Universidad de Valladolid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Grado en Ingeniería de Tecnologías Específicas de Telecomunicación. Mención en Telemática

Desarrollo de un sistema de Machine Learning para Eye-Tracking sin hardware específico

Eduardo Martínez Juárez

Tutores: Rodrigo de Luis García y Federico Simmross Wattenberg

Junio de 2025

Resumen

El seguimiento ocular o *Eye-Tracking* ha sido objeto de estudio desde el siglo XIX, dado su interés en el campo de la ciencia y psicología cognitiva. En la actualidad, gracias al indiscutible avance de la tecnología , cobra una nueva dimensión de áreas en las que su aplicación resulta relevante, como lo son la interacción persona-computadora, la accesibilidad o el marketing.

La forma más habitual de adquirir datos para *Eye-Tracking* en un ordenador requiere dispositivos dedicados basados en infrarrojos, con los inconvenientes que supone utilizar hardware específico.

El presente trabajo desarrolla un sistema de *Eye-Tracking* basado en *Deep Learning* que permite estimar la dirección de la mirada sin hardware especializado, utilizando solo la cámara del dispositivo del usuario. Para ello se adquieren datos faciales y posicionales con los que posteriormente se evalúan diferentes propuestas de arquitecturas de redes convolucionales y estrategias de preprocesamiento de datos para optimizar la precisión del sistema.

Los resultados obtenidos demuestran la viabilidad de la propuesta en un mercado en el que hay soluciones comerciales existentes, aunque escasas, abriendo la puerta a futuras mejoras e implementaciones en entornos reales.

Palabras clave: Seguimiento ocular, aprendizaje profundo, adquisición, detección facial, detección ocular, validación cruzada.

Abstract

Eye-Tracking has been subject of study since 19th century, given its interest in the field of cognitive science and psychology. Nowadays, thanks to the indisputable advance of technology, it is taking on a new dimension in areas where its application is relevant, such as human-computer interaction, accessibility or marketing.

The most common way of acquiring data for Eye-Tracking in a computer requires dedicated infrared-based devices, with the disadvantages of using specific hardware.

This paper develops an eye-tracking system based on Deep Learning that allows gaze direction estimation without specialised hardware, using only the camera of the user's device. For this purpose, facial and positional data are acquired, with which different proposals for convolutional network architectures and data pre-processing strategies are subsequently evaluated to optimise the accuracy of the system.

Results demonstrate the feasibility of the proposal in a market where there are existing commercial solutions, although scarce, opening the door to future improvements and implementations in real environments.

Keywords: Eye-tracking, deep learning, acquisition, face detection, eye detection, cross validation.

Agradecimientos

A la decisión de estudiar Teleco. Algo en lo que de vez en cuando pienso –y me aterra– es la enorme cantidad de variables de mi vida cuyos valores fueron definidos al escoger realizar estos estudios, y lo que me hubiera perdido si no lo hubiera hecho...

A la ETSIT, por haber hecho que mi etapa universitaria superase toda expectativa y por el sentimiento de pertenencia, reafirmado en cada curso.

A los tutores de este Trabajo, Fede y Rodrigo, por definir y esclarecer el camino a seguir.

A mi grupo de amigos de Cabezón, por estar siempre ahí, y por ser el apoyo que necesitaba en los primeros duros años del Grado.

A Alberto, César, Ángel, Javi, Arturo, Berti y compañía. Gente valiosa que ha amenizado este camino, del que también han sido partícipes y que con mucho gusto me llevo de Teleco.

A David, Juanchi, Ana, Marco, Dani y demás compañeros del Radioclub, por ser excelentes amigos, por lo que hemos crecido juntos, por el apoyo incondicional en el proceso, por la ingente cantidad de buenas anécdotas que nos quedan para contar, y por la suerte que he tenido al formar parte de esta iniciativa con vosotros.

A Iván –compañero de Mención y de estudios, un gran amigo casi desde el primer día y parte fundamental de esta historia– por estar siempre ahí, en cada una de las batallas, en las buenas y en las malas, en las duras y en las maduras.

A Jana, mi pareja, que vino invitada por Ana (¡Gracias Ana!) a una de las fiestas patronales de nuestra Escuela, en la que nos conocimos. Más de dos años han pasado desde entonces. Gracias por ser mi sustento día a día, por ser una fuente de motivación y un ejemplo a seguir; por haber estado y haberme ayudado a salir adelante en momentos muy duros, y por haberme puesto todas las facilidades que han sido posibles para avanzar en este Trabajo cuando parecía inacabable.

A mi familia, por haber hecho posible que haya llegado hasta aquí, entre muchas otras cosas. Por su apoyo incondicional permanente y por ser muchísimo más de lo que uno puede pedir. Ellos son los autores espirituales de este Trabajo, y quienes realmente merecen el Título al que espero optar cuando lo defienda.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Contexto y motivación

El seguimiento ocular, o *Eye-Tracking*, en el marco de la ciencia de la visión, es el estudio del movimiento del ojo en relación a la cabeza. Con el equipamiento de medida correspondiente, también se puede definir como la evaluación del punto al que se dirige la mirada: dónde mira una persona, durante cuánto tiempo fija la mirada en un punto determinado y cómo se desplaza su atención visual por una escena. Esta información es clave para entender cómo las personas interactúan visualmente con su entorno y qué elementos captan su atención de forma consciente o inconsciente, lo que caracteriza tanto a los ojos –la entrada principal del sistema visual– como al cerebro, ya que los movimientos oculares están estrechamente relacionados con procesos cognitivos como la atención, la memoria o la toma de decisiones, analizar cómo se mueven los ojos nos ofrece una ventana directa al procesamiento cerebral [1].

Los primeros estudios sobre el movimiento ocular datan de finales del siglo XIX, motivados por la búsqueda de respuestas sobre neurociencia y psicología cognitiva [2]. En la actualidad, el contexto no es el mismo, pues entonces no existían los ordenadores personales. Ahora que una parte importante de las actividades del día a día resulta impensable sin ellos, el *Eye-Tracking* deja de estar relegado exclusivamente a la academia y la investigación para ser una herramienta poderosa en lo digital, entre cuyas aplicaciones más comunes se encuentran el diseño de interfaces de usuario, la publicidad, la accesibilidad, y la investigación en marketing [3].

En el ámbito científico, se ha convertido en una herramienta esencial para estudiar el funcionamiento del cerebro humano, ya que es una parte considerable del cerebro la que se encarga de extraer características de las imágenes. Esto sucede de forma jerárquica, empezando por aspectos muy básicos en el lóbulo occipital, como bordes u orientación; para posteriormente evaluar formas, colores, y finalizar recorriendo áreas visuales secundarias agrupadas según el camino que toma la información: la vía dorsal, que se extiende

hacia el lóbulo parietal y se encarga de la percepción espacial y el movimiento; y la vía ventral, que se dirige al lóbulo temporal y se especializa en el reconocimiento de objetos y rostros [4].

Tradicionalmente, el *Eye-Tracking* ha requerido el uso de hardware especializado, como cámaras de alta precisión o gafas equipadas con sensores infrarrojos. Estos dispositivos, aunque eficaces, suelen ser costosos y de difícil acceso para muchas instituciones o investigadores independientes. Además, requieren de condiciones controladas y de una calibración previa, lo que limita su uso a entornos muy específicos y reduce su aplicabilidad en estudios a gran escala o en contextos más informales [5].

Esta barrera tecnológica podría eliminarse si se desarrollaran soluciones capaces de realizar seguimiento ocular utilizando dispositivos mucho más accesibles, como un ordenador con una webcam estándar. Dado que hoy en día prácticamente cualquier portátil o teléfono móvil dispone de cámara frontal, una herramienta de *Eye-Tracking* basada en este tipo de hardware permitiría democratizar el acceso a esta tecnología y abrir nuevas posibilidades para su estudio y aplicación.

En la industria de la visión artificial se ha formado cierto consenso en torno a que las cámaras, gracias a los avances en Inteligencia Artificial y *Deep Learning* pueden ser igual de efectivas que los sensores dedicados a la hora de obtener información sobre el entorno y analizar patrones, aportando un valor añadido en cuanto a versatilidad, flexibilidad, simplicidad y reducción de costos, ya que son un dispositivo mucho más común, económico y de propósito general [6]. Este efecto se observa con frecuencia en el contexto del aprendizaje automático, ya que el propósito de la inteligencia artificial es –a grandes rasgos– resolver de forma fácil y eficiente ciertos problemas que con los paradigmas computacionales tradicionales requerían de soluciones mucho más complejas.

Precisamente, esta es la idea que motiva el presente trabajo: explorar la viabilidad de realizar *Eye-Tracking* utilizando únicamente una cámara web convencional, investigando qué nivel de precisión y utilidad se puede alcanzar con métodos basados en visión por computador. De este modo, se busca sentar las bases para un sistema de seguimiento ocular más accesible, versátil y escalable.

1.2 Hipótesis y objetivos

El método más común mediante el cual los ordenadores pueden seguir los ojos de sus usuarios es utilizando dispositivos dedicados. Hay dos categorías claramente diferenciables de estos, en función del área de trabajo:

- Aquellos que se montan sobre la cabeza del usuario: Estos registran la dirección de la mirada del usuario en su entorno. Se suelen utilizar en gafas o en el interior de cascos de realidad virtual.
- Aquellos que se montan sobre la pantalla del equipo: Son los más comunes. Su

propósito es más general y también son más económicos, aunque su zona de trabajo queda relegada a la pantalla del dispositivo sobre el que se montan y por tanto, solo tienen cabida durante la utilización del mismo [5].

El funcionamiento de todos ellos es simple: Cuentan con iluminadores (por lo que funcionan incluso en la oscuridad) que emiten luz infrarroja hacia los ojos, creando patrones cuyos reflejos se capturan mediante cámaras con sensor CCD o CMOS sin el habitual filtro de infrarrojos que suelen montar las cámaras digitales de propósito general. Las imágenes son procesadas bien en el dispositivo de *Eye-Tracking* o en la CPU del ordenador. El análisis de estos datos mediante software permite identificar los diferentes movimientos oculares y los puntos del área de trabajo hacia los que se dirige la mirada [1].

El error de estos dispositivos es, en el caso de la gama profesional de Tobii, el principal fabricante de *Eye-Trackers* a nivel mundial al momento de la redacción de este Trabajo, del orden de 0,5°, lo que los hace muy precisos y aptos para tareas con cierto nivel de criticidad, incluyendo los modelos más básicos. Su principal inconveniente es el coste de adquisición, que parte de los 279 euros para el Tobii Eye-Tracker 5, de gama de entrada; dedicado al ámbito doméstico, para videojuegos y streaming principalmente (Para más información sobre la oferta comercial de Tobii al momento de redacción de este TFG, léase el punto 2.1, *Eye-Tracking*, del Capítulo 2, Estado del arte). Si bien, para según qué aplicaciones se puede considerar asequible, para otras puede considerarse prohibitivo, dejando sin cubrir un nicho de mercado del que forman parte aplicaciones en las que ni la precisión ni la tasa de muestreo son tan importantes como la economía, en cuyo caso un dispositivo específico es una solución desproporcionada [5].

En la actualidad, gracias a los avances en Inteligencia Artificial y *Deep Learning*, las soluciones basadas en estas tecnologías pueden ser igual de efectivas que los sensores dedicados a la hora de obtener información sobre el entorno y analizar patrones, aportando un valor añadido en cuanto a versatilidad, flexibilidad, simplicidad y reducción de costos, ya que son un dispositivo mucho más común, económico y de propósito general [6]. Una referencia clara de esto es el fabricante de vehículos eléctricos Tesla y su sistema de conducción autónoma total supervisada, *FSD*. Las versiones preliminares contaban con un sensor LiDAR en el frontal del vehículo complementado por cámaras que lo rodeaban, pero en 2021 comenzaron la transición hacia lo que denominan Tesla Vision, un sistema de adquisición basado exclusivamente en cámaras, de forma que en 2023 ya no entregaban el sensor LiDAR en ninguno de sus modelos. Según la marca, todos sus vehículos equipados con Tesla Vision han conseguido al menos mantener la misma valoración de seguridad activa en Europa y Estados Unidos que sus predecesores con LiDAR, con la peculiaridad de que mediante las cámaras y el reconocimiento de patrones con aprendizaje automático, ahora son capaces de identificar y diferenciar objetos mejor que antes, lo que redunda en un mejor comportamiento del sistema de frenada de emergencia frente a peatones. Todo ello logrando a la vez una evidente reducción de costos [7].

La detección de patrones en redes neuronales permite extraer una información con-

cisa de un conjunto mucho más amplio y genérico con muchos menos recursos. Ello posibilita utilizar imágenes donde antes se requería de complejos sensores y técnicas de detección. Además, dado que estos modelos son capaces de aprender, es posible adaptarlos a diferentes entornos de trabajo, mejorarlos con el tiempo o modificarlos en caso de que el problema cambie ligeramente.

Se defiende entonces, que gracias al aprendizaje profundo, es posible hacer *Eye-Tracking* sin necesidad de hardware específico, bastándose únicamente de un ordenador, una cámara web –que puede ser la integrada en el caso de portátiles– y un programa desarrollado para este fin, logrando una precisión aceptable para un cierto número de aplicaciones que no sean críticas.

Se propone, por tanto, para este Trabajo de Fin de Grado, el desarrollo de una prueba de concepto de un sistema de *Eye-Tracking* tomando ventajas de los avances en Inteligencia Artificial y *Deep Learning*, que se apoye únicamente en el hardware local del del equipo en el que se ejecute (es decir, su cámara web, CPU y opcionalmente GPU) y en técnicas de visión artificial para inferir las coordenadas de la pantalla en píxeles hacia las que se dirige la mirada del usuario; con el siguiente objetivo general:

- Desarrollar un sistema preliminar basado en *Deep Learning* con plena funcionalidad junto con un mecanismo de evaluación que permita determinar la viabilidad de la puesta en producción del mismo, entendiendo como viabilidad la capacidad de suplir la carencia en el mercado de una solución de *Eye-Tracking* técnicamente inferior (en términos de precisión y de mediciones por segundo) a aquellas existentes basadas en hardware; orientada a aplicaciones de baja criticidad y con un coste mucho más bajo.

Y los siguientes objetivos específicos:

- Crear un conjunto de datos que permita definir, entrenar y evaluar el sistema propuesto, para lo que se ha desarrollado una aplicación web que implementa un sistema de adquisición de datos.
- Desarrollar el sistema propiamente, a partir de los datos obtenidos.
- Desarrollar pruebas y métricas que permitan analizar el sistema desarrollado y obtener resultados teóricos que permitan compararlo con soluciones ya existentes.
- Implementar e integrar soluciones parciales ya existentes y accesibles al momento de la realización del proyecto.
- Aplicar los conocimientos específicos y transversales adquiridos durante el Grado así como las tecnologías aprendidas, y complementar la formación de forma autodidacta cuando sea necesario.

1.3 Recursos

Para la realización del proyecto, se ha contado con una serie de elementos *hardware* y *software* proporcionados por el grupo de investigación al que pertenecen los tutores de este Trabajo, el Laboratorio de Procesado de Imagen, LPI, de la Universidad de Valladolid. Todos ellos desempeñando funciones que han sido imprescindibles para la correcta consecución de los objetivos:

- Aplicación web para la adquisición de datos de entrenamiento para *Eye-Tracking* desarrollada por Raúl Barba Alonso [8]: Es una solución parcial conceptual, que forma parte del Trabajo de Fin de Grado de Raúl Barba Alonso, antiguo alumno del Grado en Ingeniería de Tecnologías de Telecomunicación, con los mismos tutores que el actual Trabajo. Consiste en una aplicación que se utiliza con la finalidad de adquirir los datos con los que se entrenan las redes neuronales desarrolladas en el proyecto, lo que coincide con una de las líneas futuras planteadas en la memoria de su proyecto. Para implementarla en el actual se ha realizado un proceso de adaptación con el que se ha implementado una forma de volcar los datos almacenados, se ha adecuado su funcionamiento a unos términos y condiciones basados en el Reglamento General de Protección de Datos, se han optimizado los servicios del *backend* de los juegos existentes y se ha desarrollado un juego adicional, de un tipo diferente a los que existentes, con la intención de realizar pruebas con diferentes casos de uso. Para más detalles sobre el formato de los datos que maneja, léase el Capítulo 3. Para ampliar la información sobre el trabajo de desarrollo y adaptación de la aplicación web para el presente proyecto, léase el Apéndice C.
- Servidor web `saqqara`: Es una máquina virtual alojada en una subred privada del Grupo, accesible a través de internet mediante reenvío de puertos en el firewall de dicha subred, un nombre canónico en el sistema de nombres de dominio (DNS) de la Escuela, necesario para la realización del proyecto. Ejecuta el servidor web Apache, PHP y el servidor de base de datos relacional MySQL sobre Ubuntu. Contiene la aplicación web de aquisición y almacena en su sistema de ficheros los datos que esta genera.
- Servidor `isis`: Es una máquina del Grupo de relativamente nueva adquisición. Cuenta con cuatro GPUs nVidia RTX A5000, lo que resulta ideal para acelerar la mayoría de procesos asociados a trabajar con redes neuronales. Entre otras herramientas, tiene instalado el servicio servidor de Code para trabajar en remoto desde el cliente VS Code de cualquier equipo mediante SSH de forma cómoda y con compatibilidad con cuadernos de Jupyter; Python, Pip y Conda para trabajar con entornos virtuales. Con ello se ha conformado el entorno en el que se ha desarrollado el proyecto.

Los recursos *software* no proporcionados por el Grupo, como librerías, *frameworks*, kits de desarrollo de lenguajes, plataformas y herramietas de trabajo que han conformado

el entorno del proyecto, replicable en otras máquinas de ser necesario; han sido los siguientes:

- Python 3: Es un lenguaje interpretado, sencillo de utilizar, no tipado, con buena legibilidad gracias a la conformación de bloques de código basado en la indentación. Muy utilizado en la actualidad en la investigación académica al contar con gran variedad de librerías adecuadas para tal fin. Por todo ello, es el lenguaje de programación más común en Inteligencia Artificial y Aprendizaje Automático [9].

- Cuadernos de Jupyter: JupyterLab es una herramienta con la que se puede añadir marcado web o LaTeX a scripts de Python (entre otros lenguajes), que añade interactividad, es decir, permite renderizar y mostrar por pantalla el retorno de métodos que no sea exclusivamente texto, que es una limitación inherente a la interfaz de línea de comandos. El tipo de fichero que utiliza tiene extensión ".ipynb", cuyas siglas vienen de "InteractivePython Notebooks", que es el origen de los cuadernos de Jupyter [10]. Si bien se instaló el servicio web de JupyterLab para utilizar los cuadernos, finalmente se optó por trabajar con ellos mediante VS Code, que ofrece compatibilidad prácticamente nativa con la extensión "Jupyter", desarrollada por Microsoft.

- TensorFlow: El *framework* de *Deep Learning* escogido para el proyecto por su simplicidad, popularidad y escalabilidad. Desarrollado por Google. Para profundizar en la implicación de TensorFlow con el proyecto, véase el véase el Capítulo 5 [11].

- Keras: Es una API desarrollada en Python por François Chollet como parte de la investigación del proyecto ONEIROS, un sistema operativo de código abierto para robots inteligentes. Tiene soporte oficial de Google en las librerías de TensorFlow desde 2017. Su cometido es proporcionar un nivel de abstracción superior al que ofrece por defecto el *framework*, reduciendo la complejidad técnica del proceso de definir modelos y entrenarlos; y optimizando el flujo de trabajo. Para conocer más detalles sobre el uso de Keras en el proyecto, véase el Capítulo 5 [12].

- CUDA: Plataforma de computación paralela creada por Nvidia, que proprciona un conjunto de herramientas y librerías para desarrollar con sus GPUs, acelerando los cálculos complejos [13].

- `cuDNN`: Librería acelerada por GPU que contiene primitivas para redes neuronales. Forma parte de la plataforma CUDA, aunque no siempre se entregan juntas [13].

- OpenCV: Es una librería de código abierto pensada para facilitar el desarrollo de aplicaciones basadas en visión artificial y aprendizaje automático. Inicialmente fue desarrollada por Intel, actualmente mantenida por una comunidad global; proporciona métodos para procesar imágenes, detectar caras y en estas a su vez otras facciones; entre muchas otras cosas [14]. Para ampliar la información sobre el uso de OpenCV en este proyecto, léase la sección 5.2 del Capítulo 5.

- Numpy: La librería matemática para Python por excelencia. Proporciona métodos que implementan operaciones matemáticas rápidas, para optimizar y simplificar el código que requiera de ellas. Además permite el manejo de arreglos multidimensionales en Python, pudiendo gestionar cantidades grandes de datos de forma eficiente, lo cual es clave a la hora de hacer análisis de datos en el desarrollo de redes neuronales [9].
- Pandas: Librería que permite analizar y manipular datos tabulares, como aquellos provenientes de hojas de cálculo y bases de datos; en Python. Su estructura de datos fundamental se denomina `dataframe` y se presenta al usuario en formato de tabla [15].
- `matplotlib`: Librería para visualización de datos, que permite crear gráficos en dos dimensiones [9].
- SSHTunnelForwarder: Librería que proporciona métodos para establecer conexiones seguras tunelizadas en Python mediante el protocolo de *shell* seguro, SSH [9].
- Mysql.connector: Librería que proporciona integración en Python con servidores de bases de datos MySQL y MariaDB [9].

1.4 Estructura del documento

Una vez introducido y motivado el proyecto, se realizará un análisis del estado de la técnica al momento de la redacción de la memoria. Después se hará una descripción esencial de la aplicación web con la que se han adquirido los datos de entrenamiento necesarios para la realización del proyecto, y el proceso de recuperación y presentación de los mismos como conjunto en el entorno de trabajo.

Con el contexto de los datos que se van a manejar se explicará la propuesta de arquitectura que se ha desarrollado a su alrededor a lo largo del proyecto, describiendo las herramientas fundamentales en las que se apoya, los procesos utilizados, los modelos de redes neuronales entrenados y los métodos de evaluación desarrollados para medir el rendimiento obtenido.

Si bien el capitulo anterior explicará el mecanismo de evaluación, lo hará a nivel definición. Los resultados de la ejecución del mismo y su análisis se realizarán en un capítulo aparte.

Por último se concluirá la memoria valorando la consecución de los objetivos planteados en su introducción y planteando líneas futuras para el proyecto.

Dado que el Trabajo ha consistido fundamentalmente en desarrollar dos sistemas software, se ha buscado hacer presentes los fragmentos de código más relevantes sin afectar a la legibilidad del documento. Por ello, y aprovechando la utilidad didáctica de los cuadernos de Jupyter, se ha reducido al mínimo la presencia de código en el cuerpo

de la memoria para incluir en dos anexos los cuadernos íntegros. Más específicamente, en el Apéndice A estarán los cuadernos que incluyan los análisis que justifiquen ciertas decisiones tomadas a lo largo de la realización del proyecto; y el Apéndice B que incluya precisamente eso, un cuaderno correspondiente al procesado de datos, definición de modelos y entrenamiento y evaluación de los mismos.

Por otro lado, la adaptación y puesta en marcha de la aplicación desarrollada por Raúl Barba Alonso [8] para la adquisición de datos de entrenamiento no forma parte de los objetivos de este proyecto pero sí fue necesaria para su consecución, por lo que también se ha documentado el proceso en el Apéndice C.

Capítulo 2

Estado del arte

La razón de ser de los estudios sobre el movimiento ocular radica en la fisionomía del ojo humano: La visión no es uniforme a lo largo de la zona fotorreceptora del globo ocular. La fóvea, que se ubica en el centro de la retina, es la única parte del ojo en la que hay conos, las células fotorreceptoras policromáticas, por lo que se encarga de la visión de alta resolución; pero es muy pequeña. Su diámetro es de solo 1.5mm [16], por lo que la porción del campo visual que abarca es realmente baja. Corresponde, concretamente, a tres grados centrales [17]. Del resto del campo de visión se encargan los bastones, las células fotorreceptoras monocromáticas, que se encuentran distribuidas a lo largo de la retina. Estas son especialmente sensibles a la luz y el movimiento, pero no son capaces de percibir colores ni un alto nivel de detalle, por lo que la visión periférica aporta mucha menos información.

Todo ello da sentido a un fenómeno que se observa en los ojos de muchas especies que comparten estas características con el ojo humano: Al contemplar una escena, los ojos no permanecen estáticos, sino que se mueven rápidamente deteniéndose en puntos de interés para construir un mapa mental tridimensional. Estos son los movimientos sacádicos, o sacadas. Suceden a la máxima velocidad angular a la que puede desplazarse el ojo, por lo que no son ni voluntarios ni controlables. Ello constituye un mecanismo por el que los ojos son capaces de recuperar información de una escena muy superior en tamaño a lo que puede percibir en un simple vistazo la fóvea con su campo visual reducido en un intervalo de tiempo relativamente corto. El motivo de que sea así y la fóvea no sea más grande para evitar o disminuir la necesidad de los movimientos sacádicos es una cuestión de eficiencia, pues en el supuesto caso de que el ojo pudiera percibir la escena por completo con el nivel de detalle con el que lo hace la fóvea, el sistema nervioso actual no sería capaz de procesarlo [18].

Cuando en vez de explorar una escena, los ojos apuntan a un punto fijo, manteniéndolo en el centro del campo visual, se da un movimiento de fijación. Estos se asemejan a las sacadas, pero se desencadenan y se concluyen voluntariamente [18].

Por supuesto, las sacadas y las fijaciones no son el único tipo de movimiento que pueda describir el ojo. De hecho, estas solo tienen lugar cuando se observan escenas o sujetos estáticos. Cuando la mirada se centra en un sujeto móvil, los ojos lo acompañan describiendo lo que se conoce como movimientos de persecución suave. Como bien puede anticipar su nombre, son mucho más lentos en comparación. En la misma categoría de estos, ya que también tienen el propósito de fijar la mirada en presencia de movimiento, se pueden enmarcar los movimientos vestibulares, que son reflejos que ayudan a estabilizar la imagen en la retina ante movimientos de la cabeza [19]. Resulta interesante relacionarlos, pues durante la persecución suele aparecer un ligero movimiento de cabeza reflejo que reduce el movimiento ocular en favor del cervical [20].

Velocidad	**Desencadenado**	
	Voluntario	Involuntario
Lenta	Persecución suave	Vestibular
Rápida	Fijación	Sacada

Tabla 2.1: Tipos de movimientos oculares [19, 18].

En este breve contexto sobre el ojo humano y sus movimientos resulta evidente la estrecha relación entre estos y la atención del individuo. La necesidad de sistemas de *Eye-Tracking* radica en su capacidad para ofrecer una comprensión profunda y detallada del comportamiento visual humano. Al analizar dónde y cómo se concentra la mirada de una persona, es posible obtener información valiosa sobre sus intereses, preferencias y procesos cognitivos [3].

2.1 Usos y aplicaciones

Si bien, ello se defendía cuando se realizaron los primeros estudios sobre el movimiento ocular, estos se hicieron a finales del siglo XIX, motivados por la búsqueda de respuestas sobre neurociencia y psicología cognitiva; en la actualidad, el contexto no es el mismo, pues entonces no existían los ordenadores personales. Ahora que una parte importante de las actividades del día a día resulta impensable sin ellos, el *Eye-Tracking* deja de estar relegado a la academia y la investigación para ser una herramienta poderosa con dos implementaciones clave:

- Interfaces Humano-Computadora (del inglés, *Human-Computer Interface*, *HCI*): Mediante una combinación de hardware y software, se define un sistema de interacción con el que el ordenador utiliza datos de seguimiento ocular como *input* para proporcionar un mecanismo con el que controlar una interfaz de usuario, complementando o sustituyendo el ratón y el teclado.
- Análisis cognitivos y de interacción: Mediante la grabación y el análisis de datos

de seguimiento ocular es posible extraer información valiosa y hacer análisis de dos principales conceptos:

- Interacción: Un uso interesante para los datos obtenidos durante la lectura de un documento o de una página web es la de generar gráficos como mapas de calor de atención visual, o de trayectorias descritas por los ojos, por ejemplo, para evaluar la efectividad de una interfaz o un anuncio [21].

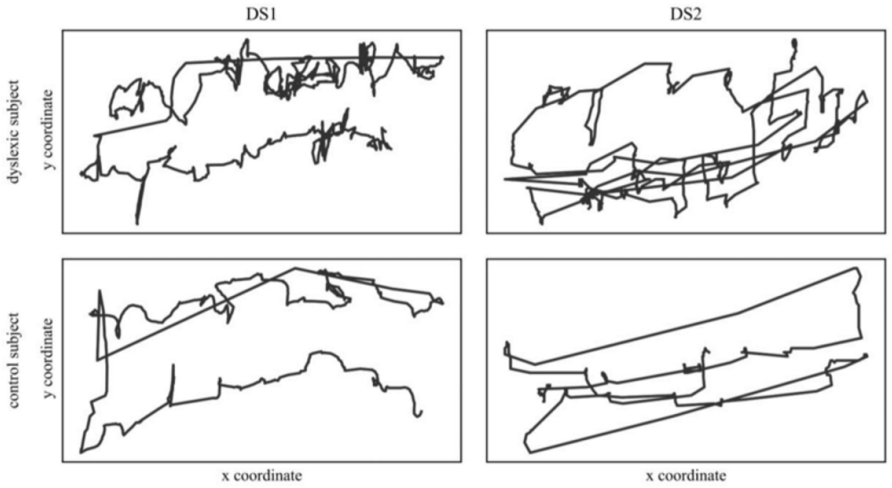


Figura 2.1: Codificación de *Eye-Tracking*, tomada de [21]

- Procesos cognitivos y psicológicos: La visión es un potente indicador de la cognición. En este caso, se buscaría detectar mediante *Eye-Tracking* los diferentes tipos de movimientos oculares; pues es posible correlar la cantidad y duración de cada uno de ellos con la concentración, el estrés y la fatiga. Por ejemplo, una alta presencia de fijaciones sería indicador de concentración. En cambio, una disminución de su duración junto con un aumento de la frecuencia del parpadeo, entre otras cosas, indicaría la presencia de fatiga[20].

Resulta, entonces, aplicable a ámbitos como los siguientes:

- Accesibilidad: Desarrollo de interfaces humano-computadora que ayuden a pacientes con movilidad reducida a interactuar con dispositivos.
- Marketing y usabilidad en interfaces (*UX/UI*): Analizando el desplazamiento de los ojos es posible conocer detalles sobre cómo interactúan los usuarios con los anuncios o las interfaces de usuario, e incluso elaborar mapas de calor con los que visualizar en qué puntos centraron su atención. Esto ayudaría a diseñar interfaces y contenidos más amigables, atractivos y efectivos.

- Educación y aprendizaje: Es posible aplicar *Eye-Tracking* a la educación, para monitorizar los procesos de lectura y comprensión, pudiendo servir como base para desarrollar sistemas de retroalimentación basados en la atención visual.

- Seguridad en automoción: Además de la aplicación más obvia, que es detectar la fatiga y las distracciones en conductores, también se pueden tomar las mismas consideraciones que en apartado de marketing y *UX/UI* anterior, ya que cada vez, las pantallas cobran más protagonismo en la automoción, en detrimento de las instrumentaciones tradicionales. Cuando estas pantallas pasan a controlar funciones elementales como las luces, la selección de marchas o la climatización además de presentar los parámetros del vehículo al conductor, el buen diseño de las interfaces de usuario del software que ejecutan pasa a ser también una cuestión de seguridad.

- Videojuegos y realidad virtual: Existen propuestas para interactuar con videojuegos mediante la mirada, lo cuál resulta muy útil, por ejemplo, en simuladores de vuelo, en los que el entorno virtual es complejo. En otros casos, mediante *Eye-Tracking* se puede medir la atención y el estrés del usuario, que puede servir como una entrada adicional, por ejemplo para adaptar la dificultad en consecuencia. Por último, en los motores 3d de videojuegos, existe un concepto aplicado a la realidad virtual denominado *foveated rendering* que consiste en renderizar con alto nivel de detalle aquello a lo que apunte el centro del campo visual del jugador y bajar la calidad del resto de la escena. De esta manera, la calidad percibida por el usuario es más o menos constante pero la carga en la unidad de procesamiento gráfico disminuye significativamente. Este mecanismo no solo se puede gobernar mediante *Eye-Tracking* sino que además es la forma más conveniente y precisa de hacerlo.

2.2 *Eye-Tracking*

Dado que el proyecto propuesto consiste en un sistema de *Eye-Tracking* basado en *Deep Learning* que no utiliza hardware específico y de hecho cuenta con requerimientos de hardware muy básicos, es preciso analizar otros sistemas de *Eye-Tracking* que existen al momento de la redacción de la memoria, incluidos aquellos que a diferencia de la propuesta sí utilizan hardware dedicado, ya que estos sientan las bases del sistema que se va a desarrollar y en el hipotético caso de que la tecnología se establezca en el mercado, conviviría con los sistemas de *Eye-Tracking* tradicionales.

Para hablar de dispositivos de *Eye-Tracking* contemporáneos merece la pena remitirse a los primeros que se construyeron, ya que el fundamento es en esencia el mismo.

Si bien Louis Émile Javal fue el primer científico que analizó el comportamiento de la mirada, su estudio fue meramente observacional, para analizar los movimientos descritos por los ojos durante la lectura; en 1879. Aunque la naturaleza de este experimento desestima la precisión de los resultados, sirvió para comprobar que el patrón que seguían

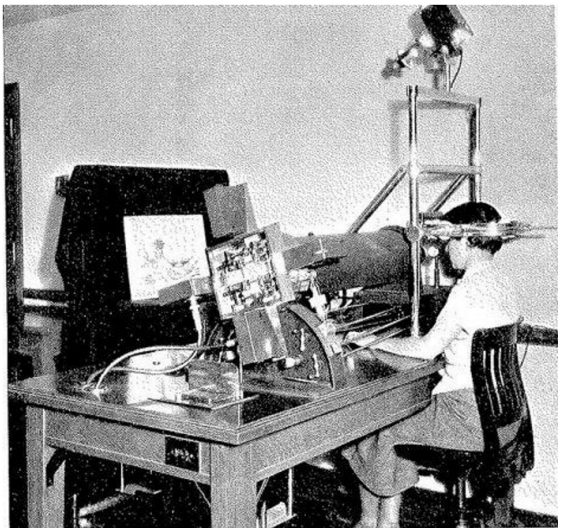


Figura 2.2: Imagen del dispositivo fabricado por Buswell, tomada de [22]

los ojos al leer no era un desplazamiento fluido sino una combinación de fijaciones y sacadas, lo cual se sigue considerando como válido a día de hoy [22].

No fue hasta 1908 cuando Edmund Huey, para profundizar en los descubrimientos de Javal, construyó el primer dispositivo de *Eye-Tracking* a fin de analizar las palabras en las que a lo largo de la lectura se producían las fijaciones. Consistió en un par de lentes de contacto con una pequeña apertura para la pupila y con un puntero acoplado, que cambiaba de posición en función de los movimientos oculares. Tal y como cabría esperar, el dispositivo era demasiado incómodo de utilizar. La solución para que fuera tolerable para sus usuarios fue suministrarles cocaína, lo que en aquel entonces no se consideraba tan descabellado como lo sería en la actualidad. Sus descubrimientos fueron publicados y aún a día de hoy no han sido desestimados [22].

En 1937, Guy Thomas Buswell fabricó el primer dispositivo de *Eye-Tracking* bien documentado. Utilizaba rayos de luz que se reflejaban en los ojos del usuario para ser grabados en una película que luego era analizada, sentando una base muy importante del *Eye-Tracking* en la actualidad, que es la generación de patrones mediante el reflejo de la luz en el ojo. Con este dispositivo concluyó que existe diferencia importante entre la lectura oral y en silencio así como la lectura en diferentes momentos, en función de la edad y el aprendizaje. Además, con este dispositivo, por primera vez en la historia se grabaron los movimientos oculares durante la visualización de imágenes [22, 2].

Hacia la década de 1960, Alfred Lukyanovich Yarbus utilizó un dispositivo con el que llevó a cabo diferentes estudios que aportaron numerosos estudios sobre las fijaciones y su correlación con el interés y la atención. Por primera vez el dispositivo utilizado permitía una interacción relativamente fluida con el entorno del usuario, por lo que para los estudios, se proporcionaban instrucciones de diversa índole. Los resultados fueron

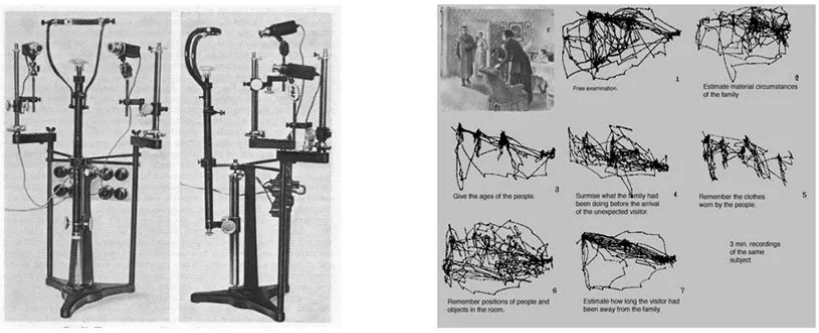


Figura 2.3: Imágenes del dispositivo de *Eye-Tracking* utilizado por Yarbus y muestra de análisis realizados con él. Tomadas de [22].

publicados en su libro *Eye Movements and Vision* [23].

Hacia finales de la misma década, Honeywell y EG&G produjeron y desarrollaron junto con y para la Armada y la Fuerza Aérea estadounidenses los primeros dispositivos de *Eye-Tracking* móviles y no invasivos. Consistían en el conjunto de una unidad optomecánica encargada de capturar las imágenes y una unidad electrónica que reconocía el iris en una pantalla de vídeo, y mediante algoritmos determinaba su centro geométrico y la dirección de la mirada [2].

El siguiente gran avance del *Eye-Tracking* se dio entre los años 80 y 90, gracias al rápido avance de la computación y el procesado de datos. Por un lado, la potencia de los ordenadores de la época hizo posible el *Eye-Tracking* en tiempo real. Por el otro, debido también al avance de la tecnología, surgieron nuevas potenciales aplicaciones para el *Eye-Tracking*, que hasta entonces había quedado relegado a la neurociencia y a la psicología cognitiva, pues en esta época se implementó por primera vez en el ámbito de la experiencia de usuario en la web. Además, los dispositivos se volvieron más tolerantes al movimiento de la cabeza, hasta el punto de ser capaces de distinguirlo y no requerir su inmovilización [23].

Los dispositivos en los que se centrará este análisis son aquellos cuya unidad óptica se monte sobre la pantalla del equipo y midan la dirección de la mirada en unidades relativas a la pantalla, dentro de la misma, ya que sin ser lo mismo que la solución propuesta en este proyecto, son los que, por ámbito, se asemejan lo suficiente como para ser comparados de forma justa. Dentro del sector, existen dos empresas conocidas ofreciendo este tipo de soluciones a día de hoy:

- IRISBOND: Esta empresa ofrece tecnología de *Eye-Tracking* para diversas aplicaciones, como la neuroinvestigación, la sanidad y la industria. Su principal campo de trabajo es el de las interfaces humano computadora (*HCI*), con las que permite controlar ordenadores con la mirada, mejorando la productividad y la seguridad

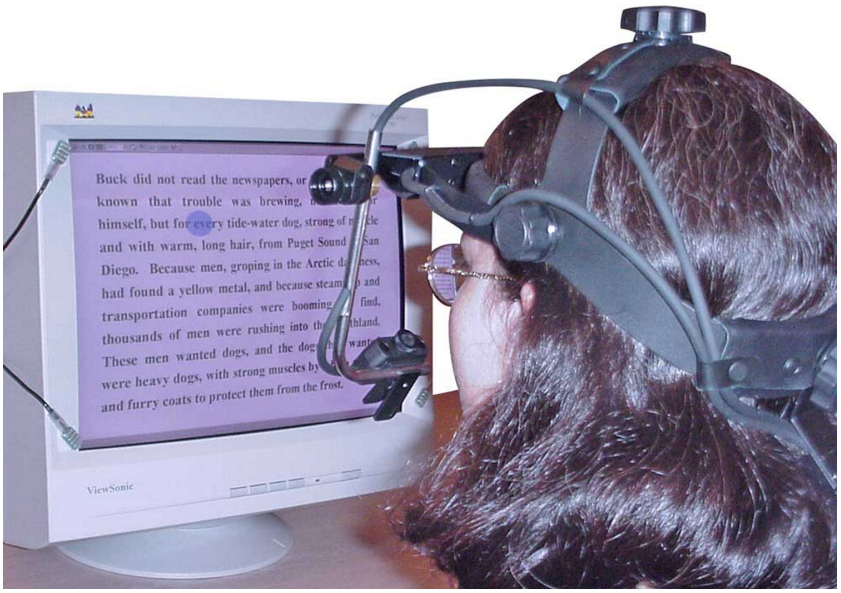


Figura 2.4: Imagen del EyeLink II de SR Research, cuya primera iteración debutó en los años 90. Tomada de [24]

en entornos laborales, o facilitando la interacción con el sistema operativo a personas con discapacidades motrices. Actualmente ofrece una solución de *Eye-Tracking* montada sobre la pantalla, con la peculiaridad de ser compatible con iPadOS, el sistema operativo del iPad de Apple, además de Windows, siendo el primer dispositivo de *Eye-Tracking* compatible con tablets, en un mercado donde lo habitual es la compatibilidad exclusiva con Windows. Se denomina EYE am Hiru. Tiene una tasa de muestreo de 60Hz, trabaja en pantallas de entre 10 y 20 pulgadas y su fabricante declara una exactitud de 0.4° [25].

- Tobii: El principal fabricante de dispositivos de *Eye-Tracking*, que posee desde 2014 la empresa DynaVox, aún más veterana en el sector de la investigación y el desarrollo en *Eye-Tracking*. Fundada en 2001 con sede en Estocolmo, Suecia. Ofrecen su tecnología para múltiples aplicaciones, desde análisis de datos hasta videojuegos y realidad virtual pasando por salud y neurociencia [5].

Dada la relevancia de esta compañía en el sector, es preciso profundizar en su oferta comercial. En la actualidad cuatro 4 modelos de dispositivo de *Eye-Tracking* categorizados como *screen based*.

Por un lado cuentan con la gama profesional, conformada por: Tobii Pro Spark, Tobii Pro Fusion y Tobii Pro Spectrum, bajo pedido y para investigación, con frecuencias de muestreo de 60Hz, 250Hz y hasta 1200Hz respectivamente. Para los fines que están pensados, se proporciona un *SDK* gratuito para todos ellos. Se contactó a la empresa para conocer más detalles acerca de la precisión, a lo que respondieron con con reportes

de métricas de campo y de laboratorio para los modelos Spectrum[1] y Fusion. En los primeros, realizaban pruebas en entornos reales con poblaciones de entre 400 y 500 participantes provenientes de Suecia y China. En los segundos, en condiciones óptimas a modo de control de calidad. Del modelo Pro Spark, no proporcionaron ningún reporte aunque hay datos de laboratorio públicos en su web. Los resultados con los que concluyen se recogen en la tabla 2.2[2].

Dispositivo	Laboratorio (Pantalla completa, condiciones óptimas) **Exactitud — Precisión**	Prueba de campo (Resultados medios) **Exactitud — Precisión**
Tobii Pro Spark	0,45° | 0,26°	– – –° | – ––°
Tobii Pro Fusion	0,37° | 0,21°	0,58° | 0,25°
Tobii Pro Spectrum	0,33° | 0,07°	0,55° | 0,13°

Tabla 2.2: Exactitud y precisión de diferentes dispositivos Tobii en condiciones de laboratorio y de campo [26].

Por el otro lado, ofrecen, en la gama de entrada, para uso doméstico, el Tobii Eye Tracker 5, a través de su propia plataforma de venta online y Amazon por 279 euros. Este modelo proporciona mediciones a una tasa de 33Hz, aunque puede hacerlo artificialmente mediante interpolación a 133Hz; y funciona en pantallas de tamaños desde 15 pulgadas hasta 27 en relaciones de aspecto estándar de 16:9 o hasta 30 en relaciones 21:9. Está pensado para su uso en videojuegos, y como tal, viene con software desarrollado por la propia empresa para ese fin. Es habitual que los dispositivos de *Eye-Tracking* para *gaming* también implementen seguimiento de cabeza, ya que mediante software es posible hacerlo simultáneamente en los mismos datos que utiliza el sistema para *Eye-Tracking* y por supuesto, con el mismo hardware. A pesar de esta clara orientación en cuanto a los usos de este modelo, Tobii también proporciona bajo licencia el *SDK* que incluyen los modelos de gama profesional, para hacer otros usos más profesionales del dispositivo. Si bien el fabricante no proporciona datos de precisión y exactitud de este modelo, se ha encontrado un estudio realizado por terceros que muestra que la exactitud es de 1.01°, lo que en el monitor del equipo que utilizaron (24 pulgadas, 16:9, 1920x1080 píxeles) los propios autores calculan que equivale a 30 píxeles [27].

[1] Muestreando a 600Hz

[2] Para conocer más detalles acerca de las métricas de exactitud y precisión, comunes en *Eye-Tracking*, véase la sección 5.5 del Capítulo 5.

2.3 *Eye-Tracking* sin hardware específico

Esta categoría engloba a todas las opciones de *Eye-Tracking* que no utilizan más periféricos que una cámara facial del dispositivo en el que se implementan, pudiendo ser incluso embebida en el caso de portátiles, móviles o tablets. Su lógica de negocio reside casi exclusivamente en un software que contiene los algoritmos necesarios para procesar las imágenes e inferir la dirección de la mirada a partir de los patrones oculares reconocidos en dichas imágenes. Habitualmente y sobre todo en sistemas recientes, se utiliza Aprendizaje Profundo para este fin. Este tipo de sistemas necesitan de muchos menos recursos para ser desarrollados y operados, por lo que existen propuestas relativamente nuevas de empresas pequeñas y emergentes que no necesariamente cuentan con los años de investigación y de desarrollo que sí tienen IRISBOND y Tobii. Aquí algunas de las más destacables:

- Eyeware: Compañía especializada en soluciones exclusivamente basadas en software para *Eye-Tracking* en ordenadores establecida en Suiza en 2016. Son los desarrolladores de Beam Eye Tracker, un software de *Eye-Tracking* sin hardware dedicado orientado a videojuegos; y GazeSense, un sistema similar pero de propósito general, que ofrecen en formato de aplicación, para investigadores; y de *SDK*, para desarrolladores. Ambas incorporan seguimiento de cabeza. Los precios de sus herramientas son 30$ y 199$ (pago único) para Beam Eye Tracker y Gaze Sense respectivamente. La tasa de muestreo que proporcionan va estrechamente relacionada con la tasa de generación de imágenes de la cámara web, pero la acotan entre 30Hz y 60Hz en un caso general, pudiendo llegar incluso hasta 120Hz en ciertos modelos específicos. Por desgracia, no ha sido posible acceder a sus estudios sobre la precisión y exactitud de sus soluciones [28].

- RealEye: Es la empresa desarrolladora del que probablemente el sistema de *Eye-Tracking* sin hardware dedicado basado en software más completo hasta la fecha. Se fundó en Polonia en 2017. Su principal caso de uso es el análisis de *UX/UI*; es decir, estudiar el comportamiento de la mirada durante la utilización de interfaces de usuario o el consumo de contenidos como páginas web o anuncios. Por ello, proporcionan herramientas de análisis para realizar mapas de calor de atención visual, gráficas de mirada y fijación, de datos sin procesar y volcado de estadísticas. La frecuencia de muestreo de su sistema va desde 30Hz a 60Hz, por supuesto dependiendo de las especificaciones de la cámara utilizada, pudiendo funcionar en un modo de características reducidas con un mínimo de 15 imágenes por segundo. La empresa proporciona mediante solicitud con dirección de correo académica o empresarial los reportes de un estudio sobre la precisión y la exactitud de su sistema, donde concluyen que la exactitud en su estudio fue de 140.30 píxeles y la precisión, de 147.20 píxeles (en la media de la distancia euclídea entre fijaciones sin excluir valores atípicos) en un equipo de sobremesa con un monitor de 24 pulgadas y resolución 1920x1080. También cuentan con una versión de su sistema para

smartphones, y como peculiaridad, tanto la versión de PC como en la de dispositivos móviles necesitan conexión a internet, a diferencia de las demás propuestas aquí analizadas [29].

- Brown University Department of Computer Science: No son una empresa, sino un departamento de la Universidad Brown University, ubicada en Rhode Island, Estados Unidos. Como parte de su investigación desarrollaron entre 2016 y 2023 un sistema de EyeTracking llamado WebGazer.js, basado en cámara web como una librería de Javascript, que permite integrarlo en aplicaciones web de forma sencilla. Esto, junto a su licencia de código abierto GPLv3 y su capacidad de autocalibración mediante clics y movimientos del cursor del ratón lo diferencian de cualquier otro sistema de *Eye-Tracking* sin hardware dedicado disponible hasta la fecha, ya que además de calibrarse de forma transparente al usuario, esto también le permite adaptarse en tiempo real a los movimientos de la cabeza. En su web tienen publicado un artículo en el que plasman los reportes de un estudio del error de las predicciones de sus modelos de *Deep Learning*; una vez más, medidas con un monitor de 24 pulgadas, aunque en este caso de relación de aspecto 16:10 y resolución 1920x1200, algo menos típica. El promedio que calculan para todas sus librerías (seleccionadas en función de la actividad instruida al usuario) tiene las medias (métrica de veracidad) y desviaciones estándar (métricas de precisión) calculadas según la distancia euclídea media según el modelo viene recogido en la tabla 2.3 [30].

- Apple: En la versión 18 de iOS, el sistema operativo de sus teléfonos inteligentes, Apple implementó una función de accesibilidad basada en *Eye-Tracking* sin hardware específico, ya que se apoya en la cámara de los dispositivos, en todos los iPhone posteriores al 12 incluyendo a este. Sirve para controlar la interfaz gráfica del dispositivo solo con la mirada, como si de una *HCI* se tratase. No hay datos oficiales publicados acerca de las características de este sistema [31]. Tampoco ha sido anunciado por Apple como una característica mayor, por lo que se entiende que la fase en la que se encuentra es experimental. Aún así, a diferencia de los demás sistemas, este se ha podido probar durante la realización del proyecto de este TFG. La propuesta es similar. De hecho, la pantalla de calibración, que se muestra cada vez que se activa la función de seguimiento ocular, es muy similar a la interfaz de uno de los juegos del sistema de adquisición propuesto por Raúl Barba en su Trabajo de Fin de Grado en 2021 [8].

2.4 Detección facial

Uno de los principales motivos de realizar detección facial en un sistema de *Eye-Tracking* tradicional basado en hardware es la orientación a un entorno no controlado, ya que incluyendo los datos generados por esta función en los cálculos de posprocesado, es posible

Modelo	**Media**	**Desviación estándar**
Regresor lineal	256.9 píxeles	75.0 píxeles
Regresor *ridge*	232.4 píxeles	92.3 píxeles
Regresor *ridge* + muestras extra con *buffer* de fijación*	251.5 píxeles	89.0 píxeles
Regresor *ridge* + muestreo de movimientos de cursor*	174.0 píxeles	91.6 píxeles
Regresor *ridge* + combinación de las anteriores técnicas*	210.6 píxeles	86.3 píxeles
TOTAL (Calculado para esta memoria, no proviene de la fuente original)	Max.: 256.9 píxeles Mín.: 174.0 píxeles Media: 225,08 píxeles	Max.: 91.6 píxeles Mín.: 75.0 píxeles Media: 86,84 píxeles

Tabla 2.3: Comparativa de precisión en píxeles de los modelos regresores de WebGazer.js [30]

salvar el error causado por ciertos movimientos del usuario, mejorando la tolerancia. También es posible darle un uso a estos datos fuera del dominio del sistema. La aplicación más obvia son los videojuegos.

Prueba de ello es que las dos soluciones para este fin mencionadas en este análisis, el dispositivo Tobii Eye Tracker 5 [27] y el software Beam Eye Tracker de Eyeware [28], incorporan seguimiento de cabeza, que en ambos casos se realiza procesando las coordenadas de la región de interés en la que se detecta la cara del usuario. En videojuegos estos sistemas se suelen utilizar para controlar la cámara en primera persona. Comúnmente, para esto se han utilizado soluciones de seguimiento de cabeza, más que probadas y con buenos resultados, como por ejemplo TrackIR, que es un dispositivo de Head-Tracking que utiliza en conjunto un sensor infrarrojo montado en la pantalla y un iluminador que se coloca en la cabeza del usuario. Por ello, los sistemas de *Eye-Tracking* con seguimiento de cabeza proporcionan un valor añadido a una solución existente en ese campo, ya que añade el *Eye-Tracking* por un lado y por el otro –al utilizar detección facial para hacer Head-Tracking– no requiere colocar ningún dispositivo sobre el usuario [5].

Por el contrario, si se hace referencia a sistemas basados en software, como el que se propone, la detección facial se vuelve un añadido imprescindible. Esto se debe a que un sistema de *Eye-Tracking* convencional cuenta con sensores ópticos orientados directamente a los ojos o al rostro, que son evidentemente su objeto de estudio. Sin embargo, en el caso de no usar hardware dedicado, el elemento análogo a los sensores ópticos es la cámara web del equipo del usuario, que captura una escena mucho más grande. Para discriminar de la imagen todo aquello que no sea el rostro, se utiliza detección facial

para caracterizar la región que lo contiene. Si se requiere trabajar exclusivamente con los ojos, se puede utilizar un detector de características entrenado para identificar los ojos o recortar una porción estimada a sabiendas de que contendrá los ojos (por ejemplo, la mitad superior del rostro). En cualquier caso, el espacio de búsqueda debe quedar acotado al rostro y por ello es necesario detectarlo primero.

