Tutorial 2

Creación de una webcam inteligente con COCO-SSD

Introducción

En este tutorial, crearás una aplicación de cámara web inteligente que detecta objetos específicos en tiempo real. La aplicación utiliza un modelo de detección de objetos, una tecnología que impulsa dispositivos ampliamente utilizados como Nest-Cams y sistemas de timbres inteligentes. En esta aplicación, utilizarás el modelo pre-entrenado prefabricado COCO-SSD.

Objetivos

En este tutorial aprenderás a:

- Cargar un modelo pre-entrenado prefabricado de TensorFlow.js
- Clasificar una imagen para encontrar los cuadros delimitadores de objetos
- Utilizar las API de TFJS para recuperar fotogramas de la cámara Web como tensores
- Potenciar la cámara web utilizando los datos transmitidos desde el modelo y así resaltar los objetos encontrados

Requisitos

- Una versión reciente de Chrome o de otro navegador actualizado.
- Tu editor favorito que se **ejecute localmente en tu máquina**, por ejemplo Visual Studio Code con la extensión Live Server, **o en la Web** con servicios como codesandbox.io o glitch.com.
- Familiaridad con JavaScript.
- Familiaridad con las herramientas para desarrolladores de Chrome (DevTools) o de otro navegador.

Código

- Código tutorial

Tutorial

El modelo COCO-SSD utiliza el dataset Common Objects in COntext (COCO) y la arquitectura del modelo Single Shot MultiBox Detection (SSD). COCO es un dataset para la **detección**, **segmentación** y **etiquetado** de objetos que fue creado y organizado por Microsoft. Básicamente, esto significa que el modelo se puede entrenar para detectar objetos y localizarlos **devolviendo las coordenadas de los cuadros delimitadores que se pueden dibujar a su alrededor**, y también proporciona **etiquetas o leyendas para los objetos detectados**. El modelo COCO-SSD puede reconocer 90 objetos comunes listos

para usar. Si bien la función central del modelo es detectar objetos, depende del desarrollador usar esta función para desarrollar la lógica de la aplicación para casos de uso específicos.

Imagina poder detectar si una persona estaba en un video, por lo que luego podrías contar cuántas personas estaban presentes en un momento dado para estimar qué tan ocupada estuvo un área determinada durante el día, o enviarte una alerta cuando se detecte que tu perro está en una habitación de tu casa mientras estás fuera, en la que tal vez no debería estar. Si pudiera hacer eso, estaría bien encaminado para hacer tu propia versión de una cámara Nest de Google que podría avisarte cuando ve un intruso (de cualquier tipo) utilizando su propio hardware personalizado.

Paso 1. Esqueleto HTML

1.1. Cargar TensorFlow.js y el modelo COCO-SSD

Empieza creando el esqueleto de una página Web. Crea un archivo y nómbralo **index.html**. El archivo contendrá el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <title>Tutorial COCO-SSD</title>
   <meta charset="utf-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1" />

 </head>
 <body>

 </body>
</html>
```

En la etiqueta `body`, al final, introduce el código para cargar la última versión de la librería TensorFlow.js y también para cargar el modelo COCO-SSD. También incluirás el código para cargar el archivo **index.js**, que es el fichero donde escribirás el código JavaScript de la aplicación para la detección de objetos con COCO-SSD.

Crea un archivo **index.js** que inicialmente estará vacío.

```
<head>
  …
    <!-- TensorFlow.js library -->
   <script
     src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"
     type="text/javascript"
```

```
  ></script>

  <!-- COCO-SSD -->
  <script
src="https://cdn.jsdelivr.net/npm/@tensorflow-models/coco-ssd
"
  ></script>
 </head>
```

1.2. Rellenar el esqueleto de la página HTML

Siempre en la etiqueta body, introduce también el código a continuación, justo antes de la etiqueta script:

```
 <body>
  <h1>Intelligent webcam with COCO-SSD</h1>
  <div id="outputMessage">Wait for the COCO-SSD model to load before
enabling the webcam and starting object detection</div>


  <section id="app">
    <div id="liveView" class="camView">
      <button id="btnEnableWebcam" disabled>Enable webcam  </button>
      <video id="webcam" autoplay muted width="640" height="480"></video>
    </div>
  </section>

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

1.3. Entender el código

Observa que:

- Has agregado una etiqueta h1 y una etiqueta etiqueta div para el encabezado y alguna información que te dice que esperes a que el modelo se haya cargado correctamente
- Has agregado una etiqueta section que contiene el espacio principal de la aplicación.
- Has agregado el botón para habilitar la cámara web con la etiqueta button con el id **btnEnableWebcam**, que aparece inicialmente deshabilitado mediante el atributo disabled.
- También has agregado una etiqueta video a la que transmitirás la entrada de la cámara web. En breve lo configurarás en el código JavaScript.

