Tutorial 4

Detección de puntos clave de las manos con hand-pose-detection

Introducción

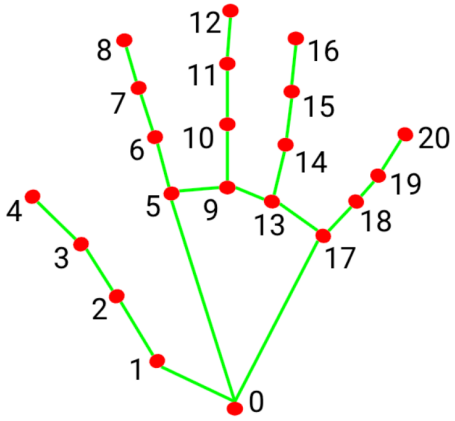
En este tutorial aprenderemos a usar el modelo prefabricado hand-pose-detection de TensorFlow.js para detectar las posiciones de puntos clave en las manos en tiempo real usando la cámara web. Este modelo tiene aplicaciones en reconocimiento de gestos, interacción en realidad aumentada y más.

El modelo hand-pose-detection puede detectar hasta 21 puntos clave en cada mano, incluyendo las posiciones de los dedos y las articulaciones. Funciona en dispositivos modernos con una velocidad de más de 30 FPS, permitiendo aplicaciones interactivas y en tiempo real. El modelo hand-pose-detection de TensorFlow.js está diseñado para procesar imágenes RGB de 252x252 píxeles, como las capturadas por cámaras web o dispositivos móviles. Este modelo puede detectar múltiples manos en una sola imagen y, para cada mano detectada, proporciona 21 puntos clave en 2D y 3D que representan las posiciones de las articulaciones y extremos de los dedos. Funciona de manera óptima cuando las manos están claramente visibles en la imagen, típicamente en un rango de distancia entre **0,5 y 2** metros de la cámara.

El modelo devuelve un array con las predicciones de las manos detectadas en la imagen. Para cada mano, la información incluye:

- **score**: Un valor entre 0 y 1 que indica la confianza del modelo en la detección de la mano.
- **handedness**: Una predicción sobre si la mano es izquierda o derecha.
- **keypoints**: Un array de 21 puntos clave, cada uno con coordenadas x y y que representan la posición en píxeles dentro de la imagen, y un name que identifica la articulación específica (por ejemplo, “wrist”, “thumb_tip”).
- **keypoints3D**: Un array de 21 puntos clave en 3D, cada uno con coordenadas x, y y z que representan la posición en un espacio métrico tridimensional, donde el origen se define como el promedio entre las primeras articulaciones de los dedos índice, medio, anular y meñique.

El modelo en la actualidad permite la única opción de utilizar el modelo de detección de puntos clave de las manos ofrecido por la librería MediaPipe. Los 21 puntos clave detectados siguen el diagrama a continuación, que indica también su posición en el array.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Las distintas siglas hacen referencias los términos anatómicos relacionados con las articulaciones y puntos de referencia de los dedos:

Sigla	Término anatómico	Traducción
CMC	CarpoMetaCarpal joint	Articulación carpometacarpiana
MCP	MetaCarpoPhalangeal joint	Articulación metacarpofalángica
PIP	Proximal InterPhalangeal joint	Articulación interfalángica proximal
DIP	Distal InterPhalangeal joint	Articulación interfalángica distal
TIP	fingerTIP	Punta del dedo

Para más información sobre el modelo, puedes consultar su página en el repositorio de modelos de TFJS: hand-pose-detection.

Objetivos

En este tutorial aprenderás a:

- Cargar el modelo hand-pose-detection en TensorFlow.js
- Ejecutar el modelo prefabricado hand-pose-detection para realizar estimaciones sobre fotogramas de la cámara web
- Visualizar los puntos clave de las manos detectadas y las conexiones entre articulaciones

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

- Tutorial para cámara web

Tutorial para cámara web

En este primer tutorial aprenderás a realizar estimaciones con movenet/singlepose/lighting utilizando como entrada al modelo los fotogramas de la cámara web.