Resulta complicado hacer un análisis del estado del arte de la detección facial como se ha hecho con las soluciones de *Eye-Tracking*, pues en el mercado no existen dispositivos ni sistemas completos de detección facial como tal, sino librerías y *APIs* para ser implementadas en otros desarrollos, como los clasificadores en cascada Haar Cascade [32] o la red neuronal convolucional YuNet [33]. En este caso no es preciso hablar de estado del arte, sino de componentes clave del proyecto. Se desarrollarán los detalles sobre estas soluciones y su implementación en el sistema en el Capítulo 5.

Por supuesto, el seguimiento de cabeza (y por consecuencia la detección facial) no son la única solución para mejorar la usabilidad y la precisión de los sistemas en entornos no controlados. Por ejemplo, el sistema software de RealEye, pensado para análisis de usabilidad en *UX/UI*, no cuenta con seguimiento de cabeza aunque está pensado para ser utilizado fuera del laboratorio. En su defecto cuenta con una solución que ellos han bautizado como Virtual Chinrest (reposabarbillas virtual) haciendo referencia al punto de apoyo que se solía usar para fijar la cabeza en dispositivos no tolerantes a su movimiento. El programa requiere de una calibración al inicio de cada sesión, en la que memoriza la posición de los ojos. Ante movimientos sutiles de la cabeza, ajusta los cálculos para seguir proporcionando datos válidos a pesar de la desviación. Cuando esta se vuelve significativa, muestra una ventana indicando la posición de los ojos al momento de la calibración sobre la cámara del usuario, para ayudarle a recuperarla [29]. Se muestra la interfaz en la figura 2.5.

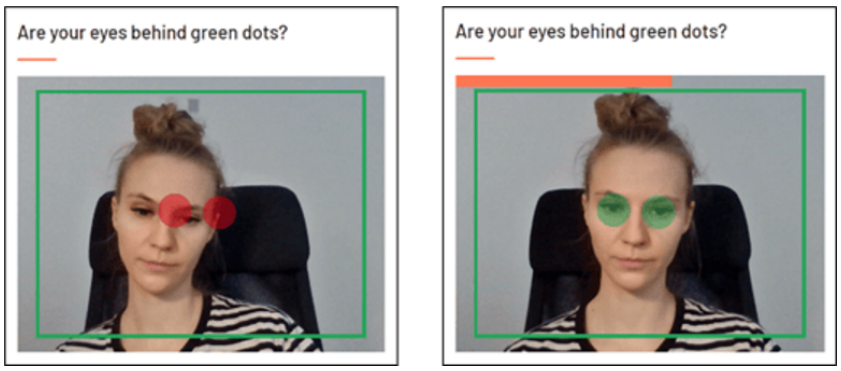


Figura 2.5: Virtual Chinrest de RealEye. Obtenido de [29]

Capítulo 3

Aplicación Web para adquisición de datos de entrenamiento

Tal y como se introdujo en el Capítulo 1, para obtener los datos de entrenamiento necesarios para construir el sistema de aprendizaje automático que estime la posición de la mirada, ha sido necesario poner en marcha el sistema cuyo desarrollo fue iniciado por Raúl Barba Alonso como parte de su Trabajo de Fin de Grado. Es una aplicación web pensada para adquirir datos de entrenamiento de *Eye-Tracking*. Una de las líneas futuras que definió fue utilizar los datos de esta aplicación para el desarrollo de modelos, a explorar en el presente trabajo.

3.1 Descripción general

La aplicación cuenta con un *frontend* desarrollado puramente en HTML, CSS y JavaScript, que contiene formularios de gestión de usuarios, comprobaciones de cámara web y juegos cognitivos, que denomina experimentos, que solicitan movimientos oculares al usuario y capturan imágenes y otros datos asociados. Y un *backend* desarrollado en PHP que implementa la lógica de negocio de la aplicación, utilizando para el almacenamiento una base de datos MySQL y el sistema de ficheros del servidor en cuestión [8].

El servidor en el que se aloja es una máquina a la que de ahora en adelante se hará referencia por su nombre de host, `saqqara`. Ejecuta sobre Linux el servidor web Apache con PHP y el servidor MySQL de la base de datos. `saqqara` accede a internet mediante uno de los *firewalls* de la red del grupo de investigación, y gracias al reenvío de puertos y al registro de un nombre canónico en el sistema de nombres de dominio (DNS) de la red del Grupo y/o de la Escuela, fue accesible en la URL `https://eyetracking.lpi.tel.uva.es`.

Para adaptar la aplicación a las necesidades del proyecto actual se modificaron y se mejoraron ciertos aspectos de la misma. Algunas partes fueron descartadas y desarrolla-

das desde cero y otras fueron implementadas por primera vez. Para obtener más detalles técnicos sobre el despliegue y el desarrollo de la aplicación así como de las mejoras realizadas para su implantación en este proyecto, léase el Apéndice C.

Figura 3.1: Página principal de la aplicación web de adquisición. Autoría propia.

Al tratar con datos biométricos hay que tener especial cuidado y ciertas consideraciones. Entre ellas, que los usuarios deben ser informados en cualquier caso del tratamiento de sus datos y que deben tener la posibilidad de desistir y borrar toda su información almacenada si así lo desean. Si bien mantener anónimos los datos parecía la forma más sencilla de reducir la criticidad de los mismos, no es del todo compatible con el requisito que se acaba de plantear. Además, de cara al procesamiento cuando se pase a la fase de análisis, resulta casi imprescindible asociar los datos con el usuario del que provienen para fraccionar el conjunto en función de los sujetos para que sendos validación y test se hagan correctamente y las métricas sean justas. La solución final consiste en implementar un control de sesiones de usuario que durante el registro proporcione un código de cuatro caracteres aleatorios requerido para posteriores inicios de sesión. Este código identificará unívocamente a todos los recursos almacenados en el servidor por el usuario al que pertenezca. Así los usuarios podrán borrar sus datos desde su sesión si así lo desean, y durante el análisis, los registros se podrán filtrar por usuario, manteniendo así

el anonimato, ya que se almacenan solo los datos personales justos y necesarios para el proyecto. Durante el registro se solicita lo siguiente:

- Contraseña (Se almacena un *hash*)
- Edad
- Género
- Color de ojos
- Dimensiones de la diagonal de la pantalla (internamente se obtienen la resolución de la pantalla completa en píxeles y con esta la relación de aspecto para descomponer la diagonal)
- Posición de la cámara respecto a la pantalla.

Para completar el registro, además de aportar los datos anteriores, es necesario aceptar los términos y condiciones, elaborados en base al Reglamento (UE) 2016/679 de 27 de abril de 2016 (Reglamento General de Protección de Datos) y a la Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales [34].

Una vez iniciada la sesión, se muestra la pantalla de selección de experimentos (3.2): tres juegos cognitivos que utilizan la pantalla al completo y; mediante información textual, visual y/o acústica dan instrucciones al usuario para generar situaciones que son fotografiadas con la cámara del equipo y etiquetadas en la base de datos ("experimentos" en la aplicación y en esta memoria de ahora en adelante). También se muestran en la barra de navegación, solo a partir de este momento, ya que hasta que no se inicia la sesión permanecen ocultas, las opciones de modificación y eliminación del usuario en cuestión.

Para los usuarios con privilegios de administrador (controlados por un parámetro de la tabla de usuarios de la base de datos) aparece a mayores una opción para visualizar la información esencial de los registros almacenados hasta el momento (3.3).

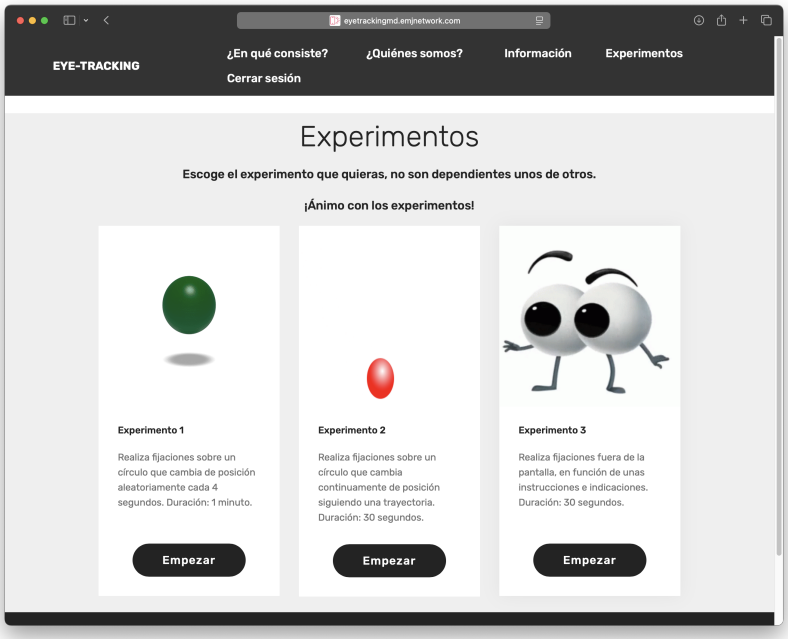


Figura 3.2: Pantalla de selección de experimentos de la aplicación de adquisición.

Figura 3.3: Pantalla de previsualización de muestras de la aplicación de adquisición.

Si bien hay tres experimentos, son dos las categorías que los engloban, y que corresponden a los casos de uso de sus datos durante el análisis:

- Seguimiento ocular de un objeto en pantalla: Estos almacenan varios registros a lo largo de la realización que contienen, entre otras cosas, imágenes y coordenadas en píxeles, lo cuál es en esencia, lo que se va a utilizar en el entrenamiento de los modelos regresores que inferirán la dirección de la mirada en base a las imágenes provenientes de la cámara del equipo. A esta categoría corresponden:
  - Experimento 1: El propósito de este experimento es adquirir datos al provocar el movimiento ocular de desencadenado voluntario rápido, la fijación. En este aparece una circunferencia negra que periódicamente (con un periodo de 4 segundos) cambia aleatoriamente de posición a lo largo y ancho de la pantalla durante 60 segundos. Se le pide al usuario que cada vez que reaparezca, con el ratón, haga clic sobre la misma. Cuando lo hace, se calcula la desviación (entendiendo como tal la distancia euclídea en píxeles del clic respecto de la circunferencia) y se comprueba. Si está debajo del umbral establecido, la cámara toma una imagen del usuario que se guarda temporalmente en memoria junto a otros datos asociados, entre ellos las coordenadas en píxeles de la posición de la circunferencia y las coordenadas en píxeles donde el usuario hizo clic. Dado que esto se hace una vez por cada cambio de posición de la circunferencia (o no se hace si el usuario no hace clic) y se hace uno cada 4 segundos, en todo el experimento, que dura 60, se pueden hacer hasta 15 capturas. Si la diferencia de la posición de la circunferencia respecto al clic es superior al umbral, la iteración se descarta y no se guarda la imagen. Para informar al usuario sobre si esto ocurre y mejorar la usabilidad, al hacer clic aparece temporalmente una circunferencia extra bajo el cursor. Si se considera válido, sobre esta nueva circunferencia aparece un contorno verde. Si resulta no ser válido, ocurre lo mismo pero el contorno es rojo. Al concluir el experimento se suben al servidor los datos almacenados en memoria y se muestra la puntuación obtenida por el usuario durante la realización del experimento así como en otras realizaciones anteriores, tanto suyas como de otros usuarios; como parte de un sistema de gamificación que premia las desviaciones bajas.
  - Experimento 2: Con este experimento se adquieren datos provocado el movimiento ocular desencadenado voluntariamente antagónico a la fijación por su velocidad, la persecución suave. A nivel lógico se desarrolla partiendo del anterior y reutilizando gran parte de su código, pero con ciertas diferencias. La primera es que la circunferencia deja de aparecer estática en posiciones aleatorias de la pantalla para desplazarse siguiendo una de cuatro trayectorias predefinidas (circunferencia, elipse o curva de Lissajous con relación 1/3 o 7/9) durante 30 segundos. La selección de las mismas es aleatoria, sucede al comienzo del experimento y la trayectoria permanece hasta la finalización del mismo. En cualquier caso se muestra de fondo en la pantalla. Es importante notar que debido a que las trayectorias de la circunferencia vienen gobernadas

por la función continua que las define, existe una cierta correlación entre las muestras adquiridas en una realización de este experimento que se vuelve muy alta cuando son contiguas, y que por supuesto no existe en el experimento anterior. Este efecto debe ser tenido en consideración durante las fases de entrenamiento y validación de modelos.

- Generación de situaciones anómalas mediante instrucciones: De este tipo es el experimento 3, que proporciona instrucciones al usuario para desviar la cabeza del centro de la pantalla o apartar la mirada de la misma. Se consideran situaciones anómalas porque se alejan de lo que sería objeto de inferencia de los modelos principales del sistema, aquellos entrenados con datos generados con experimentos del primer grupo. Es decir, no son caras frente a la cámara mirando a diferentes puntos de la pantalla. De hecho, para describir el lugar en el espacio al que se dirige la mirada del usuario no se usan píxeles. Se rigen por un sistema de coordenadas de 7x7 cuya equivalencia se define en la figura 3.4.

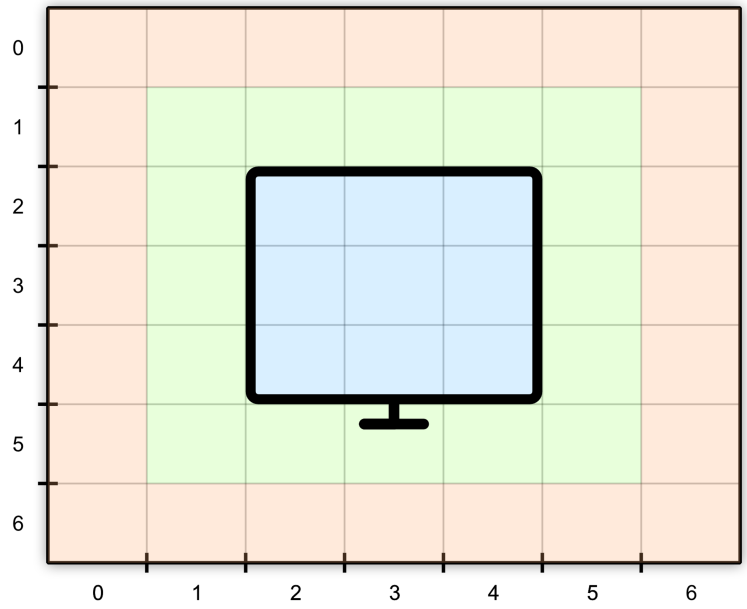


Figura 3.4: Sistema de coordenadas Experimento 3. Autoría propia.

La cuadrícula de 7x7 representa el entorno del usuario frente a la pantalla de su ordenador, pudiendo ser esta un monitor en un escritorio o la pantalla de un ordenador portátil.

- La zona interior, resaltada en azul y comprendida en la ilustración de un monitor representa lo que parece, la pantalla del dispositivo. Cuando durante la sesión del experimento se hace referencia esta zona, se proporciona la instrucción “dentro de la pantalla” u otras similares.

- La zona intermedia, resaltada en verde representa las proximidades de la pantalla. Cuando el experimento se refiere a esta zona, utiliza la instrucción "ligeramente hacia la derecha/la izquierda/arriba/abajo (o diagonales resultantes de la combinación de estas), fuera de la pantalla". En ocasiones, sustituye "ligeramente" por "45 grados", manteniendo el mismo significado.

- La zona exterior, resaltada en un color rojizo, representa los extremos de la visión del usuario frente a la pantalla. La instrucción que suele proporcionar el experimento para referirse a esta zona es "totalmente hacia la derecha/la izquierda/arriba/abajo (o diagonales resultantes de la combinación de estas), fuera de la pantalla". En ocasiones, sustituye "totalmente" por "90 grados", manteniendo el mismo significado.

Este mecanismo se implementa de tal forma que se contemplan dos casuísticas: una en la que el usuario mira a un determinado punto de su entorno moviendo la cabeza para ello; y otra en la que mantiene la cabeza orientada al centro de la pantalla y dirige únicamente la mirada al punto que especifiquen las instrucciones se proporcionan más detalles en la siguiente sección, cuando se describen los campos de la tabla posición_tiempo, de la base de datos de la aplicación). Esto hace posible generar registros del experimento 3 artificialmente a partir de los registros de los experimentos 1 y 2 convirtiendo convenientemente las coordenadas a este sistema, aunque esto solo es recomendable si al hacerlo queda un balance del 50 % entre muestras positivas y negativas [35].

3.2 Estructura de la base de datos

Véase la figura 3.5.

3.3 Recuperación de datos en el entorno de trabajo

Una vez el desarrollo de la aplicación web se dio por concluido, se puso el servidor en marcha y se informó a un grupo de usuarios voluntarios para que hicieran adquisiciones durante un periodo de varias semanas. En paralelo se comenzaron las tareas de preparación y configuración del entorno de trabajo en el que se hizo el análisis de los datos obtenidos y se llegaron a realizar las primeras pruebas con conjuntos de datos anteriores y más pequeños que el conjunto final utilizado para entrenar y evaluar el sistema a lo largo de esta memoria. Ello demostró la importancia del número de sujetos y adquisiciones a la hora de mejorar la precisión de la red, además de una clara limitación por la naturaleza de este proyecto: el número de sujetos es escaso como para tener total libertad a la hora de realizar pruebas. Para consultar más detalles sobre este asunto, véase el Capítulo 6.

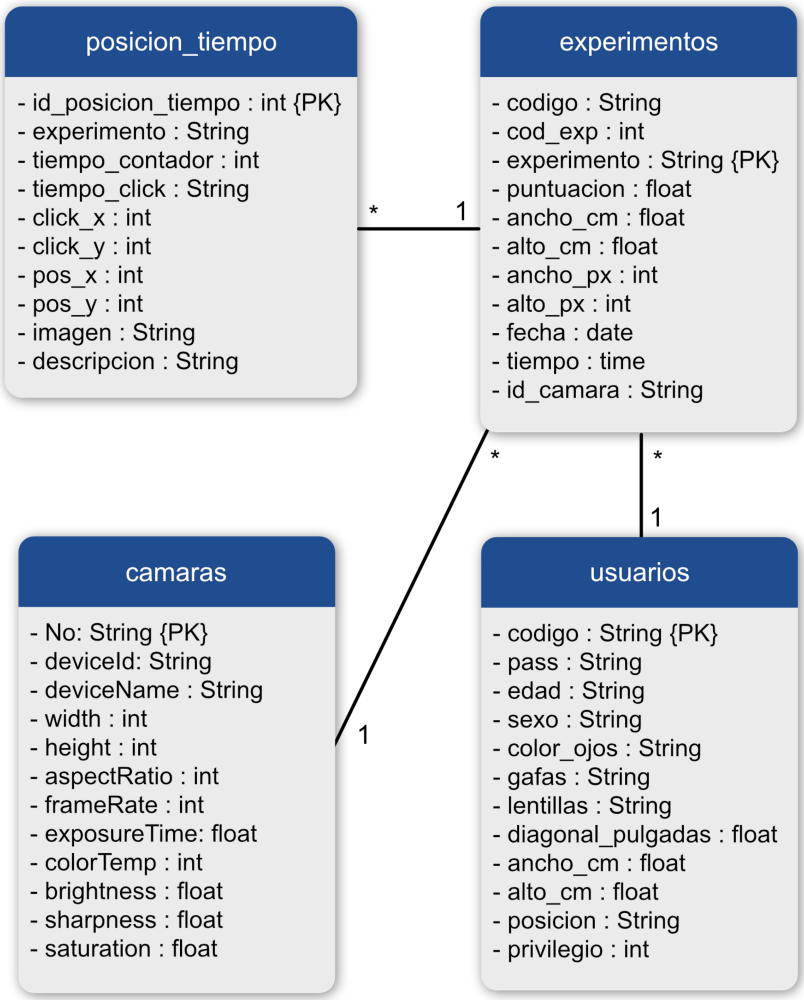


Figura 3.5: Estructura de la base de datos de la aplicación. Autoría propia

El sistema de adquisición fue pensado para ser totalmente disjunto del proyecto actual. Si bien, dado que tanto la máquina que aloja la web como la máquina utilizada para el desarrollo de la arquitectura de *Eye-Tracking* se encuentran próximas en la red y dentro de la primera se ejecuta el servidor de la base de datos y el servidor web, que mediante HTTP puede proporcionar las imágenes adquiridas, sería posible plantear la solución como una implementación conjunta en la que los métodos utilizados en el proceso de entrenamiento de modelos se comuniquen con el servidor para obtener los datos, lo que abre un abanico de posibilidades que se explican con detenimiento hacia el final del Capítulo 7. Sin embargo, se prefirió mantener la disyunción inicial, y generar un conjunto de datos perfectamente útil fuera de línea que se pueda analizar desde cualquier entorno de trabajo, lo que aporta resiliencia a la solución, al poder funcionar independientemente del estado del servidor, de la aplicación o de la red. Para recuperar los datos del servidor y pasarlos a un formato manejable en el entorno de trabajo en el que se analizarán, se desarrolló un script en Python en el servidor `isis`, en el que más tarde se configuraría dicho entorno. Su cometido es automatizar la tarea de volcar los datos desde el servidor a un formato adecuado en el sistema de ficheros de la máquina en la que se ejecuta, en este caso `isis`, pero con pequeños ajustes que pudieran ser necesarios por el contexto de las máquinas (credenciales de usuario, direcciones IP, puertos o conectividad directa entre ellas) sería posible utilizarlo en cualquier otra.

El formato en el que se presentan los datos tras la ejecución del script es el siguiente (ver figura 3.6):

- Un fichero con extensión CSV con una marca de tiempo que incluye desde el año hasta los segundos como nombre. Este puede contener tantas filas como la tabla `posicion_tiempo` al momento de la ejecución del script. De hecho, los datos se almacenan como una concatenación de los campos de la tabla `posicion_tiempo` con la gran mayoría de campos del resto de tablas, utilizando las relaciones de la base de datos para asociar los valores correctamente (se omiten solo aquellos irrelevantes para el análisis de los datos, como los identificadores enteros incrementales o los *hashes* de las contraseñas de acceso).
- Un directorio con el nombre compuesto por la misma marca de tiempo que el fichero anterior concatenada a "`_imageset`". Este a su vez contiene el mismo árbol de directorios que el directorio "`capturas`" en el servidor web, incluyendo el mismo. Esto permite referenciar cómodamente el conjunto de datos en el sistema de ficheros mediante el valor de la marca de tiempo, pudiendo utilizar la misma ruta que utilizaba el servidor web originalmente, por lo que no hace falta modificarla al volcar los datos.

Para lograr esto, el script utiliza la librería `sshtunnel` para establecer túneles SSH entre el servidor web y la máquina local, con los que reenvía los puertos de los servicios HTTP y MySQL, lo que simplifica las labores de configuración a la vez que hace el proceso seguro, pues de esta forma el servidor aplica las restricciones de conexión locales,

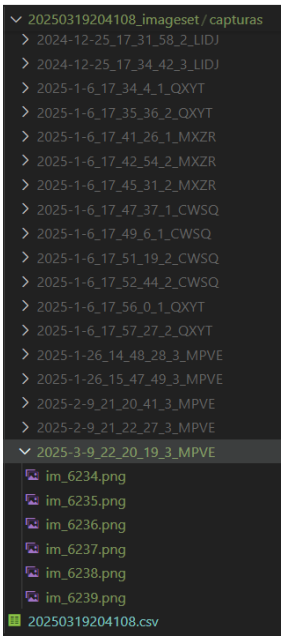


Figura 3.6: Estructura del conjunto de datos en VS Code.

más permisivas que aquellas de las conexiones remotas; pero la conexión se cifra punto a punto, como ocurre siempre en el caso del protocolo SSH. Por supuesto, si bien no es necesario que HTTP y MySQL sean accesibles desde fuera de la máquina para la ejecución del *script*, sobra decir que para establecer un túnel SSH sí que debe poderse establecer una conexión mediante este protocolo. Para realizar la consulta que devuelve el contenido del fichero CSV se utilizan `mysql.connector` y `pandas`, que conjuntamente establecen la conexión y vuelcan los datos obtenidos al fichero CSV.Por último, para descargar las imágenes, se utiliza la librería `requests` para ejecutar recursivamente el método `GET` de HTTP sobre los *endpoints* en los que residen dichas imágenes en la aplicación.

3.4 Conjunto de datos

Así pues, se ha procedido a volcar una última vez los datos del servidor para trabajar en esta memoria con una iteración lo más actualizada posible, a fecha del 20 de marzo de 2025 a las 20:44:08 horas. Las características cuantitativas de este conjunto son las siguientes:

- 15 sujetos diferentes.
- 1380 registros totales (imagen más datos asociados), de los cuales (Ver figura 3.7):

  - 699 pertenecen al Experimento 1.
  - 618 pertenecen al Experimento 2.
  - 63 pertenecen al Experimento 3.

- La totalidad de las cámaras utilizadas para adquirir los datos del conjunto estaban colocadas sobre la pantalla. La gran mayoría de ellas (constituyendo 1234 registros) capturaron las imágenes en formato apaisado a resolución *HD* (1280x720). Ver figura 3.9.

- Existe un usuario cuya cámara capturó imágenes a resolución *FullHD* pero en vertical (relación de 9:16).

- El 87 % de los registros los realizaron usuarios con ojos de color marrón. El 13 % restante corresponde a usuarios con ojos verdes. Por tanto, solo hay dos posibles colores de ojos en el conjunto, como se muestra en la figura 3.11.

- La pantalla más grande utilizada para realizar experimentos tenía una diagonal de 27 pulgadas, seguida de otra de 24. El resto se hizo con pantallas de 16 pulgadas o menos, todas ellas presumiblemente de ordenadores portátiles. Ver la figura 3.8

- La relación de aspecto de las pantallas utilizadas para realizar experimentos fue de en su mayoría 16:9 o 16:10, siendo la primera muy común en todo tipo de dispositivos y la segunda, más habitual en portátiles. Véase la figura 3.10.

- Tanto la resolución de las pantallas como las coordenadas vienen expresadas en unidades del *viewport*, que es el área de la ventana del navegador a través del cual se visualiza la página. Su tamaño se mide en una magnitud que puede diferir de los píxeles físicos de la pantalla en función de los ajustes de escalado del sistema operativo y del navegador. En cualquier caso, estas coordenadas se normalizarán más adelante, por lo que no aportan ningún valor a este análisis.

Caracterizando el número de muestras que cumplen ciertas condiciones en función de los sujetos se pueden conocer ciertos detalles que sirvan para justificar ciertos comportamientos del sistema que se entrene con este conjunto de datos, por ejemplo, al evaluarlo mediante validación cruzada.

- Hay un promedio de 92 muestras por sujeto, como muestra la figura 3.12.

- Un 28.6 % de los sujetos llevó gafas durante la realización de los experimentos, como se aprecia en la figura 3.15.

- Se ha localizado a los usuarios cuyas cámaras emitieron imágenes a resoluciones por encima de la media, o utilizaron pantallas sensiblemente más grandes para realizar los experimentos. Ver figuras 3.14 y 3.13.

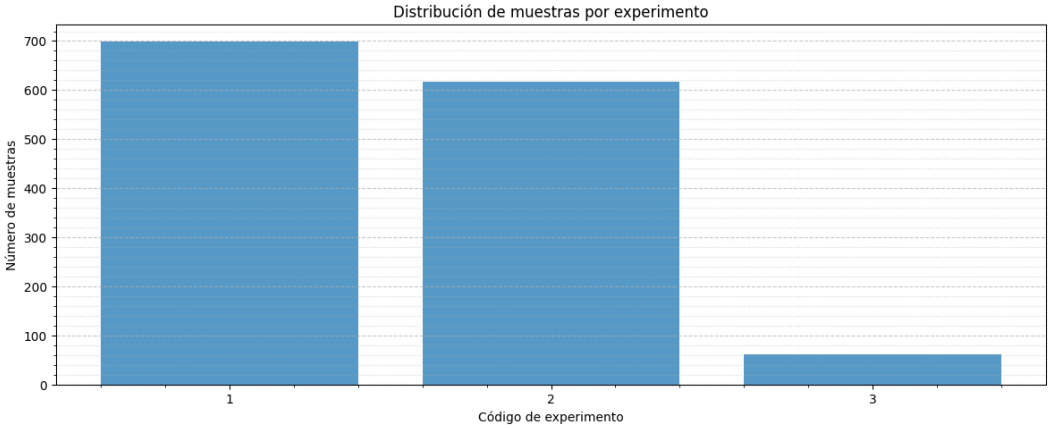


Figura 3.7: Distribución de muestras por tipo de Experimento.

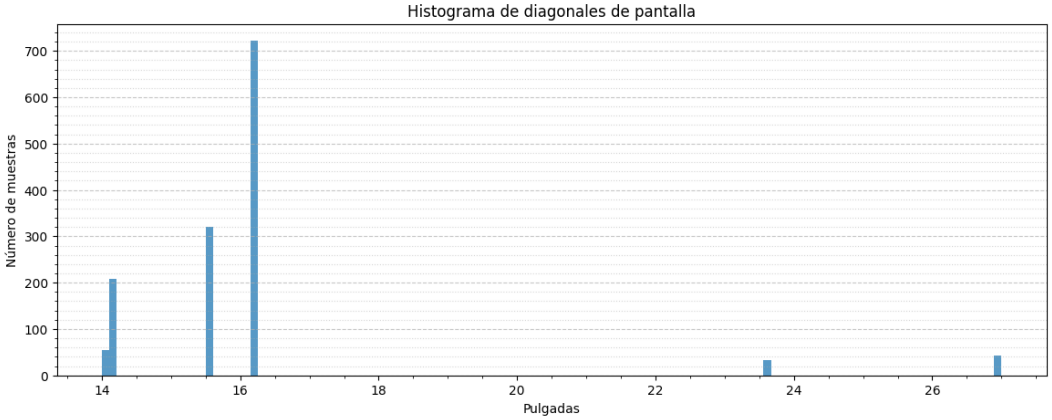


Figura 3.8: Distribución de muestras por tamaño de diagonal de pantalla en pulgadas.

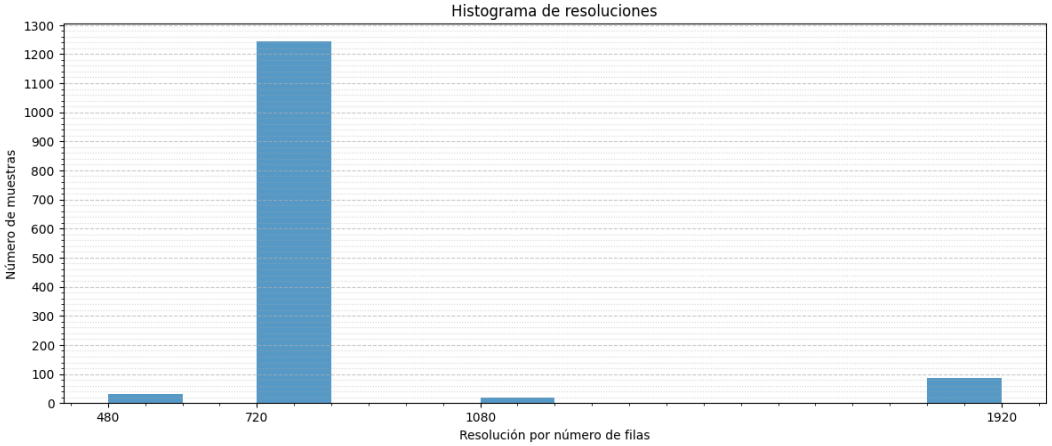


Figura 3.9: Distribución de muestras por resolución vertical de cámara.

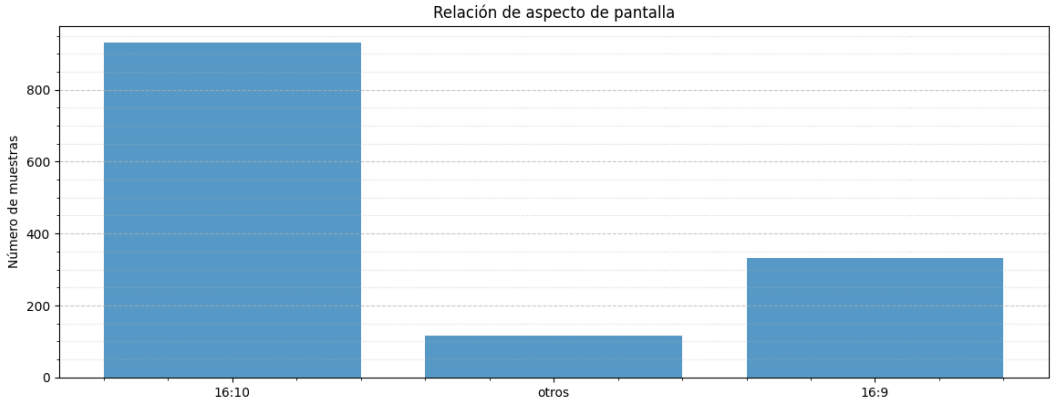


Figura 3.10: Distribución de muestras por relación de aspecto de pantalla

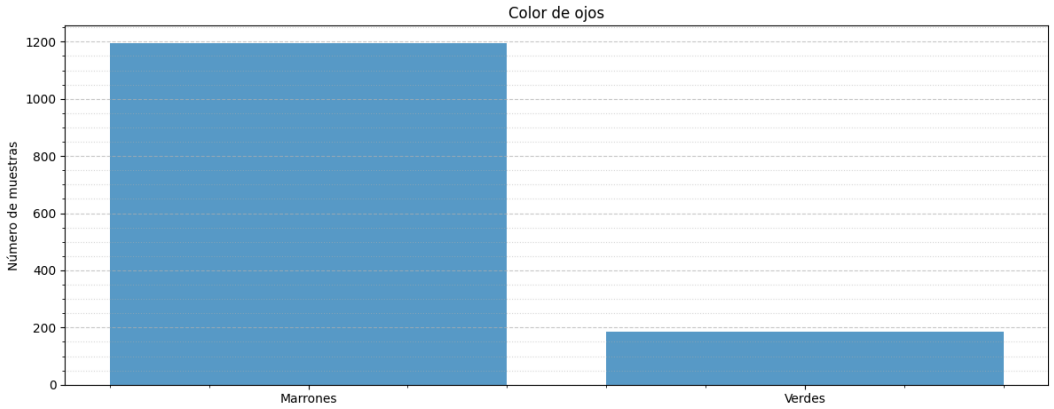


Figura 3.11: Distribución de muestras por color de ojos.

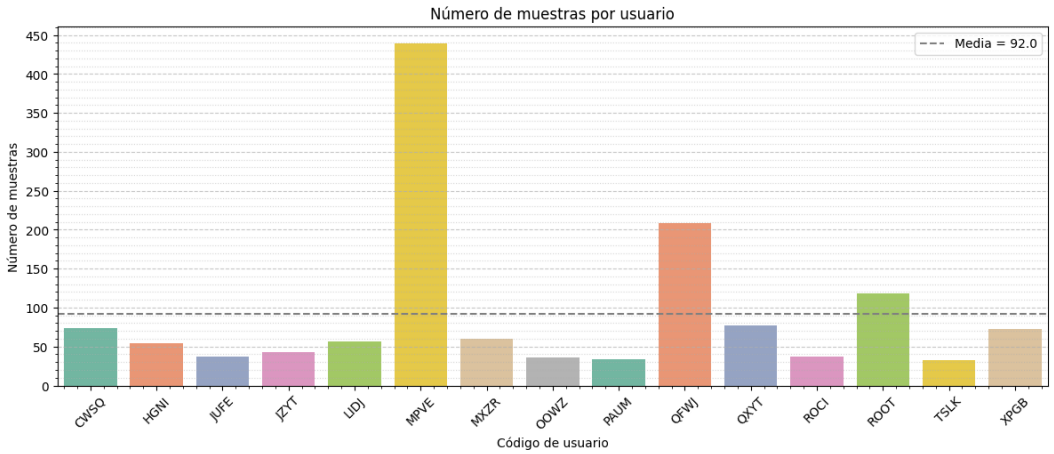


Figura 3.12: Distribución de muestras por sujeto.

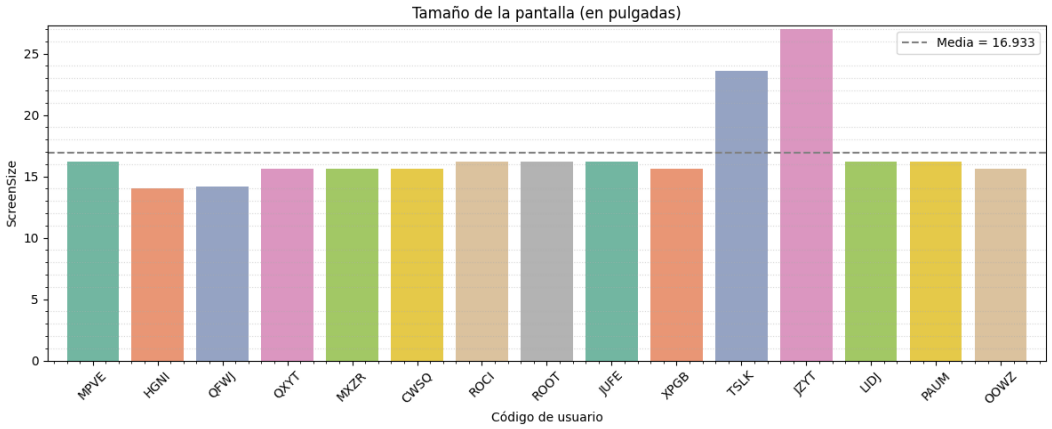


Figura 3.13: Tamaños de diagonal de pantalla en pulgadas por sujeto.

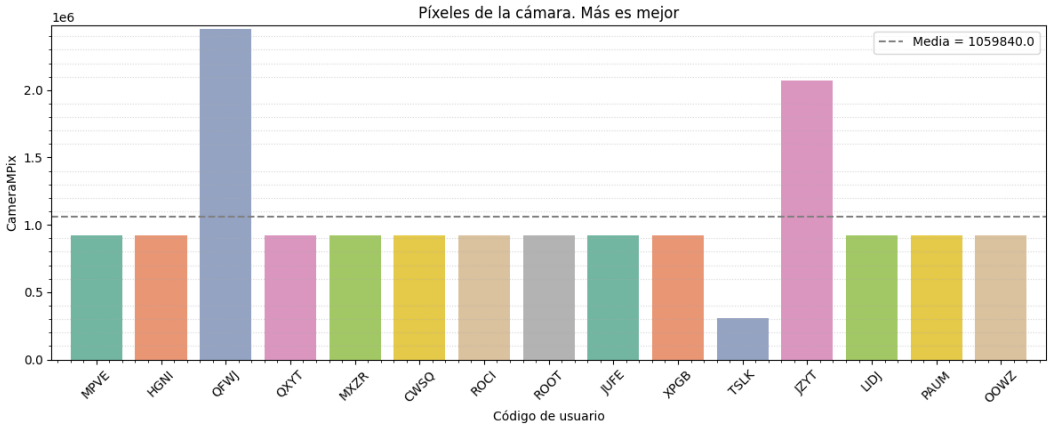


Figura 3.14: Píxeles de dispositivo de captura por sujeto.

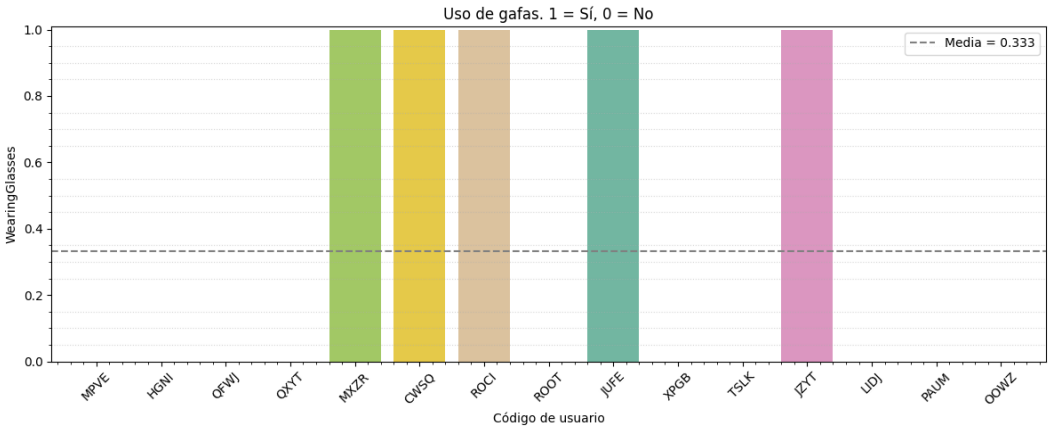


Figura 3.15: Uso de gafas por sujeto.

Es importante notar en este punto que el conjunto de datos es sensiblemente más pequeño de lo habitual, pues conjuntos bien conocidos en este ámbito, como MNIST, tienen un tamaño del orden de entre 10 y 50 veces más muestras. Por tanto, esto va a ser un factor limitante a tener en cuenta durante la mayoría de pasos que involucran a los modelos.

Capítulo 4

Propuesta del sistema

4.1 Descripción funcional

Teniendo presente el contexto actual de las librerías para visión artificial y los marcos de trabajo para aprendizaje automático, se ha concebido una arquitectura que encadena varias soluciones, aprovechando las virtudes sin olvidar las contrapartes de cada una de ellas. El único actor, presente en todos los casos de uso, dado que el propósito del sistema es, recibiendo una imagen del usuario, devolver coordenadas del lugar geométrico de la pantalla al que se dirige su mirada. Se enumeran y justifican los elementos del sistema a continuación:

- Cámara: La función de este elemento es emitir las imágenes que se introducen en el sistema. Aunque está pensado para funcionar con cámaras web, si bien, procesaría imágenes provenientes de otras fuentes de la misma forma.
- Detector facial: Es un elemento desarrollado por terceros cuyo cometido es, a partir de una imagen en la que aparece una cara, inferir las coordenadas de la región cuadrada más pequeña que contenga a la cara. Disponer de ellas permite recortar dicha región, prescindiendo del resto, que contiene información irrelevante para el cometido del sistema. De ahora en adelante, se hará referencia a estas imágenes recortadas como recortes faciales.
- Detector ocular: Al igual que el detector facial, es un elemento desarrollado por terceros cuyo cometido es, a partir de la mitad superior de una imagen cuadrada contenedora de una cara resultante de recortar la imagen proveniente de la cámara según las coordenadas inferidas por el detector facial, inferir las coordenadas de dos regiones cuadradas que contengan los ojos, permitiendo recortar una porción de la imagen igual a dichas regiones o contenedora de las mismas, prescindiendo del resto, que contiene información menos relevante para el cometido del sistema. Dado que la aplicación de este detector es menos común y resulta más complicado identificar

ojos inequívocamente mediante patrones que caras, este detector es también menos efectivo y preciso que el facial. De ahora en adelante, se hará referencia a estas imágenes recortadas como recortes oculares.

- Regresor de *Eye-Tracking* con entrada ocular: Es una red neuronal convolucional entrenada con recortes oculares etiquetados con las coordenadas normalizadas del lugar geométrico de la pantalla al que se dirige la mirada del usuario. Es decir, el modelo recibe una imagen que contiene ambos ojos (y algo de su contorno) e infiere coordenadas bidimensionales. Es importante notar que esta es una descripción funcional que por simplicidad omite detalles sobre el modelo final que se desarrollará en las secciones 5.3 y 5.4.

- Regresor de *Eye-Tracking* con entrada facial: Por los motivos sobre la reducida efectividad del detector ocular que se han explicado en su descripción, resulta interesante buscar alternativas que no requieran del mismo en cada iteración de la secuencia del sistema. La más obvia es esta misma, una red neuronal similar a la anterior que trabaje con recortes faciales en vez de oculares, lo que hace que no sea necesario detectar los ojos y además, conlleva ciertas ventajas; pues al eliminar un paso de detección complejo, el sistema final resultará, no solo más simple y eficiente, sino más tolerante a condiciones de iluminación, resoluciones o calidades de imagen más pobres. Además, al conservar la cara completa, el modelo podría aprender inherentemente a considerar los movimientos involuntarios de la cabeza, que en ocasiones resultan inevitables. Todo esto a costa de sacrificar algo de precisión y exactitud, además de reducir la efectividad del entrenamiento, pues los recortes faciales contienen inevitablemente más información que los oculares, de la cuál la mayor parte es muy poco relevante para el cometido del modelo, pudiendo considerarse ruido, lo que puede llevar a que aprenda relaciones espurias y no generalice correctamente (riesgo de sobreentrenamiento) [36].

- Clasificador binario de mirada a pantalla con entrada facial: Es una red neuronal entrenada con recortes faciales obtenidos a partir de registros del Experimento 3 de la aplicación de adquisición, cuyas etiquetas han sido modificadas para ser binarias, en función de si el usuario de la imagen mira o no a la pantalla. Su razón de ser es que los modelos regresores no son capaces de distinguir si la entrada que reciben es correcta (es decir, no tienen forma de saber si la imagen de entrada se asemeja a aquello con lo que han sido entrenados, que en esta ocasión son los ojos o la cara de un usuario mirando a algún lugar de la pantalla), e infieren coordenadas en cualquier caso.

4.2 Diagrama de secuencia y casos de uso

Se muestran a continuación en la figura 4.1 y en las tablas 4.1 4.3 y 4.2.

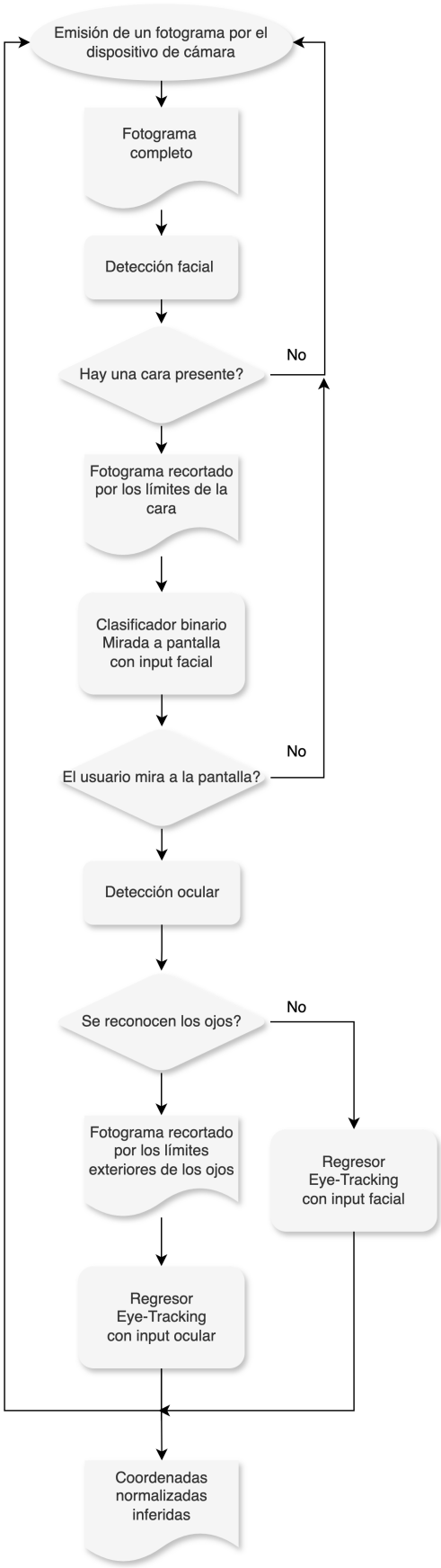


Figura 4.1: Diagrama de secuencia propuesto para el sistema

Nombre	DetecciónBinariaMirada
Elementos del sistema involucrados	Cámara Detector facial Clasificador binario mirada a pantalla con input facial
Precondiciones	Ninguna.
Secuencia de éxito	1. Se recibe un fotograma 2. Se detecta una cara en el fotograma 3. Se recorta la imagen con la cara 4. Se analiza con el clasificador 5. Se infiere la clase afirmativa
Postcondiciones	Se posibilita la ejecución de métodos siguientes de la secuencia.
Secuencia alternativa 1	1. Se recibe un fotograma 2. Se detecta una cara en el fotograma 3. Se recorta la porción cuadrada de la imagen más pequeña posible que contenga la cara 4. Se somete el recorte facial a un modelo clasificador entrenado para distinguir imágenes en las que el usuario mira a la pantalla de su equipo 5. El modelo infiere la clase negativa
Postcondiciones S.A. 1	El sistema no realiza más operaciones con el fotograma y espera a recibir el siguiente.
Secuencia alternativa 2	1. Se recibe un fotograma 2. No se detecta ninguna cara en el fotograma
Postcondiciones S.A. 2	El sistema no realiza más operaciones con el fotograma y espera a recibir el siguiente.

Tabla 4.1: Caso de uso: `DetecciónBinariaMirada`.

Nombre	SeguimientoOcularAltaPrecisión
Elementos del sistema involucrados	Detector ocular Detector facial Clasificador binario mirada a pantalla con input facial Regresor de *Eye-Tracking* con input facial
Precondiciones	Se ha detectado la cara del usuario mirando a la pantalla (ejecución correcta del caso de uso anterior)
Secuencia de éxito	1. Se recibe una imagen de la región de interés que contiene los ojos del usuario. 2. Se somete la misma al modelo regresor de *Eye-Tracking* con input facial. 3. El modelo infiere coordenadas.
Postcondiciones	La secuencia finaliza y el sistema vuelve al estado inicial.

Tabla 4.2: Caso de uso: `SeguimientoOcularAltaPrecisión`.

Nombre	SeguimientoOcularBajaPrecisión
Elementos del sistema involucrados	Cámara Detector facial Clasificador binario mirada a pantalla con input facial Regresor de *Eye-Tracking* con input facial
Precondiciones	Se ha detectado la cara del usuario mirando a la pantalla (ejecución correcta del caso de uso DetecciónBinariaMirada) pero en la región superior de esta no se han detectado los ojos (ejecución alternativa 1, incorrecta, del caso de uso SeguimientoOcularAltaPrecisión)
Secuencia de éxito	1. Se recibe una imagen cuadrada que contiene la cara del usuario. 2. Se somete el recorte facial al modelo regresor de *Eye-Tracking* con input facial. 3. El modelo infiere coordenadas.
Postcondiciones	La secuencia finaliza y el sistema espera recibir un fotograma para comenzarla de vuelta.

Tabla 4.3: Caso de uso: `SeguimientoOcularBajaPrecisión`.

Capítulo 5

Entrenamiento y evaluación de modelos

En el anterior Capítulo se explicó el funcionamiento del sistema modelándolo como se suele hacer en la ingeniería de sistemas de software, definiendo los componentes y representando su interacción en un diagrama de secuencia acompañado de sus casos de uso. Estos elementos forman parte del lenguaje de modelado UML, útil para diseñar y documentar en este ámbito.

Resta, por tanto, en base a esta arquitectura funcional, implementar el sistema tal y como se ha concebido, construyendo una tubería o *pipeline* –términos con los que se hace referencia al tratamiento de datos en cascada, donde la salida de cada punto es la entrada del siguiente– encargada de orquestar cada uno de los pasos definidos previamente, de forma que al introducir una imagen del usuario en el sistema, su salida sea las coordenadas del píxel de la pantalla al que dirige su mirada .

En primer lugar se explorarán las posibilidades y se justificarán las decisiones en cuanto a lenguajes de programación, entornos de desarrollo, abstracciones, librerías y *frameworks* así como otras herramientas existentes desarrolladas por terceros.

A continuación se definirán e implementarán los primeros pasos de la tubería. Concretamente aquellos dedicados al preprocesado de los datos, desde la carga en memoria desde el almacenamiento del conjunto hasta que son compatibles con las entradas del modelo.

Posteriormente, dado el reducido tamaño del conjunto de datos del proyecto, se desarrollarán los modelos en dos fases:

- Selección: Se analiza el conjunto de datos para definir los hiperparámetros esenciales y a grandes rasgos, de los modelos que se van a entrenar con el mismo. A continuación se realiza dicho entrenamiento y se analiza el comportamiento ante el conjunto de validación para hacer ajustes finos y terminar de definirlos.

- Evaluación: Se propone un mecanismo consistente en una técnica de validación cruzada que promedia los resultados de evaluar los modelos con cada uno de los sujetos del conjunto, habiendo sido entrenados con el resto, ofreciendo un indicador robusto de la exactitud del modelo al hacer predicciones con sujetos con los que no ha sido entrenado. Paralelamente, se ha mantenido un mecanismo de evaluación convencional en el que el subconjunto de evaluación se forma al separar aleatoriamente entre un 10 % y un 20 % de las muestras del conjunto y por tanto, un mismo sujeto puede aparecer en sendos conjuntos de evaluación y entrenamiento; ya que la mayoría de sistemas de *Eye-Tracking* del mercado, durante su inicialización adquieren datos del usuario para integrarlos en sus cálculos, calibrarse y mejorar su rendimiento [28, 29, 30, 31].

