer.removeChild(webcamContainer.childNodes[0]);
}
```

Copiar

Con esto, antes de añadir un nuevo canvas, se elimina el que pudiera haber en `webcamContainer`. Esto asegura que no haya más de un canvas en el contenedor al mismo tiempo.

Esta sí que ha sido una solución válida. Vemos que a través de los *childNodes* es posible vaciar el contenedor. En cuanto *webcam-container* tiene un *childNode* se ejecuta el código en el que lo elimina.

Seguido de esto, he usado al ChatGPT para crear la función **initStop()**, que realiza las siguientes funciones:

- Detener la webcam
- Añadir filtro de color blanco y negro
- Ocultar el botón *stop* y mostrar el botón *start*

ChatGPT > ↑

Crea una función initStop() que contenga las siguientes funcionalidades:
- Detener la webcam
- Añadirle un filtro de color blanco y negro a través de TailwindCSS
- Ocultar el botón "stop" y mostrar el botón "start"

Aquí tienes la función initStop() que cumple con lo que pides:

```
javascript Copiar
async function initStop() {
  const btnStart = document.getElementById("btn-start");
  const btnStop = document.getElementById("btn-stop");
  const webcamContainer = document.getElementById("webcam-container");

  // Detener la webcam
  if (webcam) {
    await webcam.stop();
    // Añadir filtro blanco y negro usando TailwindCSS
    webcamContainer.classList.add("filter", "grayscale");
  }

  // Mostrar y ocultar botones
  if (btnStart.classList.contains("hidden")) {
    btnStart.classList.remove("hidden");
    btnStop.classList.add("hidden");
  } else {
    btnStart.classList.add("hidden");
    btnStop.classList.remove("hidden");
  }
}
```

Explicación:

1. **Detener la webcam:** Usa `await webcam.stop()` para detener la cámara.
2. **Filtro blanco y negro:** Se utiliza el filtro de Tailwind `filter grayscale` para poner la imagen en blanco y negro.
3. **Mostrar/ocultar botones:** Al presionar "stop", el botón "start" se muestra y el de "stop" se oculta.

¡Listo para usarlo! ¿Te gustaría ajustar algo más? 😊

↓

undo redo back forward

Por último le he pedido que modifique el código de la función “predict()” para que el porcentaje del resultado que devuelve el modelo se vea representado en una barra de progreso.

The screenshot shows a ChatGPT interface. At the top, it says "ChatGPT >". Below that, a message reads: "Puedes usar una barra de progreso de Tailwind para mostrar el porcentaje de cada predicción en lugar de números. Aquí tienes cómo hacerlo:". Underneath, a section titled "Modificaciones en predict()" with a red checkmark icon contains the instruction: "Reemplaza la parte donde actualizas labelContainer con una barra de progreso." A horizontal line separates this from a code editor area. The code editor has a title "Código modificado" with a file icon. It shows a snippet of JavaScript code:

```
js Copiar

async function predict() {
  const prediction = await model.predict(webcam.canvas);
  labelContainer.innerHTML = ""; // Limpiar etiquetas previas

  for (let i = 0; i < maxPredictions; i++) {
    const className = prediction[i].className;
    const probability = (prediction[i].probability * 100).toFixed(2); // Convertir a porcentaje

    // Crear el contenedor de la barra
    const container = document.createElement("div");
    container.className = "mb-2 w-64"; // Ajusta el ancho si es necesario

    // Etiqueta de la clase
    const label = document.createElement("div");
    label.className = "text-sm font-semibold text-gray-700 mb-1";
    label.innerText = `${className} (${probability})`;

    // Barra de progreso
    const progressBar = document.createElement("div");
    progressBar.className = "h-4 bg-gray-300 rounded-lg overflow-hidden";
    progressBar.innerHTML = `<div class="h-full bg-blue-500" style="width: ${probability}%;"></div>`;

    // Agregar elementos al contenedor principal
    container.appendChild(label);
    container.appendChild(progressBar);
    labelContainer.appendChild(container);
  }
}
```

Y el resultado es todo un éxito. Para comprobarlo adjunto a continuación la URL de la web.

Al igual que en el apartado anterior he usado Netlify para desplegar el proyecto.

<https://posture-correction.netlify.app/>