Paso 1. Esqueleto HTML

1.1. Cargar TensorFlow.js y el modelo
Crea un archivo llamado index.html y copia el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <title>Tutorial movenet/singlepose/lighting</title>
   <meta charset="utf-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1,
maximum-scale=1, user-scalable=no" />

   <link rel="stylesheet" href="style.css" />
   <script src="https://cdn.jsdelivr.net/npm/@mediapipe/hands"></script>

   <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"
type="text/javascript"></script>

   <script
src="https://cdn.jsdelivr.net/npm/@tensorflow-models/hand-pose-detection"></scrip
t>
 </head>
<body>
   <script src="index.js"></script>
 </body>

</html>
```

Observa que, como hiciste para el modelo prefabricado COCO-SSD, se incluye el código para cargar el modelo hand-pose-detection así como el modelo hands, que también es necesario para el correcto funcionamiento.

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

1.2. Rellenar el esqueleto de la página HTML

En la etiqueta body, introduce también el código a continuación, justo antes de las etiquetas script:

```
  <section>
    <div>
      <canvas id="canvas" width="252" height="252"> </canvas>
      <video
        playsinline
        muted
        id="webcam"
        width="252px"
        height="252px"
      ></video>
    </div>
  </section>
```

1.3. Entender el código

Observa que en la etiqueta **section,** el espacio principal de la aplicación, hay **div** que contiene:

- El elemento <canvas> que utilizarás para visualizar la salida de hand-pose-detection.
- En el elemento <video> se transmitirá la entrada de la cámara web, al igual que hiciste en el tutorial anterior. El tamaño del video es de 252x252 pixeles.

Para el elemento <video> añadimos en el fichero **style.css** una directiva para que transforme la imagen, rotándola de 180 grados en el eje y para así obtener un efecto de espejo.

```
video {
 transform: scaleX(-1);
}
```

Paso 2. Código 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.

2.1. Inicialización del entorno

Primero, asegúrate de poder acceder a las partes clave de la página que necesitarás manipular o acceder más adelante, como el elemento video con la salida de la cámara web o el elemento canvas. Agrega el siguiente código para preparar los elementos principales de la página:

```
const webcamEl = document.querySelector("#webcam");
```

```
const canvas = document.querySelector("#canvas");
const outputMessageEl = document.querySelector("#output-message");

function initTFJS() {
 if (typeof tf === "undefined") {
   throw new Error("TensorFlow.js not loaded");
 }
}
```

2.2. Definir la función principal de la aplicación
Como hemos estado trabajando hasta ahora, el código que utilizaremos contendrá llamadas a funciones asíncronas. Para esto, creamos una función principal asíncrona app(), que contendrá nuestra aplicación:

```
async function app() { }
```

Escribiremos el código fuente principalmente en esta función.

Paso 3. Cargar el modelo de aprendizaje automático

Ahora estás listo para cargar el modelo hand-pose-detection.

3.1. Cargar hand-pose-detection
Para cargar el modelo, tendrás que añadir el siguiente código al principio de la función app(), en el fichero **index.js**.

```
async function app() {
  const model = handPoseDetection.SupportedModels.MediaPipeHands;
 const detectorConfig = {
   runtime: 'mediapipe',
   modelType: 'full',
   maxHands: 2,
   solutionPath: 'https://cdn.jsdelivr.net/npm/@mediapipe/hands'
   // or 'base/node_modules/@mediapipe/hands' in npm.
 };
 detector = await handPoseDetection.createDetector(model, detectorConfig);