5.1 TensorFlow con Keras

En la actualidad, se puede usar casi cualquiera de los lenguajes de programación más comunes para desarrollar sistemas de *Deep Learning*, aunque, dada la popularidad de la disciplina, lo más habitual es trabajar con librerías y *frameworks* que permiten a los desarrolladores abstraerse de la mayor parte de las tareas elementales de naturaleza matemática asociadas al aprendizaje profundo, y enfocarse en los procesos directamente relacionados con la definición, el entrenamiento, la validación y la evaluación de modelos. Estos acotan en cierto modo el abanico de lenguajes a elegir, ya que suelen estar implementados para solo unos pocos. Estos suelen ser:

- Java: Es un lenguaje de programación y una plataforma, debido a las utilidades, kits y a la máquina virtual (JVM) que lo acompañan, que lo hacen fácilmente compatible con diferentes sistemas operativos y arquitecturas de microprocesador, lo que a su vez lo hace ideal para sistemas distribuidos. Como lenguaje, es compilado, orientado a objetos y fuertemente tipado, y exclusivamente de alto nivel. El principal atractivo de desarrollar *Deep Learning* en Java es la compatibilidad con sistemas existentes basados en Java pero que aún no lo implementan, lo que, en el contexto actual del avance imparable de la inteligencia artificial, resulta una opción muy atractiva.

- C++: Es el sucesor por excelencia de C, planteado como una extensión de este para añadir mecanismos que permitieran la manipulación de objetos, lo que lo hace multiparadigma. Como su antecesor, es fuertemente tipado, compilado, y muy rápido en comparación con Java, entre otras cosas, gracias a sus utilidades de bajo nivel, con la contraparte de ser técnicamente más complejo para el desarrollador. Se suele usar en entornos de producción donde el rendimiento o el uso eficiente de recursos sea prioritario.

- Python: Es un lenguaje de programación interpretado, multiparadigma, de alto nivel y no tipado, a diferencia de los anteriores. Su filosofía hace hincapié en la

legibilidad de su código, lo que explica peculiaridades como basar la segmentación de bloques de código en la indentación, que en otros lenguajes suele tener fines meramente estéticos. El código en Python, por esto, suele resultar más visual; y al ser interpretado, permite desarrollar scripts de manera simple, y escribir código en tiempo de ejecución. Por todo esto resulta una opción muy atractiva para el ámbito académico, la investigación y el desarrollo de pruebas de concepto. No por menos cuenta con la gran mayoría de librerías y *frameworks* que más se utilizan en el campo del *Deep Learning*. Su desventaja es su rendimiento inevitablemente más bajo que el de otros lenguajes compilados [9].

En cuanto a librerías y/o *frameworks* para *Deep Learning*, existen una serie de posibilidades bien conocidas, documentadas y respaldadas por grandes comunidades entre las que participan referentes como Google, Tesla, Meta, Netflix o Spotify:

- Scikit-Learn: Librería diseñada para manejar álgebra lineal y análisis estadístico con un alto rendimiento. Es muy robusto, pudiendo usarse en entornos de producción. introducido en 2007 en un proyecto de Google. Ofrece herramientas eficientes para regresión, clasificación, clusterización, selección de modelo, preprocesado y reducción de dimensionalidad. Está disponible para Python únicamente.

- Keras: Es una librería altamente productiva orientada a soluciones. Es de alto nivel y está pensada para ser utilizada como un wrapper de otros *frameworks* de más bajo nivel, proporcionando un nivel de abstracción a mayores. Aunque es multiplataforma y fácilmente escalable, se enfoca en la velocidad de depuración, legibilidad del código, mantenibilidad y facilidad de iteración [12].

- Deeplearning4j: *framework* para Java y Scala, desarrollado en Java como contribución del proyecto Eclipse en 2017. Está pensado para su integración en sistemas existentes, dada su compatibilidad con la máquina virtual de Java (JVM) y sistemas distribuidos, lo que lo hace óptimo para entrenar modelos de DeepLearning de gran escala [37].

- PyTorch: Uno de los *frameworks* más conocidos para *Deep Learning*, introducido en 2016 de parte de un grupo de integrantes del laboratorio de inteligencia artificial de Facebook con los fines de prescindir de la librería NumPy en favor de los tensores, un tipo de objeto que se asemeja a las matrices pero que puede usarse en GPUs; y para ofrecer una librería de diferenciación automática para el cálculo de gradientes en cualquier gráfico computacional. Sus APIs están disponibles en Python y C++, que son dos de los lenguajes para los que fue desarrollado junto con C y CUDA (CUDA C). Es utilizado por gigantes de la industria como Tesla, Google y Salesforce [37].

- TensorFlow: Junto con PyTorch, es el *framework* más popular, utilizado principalmente para cómputo numérico en *Deep Learning*. Concebido por Google para uso interno, como sucesor de su sistema de aprendizaje automático propietario,

DistBelief; y liberado al público general bajo licencia de código abierto en 2015. Simplifica el flujo de trabajo en el desarrollo de sistemas de deep learning mediante la definición de estructuras de datos con entradas y salidas, el uso de tensores y cuenta con librerías desarrolladas en CUDA para aceleración por hardware mediante GPUs NVIDIA, además de ser compatible con TPUs. Desarrollado en Python y C++, proporciona APIs para Python, C y C++, Java, Go y Rust. Además, existe una librería oficial para Javascript (TensorFlow.js) pensada para implementaciones. Es decir, ejecutar en aplicaciones web modelos existentes y desarrollados en Python, aunque también permite reentrenar modelos existentes y hacer *Transfer Learning* para personalizarlos; e incluso crearlos desde cero y entrenarlos, por lo que prácticamente se puede considerar una API disjunta de TensorFlow [11].

Para este proyecto –que es una prueba de concepto– se precisa de un entorno de trabajo con las siguientes características:

- Debe ser eficiente para el desarrollador y orientado a productividad. Es decir, debe permitir desarrollar e implementar soluciones de forma rápida y sencilla, a fin de facilitar la investigación de mecanismos y fórmulas sin tener que profundizar en los aspectos más específicos del proceso o del entorno, por lo que se valoran positivamente las opciones con alto nivel de abstracción, por encima del alto rendimiento.
- Debe ser escalable, pues una prueba de concepto concluye cuando contrasta la viabilidad de la propuesta, cosa que en este proyecto, como en la gran mayoría, ocurrirá mucho antes de alcanzar la primera versión de producción del sistema. Para garantizar dicha evolución a un entorno de producción, aquellas propuestas que cuenten con ecosistemas pensados para la escalabilidad y la implementación, presentan un valor añadido.

Teniendo en cuenta todo ello, **se ha elegido Python como lenguaje de programación** por los siguientes motivos [9]:

- Al ser interpretado no se requiere de un entorno de desarrollo integrado (*IDE*) ni de un profundo conocimiento de los compiladores, cosa que en este proyecto no forma parte de los objetivos y en el peor de los casos podría suponer una dificultad añadida para su consecución.
- Presenta un alto nivel de abstracción y su tipado es dinámico, por lo que resulta muy sencillo de utilizar.
- Debido a su enfoque en la legibilidad del código, se presenta como una opción interesante de cara a ser mostrado en un contexto académico.
- La gran mayoría de librerías y *frameworks* populares en el ámbito del aprendizaje automático han sido desarrollados total o parcialmente en Python y las APIs más con mayor respaldo por parte de los mantenedores de los *frameworks* también son las implementaciones para este lenguaje.

Por otro lado, **el *framework* elegido es TensorFlow con Keras**, por las siguientes razones [11, 12]:

- Popularidad: Cuenta con una amplia comunidad de desarrolladores, documentación detallada y soporte continuo de su desarrollador, que es Google, lo que facilita la resolución de problemas y el acceso a las últimas innovaciones en *Deep Learning*.
- Compatibilidad muy amplia: dado su enfoque multiplataforma, es posible su entrenamiento y sobre todo su ejecución en casi cualquier entorno, como la web o los dispositivos móviles, para los que cuenta con las librerías TensorFlow.js y TensorFlow Lite.
- Elevado nivel de abstracción: Gracias a Keras, integrado en el TensorFlow y entregado junto al mismo en los canales de descarga oficiales, es superior al que ofrece el *framework* de base. Además proporciona nuevas interfaces para definir modelos mediante el partón funcional y simplificar el proceso de entrenamiento al incluir optimizadores, funciones de pérdidas y métricas directamente en la propia librería. También lleva el enfoque multiplataforma de TensorFlow a otro nivel, ya que con Keras, es posible transferir estos modelos a otros *frameworks* de *Deep Learning* si es preciso.

5.2 OpenCV

En el arquitectura propuesta en el Capítulo anterior se definían los detectores ocular y facial. Dos elementos funcionales que, al implementar funciones muy conocidas y no ser un fin sino un medio para conseguir los objetivos de este Trabajo, es preferible implementarlo desde otras fuentes de código abierto antes que hacer un desarrollo propio; dado que la investigación, los recursos y el conocimiento detrás de ciertos proyectos resultan difícilmente igualables en el alcance de un Trabajo de Fin de Grado.

La función del detector facial se implementa con dos elementos:

- Un modelo de *Deep Learning* entrenado para reconocer los patrones que definen un rostro humano. La entrada de este es una imagen. La salida es una matriz de números que describen los lugares geométricos donde encuentra cada uno de los rostros presentes dentro de la imagen (si los hay y el modelo es capaz de detectarlos). El formato de esta salida depende del modelo en cuestión pero suele tener cuatro valores que determinan las coordenadas de un área rectangular o cuadrada que contiene el rostro. Lo normal es que los dos primeros correspondan a las coordenadas en píxeles de la esquina superior izquierda, y los dos últimos, al ancho y alto de dicho área.
- Un método que implemente las operaciones lógicas para tratar los datos de los clasificadores y las imágenes en consecuencia. Su función en este caso es más compleja

y crucial que en el detector facial, ya que si es preciso, debe aplicar ciertas transformaciones a los valores numéricos inferidos por el detector, para que asegurar una coherencia en el formato de la salida en diferentes iteraciones, que los pocos modelos disponibles no suelen garantizar. También debe rechazar falsos positivos que manifiestamente lo sean, lo cual suele ser una tarea sencilla, pues conociendo el formato de las imágenes faciales que esta parte del sistema maneja, resultan obvias ciertas observaciones, como que por ejemplo, los ojos siempre se encuentran en la mitad superior de la imagen y cualquier posible detección fuera de dicha zona no es válida. O también, que el tamaño habitual del ojo en píxeles en el conjunto de datos del proyecto es de 60 con un tercer cuartil de 66 y un máximo atípico de 110 (la mayor parte de los valores se distribuyen entre 50 y 60), por lo que cualquier detección de tamaño muy diferente a estos, probablemente no será correcta.

Hay una función que puede implementarse junto con los detectores como parte del tratamiento que realizan, que es la de un reescalador, que sirva para aumentar o disminuir la resolución, en función de las necesidades de cada modelo [32]. Esto se debe a que la forma de las imágenes no es fija. En el caso de las imágenes de entrada depende de la resolución de la cámara. En el caso de los recortes faciales, de las facciones del usuario, de su posición respecto de la cámara (si se aleja decrementa y viceversa); y en el caso de los recortes oculares, de nuevo de las facciones, pero también de la forma en la que se ejecuten (si se utilizan dos áreas cuadradas, una sola rectangular que englobe a ambos, si se aplica un relleno o margen, etc.).

A la hora de elegir una sistema de código abierto para implementar el núcleo del detector facial, se utilizaron como criterio las siguientes necesidades:

- Debe ser implementable en el entorno de trabajo del proyecto. Es decir, ser compatible con versiones actuales de Python.
- Su rendimiento debe permitir, como mínimo, realizar detecciones a tiempo real a una resolución suficiente para las redes neuronales que se desarrollarán más adelante, a un mínimo de 30 FPS, necesarios para proporcionar una experiencia de uso aceptable en futuras implementaciones interactivas.
- La compatibilidad con CPU se valora muy positivamente. No es un requisito imprescindible, pero sí a tener en cuenta, dado que la mayoría de dispositivos en el mercado al momento de la redacción de esta memoria no tiene una GPU dedicada suficientemente potente para acelerar este tipo de modelos.
- En caso de no encontrar un modelo que satisfaga todos los requisitos anteriores, se plantea la necesidad de que exista una solución disponible en otros entornos, por ejemplo basados en C++, donde el rendimiento suele ser bastante superior, de cara a una posible futura puesta en producción, pudiendo usar otra solución que se le aproxime en la prueba de concepto actual.

En el ámbito de la detección facial existen numerosas soluciones plenamente funcionales y bien probadas, con unos niveles de desarrollo que exceden el alcance de este proyecto en cuanto a tiempo y recursos. Por tanto, no tiene sentido tratar de desarrollar una solución propia para este fin. Las posibilidades que se han barajado entonces, son las que se muestran en la tabla 5.1.

En cuanto a la detección ocular, el proceso no es tan común ni está tan bien documentado, por lo que no hay tantas soluciones ya desarrolladas. El método más habitual consiste en una secuencia compuesta por un detector facial como el definido anteriormente, seguido de un modelo que, dentro del área contenedora del rostro inferida por el detector facial, detecta e infiere la ubicación geométrica de los ojos. Esto se suele referenciar con el término de "tubería de detección ocular". Para implementarlo se suelen utilizar clasificadores Haar Cascade con OpenCV, la librería de visión artificial más grande y conocida hasta la fecha, que incluye tanto una implementación de Haar Cascade como varios ficheros XML con parámetros resultantes de diferentes entrenamientos, para poder utilizarlo en detección facial y ocular, entre otros posibles escenarios. Además, cuenta con una muy amplia variedad de métodos para implementar casi cualquier tratamiento de imagen que pueda ser preciso en un sistema como este y como el del proyecto, como reescalados, interpolaciones, conversiones de espacios de color, lectura y volcado de ficheros, etc. [45]

Como se ha mencionado anteriormente, los modelos para detección ocular son menos eficaces que aquellos para detección facial, tanto por la complejidad de la operación en el contexto de este proyecto al manejar un número reducido de píxeles para los ojos, como por la falta de madurez de las soluciones existentes al no ser una práctica tan habitual como la detección facial. Por tanto, en este caso sí puede merecer la pena recurrir a otros mecanismos que no dependan exclusivamente de un modelo preentrenado. En cualquier caso, el sistema completo considera el uso de una modelo neuronal regresor para *Eye-Tracking* que no requiera de detección ocular, al utilizar únicamente recortes faciales para este fin [45].

Para el proyecto, dado que la librería OpenCV proporciona todas las funciones que se podrían necesitar para tratar las imágenes, y contiene interfaces con las que implementar diferentes modelos de forma sencilla, se ha optado por utilizarla, en su versión 4.11.0.

OpenCV fue desarrollada por Intel en el año 2000, en C++ y bajo licencia de código abierto, con el objetivo de ser multiplataforma, altamente eficiente y multihilo. En procesadores Intel puede acceder a las primitivas IPP (*Integrated Performance Primitives*), propietarias y de bajo nivel, para aumentar incluso más su rendimiento. Estas son sus fortalezas [14]:

- Madurez y confiabilidad: OpenCV (*Open Source Computer Vision Library*) es una de las bibliotecas más consolidadas en el ámbito de la visión por computadora. Su trayectoria y constante evolución la han convertido en una opción estable y confiable, ideal para proyectos tanto académicos como industriales.

- Velocidad y eficiencia: Gracias a su implementación en C++ (con interfaces para Python y otros lenguajes), OpenCV ofrece un rendimiento excepcional en tiempo real, lo cual es crucial para tareas como la detección facial u ocular en aplicaciones interactivas o de vigilancia. Además, C++ es uno de los lenguajes a considerar en caso de seguir la línea futura de llevar este proyecto, que es una prueba de concepto, a una versión de producción, al ser compilado, mucho más rápido y eficiente. El hecho de estar desarrollada nativamente en este lenguaje facilitaría en cierto modo una migración.

- Modelos preentrenados: La biblioteca incluye clasificadores Haar Cascade y LBP listos para usar, lo que permite implementar soluciones funcionales con apenas unas líneas de código, y permite descargar y utilizar otros modelos si los incluidos no fueran suficiente. Esto reduce la necesidad de recursos computacionales y datos de entrenamiento, especialmente útil en prototipos o dispositivos con capacidad limitada.

- Versatilidad y compatibilidad: OpenCV se adapta tanto a sistemas embebidos como a plataformas más robustas. Puede integrarse con otros *frameworks* como TensorFlow o PyTorch si se necesita escalar a modelos más sofisticados, haciendo que funcione bien como una base ligera y ágil o como complemento de soluciones más complejas, que es el caso del proyecto. A nivel de lenguajes, es compatible con C++, Python, Java, Matlab/Octave y JavaScript, estando este último enfocado a implementaciones, mientras que el resto también son adecuados para desarrollo.

- Comunidad y documentación: Una de las mayores fortalezas de OpenCV es su comunidad activa y extensa documentación, lo que facilita la resolución de problemas.

Finalmente, para implementar los detectores facial y ocular con OpenCV; se han probado dos modelos diferentes capaces de trabajar en tiempo real en CPU:

- **Haar Cascade**: Es un detector en cascada que utiliza extracción de características de Haar, propuesto en 2001. Estas funcionan de forma similar en cierto modo a un kernel convolucional: La suma de los píxeles bajo las zonas blancas se sustrae de la suma de los píxeles bajo las zonas negras. Dichas características, se evalúan a lo largo de diferentes zonas de la imagen, mediante una técnica de ventana deslizante. Las zonas con alto contraste (Por ejemplo, los ojos son más oscuros que el puente nasal o las mejillas) proporcionarán valores cuyas correlaciones con la presencia de un rostro sea más alta (Por ello las características que se muestran en la imagen 5.2 se pueden considerar adecuadas), mientras que otras sin contraste resultarán irrelevantes (como por ejemplo, la segunda característica (a) aplicada a las dos mejillas). Cabe destacar que con motivo de que las características de Haar se utilicen para detectar contrastes, el sistema se apoya en la luminancia para realizar detecciones. Utilizar imágenes a color en estas circunstancias no aportaría

nada y supondría un desperdicio de recursos. Por ello, este modelo solo trabaja con matrices bidimensionales de imágenes en escala de grises [45].

Durante el entrenamiento para la versión para detección facial, se utiliza un set de imágenes etiquetadas de forma binaria en función de si contienen o no un rostro; y se evalúan todas las combinaciones de características y lugares posibles para determinar cuáles presentan una buena correlación con la etiqueta mediante un umbral. Esto tiene un coste computacional elevado, por lo que determinar cuáles son las más relevantes es crucial para que el modelo, tras su entrenamiento, no desperdicie recursos en comprobar el resto [45].

El concepto de clasificador en cascada no es fortuito. El modelo se compone de una serie de clasificadores (en "cascada") que comprueban uno tras otro, en cada zona de la imagen, si una determinada característica supera un cierto umbral. El valor de este mecanismo reside en la acción conjunta de todos los clasificadores de características, que es, efectivamente, la detección facial en este caso [45].

Todo esto lo convierte en un sistema muy flexible, pues en esencia, evalúa características de forma binaria. Mediante un entrenamiento es posible seleccionar qué características resultan relevantes para el problema en cuestión. Si bien la explicación anterior quedaba ligeramente particularizada para el caso de detección facial, es posible detectar cualquier elemento con el entrenamiento adecuado. Ese es el caso de los ojos, para implementar la detección ocular. Por suerte, tampoco será necesario un entrenamiento porque OpenCV proporciona una versión del modelo ya entrenada para este fin.

La salida del clasificador es una matriz en la que cada fila contiene los valores que definen la región de interés, o caja delimitadora que contiene el elemento detectado, en formato $[X, Y, W, H]$, donde X e Y representan las coordenadas de la esquina superior izquierda de la caja; y W y H, el ancho y el alto respectivamente (aunque son siempre cuadradas y por tanto iguales); todas ellas en píxeles [45].

- **YuNet**: Es una red neuronal convolucional ligera, pensada para ser ejecutada en CPU, mucho más actual que Haar Cascade, tanto por la tecnología que utiliza, tomando ventaja de los avances del *Deep Learning*; como por fecha del desarrollo, ya que la versión del modelo utilizado data de 2023. De hecho, su propósito es mejorar y actualizar la propuesta de detectores como Haar Cascade. YuNet se definió y entrenó en un *framework* de *Deep Learning* en Python, con una arquitectura en capas ilustrada en la figura 5.3 que no resulta muy diferente a las que se usarán en este proyecto [42]. No obstante, la principal diferencia entre ambos es que YuNet es un regresor, mientras que Haar Cascade es un clasificador, a pesar de que ambos pueden devolver las coordenadas de regiones de interés que contienen rostros. El motivo es que el primero realmente calcula dichas coordenadas a partir de la imagen de entrada, mientras que el segundo evalúa ventanas deslizantes de características, devolviendo las coordenadas de aquellas cuya inferencia sea positiva, logrando ambos un efecto similar a grandes rasgos, aunque en el caso de YuNet, el proceso sea más intuitivo y eficiente [42].

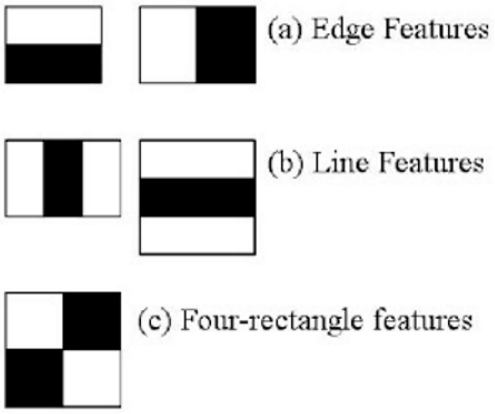


Figura 5.1: Leyenda de características de Haar. Tomado de [45].

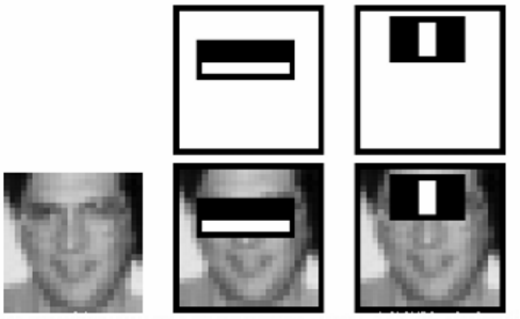


Figura 5.2: Ilustración de evaluación de características de Haar. Tomado de [45].

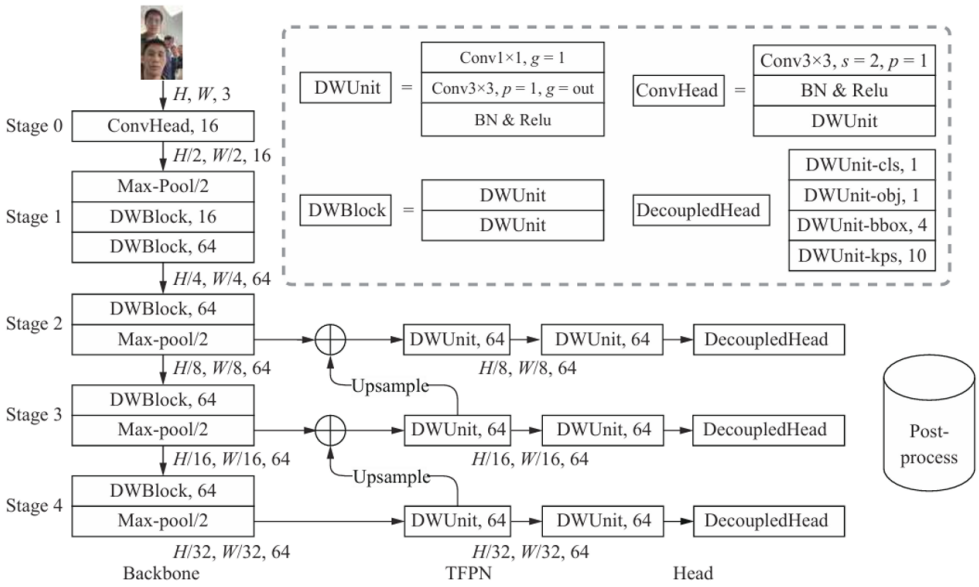


Figura 5.3: Diagrama del modelo de la red neuronal YuNet. Tomado de [42].

Para acotar las funciones que desempeña cada parte del diagrama que caracteriza la arquitectura del modelo (Ver figura 5.3) los desarrolladores las han delimitado bajo las siguientes denominaciones [42]:

- **Backbone**: Es el núcleo de la red neuronal, encargado de extraer las características básicas para la detección facial. Para hacer más eficiente y ligera la red, se utilizan las técnicas de Depthwise Convolution y Pointwise Convolution, que consisten en aplicar un filtro de convolución por separado a cada canal de la imagen (lo que significa que YuNet sí trabaja con imágenes a color, a diferencia de Haar Cascade) en combinación con filtros de convolución de 1x1, que afectan a cada píxel por separado sin tomar valores de los píxeles aledaños. Este mecanismo fue implementado originalmente en la MobileNet, pensada para dispositivos móviles
- *TFPN*: Acrónimo de *Tiny Feature Pyramid Network*, es una forma de llamar a una arquitectura de capas jerárquica que recibe los mapas de características del *backbone*, que vienen en tres niveles diferentes, para fusionarlos y generar características de mayor resolución. Si bien esta zona se denomina *TFPN*, también matizan que esta pirámide tiene dos partes, y que esta se consideraría el "cuello".
- *Head*: Corresponde a la parte externa de la *TFPN*, o a las capas de salida de la red neuronal. Cada una de ellas se compone de 16 neuronas, siendo las más relevantes para el proyecto actual la de clasificación (que es binaria e indica si la red es capaz de detectar al menos un rostro en la imagen de entrada) y las 4 de región de interés, que al igual que en el Haar Cascade, se representa por cuatro valores correspondientes a las coordenadas de la esquina superior izquierda y el ancho y alto de la caja, aunque en YuNet no son necesariamente iguales. Cabe matizar el hecho de que existan varias capas de salida. El motivo es que en este punto se mantienen también los tres niveles de la pirámide y se asocia una capa de salida a uno de ellos. Para conformar la salida final, definen un bloque de posprocesado tras la red neuronal que toma los valores de los tres niveles y suprime aquellos que no sean máximos

5.3 Preprocesado de los datos

En esta sección se describe la secuencia de operaciones a las que se someten los datos en los scripts de definición y entrenamiento, para pasar del formato en el que se presentan en el conjunto de datos del proyecto hasta los formatos que manejan los diferentes modelos de *Deep Learning* desarrollados, que en el caso de las entradas son los siguientes:

- Imagen: Matriz bidimensional cuya dimensión, constante, se define en el proceso; de valores normalizados (originalmente enteros de 8 bits sin signo). Sus valores representan la luminancia de los píxeles que componen una imagen en escala de

grises o de los tres canales RGB que componen una imagen a color, que puede contener la cara o ambos ojos dependiendo del modelo.

- Coordenadas faciales: Matriz unidimensional que contiene las coordenadas de la caja delimitadora que contiene la cara en la imagen original (coordenadas de las esquinas superior izquierda e inferior derecha, $[X1, Y1, X2, Y2]$)
- Diagonales: Variable que contiene el valor decimal de la longitud de la diagonal en pulgadas de la pantalla del usuario.
- Uso de gafas: Variable que contiene el valor booleano que indica si el usuario lleva gafas o no.

La salida, en el caso de los modelos regresores de *Eye-Tracking*, es una matriz unidimensional de dos elementos, que corresponden a las coordenadas X e Y normalizadas, que representan el lugar geométrico de la pantalla al que se dirige la mirada. En el modelo clasificador para detectar si la mirada se dirige a la pantalla, la salida es un solo elemento booleano, que toma valor 1 en caso de que sí lo haga.

En total se han desarrollado tres modelos: El primero recibe como entrada recortes faciales, y los dos restantes, recortes oculares. Para el modelo de recorte ocular se han desarrollado dos idénticos con una salvedad en el procesado de los datos, de forma que se pueda evaluar su efectividad comparando el rendimiento de ambas implementaciones: Una realiza detección ocular mediante el Haar Cascade apropiado en tiempo real (es decir, cada vez que recibe una imagen) como cabría esperar. La otra utiliza el mismo modelo pero solo durante el entrenamiento, con el mismo set que los modelos regresores, para ajustar los parámetros de un método heurístico desarrollado para el proyecto que prácticamente elimina las pérdidas a costa de aumentar mínimamente los falsos positivos y disminuir ligeramente la exactitud.

5.3.1 Modelo I: Detección facial mediante Haar Cascade

La secuencia de pasos hasta la detección facial es común a todos los modelos e implementaciones del proyecto basados en Haar Cascade:

1. Carga del conjunto de datos I: Se carga la tabla del fichero CSV del conjunto de datos en memoria, en formato de `dataframe` de pandas.

2. Carga del conjunto de datos II: Leyendo las rutas relativas del `dataframe` anterior, se cargan las imágenes del conjunto de datos en memoria mediante OpenCV, en matrices tridimensionales de enteros de 8 bits sin signo, lo que significa que cada elemento representa un píxel. Su valor, la proporción de los colores asociados a cada canal en cuestión. En el caso de OpenCV, el formato en el que se presentan en memoria las imágenes a color es BGR (Blue-Green-Red). No tiene demasiada importancia, ya que solo se diferencia en una permutación del mucho más común RGB (*Red-Green-Blue*) y OpenCV incluye

métodos para convertir el espacio de color sin mayor complicación, pero es importante tenerlo en cuenta, ya que la gran mayoría de librerías, como `matplotlib`, utilizan por defecto RGB.

3. Filtrado manual: Se aplican filtros al `dataframe` para eliminar aquellos datos que no se precisen por diferentes razones (Por ejemplo, para hacer un entrenamiento solo con usuarios que lleven gafas, o entrenar dos modelos, cada uno con un tipo de experimento, y comparar sus resultados; etc.)

4. Permutación aleatoria [46]: Se aplica una permutación aleatoria sobre el `dataframe` del conjunto. Este paso es necesario debido al orden en el que aparecen las muestras, que es el mismo con el que se añaden los registros a la tabla `posicion_tiempo` de la base de datos de la aplicación de adquisición: de más antiguo a más reciente, lo que hace que los registros consecutivos en la aplicación también lo sean en la base de datos, y por tanto, que en el caso de las muestras generadas en el Experimento 2, que las contiguas sean muy similares entre sí, cosa que no es deseable durante el entrenamiento. Recorrer estas muestras incrementalmente para entrenar una red neuronal produciría sesgos y sobreentrenamiento. La permutación aleatoria evita este efecto.

5. Separación en entrenamiento, validación y test [46, 47]: Como suele ser habitual en aprendizaje automático, se divide meticulosamente el conjunto de datos para entrenar la red neuronal y controlar la corrección del proceso y la adecuación de los hiperparámetros a los datos, así como el rendimiento del modelo tras su entrenamiento.

- Subconjunto de entrenamiento: Es el subconjunto a partir del cual se ajustan los parámetros de la red neuronal, siendo este su mecanismo para aprender las relaciones entre los patrones de la información que se proporciona como entrada y sus etiquetas. Ocurre de forma automática. Este subconjunto debe ser el más grande (en términos numéricos, al menos un 70 % del conjunto completo debería usarse para entrenamiento), ya que es el único que tiene una implicación directa en el aprendizaje de la red neuronal.
- Subconjunto de test: Su cometido es ser utilizado tras el entrenamiento del modelo para evaluar su rendimiento (exactitud y pérdida entre otros parámetros). Para que la evaluación sea justa, la información de este subconjunto debe mantenerse estrictamente separada del resto de los datos. Al contener información que el modelo no ha "visto" durante el entrenamiento, incluso en aquellos casos en los que se haga correctamente, durante la evaluación con el conjunto de test cabe esperar un rendimiento ligeramente inferior al que muestra el *framework* durante el entrenamiento, sobretodo en las últimas etapas del mismo.
- Subconjunto de validación: Es un subconjunto que no resulta imprescindible, y de hecho, es habitual que ciertos artículos sobre la separación de conjuntos en aprendizaje automático lo omitan o lo mencionen de forma marginal. Si bien, utilizarlo permite al desarrollador visualizar información muy valiosa sobre el proceso de entrenamiento, con la que juzgar si se está haciendo correctamente y realizar

ajustes finos sobre los hiperparámetros del modelo (número de neuronas y capas, disposición y tipo de las mismas, funciones de activación de las neuronas, tasas de aprendizaje, optimizador, etc.) para optimizar su rendimiento. El proceso de validación tiene lugar en paralelo al entrenamiento, pero el modelo no aprende como resultado del mismo, sino que se evalúan las mismas métricas pero con el propio subconjunto. Por ello, no es necesaria una distinción estricta entre los datos de los subconjuntos de entrenamiento y validación como ocurre con el subconjunto de test. De hecho, lo más habitual y lo que mejor funciona es separar una porción del conjunto de entrenamiento para conformar el de validación (manteniendo una disjunción, pues lo que se busca en el conjunto de validación son muestras similares al de entrenamiento, no idénticas), no resultando este más grande que el conjunto de test.

Si bien se han dado pautas genéricas para dimensionar los diferentes subconjuntos, hay que considerar el contexto particular del proyecto y más concretamente, del conjunto de datos, para tomar una decisión. En este caso hay un total de 1380 muestras. Aunque se pueden aplicar técnicas de aumentación de datos para mejorar esta cifra artificialmente, es un conjunto relativamente pequeño. En estas circunstancias hay que tratar de asegurar una cantidad aceptable de datos en cada subconjunto, pero teniendo en cuenta que cualquier reducción en el conjunto de entrenamiento puede tener un impacto negativo significativo en el entrenamiento, aumentando las posibilidades de provocar *underfitting* u *overfitting* (Más información sobre estos conceptos a continuación en esta sección).

Se suele recomendar partir de un 80 % del conjunto de datos para el subconjunto de entrenamiento, y repartir el 20 % restante entre los de validación y test. En contraste de esto, cuando se trabaja con conjuntos más grandes, la situación cambia. Si bien es cierto que no existe consenso, pues hay quien defiende que en estos casos el impacto negativo de reducir el conjunto de entrenamiento en favor de los de test y validación es suficientemente bajo como para que compense hacerlo, ya que a cambio, la evaluación y la validación resulta mucho más fiable sin arriesgarse a hacer el entrenamiento incorrectamente [46].

No obstante, también se defiende que en los subconjuntos de validación y test no es tan crucial tener una cierta proporción como sí lo es partir de una cantidad mínima fija. Por ello, a mayor sea el conjunto, la proporción destinada al conjunto de entrenamiento también debería ser mayor, ya que de no ser así se estaría desperdiciando una cantidad de datos considerable [47].

Sea como sea, la conclusión más clara que se puede extraer de esta discrepancia es que en conjuntos pequeños, la división del conjunto de datos debe tratarse con más cuidado que en el caso de los conjuntos grandes.

Otro efecto que se puede producir con los conjuntos de datos tan pequeños como el del proyecto es que un subconjunto de test del 10 % o 15 % del conjunto completo no sea suficiente para hacer una evaluación correcta del modelo, por no haber margen

para diferir del resto del conjunto ni tampoco muestras suficientes para generalizar bien, causando sesgos e inconsistencias en los resultados, que no son representativos del comportamiento del modelo en un caso de uso común en el que se exponga a un usuario que no forma parte del conjunto de entrenamiento.

Esta incertidumbre se puede solucionar con la técnica de validación cruzada *leave-one-out* ("dejar uno fuera"). Se proporcionan más detalles sobre esto en la sección 5.5, de este Capítulo. La mención anticipada de este concepto se debe a que efectivamente, en el caso del proyecto, la separación típica aproximada de 80 %, 10 % y 10 % para los subconjuntos de entrenamiento, validación y test respectivamente, en este caso sirve para definir y ajustar los hiperparámetros (fase de selección); y para evaluar su rendimiento solo cuando se somete a sujetos que ya han aparecido en el conjunto de entrenamiento y por tanto resultan "conocidos" para el modelo (primera parte de la fase de evaluación). Por supuesto, si lo que se busca es analizar el comportamiento del modelo ante sujetos desconocidos, este método no ofrece resultados rigorosos, y la solución es recurrir a la técnica de validación cruzada.

En cualquier caso, para evitar que en el conjunto de entrenamiento aparezcan muestras muy similares a las del conjunto de evaluación –en cuyo caso aparecería un sesgo que mejoraría artificialmente el resultado al no forzar la generalización– para evaluar el modelo solo se utilizan muestras generadas por el Experimento 1. Estas, al tener coordenadas generadas aleatoriamente, no están correladas entre sí, por lo que perteneciendo al mismo sujeto, diferirán de aquellas que aparezcan en el conjunto de entrenamiento con mayor probabilidad que si se utilizaran indistintamente muestras generadas por el Experimento 2, que sí presentan una autocorrelación que alcanza máximos en las muestras temporalmente contiguas, que no son iguales pero si muy similares. Véase el Capítulo 3 para consultar más detalles sobre este aspecto [46].

Para el conjunto de validación no existe la necesidad de diferenciarse del conjunto de entrenamiento siempre y cuando sean disjuntos, por lo que puede componerse de una serie de muestras de menor tamaño pero con las mismas características [47].

No debe confundirse el término de "validación cruzada" con la connotación que se le da en el contexto de los *frameworks* de *Deep Learning*. El fin de la prueba de validación cruzada, en sentido estricto, es evaluar. Por tanto, se da la necesidad de definir en la tubería una fase previa a la evaluación Se ha establecido una separación aproximada de 70 %, 15 % y 15 % para los subconjuntos de entrenamiento, validación y test respectivamente, donde inicialmente se separan el subconjunto de test del `dataframe` previamente desordenado los registros asociados a determinados usuarios de la aplicación de adquisición mediante su código de usuario, que el desarrollador puede seleccionar antes de la ejecución del programa, buscando que el número de registros asociados a estos usuarios corresponda al 10 % del total de 1380 aproximadamente. De esta forma, el conjunto de test se forma con imágenes de sujetos que el modelo no ha "visto"durante el entrenamiento, lo cual es un requisito indispensable para que los resultados de la evaluación puedan considerarse realistas y rigurosos. Para formar el conjunto de validación, se to-

man los datos restantes y se separa aleatoriamente una muestra de tamaño similar al del conjunto de test. El resto de ambas operaciones es el conjunto de entrenamiento.

6. Detección facial: En este punto se itera sobre las imágenes de los diferentes subconjuntos de datos y se cargan en un modelo de detección facial mediante OpenCV. Se ha probado con Haar Cascade[1][2] y con YuNet. Ambos modelos pensados para CPU, no muy pesados, capaces de dar un buen rendimiento para trabajar a tiempo real con un flujo de video proveniente de una webcam, al menos a 640x480. Por supuesto, existen otros modelos con diferentes proporciones de precisión, velocidad y eficiencia en cuanto a recursos de hardware. En la práctica no existe un modelo que sea bueno en las tres, sino que se suele sacrificar una característica en favor de las demás. En el caso de los modelos del proyecto, Haar Cascade y YuNet son de lo más equilibrado que se puede encontrar, quizás priorizando ligeramente la rapidez y la eficiencia en cuanto a recursos, que es lo propio en sistemas a tiempo real, como se pretende demostrar que es el caso. No obstante, la implementación se ha hecho de la forma más modular posible, para que en un futuro resulte sencillo sustituir los modelos por otros como BlaceFace o LBP Cascade, más rápidos donde el hardware sea un factor limitante (por ejemplo en equipos móviles o embebidos); u otros más precisos en aplicaciones críticas en las que se requiera de alta precisión, como puede ser MTCNN.

La tasa de fallo del clasificador fue del 8,93 %. Es decir, del total de las muestras del conjunto, la implementación propuesta no fue capaz de detectar el rostro del sujeto en un 8,93 % de ellas.

7. Redimensión de imágenes: La resolución de las imágenes de salida de detector facial es, inevitablemente, variable. Cualquier mínimo movimiento dentro del plano puede alterar el área de la caja delimitadora que contiene la cara. En *Deep Learning* no es común utilizar entradas de dimensiones variables, pues, aunque es posible utilizando redes convolucionales, como es el caso; esto añade un nivel de complejidad considerable. Además, las redes entrenadas con entradas de tamaño variable son más propensas a aprender dependencias espurias del tamaño en lugar del contenido, y generalizar mal ante entradas nuevas que tengan una estructura diferente, debido principalmente a que la fragmentación en lotes no se puede hacer con facilidad cuando el tamaño de las muestras no es fijo. Una limitación importante de este proyecto es el reducido tamaño del conjunto de datos, lo que también hace a la red propensa al sobreentrenamiento, cuyos efectos son muy similares [47, 48].

Alternativamente se puede implementar alguna técnica de preprocesado para tratar las imágenes de resolución variable, de manera que a la red neuronal siempre lleguen con el tamaño de su capa de entrada. OpenCV proporciona un método eficiente para redimensionar las imágenes, tanto para aumentar como para reducir su tamaño. Al pertenecer a esta librería, no solo hace un buen trabajo a nivel visual, sino que está

[1] *Frontalface Default*, la versión principal entrenada para este fin

[2] Haar Cascade trabaja con imágenes en escala de grises. En caso de utilizarlo, el paso previo es hacer esta conversión del mapa de color.

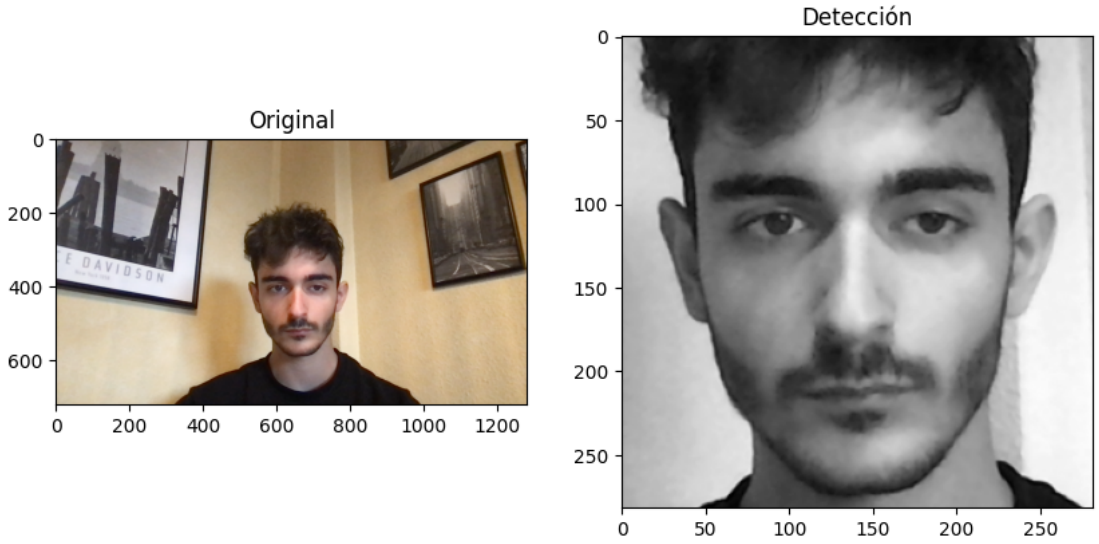


Figura 5.4: Ejemplo de detección facial con Haar Cascade resultante de los pasos 6 y 7.

pensado para mantener la integridad de la información, crucial en este ámbito. Para ello, proporciona varios interpoladores a elegir en función del uso que se le vaya a dar, recogidos en la tabla 5.2.

Cuando la relación de aspecto (es decir, la proporción del ancho y alto) de las imágenes es variable, hay varias posibilidades para regularizar sus dimensiones. La más sencilla consistiría en analizar los datos para decidir una resolución adecuada, preferiblemente inferior a las que se manejan en los datos, y redimensionar cada imagen con el método de OpenCV. Sin embargo, esto no es una buena idea, ya que se estaría potencialmente introduciendo una deformación que no es constante, alterando inconsistentemente los patrones que la red debe aprender, y por tanto, la información que se le está proporcionando. No es una alternativa viable.

Una mejor solución para este problema es introducir padding o relleno. Consiste en rellenar los espacios vacíos, resultantes al redimensionar, con píxeles blancos o negros, evitando deformaciones. Si se dispone de la imagen completa, como es el caso, se pueden usar los píxeles originales para rellenar dichos huecos, en lugar de píxeles en blanco o negro. El único inconveniente que puede aparecer con este mecanismo es que el modelo que se entrene con estas imágenes aprenda relaciones espurias entre el relleno y las etiquetas. En cualquier caso, esto no resulta tan preocupante como las deformaciones, ya que no se producen alteraciones en la información [48].

En el caso del detector facial con Haar Cascade, dado que las imágenes de salida siempre tienen la misma relación de aspecto de 1:1, no hay que preocuparse por estas problemáticas. Con buscar una resolución adecuada será más que suficiente. El único aspecto a tener en consideración es que al reescalar hacia arriba, el interpolador genera

información a partir de los píxeles originales mediante sus métodos internos, para rellenar los que faltan. Los interpoladores más básicos pueden causar alteraciones en los patrones de información. Los más complejos conservan mejor los detalles, pero el reescalado hacia arriba con estos resulta más costoso a nivel computacional. Por tanto, la resolución escogida debería ser, preferiblemente, más pequeña que la mayoría de tamaños de recortes faciales presentes en el conjunto de datos, siempre y cuando esto sea posible [45, 48].

En el punto anterior se ha mostrado como el tamaño mínimo es de 238x238, que a pesar de serlo, es superior a las resoluciones que se suelen manejar habitualmente en redes convolucionales, de 224x224. Cualquier valor comprendido entre o próximo a estos puede ser considerado para el modelo regresor que utiliza recortes faciales, aunque finalmente se ha optado por utilizar 282x282 en dicho modelo, que también es frecuente en redes convolucionales [11], y al ser algo más grande, el nivel de detalle será mayor .

8. Aumentación de datos por reflejo: Este es un apartado que requiere especial cuidado, ya que no se pueden seguir los pasos que se suelen hacer en otras redes neuronales con datos de otros tipos. En este proyecto, la posición de los ojos está en las imágenes está relacionada unívocamente con sus etiquetas. Alterar las imágenes variando su geometría rompe su relación con las coordenadas de la pantalla a las que se dirige la mirada. En otras palabras, si se mueven los ojos en la imagen, sus etiquetas asociadas deben ser modificadas en consecuencia o dejan de ser válidas.

Hay un caso en el que se puede hacer aumentación alterando imágenes y etiquetas: Reflejando ambas (Ver figura 5.5.[3].). Para las imágenes se puede aplicar un método de OpenCV [32]. Para reflejar una coordenada horizontal se aplica esta fórmula:

$$x_{reflejada} = x_{max} - x_{original}$$

9. Recolección final de datos y normalización: Una peculiaridad de las redes neuronales definidas para el proyecto consiste en utilizar tanto imágenes como valores numéricos en la entrada. Estos valores numéricos contienen características asociadas a la imagen que acompañan, con el fin de proporcionar a la red una característica implícita en la imagen de forma explícita, evitando que aprenda relaciones que no sean cruciales para el problema que resuelve el modelo:

- Datos posicionales inferidos por el modelo de detección facial: El cometido del detector facial es extraer el área que contiene un rostro de una imagen completa. Sin embargo, al hacer esto se está perdiendo la información posicional del rostro respecto de la imagen, pese a que es importante; pues una determinada mirada no se dirige al mismo sitio si el usuario está perfectamente en el centro de la imagen y de su pantalla que si se desplaza hacia la derecha o hacia la izquierda. Asimismo, el desplazamiento de los ojos para barrer una distancia determinada se hace más

[3]No es correcta en sentido estricto Las líneas no ilustran la posición mostrada en la leyenda, sino la reflejada, que es la que percibiría un observador desde el lugar de la cámara.

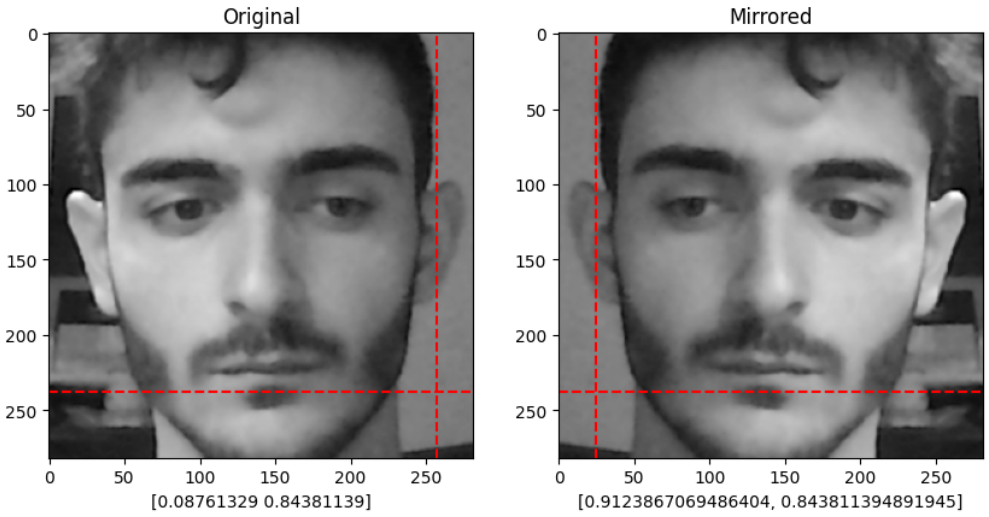


Figura 5.5: Ilustración del mecanismo de aumentación por reflejo.

pequeño conforme el usuario se aleja de la pantalla y esta pasa a ocupar una zona más reducida de su campo visual y viceversa. La información posicional del rostro es relativa, y viene acotada por la resolución de la imagen, por lo que se puede normalizar.

- Gafas: El conjunto de datos contiene imágenes de usuarios que llevaron gafas durante la adquisición, pero son minoría. Dado que no hay suficientes datos para entrenar un modelo aparte solo con usuarios portadores de gafas, se ha optado por dejar una entrada binaria con la que indicar al modelo si el usuario está usando gafas (0: Falso, 1: Verdadero). Dicha información, en el entorno del proyecto aparece en cada registro del conjunto, y en un entorno real podría provenir de una configuración a nivel usuario, o incluso de otra red o modelo encargados de detectar si el usuario lleva gafas.

- Dimensiones de pantalla: Este dato viene asociado al usuario en la aplicación de adquisición y se almacena en cada registro del conjunto. Dado que en una pantalla más grande, los ojos describirán un movimiento más amplio que en una pantalla más pequeña para desplazar la mirada a lo largo de esta en las mismas circunstancias, es preciso afirmar que existe una relación entre estos aspectos que el modelo puede aprender para mejorar su adaptabilidad y precisión. Al contrario que la resolución, esta magnitud es absoluta y no se puede normalizar.

En este punto, los datos están listos para ser procesados por el modelo regresor de *Eye-Tracking* con detección facial, que implementa la secuencia de éxito del caso de uso de Seguimiento Ocular de Baja Precisión. Para los modelos regresores basado en detección ocular, aparecen algunos pasos adicionales en la secuencia, relativos a esta función, entre los puntos 7 y 8.

5.3.2 Modelo II: Detección ocular basada en Haar Cascade

La implementación de detección ocular más obvia es aquella que desarrolla OpenCV en su documentación [45]. Consiste en introducir la imagen de salida del detector facial, de nuevo en un modelo Haar Cascade (`Eye`, modelo preentrenado para este fin), que aplica el mismo procedimiento que se ha detallado anteriormente para evaluar un conjunto de características, en este caso para identificar ojos en vez de rostros. Los pasos alternativos del procesado entre los puntos 7 y 8 para el modelo basado en detección facial, que conforman la secuencia para el modelo regresor basado en detección ocular son los siguientes:

a. Recorte del recorte facial: Para simplificar la operación reduciendo el espacio de búsqueda, se prescinde de la mitad inferior de la imagen del rostro, ya que los ojos siempre se encuentran en la superior. Adicionalmente se reduce el riesgo de falso positivo, al dejar de buscar ojos en una zona que no están.

b. Modelo de detección ocular: Debido a los aspectos del funcionamiento del Haar Cascade que se comentaron anteriormente, las regiones de interés inferidas vienen definidas por una caja delimitadora de relación de aspecto 1:1 una vez más. Por ello , una detección correcta devuelve una matriz bidimensional con un par de coordenadas de las regiones que contienen los ojos, en el formato de cuatro valores [X Y W H] en píxeles, siendo los dos primeros la coordenada de la esquina superior izquierda; y los dos últimos, el ancho y el alto [45].

c. Recorte ocular: El procedimiento es análogo al del recorte facial, ya que también se hace segmentando la matriz de una imagen más grande. Sin embargo, este caso resulta más complejo al contar con dos regiones de interés. La alternativa más obvia consiste en recortarlas directamente con los datos inferidos y almacenar por separado los recortes de cada ojo, de forma que el modelo debiera tener dos entradas de imagen en vez de una, con el consecuente incremento de la complejidad de su arquitectura, además de limitar la compatibilidad con otros modelos de detección ocular, ya que en este caso, no existe una metodología ampliamente utilizada como para considerarse estándar, como ocurre con las cajas delimitadoras en detección facial. Además, en estas circunstancias, la disparidad de los tamaños es un problema aún más preocupante, pues las condiciones lumínicas, la fisionomía de la persona o un mínimo giro de la cabeza pueden causar diferencias en el tamaño de las cajas inferidas por el modelo entre sí. Para solucionar estas problemáticas se ha implementado un postprocesado que mediante las siguientes fórmulas calcula, a partir de las coordenadas de las regiones de interés inferidas por el modelo detector, la caja más pequeña posible que contenga a ambas:

$$\text{eyes\_x1} = \text{mín}(x_0, x_1)$$

$$\text{eyes\_x2} = \text{máx}(x_0 + w_0, x_1 + w_1)$$

$$\text{eyes\_y1} = \text{mín}(y_0, y_1)$$

$$\text{eyes\_y2} = \text{máx}(y_0 + h_0, y_1 + h_1)$$

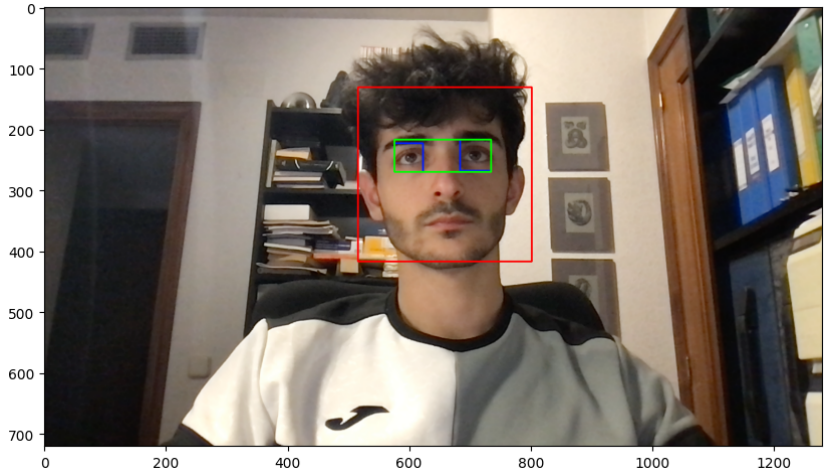


Figura 5.6: Ilustración de las regiones de interés detectadas. En rojo y azul, el rostro y los ojos localizados con Haar Cascade. En verde, la región de interés calculada mediante postprocesado.

donde (x_0, y_0, w_0, h_0) y (x_1, y_1, w_1, h_1) son los valores que definen las regiones de interés de cada uno de los ojos. La nueva caja viene definida por sus esquinas superior izquierda e inferior derecha: (eyes_x1, eyes_y1); (eyes_x2, eyes_y2)

Este método mejora el rendimiento de la implementación en cuanto a aciertos, ya que en ocasiones, el modelo hace una detección incorrecta en alguno de los ojos pero el rectángulo contenedor sigue siendo correcto, como se muestra en la figura 5.8.

