uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática y Administración de Empresas

2020-2021

Trabajo Fin de Grado

"Reconocimiento Facial en el Navegador con Tensorflow.js y WebAssembly"

Pablo Escrivá Gallardo

Tutor

Dr. Ricardo Martín Manso

Madrid, junio 2021

RESUMEN

La presente Memoria corresponde al estudio sobre tecnologías de reconocimiento facial en el navegador web mediante el uso de redes neuronales. Se trata de un trabajo eminentemente práctico cuyo objetivo es analizar los últimos avances en métodos de ejecución de algoritmos de manera local a través de librerías de código abierto como TensorFlow.

Para el estudio se ha desarrollado una aplicación web que permita comparar WebGL, WebAssembly, JavaScript puro y Node.js en las mismas condiciones y poder obtener las métricas que indiquen la eficiencia de cada uno de ellos.

El experimento que se ha llevado a cabo incluye el uso de múltiples dispositivos (tanto móviles como de escritorio) que permiten obtener una representación lo más fiel posible del público general.

Finalmente se han expuesto los resultados obtenidos y, a través de éstos, se ha realizado una recomendación del uso de cada una de las tecnologías para diferentes requisitos de cada aplicación.

Palabras clave: TensorFlow, WebAssembly, WebGL, Reconocimiento Facial.

AGRADECIMIENTOS

En primer lugar, agradecer el esfuerzo y dedicación de mi tutor, Dr. Ricardo Martín Manso, que me ha acompañado durante los últimos meses en la elaboración de este proyecto. Además, me ha introducido al mundo de la investigación y la divulgación científica a través del artículo "*Face Recognition Efficiency Improvements Using TensorFlow's WebAssembly Backend: a practical approach"* presentado en la Conferencia Iberoamericana de Interacción Persona-Computadora (HCI 2021) [1], el cual hemos escrito de manera conjunta.

Por otro lado, dedico este TFG a mis abuelos, María del Carmen e Ignacio, sin los cuales no habría podido estudiar en la Universidad Carlos III de Madrid y que por desgracia ya no están aquí para ver a su nieto graduarse.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Estructura de la memoria

A continuación, se describen brevemente los apartados de los que consta este documento:

Estado del Arte: Se presenta la librería TensorFlow y Tensorflow.js, diferentes tecnologías web de ejecución de modelos neuronales en navegadores web y la librería face-api.js que servirá de apoyo para el desarrollo de la aplicación de pruebas.

Gestión del proyecto: Se describen las diferentes etapas del proyecto: análisis, diseño, desarrollo y experimentación a través de un diagrama de Gantt.

Diseño de la solución: Incluye la definición de requisitos funcionales y no funcionales de la aplicación de pruebas FaceApp, diagramas de secuencia de las principales operaciones que los usuarios podrán realizar y un diagrama simplificado de clases de la aplicación. Por último, se exponen las tecnologías y librerías a utilizar junto a otras decisiones de diseño y su justificación.

Desarrollo de la solución: Descripción del proceso de desarrollo de la aplicación, tanto de la parte del cliente como del servidor. Principales problemas encontrados y cómo se han solucionado, además de las aportaciones que han sido realizadas a proyectos de código abierto a lo largo del proyecto.

Experimentación: Se trata de la fase del proyecto en la que se comienza a utilizar la aplicación FaceApp para obtener resultados de la eficiencia de cada implementación de Tfjs para el uso de modelos neuronales en distintos dispositivos. Se describen los resultados obtenidos de manera objetiva a través de tablas y gráficos.

Marco Regulador: Descripción de la legislación vigente relacionada con el reconocimiento facial.

Entorno Socioeconómico: Presupuesto de la elaboración de este trabajo y estudio del impacto de las tecnologías de reconocimiento facial, especialmente en el segmento de la población de tercera edad.

Conclusiones y líneas futuras: A raíz de los resultados obtenidos en la etapa de experimentación se llega a una conclusión en cuanto a cuáles son los mejores modos de ejecución de modelos neuronales en navegadores web para el reconocimiento facial.

1.2. Motivación

El reconocimiento facial, junto con otros métodos de identificación biométricos, están cada vez más presentes en el día a día de las personas. Este crecimiento en su popularidad se ha visto incrementado en gran medida gracias a los avances en los campos de la inteligencia artificial, especialmente en redes neuronales que permiten solventar este tipo de problemas de una manera mucho más eficiente [2].

Los desarrolladores de software tradicionalmente han delegado estos procesos a servicios bajo demanda como Amazon Web Services o Microsoft Azure de forma que tengan a su disposición potentes servidores y únicamente pagar por la cantidad de usuarios que lo usen, reduciendo la cantidad de inversión inicial necesaria para desplegar aplicaciones que dependan de sistemas de reconocimiento facial [3].

Sin embargo, este tipo de arquitecturas pueden suponer un grave riesgo para la privacidad de los usuarios ya que es necesario realizar un envío de información sensible (por ejemplo, imágenes del rostro del usuario o coordenadas de puntos faciales) [4].

Por otro lado, el incremento de la capacidad de cómputo de los dispositivos y la aparición de nuevas tecnologías web hacen posible el reconocimiento facial en el lado del cliente, sin necesidad de utilizar potentes servidores. De esta manera, no solo se incrementa en gran medida la escalabilidad de las aplicaciones que dependen de este tipo de identificación de usuarios, sino que también se garantiza la seguridad y la privacidad de las personas. Así, el procesamiento a nivel local en los navegadores web cada vez se sitúa más como una alternativa real a la computación en la nube para aplicaciones que necesiten hacer uso del reconocimiento facial [5].

1.3. Objetivos

En primer lugar, se analizará el estado actual de las tecnologías de reconocimiento facial disponibles para su uso desde el navegador web, tanto en dispositivos móviles como de escritorio. Para ello se desarrollará una aplicación web que permita utilizar varias tecnologías web de interés para el tema y calcular su eficiencia mediante múltiples parámetros.

Dicha aplicación web servirá para llevar a cabo una serie de experimentos que permitan la comparación entre distintas implementaciones de la librería Tensorflow.js y poder determinar su viabilidad en aplicaciones en el mundo real.

A lo largo del proyecto se realizarán varias aportaciones a repositorios de código abierto, necesarias para poder probar el funcionamiento de tecnologías experimentales muy relevantes para el futuro del reconocimiento facial en la web y a su vez contribuir al desarrollo y avance de esta área.

2. ESTADO DEL ARTE

2.1. Introducción al reconocimiento facial

A la hora de discutir sobre la seguridad en sistemas electrónicos e internet es importante saber diferenciar entre tres términos que a menudo se usan de manera indistinta: Autenticación, Identificación y Autorización. **Autenticar** a un usuario significa darle acceso a determinados recursos, lo que implica que necesita algún tipo de verificación de identidad previo. **Identificar** es el acto de asignar la autoría de un recurso o proceso a una persona. Por último, **Autorizar** es asignar privilegios a un usuario previamente autenticado [6]. Este documento se centrará, principalmente, en la identificación de usuarios mediante el reconocimiento.

Tradicionalmente se han utilizado métodos de identificación de personas mediante contraseñas o pines, que responden a la pregunta de: ¿Qué es lo que sabes?. Como forma de mejorar este tipo de identificación, se comienzan a usar técnicas basadas en *tokens* o objetos únicos que (en teoría) sólo puede poseer su dueño. Estos *tokens* pueden tratarse de tarjetas de identidad o llaveros USB y responden a la segunda pregunta básica: ¿Qué es lo que tienes? [7][1].

Sin embargo, los dos métodos anteriormente descritos tienen un fallo fundamental que les hace vulnerables a ser utilizados de manera fraudulenta, y es que no poseen ninguna característica inherente del individuo y pueden ser perdidos, olvidados o incluso replicados por terceros [8].

Por ello se comienza a usar sistemas más avanzados de identificación, basados en las características físicas o conductuales de las personas, los cuales solucionan son universales (todas las personas lo tienen), únicos y permanentes (los rasgos que se estudian apenas cambian con la edad) [8].

Los principales motivos por los cuales la identificación biométrica no se ha generalizado son la dificultad de obtener los rasgos biométricos[2] y la amenaza a la privacidad de las personas [9]. Sin embargo, el caso de la identificación biométrica basada en reconocimiento facial puede solucionar

[1] Estos dos métodos de autenticación de usuarios también se conocen como "factor único" y "doble factor" en los casos en los que se usan de manera conjunta (por ejemplo, una contraseña y un código de único uso autogenerado).
[2] Por ejemplo, el análisis de iris o de ADN necesitan equipos muy complejos y costosos.

parte de estos problemas dado que sólo es necesaria una cámara web y la información de los rasgos faciales puede estar almacenada en el propio dispositivo del usuario.

2.2. Tecnologías de aprendizaje automático en el navegador

2.2.1. TensorFlow y Tensorflow.js

TensorFlow [10] es una plataforma de código abierto, desarrollado por Google, que consta de una serie de herramientas y librerías para el desarrollo de proyectos que usen aprendizaje automático. Fue desarrollado en 2015 y su nombre hace referencia a los "Tensores", arrays multidimensionales de datos que forman la unidad básica de información a la hora de realizar operaciones con redes neuronales.

Una de las implementaciones de TensorFlow es Tensorflow.js (en adelante, Tfjs), que permite el entrenamiento, despliegue y uso de redes neuronales directamente en el navegador web, utilizando la interfaz de programación de TensorFlow en lenguajes de entornos web como JavaScript, Node.js o recientemente WebAssembly [11].

Entre las ventajas de usar Tfjs se encuentra la posibilidad de ser utilizado en multitud de entornos y arquitecturas, como servidores, clientes web o incluso aplicaciones de escritorio[3]. De esta forma, las aplicaciones que necesiten el uso de redes neuronales (por ejemplo, reconocimiento facial) pueden ser distribuidas de una manera mucho más sencilla y segura que en el caso de TensorFlow con Python [12].

2.2.2. Implementaciones de Tensorflow.js

Tfjs aprovecha todas las capacidades de los navegadores y dispositivos actuales para poder alcanzar la mayor eficiencia posible cuando se ejecutan modelos de aprendizaje automáticos. La librería ha sido diseñada de manera modular, de modo que es posible elegir de qué manera ejecutar los modelos, dependiendo de cada situación.

[3] Tfjs puede usarse en aplicaciones web con librerías como Electron.js que permiten convertirlas en aplicaciones de escritorio nativas [9].

Una de las opciones que ofrece la librería es la de seleccionar cuál es la implementación[4] de Tfjs de un modelo neuronal. De esta manera se puede mejorar la experiencia de usuario al utilizar las tecnologías web más adecuada para cada caso.

En la actualidad, Tfjs puede ejecutar modelos neuronales de tres maneras distintas:

- Utilizando la CPU del dispositivo, mediante la interpretación de las instrucciones escritas en lenguaje JavaScript. Se trata del método menos eficiente, especialmente debido a que JavaScript es un lenguaje que no permite la ejecución multi-hilo. Por otro lado, es la opción con mayor compatibilidad con distintos dispositivos.
- Convirtiendo las instrucciones a bytecode mediante la tecnología WebAssembly (WASM) y ejecutándolas en la CPU, mejorando en gran medida la eficiencia en comparación con el caso anterior (interpretación). Esta opción está actualmente limitada en algunos dispositivos (especialmente móviles) dado que la tecnología WASM está aún en proceso de desarrollo y adopción.
- Por último, utilizando la GPU del dispositivo para interpretar el código JavaScript mediante el uso del estándar WebGL[5]. Esta alternativa es la más utilizada actualmente debido a que reduce en gran medida el tiempo de cálculo de las operaciones matemáticas que intervienen en los modelos neuronales, además de ser una tecnología más extendida que WASM actualmente [12]. Sin embargo, está solamente indicado para dispositivos con tarjetas gráficas relativamente potentes.