 …
}
```

Este código configura y crea un detector para el modelo MediaPipe Hands utilizado en la detección de poses de las manos en tiempo real. Observa que utilizamos llamadas a la API específica del modelo prefabricado.

Primero se especifica el modelo a utilizar: MediaPipe Hands, que es el único modelo

soportado. Este modelo es altamente eficiente para detectar y rastrear manos, proporcionando 21 puntos clave por mano.
Luego se configura el detector de manos con los siguientes parámetros, como se especifica en la documentación del modelo:

- runtime: 'mediapipe': Indica que el modelo utiliza la solución MediaPipe para la detección. Esta opción es útil porque aprovecha una implementación optimizada para dispositivos móviles y navegadores.
- modelType: 'full': Selecciona el tipo de modelo a utilizar. En este caso, el modelo completo (full) ofrece la máxima precisión para detectar los puntos clave de las manos. También existe una opción más ligera (lite) que es más rápida, pero menos precisa.
- maxHands: 2: Configura el máximo número de manos que el detector puede identificar simultáneamente. Aquí se establece un límite de detección de dos manos.
- solutionPath: 'https://cdn.jsdelivr.net/npm/@mediapipe/hands': Especifica la ubicación donde se encuentran los archivos de solución de MediaPipe Hands. Este path puede ser (i) una URL (como en este caso, alojada en un CDN) o (ii) una ruta local (si los archivos se instalan mediante npm).

`handPoseDetection.createDetector()` es un método asincrónico que inicializa el detector utilizando el modelo y la configuración especificados. `await` indica que la creación del detector es una operación asincrónica. El código esperará a que el detector esté completamente inicializado antes de continuar.

Paso 4. Capturar fotogramas con la camara Web

Ahora pasamos a capturar los fotogramas de la cámara Web, transformándolos en tensores para así poder utilizarlos como entrada para nuestro modelo de detección de manos. Podríamos utilizar también una imagen desde la etiqueta `<img>`.

4.1. Instanciar el objeto para generar tensores

Primero, es necesario instanciar el objeto para generar tensores a partir de la cámara web. Como hemos visto en el tutorial anterior, para esto es necesario incluir el código continuación en la función app, utilizando `tf.data.webcam()`:

```
async function app() {
 const model = await tf.loadGraphModel(MODEL_URL, { fromTFHub: true });
 const webcam = await tf.data.webcam(webcamEl, {
   resizeWidth: 252,
   resizeHeight: 252
 });
 …
}
```

El segundo parámetro del método tf.data.webcam es un objeto que define la configuración de la cámara web. Con los atributos `resizeWidth` y `resizeHeight` estamos pidiendo que cada fotograma capturado por la cámara web, independientemente de su resolución, se escale a

una imagen de 252x252 píxeles. El modelo acepta imágenes 192x192 (lite) o 224x224 (full). Siendo un modelo prefabricado no necesitamos preprocesar la entrada.

A continuación, podemos empezar a generar los tensores para cada fotograma con el siguiente código, siempre en la función app():

```
while (true) {
   const img = await webcam.capture();

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

Como vimos en el primer tutorial, el método capture() (es un método asíncrono) retorna una promesa que se resuelve en el siguiente fotograma, un tensor que representa la imágen. El método tf.nextFrame() espera a que esté disponible el siguiente fotograma para que el bucle continúe. Recordad que es necesario eliminar los tensores de la memoria manualmente si no queremos generar problemas de memoria, especialmente si estamos escribiendo nuestro código en un bucle infinito. Hubiésemos podido utilizar también la función tf.tidy().

Paso 5. Realizar predicciones

Podemos ahora empezar a realizar las estimaciones de las articulaciones de las manos utilizando hand-pose-detection, y mostrar el resultado por encima de cada fotograma.

5.1. Detectar pose en cada fotograma
Añade el código a continuación para que el modelo pueda detectar los puntos clave de las manos en cada fotograma:

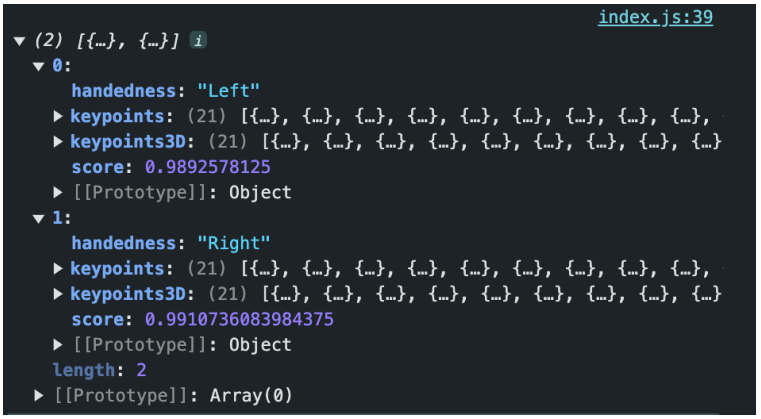
```
while (true) {
    const img = await webcam.capture();

// Prediction
   const hands = await detector.estimateHands(img, { flipHorizontal: true });


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

detector.estimateHands() es un método asincrónico que utiliza el detector creado previamente para analizar una imagen o un fotograma de video (en este caso el tensor en la variable img). El método devuelve un array de objetos, donde cada objeto representa una mano detectada en la imagen. Cada objeto contiene la información detallada al principio: **score**, **handedness**, **keypoints**, **keypoints3D**.

{ flipHorizontal: true } es un parámetro opcional que indica si se debe voltear horizontalmente la imagen antes de procesarla. Esto es útil cuando se utiliza una cámara que muestra una imagen en espejo. Al voltear la imagen horizontalmente, los resultados se alinean con la perspectiva del usuario.



Paso 6. Mostrar la salida de hand-pose-detection

Por último, necesitamos mostrar la salida del modelo, es decir los 21 puntos de cada mano detectada en los fotogramas.

6.1. Dibujar círculos para los puntos clave del cuerpo

Primero, para poder dibujar los puntos clave en los fotogramas, necesitamos mover el canvas a la misma posición del elemento **video** en la página HTML. Ponemos el siguiente código justo después de haber creado el objeto webcam en la función app():