Por desgracia, el rectángulo no mitiga el problema de la variabilidad de la resolución. De hecho, introduce uno nuevo, que es la variabilidad de la relación de aspecto. Como se ha explicado anteriormente, esto es algo totalmente indeseable, por las deformaciones que se producen al redimensionar a una resolución constante, que alteran la información. Sin embargo, se ha modificado el proceso para que la relación de aspecto se mantenga constante entre inferencias: Primero se obtiene el valor de la recta vertical que pasa por el medio del rectángulo:

$$\text{eyes\_mid\_x} = \frac{\text{eyes\_x1} + \text{eyes\_x2}}{2}$$

Después se calcula el ancho objetivo

$$\text{desired\_width} = (\text{eyes\_y2} - \text{eyes\_y1}) \cdot \text{ASPECT\_RATIO}$$

A continuación se obtienen valores alternativos para eyes_y1 y eyes_y2:

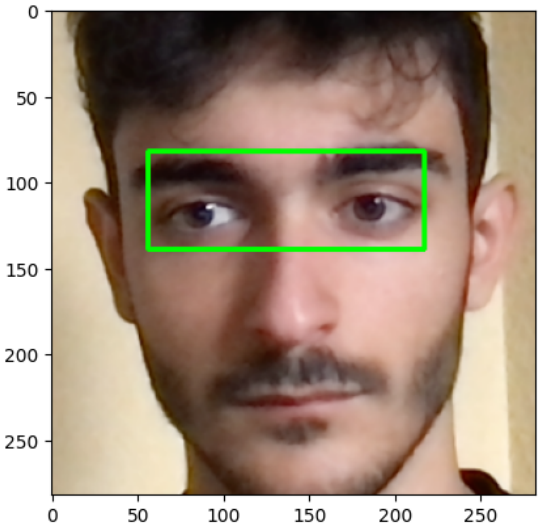


Figura 5.7: Ilustración de la región de interés final inferida por la tubería de detección ocular con Haar Cascade.

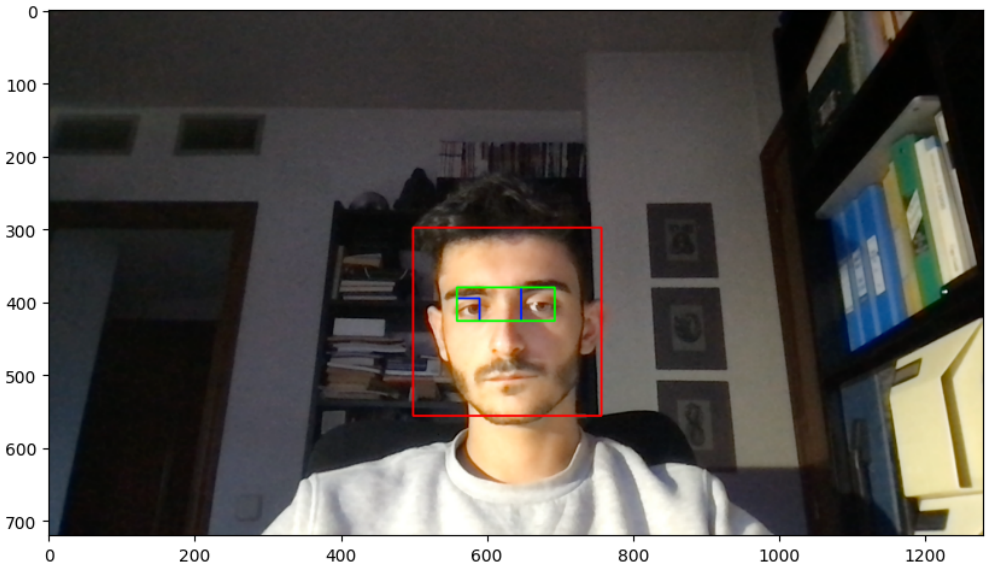


Figura 5.8: Ilustración de caso de corrección hacia delante de detección ocular con Haar Cascade. Autoría propia.

$$\text{eyes\_x1\_fixed} = \text{eyes\_mid\_x} - \frac{\text{desired\_width}}{2}$$

$$\text{eyes\_x2\_fixed} = \text{eyes\_mid\_x} + \frac{\text{desired\_width}}{2}$$

Donde $ASPECT\_RATIO$ es la relación de aspecto deseada. Su valor es configurable por el desarrollador. Para no definirlo arbitrariamente, se recurre a los datos del análisis recogido en el Apéndice A: Cuadernos de Jupyter I: Pruebas de detección facial, detección ocular y recorte ocular heurístico, mostrados en la tabla 5.3.

De esta se calcula que la relación de aspecto del rectángulo promedio es **2.82**. No obstante, hay que tener en cuenta que los valores decimales que se han usado para definir los rectángulos deben convertirse a enteros, ya que representan unidades de píxeles, no son adimensionales. Esta conversión, que se hace mediante redondeo al entero más próximo, introduce una cierta variabilidad inevitablemente, que al redimensionar produce deformaciones, lo que se traduce en una alteración de la información que reduce la efectividad del modelo que trate con estos datos. Si bien es una deformación mínima, lo deseable es que fuera nula. Para reducirla, lo más efectivo es incrementar el tamaño de las imágenes de entrada, disminuyendo así el error relativo que supone el redondeo. El tamaño de las imágenes depende principalmente de la cámara. Concretamente, de la distancia del usuario a la lente, del zoom óptico y, sobre todo, de la resolución; por lo que cabe esperar que la efectividad se incremente notablemente al utilizar cámaras más avanzadas. Esto, sumado al hecho de que, computacionalmente, esta aproximación es más exigente al utilizar dos modelos Haar Cascade en vez de uno; acota su ámbito de uso a equipos de altas prestaciones. A cambio, la precisión a esperar también es mayor que, por ejemplo, en la anterior implementación sin detección ocular.

d. Mecanismos de control: A pesar de que la detección ocular se hace también con Haar Cascade (utilizando la versión Eye, entrenada para este fin), el rendimiento del modelo en este caso de uso es bastante inferior, pues resulta habitual que el número de detecciones sea diferente de 2 (Es decir, que no llega a detectar ambos ojos o además de los ojos, falsos positivos). Los motivos que explicarían esto son los siguientes:

- Los ojos ocupan muchos menos píxeles que el rostro completo y por tanto, el número de características a evaluar para detectarlos es inevitablemente inferior. Esto dificulta las detecciones correctas y facilita las incorrectas (falsos positivos), ya que al ser un número reducido de características, no es raro encontrarlas en otras zonas de la imagen.
- Visualmente, los ojos, a las distancias que se manejan en las imágenes del conjunto de datos, también son más facilmente confundibles que los rostros.
- La detección ocular no es una práctica tan común como la detección facial. Es muy probable que las versiones Frontalface de Haar Cascade se hayan entrenado más

cuidadosa y exhaustivamente que la versión Eye, para detección ocular.

En adición a la implementación básica documentada por OpenCV y en respuesta a esta problemática, se implementan las siguientes operaciones:

- Si el tamaño de la salida del detector es inferior a 2 (Lo que significa que no se han detectado los dos ojos), la imagen se considera no válida.
- Si el tamaño de la salida del detector es superior a 2 (Hay al menos un falso positivo, pero es probable se hayan detectado correctamente los dos ojos), no es viable invalidar la iteración como en el anterior caso, ya que es demasiado habitual que el modelo haga más de dos detecciones por imagen como para prescindir de aquellas en las que esto ocurra. La solución implementada, con buenos resultados, consiste en mantener solo las dos detecciones más grandes, ya que los falsos positivos suelen ser causados por elementos pequeños, como pecas, lunares, pendientes, manchas, etc.
- Al aplicar el procedimiento descrito anteriormente para calcular el área de una región de interés rectangular que contenga a ambos ojos (Sin la corrección para regularizar la relación de aspecto), se verifica su relación de aspecto. Si es inferior a 2 (Valor muy inferior al promedio de 2.82 observado en el conjunto del proyecto) es coherente asumir que al menos una de las detecciones es incorrecta con un error demasiado grande como para ser corregido (Al contrario de lo que sucede en la figura 5.8) y que por tanto, la imagen no se puede considerar válida

Las implicaciones de invalidar una imagen varían en función del contexto. Si se hace referencia al sistema a nivel funcional, se sigue el diagrama de secuencia definido en el Capítulo anterior. Según este, ocurriría la secuencia de éxito del caso de uso `SeguimientoOcularBajaPrecisión`, pudiendo interpretar, para más rigor, que el caso de uso `SeguimientoOcularAltaPrecisión` falla al no lograr detectar los ojos y, como postcondición, continúa siguiendo los mismos pasos que la secuencia de éxito del caso `SeguimientoOcularBajaPrecisión`, que se recuerda es aquel que utiliza el modelo regresor de *Eye-Tracking* con detección facial, para inferir las coordenadas.

Si por el contrario se hace referencia al preprocesado del conjunto de datos para el entrenamiento de la red neuronal, la imagen invalidada simplemente se descarta. No se usará para entrenamiento, ni validación, ni evaluación.

e. Redimensión de recortes oculares: Finalmente se redimensionan las imágenes a una resolución constante cuya relación de aspecto sea la misma que la que se ha utilizado como objetivo a lo largo del proceso. El promedio recogido en el análisis es de 170x60. En esta aproximación, hay una probabilidad elevada de que el reescalado se deba hacer hacia una resolución superior. Por ello, puede resultar interesante, a fin de conservar el máximo nivel de detalle, utilizar un interpolador cúbico o superior [48].

La tasa de fallo de esta tubería es **33,05 %**, un valor mucho peor que en la anterior implementación.

En vista de los datos anteriores, se concluye que la implementación de detección ocular con Haar Cascade tiene una tasa de fallo suficientemente considerable como para plantear alternativas.

Utilizar un modelo de detección ocular puede servir para filtrar el conjunto de datos, ya que descartaría todas aquellos registros en cuyas imágenes, los ojos no sean perfectamente visibles por estar ocluidos, cerrados, tras el pelo; o que el usuario no estuviera prestando atención a la pantalla durante la realización del experimento como cabría esperar, asegurando una buena calidad de los datos de entrenamiento.

Sin embargo, la presencia de dichos registros incorrectos en el conjunto es muy inferior al 33 %, que es la proporción de imágenes que descarta la implementación anterior. Esta baja permisividad, en un entorno real de funcionamiento, podría ser perjudicial, al punto de resultar contraproducente. De hecho, el valor añadido de filtrar los fotogramas correctos en estas circunstancias es menor, pues en el supuesto caso de hacer *Eye-Tracking* cuando los ojos no son perfectamente visibles, lo peor que puede pasar la exactitud de las inferencias del modelo regresor de *Eye-Tracking* baje de forma transitoria. Además, en el sistema propuesto no existe (ni se implementa en este proyecto) un control de flujo basado en detección ocular, más allá de seleccionar el modelo. De hecho, según el diagrama de secuencia, si la detección facial se ejecuta correctamente, el sistema va a devolver coordenadas en cualquier caso, resulte o no la detección ocular.

Otro punto a tener en cuenta es que los modelos detectores faciales utilizan las facciones del rostro para identificarlo. El hecho de que se detecte implica una alta probabilidad de que los ojos sean perfectamente visibles, incluso si el detector ocular no es capaz de identificarlos En adición a esto, los recortes faciales de la implementación del proyecto basada en Haar Cascade tienen una cualidad interesante al mantener su relación de aspecto constante: Los ojos aparecen siempre en la misma porción de la imagen, en ambos ejes. El área que esta ocupa es variable en cuanto a número de píxeles: depende del tamaño de la imagen. Sin embargo, en relación a la imagen completa, su tamaño y su ubicación son constantes. En otras palabras, las coordenadas normalizadas (entre 0 y 1) que definen la mencionada región, son constantes para todas las imágenes. Ello motiva un método que tome ventaja de este hecho para calcular y devolver un recorte ocular de forma similar a lo que ocurre con la implementación anterior, pero sin recurrir a un modelo para ello. Las ventajas de esta aproximación alternativa son las siguientes:

- Tiempos de ejecución muy inferiores, al realizar operaciones mucho más simples.
- Relación de aspecto constante, al basarse en datos posicionales constantes y recibir como entradas imágenes de resolución también constante, eliminando el problema de las deformaciones.
- Reducción de las pérdidas a 0, ya que no hace una detección, sino simplemente un

recorte

Como contraparte, la precisión esperada de una red neuronal cuyos datos de entrenamiento se procesen con este mecanismo no puede ser tan alta como por ejemplo, en la anterior aproximación, ya que el recorte ocular no va a ser igual de exacto, y debería contar con un pequeño margen añadido para salvar las mínimas desviaciones que puedan aparecer. También, al no filtrar de ningún modo, se puede dar el caso de que ciertos recortes que en otras circunstancias no serían válidos, lleguen a los datos de entrenamiento de la red neuronal, aunque este problema se puede solucionar fácilmente reaprovechando el detector ocular anterior solo para filtrar, implementando el recorte ocular con el nuevo mecanismo propuesto.

El método final calcula una región de interés rectangular contenedora de los dos ojos mediante las siguientes ecuaciones:

$$\text{eyes\_x1} = \text{round}\left((w_{\text{img}} \cdot \text{norm\_eyes\_x1}) - \left(\frac{\text{PADDING}}{2} \cdot \text{aspect\_ratio}\right)\right)$$
$$\text{eyes\_x2} = \text{round}\left((w_{\text{img}} \cdot \text{norm\_eyes\_x2}) + \left(\frac{\text{PADDING}}{2} \cdot \text{aspect\_ratio}\right)\right)$$
$$\text{eyes\_y1} = \text{round}\left((h_{\text{img}} \cdot \text{norm\_eyes\_y1}) - \left(\frac{\text{PADDING}}{2}\right)\right)$$
$$\text{eyes\_y2} = \text{round}\left((h_{\text{img}} \cdot \text{norm\_eyes\_y2}) + \left(\frac{\text{PADDING}}{2}\right)\right)$$

Donde $PADDING$ es un parámetro configurable por el desarrollador que sirve para agrandar la región de interés manteniendo la relación de aspecto, necesario para evitar perder información si se observa que la posición de los ojos varía demasiado;

w_img y h_img son constantes que almacenan las dimensiones de las imágenes de entrada. En este caso, se reutiliza la tubería del modelo anterior, por lo que el valor es 282 para ambas;

norm_eyes_x1, norm_eyes_x2, norm_eyes_y1 y norm_eyes_y2 son constantes que contienen los valores promedios de las coordenadas normalizadas de las regiones de interés inferidas por el detector ocular basado en Haar Cascade Eye al procesar el conjunto de datos, como parte del análisis recogido en el Apéndice A, y con los siguientes valores, tomados de la tabla 5.3:

$$norm\_eyes\_x1 = 0{,}225$$
$$norm\_eyes\_x2 = 0{,}743$$
$$norm\_eyes\_y1 = 0{,}303$$

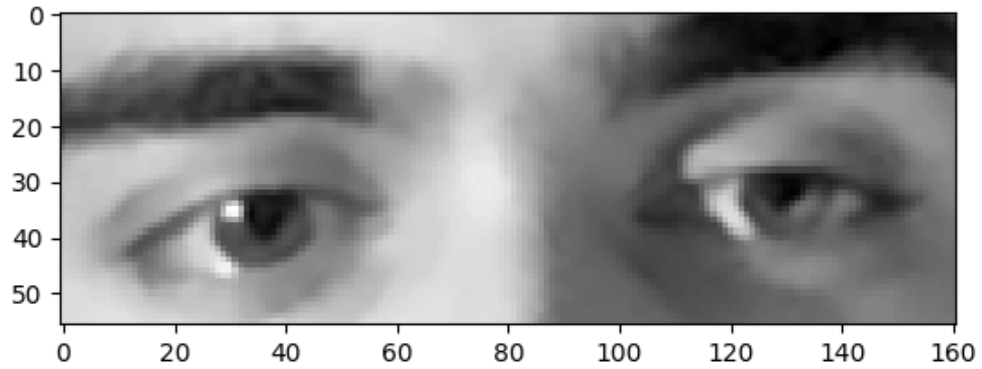


Figura 5.9: Recorte ocular resultante de la tubería de preprocesado del Modelo II.

$$norm\_eyes\_y2 = 0{,}484$$

Tal y como ocurría en anteriores implementaciones, las regiones vienen definidas por sus esquinas superior izquierda e inferior derecha: $(eyes\_x1, eyes\_y1); (eyes\_x2, eyes\_y2)$.

Introduciendo estos valores en el método propuesto en el Apéndice A, las imágenes resultantes tienen una resolución de 161x56.

La tasa de fallo de esta tubería es idéntica a la primera, la del Modelo I, basada en detección facial, ya que los pasos son idénticos a excepción del recorte ocular mediante estadísticas, que tiene probabilidad de fallo nula. Dado que las imágenes que logra son muy similares a las que se lograría usando la tubería propuesta para detección ocular con el modelo de Haar Cascade *Eye*, pero con una tasa de fallo mucho menor, solo esta será objeto de evaluación.

5.3.3 Modelo III: Detección facial y ocular basada en YuNet

A pesar de que este proyecto es una prueba de concepto, y lo que busca es demostrar funcionalidad, y no tanto la corrección de la misma, dada la superioridad técnica de la red convolucional YuNet frente a otras soluciones de detección facial más tradicionales, además de la adecuación al caso de uso, ya que es muy ligera a nivel se ha optado por desarrollar una implementación de la misma en el proyecto. Estas son sus diferencias respecto a Haar Cascade [33, 42]:

- Al ser una red convolucional, la extracción de características es más compleja y dinámica.
- Su rendimiento no es apenas dependiente de la configuración.
- No es multiescala como Haar Cascade, pero sí permite manejar diferentes tamaños de entrada fijando su valor en los parámetros de un objeto que gestiona su configu-

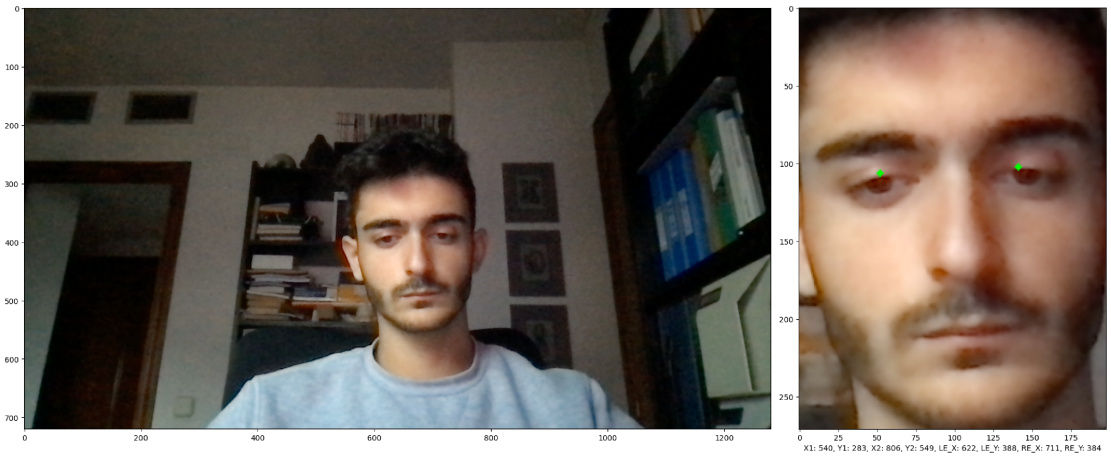


Figura 5.10: Ilustración de regresión de coordenadas de facciones sobre recorte de región de interés facial con YuNet.

ración. En cualquier caso no es un problema, pues en un supuesto caso de uso real, durante la inicialización se fijaría la resolución deseada, compatible con la cámara, y permanecería constante durante toda la sesión.

- Su entrada es tridimensional, ya que trabaja con imágenes a color.
- Su salida, entre otras cosas, no son las coordenadas de una ventana cuadrada, sino una región de interés inferida por el modelo, que no necesariamente es cuadrada.
- Además de inferir las coordenadas de la región que contiene el rostro, también devuelve las coordenadas de ciertas facciones. Entre ellas los ojos.

Solo por este último punto merece la pena considerar esta aproximación, pues permite implementar detección facial y ocular en el mismo paso, agilizando mucho el proceso y por supuesto, sin la elevada tasa de fallos del clasificador Haar Cascade *Eye*.

El proceso es el siguiente: El modelo infiere 15 coordenadas, siendo las cuatro primeras las correspondientes a la región de interés que contiene el rostro, en un formato similar al de Haar Cascade; y las cuatro posteriores, por lo que se almacenan los correspondientes píxeles para recortarla [42].

Como suele ocurrir con los modelos regresores, siempre devuelven una predicción. YuNet puede devolver una matriz vacía si no detecta rostros, pero si detecta al menos uno, la salida va a ser completa en cualquier caso, con las coordenadas del rostro y de todas las facciones, aunque no estuvieran visibles [42]. Por tanto no serviría para filtrar las muestras de mala calidad del conjunto pero sí mejoraría la elevada tasa de fallos de detección del modelo cuya tubería usa Haar Cascade para la detección ocular, asumiendo, tanto en este caso como en el del modelo cuya tubería usa el método heurístico para

seleccionar la región que contiene los ojos, que los ojos contienen características que cualquier detector facial evaluaría para cumplir su cometido. Por tanto, si el sistema ha sido capaz de detectar el rostro, los ojos son visibles en el mismo con una muy alta probabilidad, en cualquier caso superior al 67 %, la proporción de detecciones correctas que logra la tubería de detección ocular con Haar Cascade.

Las coordenadas inferidas de los ojos, como se muestra en la figura 5.10, no son una región de interés, sino el centro de los mismos. A partir de este momento es necesario decidir las operaciones que se implementarán para calcular una región de interés con los datos que se tienen actualmente.

Como ya se explicó anteriormente, la variabilidad de las resoluciones y relaciones de aspecto son un problema grave cuando se trata de aprendizaje profundo, ya que las deformaciones producidas por el interpolador encargado de redimensionar alteran la relación entre los datos y sus etiquetas.

El formato 1:1 es sencillo de manejar, no vulnerable a deformaciones al convertir a enteros (ya que el proceso es tan simple como establecer un ancho y un alto objetivos idénticos) y resulta compatible con otras soluciones existentes, como la que se mostrará más adelante en los hiperparámetros del modelo asociado a esta tubería en la siguiente sección. La desventaja en este caso, por supuesto, es la necesidad de manejar dos entradas de imagen en vez de una.

Inicialmente se calcula la mitad del lado de una caja delimitadora cuadrada, donde δ es un parámetro cuyo valor define la proporción de la distancia horizontal entre centros de ojos que se utilizará para definir el ancho de la caja. Se ha comprobado que utilizar 0.8 en vez de 1 mejora ligeramente los resultados al prescindir de píxeles innecesarios sin recortar los ojos:

$$\text{eyes\_radius} = \left\lfloor \frac{\delta \cdot |re_x - le_x|}{2} \right\rfloor$$

A continuación, una coordenada Y única, resultante de promediar las coordenadas del eje Y de cada ojo, logrando de esta manera una relación de la posición vertical de los ojos respecto al centro de las cajas con la inclinación de la cabeza, que el modelo puede aprender:

$$e_y = \left\lfloor \frac{re_y + le_y}{2} \right\rfloor$$

Región de interés ojo izquierdo:

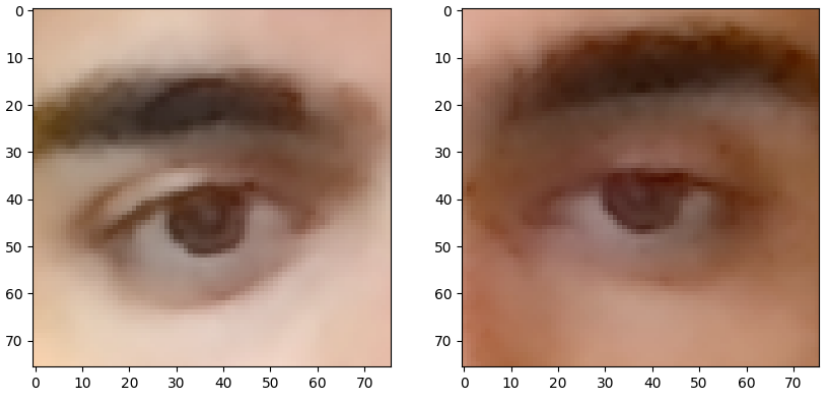


Figura 5.11: Regiones de interés oculares calculadas mediante el procedimiento anterior.

$$
\begin{aligned}
le\_bb\_x1 &= le_x - \text{eyes\_radius} \\
le\_bb\_x2 &= le_x + \text{eyes\_radius} \\
le\_bb\_y1 &= e_y - \text{eyes\_radius} \\
le\_bb\_y2 &= e_y + \text{eyes\_radius}
\end{aligned}
$$

Región de interés ojo izquierdo:

$$
\begin{aligned}
re\_bb\_x1 &= re_x - \text{eyes\_radius} \\
re\_bb\_x2 &= re_x + \text{eyes\_radius} \\
re\_bb\_y1 &= e_y - \text{eyes\_radius} \\
re\_bb\_y2 &= e_y + \text{eyes\_radius}
\end{aligned}
$$

Adicionalmente se ha añadido un mecanismo de aumentación mediante ruido, que toma una porción de muestras aleatoria y les añade ruido Gaussiano uniforme. Este no afecta a los pesos del modelo. El objetivo de hacer esto es lograr una mejor capacidad de generalización, ya que en cierto modo se fuerza el aprendizaje de relaciones más fundamentales y robustas, y en ningún caso tiene implicaciones posicionales incompatibles con este proyecto.

El tamaño promedio de los recortes oculares es de 103x103. Se recogen más detalles sobre este análisis en el Apéndice A: Cuadernos de Jupyter I: Pruebas de detección facial, detección ocular y recorte ocular heurístico.

Esta tubería de preprocesado logró una tasa de fallo de detección del 8,43 % del total del conjunto, siendo esta cifra la mejor hasta el momento.

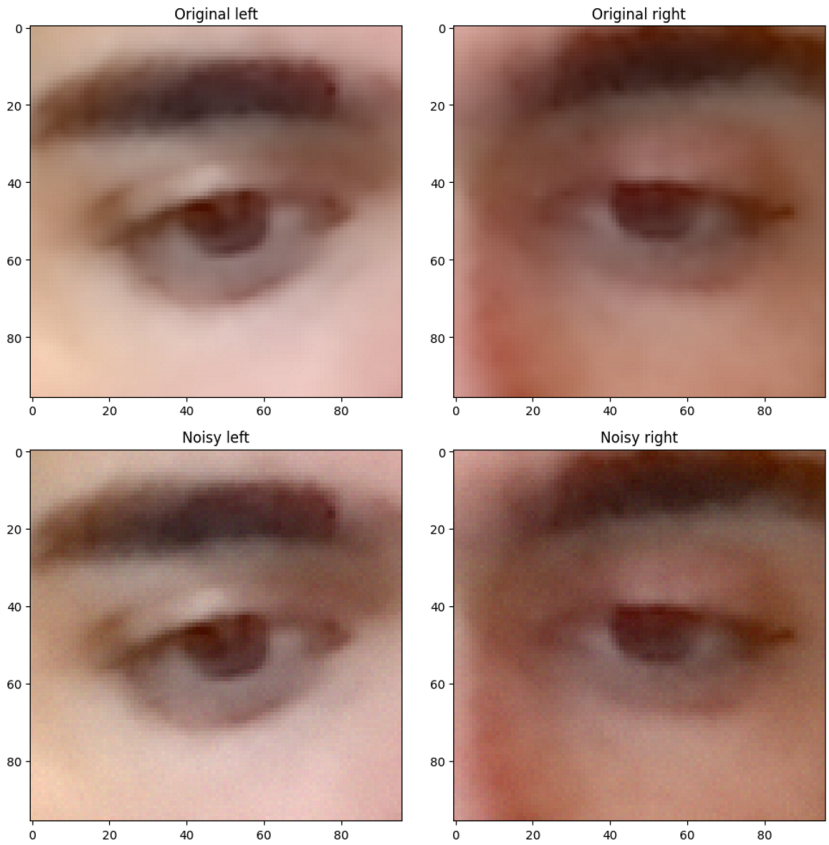


Figura 5.12: Aumentación por adición de ruido Gaussiano para mejorar la capacidad de generalización.

5.4 Selección: Arquitectura e hiperparámetros de las redes neuronales

Tal y como se ha venido explicando en las secciones y capítulos anteriores, el sistema en desarrollo debe ser capaz de inferir las coordenadas de la pantalla a las que se dirige la mirada del usuario a partir de dos posibles aproximaciones, que se implementan con redes neuronales cuya diferencia fundamental es el formato de datos de entrada.

Todas son redes neuronales convolucionales, lo que significa que combinan capas convolucionales con capas densas –formadas exclusivamente por neuronas artificiales– propias de las redes neuronales profundas tradicionales. Por el enfoque llevado a cabo, la capa de entrada incluye imágenes y características numéricas relevantes para el problema, que de otra forma se perderían. Estas son:

- Las coordenadas normalizadas de la región de interés que contiene el rostro, inferidas por el detector ocular; lo que permite al modelo aprender las relaciones entre las coordenadas de la mirada y la posición del usuario respecto al dispositivo.
- Las dimensiones de la pantalla, ya que también existe una relación entre el tamaño de la misma y el desplazamiento de los ojos necesario para barrer una porción de píxeles dada.
- Una variable binaria cuyo valor indica si el sujeto de la muestra lleva gafas, a fin de que los modelos aprendan una relación que les permita activar ciertas neuronas para actuar de forma diferente ante estos usuarios.

Si bien las redes neuronales profundas funcionan bien a la hora de resolver problemas numéricos, presentan un problema para trabajar con imágenes que las invalidan para *Eye-Tracking*: La predicción viene sesgada por las neuronas o unidades de la capa entrada que introducen la información. En otras palabras, las redes neuronales profundas aprenden y asocian relaciones a los píxeles en los que estas aparecen en el conjunto de entrenamiento, y por tanto a unidades de capa de entrada, ya que en *Deep Learning*, siempre se definen tantas como píxeles por canales (es decir, una por píxel en imágenes monocromáticas y tres en imágenes policromáticas). Esto hace a la red sensible al tamaño y posición de los elementos en el plano de la imagen, al punto de que, si difieren del conjunto de entrenamiento, las predicciones se vuelvan erróneas, suponiendo una incapacidad para generalizar totalmente incompatible con las circunstancias de este sistema, en el que la variabilidad de las muestras es muy elevada.

Las capas convolucionales bidimensionales evalúan un kernel convolucional también bidimensional –cuyo tamaño es ajustable por el desarrollador– en la matriz de valores que conforman los píxeles de la imagen de entrada. Esto guarda cierta relación de similitud con la aproximación de ventana deslizante para la evaluación de características de Haar explicada anteriormente, aunque en este caso el desarrollador no especifica los valores del

kernel. Durante el entrenamiento se ajustarán como si fueran neuronas, de acuerdo a las necesidades del problema a resolver. El resultado es una imagen filtrada que contiene la información más relevante sobre las características de la misma para las siguientes capas del modelo. Tras las capas convolucionales se implementa una capa de *MaxPooling*, cuyo cometido es reducir el número de muestras a su entrada mediante una técnica de ventana deslizante, cuyo tamaño es también ajustable por el desarrollador. Para ello, en cada paso, forma un píxel con el valor más alto de aquellos bajo la matriz de la ventana [12].

Tras todo el flujo de capas convolucionales y de *pooling*, los datos correspondientes a las imágenes se concatenan con las entradas numéricas mencionadas anteriormente. A continuación se define una capa densa oculta encargada de aprender las relaciones entre imágenes y características numéricas, y a su vez, entre estas y las dos neuronas de salida [12, 49].

Las capas densas se componen de perceptrones totalmente conectados. Esto significa que todas las salidas de la capa anterior se conectan a cada neurona artificial. En este caso, se ha optado por la función de activación ReLU (*Rectified Linear Unit*) para todas las capas neuronales, ya que permite romper la linearidad de las relaciones aprendidas, es eficiente a nivel computacional, y evita el problema de desvanecimiento de gradiente, por el que la propagación hacia atrás deja de ser capaz de hacer ajustes significativos en los pesos de las primeras capas, causando problemas en el entrenamiento [49].

El optimizador utilizado en ambos casos es Adam, el que suele ser la opción más habitual en *Deep Learning* debido a sus buenos resultados en general para redes de entrenamiento supervisado [11, 12].

En cuanto a la función de pérdidas, la más habitual y probada para estos casos es el error cuadrático medio. Al ser cuadrática, penaliza más los errores grandes que los pequeños, en relación a una función lineal, como el error absoluto medio. Sin embargo, debido a la presencia de potenciales *outliers*, que son valores muy diferentes a la tendencia, al componerse el conjunto de muestras adquiridas en entornos no controlados, y con la prudencia de asumir que este contexto se mantendría en condiciones de uso reales, se ha optado por utilizar la función de pérdidas de Huber, menos sensible a los *outliers*, con un valor de delta = 1.2 [50].

El número de épocas en la fase de selección se fija intencionadamente alto, a 50, ya que se implementa el callback de finalización adelantada, o *Early-Stopping*. Los *callbacks* son funciones que proporciona el *framework* con las que manejar ciertos parámetros del entrenamiento en función de otros. En este caso se utiliza para detener el entrenamiento antes de alcanzar el número de épocas establecido si el valor que devuelve la función de pérdida no muestra una disminución significativa en un número de épocas establecido, denominado "paciencia" (*patience*). Esto es ideal, ya que el estancamiento de la función, o lo que es peor, un punto de inflexión, serían un claro indicio de límite que separa un ajuste correcto del sobreajuste, está cerca de ser rebasado si no lo ha sido ya. Además, cuando concluye el entrenamiento, se recuperan los parámetros del número de épocas

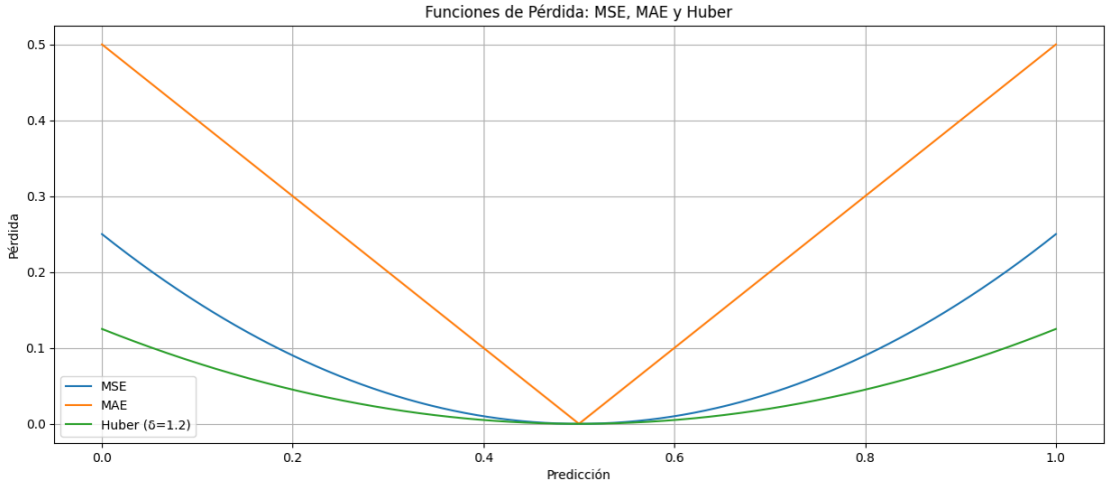


Figura 5.13: Gráfico de las diferentes funcionas de pérdida implementadas en Keras, consideradas en el proyecto. Autoría propia.

total menos el valor del parámetro `patience` [12].

Son dos las consideraciones atípicas que hay que tener a la hora de definir las arquitecturas de los modelos de este proyecto:

- Los datos de entrada, como se ha mencionado anteriormente, son tanto imágenes como características numéricas. El propósito de las capas convolucionales consiste en extraer características de las imágenes con las que trabajan. No tiene sentido entonces introducir en ellas las características numéricas de entrada, sino hacer que tanto estas como las extraídas por el conjunto convolucional entren en un bloque totalmente conectado de capas densas, más adecuado para aprender las relaciones entre estos datos.
- El reducido tamaño del conjunto de datos del proyecto supone una limitación en el aspecto de los hiperparámetros, ya que la complejidad de la arquitectura debe ser proporcional al tamaño del conjunto: Cuantas más neuronas (y por tanto, parámetros) haya que ajustar, más datos son necesarios para hacerlo correctamente. Una red densa puede aprender más relaciones, y más complejas, que un modelo más simple, que se traduzcan en una mejora de la exactitud en sus predicciones; pero esto solo ocurrirá si se utiliza un conjunto suficientemente grande [47].

5.4.1 Modelo II: Red neuronal convolucional ligera

Esta red trabaja con imágenes resultantes de recortar de la imagen completa una porción que contiene los ojos, al ser esta la principal región de interés. De este modo, el modelo trabaja con una información muy concisa, ya que las imágenes contienen lo fundamental

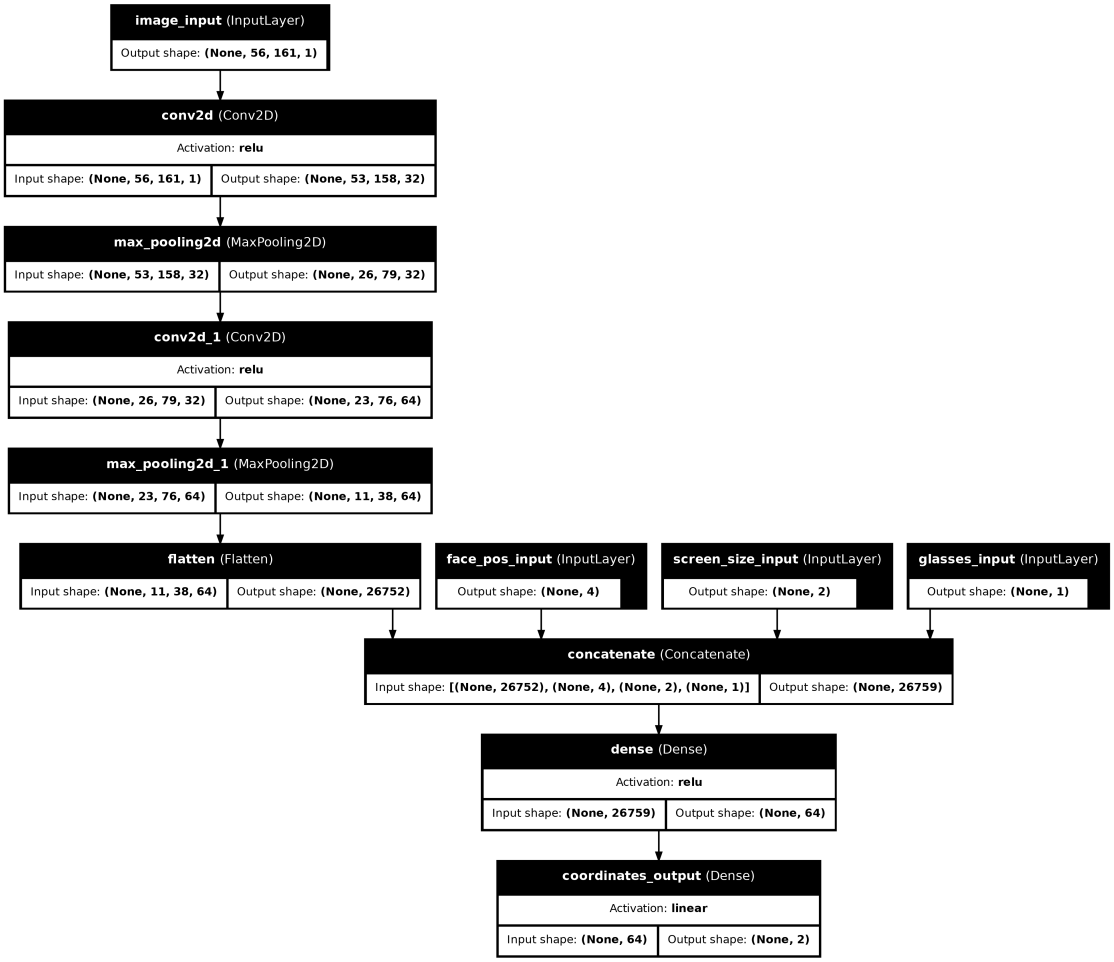


Figura 5.14: Arquitectura del modelo que utiliza Haar Cascade para detección facial y método heurístico para discriminación ocular en su tubería de preprocesado

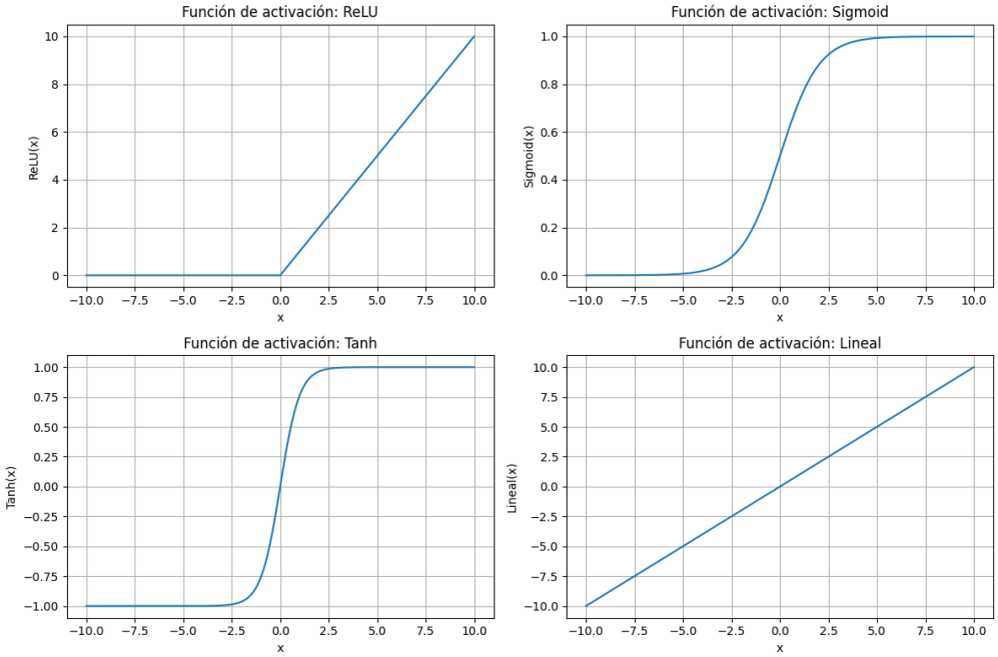


Figura 5.15: Gráficos de las implementaciones en Keras de las funciones de activación más comunes. Autoría propia.

para resolver las coordenadas de salida, y omiten otras regiones con poca o nula relación con la dirección de la mirada. Esto supone una mejora en el proceso de entrenamiento, ya que al hacer los patrones involucrados en la mirada evidentes, la convergencia de la red se producirá con menos muestras, a la vez que se evita el aprendizaje de relaciones espurias y por tanto, resulta más robusta frente al sobreentrenamiento; potencialmente logrando un modelo ligero, preciso y eficiente. A cambio, es necesario utilizar un flujo de preprocesado largo y complejo, que en ocasiones, como se ha visto con el detector ocular, tiene una tasa de fallos elevada.

Una peculiaridad de este modelo es la función de activación de la capa de salida. Si bien, lo más común en regresión es utilizar funciones lineales, en este caso se ha optado por utilizar una función sigmoide, propia de modelos clasificadores. El motivo de esto es que realmente, las capas de salida deben ser consistentes con los datos que va a emitir, razón por la cual en regresión se definen tantas neuronas como valores contenga el vector de salida, y en clasificación, tantas como clases. Lo mismo ocurre con la función de activación: Esta debe ser capaz de devolver todo el rango de valores inferibles por el modelo. Por ejemplo: En un modelo utilizado para estimar temperaturas en grados Celsius, tanto positivas como negativas, se define una capa de salida con función ReLU [49].

La función ReLU no puede ser negativa, con lo cuál, pase lo que pase en el interior de esta red, en el mejor de los casos, al tratar de inferir un valor de temperatura negativa,

el resultado a la salida será 0. Por tanto, es necesario utilizar una función lineal en esta capa. Esta sí puede devolver los valores negativos deseados.

Aplicando esta idea al modelo regresor de *Eye-Tracking* con entrada ocular, dado que este modelo infiere coordenadas bidimensionales normalizadas entre 0 y 1, resulta coherente utilizar una de estas funciones acotadas, e incluso beneficioso, pues fuerza la red a trabajar dentro de dicho rango, siendo el único en el que las predicciones pueden ser correctas; lo que a su vez puede acelerar la convergencia [49].

Al manejar un tamaño de entrada reducido, el entrenamiento de este modelo es suficientemente sencillo a nivel computacional como para variar ciertos hiperparámetros que mejoren la exactitud de las predicciones a costa de elevar la carga durante el entrenamiento. Por ello, se ha ajustado el **tamaño del lote a 16** respecto al estándar de 32. Dado que esto causa que el número de bloques aumente y en consecuencia, en cada época el modelo sea expuesto a más bloques, que además, son más simples, es necesario bajar la **tasa de aprendizaje**, del valor estándar de 0.001 a **0.0005**. Para finalizar, el número de épocas debe ser uno que entrene el modelo tanto como sea posible sin cometer sobreajuste, por lo que habrá que evitar que las métricas de validación se diferencien demasiado de las de entrenamiento. En cualquier caso esto no sucede, y el mecanismo de *EarlyStopping* se activa a las 48 épocas con un valor de paciencia de 5. A juzgar por los gráficos de la figura 5.16 , se ha fijado el **número de épocas a 43**.

5.4.2 Modelo I: Red neuronal convolucional intermedia

Esta red trabaja con imágenes de rostros. Concretamente los recortes faciales generados con el detector facial basado en Haar Cascade Frontalface desarrollado en la sección anterior. La cantidad información que contienen estas imágenes siempre es superior a recortes más pequeños de ciertas facciones. Si bien haciendo esto, el modelo estaría recibiendo píxeles cuyos valores no aporten nada al problema a resolver, lo que puede ser perjudicial debido al posible sobreentrenamiento, pero también puede mejorar la flexibilidad del sistema, ya que de esta manera, el modelo puede apoyarse en otros aspectos de la imagen para inferir las coordenadas, como pueden ser la orientación del rostro o la expresión, posibilitando la predicción en casos en los que la anterior aproximación no podría trabajar. Además, al prescindir del bloque funcional de detección ocular, el preprocesado resulta mucho más simple. A cambio, la arquitectura deberá ser algo más densa para poder aprender más relaciones, y con el reducido tamaño del conjunto de datos del proyecto, la precisión esperable será menor inevitablemente.

En este modelo, debido al gran tamaño de la capa de entrada, es imprescindible fijar el **tamaño del lote a 16**, y no simplemente preferible, ya que en el momento de extraer los sujetos con más muestras para evaluación durante el proceso de validación cruzada, el conjunto de entrenamiento quedaría demasiado pequeño y se produciría subajuste, lo que se evidenciaría como una recta sin apenas pendiente en la función de pérdidas. La tasa de aprendizaje se ha mantenido por defecto (**0.001**) para tratar de reducir el número

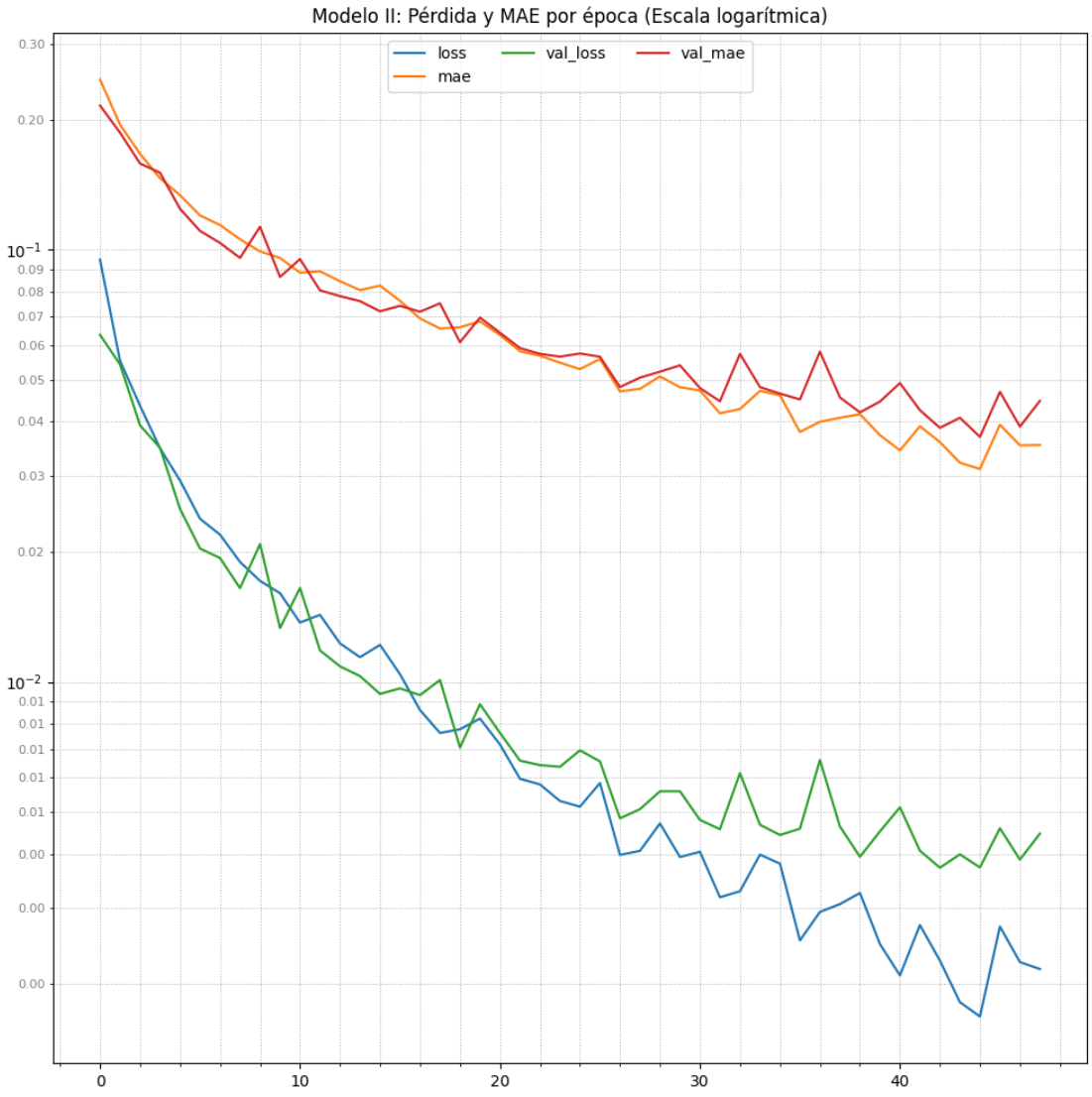


Figura 5.16: Métricas de entrenamiento y validación en fase de selección del Modelo II.