2.2.3. WebAssembly

WebAssembly es un estándar abierto de instrucciones en el navegador web que permite alcanzar velocidades de ejecución de modelos neuronales muy cercanas a las aplicaciones nativas escritas en C [2]. Con WebAssembly es posible compilar código C o C++ en un lenguaje similar a ensamblador mediante el uso de un compilador como Emscripten. Proyectos como Google Earth o Figma utilizan WebAssembly para ofrecer una experiencia similar a las aplicaciones de escritorio en la web.

[4] "Implementación" es la traducción del término anglosajón "Backend".

[5] WebGL es una interfaz de programación de código abierto y multi-plataforma para la generación de gráficos en 3 dimensiones [9].

El proceso de desarrollo de WebAssembly se lleva a cabo mediante propuestas a través del repositorio Github oficial de WASM. En este sentido, cada propuesta pasa por un total de cuatro fases, siendo la última de ellas la estandarización de la propuesta por los navegadores web [13].

Una de las propuestas más importantes para WebAssembly es la de la adopción de la especificación SIMD (Single Instruction Multiple Data). Se trata de un juego de instrucciones del procesador que permite realizar operaciones sobre multitud de datos de manera simultánea mediante vectores y así acelerar la ejecución de algoritmos [14]. La propuesta de adopción SIMD se encuentra actualmente en la fase 4 del proceso de desarrollo de WebAssembly, es decir, está en proceso de estandarización.

Otra de las propuestas actuales de WebAssembly es la incorporación de threads a través de webworkers de JavaScript y así ejecutar algoritmos de forma paralela. En este caso se encuentra en la segunda fase, por lo que todavía faltan unos meses para ser desplegada.

SIMD y MT no están disponibles de manera generalizada en los navegadores web, aunque se pueden activar de manera experimental. Google Chrome, por ejemplo, lo permite a través de las flags "#enable-webassembly-simd" y "#enable-webassembly-threads" en Chrome://flags.

2.2.4. face-api.js

face-api.js es una librería de código abierto desarrollada por Vincent Mühler que utiliza la API de Tfjs para el reconocimiento facial en el navegador web [15]. Para ello incluye una serie de redes neuronales pre-entrenadas muy ligeras que permiten ser ejecutadas de manera relativamente eficiente en entornos web.

Cada red neuronal está diseñada para un fin específico, como la detección de una o múltiples caras en una imagen, la generación de puntos faciales, reconocimiento facial a partir de descriptores faciales o incluso la detección del sexo, edad y expresión facial de una persona.

El principal objetivo de esta librería es proporcionar una interfaz más reducida que la de Tfjs y liberar al desarrollador de la carga de crear y entrenar modelos neuronales, de forma que se puedan crear aplicaciones que usen el reconocimiento facial de manera mucho más rápida y sencilla.

2.2.5. face-api.js en el cliente y en el servidor

face-api.js también puede ser ejecutado en el servidor mediante el uso de Node.js y su gestor de dependencias, NPM. De esta manera se puede liberar al cliente de la carga de ejecutar los modelos con Tfjs y dotar al servidor de un hardware más potente para ejecutar las operaciones necesarias.

Sin embargo, esta arquitectura puede provocar un cuello de botella al tener que gestionar múltiples peticiones de diferentes clientes de manera simultánea, además de depender del protocolo HTTP para la comunicación con el cliente, que incrementa los milisegundos entre cada fotograma procesado.

Por otro lado, ejecutar los modelos neuronales en el cliente tiene una serie de ventajas, como por ejemplo la privacidad, dado que los datos biométricos del usuario no tienen que ser mandados al servidor.

2.3. Uso de modelos neuronales

face-api.js contiene una serie de modelos neuronales pre-entrenados para ser utilizados con diversos fines, por ejemplo, para reconocimiento de expresiones faciales o estimación de la edad de la persona. En este proyecto únicamente se utilizarán los modelos estrictamente necesarios para poder implementar una solución de reconocimiento facial a través de imágenes de una cámara web. A continuación, se detallan las características de cada uno de los modelos.

2.3.1. Face Recognizer Net

Esta red neuronal está diseñada para generar vectores de características (128 números) a partir de los rasgos faciales de una persona. Estos vectores son una representación numérica de la forma y características del rostro de la persona, los cuales se guardarán para poder compararlos con vectores extraídos de otras imágenes.

La arquitectura de esta red neuronal convolucional de 29 capas de tipo residual basadas en la red ResNet-34[6]. Este tipo de arquitecturas se utilizan en redes muy profundas con "propagación hacia atrás[7]" para intentar reducir la pérdida de eficiencia de la red neuronal.

En la fase de entrenamiento, el autor de la librería face-api.js utilizó alrededor de 3 millones de imágenes de personas, tras la cual midió la precisión y se obtuvo un valor de 0,99 con una desviación típica de 0.0027 [15]. La red neuronal ocupa un total de 6.2 MB en la librería de face-api.js.

Para reconocer a una persona, esto es, medir el parecido de ésta con otra que haya sido registrada previamente, se usará el método de la distancia euclídea entre dichos vectores y se extraerá un nivel de confianza en forma de porcentaje. Por último, se establece un nivel mínimo para determinar que una persona ha sido reconocida por la aplicación (dependiendo del nivel de confianza que requiera cada caso de uso).

2.3.2. Tiny Face Detector

Tiny Face Detector es un modelo neuronal que permite detectar rasgos faciales en tiempo real. Está orientado a dispositivos móviles o con una capacidad de procesamiento limitado. Además, su pequeño tamaño (190 KB) lo hace ideal para ser utilizado en la web. El entrenamiento de esta red ha sido realizado con un conjunto de 14.000 imágenes por parte del autor de Face-Api.js [15].

El modelo permite extraer la parte de una imagen en la que se encuentra el rostro de una persona. Esto es muy útil de por sí, dado que puede dibujar un recuadro alrededor de la cara. Sin embargo, en el contexto de este proyecto se ha usado en conjunto con la red neuronal "Face Landmark Model" para mejorar la precisión de la última.

2.3.3. Face Landmark Model

Esta red neuronal convolucional de ocho capas permite extraer de una imagen de un rostro un vector de 68 puntos faciales o "descriptores faciales" [16]. Dada la complejidad de esta tarea, la red necesita pasar previamente por el detector de rostros del apartado anterior

[6] Algunas de las capas se eliminaron de la red original, pasando de 34 a 29.
[7] Traducción del inglés "Backpropagation"

Los autores de la red neuronal han realizado el entrenamiento con un total de 35.000 imágenes previamente etiquetadas con los 68 puntos faciales. Dichas imágenes proceden de la base de datos ibug 300-W [16].

Fig. 1. DISTRIBUCIÓN DE LOS 68 PUNTOS FACIALES UTILIZADOS POR LAS REDES NEURONALES PARA EL RECONOCIMIENTO FACIAL.

3. GESTIÓN DEL PROYECTO

El desarrollo de la aplicación de pruebas de reconocimiento facial y la subsiguiente experimentación se dividirá en las siguientes tareas representadas a través de un diagrama de Gantt.

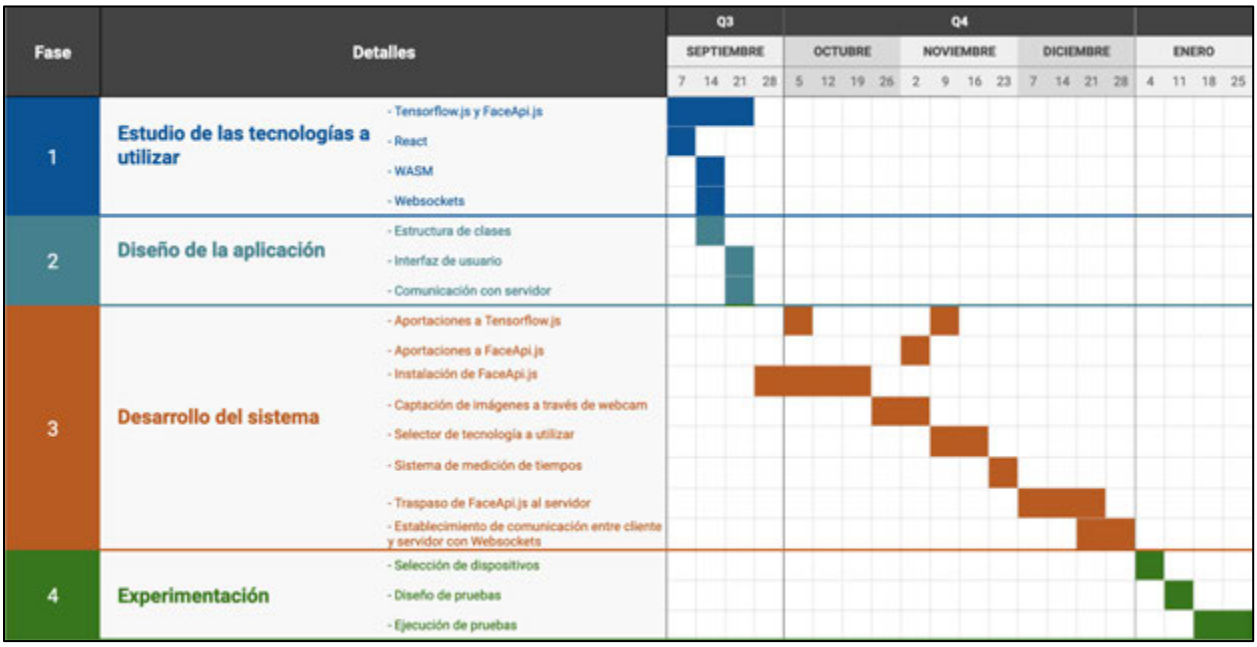

Fig. 2. DIAGRAMA DE GANTT DEL PLAN DE DESARROLLO DE LA APLICACIÓN FACEAPP.

Las fases de desarrollo de la aplicación FaceApp y el subsiguiente proceso de experimentación son las siguientes:

1. Estudio de las tecnologías a utilizar
    a. Tensorflow.js y face-api.js
    b. React
    c. WASM
    d. WebSockets
2. Diseño de la aplicación
    a. Estructura de clases
    b. Interfaz de usuario
    c. Comunicación con servidor
3. Desarrollo del sistema
    a. Aportaciones a proyectos de código abierto
        i. Tensorflow.js

    ii. face-api.js
  b. Cliente
    i. Instalación de face-api.js
    ii. Captación de imágenes a través de webcam
    iii. Selector de tecnología a utilizar
    iv. Sistema de medición de tiempos
  c. Servidor
    i. Traspaso de face-api.js al servidor
    ii. Establecimiento de comunicación entre cliente y servidor con Websockets
4. Experimentación
  a. Selección de dispositivos
  b. Diseño de pruebas
  c. Ejecución de pruebas

Los primeros dos meses de trabajo se han empleado casi en su totalidad al aprendizaje y la familiarización con las herramientas y tecnologías necesarias durante todo el proyecto. Para ello se ha comenzado con el desarrollo de pequeños prototipos siguiendo los tutoriales de Tfjs y de face-api.js hasta conseguir que funcionaran correctamente. A continuación, se han utilizado los diferentes modos de ejecución de Tfjs, con especial atención a WASM dado que en ese momento no funcionaba correctamente con face-api.js.