Paso 2. Añadir estilo con un fichero CSS

En la etiqueta head, justo después de la última etiqueta meta, añade el código para cargar el fichero **style.css** donde se especifica el estilo de la aplicación.

```
<head>
  …
 <meta name="viewport" content="width=device-width, initial-scale=1"/>

  <!-- CSS Styling -->
  <link rel="stylesheet" href="style.css" />
  …
</head>
```

A continuación, crea el archivo **style.css** y, luego, incluye el siguiente código para definir el estilo de tu aplicación. Primero, los estilos para los elementos HTML que acabas de agregar para asegurarte de que se representen correctamente:

```
body {
 font-family: helvetica, arial, sans-serif;
 margin: 2em;
 color: #3d3d3d;
}

h1 {
 font-style: italic;
 color: #ff6f00;
}

video {
 display: block;
}
```

A continuación, agrega algunas clases CSS útiles para gestionar estados diferentes de la interfaz de usuario, como ocultar el botón o hacer que el área principal de la aplicación no aparezca disponible si el modelo aún no está listo.

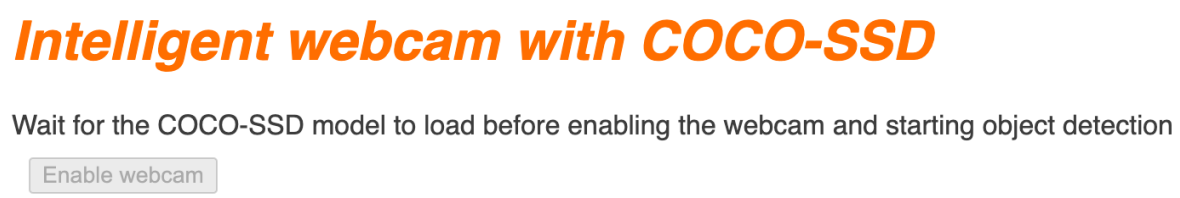
```
.removed {
 display: none;
}

.camView {
 position: relative;
```

```
 float: left;
 width: calc(100% - 20px);
 margin: 10px;
 cursor: pointer;
}

.highlighter {
 background: rgba(0, 255, 0, 0.25);
 border: 1px dashed #fff;
 z-index: 1;
 position: absolute;
}
```

Ejecuta la aplicación web y observa cómo el botón no está habilitado. Utilizarás JavaScript para eliminar esta clase una vez que el modelo esté listo para usarse.



Paso 3. Crear el esqueleto JavaScript

El código en este paso se muestra en orden, pero se divide en partes para una explicación más detallada. Si copias y pegas cada segmento al final del archivo JavaScript cada vez, o reemplaza una función vacía definida del paso anterior como se recomienda, todo debería funcionar bien.

3.1. Referenciar los elementos del DOM

Primero, asegúrate de poder acceder a las partes clave de la página que necesitarás manipular o acceder más adelante:

```
const webcamEl = document.querySelector("#webcam");
const liveView = document.querySelector("#liveView");
const appSection = document.querySelector("#app");
const btnEnableWebcam = document.querySelector("#btnEnableWebcam");
const outputMessageEl = document.querySelector("#outputMessage");
```

3.2. Controlar el acceso a la cámara web

Ahora puede agregar algunas funciones de asistencia para verificar si el navegador admite el acceso a la transmisión de la cámara web a través del API getUserMedia. Añade este control en la función con la lógica principal de la aplicación (función app).

```
let webcam;

function getUserMediaSupported() {
 return Boolean(navigator.mediaDevices &&
navigator.mediaDevices.getUserMedia);
}

async function app() {
 // Check for webcam support before proceeding
 if (!getUserMediaSupported()) {
   console.warn("getUserMedia() is not supported by your browser");
   outputMessageEl.innerText = "Webcam not supported by your browser.";
   return; // Exit if webcam is not supported
 }

btnEnableWebcam.addEventListener("click", enableCam);

}

async function enableCam(event) { }
```

Si tienes acceso a la cámara web, puedes añadir un detector de eventos **“click”** al botón para activar la cámara web. De esta manera, al hacer clic en el botón, se llamará a la función **enableCam** que implementarás a continuación.