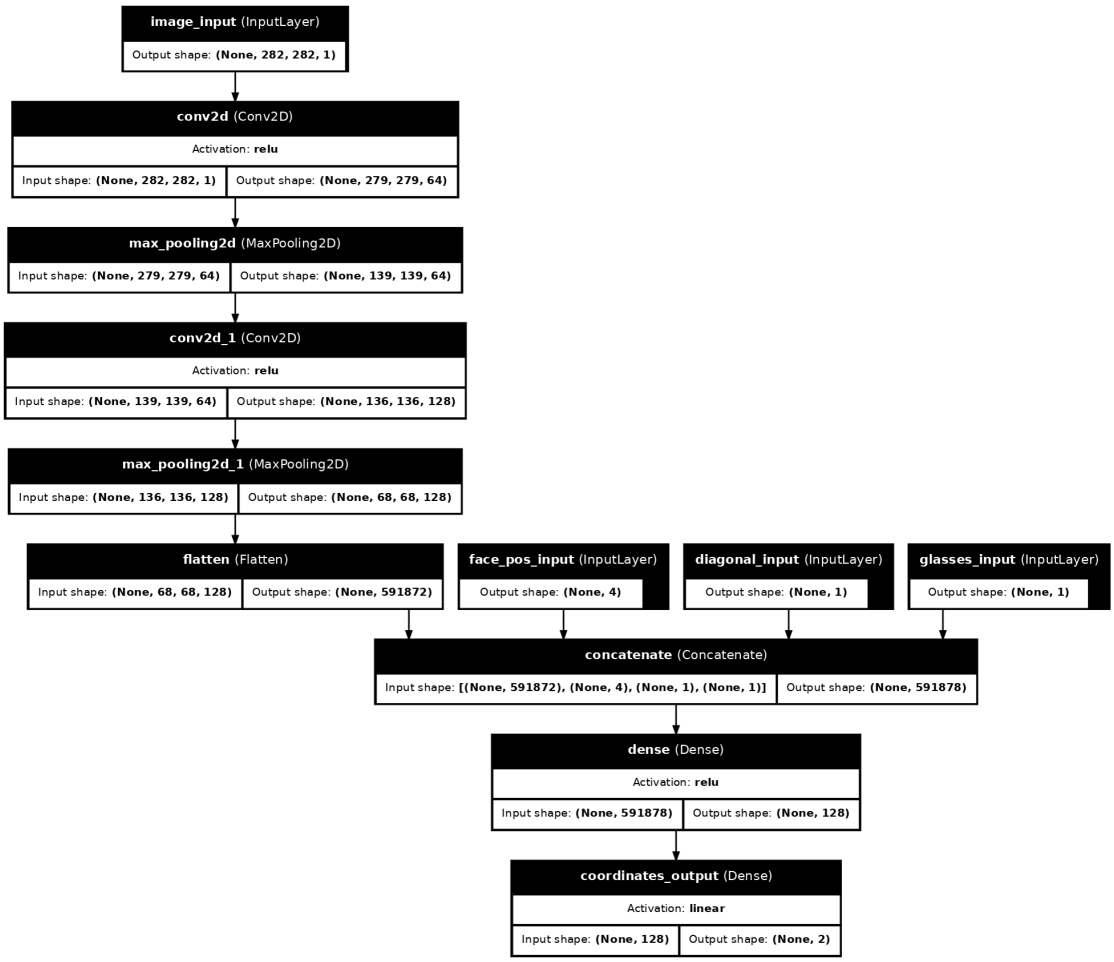


Figura 5.17: Arquitectura del modelo que utiliza Haar Cascade para detección facial en su tubería de preprocesado

de épocas necesario, ya que el entrenamiento con imágenes de este tamaño es costoso en cuanto a memoria y tiempo, lo cuál es un problema dada la cantidad de iteraciones en validación cruzada. En este caso, el entrenamiento concluye a las 30 épocas con paciencia de 5, por lo que se fijará el **número de épocas a 25**.

5.4.3 Modelo III: Aprendizaje por transferencia

Adicionalmente se ha definido otra red más compleja, aprovechando las ventajas de la detección facial y ocular mediante YuNet. En esta ocasión, con la intención de probar si es posible mejorar la exactitud en este caso con un modelo más complejo, se ha definido una arquitectura mucho más densa, que entre sus capas de entrada convolucionales implementa dos instancias de MobileNet en su versión V2, una red convolucional ligera y preentrenada, pensada para realizar extracción de características de imágenes en dispositivos móviles (Es decir, con *hardware* de pocos recursos, donde la eficiencia energética es una prioridad) y con muy baja latencia (Es decir, que es apta para aplicaciones en tiempo real). La práctica de integrar modelos previamente entrenados para resolver problemas no necesariamente idénticos en arquitecturas nuevas se denomina *Transfer Learning* o aprendizaje por transferencia. Los parámetros de estos se pueden bloquear durante los entrenamientos, para abstraer el modelo preentrenado como un bloque transparente con entradas y salidas, logrando así un ajuste que no se puede igualar en redes convolucionales definidas desde cero con conjuntos de entrenamiento demasiado pequeños; y centrando los recursos en ajustar los parámetros de las capas densas inferiores. En este caso, dado que el problema es muy específico, se ha conservado una capa convolucional por debajo de las instancias de MobileNet [51]. Igualmente, dado que las redes complejas son más propensas a problemas de ajuste cuando se entrenan con conjuntos pequeños, se han implementado capas y regularizadores para evitarlos [12]:

- Regularización de kernel: Es una técnica que se implementa en capas densas o convolucionales para añadir un término de penalización a la función de pérdidas, que favorece los pesos pequeños para evitar que crezcan demasiado, que es la causa del sobreajuste. Los tipos más conocidos son L1 y L2. El primero es más agresivo, ya que puede causar que algunos pesos se ajusten a 0. Con el segundo es poco probable que suceda.

- Normalización del lote: Es una capa que normaliza y desplaza las funciones de activación de las entradas lote a lote, aumentando la estabilidad del entrenamiento, permitiendo aumentar la tasa de aprendizaje y adelantando la convergencia del modelo.

- *Dropout*: Es una técnica de regularización cuyas unidades de ejecución se definen en la arquitectura de la red. Estas desactivan las neuronas contiguas con una probabilidad especificada por el desarrollador, reduciendo la dependencia excesiva en ciertas neuronas y forzando a la red a aprender relaciones más robustas, dismi-

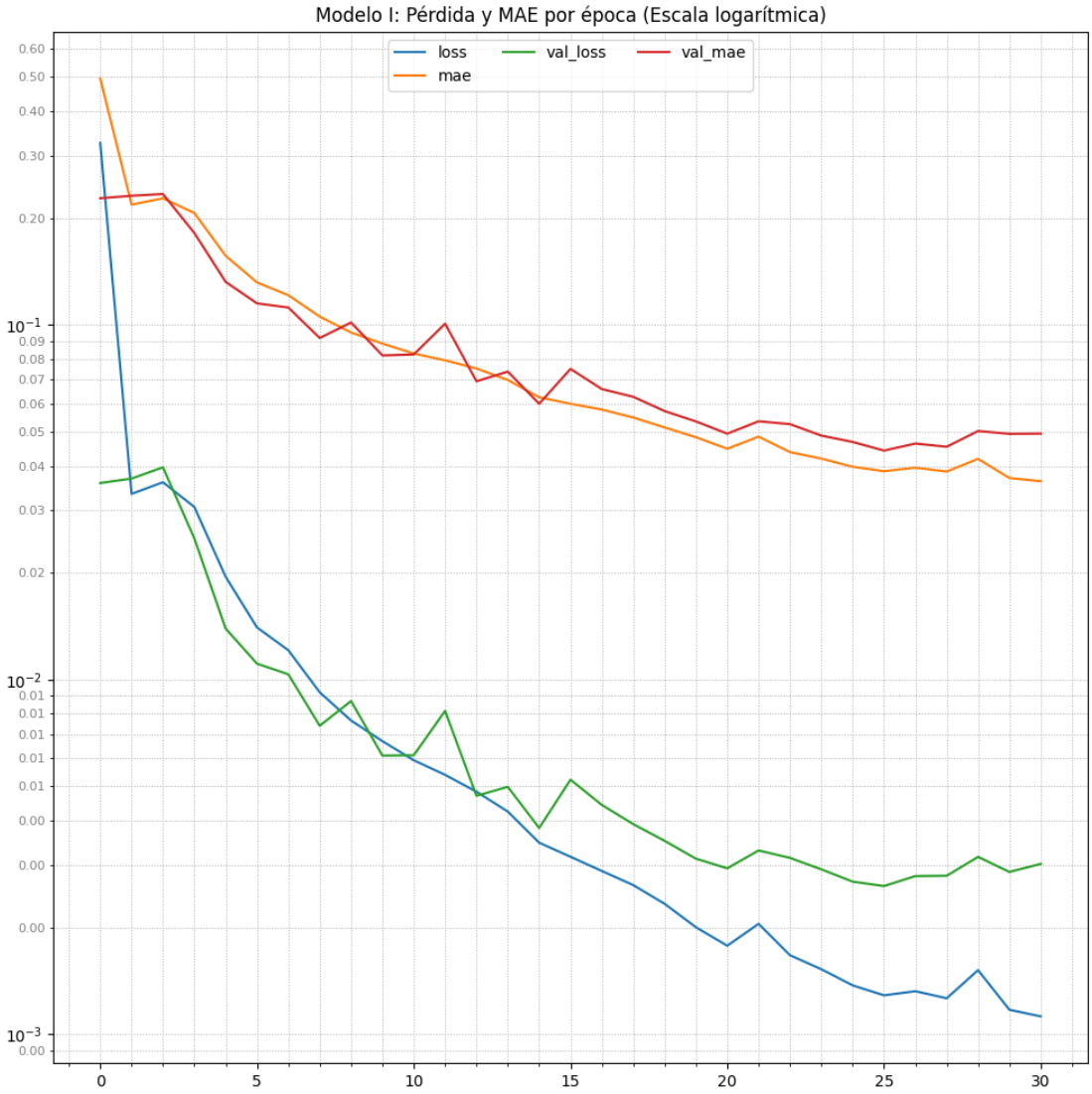


Figura 5.18: Métricas de entrenamiento y validación en fase de selección del Modelo I.

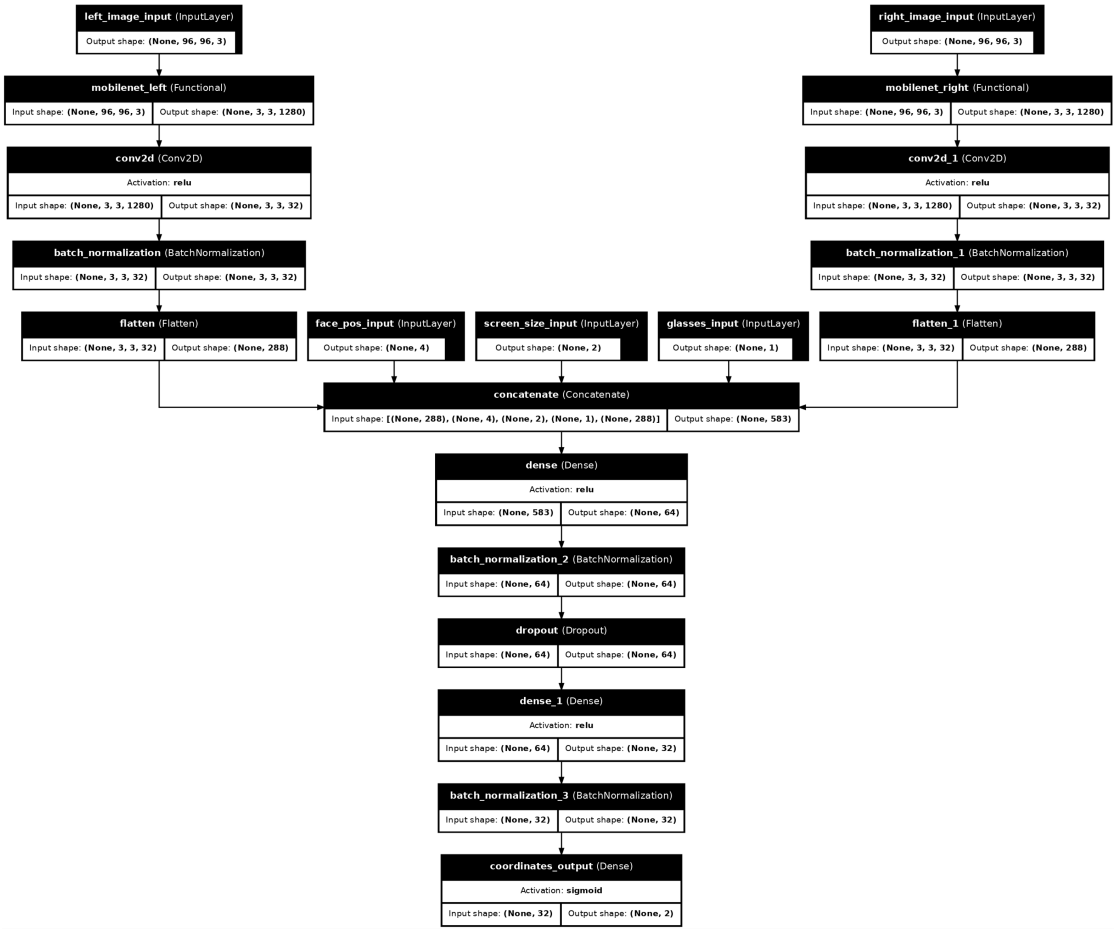


Figura 5.19: Arquitectura del modelo que implementa YuNet en su tubería de preprocesado.

nuyendo en consecuencia el sobreajuste. Solo actúa durante el entrenamiento, a diferencia de las anteriores.

Con esta red se ha implementado el *callback* `ReduceLROnPlateau`, que reduce la tasa de aprendizaje cuando la función de pérdidas (evaluada en validación en fase de selección y evaluada en entrenamiento en fase de validación cruzada) no se reduce en un determinado número de épocas, en este caso 4. Esto aumenta la tolerancia a la hora de elegir un número de épocas, ya que la reducción de tasa de aprendizaje hará que los ajustes sean mínimos cuando el número de épocas sea elevado, impidiendo el sobreajuste, aunque esto causa una disminución insignificante de la función de pérdida hacia el final del entrenamiento que impide que se active el *callback* de finalización adelantada. En cualquier caso, no es un problema, pues esto se debe a una tasa de aprendizaje tan reducida en este punto que el impacto del optimizador es mínimo, por lo que el resultado es similar o ligeramente mejor que con el anterior *callback*, a pesar de llevar más épocas.

La tasa de aprendizaje parte de 0.001 y se reduce notablemente hacia las 20 épocas, alcanzando el efecto comentado pasadas las 30. Finalmente se ha establecido un número de épocas de 25, menor que el de los anteriores modelos, ya que en lugar de las capas convolucionales de 64x64, las instancias de MobileNet no son alteradas durante el entrenamiento, y por tanto el ajuste necesario es menor (ver figura 5.20).

5.5 Evaluación: Métricas, validación cruzada y sujetos conocidos

5.5.1 Métricas

A la hora de escoger las métricas para evaluar el rendimiento de un sistema como este, se buscan parámetros que resulten comprensibles visualmente, que tengan en consideración todos los aspectos relevantes para la evaluación y que permitan comparar su rendimiento con otras soluciones existentes.

Asociar las métricas con píxeles es precisamente el primer paso para poder comparar la efectividad de este sistema con otros, ya que, como se vio en el Capítulo 2; los sistemas de *Eye-Tracking* basados en software suelen utilizar los píxeles de monitores de 24 pulgadas, relación de aspecto 16:9 y resolución *FullHD* (1920x1080 píxeles) para expresar el error que cometen sus sistemas; y los sistemas basados en hardware, grados sexagesimales.

La función de métricas `MAE` de Keras es la aproximación más simple para cuantificar el error. Esta calcula el error medio absoluto entre los valores de dos listas o incluso matrices. Implementa la siguiente fórmula:

Figura 5.20: Métricas de entrenamiento y validación en fase de selección del Modelo III.

Figura 5.21: Tasa de aprendizaje variable en fase de selección del Modelo III.

$$\sum_{i=1}^{D} |y_i - \hat{y}_i| = 0$$

Esta función es idéntica a la función de pérdidas del mismo nombre, también disponible en Keras, pero no se utiliza para el entrenamiento, ya que en problemas de regresión, por la naturaleza cuadrática de la función de pérdidas `MSE` (error cuadrático medio), el optimizador penaliza más agresivamente los errores grandes que a los menores, respecto al supuesto caso de utilizar el error medio absoluto, agilizando la convergencia del modelo y produciendo, casi siempre, mejores resultados [11].

No obstante, para computar la desviación entre los pares de coordenadas inferidas y esperadas de cada eje sí que es útil, puesto que el resultado que devuelve es una lista de las distancias entre cada par de coordenadas. Se recuerda que, al estar normalizadas, las coordenadas de un eje toman valores entre 0 y 1, siendo (0, 0) la coordenada del píxel de la esquina superior izquierda; y (1, 1), la coordenada del píxel de la esquina inferior derecha. Este es, por tanto, un espacio euclídeo en el que las desviaciones de cada uno de los ejes componen una desviación escalar definida por la siguiente fórmula:

$$d_i = \sqrt{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2}$$

En este punto surgen discrepancias con el `MAE` de Keras, ya que los valores de los ejes, por separado, son, efectivamente, la media de las diferencias absolutas. Sin embargo, al invocar el método con las matrices de predicciones y de etiquetas como argumentos, este devuelve la media de ambas medias de las diferencias absolutas de cada eje. Es una aproximación más simple que no considera las circunstancias euclídeas de las coordenadas.

Otra diferencia entre las dos aproximaciones es que la primera, que utiliza la norma euclídea, al evaluar la raíz de la suma de dos diferencias al cuadrado, cuyo valor máximo en ambos casos es 1, tiene un valor máximo de $\sqrt{2}$. En cambio la segunda aproximación, al evaluar la media de los mismos dos valores de máximo 1, el resultado tiene el mismo máximo de 1, siendo necesario dividir el resultado de aplicar la fórmula de la norma euclídea entre $\sqrt{2}$ para obtener un valor normalizado. En el Capítulo 6, se puede ver cómo las métricas en cuya definición se utiliza la norma euclídea para vectores bidimensionales, suele devolver valores de error ligeramente más altos que la implementación del error absoluto medio de Keras, ya que la norma euclídea produce este efecto cuando la desviación se debe a las componentes de ambos ejes en proporciones similares.

Se van a definir entonces las métricas correspondientes [4]:

[4]La desviación estándar de cualquiera de estas no es una métrica como tal, pero sirve de indicador de la confiabilidad de la métrica. Ver correlación entre ambas en 6.5, 6.6 y 6.7.

- Desviación absoluta media en el eje X (`XAbsDevMean`):

$$X_{AbsDevMean} = \frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} |(x_i - \hat{x_i})|$$

- Desviación absoluta media en el eje Y (`YAbsDevMean`):

$$Y_{AbsDevMean} = \frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} |(y_i - \hat{y_i})|$$

- Desviación absoluta media euclídea normalizada (`XYNormEucldAbsDevMean`):

$$XY_{NormAbsDevMean} = \frac{1}{U.S.C.} \frac{1}{\sqrt{2}} \sum_{i=1}^{U.S.C.} \sqrt{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2}$$

Estas métricas tienen un valor evidente, ya con ellas es factible cuantificar visualmente el error relativo o incluso expresarlo como porcentaje, además de por supuesto, compararlas con otros sistemas que utilicen métricas similares. Sin embargo, no son del todo adecuadas para evaluar la precisión [52]. Coloquialmente, se suele utilizar el término de "precisión" para hacer referencia a la magnitud inversa al error cuando se habla de modelos de *Machine Learning*, ya que con ella se puede expresar la corrección de las predicciones. Sin embargo esto no es del todo correcto. En ingeniería y estadística, "precisión" y "exactitud" no son términos equivalentes, aunque existe consenso sobre la diferencia entre ambos, hasta el punto en el que una norma ISO las define. Concretamente la ISO 5725 de 1994 [53], según la cual, el término general correcto es "Exactitud", que se traduce por "Accuracy" y se compone otras dos propiedades:

- Veracidad: Es la proximidad del promedio de una serie de mediciones al valor verdadero. Se evalúa a través del sesgo de estimación (*bias*). Si bien la organización desaconseja el uso de este término por tener otras connotaciones en otros campos, en *Deep Learning* es perfectamente válido, ya que se puede relacionar con una desviación constante en las predicciones. Un aspecto clave de una posible implementación del sistema, que forma parte de una de las líneas futuras de este proyecto. Como cabe esperar por la definición, la exactitud del modelo se mediría mediante la media de las diferencias (no absolutas) entre las predicciones y las etiquetas del conjunto de test, pudiendo evaluar el valor absoluto de esta media para mantener la normalización entre 0 y 1 de todas las métricas, contemplada hasta este momento.
- Precisión: Es el grado de dispersión entre una serie de mediciones repetidas bajo condiciones especificadas. Dentro de este se distinguen dos componentes: Repetibilidad (En *Eye-Tracking*, mismo sujeto y mismas condiciones) y reproducibilidad (En *Eye-Tracking*, diferentes sujetos o condiciones variadas). Una medida común

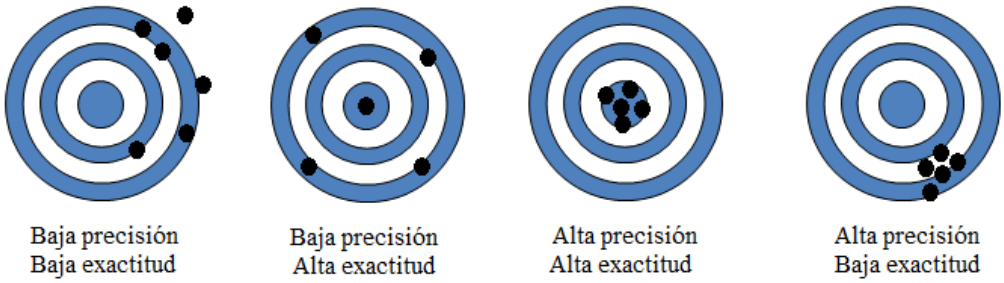


Figura 5.22: Ilustración de precisión y exactitud, donde el centro de la diana representa el lugar esperado y los puntos corresponden a las predicciones. Tomado de [54]

de dicha dispersión es la desviación estándar del conjunto de mediciones. En el caso del proyecto, la componente de reproducibilidad se puede expresar mediante la desviación estándar de la diferencia entre las predicciones y las etiquetas del conjunto de test, ya que las coordenadas son aleatorias en cualquier caso, y los sujetos pueden ser varios si el mecanismo de evaluación toma muestras de más de uno de ellos para conformar el conjunto de test. Sin embargo, no es posible evaluar la repetibilidad, ya que para hacerlo, habría que evaluar la desviación estándar de las predicciones inferidas por el modelo en varias muestras en las que un mismo sujeto mire al mismo punto. Es decir, que las etiquetas contengan una coordenada idéntica; y eso no se puede conseguir en este caso, debido al reducido tamaño del conjunto de datos y la aleatoriedad de sus muestras. De intentarlo, sería poco probable encontrar grupos de varias muestras de coordenadas similares dentro de cada sujeto. Entre las soluciones de *Eye-Tracking* sin hardware específico existentes en el mercado, los desarrolladores de los únicos dos sistemas de los que se han publicado métricas oficiales encontraron también este problema. En el caso de WebGazer.js, se limitaron a incluir la exactitud en su artículo [30]. En cambio, en el artículo de Real-Eye, caracterizaron la precisión midiendo únicamente la desviación estándar de una muestra por sujeto y promediando el resultado de todos ellos, dejando claro que en su caso, por este motivo, la medida de precisión es menos crítica que la exactitud al no haber evaluado la varianza en un mismo participante [29]. En el proyecto actual se va a hacer esto mismo, pero incluyendo en el promedio de desviaciones estándar varias muestras por sujeto con coordenadas aleatorias, siendo un caso similar, algo más próximo a lo ideal sin llegar a serlo.

Cabe destacar que una gran parte de los documentos disponibles sobre esta temática no suelen expresar métricas en función de la veracidad, y por lo general, tampoco el término de “veracidad” propuesto por la ISO. En su lugar, suelen expresar la exactitud y la precisión tal y como se han definido en este Trabajo, haciendo referencia a las mismas con los términos de “accuracy” y “precision” respectivamente. Dado que la exactitud se compone de la veracidad y la precisión, no es estrictamente necesario expresar las tres para caracterizar correctamente el rendimiento del sistema [29, 30].

A continuación se definen las métricas en función de las cuales se expresa la veracidad:

- Desviación media en X (`XDevMean`):

$$X_{DevMean} = \frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} (x_i - \hat{x_i})$$

- Desviación media en Y (`YDevMean`):

$$Y_{DevMean} = \frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} (y_i - \hat{y_i})$$

- Desviación euclídea media normalizada (`XYNormEucldDevMean`):

$$XY_{NormEucldDevMean} = \frac{1}{2\sqrt{2}} \sum_{i=1}^{U.S.C.} \sqrt{X_{DevMean}^2 + Y_{DevMean}^2}$$

- Desviación promediada media normalizada (`XYDevMean`):

$$XY_{DevMean} = \frac{1}{2\sqrt{2}} \sum_{i=1}^{U.S.C.} \sqrt{X_{DevMean}^2 + Y_{DevMean}^2}$$

Y el resto, en función de las cuales se expresa la precisión:

- Desviación típica en X (`XDevStd`):

$$X_{DevStd} = \sqrt{\frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} ((x_i - \hat{x_i}) - X_{DevMean})^2}$$

- Desviación típica en Y (`YDevStd`):

$$Y_{DevStd} = \sqrt{\frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} ((y_i - \hat{y_i}) - Y_{DevMean})^2}$$

- Desviación euclídea típica normalizada (`XYNormEucldDevStd`):

$$XY_{NormEucldDevStd} = \frac{1}{2\sqrt{2}} \sum_{i=1}^{U.S.C.} \sqrt{X_{DevStd}^2 + Y_{DevStd}^2}$$

- Desviación promediada típica normalizada (`XYDevStd`):

$$XY_{DevStd} = \frac{1}{2}(X^2_{DevStd} + Y^2_{DevStd})$$

Cabe destacar que en este caso se han desarrollado las fórmulas para calcular las desviaciones típicas de las métricas basadas en media porque, a diferencia de lo que ocurría con las expresiones basadas en el error medio absoluto, las desviaciones estándar no son un indicador de la confiabilidad de la media a la que acompañan, sino una expresión en función de la precisión, mientras que las expresiones de la media representan la veracidad, en términos de la norma ISO 5725 [53]. Como se ha explicado antes, las medias, en este caso, al ser diferencias de dos valores acotados entre 0 y 1, quedan acotadas entre -1 y 1. Por ello, para mostrarse en un formato coherente al resto de métricas, se puede calcular el valor absoluto para normalizarlas, perdiendo el contexto de la dirección del sesgo que miden, que puede resultar imprescindible en el supuesto caso de desarrollar un sistema de calibración como línea futura de este proyecto, pero que actualmente no tiene mayor importancia.

Asimismo es necesario considerar que estas métricas no cuantifican la veracidad y la precisión como tal, sino el sesgo de estimación y la dispersión, que son inversas a las primeras respectivamente, pero esto permite expresarlas como funciones. Es decir, a menor sesgo de estimación, mayor veracidad; y a menor dispersión, mayor precisión y viceversa.

En este análisis se han incluido también ciertos parámetros recogidos del preprocesado y de la tabla del conjunto de datos del proyecto, que no tienen relevancia para el proceso de entrenamiento, pero que pueden servir para justificar ciertos aspectos y comprender limitaciones del sistema y del conjunto de datos. Para ello, finalmente se mostrará un mapa de correlación entre las métricas definidas en la última sección del Capítulo anterior, y las variables mencionadas, que son las siguientes:

- Uso de gafas (`WearingGlasses`): Esta variable corresponde a un valor de entrada de las redes neuronales. Ante sujetos con gafas, debido al efecto de las lentes y a los posibles reflejos, los modelos son más suceptibles de cometer errores. Es preciso analizar si esto se cumple.
- Tasa de fallo de detección (`DetectionFailRate`): Contiene un valor normalizado que muestra la proporción de errores del clasificador en relación al número de muestras totales del sujeto.
- Píxeles de cámara (`CameraMPix`): Se recuperan los valores de resolución de las cámaras del sujeto, para valorar el impacto de la cámara sobre la exactitud en términos generales del modelo.
- Tamaño del conjunto de entrenamiento (`TrainingSetSize`): Se almacena el tamaño del conjunto de entrenamiento en cada iteración del bucle de validación

cruzada, resultante de restar el tamaño del número de muestras del sujeto con el que se evalúa al número total de muestras del conjunto.

- Tamaño de la diagonal de la pantalla en pulgadas (`ScreenSize`): Se recupera este valor del conjunto de datos para mostrar su impacto en la exactitud en términos generales.
- `MAE` del modelo según el *framework* durante la evaluación (`TestOverallAcc`): Se recupera del historial almacenado por el *framework* para comprender la relación de esta métrica, que es abstracta y genérica, con el resto. También está normalizada entre 0 y 1.

5.5.2 Evaluación I: Caso general con sujetos no conocidos. Validación cruzada

Ante la limitación del reducido tamaño del conjunto de datos de este proyecto, que se recuerda que es de 1380 muestras, el procedimiento común de separar una pequeña muestra del conjunto de datos para conformar el conjunto de test, no sirve para hacer una evaluación justa, entendiendo como tal recopilar datos sobre el comportamiento del modelo ante muestras de sujetos cuyas muestras restantes no aparezcan en el conjunto de entrenamiento [46]:

- Las muestras provienen de una población demasiado pequeña inevitablemente, ya que hacerla más grande supone una disminución perjudicial del conjunto de entrenamiento. En el caso del proyecto, uno o dos sujetos como mucho.
- El rendimiento del modelo en este conjunto no es representativo de un caso de uso genérico. Puede ocurrir que con el sujeto o los sujetos de la muestra funcione mejor o peor de lo habitual.
- Actualmente, la separación de conjuntos se hace en base a los sujetos. Tomar una muestra aleatoria, sin tener en cuenta los sujetos pueden solucionar el problema de la poca generalidad del conjunto de evaluación, pero produciría un sesgo en la evaluación, ya que compartirían sujetos, y habría registros no iguales pero muy similares, que en caso de sobreentrenamiento mostrarían una precisión muy elevada a pesar de que el modelo esté memorizando el set de entrenamiento y sea incapaz de generalizar.

Durante el desarrollo ha utilizado el método habitual de entrenamiento, validación y test como referencia, eligiendo con cierto criterio los sujetos para el set de evaluación. Esto permite un rigor suficiente como para controlar el proceso de entrenamiento y detectar subajuste o sobreajuste en caso de que se produzca, pero no es aceptable como para plasmar los resultados en un artículo o en la memoria de un Trabajo de Fin de Grado,

hay que recurrir a métodos más rigurosos en estas circunstancias, como en este caso *Leave-One-Out Cross-Validation (LOOCV)*:

La técnica consiste en dividir el conjunto de datos en múltiples porciones. Una parte grande de ellas se dedican al conjunto de entrenamiento y, el remanente, al conjunto de evaluación o test. Esto se debe repetir varias veces, entrenando, evaluando y variando la repartición de las porciones en cada una. Al terminar, se calcula el promedio de las métricas de todas las iteraciones. El resultado es una estimación muy robusta y realista del rendimiento del modelo.

En este caso, la segmentación en porciones es muy evidente: Mediante sujetos, tal y como se ha hecho anteriormente. El procedimiento llevado a cabo entrena el modelo en bucle tantas veces como sujetos haya en el conjunto de datos. En cada iteración se forma el conjunto de test con las muestras de un sujeto diferente, y el conjunto de entrenamiento con el resto. No es preciso hacer validación en esta prueba, puesto que los modelos ya han sido definidos y seleccionados en su desarrollo mediante los mecanismos tradicionales de entrenamiento, validación y test.

En las figuras 5.23[5], 5.24 y 5.25 se muestran las gráficas correspondientes a la superposición de las métricas de entrenamiento en la prueba de validación cruzada resultantes de entrenar con los hiperparámetros del método `fit()`

5.5.3 Evaluación II: Caso específico con sujetos conocidos

El procedimiento desarrollado en la anterior subsección caracteriza el comportamiento de los modelos cuando se someten a nuevos sujetos cuyas muestras no han formado parte del conjunto de entrenamiento. Si bien es cierto que no es viable hacer una evaluación rigurosa para este caso de uso mediante una separación en conjuntos de entrenamiento y evaluación tradicional, si es útil para analizar el rendimiento del sistema con sujetos que sí ha "visto" durante el entrenamiento. Aunque lo ideal sería lograr una red neuronal con una excelente capacidad de generalización, que no necesite conocer a los sujetos con los que se va a probar; la realidad es que lo más habitual es incorporar datos del usuario al sistema para mejorar su exactitud, en lugar de generalizar.

Volviendo a las soluciones de *Eye-Tracking* sin hardware específico existentes, ninguna de las cuatro citadas en el Capítulo 2 se apoya exclusivamente en la buena generalización de sus modelos para lograr sus resultados. Las cuatro implementan sistemas de calibración durante su inicialización, que adquieren datos sobre el usuario que incorporan a sus cálculos durante la sesión. En adición a esto, WebGazer.js, durante su funcionamiento utiliza las pulsaciones del ratón para establecer puntos de referencia, recopilando el fotograma de la cámara y las coordenadas del cursor en la pantalla al momento de

[5]El patrón extraño que se observa en las primeras épocas del gráfico se debe a la inicialización aleatoria de los parámetros de la red, ya que cuando se asigna memoria para el modelo, los parámetros toman el valor que hubiera en las posiciones que ocupan. Este efecto se observa con mayor probabilidad en redes con entradas grandes,como es el caso.

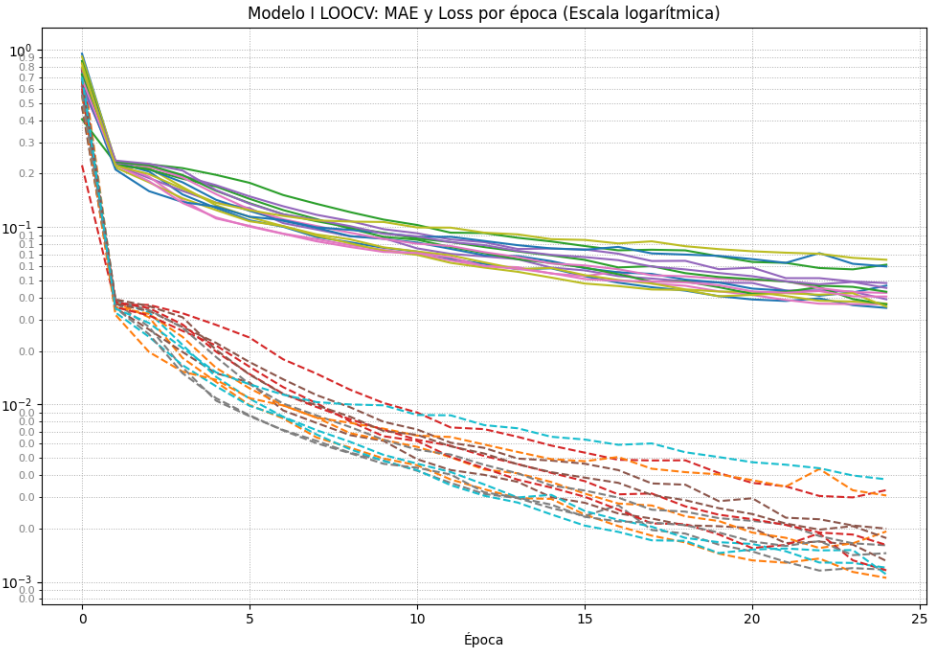


Figura 5.23: Métricas de entrenamiento en validación cruzada del Modelo I. Líneas discontinuas: Función de pérdida; líneas continuas: `MAE`

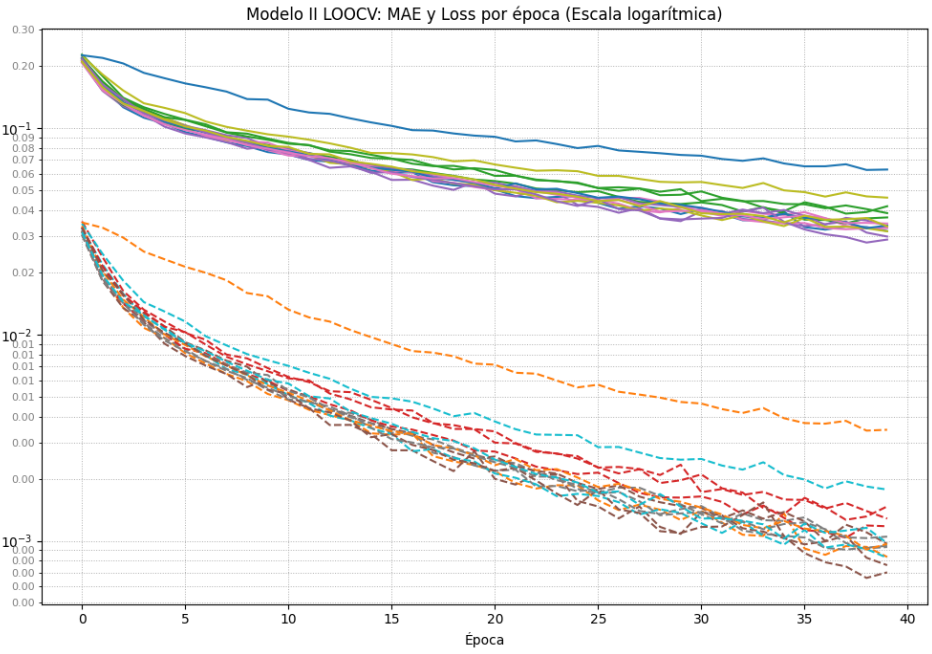


Figura 5.24: Métricas de entrenamiento en validación cruzada del Modelo II. Líneas discontinuas: Función de pérdida; líneas continuas: `MAE`

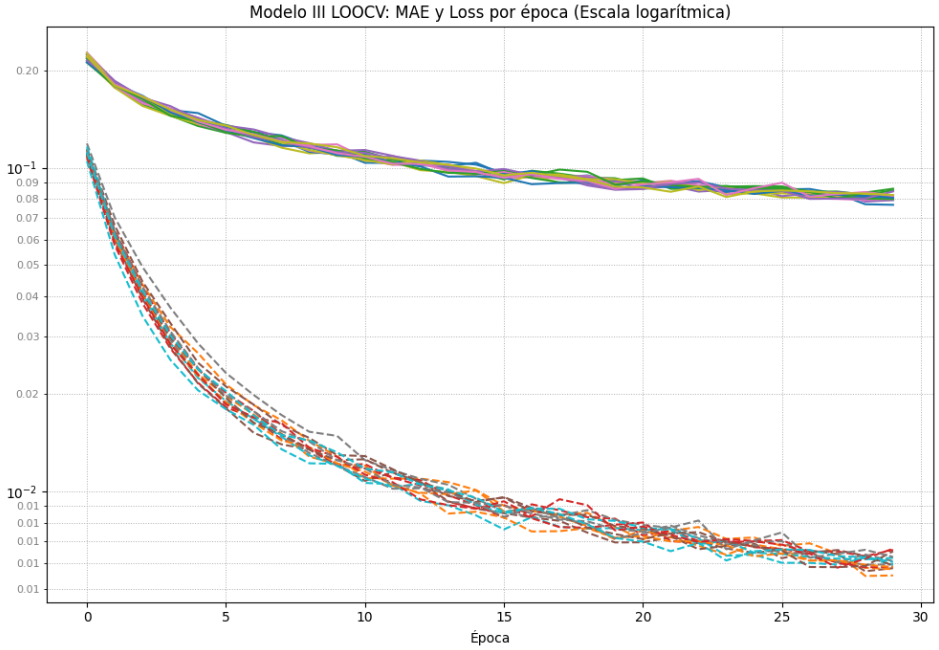


Figura 5.25: Métricas de entrenamiento en validación cruzada del Modelo III. Líneas discontinuas: Función de pérdida; líneas continuas: `MAE`

hacer clic, para recalibrar el sistema y reducir el sesgo de estimación sobre la marcha [30]. Por su parte, la solución de Real-Eye, si el usuario se desplaza significativamente, muestra unas guías en pantalla para ayudarlo a recuperar su posición original (Aquella que el sistema capturó durante su calibración inicial) para evitar que las muestras difieran como causa del movimiento [29, 31].

En el proyecto, no es posible implementar un sistema de calibración de estas características, al no integrar una interfaz gráfica. Sin embargo, si la tuviera, probablemente sería muy similar al Experimento 1 de la aplicación web de adquisición, que a su vez se asemeja a las interfaces que implementan los sistemas de calibración de las soluciones enumeradas en el Capítulo 2. Este recopila entre 15 y 20 muestras en un minuto, las cuales podrían utilizarse para detectar y corregir el sesgo de estimación, que causa una pérdida de veracidad, o para reentrenar el modelo e incorporarse al mismo.

Esto último es precisamente lo que se va a emular con este segundo mecanismo de evaluación, consistente en una separación aleatoria de en torno a un 15 % de las muestras del conjunto de datos para evaluación, de la forma que se ha explicado en la sección 5.3 de este Capítulo, tomando aleatoriamente muestras del conjunto bajo la única condición de que se hayan adquirido mediante el Experimento 1, ya que las coordenadas de estas son aleatorias e incorreladas entre sí, mientras que las coordenadas del Experimento 2 describen un patrón definido por cuatro posibles funciones continuas, lo que causa que dos muestras contiguas contengan coordenadas no iguales pero sí muy próximas.

Nombre	**Descripción**	**Entornos compatibles**	**Tiempo real en CPU**	**Tiempo real en GPU**
FaceBoxes [38]	Modelo ligero diseñado para detección de rostros en tiempo real.	Python	Inviable	Viable
BlaceFace [39]	Detector de rostros ligero y alto rendimiento, optimizado para GPUs de dispositivos móviles.	Python (TF Lite) / Qualcomm AI Hub	N/A	Viable
Libfacedetection [40]	Biblioteca de detección facial de alto rendimiento escrita en C++ basada en CNN.	C++	Viable	N/A
Ultra-Light-Fast-Generic-Face-Detector-1MB [41]	Modelo de tamaño muy reducido basado en OpenCV DNN diseñado para *Edge Computing* en dispositivos con recursos limitados.	Python	Viable	N/A
YuNet [42]	Modelo ligero basado en CNN con solo 75,856 parámetros, adecuado para dispositivos con recursos limitados.	Python, C++	Viable	N/A
MTCNN [43]	Modelo de muy alta precisión basado en CNN que utiliza una red neuronal en cascada para la detección de rostros y puntos clave faciales.	Python	Inviable	Viable
Haar Cascade [32]	Algoritmo clásico de detección de objetos que utiliza características de Haar. Adecuado para dispositivos de recursos limitados. Menos preciso que implementaciones modernas.	Python/C++	Viable	N/A
LBP Cascade [44]	Algoritmo similar al Haar Cascade, aún más ligero y adecuado para dispositivos embebidos al usar internamente enteros en vez de decimales.	Python/C++	Viable	N/A

Tabla 5.1: Resumen comparativo de modelos de detección facial en términos de compatibilidad y viabilidad para tiempo real.

Método	**Descripción**	**Uso**
`INTER_NEAREST`	Interpolación *nearest-neighbor* (la más rápida, pero de baja calidad)	Cambio de tamaño simple y rápido (por ejemplo, pixel art, imágenes binarias)
`INTER_LINEAR`	Interpolación bilineal (método predeterminado)	Redimensionamiento de uso general (buen equilibrio entre velocidad y calidad)
`INTER_CUBIC`	Interpolación bicúbica (utiliza una vecindad de 4×4 píxeles)	Escalado de alta calidad, resultados más suaves
`INTER_AREA`	Remuestreo mediante la relación de área de píxeles	Mejor opción para reducir imágenes (evita el solapamiento)
`INTER_LANCZOS4`	Interpolación de Lanczos mediante una vecindad de 8×8 píxeles	Aumento y reducción de resolución de alta calidad (conserva los detalles finos)

Tabla 5.2: Interpoladores proporcionados por OpenCV. Tomado de [48]

	faces_w	faces_h	eyes_w	eyes_h	eyes aspectratio	norm eyes x1	norm eyes x2	norm eyes y1	norm eyes y2	norm eyes w	norm eyes h
mean	329.021	329.021	170.617	59.738	2.884	0.225	0.743	0.303	0.484	0.519	0.181
std	64.742	64.742	34.510	13.804	0.280	0.019	0.021	0.024	0.013	0.029	0.018
min	238.000	238.000	97.000	34.000	2.205	0.149	0.607	0.212	0.426	0.394	0.125
25 %	279.000	279.000	149.000	50.000	2.707	0.213	0.734	0.292	0.482	0.500	0.170
50 %	315.000	315.000	162.000	56.000	2.885	0.225	0.744	0.307	0.487	0.520	0.181
75 %	356.000	356.000	186.750	65.000	3.030	0.236	0.753	0.319	0.492	0.536	0.190
max	597.000	597.000	286.000	110.000	5.208	0.310	0.942	0.358	0.500	0.731	0.249

Tabla 5.3: Estadísticas descriptivas para el ancho y alto del área ocular tomadas del Apéndice A.

Capítulo 6

Análisis de resultados

6.1 Presentación

Se muestran a continuación los resultados obtenidos mediante el mecanismo de evaluación por validación cruzada, para cada uno de los tres modelos, entrenados con diferentes fracciones del conjunto de datos. En las figuras 6.1 y 6.2 se recogen los valores de las tres métricas basadas en la desviación euclídea normalizada, en función de las cuales se expresa la exactitud, veracidad y precisión. Este énfasis se debe a que son dichas métricas las que mejor caracterizan –de forma global– el error que cometen los modelos, al componerse de sendos ejes X e Y, sin obviar la cualidad espacial de las coordenadas.

Se ha analizado la correlación de la totalidad de las métricas tomadas en el proceso con los datos estadísticos del conjunto de datos que se mostraron en el Capítulo 3 para visualizar la implicación de ciertos factores en el resultado final de cada modelo en las figuras 6.5, 6.6 y 6.7.

También se muestran las las tres métricas basadas en la desviación euclídea normalizada obtenidas mediante el mecanismo de evaluación para el caso específico en el que los sujetos del conjunto de test aparecen también en el conjunto de entrenamiento, en la figura 6.3 y la comparación de todas las métricas mostradas anteriormente, en gráficos de barras con la misma escala, en la figura 6.4.

Por último, se han recogido todos los valores numéricos de todas las mediciones realizadas –incluyendo aquellos a partir de los cuales se han generado las figuras anteriores– en las tablas 6.1, 6.2 y 6.3 para el Modelo I; en las tablas 6.4, 6.5 y 6.6 para el Modelo II; y en las tablas 6.7, 6.8 y 6.9 para el Modelo III.

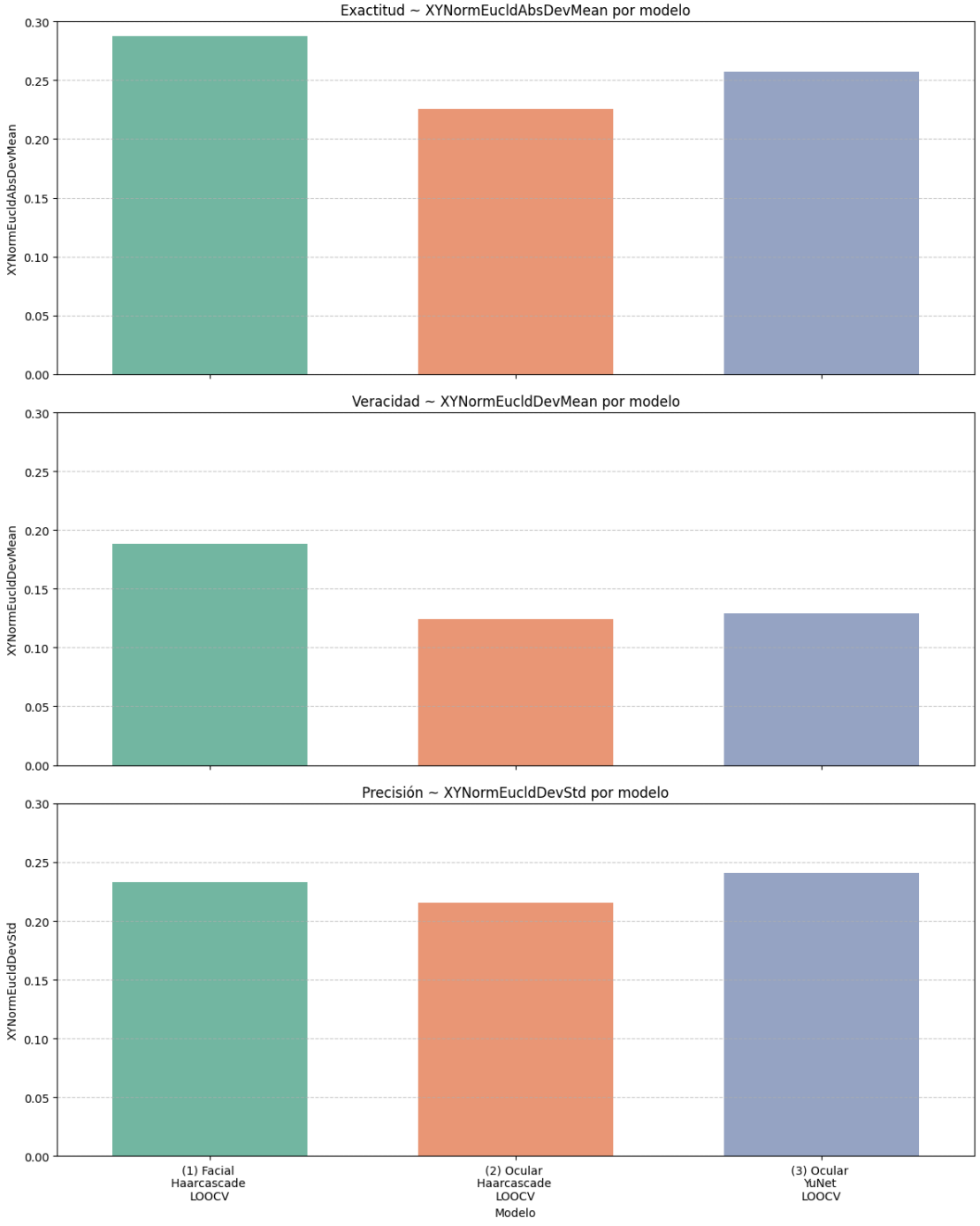


Figura 6.1: Métricas de exactitud, veracidad y precisión de la evaluación de validación cruzada para los tres modelos

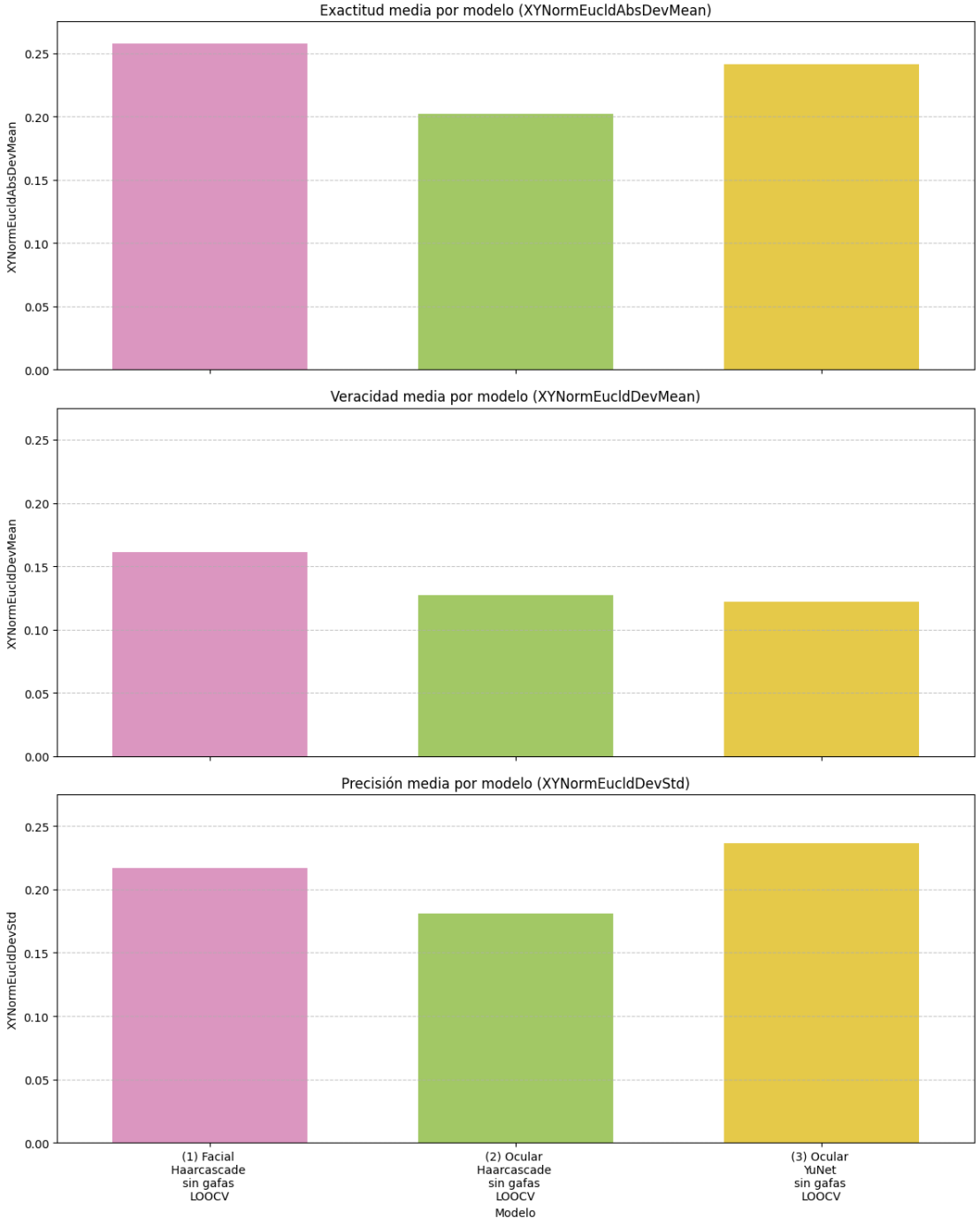


Figura 6.2: Métricas de exactitud, veracidad y precisión de la evaluación de validación cruzada para los tres modelos omitiendo sujetos con gafas

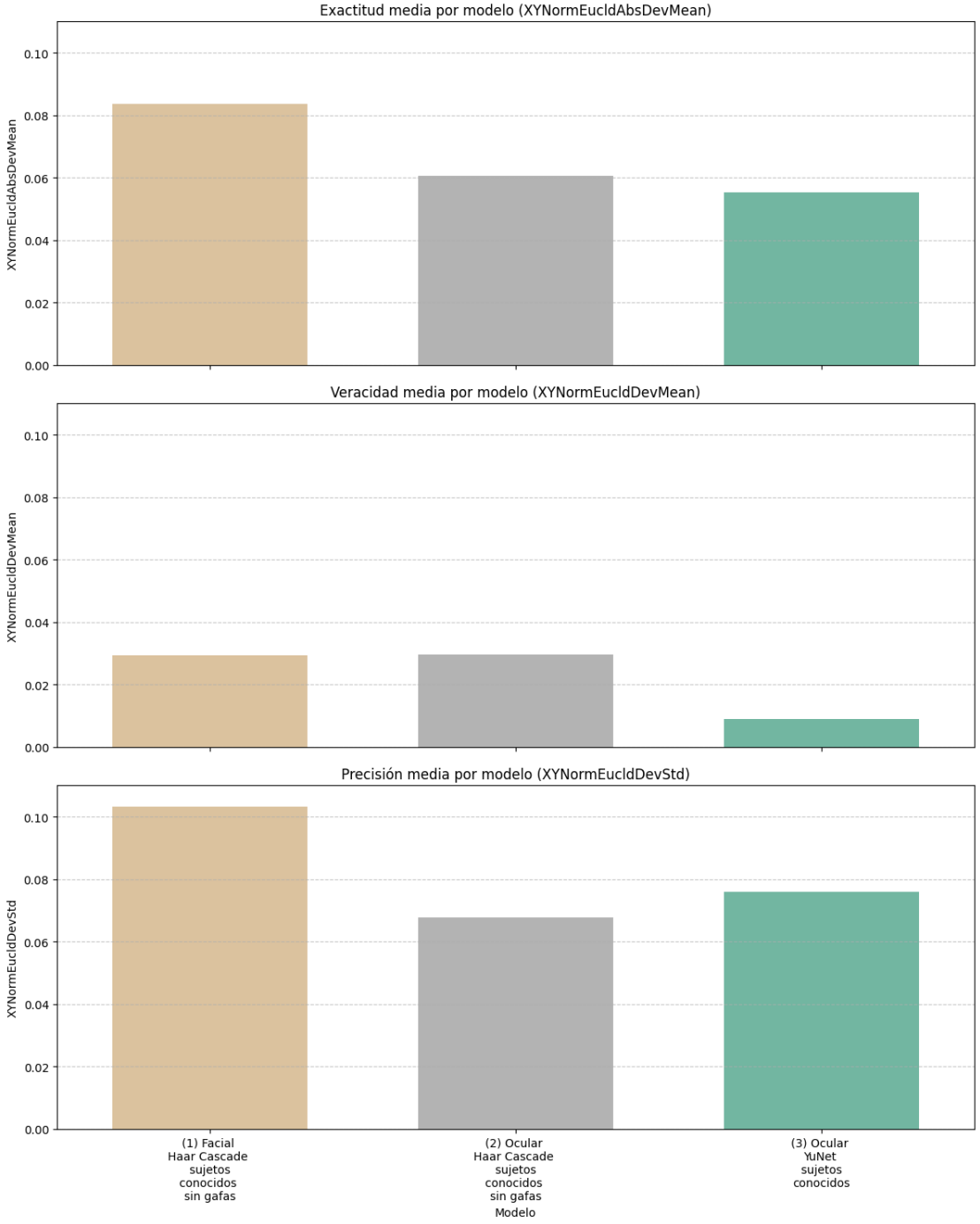


Figura 6.3: Métricas de exactitud, veracidad y precisión de la prueba de evaluación por sujetos conocidos para los tres modelos con el conjunto mejor resultante en las pruebas anteriores.