A su vez, se ha diseñado la aplicación FaceApp que sirve para realizar todas las pruebas necesarias para poder extraer las conclusiones de este proyecto. Primero se han diseñado los distintos componentes del proyecto y cómo interactúan entre sí (especialmente Tfjs y face-api.js). También se ha estudiado la posibilidad de poder desacoplar la implementación de todos los componentes relacionados con el reconocimiento facial de forma que pueda ser ejecutado en el cliente o en el servidor indistintamente. Además, se ha elegido entre la comunicación con el servidor mediante websockets o llamadas HTTP, eligiendo finalmente la primera de ellas.

A continuación, comienza el desarrollo de la aplicación FaceApp basada en los prototipos del mes anterior. A lo largo del desarrollo se han realizado una serie de aportaciones a repositorios de código abierto que se detallan más adelante en este documento.

Primero, se ha desarrollado una versión básica de la aplicación en la que el usuario sube manualmente imágenes y se pasan por la red neuronal. Después se añade la posibilidad de acceder a la cámara web del dispositivo para extraer fotogramas y ver en tiempo real la identificación facial.

Para poder elegir entre los distintos modos de ejecución, se desarrolla un selector que permite al usuario cambiarlo de forma sencilla (previamente había que modificar el código fuente de la aplicación), además de añadir el sistema de medición de tiempos para poder extraer estos datos y hacer comparativas.

El último mes de desarrollo se enfoca a replicar el comportamiento de FaceApp en el servidor, creando un segundo repositorio (FaceApp-Server) y subiéndolo a Heroku. Finalmente se establece la conexión entre FaceApp y FaceApp-Server y se comienza con la última etapa del proyecto: la experimentación.

Durante todo el mes de enero se han realizado pruebas a los distintos dispositivos con la aplicación FaceApp, anotando los resultados en una hoja Excel.

4. DISEÑO DE LA SOLUCIÓN

4.1. Planteamiento

El propósito principal del desarrollo de la aplicación FaceApp es permitir el análisis de las distintas tecnologías disponibles para el reconocimiento facial y determinar cuales son las más convenientes para su uso en navegadores web, especialmente en dispositivos móviles y/o con pocos recursos de hardware.

Esta aplicación web permitirá medir los fotogramas por segundo para el reconocimiento facial a través de una cámara, además de poder cambiar fácilmente la implementación de Tfjs para el uso de los modelos neuronales (WebGL, WASM o CPU).

4.2. Requisitos funcionales y no funcionales

4.2.1. Requisitos No Funcionales

ID	RNF-01
Nombre	Disponibilidad
Descripción	La aplicación deberá funcionar en dispositivos móviles (iOS y Android) y de escritorio mediante el navegador Google Chrome en su versión 86.0 o superior.

ID	RNF-02
Nombre	Guardado de datos de reconocimiento facial
Descripción	Los descriptores faciales extraídos durante el registro de personas serán almacenados en el dispositivo a través del uso de cookies del navegador.

4.2.2. Requisitos Funcionales

ID	RF-01
Nombre	Acceso al hardware
Descripción	La aplicación deberá ser capaz de acceder a la cámara web del dispositivo.

ID	RF-02
Nombre	Registro de usuario
Descripción	El usuario podrá subir una o varias imágenes de una persona a la aplicación, además de su nombre, para que sea registrada y pueda ser reconocida mediante el uso de modelos neuronales.

ID	RF-03
Nombre	Selección de tecnología a utilizar
Descripción	El usuario podrá cambiar la tecnología utilizada para ejecutar los modelos neuronales de entre las siguientes opciones: WebGL, WASM, CPU o Servidor.

ID	RF-04
Nombre	Uso de servidor externo
Descripción	La aplicación permitirá la ejecución de los modelos neuronales en el servidor además de en el propio cliente.

4.3. Diagramas de secuencia

A continuación, se definirán los diagramas de secuencia para las principales operaciones que se llevarán a cabo en la aplicación FaceApp durante la fase de experimentación.

4.3.1. Registro de persona

Tanto en el caso del reconocimiento facial en el navegador como en el servidor es necesario haber obtenido previamente los descriptores (vectores de coordenadas de puntos faciales) del rostro de al menos un usuario. Estos descriptores se guardan en ambos casos en el almacenamiento del propio navegador, de esta forma se evita tener que almacenarlos en una base de datos externa y se protege la privacidad de la persona.

En este caso el usuario selecciona una o varias imágenes de la persona a registrar y se especifica su nombre. Las imágenes se pasan a *face-api.js* donde primero se extrae la cara más reconocible

de la imagen (en caso de que aparezca más de una persona en ella) y se calculan los 128 puntos faciales necesarios para reconocer a la persona.

Estos puntos después se guardan en el almacenamiento interno para poder ser utilizados posteriormente.

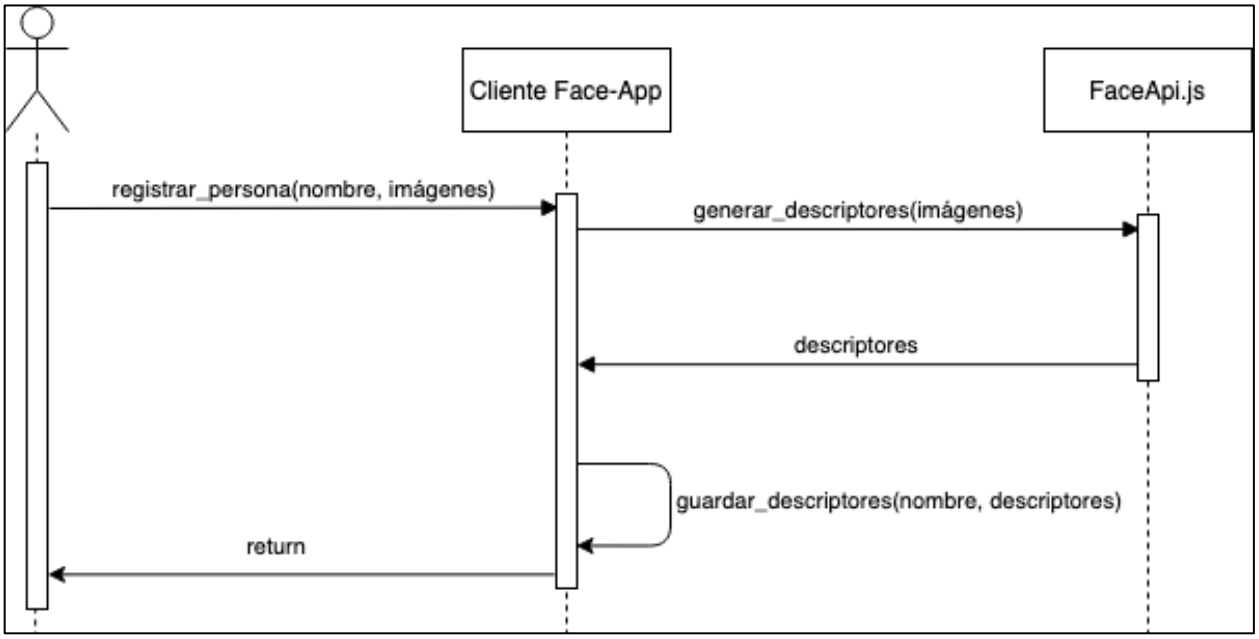

Fig. 3. DIAGRAMA DE SECUENCIA PARA EL REGISTRO FACIAL DE UN USUARIO.

4.3.2. Reconocimiento facial en el navegador

Para reconocer a una persona, lo primero que se hará es acceder a la cámara web (después de que el usuario apruebe explícitamente que la aplicación puede acceder a ella). Se extraerá un fotograma de la cámara, además de los descriptores faciales que estén almacenados previamente en el almacenamiento del navegador.

Con esta información se solicitará a *face-api.js* que compare los descriptores con los puntos faciales que se obtengan del fotograma proporcionado.

Por último, se dibujarán los resultados en un canvas HTML (el fotograma original con un recuadro que señale la cara y el nombre de la persona, en el caso de que haya sido reconocida).

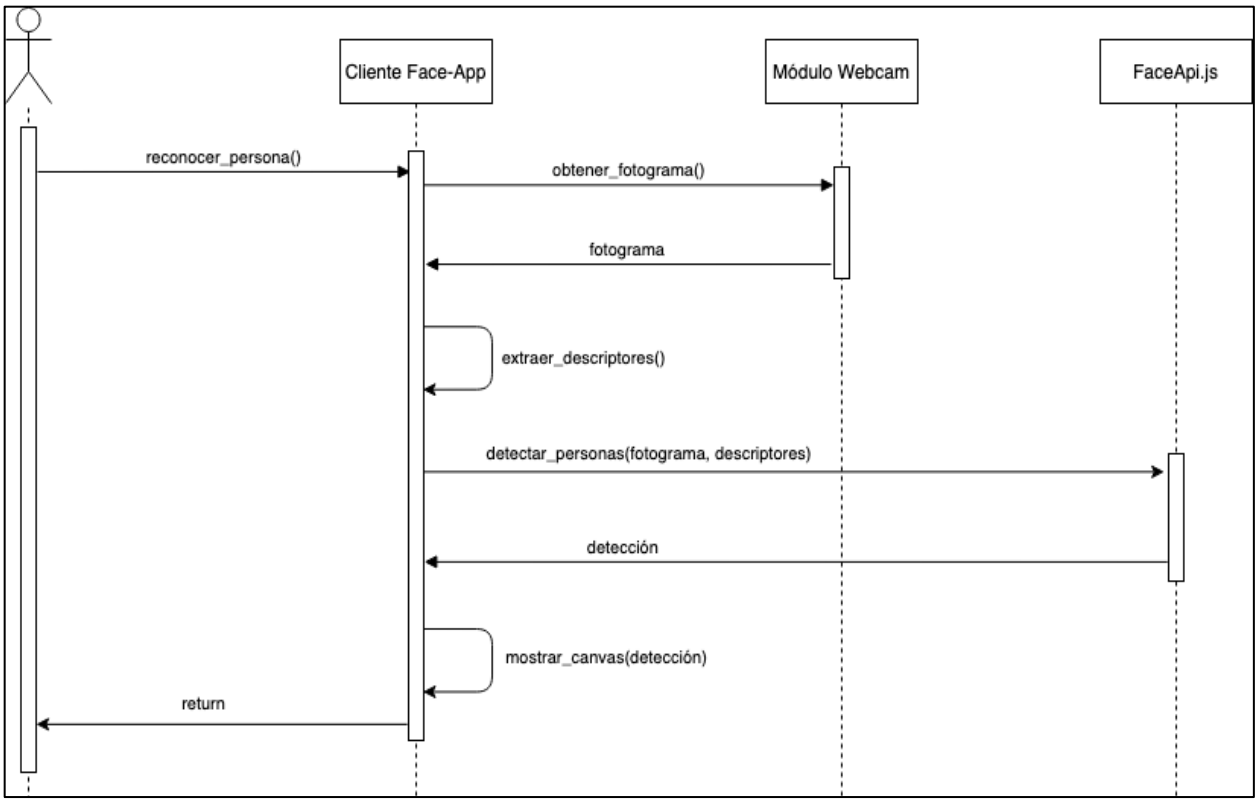

Fig. 4. DIAGRAMA DE SECUENCIA PARA EL RECONOCIMIENTO FACIAL DE UN USUARIO EN EL NAVEGADOR.