3.3. Obtener la transmisión de la cámara web

Completa el código para la función **enableCam** previamente vacía copiando y pegando el código a continuación:

```
async function enableCam(event) {
 if (!model) return;

 event.target.classList.add("removed");
 try {
   webcam = await tf.data.webcam(webcamEl);
   outputMessageEl.innerText = "Webcam enabled! Detecting objects...";
   await predictWebcam();
 } catch (error) {
   console.error("Webcam access failed:", error);
   outputMessageEl.innerText = "Webcam access denied.";
 }
```

```
}
```

Si la cámara es accesible, el flujo de vídeo se inicia y se llama a la función predictWebcam() para comenzar la detección de objetos en tiempo real. En caso de que el acceso a la cámara sea denegado o falle, se mostrará un mensaje de error en outputMessageEl y el error será registrado en la consola.

Finalmente, agrega un código temporal para que puedas probar si la cámara web funciona. El siguiente código pretenderá que el modelo está cargado y habilitará el botón de la cámara, para que puedas hacer clic en él. Reemplazarás todo este código en el próximo paso:

```
let model = true;
async function predictWebcam() {}
```

Ahora, si ejecutas el código tal como está y haces clic en el botón una vez que esté disponible, deberías ver abierta una vista de cámara web en vivo, después de aceptar el acceso a tu cámara web en tu navegador web (que generalmente aparecerá en la parte superior izquierda del navegador como una ventana emergente).

3.4. Análisis del código

Utilizamos el API de tensorflow a través del objeto tf para recuperar datos de la cámara Web. El método tf.data.webcam() es un método asíncrono, si utilizamos la sintaxis async/await tendrá que estar dentro de una función async. Por esta razón la función enableCam(event) se ha definido como async. tf.data.webcam() retorna una promesa que se resuelve en iterador que genera tensores de rango 3 que representan imágenes RGB desde el flujo de datos de la cámara Web. Esta API solo funciona en el entorno del navegador cuando el dispositivo tiene cámara web. El método acepta como parámetro (opcional) el elemento video que contiene la salida de la cámara Web en la página HTML (HTMLVideoElement). Este método de TFJS, por tanto, nos permite recuperar fotogramas para analizarlos con el modelo COCO-SSD y permite visualizar la salida de la cámara web en el elemento que le pasamos como argumento.

Paso 4. Añadir el modelo de aprendizaje automático

Ahora estás listo para cargar el modelo COCO-SSD.

4.1. Cargar COCO-SSD

Añade el siguiente código al final del fichero **index.js** para cargar el modelo COCO-SSD

```
async function loadCocoSsdModel() {
 model = await cocoSsd.load();
 btnEnableWebcam.disabled = false;
 outputMessageEl.innerText = "Model loaded! Enable the webcam to start
detection.";
}
```

Añade la llamada a la función loadCocoSsdModel() para cargar el modelo al final de la función app(), que quedará así:

```
async function app() {
 // Check for webcam support before proceeding
 if (!getUserMediaSupported()) {
   console.warn("getUserMedia() is not supported by your
browser");
   outputMessageEl.innerText = "Webcam not supported by your
browser.";
   return; // Exit if webcam is not supported
 }
 btnEnableWebcam.addEventListener("click", enableCam);

 await loadCocoSsdModel(); // Load the model if webcam
support is confirmed
 }
```

Una vez que hayas agregado el código anterior y actualices la vista en vivo, notarás que unos segundos después de que la página se haya cargado (dependiendo de la velocidad de la conexión a internet) el botón para habilitar la cámara web se activará automáticamente cuando el modelo esté listo para usar.

Ahora es el momento de definir la función para hacer las predicciones, es decir detectar objetos utilizando COCO-SSD con las imágenes (fotogramas) de la cámara web.

4.2. Detectar objetos en cada fotograma
Añade el código a continuación para permitir que la aplicación tome continuamente un fotograma de la transmisión de la cámara web cuando el navegador esté listo y lo pase al modelo para su clasificación.

Luego, el modelo analizará los resultados y, si los resultados están por encima de un nivel de confianza determinado (en nuestro caso 0.66), dibujará una etiqueta p en las coordenadas que devuelve la predicción, poniendo como texto la clase del objeto (el atributo class de la predicción devuelta por el modelo. También dibujará el cuadro delimitador utilizando la información en el atributo bbox. El nivel de confianza es el valor del atributo score.

```
async function predictWebcam() {
 const objects = [];
 while (true) {
```

```
    const frame = await webcam.capture();
    const predictions = await model.detect(frame);
    objects.forEach((object) => liveView.removeChild(object));
    objects.length = 0;


    for (let n = 0; n < predictions.length; n++) {
      if (predictions[n].score > 0.66) {
        const p = document.createElement("p");
        const c = predictions[n].class;
        const score = Math.round(parseFloat(predictions[n].score) * 100);
        p.innerText = `${c} - with ${score}% confidence.`;
        p.style = `margin-left: ${predictions[n].bbox[0]}px;
                   margin-top: ${predictions[n].bbox[1] - 10}px;
                   width: ${predictions[n].bbox[2] - 10}px;
                   top: 0;
                   left: 0;`;

        const highlighter = document.createElement("div");
        highlighter.setAttribute("class", "highlighter");
        highlighter.style = `left: ${predictions[n].bbox[0]}px;
                             top: ${predictions[n].bbox[1]}px;
                             width: ${predictions[n].bbox[2]}px;
                             height: ${predictions[n].bbox[3]}px;`;

        liveView.appendChild(highlighter);
        liveView.appendChild(p);
        objects.push(highlighter);
        objects.push(p);
      }
    }

    frame.dispose();
    await tf.nextFrame();
  }
}
```