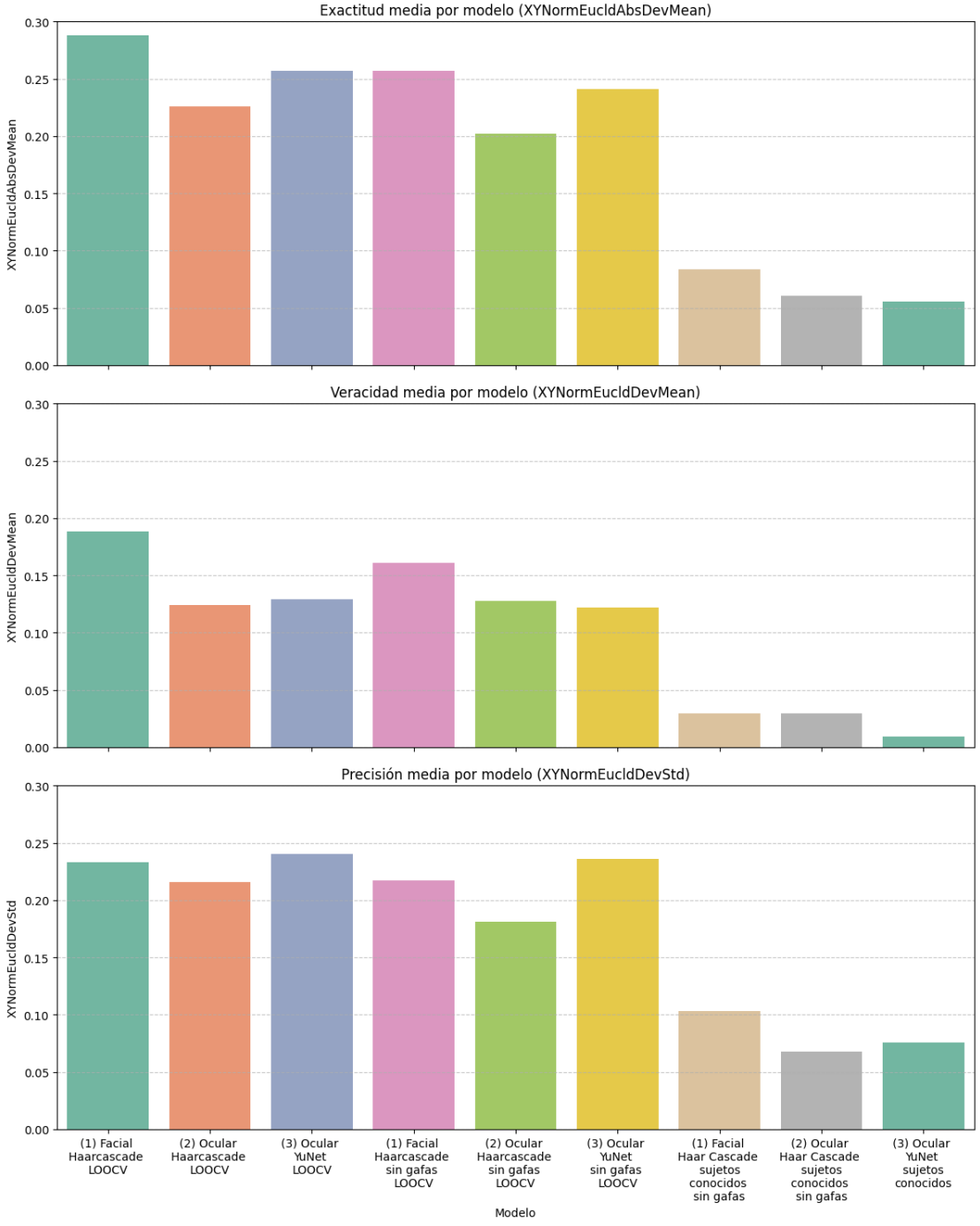


Figura 6.4: Métricas de exactitud, veracidad y precisión para los tres modelos en todas las pruebas anteriores

6.1.1 Modelo I: Desglose de métricas

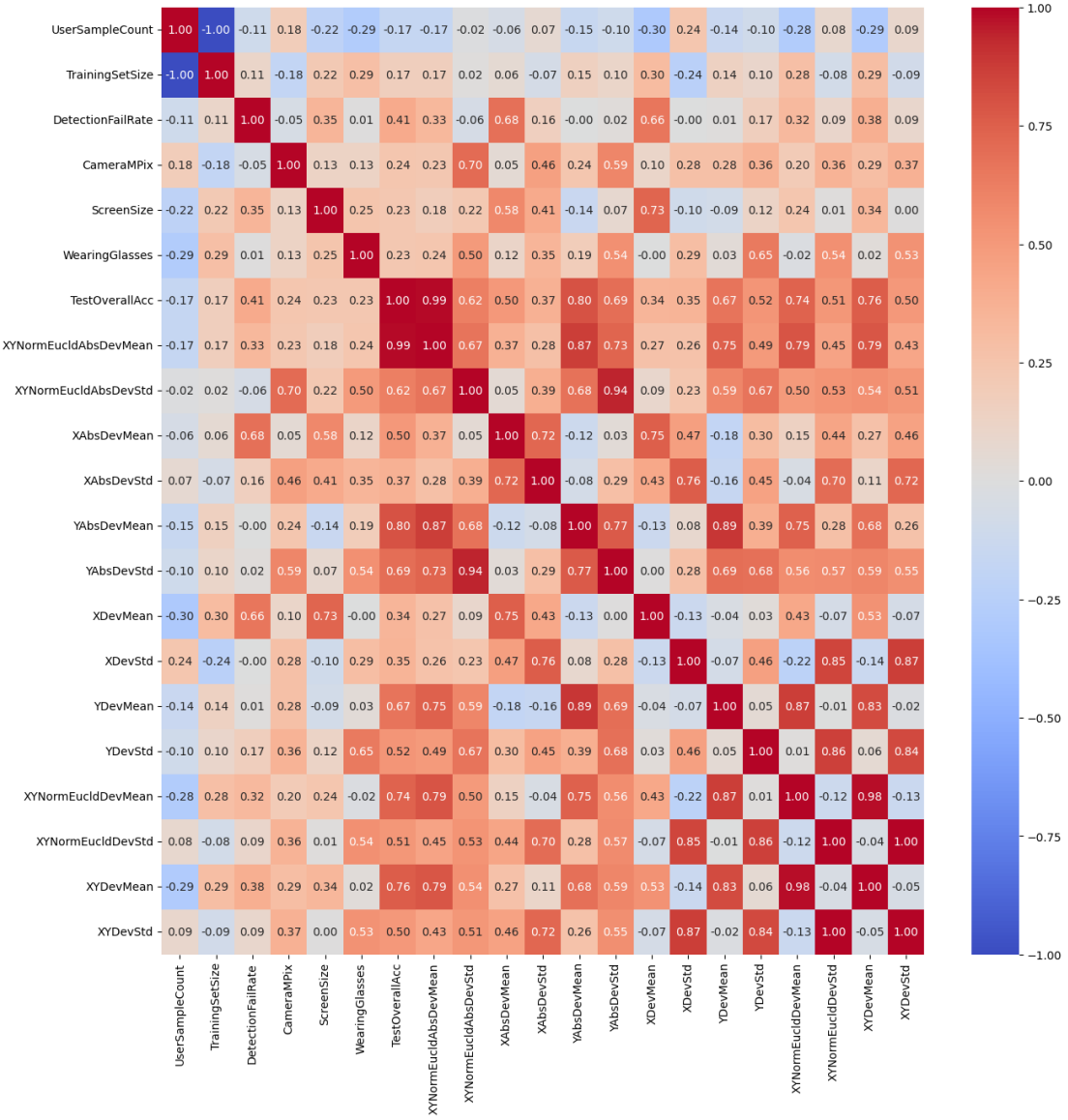


Figura 6.5: Mapa de correlación por pares de las métricas del Modelo I con estadísticas del conjunto de datos tomadas de la sección 3.4 del Capítulo 3.

	User Code	User Sample Count	Training Set Size	Detection Fail Rate	Camera MPix	Screen Size	Wearing Glasses	Test Overall Acc	XYNorm Eucld AbsDev Mean	XYNorm Eucld AbsDev Std	XAbs Dev Mean	XAbs Dev Std	YAbs Dev Mean	YAbs Dev Std	X Dev Mean	X Dev Std	Y Dev Mean	Y Dev Std	XYNorm Eucld Dev Mean	XYNorm Eucld Dev Std	XY Dev Mean	XY Dev Std
0	QFWJ	338	2140	0.191	2457600	14.200	No	0.338	0.376	0.176	0.254	0.170	0.423	0.271	0.146	0.268	0.388	0.319	0.293	0.295	0.267	0.294
1	MPVE	772	1706	0.020	921600	16.200	No	0.199	0.220	0.106	0.220	0.150	0.178	0.128	0.012	0.267	0.038	0.216	0.028	0.243	0.025	0.242
2	LIDJ	104	2374	0.037	921600	16.200	No	0.195	0.218	0.099	0.210	0.152	0.181	0.124	0.124	0.228	0.152	0.159	0.139	0.196	0.138	0.193
3	QXYT	154	2324	No	921600	15.600	No	0.316	0.394	0.140	0.087	0.066	0.545	0.201	0.007	0.109	0.545	0.201	0.385	0.161	0.276	0.155
4	JUFE	58	2420	0.216	921600	16.200	Sí	0.335	0.373	0.126	0.287	0.170	0.383	0.232	0.112	0.316	0.333	0.300	0.248	0.308	0.222	0.308
5	HGNI	108	2370	No	921600	14	No	0.168	0.189	0.086	0.189	0.137	0.146	0.103	0.172	0.158	0.015	0.179	0.122	0.169	0.093	0.168
6	OOWZ	66	2412	No	921600	15.600	No	0.168	0.190	0.106	0.101	0.073	0.235	0.156	0.009	0.125	0.088	0.269	0.063	0.210	0.049	0.197
7	ROOT	222	2256	0.009	921600	16.200	No	0.186	0.208	0.106	0.182	0.128	0.190	0.153	0.010	0.223	0.124	0.210	0.088	0.216	0.067	0.216
8	MXZR	120	2358	No	921600	15.600	Sí	0.309	0.373	0.179	0.155	0.123	0.464	0.296	0.089	0.177	0.457	0.308	0.329	0.251	0.273	0.242
9	XPGB	140	2338	No	921600	15.600	No	0.311	0.339	0.112	0.249	0.134	0.372	0.190	0.067	0.275	0.364	0.206	0.262	0.243	0.215	0.241
10	JZYT	78	2400	0.025	2073600	27	Sí	0.282	0.316	0.180	0.288	0.212	0.275	0.247	0.281	0.222	0.234	0.287	0.259	0.256	0.258	0.254
11	TSLK	40	2438	0.375	307200	23.600	No	0.316	0.339	0.095	0.403	0.134	0.229	0.123	0.403	0.134	0.119	0.234	0.297	0.190	0.261	0.184
12	CWSQ	136	2342	0.081	921600	15.600	Sí	0.156	0.172	0.105	0.124	0.085	0.188	0.153	No	0.150	0.089	0.226	0.063	0.192	0.045	0.188
13	ROCI	74	2404	No	921600	16.200	Sí	0.306	0.335	0.141	0.283	0.196	0.329	0.197	0.031	0.344	0.012	0.385	0.023	0.365	0.021	0.365
14	PAUM	68	2410	No	921600	16.200	No	0.250	0.274	0.115	0.180	0.132	0.319	0.156	0.087	0.206	0.304	0.184	0.224	0.196	0.196	0.195
Mean	NaN	165.200	2312.800	0.064	1059840	16.933	0.333	0.256	0.288	0.125	0.214	0.137	0.297	0.182	0.103	0.213	0.217	0.245	0.188	0.233	0.160	0.229

Tabla 6.1: Métricas por sujeto del Modelo I en validación cruzada con el conjunto completo

	User Code	User Sample Count	Training Set Size	Detection Fail Rate	Camera MPix	Screen Size	Wearing Glasses	Test Overall Acc	XYNorm Eucld AbsDev Mean	XYNorm Eucld AbsDev Std	XAbs Dev Mean	XAbs Dev Std	YAbs Dev Mean	YAbs Dev Std	X Dev Mean	X Dev Std	Y Dev Mean	Y Dev Std	XYNorm Eucld Dev Mean	XYNorm Eucld Dev Std	XY Dev Mean	XY Dev Std
0	ROOT	222	1790	0.009	921600	16.200	No	0.210	0.239	0.133	0.147	0.117	0.272	0.201	0.032	0.185	0.257	0.220	0.183	0.204	0.144	0.203
1	OOWZ	66	1946	No	921600	15.600	No	0.196	0.218	0.136	0.136	0.120	0.256	0.185	0.088	0.159	0.040	0.315	0.068	0.249	0.064	0.237
2	MPVE	772	1240	0.020	921600	16.200	No	0.169	0.188	0.098	0.157	0.120	0.180	0.136	0.006	0.198	0.043	0.221	0.031	0.210	0.024	0.209
3	LIDJ	104	1908	0.037	921600	16.200	No	0.187	0.207	0.114	0.214	0.168	0.160	0.110	0.060	0.266	0.099	0.168	0.082	0.223	0.079	0.217
4	XPGB	140	1872	No	921600	15.600	No	0.182	0.200	0.094	0.215	0.136	0.148	0.106	0.067	0.246	0.048	0.177	0.058	0.214	0.058	0.211
5	PAUM	68	1944	No	921600	16.200	No	0.350	0.386	0.101	0.215	0.110	0.485	0.156	0.004	0.243	0.485	0.158	0.343	0.205	0.244	0.200
6	HGNI	108	1904	No	921600	14	No	0.215	0.224	0.087	0.217	0.090	0.213	0.124	0.008	0.236	0.210	0.128	0.149	0.190	0.109	0.182
7	QXYT	154	1858	No	921600	15.600	No	0.280	0.346	0.141	0.088	0.065	0.472	0.212	0.023	0.108	0.471	0.214	0.333	0.170	0.247	0.161
8	QFWJ	338	1674	0.191	2457600	14.200	No	0.299	0.331	0.164	0.245	0.168	0.352	0.246	0.110	0.276	0.272	0.332	0.208	0.305	0.191	0.304
9	TSLK	40	1972	0.375	307200	23.600	No	0.209	0.233	0.095	0.171	0.132	0.247	0.139	0.125	0.178	0.181	0.219	0.156	0.200	0.153	0.199
Mean	NaN	201.200	1810.800	0.063	1013760	16.340	No	0.230	0.257	0.116	0.181	0.123	0.279	0.161	0.052	0.209	0.211	0.215	0.161	0.217	0.131	0.212

Tabla 6.2: Métricas por sujeto del Modelo I en validación cruzada excluyendo sujetos con gafas del conjunto

	XYNorm Eucld AbsDev Mean	XYNorm Eucld AbsDev Std	XAbs Dev Mean	XAbs Dev Std	YAbs Dev Mean	YAbs Dev Std	X Dev Mean	X Dev Std	Y Dev Mean	Y Dev Std	XYNorm Eucld Dev Mean	XYNorm Eucld Dev Std	XY Dev Mean	XY Dev Std
0	0.084	0.067	0.065	0.067	0.084	0.085	0.001	0.093	0.042	0.112	0.029	0.103	0.021	0.103

Tabla 6.3: Métricas del Modelo I en evaluación con sujetos conocidos excluyendo sujetos con gafas del conjunto.

6.1.2 Modelo II: Desglose de métricas

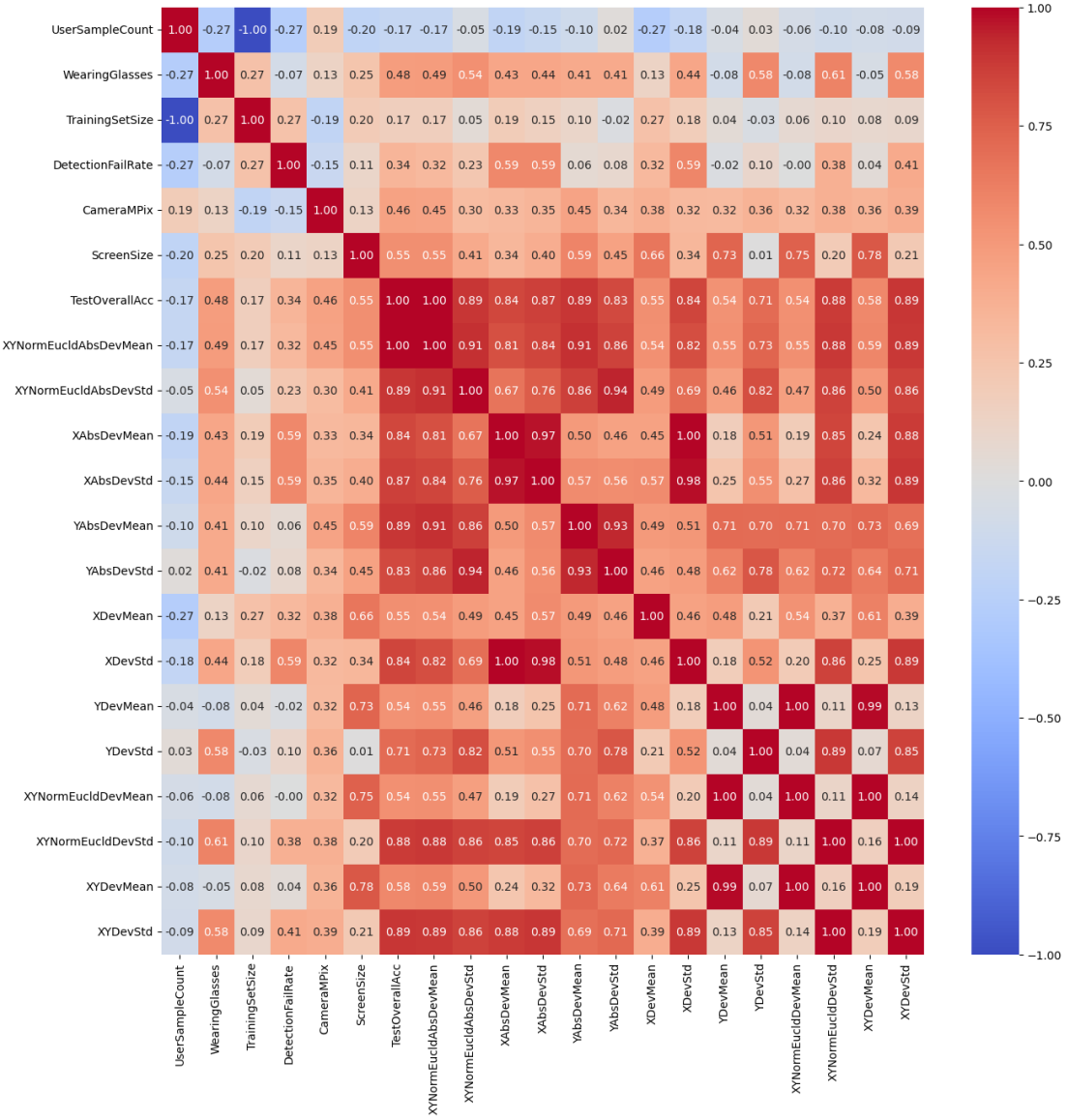


Figura 6.6: Mapa de correlación por pares de las métricas del Modelo II con estadísticas del conjunto de datos tomadas de la sección 3.4 del Capítulo 3.

	User Code	User Sample Count	Wearing Glasses	Training Set Size	Detection Fail Rate	Camera MPix	Screen Size	Test Overall Acc	XYNorm Eucld AbsDev Mean	XYNorm Eucld AbsDev Std	XAbs Dev Mean	XAbs Dev Std	YAbs Dev Mean	YAbs Dev Std	X Dev Mean	X Dev Std	Y Dev Mean	Y Dev Std	XYNorm Eucld Dev Mean	XYNorm Eucld Dev Std	XY Dev Mean	XY Dev Std
0	MPVE	766	No	1560	0.028	921600	16.200	0.817	0.138	0.085	0.104	0.088	0.141	0.119	0.016	0.135	No	0.185	0.012	0.162	0.008	0.160
1	XPGB	140	No	2186	No	921600	15.600	0.679	0.229	0.124	0.111	0.079	0.293	0.178	0.004	0.137	0.288	0.186	0.203	0.163	0.146	0.161
2	MXZR	120	Sí	2206	No	921600	15.600	0.492	0.319	0.174	0.161	0.105	0.393	0.270	0.016	0.192	0.139	0.457	0.099	0.350	0.078	0.324
3	QXYT	150	No	2176	0.026	921600	15.600	0.680	0.200	0.118	0.130	0.097	0.220	0.182	0.017	0.162	0.205	0.199	0.145	0.181	0.111	0.180
4	ROOT	222	No	2104	0.009	921600	16.200	0.730	0.160	0.096	0.088	0.071	0.190	0.144	0.028	0.110	0.164	0.173	0.118	0.145	0.096	0.141
5	HGNI	74	No	2252	0.315	921600	14	0.824	0.116	0.064	0.081	0.056	0.126	0.099	0.013	0.098	0.097	0.128	0.069	0.114	0.055	0.113
6	CWSQ	146	Sí	2180	0.014	921600	15.600	0.644	0.220	0.104	0.133	0.094	0.252	0.169	0.072	0.147	0.147	0.266	0.116	0.215	0.110	0.206
7	QFWJ	338	No	1988	0.191	2457600	14.200	0.482	0.357	0.158	0.271	0.185	0.370	0.246	0.018	0.328	0.217	0.388	0.154	0.359	0.117	0.358
8	LIDJ	84	No	2242	0.222	921600	16.200	0.619	0.219	0.101	0.151	0.133	0.240	0.136	0.001	0.202	0.224	0.162	0.158	0.183	0.112	0.182
9	PAUM	66	No	2260	0.029	921600	16.200	0.773	0.191	0.063	0.138	0.061	0.223	0.091	0.004	0.152	0.209	0.120	0.148	0.137	0.107	0.136
10	ROCI	58	Sí	2268	0.216	921600	16.200	0.569	0.305	0.156	0.285	0.178	0.286	0.202	0.048	0.334	0.211	0.280	0.153	0.308	0.129	0.307
11	JUFE	56	Sí	2270	0.243	921600	16.200	0.411	0.324	0.179	0.345	0.203	0.272	0.201	0.012	0.403	0.181	0.287	0.128	0.350	0.096	0.345
12	TSLK	40	No	2286	0.375	307200	23.600	0.550	0.276	0.140	0.190	0.120	0.302	0.225	0.058	0.219	0.298	0.230	0.215	0.225	0.178	0.224
13	OOWZ	66	No	2260	No	921600	15.600	0.818	0.109	0.063	0.066	0.051	0.126	0.093	0.002	0.084	0.023	0.156	0.016	0.125	0.013	0.120
Mean	NaN	166.143	0.286	2159.857	0.119	987428.571	16.214	0.649	0.226	0.116	0.161	0.109	0.245	0.168	0.022	0.193	0.172	0.230	0.124	0.216	0.097	0.211

Tabla 6.4: Métricas por sujeto del Modelo II en validación cruzada con el conjunto completo.

	User Code	User Sample Count	Training Set Size	Detection Fail Rate	Camera MPix	Screen Size	Test Overall Acc	XYNorm Eucld AbsDev Mean	XYNorm Eucld AbsDev Std	XAbs Dev Mean	XAbs Dev Std	YAbs Dev Mean	YAbs Dev Std	X Dev Mean	X Dev Std	Y Dev Mean	Y Dev Std	XYNorm Eucld Dev Mean	XYNorm Eucld Dev Std	XY Dev Mean	XY Dev Std
0	LIDJ	84	1862	0.222	921600	16.200	0.643	0.284	0.112	0.167	0.152	0.330	0.165	0.044	0.222	0.330	0.165	0.235	0.195	0.187	0.193
1	MPVE	766	1180	0.028	921600	16.200	0.804	0.147	0.090	0.115	0.089	0.147	0.129	0.018	0.145	0.056	0.187	0.042	0.167	0.037	0.166
2	ROOT	222	1724	0.009	921600	16.200	0.779	0.158	0.105	0.108	0.085	0.181	0.144	0.032	0.134	0.144	0.181	0.104	0.159	0.088	0.158
3	QFWJ	338	1608	0.191	2457600	14.200	0.447	0.334	0.158	0.213	0.158	0.374	0.251	0.035	0.263	0.281	0.352	0.201	0.310	0.158	0.307
4	XPGB	140	1806	No	921600	15.600	0.671	0.249	0.127	0.170	0.115	0.285	0.180	0.019	0.204	0.282	0.185	0.200	0.195	0.151	0.195
5	TSLK	40	1906	0.375	307200	23.600	0.725	0.219	0.125	0.184	0.113	0.207	0.195	0.033	0.215	0.195	0.208	0.140	0.212	0.114	0.212
6	QXYT	150	1796	0.026	921600	15.600	0.640	0.248	0.124	0.133	0.101	0.293	0.200	0.027	0.165	0.292	0.201	0.207	0.184	0.159	0.183
7	HGNI	74	1872	0.315	921600	14	0.838	0.111	0.046	0.086	0.052	0.114	0.077	0.025	0.098	0.015	0.137	0.020	0.119	0.020	0.118
8	OOWZ	66	1880	No	921600	15.600	0.818	0.124	0.074	0.086	0.056	0.132	0.116	0.008	0.103	0.047	0.170	0.034	0.141	0.028	0.136
9	PAUM	66	1880	0.029	921600	16.200	0.803	0.147	0.049	0.116	0.053	0.162	0.075	0.011	0.128	0.129	0.124	0.092	0.126	0.070	0.126
Mean	NaN	194.600	1751.400	0.120	1013760	16.340	0.717	0.202	0.101	0.138	0.097	0.223	0.153	0.025	0.168	0.177	0.191	0.127	0.181	0.101	0.179

Tabla 6.5: Métricas por sujeto del Modelo II en validación cruzada excluyendo sujetos con gafas del conjunto.

	XYNorm Eucld AbsDev Mean	XYNorm Eucld AbsDev Std	XAbs Dev Mean	XAbs Dev Std	YAbs Dev Mean	YAbs Dev Std	X Dev Mean	X Dev Std	Y Dev Mean	Y Dev Std	XYNorm Eucld Dev Mean	XYNorm Eucld Dev Std	XY Dev Mean	XY Dev Std
0	0.061	0.042	0.044	0.041	0.064	0.056	0.005	0.060	0.041	0.075	0.030	0.068	0.023	0.067

Tabla 6.6: Métricas del Modelo II en evaluación con sujetos conocidos excluyendo sujetos con gafas del conjunto.

6.1.3 Modelo III: Desglose de métricas

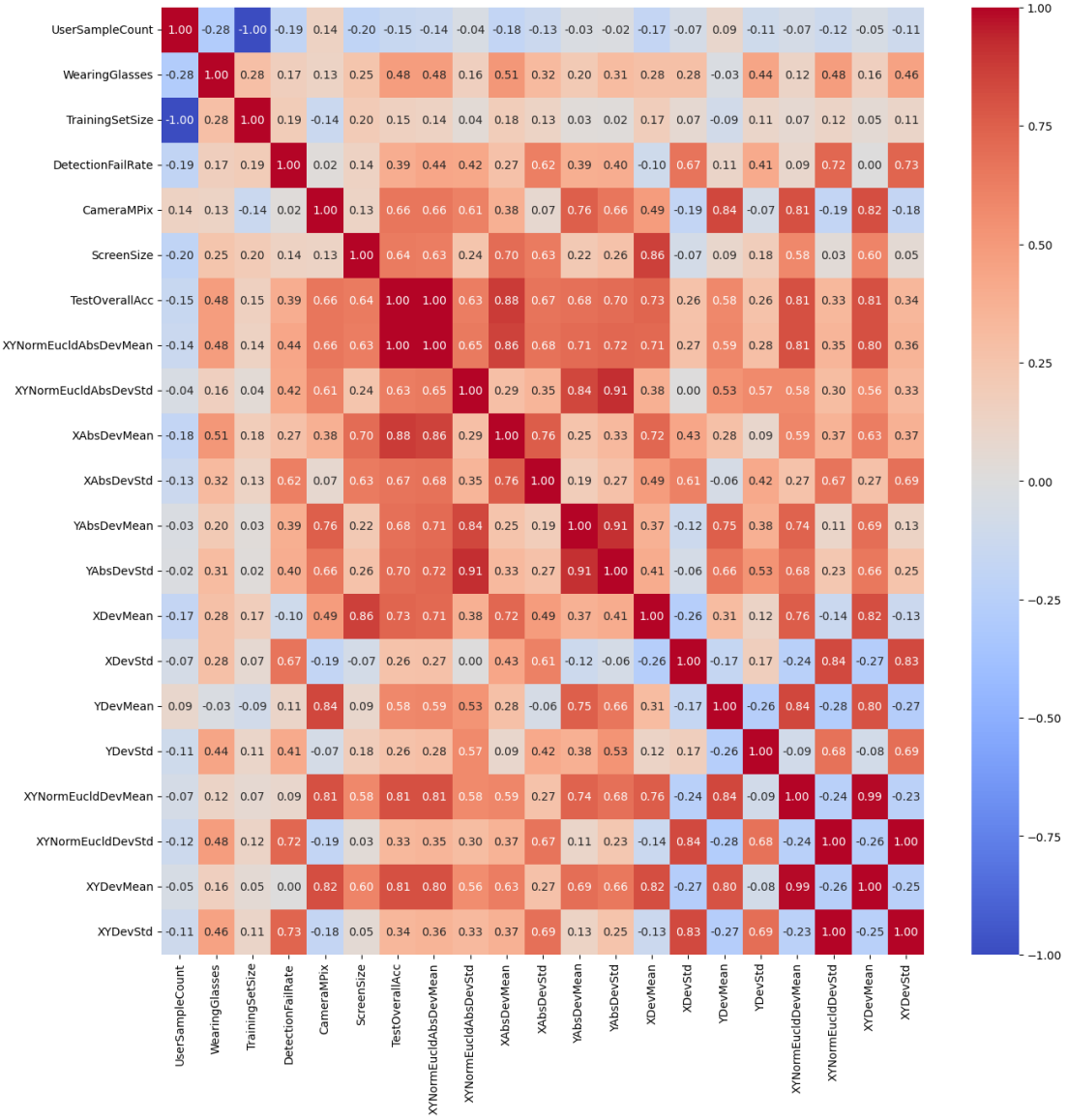


Figura 6.7: Mapa de correlación por pares de las métricas del Modelo III con estadísticas del conjunto de datos tomadas de la sección 3.4 del Capítulo 3.

	User Code	User Sample Count	Wearing Glasses	Training Set Size	Detection Fail Rate	Camera MPix	Screen Size	Test Overall Acc	XYNorm Eucld AbsDev Mean	XYNorm Eucld AbsDev Std	XAbs Dev Mean	XAbs Dev Std	YAbs Dev Mean	YAbs Dev Std	X Dev Mean	X Dev Std	Y Dev Mean	Y Dev Std	XYNorm Eucld Dev Mean	XYNorm Eucld Dev Std	XY Dev Mean	XY Dev Std
0	MPVE	752	No	1590	0.046	921600	16.200	0.200	0.224	0.109	0.216	0.154	0.183	0.141	0.042	0.262	0.098	0.209	0.075	0.237	0.070	0.235
1	CWSQ	144	Sí	2198	0.027	921600	15.600	0.207	0.238	0.119	0.156	0.126	0.258	0.186	0.093	0.178	0.074	0.310	0.084	0.253	0.084	0.244
2	HGNI	105	No	2237	0.028	921600	14	0.166	0.190	0.098	0.151	0.134	0.182	0.135	0.046	0.197	0.165	0.155	0.121	0.177	0.106	0.176
3	JZYT	78	Sí	2264	0.025	2073600	27	0.416	0.455	0.137	0.521	0.196	0.311	0.217	0.521	0.196	0.304	0.226	0.426	0.212	0.412	0.211
4	PAUM	66	No	2276	0.029	921600	16.200	0.180	0.199	0.099	0.231	0.141	0.129	0.092	0.074	0.262	0.073	0.141	0.074	0.210	0.074	0.201
5	QXYT	149	No	2193	0.032	921600	15.600	0.173	0.192	0.115	0.178	0.129	0.169	0.153	0.111	0.190	0.054	0.222	0.087	0.207	0.082	0.206
6	QFWJ	286	No	2056	0.316	2457600	14.200	0.296	0.340	0.155	0.188	0.148	0.404	0.244	0.101	0.217	0.404	0.244	0.294	0.231	0.252	0.231
7	ROOT	215	No	2127	0.040	921600	16.200	0.192	0.213	0.123	0.189	0.145	0.195	0.161	0.065	0.230	0.128	0.218	0.101	0.224	0.096	0.224
8	XPGB	136	No	2206	0.029	921600	15.600	0.213	0.233	0.106	0.221	0.156	0.205	0.127	0.078	0.260	0.017	0.241	0.057	0.250	0.048	0.250
9	TSLK	39	No	2303	0.391	307200	23.600	0.226	0.261	0.123	0.219	0.195	0.234	0.165	0.168	0.241	0.004	0.288	0.119	0.266	0.086	0.265
10	MXZR	117	Sí	2225	0.025	921600	15.600	0.185	0.206	0.121	0.154	0.130	0.216	0.163	0.053	0.195	0.033	0.269	0.044	0.235	0.043	0.232
11	LIDJ	97	No	2245	0.102	921600	16.200	0.229	0.252	0.144	0.215	0.167	0.243	0.189	0.124	0.243	0.071	0.300	0.101	0.273	0.097	0.272
12	JUFE	40	Sí	2302	0.459	921600	16.200	0.313	0.352	0.129	0.378	0.196	0.248	0.199	0.016	0.430	0.165	0.273	0.117	0.360	0.090	0.352
13	OOWZ	63	No	2279	0.045	921600	15.600	0.207	0.225	0.114	0.158	0.101	0.257	0.162	0.019	0.187	0.254	0.165	0.180	0.177	0.137	0.176
14	ROCI	55	Sí	2287	0.257	921600	16.200	0.249	0.277	0.113	0.294	0.180	0.204	0.138	0.081	0.337	0.026	0.246	0.060	0.295	0.053	0.292
Mean	NaN	156.133	0.333	2185.867	0.123	1059840	16.933	0.230	0.257	0.120	0.231	0.153	0.229	0.165	0.106	0.242	0.125	0.234	0.129	0.240	0.115	0.238

Tabla 6.7: Métricas por sujeto del Modelo III en validación cruzada con el conjunto completo

	User Code	User Sample Count	Training Set Size	Detection Fail Rate	Camera MPix	Screen Size	Test Overall Acc	XYNorm Eucld AbsDev Mean	XYNorm Eucld AbsDev Std	XAbs Dev Mean	XAbs Dev Std	YAbs Dev Mean	YAbs Dev Std	X Dev Mean	X Dev Std	Y Dev Mean	Y Dev Std	XYNorm Eucld Dev Mean	XYNorm Eucld Dev Std	XY Dev Mean	XY Dev Std
0	MPVE	752	1156	0.046	921600	16.200	0.188	0.211	0.105	0.213	0.148	0.164	0.129	0.008	0.259	0.051	0.202	0.036	0.233	0.029	0.231
1	LIDJ	97	1811	0.102	921600	16.200	0.241	0.268	0.147	0.219	0.171	0.263	0.200	0.036	0.277	0.103	0.315	0.077	0.297	0.069	0.296
2	PAUM	66	1842	0.029	921600	16.200	0.214	0.243	0.092	0.303	0.142	0.125	0.087	0.112	0.317	0.011	0.153	0.080	0.249	0.062	0.235
3	QXYT	149	1759	0.032	921600	15.600	0.207	0.236	0.128	0.222	0.166	0.192	0.174	0.103	0.258	0.126	0.227	0.115	0.243	0.115	0.242
4	ROOT	215	1693	0.040	921600	16.200	0.206	0.228	0.135	0.199	0.165	0.213	0.168	0.085	0.245	0.137	0.235	0.114	0.240	0.111	0.240
5	QFWJ	286	1622	0.316	2457600	14.200	0.259	0.296	0.148	0.184	0.130	0.334	0.238	0.058	0.218	0.322	0.254	0.232	0.237	0.190	0.236
6	XPGB	136	1772	0.029	921600	15.600	0.236	0.261	0.118	0.238	0.163	0.235	0.160	0.077	0.279	0.131	0.252	0.108	0.266	0.104	0.266
7	OOWZ	63	1845	0.045	921600	15.600	0.195	0.216	0.110	0.153	0.099	0.236	0.169	0.053	0.176	0.234	0.172	0.170	0.174	0.144	0.174
8	HGNI	105	1803	0.028	921600	14	0.185	0.204	0.098	0.197	0.135	0.172	0.125	0.128	0.202	0.155	0.146	0.142	0.176	0.142	0.174
9	TSLK	39	1869	0.391	307200	23.600	0.228	0.253	0.136	0.213	0.128	0.244	0.210	0.065	0.242	0.196	0.256	0.146	0.249	0.131	0.249
Mean	NaN	190.800	1717.200	0.106	1013760	16.340	0.216	0.241	0.122	0.214	0.145	0.218	0.166	0.073	0.247	0.147	0.221	0.122	0.236	0.110	0.234

Tabla 6.8: Métricas por sujeto del Modelo III en validación cruzada excluyendo sujetos con gafas del conjunto

	XYNorm Eucld AbsDev Mean	XYNorm Eucld AbsDev Std	XAbs Dev Mean	XAbs Dev Std	YAbs Dev Mean	YAbs Dev Std	X Dev Mean	X Dev Std	Y Dev Mean	Y Dev Std	XYNorm Eucld Dev Mean	XYNorm Eucld Dev Std	XY Dev Mean	XY Dev Std
0	0.055	0.053	0.049	0.051	0.049	0.064	0.011	0.070	0.007	0.081	0.009	0.076	0.009	0.076

Tabla 6.9: Métricas del Modelo III en evaluación con sujetos conocidos.

6.2 Análisis

El proceso de evaluación comienza con la prueba de validación cruzada *leave-one-out* a los tres modelos desarrollados y presentados en el anterior Capítulo, que concluye con unos resultados no demasiado prometedores recogidos en las figuras 6.1 en las que se observa que la métrica en función de la cual se expresa la exactitud se establece entre 0.3 y 0.2, lo que significa que la desviación media entre predicciones y valores reales en una pantalla comprendería entre el 20 y el 30 % de la misma. No es una buena noticia, pues aunque los modelos en este punto no son comparables de forma justa con las soluciones enumeradas en el Capítulo 2, ya que estas cuentan con mecanismos de calibración durante su inicialización y/o utilización que permite adaptarlas al usuario y a sus circunstancias, de forma que los resultados de las pruebas practicadas por sus desarrolladores no reflejan la capacidad de generalización de sus modelos, como sí ocurre en este caso.

Analizando el caso particular para cada sujeto se concluye que aquellos cuyas características difieren más de la media son los mismos con los que los modelos tuvo peor desempeño. Estos son:

- Aquellos sujetos que llevaron gafas durante la adquisición de sus muestras. Es el caso de MZXR, CSWQ, JZYT, JUFE y ROCI
- Aquellos sujetos que utilizaron equipos con monitores de tamaños superiores a 24 pulgadas, presumiblemente de sobremesa. Es el caso de JZYT y TSLK
- Aquellos sujetos cuyas cámaras tomaron muestras a resoluciones altas, superiores a 1280x720, coloquialmente denominada 720p o HD. Es el caso de JZYT y QWFJ

En el caso de los sujetos con gafas, también se observa, en el mapa de correlación de las figuras 6.5 y 6.6; una cierta relación entre la tasa de fallo del detector facial basado en Haar Cascade (`DetectionFailRate`) y la métrica booleana `WearingGlasses`, que toma un valor positivo cuando el sujeto en cuestión lleva gafas. Esto significa que el detector tuvo más dificultades para identificar los ojos de estos sujetos, lo cual tiene sentido teniendo en cuenta que estos contienen una gran parte de las características que el detector evalúa para determinar si en la región de trabajo existe o no un rostro. El hecho de que suceda evidencia que los reflejos u oclusiones causados por las gafas empeoren la calidad de las muestras de estos sujetos.

Otro efecto, que se debe al reducido tamaño del conjunto y al mecanismo de validación cruzada, se observa especialmente en el caso de los sujetos con monitores y cámara de resoluciones atípicamente grandes, que al ser muy pocos los sujetos con estas peculiaridades, en las iteraciones en las que forman parte del conjunto de test, el conjunto de entrenamiento se queda práctica o totalmente sin muestras con características similares a las de estos sujetos, ya que todas o casi todas se destinan a evaluación. La consecuencia

es que el modelo no logra aprender las relaciones que le permiten generalizar en estos casos atípicos, y el resultado no es bueno.

De hecho, se puede decir que se aleja de lo esperado. Una hipótesis que cabe plantear es que con cámaras con mayor resolución se lograrían mejores resultados, ya que un mayor nivel de detalle y de valores en las muestras permitiría a los modelos aprender más relaciones y más complejas durante el entrenamiento. Sin embargo, este análisis, de nuevo debido al reducido número de muestras del conjunto, no lo ha podido probar para los modelos I y II. Es posible ver en las figuras 6.5, 6.6 una cierta correlación positiva entre la variable `CameraMPix`, que recoge los megapíxeles de las cámaras de los sujetos; y las métricas de dispersión en función de las cuales se expresa la precisión, lo que implica que los resultados de la evaluación de validación cruzada mostraron peor precisión en sujetos con cámaras de alta resolución. No es así en el caso del Modelo III, que muestra una correlación inversa, que evidencia lo contrario. Cabe esperar que este modelo sí pueda sacar ventaja de trabajar con imágenes más grandes, ya que MobileNet está pensada para ello. Por otro lado, las métricas que expresan la veracidad empeoran cuando aumentan los megapíxeles en los tres casos, necesitando un conjunto más grande y balanceado para demostrar consistentemente este aspecto.

En cualquier caso, otra hipótesis que se manejaba, en vista de estos resultados, era que las muestras de los sujetos con gafas no terminaban de ser suficientemente buenas, y que la exactitud mejoraría si se repetía la prueba prescindiendo de ellas. Por tanto, así se procedió y se demostró con éxito, tal y como refleja la figura 6.2, logrando una mejoría de en torno al 9 % en las métricas de exactitud y precisión, y manifestando la necesidad de aumentar el número de muestras de sujetos con gafas en el conjunto, vigilando la corrección de las mismas, para mejorar los resultados en estas circunstancias e incluso entrenar dos copias de cada modelo para sujetos con gafas y sin gafas.

Una última hipótesis que se planteó durante el desarrollo es que el mecanismo de evaluación cruzada *LOOCV* caracteriza la precisión del modelo únicamente en la condición más desfavorable posible, en la que el modelo no "conoce" al sujeto cuyas muestras se usan para evaluar su exactitud. Para conseguir unos buenos resultados en esta prueba, se debería dar al menos una de estas condiciones:

- El entrenamiento de la red neuronal debe ser lo más adecuado posible al caso de uso para lograr una buena capacidad de generalización. Esto se puede lograr ajustando los hiperparámetros para minimizar las pérdidas en entrenamiento y validación; y utilizando un conjunto de datos suficientemente amplio. Lo primero ya se ha hecho, pues como se ha mostrado en la sección 5.4, se han optimizado los modelos hasta lograr valores de pérdidas muy bajos con gradientes casi nulos en las últimas épocas del entrenamiento llegando a utilizar incluso *callbacks* que reducen la tasa de aprendizaje o incluso concluyen el proceso cuando deja de ser viable mejorar más los valores. Lo segundo es una limitación asociada a este proyecto. Para tratar con ella, se ha reducido la complejidad de la arquitectura en los modelos que implementan Haar Cascade en su preprocesado y se ha hecho transferencia de

aprendizaje en el que usa YuNet, además de aumentación de datos en todos ellos.

- Las muestras del sujeto del conjunto de evaluación deberían ser similares en cuanto a características a algunas muestras del conjunto de entrenamiento, ya que, como se puede contrastar con los gráficos de la Sección 3.4 del Capítulo 3, los mejores resultados en cuanto a precisión y exactitud los obtuvieron los sujetos cuyas características fueron más próximas al promedio del conjunto. Si bien es coherente afirmar que lo ideal sería lograr un modelo capaz de generalizar tanto como para salvar las posibles peculiaridades de cada sujeto, hay ciertos aspectos, como la inclinación del monitor o de la cámara, el campo de visión de la misma, la posición del dispositivo respecto del usuario en ciertos modos y ejes, o las condiciones de iluminación; que son imposibles de registrar y cuantificar con un sistema de adquisición orientado a entornos no controlados, como la aplicación web que se ha desarrollado en este proyecto; que por supuesto pueden afectar a las imágenes de las muestras, y que no es realista esperar que el modelo sea capaz de abstraer completamente.

Esto último es lo que se ha demostrado en los gráficos de la figura 6.3, que recoge las métricas obtenidas de evaluar –mediante el mecanismo para sujetos "conocidos" propuesto en la sección 5.5– los modelos entrenados con los conjuntos que mejores resultados dieron en validación cruzada [1], evidenciando la indiscutible mejora en la exactitud del sistema cuando se incluyen datos del sujeto de evaluación en el conjunto de entrenamiento, de forma análoga a un usuario que utiliza una pantalla de calibración para adquirir datos con los que entrenar el sistema antes de su uso, adaptándolo a sus condiciones; en un hipotético entorno de producción.

De los tres modelos neuronales evaluados, el Modelo III –el único que incorpora YuNet en su tubería de preprocesado y redes preentrenadas en su arquitectura– ha obtenido los mejores resultados, lo que pone de manifiesto el potencial de esta red, la más compleja y avanzada del proyecto.

Para contextualizar y validar la calidad de estos resultados, se procederá a comparar las métricas alcanzadas por este modelo con aquellas publicadas por los desarrolladores de WebGazer.js y Real-Eye haciendo las transformaciones pertinentes, ya que estos, en sus artículos expresan las métricas en píxeles, dados la resolución y el tamaño de la diagonal en pulgadas de su monitor, lo que permite normalizar los valores e aislar en cierto modo las métricas de los parámetros del monitor [2], aunque para esta comparativa,

[1] Más adelante se comparan los resultados de la mejor propuesta con dos soluciones ya existentes. Dado que en los estudios de ambas participaron sujetos que utilizaron gafas en el proceso, para que la comparación fuera justa, se ha evaluado el Modelo III usando todo el conjunto, sin excluir a sujetos con gafas.

[2] Normalizar las métricas no necesariamente excluye el monitor de la ecuación, pues aunque ello permita extrapolarlas a pantallas de otras características, la exactitud del modelo no tiene por qué ser la misma en estas.

se va a asumir que lo hace completamente [3] [4]. Para los casos en los que las métricas se calculan mediante la distancia euclídea entre valores esperados y predicciones, se usa esta expresión:

$$z = \frac{x_{px}}{\sqrt{W_{px}^2 + H_{px}^2}}$$

donde x_{px} es la observación en píxeles, y z, el valor normalizado.

La comparativa ideal requeriría incluir en el conjunto, muestras adquiridas en dispositivos con monitores de características similares a aquellos utilizados en los estudios consultados, de varios sujetos, siendo necesario que el conjunto de test predominaran dichas muestras. O, en defecto de esta ampliación del conjunto, implementar un mecanismo de calibración que adquiera datos del usuario durante la inicialización para incorporarlos de algún modo al sistema, tal y como hacen WebGazer.js y Real-Eye, entre otras. Cualquiera de los dos casos forman parte de las líneas futuras del Trabajo y por tanto, escapan del alcance del mismo, por lo que la comparativa desarrollada es lo más adecuada posible a las circunstancias.

WebGazer.js cuenta con dos modelos. El primero de ellos es un regresor lineal, mientras que el segundo es un regresor *ridge* que consigue mejores resultados, que en [30] muestran con el modelo base y también con mecanismos de postprocesado que proporcionan realimentación al sistema para ajustarse sobre la marcha y mejorar su precisión. Además, recogen datos tanto de un estudio en línea como de un estudio presencial en el que se asume un entorno más controlado. Si bien los datos del conjunto del proyecto se adquirieron mediante una solución análoga al mencionado estudio en línea, para esta comparativa se han usado los resultados del estudio presencial, ya que es el único del que presentan una resolución del monitor con la que normalizar los resultados en píxeles.