4.3.3. Reconocimiento facial en el servidor

Los pasos para reconocer a una persona en el servidor son muy similares que en el navegador. En este caso el grueso de las operaciones computacionales se realiza en el servidor, de modo que el cliente no tiene apenas carga de trabajo.

Para reducir la latencia en la comunicación entre el cliente y el servidor se usará el protocolo *WebSockets* que crea un canal bidireccional persistente a través de un socket TCP y reduce los milisegundos entre el envío de fotogramas por el cliente y la respuesta del servidor con los datos del reconocimiento facial.

Los descriptores faciales se extraen del navegador y se envían al servidor antes de proceder a enviar fotogramas. El servidor almacena temporalmente estos descriptores y los elimina cuando se cierra la conexión mediante Websockets con el cliente.

A continuación, el proceso es análogo al caso anterior, se extrae un fotograma de la cámara web, se envía al servidor y se reciben las posibles detecciones de personas que después se muestran al usuario mediante un canvas HTML.

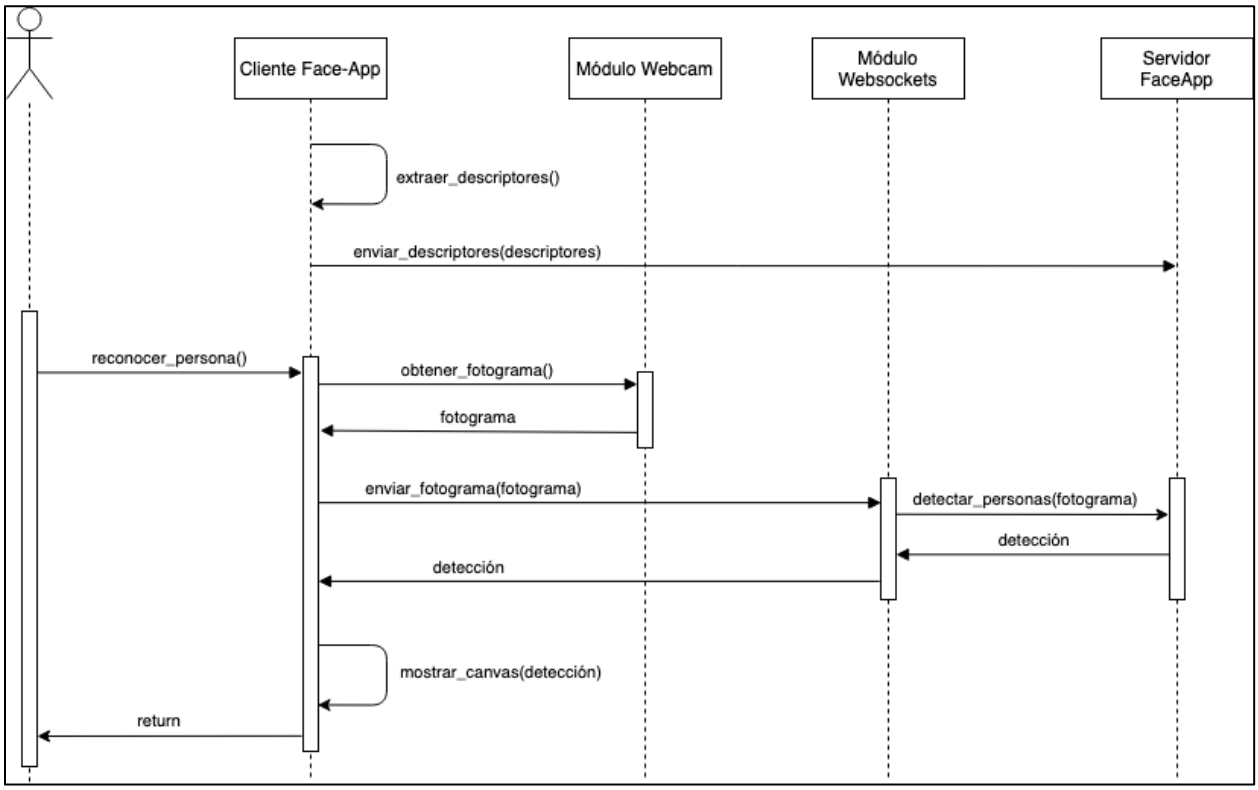

Fig. 5. DIAGRAMA DE SECUENCIA PARA EL RECONOCIMIENTO FACIAL DE UN USUARIO EN EL SERVIDOR.

4.4. Arquitectura del sistema

La aplicación FaceApp se compondrá de cuatro paquetes: scripts, estilos, componentes y dependencias.

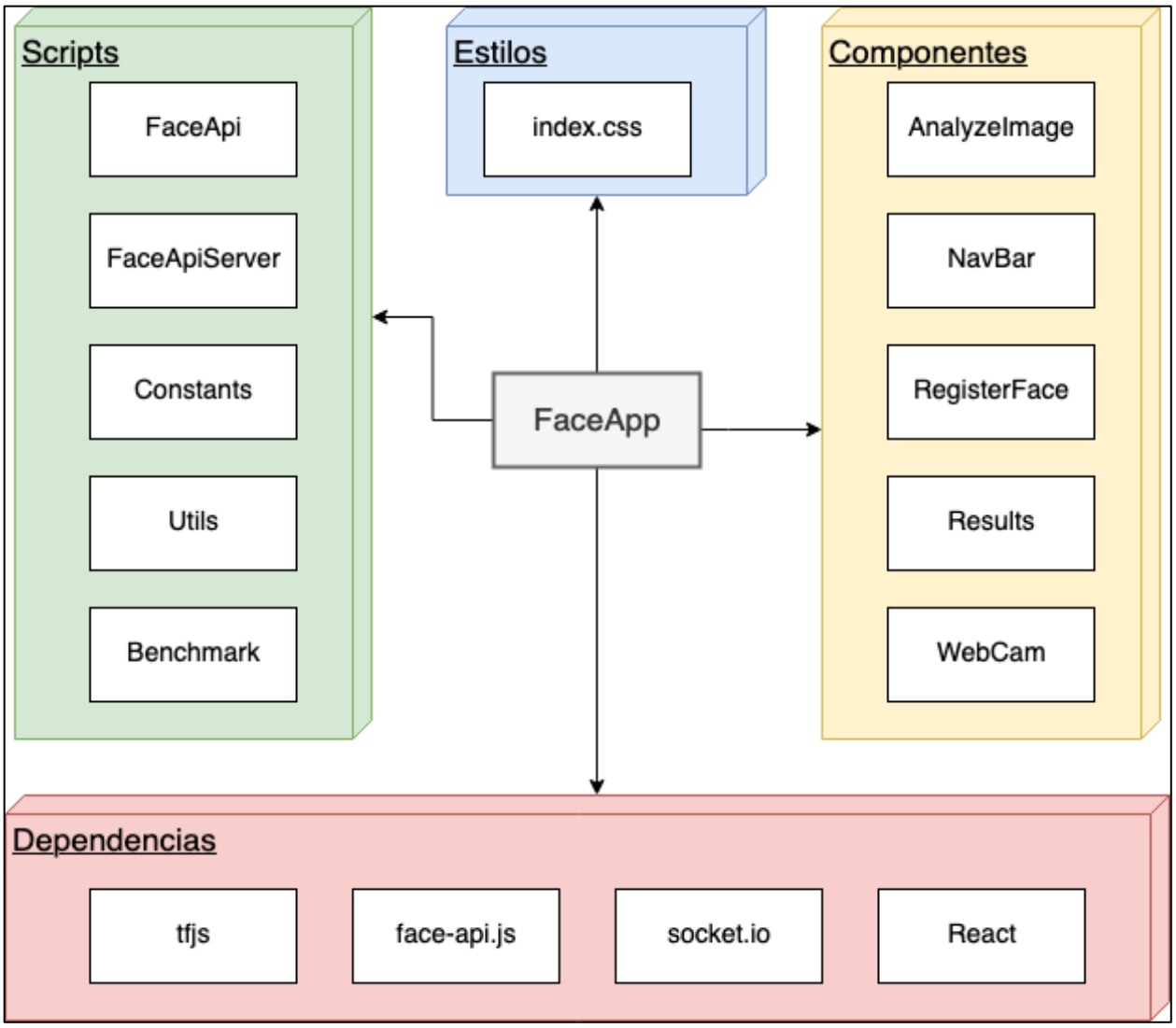

Fig. 6. DIAGRAMA DE COMPONENTES DE LA APLICACIÓN FACEAPP.

El paquete scripts contiene todo el código JavaScript con la lógica del sistema. Está compuesto por los siguientes ficheros:

- FaceApi[8]: Engloba todas las funciones relacionadas con la librería face-api.js, por ejemplo, la detección de rostros de imágenes, el registro de personas a través de sus descriptores faciales o el reconocimiento facial.
- FaceApiServer: Se encarga de toda la comunicación con el servidor FaceAppServer a través de websockets. Para ello hace uso de la dependencia Socket.io.

[8] No confundir con la librería face-api.js.

- Constants: Se trata de un fichero que contiene valores constantes, por ejemplo, el número de puntos faciales que se van a utilizar o la distancia euclídea mínima para reconocer a una persona. También incluye las direcciones URL del servidor FaceAppServer.
- Utils: Contiene funciones auxiliares, por ejemplo, el dibujado de resultados en la interfaz de usuario o la gestión de datos de usuario en el almacenamiento local.
- Benchmark: Incluye las funciones para calcular los tiempos y extraer datos sobre la eficiencia de cada implementación de Tfjs.

El paquete de componentes contiene las clases básicas que componen una aplicación React. Cuentan con fragmentos de la interfaz de usuario, además de la lógica que les da su utilidad. Los componentes son:

- AnalyzeImage (Analizar imagen): Se trata de un formulario que permite al usuario procesar imágenes de una a una, sin necesidad de acceder a la cámara web.
- NavBar (Barra superior): Es la barra de navegación de la parte superior de la aplicación, que cuenta con el selector de modos de ejecución.
- RegisterFace (Registrar rostro): Formulario para registrar a una persona en la aplicación. Permite seleccionar una o varias imágenes de la persona y añadir su nombre. A continuación, sus descriptores faciales se guardarán en el almacenamiento local del navegador, identificados por su nombre para poder reconocer a la persona más adelante.
- Results (Resultados): Para poder mostrar los resultados del reconocimiento facial es necesario superponer un canvas con los recuadros y los nombres de las personas reconocidas sobre la imagen original. En este fichero se genera el resultado de esta superposición y se muestra en la interfaz de usuario.
- WebCam (Cámara web): Solicita el acceso a la cámara del dispositivo y manda cada fotograma a FaceApi para ser procesado mediante la API de *WebRTC* [17].

Todos los componentes anteriores contienen la estructura y lógica de la interfaz, pero carecen de estilos CSS. Para ello, se usa el paquete de estilos que define la presentación de la aplicación web.