```
const camerabbox = webcamEl.getBoundingClientRect();
canvas.style.top = camerabbox.y + "px";
canvas.style.left = camerabbox.x + "px";
```

A continuación, recuperamos el contexto del canvas para poder dibujar en ella:

```
const context = canvas.getContext("2d");
```

La imagen de la cámara web está rotada de 180 grados en el eje y para obtener una imagen espejo. Al proporcionar el modelo hand-pose-detection un parámetro opcional flipHorizontal, que hemos utilizado con el valor true, no necesitamos añadir más código.

Utilizamos la misma función drawCircle() del tutorial anterior.

```
function drawCircle(context, cx, cy, radius, color) {
 context.beginPath();
 context.arc(cx, cy, radius, 0, 2 * Math.PI, false);
 context.fillStyle = "red";
 context.fill();
 context.lineWidth = 1;
```

```
 context.strokeStyle = color;
 context.stroke();
}
```

Creamos una nueva función para dibujar círculos y también conexiones entre articulaciones, utilizando la información del diagrama de los puntos claves proporcionado por el modelo.

```
function drawHandLandmarks(context, landmarks) {
 const color = 'red';
 // Define the connections between landmarks (bones of the hand)
 const connections = [
   [0, 1], [1, 2], [2, 3], [3, 4],       // Thumb
   [0, 5], [5, 6], [6, 7], [7, 8],       // Index Finger
   [0, 9], [9, 10], [10, 11], [11, 12],  // Middle Finger
   [0, 13], [13, 14], [14, 15], [15, 16], // Ring Finger
   [0, 17], [17, 18], [18, 19], [19, 20]  // Pinky Finger
 ];

 // Draw lines for each connection
 connections.forEach(([start, end]) => {
   const startPoint = landmarks[start];
   const endPoint = landmarks[end];

   context.beginPath();
   context.moveTo(startPoint.x, startPoint.y); // Move to the start landmark
   context.lineTo(endPoint.x, endPoint.y);    // Draw to the end landmark
   context.lineWidth = 2;
   context.strokeStyle = 'blue';
   context.stroke();
 });

 // Draw circles for each landmark
 landmarks.forEach((landmark) => {
   drawCircle(context, landmark.x, landmark.y, 3, color);
 });
}
```

Luego, añadimos el siguiente código en el bucle de recuperación y detección, dentro de nuestra función app:

```
   …
   const hands = await detector.estimateHands(img, { flipHorizontal: true });

   context.clearRect(0, 0, canvas.width, canvas.height);
```

```
if (hands.length > 0) {
  hands.forEach(hand => {
    const landmarks = hand.keypoints;
    drawHandLandmarks(context, landmarks);
  });
}

img.dispose();
…
```

Ejercicios

1. Resaltar dedos específicos: modifica la función drawHandLandmarks para usar colores diferentes para cada dedo.
2. Añadir reconocimiento de gestos: implementa la detección de un gesto simple (por ejemplo, “pulgar arriba”) basado en las posiciones de los puntos clave.
3. Medir distancia: calcula la distancia entre la punta del pulgar y la punta del dedo índice y muéstrala en el lienzo.