En vista de la tabla 6.10, en términos de exactitud, medida como la distancia euclídea media[5], el Modelo III supera claramente a las variantes de WebGazer.js. Este modelo alcanza una media de 0.055, frente a los 0.075 del regresor *ridge* con muestreo de movimientos de cursor y los 0.100 del regresor ridge sin realimentación.

La desviación estándar de la exactitud en el Modelo III también es inferior a las dos medidas de los regresores de WebGazer.js, lo que sugiere que la métrica de exactitud también es algo más consistente.

Real-Eye, por su parte, cuenta con una versión para móviles –que no es objeto de

[3] El tamaño medio de las pantallas utilizadas en el conjunto del proyecto es de 16,9 pulgadas, que no se aleja demasiado de los monitores de 24 pulgadas que se han empleado para los estudios en los otros dos casos.

[4] Dado que los tres sistemas adquieren datos mediante aplicaciones web, no es posible determinar la resolución de la pantalla si no se conoce antes. En el análisis del presente sistema, dado que las muestras con las que se evalúa provienen del conjunto del proyecto, solo se conocen los tamaños, no las resoluciones

[5] Menor es mejor

	Regresor ridge	Regresor ridge + muestreo de movimientos de cursor	Métrica equivalente en el proyecto
Monitor (Píxeles)	24' 1920x1200	24' 1920x1200	
Exactitud (Distancia euclídea media)	226.7px	169.0px	XYNormEucldAbsDevMean (Exactitud)
Desviación estándar de la anterior	175.3px	147.1px	XYNormEucldAbsDevStd

Tabla 6.10: Métricas de las implementaciones de WebGazer.js: Mejor y peor resultado del estudio en línea [30].

	Regresor lineal	Regresor ridge + muestreo de movimientos de cursor	Modelo III
Exactitud (Distancia euclídea media)	0.100	0.075	0.055
Desviación estándar de la anterior	0.077	0.065	0.053

Tabla 6.11: Comparación de métricas de la tabla anterior normalizadas.

esta comparativa– y otra para ordenadores de escritorio y portátiles. En [29], sus desarrolladores analizan y muestran los resultados de un estudio presencial con equipos de características similares.

Es un estudio denso que evalúa la precisión y la exactitud mediante varios métodos, en diferentes momentos temporales desde la calibración inicial, en diversas condiciones de iluminación, y para diferentes grupos de usuarios en función del uso de gafas o lentes de contacto, entre otros aspectos.

Para la comparación se han tomado los valores del caso más genérico posible cuyo método de medición sea evaluar el promedio de las distancias euclídeas entre predicciones y valores esperados, tal y como se ha hecho en [30] y en el análisis de los modelos del proyecto.

Real-Eye es poco tolerante a los movimientos de los usuarios. Por ello, emite un aviso y muestra guías en pantalla cuando detecta un desplazamiento en el plano de la cámara para ayudar al usuario a recuperar su posición inicial, registrada durante la calibración. También es este el motivo de que en el estudio se muestren las métricas tanto del momento de la calibración como de un tiempo después, ya que probablemente debido al desplazamiento –voluntario o involuntario– del usuario, las métricas siempre han sido peores al evaluarse un tiempo después. En cualquier caso, al tratarse de un efecto cuya causa no necesariamente tiene que ver con el sistema

	Real-Eye	Métrica equivalente en el proyecto
Monitor (Píxeles)	24' 1920x1080	
Exactitud (Distancia euclídea media)	149.3px	XYNormEucldAbsDevMean (Exactitud)
Precisión (Desviación típica en predicción)	147.2px	XYNormEucldDevStd (Precisión)

Tabla 6.12: Métricas de precisión y exactitud de Real-Eye, tomadas de [29].

	Real-Eye	Modelo III
Exactitud (Distancia euclídea media)	0.068	0.055
Precisión (Desviación típica en la estimación)	0.067	0.076

Tabla 6.13: Comparación de métricas de la tabla anterior normalizadas

El principal motivo de evaluar en dos instantes temporales las métricas es que siempre han sido mejores inmediatamente después de la fase de calibración inicial que un tiempo después. En [29] se sugiere que podría deberse a circunstancias relacionadas con el estudio. En cualquier caso, para la comparación se tomarán las métricas iniciales, sin tener en cuenta dicha degradación, que no necesariamente es inherente al sistema.

De la tabla 6.13[6][7] se concluye que la exactitud del Modelo III es superior también en este caso, a pesar de que los resultados de Real-Eye sean mejores que los de WebGazer.js.

No ocurre lo mismo con la métrica de precisión, que es algo mejor en el caso de Real-Eye; lo que evidencia que la veracidad, cuya medición no aparece en [29], es mejor en el Modelo III para compensar una peor precisión y así justificar la mejoría en exactitud.

[6]Calculadas mediante el método *mean_Euclidean_dist_of_all_fixations* en la fase *First Re-Validation* incluyendo valores atípicos.

[7]Menor es mejor.

Capítulo 7

Conclusiones

A lo largo del presente Trabajo, se ha mostrado paso por paso el desarrollo de una prueba de concepto de un sistema de *Eye-Tracking* basado en *Deep Learning*, sin uso de hardware específico, y se ha demostrado cómo su rendimiento se aproxima al de soluciones ya existentes o incluso lo supera cuando se incorporan muestras aleatorias del usuario en el conjunto de entrenamiento, lo cual, dado que estas soluciones se encuentran en una fase más avanzada y cuentan con mecanismos de calibración, es un indicador más que robusto del potencial del sistema propuesto.

7.1 Consecución de los objetivos

El objetivo principal, del desarrollo de un sistema preliminar para *Eye-Tracking* con funcionalidad demostrable, demostrando su viabilidad, ha sido alcanzado con éxito y con creces, ya que se ha probado que además de ser funcional, la exactitud de dicho sistema puede llegar a ser igual o superior a la de ciertas soluciones existentes en el mercado cuyo desarrollo en cualquier caso está en fases posteriores.

Esto es una muestra del potencial que tiene el *Deep Learning* para resolver este tipo de problemas, lo cual también se puede evidenciar en otros trabajos, como ocurre con la comparación entre Haar Cascade y YuNet para el caso de uso de detección facial y de facciones, donde la segunda, que está basada en redes neuronales, con los mismos recursos logra resultados mejores a la vez que es más robusta y menos sensible a su configuración.

En cuanto a los objetivos específicos, igualmente han sido alcanzados, pues se han desarrollado pruebas y métricas que han permitido expresar en función de las mismas, la exactitud de los modelos a la hora de inferir predicciones; evaluando para ello la función de error absoluto medio, medias aritméticas y desviaciones estándar, así como otros instrumentos comunes en el análisis estadístico.

Este análisis refuerza la idea de que el significado que se le da al término coloquial de precisión para expresar la corrección de los valores que infiere un sistema no es correcto desde el punto de vista estricto, y que hay que distinguir dos componentes que no necesariamente aparecen estrechamente correladas y por tanto, no necesariamente se deben a los mismos factores o elementos. Estas son la veracidad y la precisión, que a su vez componen la exactitud, como recoge la norma ISO 5725.

También se han implementado soluciones parciales, como el proyecto del Trabajo de Fin de Grado de Raúl Barba para la adquisición de datos, como los detectores de características Haar Cascade de OpenCV la red convolucional profunda para detección facial en CPU, YuNet, formando parte de la tubería de preprocesado que compone el esqueleto del sistema en los pasos previos a la inferencia de coordenadas, o incluso MobileNet, la red convolucional profunda que se instancia en el modelo que utiliza YuNet para el preprocesado, valiéndose de las ventajas del aprendizaje por transferencia. Y de nuevo, mostrando la evidente superioridad del aprendizaje profundo y las redes convolucionales frente a otras tecnologías para estos casos de uso.

En este Trabajo se han aplicado contenidos de una gran cantidad de asignaturas del Grado, fundamentalmente de la Mención en Telemática:

- Sistemas en Tiempo Real: Siendo la que más relación tiene con el proyecto, esta asignatura optativa comienza su primer bloque con un tema exclusivamente para iniciación en Python, seguido de otros contenidos relacionados con sistemas embebidos y *Deep Learning*.

- Tecnologías para Aplicaciones Web: Una de las más completas del Grado, pues cursándola se adquieren bases sobre el desarrollo Web, que es la tendencia en el ámbito de la ingeniería de software en la actualidad, dada la modularidad y la escalabilidad y la compatibilidad, frente al desarrollo multiplataforma y de aplicaciones nativas monolíticas. Los contenidos pasan por las tecnologías del lado del cliente más elementales, como HTML, las hojas de estilo CSS, el lenguaje Javascript y AJAX para la comunicación de estos scripts con un servidor. Como también por las bases de datos relacionales SQL y PHP, para conectar el cliente de las aplicaciones con la base de datos e implementar lógica en el servidor. El Trabajo de Raúl Barba hace evidente la influencia de esta asignatura en la aplicación de adquisición, ya que se vale de todo lo que se ha comentado y además, motiva una de las líneas futuras más claras de este proyecto.

- Administración y Gestión de Redes de Comunicación: Para poner en marcha la aplicación web de adquisición hizo falta configurar un servidor que hiciera accesible el servicio a los usuarios participantes. Para ello se utilizó una máquina virtual con Linux, propiedad del Grupo de investigación, y en ella se instaló el servidor web Apache, tal y como se haría durante el transcurso de las sesiones de Laboratorio de esta asignatura. También se configuró el protocolo SSH para poder utilizarlo en la comunicación con la misma durante el desarrollo y el volcado de datos y se

instalaron PHP y el servidor de MySQL para alojar la base de datos. A mayores, un obstáculo que se encontró durante el proceso fue que al ingresar a la aplicación en el servidor durante las primeras pruebas, no fue posible acceder a los experimentos, debido a errores no demasiado explicativos. Finalmente se concluyó que se debía a no utilizar una conexión HTTPS correctamente cifrada con un certificado firmado por una autoridad de confianza, ya que existe un cierto consenso entre los desarrolladores de los motores de los principales navegadores en la actualidad (Google (Chromium, en el que se basa la gran mayoría de navegadores de terceros), Mozilla (Firefox) y Apple (Safari)), por el que procuran reducir el uso de HTTP sin cifrar al máximo, y por tanto, el acceso al hardware periférico del equipo local, como lo es la cámara, se impide para conexiones de este tipo cuando el entorno no es de desarrollo.

- Sistemas Lineales: En esta asignatura introductoria a las señales y los sistemas se explican los fundamentos del procesado de señal. Operaciones como el filtrado, el muestreo o la convolución, que ha demostrado ser clave para las tecnologías con las que se logran los buenos resultados del sistema propuesto. Si bien el nivel de abstracción es muy superior en el caso de *Deep Learning* y más aún al utilizar la API de Keras, la base es la misma.

- Ingeniería de Protocolos: Como suele ocurrir en la Mención en Telemática, cualquier implementación que utilice protocolos en redes de comunicaciones tiene una relación implícita con esta asignatura.

El factor limitante principal ha sido el reducido tamaño del conjunto de datos, que no permite hacer pruebas específicas, como entrenar los modelos con muestras obtenidas mediante un solo tipo de experimento y analizar cuál es mejor; definir y entrenar un modelo solo para sujetos con gafas, etc. Por supuesto, cuando se utiliza un conjunto más grande, hay menos riesgo de sobreentrenamiento sobre todo cuando se utilizan redes más complejas, ya que se evidencia un mayor número de relaciones que el modelo se puede aprender y estas son más claras. Por tanto, también serían menos necesarios los regularizadores de kernel o las capas de normalización del lote o de *dropout*, que hacen más robusta la red frente al sobreentrenamiento a costa de ralentizar el aprendizaje; o la aumentación, que en ciertos casos mejora en mayor medida que la exactitud, la capacidad de generalización, que por supuesto, también se incrementa con el tamaño del conjunto.

Otro aspecto que mejoraría considerablemente con un conjunto más grande sería el de la evaluación. La validación cruzada es un mecanismo que funciona muy bien cuando se trabaja con conjuntos tan pequeños que la separación de una fracción para conformar el subconjunto de evaluación puede producir incertidumbre en la medición, al retirar muestras de un conjunto que ya es pequeño, y un sesgo, al estar el conjunto de evaluación inevitablemente relacionado con el de entrenamiento. Sin embargo, es un proceso muy lento, ya que requiere hacer tantos entrenamientos y evaluaciones como sujetos tenga el

conjunto, y por tanto, ocupa mucha memoria, imposibilitando su realización en muchos equipos personales. Cuando el conjunto es suficientemente grande, esto ya no ocurre. La segmentación típica en conjuntos de entrenamiento, validación y test proporciona un resultado correcto para el caso general de evaluación con sujetos que no aparecen en el conjunto de entrenamiento.

Una última conclusión que se puede extraer de este trabajo es una reducción del requisito de especialización, que tanto el *Deep Learning* como la inteligencia artificial en general plantean. Pues, al igual que un modelo de lenguaje puede ayudar a sus usuarios desarrollar tareas complejas sin necesidad de tener plenos conocimientos para ello, aunque sí básicos; se ha demostrado cómo sin contar con hardware especializado, es posible lograr resultados similares gracias al *Deep Learning*, de forma análoga a un caso mucho más grande que se planteó en la introducción: El sistema de cámaras Tesla Vision, con el que el fabricante de coches adquiere en tiempo real los datos que necesita su sistema de conducción autónoma, en reemplazo de la combinación de cámaras con un sensor LiDAR que utilizaban anteriormente.

7.2 Líneas futuras

Hasta este punto, se han mencionado en varias ocasiones hipotéticos desarrollos de este proyecto, debido a que es una prueba de concepto, y varias son las oportunidades de continuación y de mejora, que en cierto modo, están relacionadas entre sí, de forma que unas dependen o llevan a otras y viceversa.

La primera y más clara sería utilizar un conjunto más grande, a fin de mejorar la exactitud, la capacidad de generalización y hacer pruebas que permitan análisis más exhaustivos. Bien utilizando una base de datos de terceros que contenga información compatible con este problema, o bien con la aplicación de adquisición, cuyo desempeño probablemente mejoraría haciendo un rediseño teniendo en consideración las conclusiones obtenidas hasta este momento.

La segunda precisamente explora esta opción. La aplicación de adquisición, en este momento, tiene problemas de rendimiento y de escalabilidad severos. Esto se debe en cierto modo, a la forma en la que está desarrollada, pues resulta evidente la influencia de la asignatura de Tecnologías para Aplicaciones Web en la misma: Las tecnologías utilizadas son las aprendidas durante su transcurso. Como tal, la lógica se implementa en Javascript y PHP puros. Es decir, sin librerías ni *frameworks*, como Angular, React o Vue en el lado del cliente, o Laravel, Django, ASP.NET Core, Node.js Express, etc. En el lado del servidor. Todos ellos proporcionan un entorno de trabajo y un nivel de abstracción superior a los lenguajes sobre los que se implementan que facilitan mucho el mantenimiento y la escalabilidad de las aplicaciones, ya que simplifican el desarrollo, proporcionan mejor rendimiento al contar con parches y un equipo que los mantiene, y además son modulares. Esto permitiría desarrollar más eficientemente una mejor aplicación de adquisición, que a su vez produciría un mejor conjunto de entrenamiento.

Otra línea futura relacionada con la aplicación de adquisición y los *frameworks* de desarrollo web es una posible integración de las tuberías de entrenamiento de modelos propuestas en esta memoria con la interfaz de la aplicación de adquisición. Hasta este momento habían sido completamente disjuntos. Sin embargo, se podría tomar ventaja de las posibilidades que ofrecen los *frameworks* citados anteriormente, de ciertas APIs integradas en TensorFlow y de los avances en aprendizaje por transferencia mostrados en la definición del modelo regresor con input ocular con preprocesado basado en YuNet; para crear una automatización que, tras cada realización de experimentos, incorpore los datos adquiridos al modelo. Esto hace posible disponer de un modelo cuando sea necesario, y de poder reentrenarlo si así se requiere.

Reuniendo todas las líneas futuras anteriores, se llega a la necesidad de implementar un sistema de calibración con el que adaptar la tubería al caso de uso de cada usuario particular. Esto se podría hacer bien con una tubería de postprocesado en la que se defina una red neuronal adicional cuya entrada sean las coordenadas inferidas por los modelos regresores, y la salida, las mismas coordenadas corregidas; o bien, reentrenando el propio modelo regresor de *Eye-Tracking* elegido. En cualquiera de los dos casos es necesario adquirir datos del usuario para hacer los entrenamientos pertinentes antes de poner en marcha el sistema. Esto se podría hacer con la hipotética aplicación de adquisición modular integrada con la tubería del sistema de *Eye-Tracking*, explicada en la anterior línea futura. Todas las soluciones de *Eye-Tracking* sin hardware específico descritas en el Capítulo 2 implementan esta función de alguna manera, lo cuál es lógico, pues por muchos datos que se utilicen inicialmente para el entrenamiento, no es realista asumir que el modelo será capaz de generalizar lo suficiente para adaptarse a cualquier circunstancia. De hecho, cómo se vio en el Capítulo anterior, no es extraño que aparezcan sesgos de estimación. Por suerte, los sistemas de calibración permiten mitigarlos. Sin ir más lejos, la solución experimental de Apple integrada en iOS 18, durante su inicialización, muestra, para calibrarse, una pantalla muy similar al experimento 1 de la aplicación de adquisición de este proyecto.

Terminando con las líneas futuras que exploran posibles aspectos de mejora, cabe destacar la necesidad de procesar las coordenadas inferidas por los modelos para aportarles valor. Esto depende totalmente de la aplicación, pues, por ejemplo, para generar mapas de calor de atención visual, las librerías utilizadas actualmente en el sistema para su análisis serían más que suficiente. Sin embargo, para desarrollar una interfaz humano-computadora sería necesario utilizar APIs que probablemente no sean multiplataforma y que por tanto, resulte necesario estudiar cada caso de uso.

Por último y no por ello menos importante, es preciso desarrollar un programa que alimente con datos reales los modelos ya entrenados. Es decir, que en tiempo real obtenga fotogramas de la cámara, los procese mediante la tubería e infiera coordenadas, ya que, hasta este momento, el origen de los datos utilizados siempre ha sido un mismo conjunto etiquetado.

Bibliografía

[1] Tobii Dynavox. *What is eye-tracking?* Accedido en marzo de 2025. URL: https://www.tobiidynavox.com/pages/what-is-eye-tracking.

[2] M. Pluzyczka. "The First Hundred Years: A History of Eye Tracking as a Research Method". En: *Applied Linguistics Papers* 25 (2018).

[3] Karl R. Gegenfurtner. "The Interaction Between Vision and Eye Movements". En: *Perception* 45.12 (2016), págs. 1333-1357. DOI: 10.1177/0301006616657097.

[4] Santiago Nova y Blanca Navarro. *Corteza cerebral*. Accedido en marzo de 2025. 2023. URL: https://www.kenhub.com/es/library/anatomia-es/cerebro-es.

[5] *Tobii*. Accedido en marzo de 2025. URL: https://www.tobii.com/.

[6] *Sensores de visión para la detección y evaluación de objetos y escenas - IFM*. Accedido en marzo de 2025. URL: https://www.ifm.com/download/files/ifm-vision-sensors-industrial-imaging-ES/$file/ifm-vision-sensors-industrial-imaging-ES.pdf.

[7] *Tesla Vision Update - Tesla*. Accedido en marzo de 2025. URL: https://www.tesla.com/support/transitioning-tesla-vision.

[8] Raúl Barba Alonso. "Diseño e implementación de un sistema de captura de datos de entrenamiento para la realización de eye-tracking sin hardware específico". En: *Escuela Técnica Superior de Ingenieros de Telecomunicación - Universidad de Valladolid* (2021).

[9] *Python*. Accedido en marzo de 2025. URL: https://www.python.org/.

[10] *Project Jupyter*. Accedido en marzo de 2025. URL: https://jupyter.org/.

[11] *Introducción a TensorFlow*. Accedido en marzo de 2025. URL: https://www.tensorflow.org/learn.

[12] *Keras: Deep Learning for humans*. Accedido en marzo de 2025. URL: https://keras.io/.

[13] *CUDA: Free tools and training - Nvidia*. Accedido en marzo de 2025. URL: https://developer.nvidia.com/cuda-toolkit.

[14] *OpenCV - Open Computer Vision Library*. Accedido en marzo de 2025. URL: https://opencv.org/.

[15] *pandas documentation*. Versión del 20 de septiembre del 24. URL: https://pandas.pydata.org/docs/index.html.

[16] Emmett T. Cunningham y Paul Riordan-Eva. *Vaughan & Asbury's General Ophthalmology*. 18.ª ed. McGraw-Hill Medical, 2011. ISBN: 978-0-07-163420-5.

[17] Carlos Verges Rogero. *Campo visual*. Accedido en marzo de 2025. URL: http://www.neurocirugiacontemporanea.com/doku.php?id=campo_visual.

[18] *Types of eye movements - Tobii*. Accedido en marzo de 2025. URL: https://connect.tobii.com/s/article/types-of-eye-movements?language=en_US.

[19] *Smooth Pursuit Neck Torsion Test (SPNT) - Physiotutors*. Accedido en marzo de 2025. URL: https://www.physiotutors.com/es/wiki/smooth-pursuit-neck-torsion-test/.

[20] Ramtin Zargari Marandi et al. "Eye movement characteristics reflected fatigue development in both young and elderly individuals". En: *Scientific Reports* (2018). DOI: 10.1038/s41598-018-31577-1. URL: https://www.nature.com/articles/s41598-018-31577-1.

[21] Ivan A. Vajs et al. "Eye-Tracking Image Encoding: Autoencoders for the Crossing of Language Boundaries in Developmental Dyslexia Detection". En: *ResearchGate* (2023). URL: https://www.researchgate.net/publication/366912750_Eye-tracking_Image_Encoding_Autoencoders_for_the_Crossing_of_Language_Boundaries_in_Developmental_Dyslexia_Detection.

[22] Mirta Mikac. *The history behind eye tracking*. Accedido en febrero de 2025. URL: https://www.eyelogicsolutions.com/history-behind-eye-tracking/.

[23] *Eye Tracking Through History - EyeSee*. Accedido en marzo de 2025. URL: https://medium.com/@eyesee/eye-tracking-through-history-b2e5c7029443.

[24] *SR Research*. Accedido en marzo de 2025. URL: https://www.sr-research.com/.

[25] *IRISBOND EYE am Eye-Tracking*. Accedido en mayo de 2025. URL: https://www.irisbond.com/.

[26] Tobii. *Field Metrics and Data Quality Reports - Tobii Pro Spectrum and Fusion*. Informes internos Tobii Pro. 2023.

[27] Andrew Housholder et al. *Evaluating Accuracy of the Tobii Eye Tracker 5*. Inf. téc. Augustana College, 2022. URL: https://www.researchgate.net/publication/359339342_Evaluating_Accuracy_of_the_Tobii_Eye_Tracker_5.

[28] *Eyeware*. Accedido en marzo de 2025. URL: https://eyeware.tech/.

[29] Martyna Pietrzak et al. *RealEye Webcam Eye-Tracking for Desktops, Laptops and Smartphones Technology White Paper*. Inf. téc. RealEye, 2023.

[30] Alexandra Papoutsaki et al. "WebGazer: Scalable Webcam Eye Tracking Using User Interactions". En: *Brown University* (2023).

[31] *Apple*. Accedido en marzo de 2025. URL: https://support.apple.com/es-es/guide/iphone/iph66057d0f6/ios.

[32] *OpenCV Developer Site*. Accedido en marzo de 2025. URL: http://code.opencv.org.

[33] Phil Nelson. *OpenCV Face Detection: Cascade Classifier vs. YuNet*. Accedido en marzo de 2025. URL: https://opencv.org/blog/opencv-face-detection-cascade-classifier-vs-yunet/.

[34] *Ley Orgánica 3/2018, de Protección de Datos Personales y garantía de los derechos digitales*. Accedido en marzo de 2025. URL: https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673.

[35] Lucía Olmos Vela. "Soluciones Deep Learning para el aprendizaje con datos desbalanceados". En: *Escuela de Ingeniería Informática de Segovia - Universidad de Valladolid* (2024).

[36] Afshine Amidi y Shervine Amidi. "VIP Cheatsheet : Machine Learning Tips". En: *Standford University* (2018). URL: https://stanford.edu/~shervine/teaching/cs-229/.

[37] *7 Best Deep Learning Frameworks You Should Know in 2025*. Accedido en marzo de 2025. URL: https://www.geeksforgeeks.org/deep-learning-frameworks/.

[38] Shifeng Zhang et al. "FaceBoxes: A CPU Real-time Face Detector with High Accuracy". En: (2017). arXiv: arXiv:1708.05234 [cs.CV].

[39] Valentin Bazarevsky et al. "BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs". En: (2019). arXiv: arXiv:1907.05047 [cs.CV].

[40] Shiqi Yu. *libfacedetection*. 2023. URL: https://github.com/ShiqiYu/libfacedetection.

[41] Linzaer. *Ultra-Light-Fast-Generic-Face-Detector-1MB*. 2023. URL: https://github.com/Linzaer/Ultra-Light-Fast-Generic-Face-Detector-1MB.

[42] Wei Wu, Hanyang Peng y Shiqi Yu. "YuNet: A Tiny Millisecond-level Face Detector". En: *Southern University of Science and Technology, Shenzhen* (2023).

[43] *MTCNN Documentation*. Accedido en marzo de 2025. URL: https://mtcnn.readthedocs.io/en/latest/.

[44] Adrian Rosebrock. *Local Binary Patterns with Python & OpenCV*. Accedido en marzo de 2025. URL: https://pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/.

[45] Ana Huamán. *Cascade Classifier - OpenCV Docs*. Accedido en marzo de 2025. URL: https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html.

[46] *Why Train-Test Split is a Big Deal (and How to Use It)*. Accedido en marzo de 2025. URL: https://toxigon.com/train-test-split-importance.

[47] Anum Fatima. *How To Choose Train Validation and Test Sets For Your Model?* Accedido en marzo de 2025. URL: https://machinemindscape.com/how-to-choose-our-train-validation-and-test-sets.

[48] *Resizing and Rescaling Images with OpenCV*. Accedido en marzo de 2025. URL: https://opencv.org/blog/resizing-and-rescaling-images-with-opencv/.

[49] Kinder Chen. *Hidden Layer Activation Functions*. Accedido en marzo de 2025. URL: https://kinder-chen.medium.com/hidden-layer-activation-functions-6fd65489ed25.

[50] Richmond Alake. *Loss Functions in Machine Learning Explained*. Accedido el 11 junio 2024. 2023. URL: https://www.datacamp.com/tutorial/loss-function-in-machine-learning.

[51] *MobileNet, MobileNetV2, and MobileNetV3 - Keras 3 API documentation*. Accedido en marzo de 2025. URL: https://keras.io/api/applications/mobilenet/.

[52] Sándor Csősz et al. "Insect morphometry is reproducible under average investigation standards". En: *Ecology and Evolution* 11 (2020), págs. 1-13. DOI: 10.1002/ece3.7075.

[53] *Accuracy (trueness and precision) of measurement methods and results - Part 1: General principles and definitions.* BS ISO 5725-1, p.1. ISO, 1994.

[54] Isaí Villa, Ivan Perez Olguin y Consuelo Fernández Gaxiola. "Reducción de Fallas en Máquina de Fusión Mediante un Análisis de Tolerancias". En: ago. de 2012, págs. 67-77. ISBN: 978-607-8262-00-7.

[55] *Media Devices API - MDN WebDocs*. Accedido en marzo de 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices.

[56] *Secure contexts - Security on the web - MDN WebDocs*. Accedido en marzo de 2025. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Secure_Contexts.

Apéndice A

Cuadernos de Jupyter I: Pruebas de detección facial, detección ocular y recorte ocular heurístico

En este Apéndice se recogen los flujos de código utilizados para implementar las diferentes funciones de detección facial y ocular explicadas a lo largo de la Memoria. Inicialmente se mostrarán los pasos seguidos para los clasificadores de Haar Cascade, para terminar con la secuencia que utiliza la red convolucional YuNet.

Detección facial utilizando OpenCV: Haar Cascade Frontalface

Este cuaderno contiene el flujo de código necesario para cargar un .csv extraído desde saqqara, leer sus datos, cargar imágenes a partir de estos, y de detectar y recortar el rostro en las imágenes

Primero los imports:

In [ ]:
```
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import clear_output
```

Se elige un dataset correcto cargando en esta variable la cadena que devolvió el script durante su ejecución y una fila con la que trabajar

In [133…
```
DATASET_CODE = 20250320204408
```

In [134…
```
row = 500
```

Cargamos el contenido del .csv en un dataframe de pandas

In [135…
```
df = pd.read_csv('datasets/' + str(DATASET_CODE) + '.csv')
```

In [136…
```
df
```

Out[136…

	imagen	click_x	click_y	pos_x	pos_y	ancho_cm
0	capturas/2024-9-24_11_52_14_1_QFWJ/im_3706.png	876	419	875	416	30.2
1	capturas/2024-9-24_11_52_14_1_QFWJ/im_3707.png	1125	852	1124	845	30.2
2	capturas/2024-9-24_11_52_14_1_QFWJ/im_3708.png	756	480	756	491	30.2
3	capturas/2024-9-24_11_52_14_1_QFWJ/im_3709.png	374	571	375	564	30.2
4	capturas/2024-9-24_11_52_14_1_QFWJ/im_3710.png	1428	770	1427	769	30.2
...	...	...	...	...	...	...
1375	capturas/2025-3-9_22_20_19_3_MPVE/im_6235.png	4	2	4	2	34.9
1376	capturas/2025-3-9_22_20_19_3_MPVE/im_6236.png	3	1	3	1	34.9
1377	capturas/2025-3-9_22_20_19_3_MPVE/im_6237.png	5	3	3	3	34.9
1378	capturas/2025-3-9_22_20_19_3_MPVE/im_6238.png	0	3	0	3	34.9
1379	capturas/2025-3-9_22_20_19_3_MPVE/im_6239.png	1	1	3	3	34.9

1380 rows × 33 columns

Con el atributo loc del dataframe podemos obtener bien recogidos los datos de la fila con la que se va a trabajar

In [137…

```
df.loc[row]
```

Out[137…
```
imagen                capturas/2024-10-25_19_9_1_1_MPVE/im_5285.png
click_x                                                         976
click_y                                                         920
pos_x                                                           975
pos_y                                                           919
ancho_cm                                                       34.9
alto_cm                                                        21.8
ancho_px                                                       1324
alto_px                                                        1018
diagonal_Pulgadas                                              16.2
fecha                                                    2024-10-25
tiempo                                             0 days 19:09:01
codigoExperimento                                                1
codigoUsuario                                                 MPVE
edad                                                            22
sexo                                                     Masculino
color_ojos                                                Marrones
gafas                                                           No
lentillas                                                       No
puntuacion                                                   80.09
tiempo_contador                                                  4
tiempo_click                                            00:14:210
nombreCamara          Cámara FaceTime HD (integrada)
posicion                                                  Estándar
anchoImagenCam                                                1280
altoImagenCam                                                  720
aspectRatio                                                1.77778
frameRate                                                     30.0
exposureTime                                                  -1.0
colorTemp                                                       -1
brightness                                                    -1.0
sharpness                                                     -1.0
saturation                                                    -1.0
Name: 500, dtype: object
```

Este es el estándar para ubicar las imágenes en los datasets de este proyecto:

In [138…
```
image_path = 'datasets/' + str(DATASET_CODE) + '_imageset/' + df['imagen'
print(image_path)

img = cv2.imread(image_path)
```

```
datasets/20250320204408_imageset/capturas/2024-10-25_19_9_1_1_MPVE/im_528
5.png
```

In [139…
```
img.shape
```

Out[139…
```
(720, 1280, 3)
```

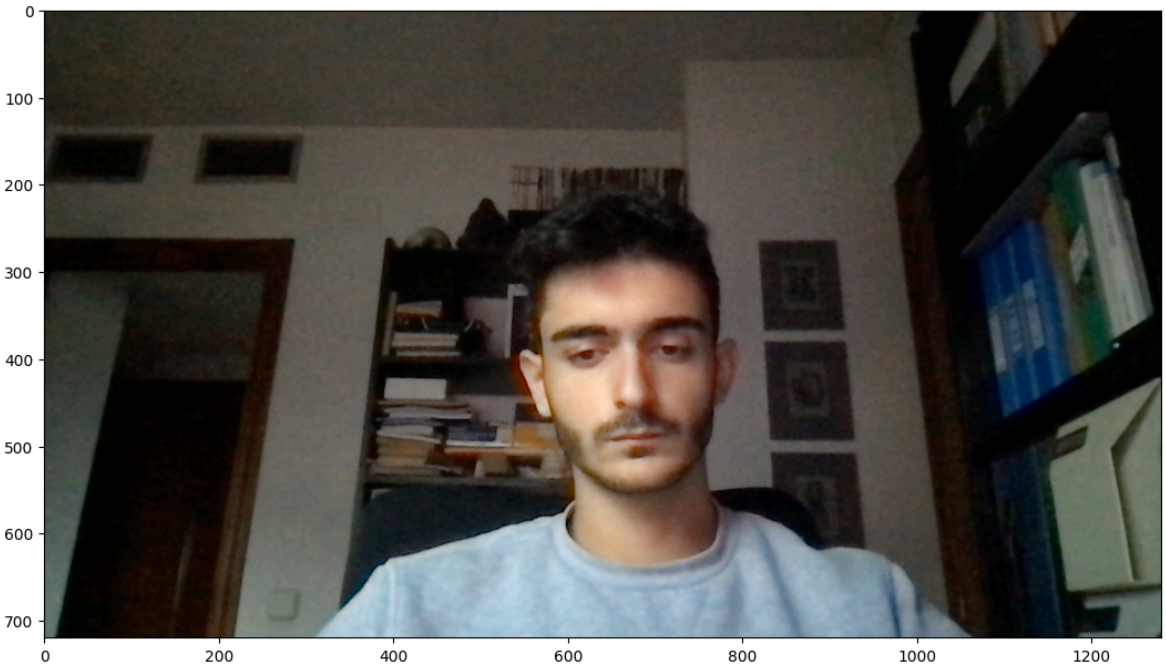
Las imágenes cargadas como objetos de OpenCV son arrays tridimensionales (un color por cada dimensión) ordenadas de la siguiente manera: Azul, verde, rojo (BGR). Para convertirlos a espacios más típicos, se usa el método cvtColor(). Requerimos de la imágen bidimensional (escala de grises) para la detección de la cara, y opcionalmente la imágen RGB para visualizarla en color

In [140…
```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

In [141…
```
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

In [142…
```
plt.figure(figsize=(13,8))
plt.imshow(rgb_img)
```

Out[142… `<matplotlib.image.AxesImage at 0x7f0cb4210bb0>`



In [143…
```
def face_crop_resize(original_img, new_wh, interpolation_type=0, crop_upp

    if interpolation_type == 0:
        interpolation = cv2.INTER_NEAREST
    elif interpolation_type == 1:
        interpolation = cv2.INTER_LINEAR
    elif interpolation_type == 2:
        interpolation = cv2.INTER_AREA
    elif interpolation_type == 3:
        interpolation = cv2.INTER_CUBIC
    elif interpolation_type == 4:
        interpolation = cv2.INTER_LANCZOS4
    else:
        interpolation = cv2.INTER_NEAREST

    resized_img = cv2.resize(original_img, (new_wh, new_wh), interpolatio
    half_img = resized_img[:new_wh//2]
    return resized_img if not crop_upper_half else half_img
```

Para detectar las caras vamos a utilizar un clasificador en cascada de OpenCV [1].

Los clasificadores en cascada son modelos que contienen clasificadores "débiles" (que por sí solo no es capaz de clasificar nada más que una característica de Haar) uno tras otro para formar uno único más potente. OpenCV proporciona tanto métodos para entrenar clasificadores como modelos preentrenados, que vienen con la instalación por defecto, en ficheros .xml. El entrenamiento de estos se hace con datasets enormes que contengan imágenes positivas (con el objeto a detectar) y negativas.

Al no ser la detección facial el objeto de este proyecto, no hay imágenes negativas en los datasets que se generan, como para entrenar un clasificador, por lo que optamos por cargar un modelo preentrenado cargando el .xml adecuado:

In [144…
```
FACE_CASCADE = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade
```

Para ejecutar la detección se usa el método detectMultiScale() de la clase del clasificador

Este típicamente recibe una imagen en escala de grises para hacer la detección.

`scaleFactor` nada tiene que ver con el tamaño del rectángulo que delimita el área de la cara. Es el valor por el que se reduce la imagen cada vez que avanza un paso en el modelo. Valores más bajos mejorarán la precisión, valores más altos acelerarán la búsqueda. En este caso no hay prisa, asi que dejaremos uno bajo. Por defecto viene en 1.1

`minNeighbors` controla la sensibilidad del algoritmo, estableciendo el umbral del número de ventanas contiguas en las que la clasificación deba resultar positiva para considerar válida una deteción. Con un número mayor, será más exigente, lo que puede reducir falsos positivos pero pasando por alto algunas caras válidas. Por defecto viene en 5

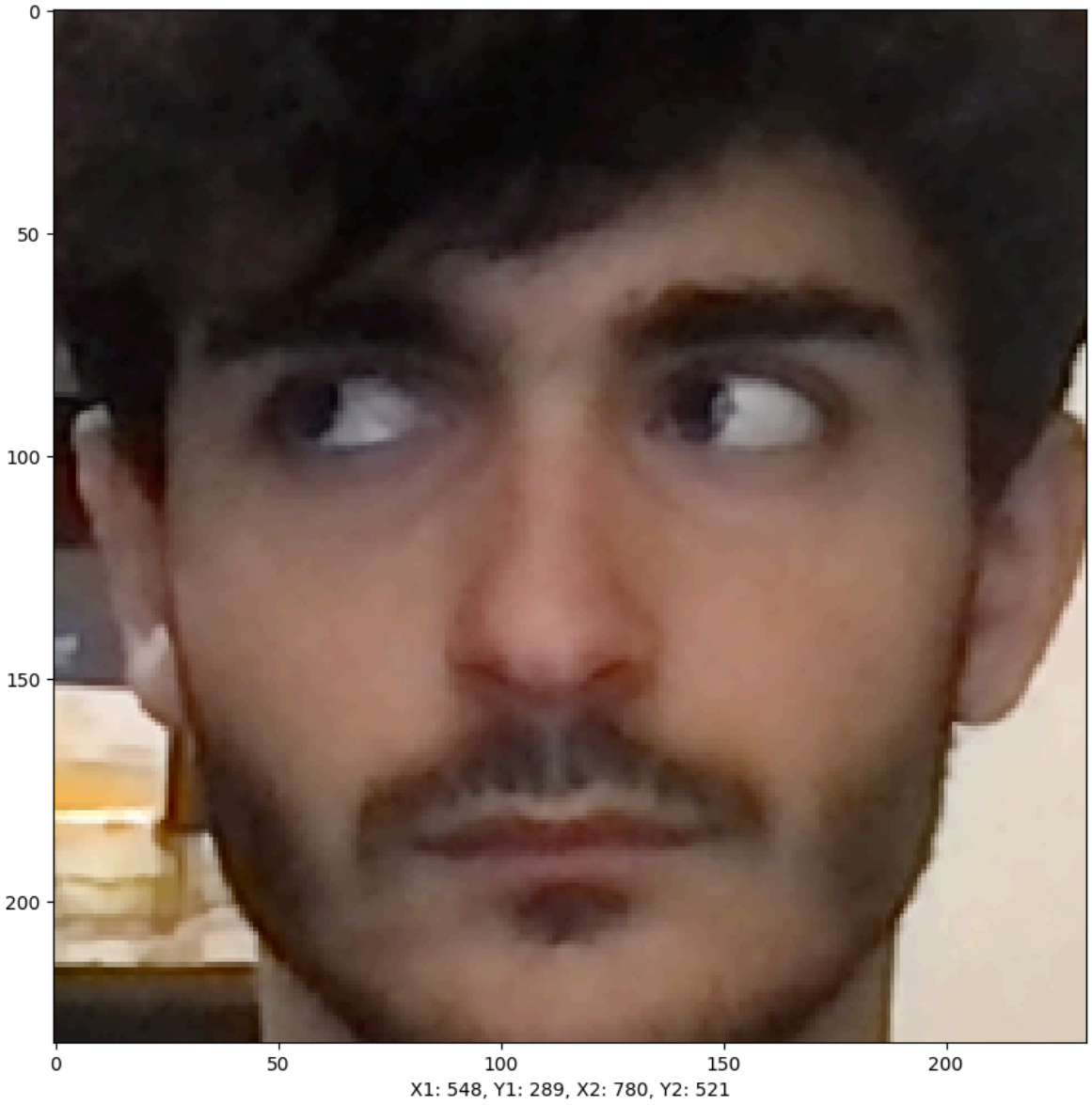
`minSize` tiene un nombre bastante descripivo. Ajustarlo bien puede acotar mucho el espacio de búsqueda sin sacrificar muestras, lo que se traduce en una reducción drástica del tiempo de detección sin ningún inconveniente añadido.

In [152…
```
def detect_crop_face(gray_img, upper_half_only = False):
    global FACE_CASCADE
    face = FACE_CASCADE.detectMultiScale(gray_img, scaleFactor=1.1, minNe
    if len(face) == 0: raise Exception("Error 0 de detección facial: No s
    if len(face) > 1: raise Exception("Error 1 de detección facial: Se en

    face = np.reshape(face,4)

    x1 = face[0]
    x2 = face[0] + face[2]
    y1 = face[1]
    y2 = int(face[1] + face[3]/2) if upper_half_only else face[1] + face[

    face_img = gray_img[y1:y2, x1:x2]

    return x1, x2, y1, y2, face_img
```

In [153…
```
x1, x2, y1, y2, face_rgb_img = detect_crop_face(rgb_img, upper_half_only=
```

In [154…
```
plt.figure(figsize=(10,10))
plt.imshow(face_rgb_img, cmap='gray')
plt.xlabel(f"X1: {x1}, Y1: {y1}, X2: {x2}, Y2: {y2}")
```

Out[154…
```
Text(0.5, 0, 'X1: 548, Y1: 289, X2: 780, Y2: 521')
```



In [155…
```python
def get_image(row, df):
    global DATASET_CODE
    bgr_img = cv2.imread('datasets/' + str(DATASET_CODE) + '_imageset/' +
    rgb_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2RGB)
    gray_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2GRAY)
    return bgr_img, rgb_img, gray_img
```

In [156…
```python
gray_imgs = []
rgb_imgs = []

for i in range(len(df)):
    try:
        bgr_img, rgb_img, gray_img = get_image(i, df)
        gray_imgs.append(gray_img)
        rgb_imgs.append(rgb_img)
    except Exception as e:
        print(f"Error en la imagen {i}: {e}")
        continue

clear_output(wait=True)
print(f"Total de imágenes procesadas: {len(gray_imgs)}")
```

```
Total de imágenes procesadas: 1365
```

In [ ]:

```
image_error_count = 0
face_detection_error_count = 0

image_sizes = []
face_sizes = []

i = 0

for gray_img in gray_imgs:
    i += 1

    try:
        x1, y1, x2, y2, face_gray_img = detect_crop_face(rgb_img)
    except:
        face_detection_error_count += 1
        continue

print("Haar Cascade:")
print(f"Errores de detección facial: {face_detection_error_count}")
print(f"Total de imágenes procesadas: {i}")
```

```
Haar Cascade:
Errores de detección facial: 117
Total de imágenes procesadas: 1365
```

Detección ocular utilizando OpenCV: Haar Cascade

Este cuaderno contiene el flujo de código necesario para cargar un .csv extraído desde saqqara, leer sus datos, cargar imágenes a partir de estos, y de detectar y recortar los ojos en las imágenes. Para ello, es un paso intermedio detectar y recortar la cara primero, ya que de no hacerlo, el clasificador en cuestión podría reconocer patrones y dar falsos positivos en zonas que no son la cara y por tanto, no contienen ojos.

Esta versión calcula el punto medio del mismo rectángulo que engloba los dos ojos en la propuesta 1, para luego trazar un recorte de proporciones fijas

El procedimiento de detección facial replica al mostrado en el anterior cuaderno.

In [ ]:
```
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from random import randint
from IPython.display import clear_output
```

In [91]:
```
dataset_code = str(20250320204408)
```

In [92]:
```
df = pd.read_csv('datasets/' + dataset_code + '.csv')
```

In [93]:
```
df
```

Out[93]:

	imagen	click_x	click_y	pos_x	pos_y	ancho_cm
0	capturas/2024-9-24_11_52_14_1_QFWJ/im_3706.png	876	419	875	416	30.2
1	capturas/2024-9-24_11_52_14_1_QFWJ/im_3707.png	1125	852	1124	845	30.2
2	capturas/2024-9-24_11_52_14_1_QFWJ/im_3708.png	756	480	756	491	30.2
3	capturas/2024-9-24_11_52_14_1_QFWJ/im_3709.png	374	571	375	564	30.2
4	capturas/2024-9-24_11_52_14_1_QFWJ/im_3710.png	1428	770	1427	769	30.2
...	...	...	...	...	...	...
1375	capturas/2025-3-9_22_20_19_3_MPVE/im_6235.png	4	2	4	2	34.9
1376	capturas/2025-3-9_22_20_19_3_MPVE/im_6236.png	3	1	3	1	34.9
1377	capturas/2025-3-9_22_20_19_3_MPVE/im_6237.png	5	3	3	3	34.9
1378	capturas/2025-3-9_22_20_19_3_MPVE/im_6238.png	0	3	0	3	34.9
1379	capturas/2025-3-9_22_20_19_3_MPVE/im_6239.png	1	1	3	3	34.9

1380 rows × 33 columns

In [94]:
```
row = 554 # Funciona bien a pesar de que falla en un ojo
```

In [95]:
```
row = 1362 # Ejemplo perfecto
```

In [96]:
```
df.loc[row]
```

Out[96]:
```
imagen                capturas/2025-2-9_21_20_41_3_MPVE/im_6222.png
click_x                                                           3
click_y                                                           1
pos_x                                                             3
pos_y                                                             1
ancho_cm                                                       34.9
alto_cm                                                        21.8
ancho_px                                                       1792
alto_px                                                        1120
diagonal_Pulgadas                                              16.2
fecha                                                    2025-02-09
tiempo                                             0 days 21:20:41
codigoExperimento                                                 3
codigoUsuario                                                  MPVE
edad                                                             22
sexo                                                       Masculino
color_ojos                                                  Marrones
gafas                                                             No
lentillas                                                         No
puntuacion                                                       0.0
tiempo_contador                                                    0
tiempo_click                                                00:10:434
nombreCamara          Cámara FaceTime HD (integrada)
posicion                                                    Estándar
anchoImagenCam                                                 1280
altoImagenCam                                                   720
aspectRatio                                                 1.77778
frameRate                                                      30.0
exposureTime                                                   -1.0
colorTemp                                                        -1
brightness                                                     -1.0
sharpness                                                      -1.0
saturation                                                     -1.0
Name: 1362, dtype: object
```

In [97]:
```
image_path = 'datasets/' + dataset_code + '_imageset/' + df['imagen'][row
print(image_path)

img = cv2.imread(image_path)
```

```
datasets/20250320204408_imageset/capturas/2025-2-9_21_20_41_3_MPVE/im_622
2.png
```

In [98]:
```
img.shape
```

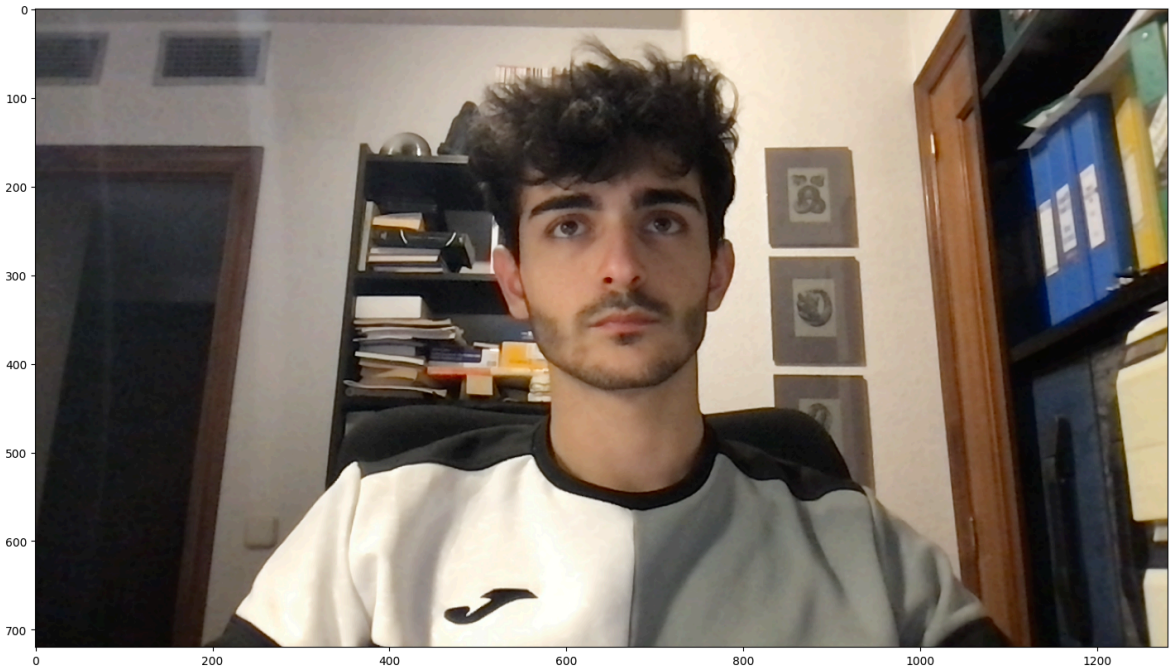
Out[98]:
```
(720, 1280, 3)
```

In [99]:
```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

In [100…
```
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

In [101…
```
plt.figure(figsize=(20,10))
plt.imshow(rgb_img)
```

Out[101…
```
<matplotlib.image.AxesImage at 0x7efe782538e0>
```



In [102…
```
face_classifier = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcasc
```

In [103…
```
face = face_classifier.detectMultiScale(
    gray_img, scaleFactor=1.1, minNeighbors=10, minSize=(40, 40)
)
```

In [104…
```
face
```

Out[104… `array([[516, 131, 286, 286]], dtype=int32)`

In [105…
```
face = np.reshape(face,4) # Es un array bidimensional con una dimensión v
print(face)
```

```
[516 131 286 286]
```

En este caso, para reducir los falsos positivos, la recta horizontal inferior (y2) se va a colocar en la mitad de la cara (dividiendo entre dos `face[3]`, que contendría el alto del recorte). De esta forma, nos quedamos solo con la mitad superior de la cara, que al fin y al cabo es la que contiene los ojos. La imágen resultante tiene en cualquier caso una relación de aspecto 2:1

In [106…
```
x1 = face[0]
x2 = face[0] + face[2]
y1 = face[1]
y2 = int(face[1] + face[3]/2)

# Recortar ojos
# y1 = int(face[1]+face[1]*0.2)
# y2 = int(face[1]+face[3]/2)
```

In [107…
```
# Posiciones normalizadas del centro de la foto
x_pos = face[0] + face[2]/2
y_pos = face[1] + face[3]/2

# Se podrían utilizar los datos de resolución del dataset tb
```

```
x_pos_norm = x_pos/len(img[0])
y_pos_norm = y_pos/len(img)
```

In [108…
```
crop_img = cv2.cvtColor(img[y1:y2, x1:x2], cv2.COLOR_BGR2RGB)
```

In [109…
```
plt.figure(figsize=(20,10))
plt.imshow(crop_img)
plt.xlabel(f"XY face center position: ({x_pos},{y_pos})\nNormalized: ({x_
```

Out[109…
```
Text(0.5, 0, 'XY face center position: (659.0,274.0)\nNormalized: (0.514
84375,0.38055555555555554)')
```

Procedemos a detectar los ojos. Para ello cargamos otro clasificador en cascada, en este caso con otro .xml adecuado para este fin

In [110…
```
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_
```

Al igual que el clasificador facial, este trabaja con imágemes en escala de grises, por lo que la recuperaremos con cvtColor().

In [111…
```
gray_crop_img = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)
```

In [112…
```
eyes = eye_cascade.detectMultiScale(gray_crop_img)
```

In [113…
```
eyes
```

Out[113…
```
array([[ 60,  92,  47,  47],
       [168,  86,  51,  51]], dtype=int32)
```

El clasificador ocular no es tan bueno como por ejemplo, el facial. Puede detectar otras facciones, como las fosas nasales, como ojos. Por suerte, estos falsos positivos suelen tener un tamaño muy inferior a los ojos reales. Para eso vamos a reordenar las coordenadas de los ojos dentro del array de mayor a menor tamaño del cuadrado que lo contiene. Así, en caso de detectar más de dos, los dos primeros serán los correctos con casi total seguridad.