Las dependencias se gestionan a través del gestor NPM, el cual descarga el código fuente de cada una de ellas. La aplicación FaceApp cuenta con muchas dependencias, pero las más importantes son:

- face-api.js: librería que contiene modelos neuronales previamente entrenados para el reconocimiento facial, además de funciones que simplifican su uso.
- Tfjs: versión para el navegador de TensorFlow, que permite usar modelos neuronales en el navegador web. Es usado por face-api.js.
- socket.io: Se usa cuando la librería face-api.js se ejecuta en el servidor FaceAppServer para tener una comunicación en tiempo real entre el cliente y el servidor a través de WebSockets.
- React: biblioteca JavaScript que facilita la creación de interfaces de aplicaciones de una sola página.

4.5. Tecnologías

Para el alojamiento de la aplicación web se ha utilizado el servicio gratuito de *GitHub Pages* que permite publicar en la web aplicaciones estáticas. Para ello se ha creado un repositorio para el código fuente del cliente y se diseñará un sistema de despliegue continuo que será accionado por cada cambio en el repositorio. En este caso se ha usado el servicio *GitHub Actions,* que generará una versión de producción de la aplicación y la hará disponible automáticamente a los usuarios.

En cuanto al servidor, se ha desarrollado en el entorno *Node.js* para mantener la consistencia con el cliente en cuanto a lenguaje de programación (JavaScript) y las dependencias utilizadas. El código del servidor se ejecuta en una máquina gratuita proporcionada por el servicio *Heroku,* que está vinculada al repositorio de *GitHub* y publica los cambios que se realizan al código fuente.

5. DESARROLLO DE LA SOLUCIÓN

5.1. Cliente

Durante el desarrollo de la aplicación se han tomado una serie de decisiones que afectaron al resultado final de la aplicación.

En cuanto a las herramientas que ofrece la aplicación, se decide eliminar la de "Analizar imagen" dado que no aportaba utilidad durante la fase de experimentación. La herramienta de "Webcam" permite la obtención de tiempos de manera más precisa dado que tiene en cuenta cientos de fotogramas y puede extraer la media entre todos ellos.

Por otro lado, el componente de la interfaz que muestra los tiempos por fotograma pasa a situarse debajo de la imagen de la webcam, a diferencia de en la barra superior. Esto es debido a que el texto ocupa mucho espacio y la barra superior ya no lo mostraba correctamente, especialmente en dispositivos móviles o con baja resolución de imagen.

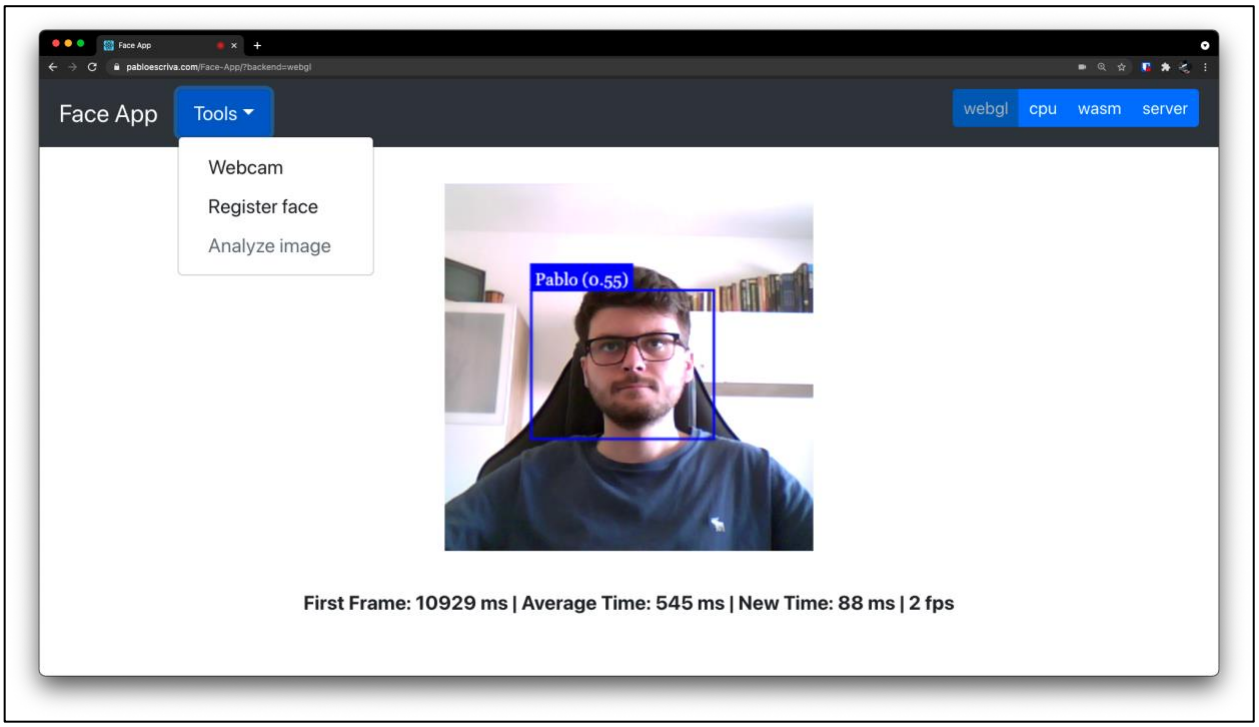

Fig. 7. CAPTURA DE PANTALLA DE LA APLICACIÓN FACEAPP.

El selector de implementaciones de Tfjs de la barra superior funciona modificando la URL de la aplicación, en concreto modificando el parámetro "backend" a CPU, WebGL, WASM o servidor mediante la siguiente línea de código:

```
const BACKEND = new
URLSearchParams(window.location.search).get('backend') ||
'webgl';
```

Fig. 8. FRAGMENTO DE CÓDIGO PARA LA SELECCIÓN DE IMPLEMENTACIÓN DE TFJS.

Además, si no hay ningún valor para este parámetro, se selecciona la opción WebGL por defecto.

La aplicación se encuentra actualmente disponible en la siguiente URL con soporte para Google Chrome y Safari:

https://www.pabloescriva.com/Face-App

En cuanto al código fuente, se encuentra en el siguiente repositorio de Github:

https://github.com/pabloegpf1/Face-App

Dicho repositorio cuenta con una herramienta de despliegue continuo, de manera que cada vez que se realiza un cambio en el código fuente se genera una versión de producción de la aplicación y se sirve a los usuarios de manera instantánea.

5.2. Servidor

El código fuente del servidor se encuentra en un repositorio distinto en la siguiente URL:

https://github.com/pabloegpf1/Face-App-Server

El alojamiento de dicho servidor se ha realizado a través del servicio Heroku, el cual también permite usar herramientas de despliegue continuo si se vincula a un repositorio de Github.

Para interactuar con el servidor es necesario establecer una conexión mediante Websockets, la cual se mantendrá abierta durante toda la sesión. Para ello se hace uso del módulo Socket.io, tanto en el cliente como en el servidor.

5.3. Aportaciones a repositorios de código abierto

Dado que este es un proyecto muy ligado a librerías de código abierto, se han realizado varias aportaciones a Tfjs y face-api.js. La primera de ellas es una simple corrección de erratas en una parte del código que, aunque no afectaba a la funcionalidad de la librería, sirvió para conocer de primera mano cómo realizar este tipo de aportaciones en el futuro [18]. Para ello fue necesario clonar el proyecto, modificarlo, ejecutar las pruebas, crear la solicitud de modificación en GitHub y darse de alta como desarrollador de un proyecto de Google.

La segunda aportación se ha realizado a la librería face-api.js en forma de solicitud al desarrollador para arreglar un fallo que hacía imposible el uso de WASM con la versión 2.0 de Tfjs. Finalmente, el desarrollador de Tfjs modificó la librería para poder usarla temporalmente dado que el problema no era de face-api.js sino de Tfjs y sugirió que proporcionase un parche por mi cuenta [19].

Por último, en relación con el problema del párrafo anterior, se modificó la librería Tfjs para solucionar el problema con WASM. Se trata de un pequeño fallo de lógica en la que la interfaz de programación con WASM es ligeramente distinta que con el resto de los modos de ejecución, de forma que al cambiar entre ellos producía un error. Esta lógica reside en un “switch” que devuelve el tipo de una variable en la que la opción “default” resulta en un error en vez de devolver el valor correcto (Float32). Esta aportación todavía no ha sido aceptada por el equipo de Tfjs [19].

6. EXPERIMENTACIÓN

6.1. Diseño del banco de pruebas

Para poder comprobar la eficiencia de WebGL, WASM, CPU y servidor para el reconocimiento facial con modelos neuronales se ha llevado a cabo una serie de pruebas en dispositivos de diferentes características. Se han seleccionado dispositivos móviles y de escritorio con hardware y sistemas operativos distintos para abarcar el máximo número de casos típicos. A continuación, se detallan las características de dichos dispositivos:

Tabla 1. DESCRIPCIÓN DE LOS DISPOSITIVOS DE PRUEBA.

	CPU	GPU	RAM
MacBook Pro 13	2,9 GHz Intel Core i5	Intel Iris Graphics 550 1536 MB	8 GB 2133 MHz LPDDR3
iPhone X	2,39 GHz Hexa-core	Apple GPU de tres núcleos	3GB LPDDR4
Portátil ASUS	Intel Core i5	Integrada Intel GMA HD Graphics 766 MHz	8GB DDR3 1066 MHz
Sobremesa Windows	3,9GHz AMD Ryzen 3 3100	GTX 970 4GB GDDR5 7 GHz	16GB DDR4
Intel NUC	3 GHz Intel Core i3 8109U	Iris Plus Graphics 655	16GB DDR4-2400
iMac 2010	2.8GHz Core i5	ATI Radeon HD5750 1024 MB	10GB DDR3 1333 MHz
iPad Air 2	1.5GHz A8X 64-bit ARM	PowerVR GXA6850 450 MHz	2GB LPDDR3 RAM

6.2. Mecanismos

La aplicación FaceApp muestra la información necesaria para comparar las distintas tecnologías de reconocimiento facial. Para ello se ha establecido un umbral del 60% de similitud para el reconocimiento facial.

En concreto, muestra los siguientes datos:

- **Tiempo para cargar el primer fotograma**: generalmente es mucho mayor que el resto dado que incluye el tiempo de carga inicial.
- **Tiempo medio por fotograma**: media de todos los fotogramas obtenidos y que hayan resultado en el reconocimiento de una persona (esto es para reducir la varianza al eliminar los fotogramas en los que no se ha encontrado ningún rostro). Para obtener este dato se suma el total de milisegundos de todos los fotogramas previos y se divide entre el número total de fotogramas.
- **Tiempo actual**: los milisegundos que se ha tardado en obtener el último fotograma, para comparar con la media.
- **Fotogramas por segundo medios**: se obtienen directamente de la media de fotogramas.