4.3. Análisis del código

La llamada realmente importante en este nuevo código es model.detect().

Todos los modelos pre-entrenados para TensorFlow.js tienen una función como esta, o parecida, el nombre puede cambiar de un modelo a otro, que realiza la inferencia de aprendizaje automático. Por ejemplo, en el tutorial anterior, con el modelo MobileNet las predicciones se realizaban con el método classify().

La inferencia es simplemente el acto de tomar algunas entradas y ejecutarlas a través del modelo de aprendizaje automático, y luego proporcionar algunos resultados. Con los modelos pre-entrenados de TensorFlow.js, las predicciones se devuelven en forma de objetos JSON, por lo que son más fáciles de usar para la creación de aplicaciones web interactivas.

Puedes encontrar los detalles completos del método detect() en la documentación de GitHub para el modelo COCO-SSD. Este método puede aceptar cualquier objeto que represente una imágen como parámetro, como una imagen, un video, un canvas, etc. y retorna un array de objetos que representan las predicciones.

Como se ha adelantado, cada predicción contiene un atributo score (nivel de confianza), un atributo class con la etiqueta del objeto y un atributo bbox con las coordenadas de donde se encuentra el objeto en la imagen. La variable predictions es un array que contiene los objetos JSON que describen el resultado de la detección de cada objeto en la escena (fotograma)

La variable objects es un array que sirve para almacenar la etiqueta y el cuadro delimitador que se dibujará en la salida de la cámara web para cada objeto detectado. Este cuadro se dibuja a través de elementos HTML que se añaden como hijos del elemento div que contiene el elemento video donde se visualiza la salida de la cámara web (id **liveView**). En cada fotograma, antes de dibujar los nuevos cuadros delimitadores, tenemos que eliminar los anteriores. El array objects sirve el propósito de tener referencias a los cuadros del fotograma anterior, para así eliminarlos en cada iteración del bucle, después de haber esperado que el siguiente fotograma esté disponible (método tf.nextFrame()). La porción de código que se encarga de eliminar los cuadros delimitadores anteriores es esta:

```
// Remove previously created DOM elements from the live view
objects.forEach((object) => liveView.removeChild(object));
objects.length = 0;
```

El código objects.length = 0; es una de las posibles maneras para vaciar un array.

Al final de cada iteración del bucle, eliminamos también de la memoria el objeto frame, que es el tensor que representa la imagen devuelto por el método webcam.capture(). Esto se hace a través del dispose() de un tensor. En TensorFlow es necesario eliminar los tensores de la memoria manualmente, a no ser que el código se ejecute dentro de la función tf.tidy().

La eliminación de los tensores que no utilizamos es una tarea que tenemos que realizar manualmente (no hay recolector de basura) y es crucial hacerlo en un bucle para evitar memory leaks.

Para cada uno de los objetos en el array predictions, si el valor de score es superior a **0.66**, se crea una nueva etiqueta (elemento p) con el nombre de la clase del objeto y el valor de la confianza de la predicción. También se crea el cuadro (elemento div) con los valores recuperados del atributo bbox. Ambos elementos se añaden al div **liveView** y se almacenan temporalmente en el array objects. En el código puedes ver también cómo se define el estilo de estos dos nuevos elementos utilizando el atributo style y la sintaxis de plantillas de cadenas.

Ejercicios

1. Cambiar el umbral de confianza. Ajustar el nivel de confianza para mostrar sólo objetos detectados con un score superior a un umbral configurable
2. Filtrar por tipo de objeto. Modificar el código para que la aplicación solo muestre las predicciones de un tipo específico de objeto, por ejemplo, personas.
3. Historial de detecciones. Modificar el código para que guarde un historial de las últimas detecciones de objetos y lo muestre en la pantalla. Por ejemplo, mostrar los últimos cinco objetos detectados junto con su clase y nivel de confianza.