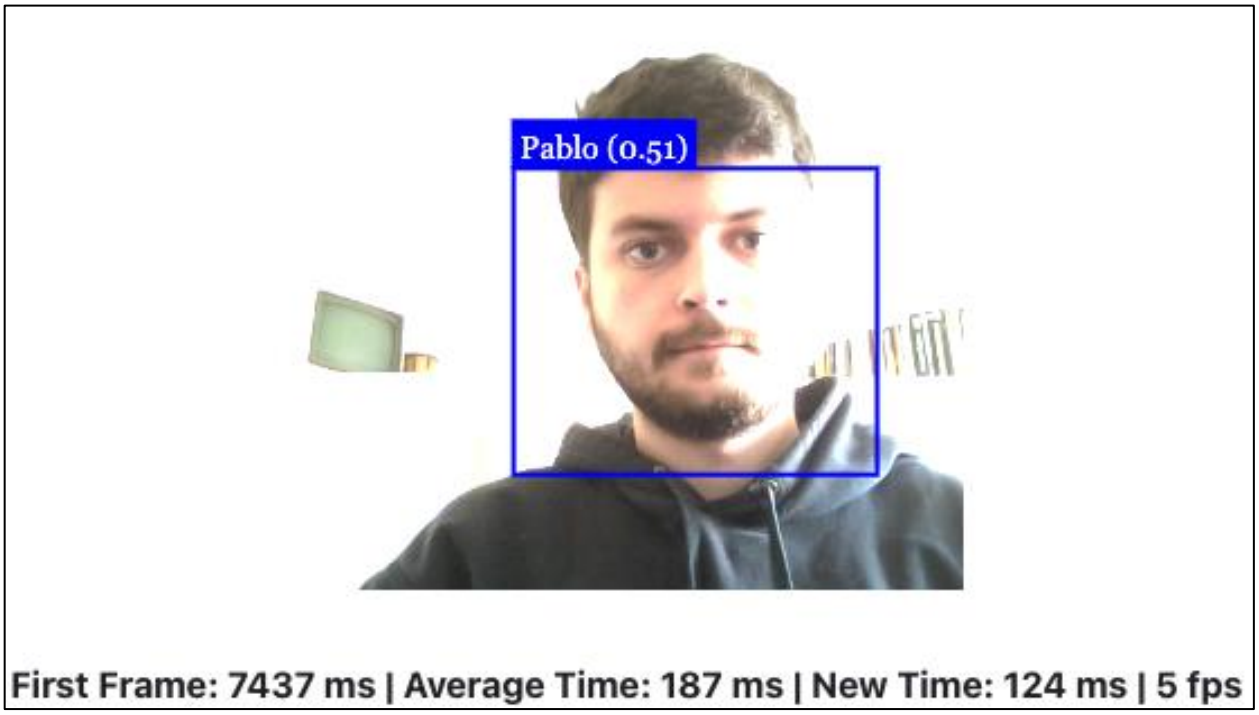

Fig. 9. FOTOGRAMA DE LA APLICACIÓN FACEAPP DURANTE EL RECONOCIMIENTO FACIAL.

Como se puede observar en la imagen, la aplicación FaceApp realiza todas las mediciones y los cálculos con el objetivo de simplificar al máximo la comparativa.

6.3. Procedimiento

Para cada tecnología analizada se ha realizado un total de 10 iteraciones para obtener una imagen más fidedigna de la eficiencia real en cada caso. Para cada iteración se apuntan los siguientes datos:

- El tiempo de carga: se extrae directamente del tiempo de procesamiento del primer fotograma.
- Tiempo medio por fotograma: Para obtener este dato es necesario mantener el reconocimiento facial activo durante unos segundos hasta que el tiempo medio se acerque o iguale el tiempo actual, de manera que podamos cerciorarnos de que se ha alcanzado realmente el tiempo promedio.
- FPS medios: se calculan directamente del tiempo medio por fotograma. Aunque no es estrictamente necesario para la comparación se considera un dato interesante para entender cómo de eficiente está siendo el reconocimiento.

Por último, se obtiene la media de todas las iteraciones y se extrae la desviación típica para poder analizar cómo de consistentes son los tiempos.

Tabla 2. FRAGMENTO DE HOJA DE CÁLCULO CON EL REGISTRO DE TIEMPOS DURANTE LA EXPERIMENTACIÓN.

	A	B	C	D	E
1	Backend	Iteración	Tiempo de carga (ms)	Tiempo medio por fotograma (ms)	FPS medios
2	Webgl	1	7147	102	10
3		2	7394	104	10
4		3	7271	103	10
5		4	7016	104	10
6		5	7943	104	10
7		6	8017	104	10
8		7	8009	105	10
9		8	7862	103	10
10		9	7788	105	10
11		10	7927	105	10
12		Media	7637.4	103.9	10.0
13		Desviación t	367.8	0.9	0.0
14	Wasm	1	776	206	5
15		2	1046	208	5
16		3	1069	204	5
17		4	1253	209	5
18		5	1322	208	5
19		6	802	209	5
20		7	727	209	5
21		8	738	209	5
22		9	1036	210	5
23		10	825	204	5
24		Media	959.4	207.6	5.0
25		Desviación t	205.6	2.1	0.0

6.4. Resultados obtenidos

6.4.1. Análisis del tiempo de carga

La Tabla 3 contiene un resumen de los resultados obtenidos en relación con el tiempo de carga de cada implementación de Tfjs. Los resultados completos por dispositivo se pueden consultar en el Anexo B de este documento.

Tabla 3. MEDIA Y DESVIACIÓN TÍPICA DE TIEMPO DE CARGA MEDIO (MS)DEL PRIMER FOTOGRAMA POR CADA IMPLEMENTACIÓN DE TFJS.

	cpu		webgl		servidor		wasm		wasm SIMD		wasm SIMD + MT	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Tiempo (ms)	1667.00	45.12	4117.00	264.34	530.70	96.63	959.40	56.46	892.20	14.21	325.70	11.09

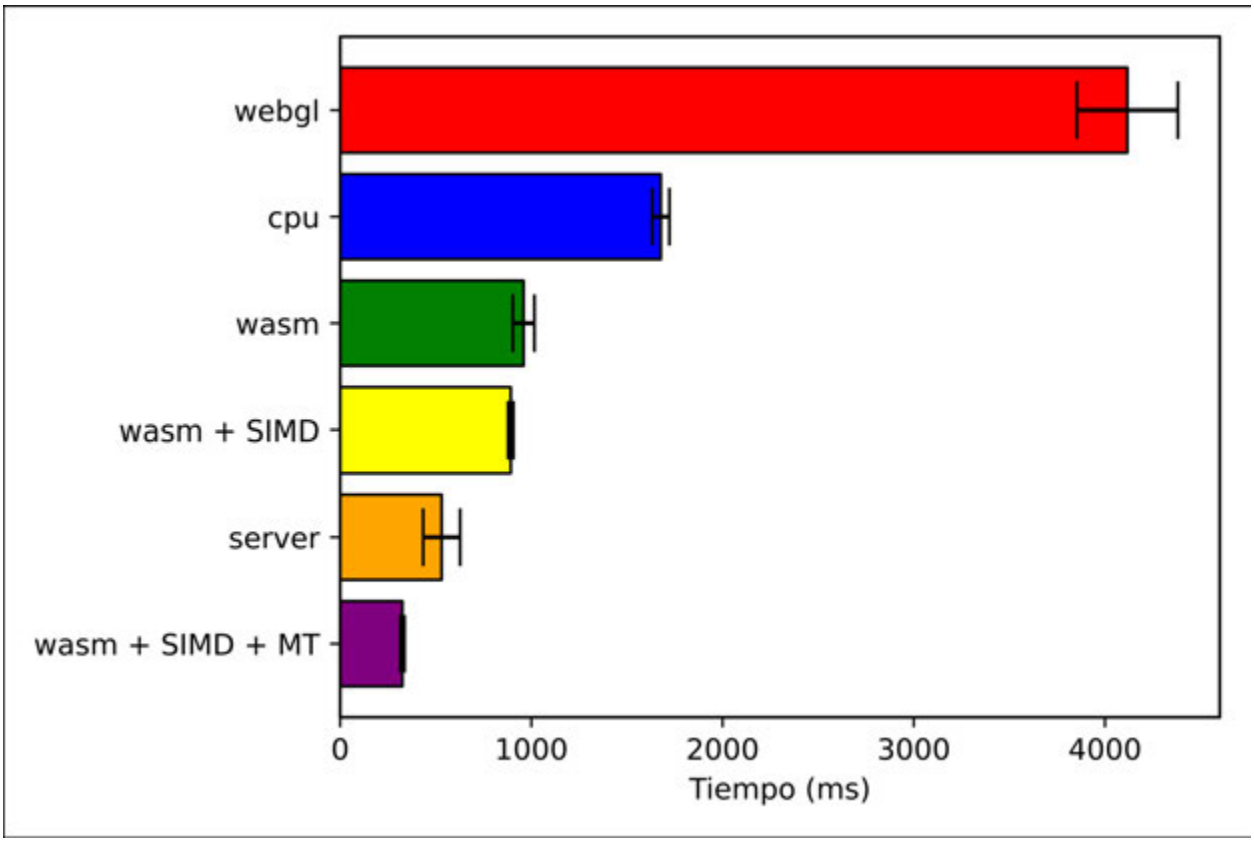

Fig. 10. GRÁFICO DE BARRAS CON LOS TIEMPOS MEDIOS Y DESVIACIÓN TÍPICA PARA LA CARGA DEL PRIMER FOTOGRAMA PARA CADA IMPLEMENTACIÓN DE TFJS.

En el gráfico se puede observar que, de media, el primer fotograma tarda algo más de 4 segundos cuando se utiliza WebGL. WASM, en cambio, tarda tan solo un 23,3% de ese tiempo en su implementación por defecto, reduciéndose aún más en caso de activar las funcionalidades SIMD y MT (7,9% del total que tarda WebGL).

Si usáramos la opción de JavaScript puro (CPU) el tiempo de carga del primer fotograma sería todavía bastante superior a WASM, aunque en menor medida. En este caso, WASM es un 73% más rápido por defecto y un 511% si se activasen SIMD y MT.

Es importante tener en cuenta que hay dispositivos que están más capacitados para el uso de WebGL y que arrojan unos tiempos mucho mejores. En este experimento, el Intel NUC y iPhone X se mantienen por debajo de los 700 ms y éste último tiene muchas más dificultades a la hora de usar WASM.

En cuanto a la implementación en el servidor Node.js, este primer fotograma tan solo tarda 530ms de media, dado que no necesita un tiempo de carga previo (el servidor siempre está preparado para procesar las imágenes).

Cuando estudiamos la variabilidad de estos tiempos, WebGL vuelve a destacar negativamente y de manera muy extrema en relación con el resto de las implementaciones, llegando a ser más de 4,5 veces más fluctuante que WASM (24 veces más si se activa SIMD y MT).

Esto se traduce en que el usuario no tendrá una experiencia de uso consistente con la aplicación y no se podrá predecir fácilmente cuánto puede tardar en cargarse los modelos neuronales.

JavaScript puro (CPU) sí que se muestra más estable, en concreto un 25% más), aunque WASM+SIMD+MT de nuevo reducen en varias órdenes de magnitud la variabilidad en comparación con CPU.

6.4.2. Análisis del tiempo por fotograma

En la Tabla 4 se muestran los resultados con los tiempos medios para el total de fotogramas procesados durante el reconocimiento de una persona a través de una cámara web. No todos los dispositivos soportan las funcionalidades de SIMD y ejecución multi hilo de la tecnología WebAssembly, por lo que en estos casos no ha sido posible realizar la comparativa completa.

Tabla 4. MEDIA Y DESVIACIÓN TÍPICA DE TIEMPO DE CARGA MEDIO (MS) DE TODOS LOS FOTOGRAMAS POR CADA IMPLEMENTACIÓN DE TFJS.

	cpu		webgl		servidor		wasm		wasm SIMD		wasm SIMD + MT	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Tiempo (ms)	1624.40	57.76	94.60	3.06	237.60	36.25	201.50	13.08	110.40	1.47	86.65	1.81

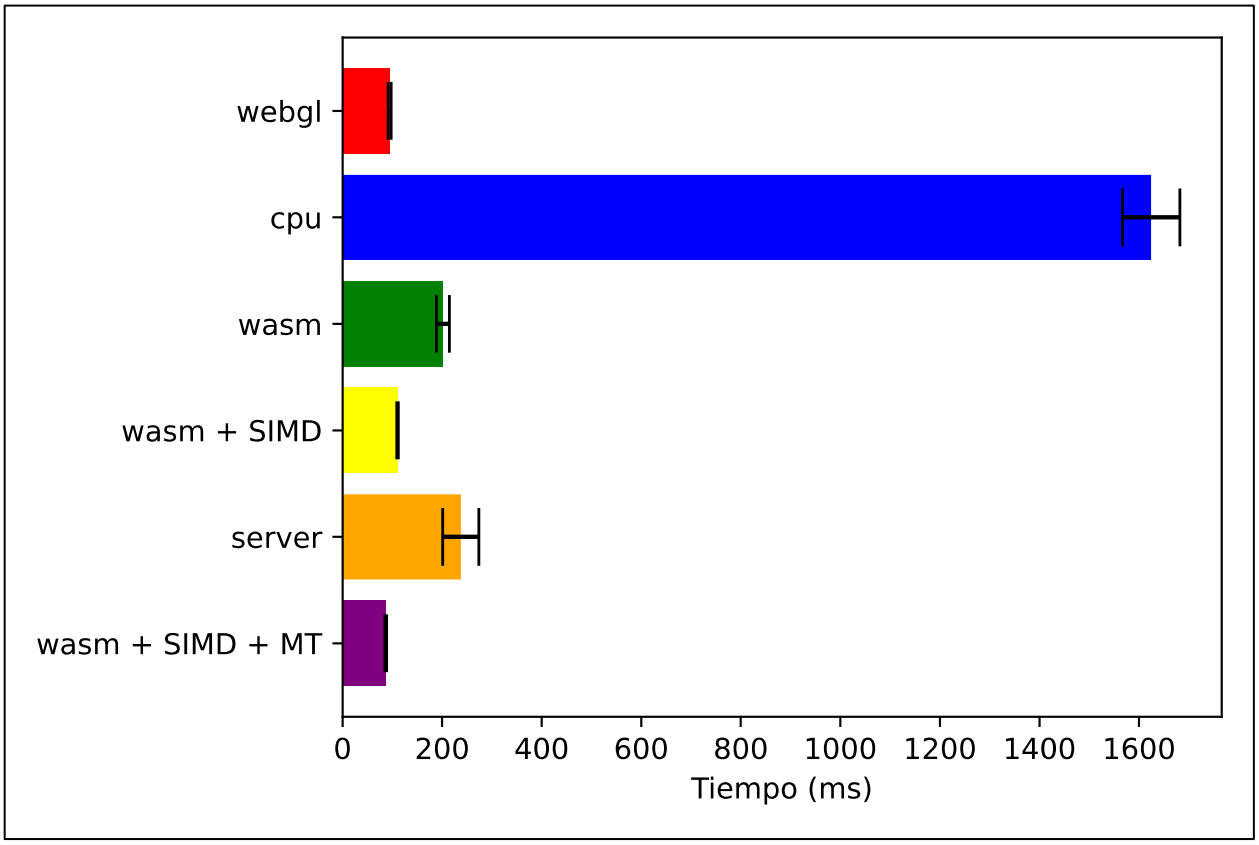

Fig. 11. GRÁFICO DE BARRAS CON LOS TIEMPOS MEDIOS Y DESVIACIÓN TÍPICA DE TODOS LOS FOTOGRAMAS PARA CADA IMPLEMENTACIÓN DE TFJS.

Una vez el primer fotograma se ha procesado correctamente, se ha analizado la eficiencia media del resto de fotogramas. En este caso, CPU se aleja mucho del resultado del resto de implementaciones con tiempos superiores a 1500ms por fotograma mientras que el resto no superan en ningún caso los 250ms.

El resto de las implementaciones de procesamiento en el cliente son más eficientes que el procesamiento en el servidor, el cual se reduce a la mitad que en el primer fotograma

presumiblemente debido a que hay un período de establecimiento de la comunicación entre el cliente y el servidor.

WebGL tarda la mitad de tiempo en procesar, de media, cada uno de los fotogramas en comparación con la implementación básica de WebAssembly. Sin embargo, si se compara con WASM y sus funcionalidades experimentales se observa una mejora del 10% con respecto a WebGL.

La desviación estándar de la media de ms por cada fotograma procesado muestra que la implementación de Tfjs que obtiene unos tiempos más variables es CPU, con casi 60ms de media de diferencia entre fotogramas.

De nuevo se demuestra que las implementaciones en el cliente (a excepción de CPU) son más estables que en el caso del servidor, el cual sufre las interferencias normales de una comunicación a través de internet con WebSockets (sumadas a la propia variabilidad del procesamiento en el servidor).

Aunque la estabilidad de WASM es baja en comparación con WebGL (algo más de 4 veces superior), si activamos SIMD la estabilidad aumenta en gran medida hasta alcanzar cifras cercanas a la unidad (ms) con alrededor del 50% de mejoría con respecto a WebGL.

En este caso, la opción de MT en WASM no mejora la estabilidad del proceso (como sí lo hacía con los tiempos de procesamiento medio), sino que añade algunas décimas de variabilidad en comparación con utilizar únicamente SIMD. Sin embargo, incluso en este caso se mejoran los tiempos de WebGL en un 40%.

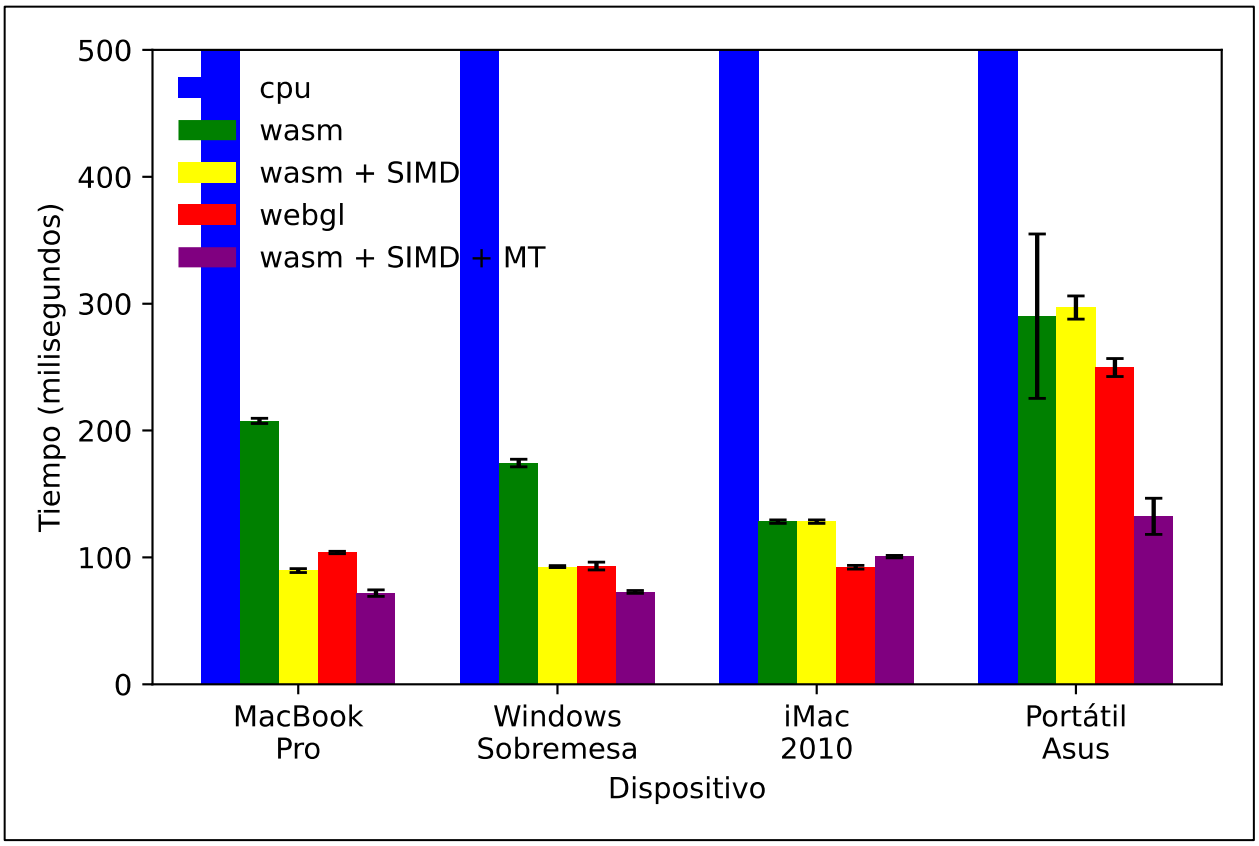

Fig. 12. COMPARATIVA ENTRE TODAS LAS IMPLEMENTACIONES LOCALES DE TFJS.

Si se analiza el segmento de implementaciones en el cliente y los dividimos por dispositivos (únicamente aquellos que soportan SIMD y MT) podemos observar que CPU en todos los casos es mucho mayor que el resto (truncado en el gráfico de la Figura 12 a partir de 500 ms).

WASM se sitúa en la mayoría de los casos por encima del resto (a excepción de JavaScript puro) pero cuenta con los mejores tiempos cuando se activan todas sus funcionalidades experimentales a excepción del iMac.

7. MARCO REGULADOR

En cuanto al tratamiento de los datos biométricos de los usuarios, es necesario acogerse al Reglamento General de Protección de Datos (RGPD) [20]. Dicha normativa hace referencia al tratamiento de información de los usuarios, algo que en este proyecto es de vital importancia dada la naturaleza de las aplicaciones de reconocimiento facial. En concreto, es necesario obtener y procesar imágenes de las personas que usan la aplicación con las que se extraen vectores de puntos faciales.

En este sentido, las imágenes no se almacenan en ningún momento y son descartadas una vez se ha terminado el proceso de obtención de puntos faciales. En cuanto a los vectores, éstos se guardan única y exclusivamente en el almacenamiento local del navegador del usuario y por tanto no sale en ningún momento de su dispositivo

Para el desarrollo de la aplicación web FaceApp, se han utilizado única y exclusivamente librerías de código abierto con licencias de tipo MIT o Apache 2.0. Así mismo, tanto el código fuente de la aplicación FaceApp como de su implementación en servidor, FaceAppServer, se han distribuido de manera abierta y gratuita bajo la licencia MIT a través de sus respectivos repositorios en GitHub.

8. ENTORNO SOCIO-ECONÓMICO

8.1. Presupuesto

En esta sección se detalla el coste económico derivado de todo el proceso de elaboración y experimentación que ha supuesto este trabajo de investigación.

En cuanto al hardware utilizado durante el proyecto, se distinguen entre dispositivos de prueba y aquellos utilizados para el desarrollo de la aplicación FaceApp. En este caso el MacBook Pro ha servido para ambos propósitos por lo que el total de meses de uso asciende a 5, mientras que el resto se ha utilizado únicamente durante la etapa de pruebas.

Tabla 5. DESCRIPCIÓN DE COSTES DE MATERIAL INFORMÁTICO.

	Valor total	Vida útil (años)	Meses de uso	Coste amortizado
MacBook Pro 13	1.200,00€	5	5	100,00€
iPhone X	800,00€	4	1	16,60€
Portátil ASUS	800,00€	4	1	16,60€
Sobremesa Windows	1000,00€	5	1	16,60€
Intel NUC	600,00€	3	1	16,60€
iMac 2010	400,00€	3	1	11,10€
iPad Air 2	400,00€	3	1	11,10€
Total				**188,60€**

En la siguiente tabla se exponen todas las licencias de software que han sido utilizados durante el proyecto, en su mayoría gratuitos o de código abierto.

Tabla 6. DESCRIPCIÓN DE COSTES DE LICENCIAS Y SOFTWARE.

	Coste mensual	Total
GitHub	0€	0€
Heroku	25€	125€
Microsoft Office	10€	50€
VSCode	0€	0€
Librerías de código abierto (Tfjs, React, Node.js, etc)	0€	0€
Total		**175€**

Por último, se detalla el coste de personal, dividido entre los distintos roles y desglosadas por horas de trabajo.

Tabla 7. DESGLOSE DE COSTES DE PERSONAL.

	Coste por horas	Total horas	Total (Bruto)
Jefe de proyecto	25€	100	2.500,00€
Analista	20€	50	1.000,00€
Programador	20€	120	2.400,00€
Ingeniero testing	20€	40	800,00€
Total			**6.700,00€**

En total, el conjunto del proyecto ha tenido un coste de:

$$Material\ inform\acute{a}tico\ +\ licencias\ y\ software\ +\ salarios\ del\ personal\ =$$

$$188{,}60\ +\ 175{,}00\ + 6.700{,}00\ =\ \mathbf{7.063{,}60}€$$

8.2. Impacto Socio-económico

Uno de los colectivos que puede resultar más beneficiado por el uso extensivo de soluciones biométricas, asequibles, precisas y que respeten la privacidad como elemento central de su diseño, son el colectivo de personas mayores [21].

Utilizando este tipo de tecnologías es posible acercar los canales digitales (banca, compañías de teléfono, administración pública) sin necesidad de métodos de segundo factor como SMS o tarjetas de coordenadas. De esta forma se puede reducir la exclusión a la que se han visto sometidos las personas con dificultades a la hora de adoptar nuevas tecnologías, las cuales sin duda van a seguir expandiéndose en el día a día de las personas.

A nivel medioambiental, el procesamiento local de la información a través de los navegadores web puede reducir la necesidad de utilizar servidores o centros de datos como los de Amazon Web Services, Google Cloud o Microsoft Azure.

9. CONCLUSIONES

El proyecto propuesto en el primer apartado de este documento ha sido completado de manera exitosa por los siguientes motivos:

- La aplicación web para la realización de pruebas, FaceApp, ha sido desarrollada íntegramente, cumpliendo con la totalidad de requisitos. El código fuente se encuentra alojado en un repositorio público con licencia de código abierto.
- Durante el desarrollo de dicha aplicación se han realizado diversas aportaciones a proyectos de código abierto ya existentes que han permitido avanzar en el desarrollo de la aplicación de pruebas a la vez que se permite al resto de usuarios disfrutar de estos avances.
- Se ha llevado a cabo una batería de pruebas con distintos dispositivos para comparar las diferentes maneras de ejecutar modelos neuronales para el reconocimiento facial, tanto en el lado del cliente como en el servidor.
- Los experimentos y resultados presentados en este trabajo han dado lugar a un artículo científico, enviado a la VII Conferencia Iberoamericana en Interacción Persona-Computadora (HCI 2021) [1] - y, a fecha de presentación de este TFG, se encuentran en estado de revisión. Así mismo, dicho trabajo va a ser expuesto por el autor de este TFG en dicha conferencia (septiembre de 2021).

Los resultados de las pruebas realizadas me llevan a las conclusiones que expongo a continuación:

Tanto WebGL como WebAssembly son dos tecnologías web muy eficientes para su uso en el área del reconocimiento facial en el navegador. WebGL cuenta con la ventaja de estar más ampliamente aceptado por los principales navegadores web, mientras que WebAssembly proporciona unos tiempos de carga mucho menores (especialmente en conjunción con las funcionalidades SIMD y Multi Thread) y tiempos de procesamiento similares a los de WebGL. WebAssembly, además, no necesita una unidad GPU, algo de especial interés para dispositivos móviles o embebidos.

Sin embargo, algunas de las funcionalidades de WebAssembly aún se encuentra en fase de desarrollo y por tanto no están listas para ser utilizadas en aplicaciones orientadas al público en general.

En definitiva, existen suficientes recursos para poder realizar reconocimiento facial sin necesidad de un servidor externo y poder así aprovechar las ventajas que este tipo de arquitecturas ofrece: mayor privacidad para los usuarios y escalabilidad para los desarrolladores de aplicaciones.

BIBLIOGRAFÍA

[1] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai y J. F. Bastien, "Bringing the web up to speed with WebAssembly", New York, NY, USA, 2017.

[2] A. E. Youssef, "Exploring cloud computing services and applications", *Psu.edu.*

[3] M. Zhou, R. Zhang, W. Xie, W. Qian y A. Zhou, "Security and privacy in cloud computing: A survey", 2010.

[4] S. Taheri, A. Vedienbaum, A. Nicolau, N. Hu y M. R. Haghighat, "OpenCV.js: Computer vision processing for the open web platform", New York, NY, USA, 2018.

[5] R. Martin Manso y P. Escriva Gallardo, "Face Recognition Efficiency Improvements Using TensorFlow's WebAssembly Backend: a practical approach", *VII Iberoamerican Conference on Human Computer Interaction,* 2021.

[6] A. Butterfield, G. E. Ngondi y A. Kerr, Eds., A Dictionary of Computer Science, London, England: Oxford University Press, 2016.

[7] H. Z. U. Khan, "Comparative study of authentication techniques".

[8] P. M. Vallone, *DNA as a Potential Biometric Tool.*

[9] A. Jain, L. Hong y S. Pankanti, "Biometric identification", *Commun. ACM,* vol. 43, no. 2, pp. 90--98, 2000.

[10] "TensorFlow", [En línea]. Disponible en: http://Tensorflow.org.

[11] "TensorFlow.js", [En línea]. Disponible en: https://www.tensorflow.org/js.

[12] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen, D. Soergel, S. Bileschi, M. Terry, C. Nicholson, S. N. Gupta, S. Sirajuddin, D. Sculley, R. Monga, G. Corrado, F. B. Viégas y M. Wattenberg, "TensorFlow.Js: Machine learning for the web and beyond", 2019.

[13] "WebAssembly Proposals", [En línea]. Disponible en: https://github.com/WebAssembly/proposals. [Consultado 03 05 2021].

[14] Czech Technical University in Prague, "Basics of SIMD Programming".

[15] V. Mühler, "face-api.js", [En línea]. Disponible en: https://justadudewhohacks.github.io/face-api.js/docs/index.html.

[16] *Facial point annotations,* Ic.ac.uk.

[17] "WebRTC", [En línea]. Disponible en: https://webrtc.org/.

[18] P. Escrivá, *Pull Request: Fixed typo in backend_wasm.ts,* 2020.

[19] P. Escrivá y V. Mandic, *Pull Request: Unknown dtype undefined.*

[20] "REGLAMENTO (UE) 2016/679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 27 de abril de 2016", [En línea]. Disponible en: https://www.boe.es/doue/2016/119/L00001-00088.pdf.

[21] V. G. Sanchez, C. F. Pfeiffer and N.-O. Skeie, "A review of smart house analysis methods for assisting older people living alone," *J. Sens. Actuator Netw.,* 2017.

ANEXOS

ANEXO A: Instrucciones para activar WASM con SIMD y Multi-hilo en Google Chrome

Dado que estas dos funcionalidades aún no están disponibles por defecto en los navegadores web, es necesario activarlas manualmente para poder realizar la comparativa de manera correcta. Para ello hay que seguir los siguientes pasos:

1. En Google Chrome, visitar la URL: chrome://flags/
2. Para SIMD, buscar "WebAssembly SIMD support". Para Multi-hilo, "WebAssembly threads support".
3. Marcar la opción como "Activada".
4. Recargar la página FaceApp.

ANEXO B: Tabla completa de los resultados

Tabla A1. TIEMPO POR FOTOGRAMA (MEDIA DE 10 ITERACIONES) (ms)

	cpu		webgl		servidor		wasm		wasm SIMD		wasm SIMD + MT	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
MacBook Pro	1624.40	50.65	103.90	0.94	237.60	36.25	207.60	2.06	89.60	1.56	71.80	2.52
Sobremesa Windows	1535.20	44.67	93.20	3.06	216.80	11.57	174.40	3.01	92.70	0.78	72.70	1.10
iMac 2010	2016.20	57.76	92.20	1.47	523.70	64.25	128.10	1.37	128.10	1.37	100.60	0.92
Portátil Asus	2366.50	169.70	249.60	7.07	N/A	N/A	290.10	64.85	296.90	9.15	132.30	14.28
iPhone X	662.00	71.51	94.60	1.43	107.70	2.24	201.50	13.08	N/A	N/A	N/A	N/A
iPad Air 2	1410.80	140.97	216.90	10.62	403.30	145.21	605.60	89.17	N/A	N/A	N/A	N/A
Intel NUC	1667.60	32.44	78.50	16.25	N/A	N/A	176.60	16.41	N/A	N/A	N/A	N/A
MEDIA	1624.40	57.76	94.60	3.06	237.60	36.25	201.50	13.08	110.40	1.47	86.65	1.81

Tabla A2. TIEMPO DE CARGA DEL PRIMER FOTOGRAMA (MEDIA DE 10 ITERACIONES) (ms)

	cpu		webgl		servidor		wasm		wasm SIMD		wasm SIMD + MT	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
MacBook Pro	1574.40	23.99	7637.40	367.79	475.80	70.23	959.40	205.56	1168.90	52.86	348.20	44.53
Sobremesa Windows	1295.40	316.10	4117.00	264.34	400.60	92.51	562.10	28.73	615.50	14.42	235.80	19.26
iMac 2010	1962.50	12.44	5093.40	601.27	901.60	203.30	547.60	14.00	547.60	14.00	303.20	2.93
Portátil Asus	2451.70	45.12	4129.80	60.66			1126.20	56.46	1800	0.00	600.00	0.00
iPhone X	923.10	72.94	678.30	9.56	652.50	96.63	2965.30	825.22				
iPad Air 2	2038.70	191.93	2422.60	282.33	530.70	164.31	7378.20	1056.98				
Intel NUC	1667.00	34.30	569.30	33.13			556.70	34.53				
MEDIA	1667.00	45.12	4117.00	264.34	530.70	96.63	959.40	56.46	892.20	14.21	325.70	11.09