In [114…
```
eyes = eyes[eyes[:, 2].argsort()[::-1]]
```

In [115…
```
eyes
```

Out[115…
```
array([[168,  86,  51,  51],
       [ 60,  92,  47,  47]], dtype=int32)
```

Nos quedamos con ellos:

In [116…
```
eye_0 = eyes[0]
eye_1 = eyes[1]
```

Los datos que almacena este clasificador corresponden a las coordenadas de esquina superior izquierda de los cuadrados que contienen los ojos y el ancho y alto de los mismos. Para obtener las rectas de un rectángulo que contenga a ambos ojos hay que tener en cuenta que no necesariamente se detectará primero el izquierdo y luego el derecho, por lo que habrá que buscar las rectas que delimitan cada ojo más "exteriores" posibles. Esto se puede hacer con los métodos min() y max() de Python, que devuelven el más alto o bajo argumento que se les pasa:

In [117…
```
eyes_x1 = min(eye_0[0], eye_1[0])
eyes_x2 = max(eye_0[0] + eye_0[2], eye_1[0] + eye_1[2])
eyes_y1 = min(eye_0[1], eye_1[1])
eyes_y2 = max(eye_0[1] + eye_0[3], eye_1[1] + eye_1[3])
```

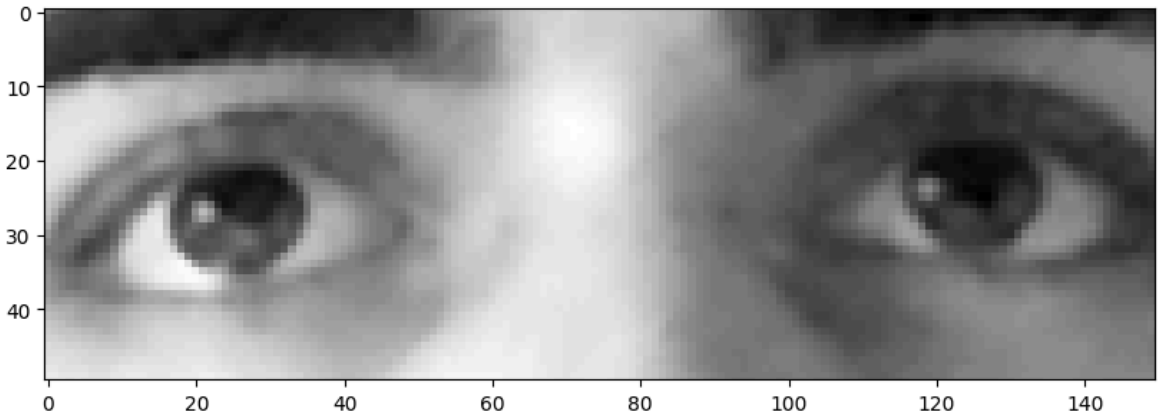
In [118…
```
eye_mid_x = (min(eye_0[0], eye_1[0]) + max(eye_0[1] + eye_0[3], eye_1[1]
eye_mid_y = (min(eye_0[1], eye_1[1]))/2
```

In [119…
```
eye_crop_img = crop_img[eyes_y1:eyes_y2, eyes_x1:eyes_x2]
```

In [120…
```
eye_crop_img = cv2.cvtColor(eye_crop_img, cv2.COLOR_RGB2GRAY)
```

In [121…
```
if((eyes_x2-eyes_x1)/(eyes_y2-eyes_y1) < 2 ):
    print("Error")
else:
    eye_crop_img = cv2.resize(eye_crop_img, (150, 50))
    pass
plt.figure(figsize=(10,5))
plt.imshow(eye_crop_img, cmap='gray')
```

Out[121…
```
<matplotlib.image.AxesImage at 0x7efe4c713400>
```

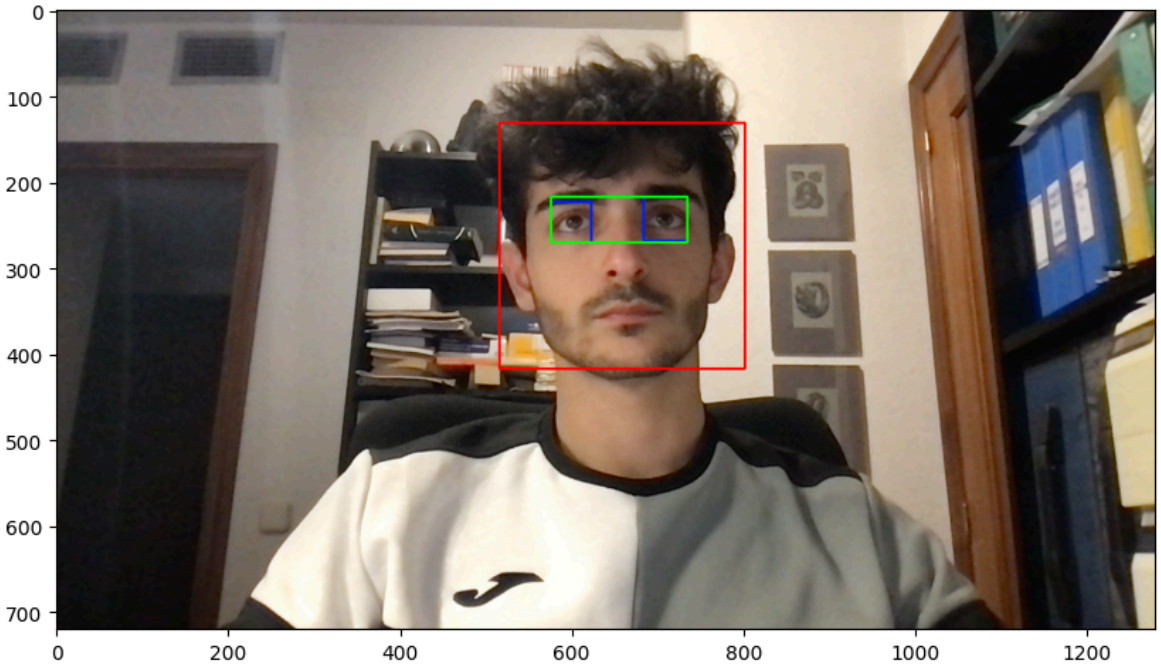


Este es el resultado de la detección de los ojos visto sobre la imagen original

In [122…
```
y2_full = face[1] + face[3]
```

In [123…
```
plt.figure(figsize=(10,10))
face_rectangle = cv2.rectangle(rgb_img, (x1, y1), (x2, y2_full), (255, 0,
eye_0_rectangle = cv2.rectangle(face_rectangle, (x1 + eye_0[0], y1 + eye_
eye_1_rectangle = cv2.rectangle(eye_0_rectangle, (x1 + eye_1[0], y1 + eye
eyes_rectangle = cv2.rectangle(eye_1_rectangle, (x1 + eyes_x1, y1 + eyes_
plt.imshow(eyes_rectangle)
```

Out[123…
```
<matplotlib.image.AxesImage at 0x7efe9bc3d9c0>
```



Detección facial y ocular utilizando OpenCV: YuNet

Este cuaderno contiene el flujo de código necesario para cargar un .csv extraído desde saqqara, leer sus datos, cargar imágenes a partir de estos, y de detectar y recortar el rostro en las imágenes

Primero los imports:

In [ ]:
```
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
from IPython.display import clear_output
```

Se elige un dataset correcto cargando en esta variable la cadena que devolvió el script durante su ejecución y una fila con la que trabajar

In [35]:
```
DATASET_CODE = 20250320204408
```

In [36]:
```
row = 500
```

Cargamos el contenido del .csv en un dataframe de pandas

In [37]:
```
df = pd.read_csv('datasets/' + str(DATASET_CODE) + '.csv')
```

In [38]:
```
df
```

Out[38]:

	imagen	click_x	click_y	pos_x	pos_y	ancho_cm
0	capturas/2024-9-24_11_52_14_1_QFWJ/im_3706.png	876	419	875	416	30.2
1	capturas/2024-9-24_11_52_14_1_QFWJ/im_3707.png	1125	852	1124	845	30.2
2	capturas/2024-9-24_11_52_14_1_QFWJ/im_3708.png	756	480	756	491	30.2
3	capturas/2024-9-24_11_52_14_1_QFWJ/im_3709.png	374	571	375	564	30.2
4	capturas/2024-9-24_11_52_14_1_QFWJ/im_3710.png	1428	770	1427	769	30.2
...	...	...	...	...	...	...
1375	capturas/2025-3-9_22_20_19_3_MPVE/im_6235.png	4	2	4	2	34.9
1376	capturas/2025-3-9_22_20_19_3_MPVE/im_6236.png	3	1	3	1	34.9
1377	capturas/2025-3-9_22_20_19_3_MPVE/im_6237.png	5	3	3	3	34.9
1378	capturas/2025-3-9_22_20_19_3_MPVE/im_6238.png	0	3	0	3	34.9
1379	capturas/2025-3-9_22_20_19_3_MPVE/im_6239.png	1	1	3	3	34.9

1380 rows × 33 columns

Con el atributo loc del dataframe podemos obtener bien recogidos los datos de la fila con la que se va a trabajar

In [39]:
```
df.loc[row]
```

Out[39]:
```
imagen                capturas/2024-10-25_19_9_1_1_MPVE/im_5285.png
click_x                                                         976
click_y                                                         920
pos_x                                                           975
pos_y                                                           919
ancho_cm                                                       34.9
alto_cm                                                        21.8
ancho_px                                                       1324
alto_px                                                        1018
diagonal_Pulgadas                                              16.2
fecha                                                    2024-10-25
tiempo                                             0 days 19:09:01
codigoExperimento                                                1
codigoUsuario                                                 MPVE
edad                                                            22
sexo                                                     Masculino
color_ojos                                                Marrones
gafas                                                           No
lentillas                                                       No
puntuacion                                                   80.09
tiempo_contador                                                  4
tiempo_click                                              00:14:210
nombreCamara          Cámara FaceTime HD (integrada)
posicion                                                  Estándar
anchoImagenCam                                                1280
altoImagenCam                                                  720
aspectRatio                                                1.77778
frameRate                                                     30.0
exposureTime                                                  -1.0
colorTemp                                                       -1
brightness                                                    -1.0
sharpness                                                     -1.0
saturation                                                    -1.0
Name: 500, dtype: object
```

Este es el estándar para ubicar las imágenes en los datasets de este proyecto:

In [40]:
```
image_path = 'datasets/' + str(DATASET_CODE) + '_imageset/' + df['imagen'
print(image_path)

img = cv2.imread(image_path)
```

```
datasets/20250320204408_imageset/capturas/2024-10-25_19_9_1_1_MPVE/im_528
5.png
```

In [41]:
```
img.shape
```

Out[41]:
```
(720, 1280, 3)
```

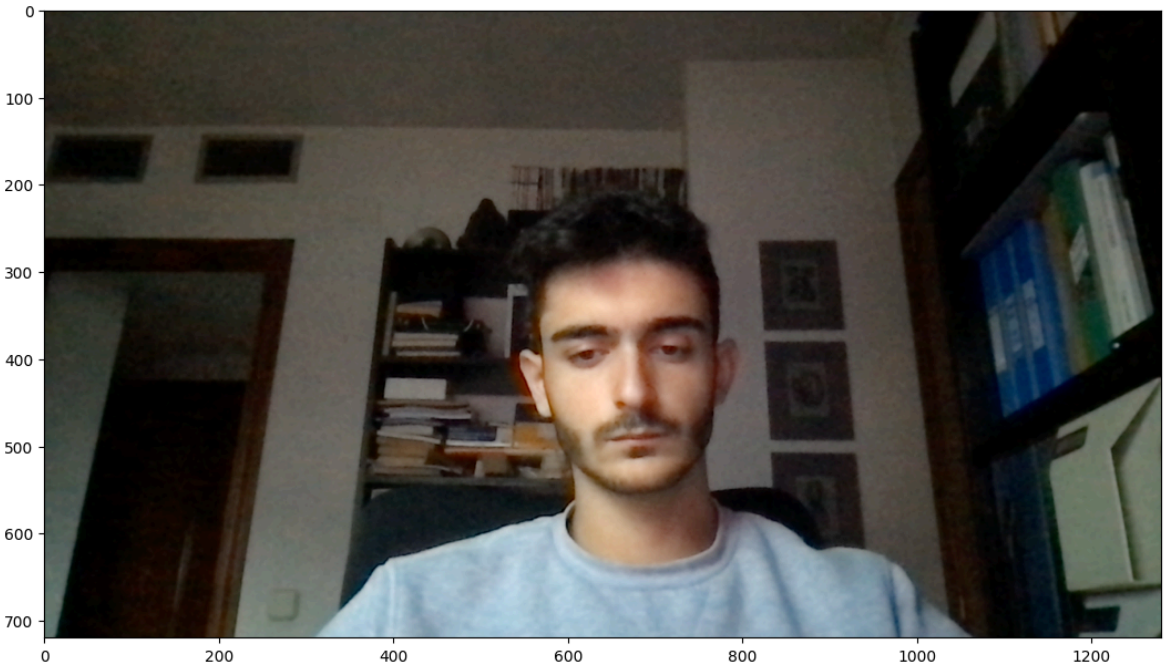
Las imágenes cargadas como objetos de OpenCV son arrays tridimensionales (un color por cada dimensión) ordenadas de la siguiente manera: Azul, verde, rojo (BGR). Para convertirlos a espacios más típicos, se usa el método cvtColor(). Requerimos de la imágen bidimensional (escala de grises) para la detección de la cara, y opcionalmente la imágen RGB para visualizarla en color

In [42]:
```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

In [43]:
```
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

In [44]:
```
plt.figure(figsize=(13,8))
plt.imshow(rgb_img)
```

Out[44]:
```
<matplotlib.image.AxesImage at 0x7fd811f22320>
```



In [45]:
```
def face_crop_resize(original_img, new_wh, interpolation_type=0, crop_upp

    if interpolation_type == 0:
        interpolation = cv2.INTER_NEAREST
    elif interpolation_type == 1:
        interpolation = cv2.INTER_LINEAR
    elif interpolation_type == 2:
        interpolation = cv2.INTER_AREA
    elif interpolation_type == 3:
        interpolation = cv2.INTER_CUBIC
    elif interpolation_type == 4:
        interpolation = cv2.INTER_LANCZOS4
    else:
        interpolation = cv2.INTER_NEAREST

    resized_img = cv2.resize(original_img, (new_wh, new_wh), interpolatio
    half_img = resized_img[:new_wh//2]
    return resized_img if not crop_upper_half else half_img
```

A continuación se procede a la instanciación y configuración del modelo para comenzar a hacer detecciones. Las versiones de OpenCV lanzadas desde el 2023 en adelante incorporan clases para trabajar con YuNet.

Los parámetros clave de este método, que integra el modelo con algo de preprocesado, son:

- `input_size`: Corresponde con la resolución de las imágenes de entrada. El método `detect` acepta entradas de diferentes tamaños siempre que coincidan con el valor fijado en este parámetro, aunque el modelo tiene pesos para una

serie de resoluciones, de entre las cuales, el método elige la más próxima a la definida en este parámetro. Este comportamiento se observa con frecuencia en redes convolucionales preentrenadas, como MobileNet.

- `score_threshold`: En clasificación, lo habitual es que el modelo proporcione un valor de probabilidad normalizado para cada clase que maneja, siendo aquella o aquellas que superan cierto umbral las que se consideran inferencia. Aquí se aplica un concepto similar.
- `nms_threshold`: Este parámetro controla el límite de supresión de no máximos, que evita hacer detecciónes contiguas del mismo rostro.
- `top_k`: Corresponde al número máximo de detecciones que el modelo puede detectar en la imágen. En este caso es 1, puesto que el sistema no está pensado funcionar con varios rostros simultáneamente. Esta única detección siempre es la de mayor confianza.

In [46]:
```
FACE_DETECTOR = cv2.FaceDetectorYN.create(
    model="face_detection_yunet_2023mar.onnx",
    config="",
    input_size=(320, 320),  # Tamaño de entrada
    score_threshold=0.9,    # Umbral de confianza
    nms_threshold=0.3,      # Umbral NMS
    top_k=1            # Número máximo de detecciones
)
```

In [47]:
```
imh, imw, _ = rgb_img.shape
FACE_DETECTOR.setInputSize((imw, imh))
nuse, faces = FACE_DETECTOR.detect(img)
```

In [48]:
```
x, y, w, h, le_x, le_y, re_x, re_y = (faces[0][:8]).astype(int)
```

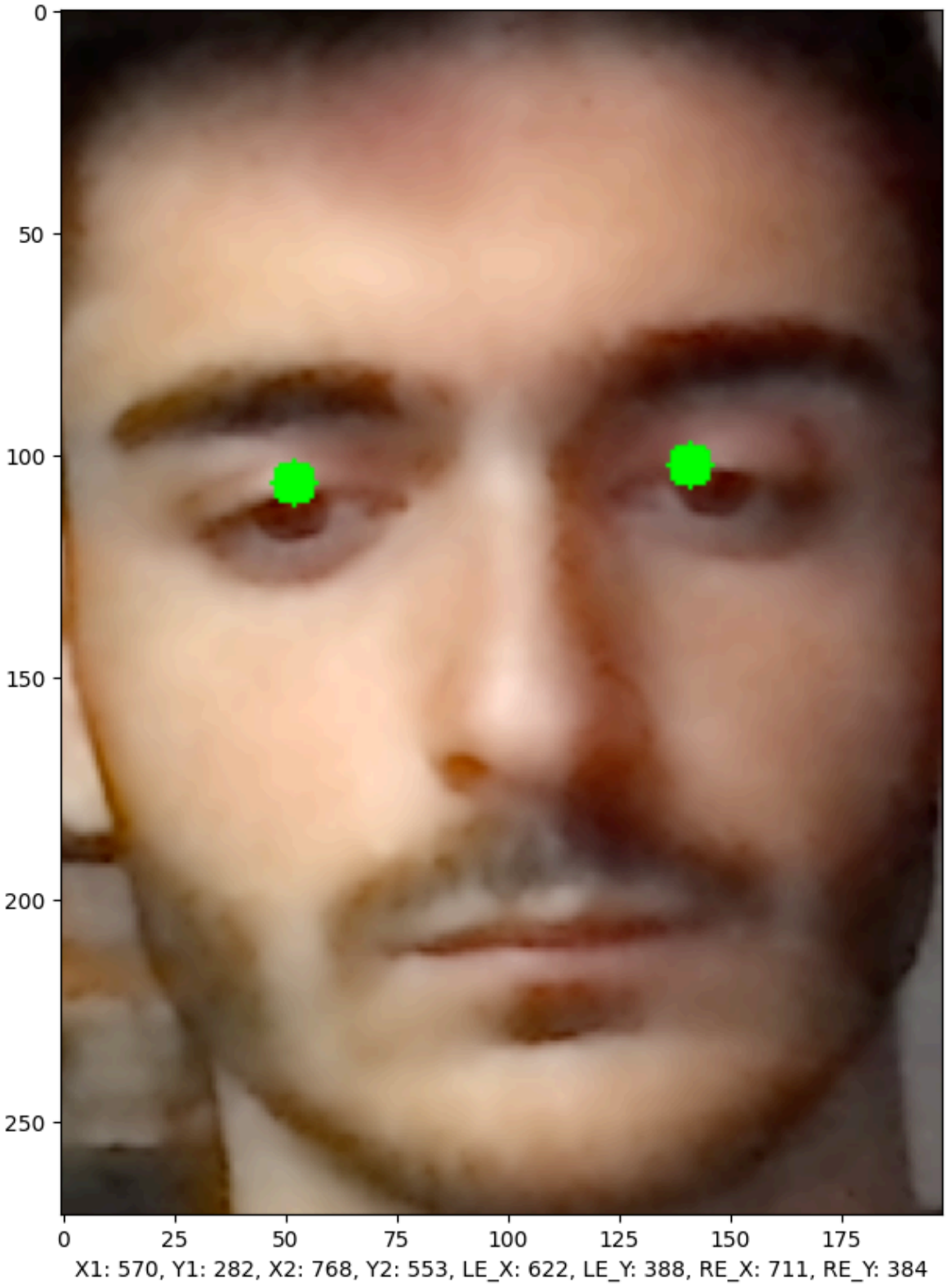
In [49]:
```
rgb_img_circles = rgb_img.copy()
cv2.circle(rgb_img_circles, (int(le_x), int(le_y)), 5, (0, 255, 0), -1)
cv2.circle(rgb_img_circles, (int(re_x), int(re_y)), 5, (0, 255, 0), -1)
```

Out[49]:
```
array([[[ 50,  45,  33],
        [ 50,  45,  33],
        [ 50,  45,  33],
        ...,
        [  0,   2,   0],
        [  0,   2,   0],
        [  0,   2,   0]],

       [[ 50,  45,  33],
        [ 50,  45,  33],
        [ 49,  44,  31],
        ...,
        [  1,   2,   0],
        [  1,   2,   0],
        [  0,   1,   0]],

       [[ 48,  43,  29],
        [ 48,  43,  29],
        [ 48,  43,  29],
        ...,
        [  1,   1,   0],
        [  0,   1,   0],
        [  0,   1,   0]],

       ...,

       [[ 99, 121, 111],
        [ 99, 121, 111],
        [ 98, 119, 110],
        ...,
        [ 52,  66,  49],
        [ 42,  57,  38],
        [ 35,  52,  30]],

       [[ 98, 119, 110],
        [ 98, 119, 110],
        [ 98, 119, 110],
        ...,
        [ 40,  57,  39],
        [ 33,  51,  30],
        [ 26,  45,  24]],

       [[ 98, 119, 110],
        [ 98, 119, 110],
        [ 98, 119, 110],
        ...,
        [ 38,  56,  38],
        [ 33,  51,  30],
        [ 29,  49,  26]]], dtype=uint8)
```

In [50]:
```
face_img = rgb_img[int(y):int(y + h), int(x):int(x + w)]
```

In [51]:
```
face_circles_img = rgb_img_circles[int(y):int(y + h), int(x):int(x + w)]
plt.figure(figsize=(10,10))
plt.imshow(face_circles_img)
plt.xlabel(f"X1: {x}, Y1: {y}, X2: {x+w}, Y2: {y+h}, LE_X: {le_x}, LE_Y:
```

Out[51]:
```
Text(0.5, 0, 'X1: 570, Y1: 282, X2: 768, Y2: 553, LE_X: 622, LE_Y: 388,
RE_X: 711, RE_Y: 384')
```



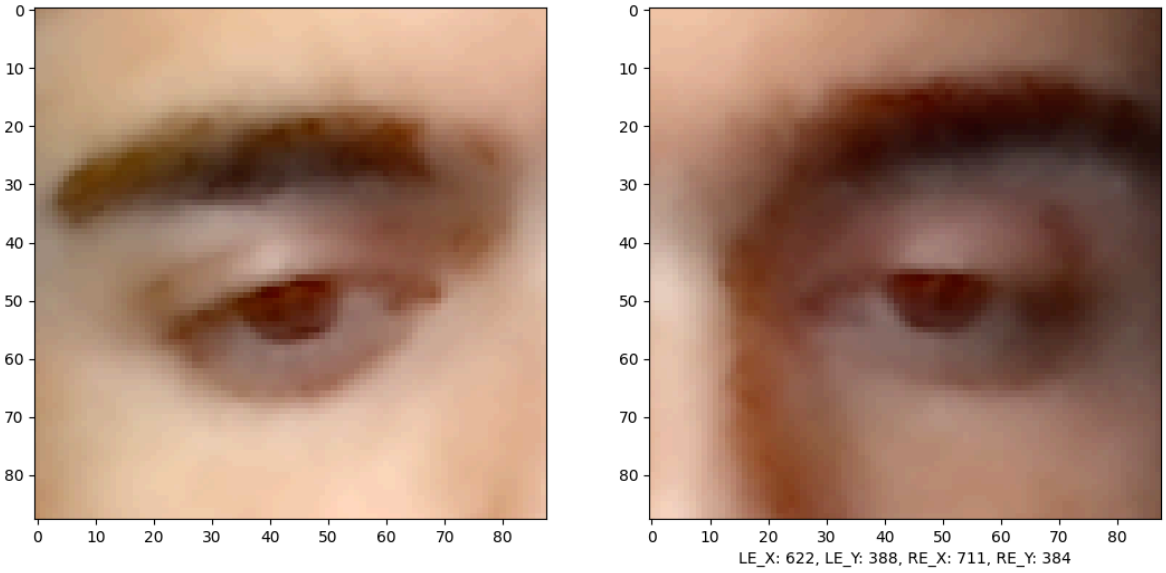
In [52]:
```
eyes_radius = abs(re_x - le_x) // 2
```

In [53]:
```
le_bb_x1 = int(le_x - eyes_radius)
le_bb_x2 = int(le_x + eyes_radius)
le_bb_y1 = int(le_y - eyes_radius)
le_bb_y2 = int(le_y + eyes_radius)
re_bb_x1 = int(re_x - eyes_radius)
re_bb_x2 = int(re_x + eyes_radius)
re_bb_y1 = int(re_y - eyes_radius)
re_bb_y2 = int(re_y + eyes_radius)
```

In [54]:
```
left_eye_img = rgb_img[le_bb_y1:le_bb_y2, le_bb_x1:le_bb_x2]
right_eye_img = rgb_img[re_bb_y1:re_bb_y2, re_bb_x1:re_bb_x2]
```

In [55]:
```
plt.figure(figsize=(13,10))
plt.subplot(1, 2, 1)
plt.imshow(left_eye_img)
plt.subplot(1, 2, 2)
plt.imshow(right_eye_img)
plt.xlabel(f"LE_X: {le_x}, LE_Y: {le_y}, RE_X: {re_x}, RE_Y: {re_y}")
```

Out[55]:
```
Text(0.5, 0, 'LE_X: 622, LE_Y: 388, RE_X: 711, RE_Y: 384')
```



In [56]:
```
def detect_face_crop_eyes_yn(rgb_img):
    imh, imw, _ = rgb_img.shape
    global FACE_DETECTOR
    if FACE_DETECTOR.getInputSize() != (imw, imh):
        FACE_DETECTOR.setInputSize((imw, imh))
    _, faces = FACE_DETECTOR.detect(rgb_img)
    if faces is None:
        raise Exception("Error 0 de detección facial: No se encontró")

    x, y, w, h, le_x, le_y, re_x, re_y = (faces[0][:8]).astype(int)

    eyes_radius = abs(re_x - le_x) // 2

    le_bb_x1 = int(le_x - eyes_radius)
    le_bb_x2 = int(le_x + eyes_radius)
    le_bb_y1 = int(le_y - eyes_radius)
    le_bb_y2 = int(le_y + eyes_radius)
    re_bb_x1 = int(re_x - eyes_radius)
    re_bb_x2 = int(re_x + eyes_radius)
    re_bb_y1 = int(re_y - eyes_radius)
    re_bb_y2 = int(re_y + eyes_radius)

    left_eye_img = rgb_img[le_bb_y1:le_bb_y2, le_bb_x1:le_bb_x2]
    right_eye_img = rgb_img[re_bb_y1:re_bb_y2, re_bb_x1:re_bb_x2]

    return x, y, (x+w), (y+h), left_eye_img, right_eye_img
```

In [57]:
```
x1, y1, x2, y2, left_eye_imgs, right_eye_imgs = detect_face_crop_eyes_yn(
```

Estadísticas

In [59]:
```python
def get_image(row, df):
    global DATASET_CODE
    bgr_img = cv2.imread('datasets/' + str(DATASET_CODE) + '_imageset/' +
    rgb_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2RGB)
    gray_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2GRAY)
    return bgr_img, rgb_img, gray_img
```

In [60]:
```python
gray_imgs = []
rgb_imgs = []

for i in range(len(df)):
    try:
        bgr_img, rgb_img, gray_img = get_image(i, df)
        gray_imgs.append(gray_img)
        rgb_imgs.append(rgb_img)
    except Exception as e:
        print(f"Error en la imagen {i}: {e}")
        continue

clear_output(wait=True)
print(f"Total images processed: {len(gray_imgs)}")
```

```
Total images processed: 1365
```

In [61]:
```python
image_error_count = 0
face_detection_error_count = 0
eye_detection_error_count = 0

image_sizes = []
face_sizes = []

i = 0

for rgb_img in rgb_imgs:
    i += 1

    try:
        x1, y1, x2, y2, left_eye_img, right_eye_img = detect_face_crop_ey

        if left_eye_img is None or right_eye_img is None:
            eye_detection_error_count += 1
            continue

        image_sizes.append(left_eye_img.shape)
        face_sizes.append((x2-x1, y2-y1))

    except:
        face_detection_error_count += 1
        continue

print("YuNet:")
print(f"Errores de detección facial: {face_detection_error_count}")
print(f"Errores de detección de ojos: {eye_detection_error_count}")
print(f"Total de imágenes procesadas: {i}")
```

```
YuNet:
Errores de detección facial: 108
Errores de detección de ojos: 0
Total de imágenes procesadas: 1365
```

Se muestran también las estadísticas de los tamaños de las regiones de interés inferidas por el mecanismo para utilizarlas como referencia a la hora de definir arquitecturas:

In [62]:
```
pdf = pd.DataFrame(image_sizes)
```

In [63]:
```
pdf.columns = ['height', 'width', 'channels']
pdf.describe()
```

Out[63]:

	height	width	channels
count	1257.000000	1257.000000	1257.0
mean	103.546539	103.546539	3.0
std	18.415618	18.415618	0.0
min	68.000000	68.000000	3.0
25%	88.000000	88.000000	3.0
50%	102.000000	102.000000	3.0
75%	114.000000	114.000000	3.0
max	158.000000	158.000000	3.0

Apéndice B

Cuadernos de Jupyter II: Redes neuronales

En este Apéndice se muestra el cuaderno de Jupyter en el que se ha desarrollado, seleccionado y evaluado –mediante los dos mecanismos propuestos– el Modelo III del proyecto. El mismo que utiliza YuNet para las funciones de detección, y MobileNet entre las capas de su arquitectura; y que logró los mejores resultados del análisis.

Para evitar redundancia, se han omitido los cuadernos correspondientes al resto de modelos, por ser demasiado similares a este, y considerar el presente como la mejor representación del trabajo realizado. El procedimiento es el mismo en cualquier caso, variando la arquitectura del modelo y los métodos específicos de cada tubería de preprocesado.

Red convolucional para Eye-Tracking sin hardware dedicado con preprocesado para recorte ocular

Propuesta 2

Esta versión es la primera en implementar la colección de métodos to wapos. implementa detección ocular (por hacer) mediante el método de recorte por coordenadas precalculadas desarrollado en `heuristic_eyes_crop_test.ipynb`. Entre el recorte facial y el recorte ocular, implementa un paso intermedio de reescalado a una resolución fija, de forma que los recortes oculares finales tienen resolución constante, que hay que hacer coincidir con la forma de la capa de entrada de la red neuronal

In [99]:
```
NOTEBOOK_VERSION = 'AI_E_V2_2DD_YN96'
```

Inputs

- `I` : Imagenes. Tensor bidimensional con imágenes de dimensión dependiente de proceso, en escala de grises normalizadas.
- `F` : Tensor unidimensional con coordenadas normalizadas de la posición la cara detectada en el sistema de coordenadas descrito por los píxeles de la imágen original completa.
- `D` : Tensor unidimensional con los tamaños en centímetros de las pantallas utilizadas.
- `G` : Tensor binario que etiqueta con valor 1 aquellos casos en los que el usuario de la captura lleve gafas.

Outputs

- `B` : *Ball*. Tensor unidimensional con posiciones normalizadas de la bola

Preparación del entorno

Importamos los paquetes necesarios para el correcto funcionamiento del script del cuaderno. Conforme avance el cuaderno y vayan siendo necesarios, se declararán los métodos desarrollados en el cuaderno `df_methods.ipynb` debidamente adaptados.

In [100…
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
import cv2
import pandas as pd
import numpy as np
```

```
import random
import math
from IPython.display import clear_output
import seaborn as sns
import gc
import os
import copy
```

In [101…
```
tf.config.list_physical_devices()
```

Out[101…
```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
 PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU'),
 PhysicalDevice(name='/physical_device:GPU:1', device_type='GPU'),
 PhysicalDevice(name='/physical_device:GPU:2', device_type='GPU'),
 PhysicalDevice(name='/physical_device:GPU:3', device_type='GPU')]
```

Ajuste de memoria dinámico en GPUs compatibles con CUDA

In [102…
```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)
```

In [103…
```
os.environ['TF_GPU_ALLOCATOR'] = 'cuda_malloc_async'
```

Carga de datos

Los conjuntos de datos se obtienen de *saqqara* mediante un script desarrollado para ese fin. El formato de estos permite identificarlos por un código numérico formado por la fecha y hora al momento de la ejecución del script. Se almacena en una variable global, para que los métodos que requieran acceder a los ficheros puedan leerla:

In [104…
```
DATASET_CODE = 20250320204408
```

Procedemos a leer el fichero .csv con el siguiente método de `df_methods.ipynb`:

In [105…
```
def get_df():
    global DATASET_CODE
    df = pd.read_csv('datasets/' + str(DATASET_CODE) + '.csv')
    print(f"Read (rows, columns): {df.shape}")
    return df
```

In [106…
```
df = get_df()
```

```
Read (rows, columns): (1380, 33)
```

In [107…
```
df
```

Out[107…

	imagen	click_x	click_y	pos_x	pos_y	ancho_cm	alto_cr
0	capturas/2024-9-24_11_52_14_1_QFWJ/im_3706.png	876	419	875	416	30.2	19.
1	capturas/2024-9-24_11_52_14_1_QFWJ/im_3707.png	1125	852	1124	845	30.2	19.
2	capturas/2024-9-24_11_52_14_1_QFWJ/im_3708.png	756	480	756	491	30.2	19.
3	capturas/2024-9-24_11_52_14_1_QFWJ/im_3709.png	374	571	375	564	30.2	19.
4	capturas/2024-9-24_11_52_14_1_QFWJ/im_3710.png	1428	770	1427	769	30.2	19.
…	…	…	…	…	…	…	
1375	capturas/2025-3-9_22_20_19_3_MPVE/im_6235.png	4	2	4	2	34.9	21.
1376	capturas/2025-3-9_22_20_19_3_MPVE/im_6236.png	3	1	3	1	34.9	21.
1377	capturas/2025-3-9_22_20_19_3_MPVE/im_6237.png	5	3	3	3	34.9	21.
1378	capturas/2025-3-9_22_20_19_3_MPVE/im_6238.png	0	3	0	3	34.9	21.
1379	capturas/2025-3-9_22_20_19_3_MPVE/im_6239.png	1	1	3	3	34.9	21.

1380 rows × 33 columns

Leer las imágenes de este *dataframe* es la operación más compleja para los datos que maneja (acceder a cualquiera de los demás es trivial). Se define para ello el siguiente método:

In [108…

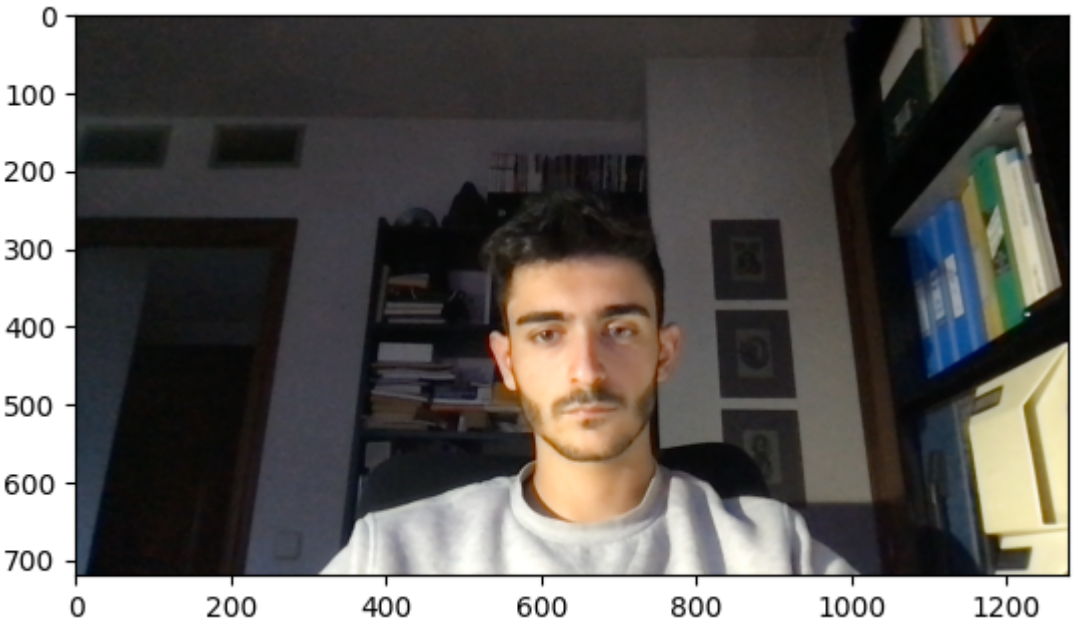
```
def get_image(row, df):
    global DATASET_CODE
    bgr_img = cv2.imread('datasets/' + str(DATASET_CODE) + '_imageset/' + df['im
    rgb_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2RGB)
    gray_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2GRAY)
    return bgr_img, rgb_img, gray_img
```

In [109…

```
_, rgb_img, _ = get_image(590, df)
plt.imshow(rgb_img)
```

Out[109…

```
<matplotlib.image.AxesImage at 0x7fdf33580700>
```



Comprobación de datos visual

Con este método se comprueban todos los usuarios con los que se trabaja:

In [110…

```
def show_users(df):
    users = df['codigoUsuario'].unique()

    # Number of users
    num_users = len(users)

    num_columns = 3
    num_rows = (num_users+num_columns-1)//num_columns

    # Create subplots
    fig, axes = plt.subplots(num_rows, num_columns, figsize=(15, 5*num_rows)) #

    axes = axes.flatten()

    for ax, user in zip(axes, users):
        num_rows = len(df[df['codigoUsuario'] == user].index)
        row = df[df['codigoUsuario'] == user].index[2]
        _, rgb_img, _ = get_image(row, df)
        ax.imshow(rgb_img)
        ax.set_title(f"{user} ({num_rows})")
        ax.axis('off')  # Hide axes
```

Por razones de privacidad, no se mostrará la salida del método.

In [111…

```
def check_user_images(df, user_code):
    while True:
        try:
            df_temp = df[df['codigoUsuario'] == user_code]
            row = random.choice(df_temp.index)
            _, rgb_img, _ = get_image(row, df_temp)
            plt.imshow(rgb_img)
            break
```

```
        except:
            continue
```

In [112…

```
check_user_images(df, 'MPVE')
```

Separación por tipos de experimento

La versión de la aplicación web desplegada en saqqara a noviembre de 2024 (que es la que se ha usado para el proyecto) tiene dos tipos de experimento:

- **Seguimiento ocular de un objeto en pantalla**: A este grupo corresponden los experimentos 1 y 2. Estos toman varias capturas a lo largo de la realización que contienen, entre otras cosas, imágenes y coordenadas en píxeles, lo cuál es en esencia, lo que se va a analizar y procesar en este cuaderno.

- Generación de situaciones anómalas mediante instrucciones: De este tipo es el experimento 3, que proporciona instrucciones al usuario para desviar la cabeza del centro de la pantalla o apartar la mirada de la misma. Se consideran situaciones anómalas porque se alejan de lo que sería objeto de inferencia de los modelos entrenados con datos generados con experimentos del primer grupo. Es decir, no son caras frente a la cámara mirando a diferentes puntos de la pantalla. De hecho, las coordenadas utilizadas en este experimento no son en píxeles. Se rigen por un sistema de coordenadas de 6x6 definido en los documentos del TFG. Su análisis permite contemplar estas posibilidades.

Por todo esto, es necesario separar los dataframes en función del tipo de experimento.

In [113…

```
def group_df_by_type(df):
    df12 = df[df['codigoExperimento'] != 3]
    df3 = df[df['codigoExperimento'] == 3]
    print(f"df12 | df3 (rows, columns): {df12.shape} | {df3.shape}")
    return df12, df3
```

In [114…
```
df12, df3 = group_df_by_type(df)
```

```
df12 | df3 (rows, columns): (1317, 33) | (63, 33)
```

In [115…
```
df12
```

Out[115…

	imagen	click_x	click_y	pos_x	pos_y	ancho_cm	alto_cr
0	capturas/2024-9-24_11_52_14_1_QFWJ/im_3706.png	876	419	875	416	30.2	19.
1	capturas/2024-9-24_11_52_14_1_QFWJ/im_3707.png	1125	852	1124	845	30.2	19.
2	capturas/2024-9-24_11_52_14_1_QFWJ/im_3708.png	756	480	756	491	30.2	19.
3	capturas/2024-9-24_11_52_14_1_QFWJ/im_3709.png	374	571	375	564	30.2	19.
4	capturas/2024-9-24_11_52_14_1_QFWJ/im_3710.png	1428	770	1427	769	30.2	19.
…	…	…	…	…	…	…	
1345	capturas/2025-1-6_17_57_27_2_QXYT/im_6198.png	895	704	874	724	34.5	19.
1346	capturas/2025-1-6_17_57_27_2_QXYT/im_6199.png	776	748	780	753	34.5	19.
1347	capturas/2025-1-6_17_57_27_2_QXYT/im_6200.png	676	736	685	736	34.5	19.
1348	capturas/2025-1-6_17_57_27_2_QXYT/im_6201.png	960	648	954	653	34.5	19.
1349	capturas/2025-1-6_17_57_27_2_QXYT/im_6202.png	1013	522	1008	550	34.5	19.

1317 rows × 33 columns

In [116…
```
df3
```

Out[116…

	imagen	click_x	click_y	pos_x	pos_y	ancho_cm	alto_cm
249	capturas/2024-9-25_14_24_28_3_JZYT/im_3981.png	3	1	3	1	99.0	99.
250	capturas/2024-9-25_14_24_28_3_JZYT/im_3982.png	4	3	4	3	99.0	99.
251	capturas/2024-9-25_14_24_28_3_JZYT/im_3983.png	3	1	3	3	99.0	99.
285	capturas/2024-10-5_14_48_33_3_OOWZ/im_4991.png	5	3	3	3	34.5	19.
286	capturas/2024-10-5_14_48_33_3_OOWZ/im_4992.png	5	3	3	3	34.5	19.
…	…	…	…	…	…	…	.
1375	capturas/2025-3-9_22_20_19_3_MPVE/im_6235.png	4	2	4	2	34.9	21.
1376	capturas/2025-3-9_22_20_19_3_MPVE/im_6236.png	3	1	3	1	34.9	21.
1377	capturas/2025-3-9_22_20_19_3_MPVE/im_6237.png	5	3	3	3	34.9	21.
1378	capturas/2025-3-9_22_20_19_3_MPVE/im_6238.png	0	3	0	3	34.9	21.
1379	capturas/2025-3-9_22_20_19_3_MPVE/im_6239.png	1	1	3	3	34.9	21.

63 rows × 33 columns

Organización en conjuntos de test, validación y entrenamiento

El proceso al que se someterán los datos más adelante para pasar del formato en el que se almacenan en el sistema de ficheros a aquel que se aloja en memoria y se utiliza en el entrenamiento y la evaluación de la red neuronal descarta toda aquella información no relevante para este fin. Si bien es el caso de los códigos de usuario; pues no sería ni útil ni deseable ni factible que la red los conociera, pero a la hora de separar los conjuntos de entrenamiento, validación y test, la mejor forma de hacerlo es en función de ellos. Así se podría validar y evaluar el modelo con adquisiciones de usuarios que no ha "visto" durante su entrenamiento para comprobar su capacidad de generalizar.

Primero desordenamos el *dataframe* completo para evitar cualquier tipo de autocorrelación o patrones en el flujo de datos con el que se entrenará la red neuronal.

In [117…

```
def df_shuffle(df, reset_index=True):
    return df.sample(frac=1).reset_index(drop=reset_index)
    # https://www.geeksforgeeks.org/pandas-how-to-shuffle-a-dataframe-rows/
```

In [118…
```
df12 = df_shuffle(df12.copy())
```

Ahora se van a agrupar en los subconjuntos de validación, test y entrenamiento aleatoriamente. De esta forma se puede simular el funcionamiento de una versión reentrenada mediante Transfer Learning en el equipo de un usuario, ya que se va a evaluar con muestras que no ha visto, de usuarios que sí "conoce".

El experimento 2 produce muestras contiguas no iguales, pero sí similares. El experimento 1 tiene total aleatoriedad, con lo que será este con el que se conforme el conjunto de evaluación.

In [119…
```
def train_test_sort_random(df, test_frac=0.2, valid_frac=0.2, random_state=None)
    # Separa el conjunto de test y validación aleatoriamente
    test_df = df[df['codigoExperimento']==1].sample(frac=test_frac, random_state
    remaining_df = df.drop(test_df.index).reset_index(drop=True)

    # Separa el conjunto de validación
    valid_df = remaining_df.sample(frac=valid_frac/(1-test_frac), random_state=r

    # El resto es el conjunto de entrenamiento
    train_df = remaining_df.drop(valid_df.index).reset_index(drop=True)

    return train_df, valid_df, test_df
```

In [120…
```
df12_train, df12_valid, df12_test = train_test_sort_random(df12, test_frac=0.15,
```

In [121…
```
df12_test
```

Out[121…

	imagen	click_x	click_y	pos_x	pos_y	ancho_cm	alto_cm
0	capturas/2024-10-23_18_50_24_1_TSLK/im_5076.png	516	279	517	283	52.2	29.4
1	capturas/2025-1-6_17_56_0_1_QXYT/im_6168.png	494	544	491	543	34.5	19.4
2	capturas/2024-10-25_19_9_1_1_MPVE/im_5295.png	738	143	740	142	34.9	21.8
3	capturas/2024-10-25_19_12_31_1_ROOT/im_5337.png	233	418	231	417	34.9	21.8
4	capturas/2024-11-16_15_17_32_1_MPVE/im_5649.png	1027	800	1028	799	34.9	21.8
...	...	...	...	...	...	...	..
100	capturas/2025-1-6_17_49_6_1_CWSQ/im_6098.png	777	443	768	412	34.5	19.4
101	capturas/2024-9-24_11_54_16_1_QFWJ/im_3731.png	300	193	300	193	30.2	19.6
102	capturas/2024-9-24_18_30_39_1_QFWJ/im_3859.png	763	203	759	202	30.2	19.6
103	capturas/2024-9-24_18_27_2_1_QFWJ/im_3827.png	1206	735	1206	735	30.2	19.6
104	capturas/2024-12-25_17_12_10_1_PAUM/im_5782.png	615	753	615	754	34.9	21.8

105 rows × 33 columns

In [122…

```
df12_valid
```

Out[122...

	imagen	click_x	click_y	pos_x	pos_y	ancho_cm	alto_cm
0	capturas/2024-10-25_18_58_23_1_ROOT/im_5192.png	897	560	896	509	34.9	21.8
1	capturas/2024-9-24_11_57_23_2_QFWJ/im_3749.png	864	520	870	525	30.2	19.6
2	capturas/2024-10-25_19_7_46_1_MPVE/im_5278.png	319	769	320	769	34.9	21.8
3	capturas/2024-12-25_17_27_55_1_LIDJ/im_5919.png	919	572	921	571	34.9	21.8
4	capturas/2024-11-23_18_25_55_1_MPVE/im_5715.png	1031	788	1033	787	34.9	21.8
...	...	...	...	...	...	...	...
209	capturas/2024-11-23_18_31_6_2_MPVE/im_5759.png	139	742	141	732	34.9	21.8
210	capturas/2024-11-6_12_6_2_2_HGNI/im_5421.png	1160	108	1160	126	31.0	17.4
211	capturas/2024-10-25_19_0_26_2_ROOT/im_5227.png	345	430	354	456	34.9	21.8
212	capturas/2024-12-25_17_21_0_2_ROCI/im_5851.png	581	889	524	845	34.9	21.8
213	capturas/2024-11-16_15_12_19_1_MPVE/im_5623.png	683	1044	682	1041	34.9	21.8

214 rows × 33 columns

In [123...

```
df12_train
```

Out[123…

	imagen	click_x	click_y	pos_x	pos_y	ancho_cm	alto_cm
0	capturas/2024-9-24_18_28_21_1_QFWJ/im_3840.png	682	762	682	762	30.2	19.6
1	capturas/2024-12-25_17_31_58_2_LIDJ/im_5947.png	809	581	812	597	34.9	21.8
2	capturas/2024-11-6_12_4_39_1_HGNI/im_5410.png	1424	66	1424	69	31.0	17.4
3	capturas/2025-1-6_17_45_31_2_MXZR/im_6064.png	512	562	486	552	34.5	19.4
4	capturas/2025-1-6_17_42_54_2_MXZR/im_6047.png	1329	794	1323	797	34.5	19.4
…	…	…	…	…	…	…	…
993	capturas/2024-11-16_15_12_19_1_MPVE/im_5614.png	814	20	817	18	34.9	21.8
994	capturas/2024-11-16_15_32_40_1_MPVE/im_5702.png	624	277	622	277	34.9	21.8
995	capturas/2024-9-24_18_33_59_2_QFWJ/im_3901.png	827	817	844	817	30.2	19.6
996	capturas/2024-9-25_14_11_54_1_JZYT/im_3951.png	2054	1047	2051	1043	99.0	99.0
997	capturas/2025-1-6_17_57_27_2_QXYT/im_6199.png	776	748	780	753	34.5	19.4

998 rows × 33 columns

Procesado de imágenes y limpieza de datos

El objetivo de este punto es pasar del formato de datos que se tiene al momento (dataframe con datos, etiquetas y referencias a imágenes: e imágenes de webcam con extensión .png en el sistema de ficheros) a tensores con imágenes recortadas (recortes oculares) y etiquetas. Se definen métodos para simplificar el proceso a continuación.

In [125…

```
FACE_DETECTOR = cv2.FaceDetectorYN.create(
    model="face_detection_yunet_2023mar.onnx",  # Modelo ONNX (descargar si es n
    config="",
    input_size=(320, 320),  # Tamaño de entrada
    score_threshold=0.9,    # Umbral de confianza
    nms_threshold=0.3,      # Umbral NMS
    top_k=1            # Número máximo de detecciones
)
```

In [126…

```
def detect_face_crop_eyes_yn(rgb_img):
    imh, imw, _ = rgb_img.shape
    global FACE_DETECTOR
    if FACE_DETECTOR.getInputSize() != (imw, imh):
        FACE_DETECTOR.setInputSize((imw, imh))
```

```
    _, faces = FACE_DETECTOR.detect(rgb_img)
    if faces is None:
        raise Exception("Error 0 de detección facial: No se encontró el rostro")

    x, y, w, h, le_x, le_y, re_x, re_y = (faces[0][:8]).astype(int)

    eyes_radius = abs(re_x - le_x)*0.8 // 2

    e_y = (re_y + le_y) // 2

    le_bb_x1 = int(le_x - eyes_radius)
    le_bb_x2 = int(le_x + eyes_radius)
    le_bb_y1 = int(e_y - eyes_radius)
    le_bb_y2 = int(e_y + eyes_radius)
    re_bb_x1 = int(re_x - eyes_radius)
    re_bb_x2 = int(re_x + eyes_radius)
    re_bb_y1 = int(e_y - eyes_radius)
    re_bb_y2 = int(e_y + eyes_radius)

    left_eye_img = rgb_img[le_bb_y1:le_bb_y2, le_bb_x1:le_bb_x2]
    right_eye_img = rgb_img[re_bb_y1:re_bb_y2, re_bb_x1:re_bb_x2]

    return x, y, (x+w), (y+h), left_eye_img, right_eye_img
```

In [128…

```
def square_crop_resize(original_img, new_wh):

    if original_img.shape[0] != original_img.shape[1]:
        raise Exception("La imagen no es cuadrada")

    interpolation = cv2.INTER_CUBIC

    if original_img.shape[0] > new_wh:
        interpolation = cv2.INTER_AREA
    elif original_img.shape[0] < new_wh:
        interpolation = cv2.INTER_CUBIC
    elif original_img.shape[0] == new_wh:
        return original_img

    resized_img = cv2.resize(original_img, (new_wh, new_wh), interpolation)
    return resized_img
```

Una vez declarados los métodos que implementan las transformaciones fundamentales para las imágenes de los conjuntos, el siguiente paso es construir el bucle que los llamará para generar las listas con los datos definitivos que se van a utilizar para el entrenamiento.

Es importante notar que las redes neuronales trabajan mejor con valores normalizados en vez de los absolutos. Para la lista I, que contiene las imágenes en formato *array* de `numpy` que a su vez contienen enteros sin signo de 8 bits, el proceso de normalización es dividir entre 255. Para la lista F, que contiene los valores inferidos por el clasificador facial para cada imagen (correspondientes a las rectas que describen el recorte facial respecto a la imagen completa), se dividen entre el ancho (para las *x*) o el alto (para las *y*) de las

In [129…

```
EYE_CROP_SIZE = 96
```

In [130…

```python
def generate_IFDGB(df12_temp):
    global EYE_CROP_SIZE
    I_l, I_r, F, D, G, B = [], [], [], [], [], []
    j, i = 0, 0
    image_error_count, face_detection_error_count = 0, 0

    for row in df12_temp.index:
        try:
            bgr_img, rgb_img, gray_img = get_image(row, df12_temp)
        except:
            image_error_count += 1
            continue

        try:
            x1, y1, x2, y2, left_eye_img, right_eye_img = detect_face_crop_eyes_
        except:
            face_detection_error_count += 1
            continue

        left_eye_img = square_crop_resize(left_eye_img, EYE_CROP_SIZE)
        right_eye_img = square_crop_resize(right_eye_img, EYE_CROP_SIZE)

        # ERROR DE FORMATO: En algunos datasets antiguos, la columna 'anchoImage

        I_l.append(left_eye_img/255)
        I_r.append(right_eye_img/255)
        F.append(np.array([x1/df12_temp['anchoImagenCam'][row], x2/df12_temp['an
        D.append(np.array([df12_temp['ancho_cm'][row], df12_temp['alto_cm'][row]
        G.append(np.array([1] if df12_temp['gafas'][row] == 'Si' else [0]))
        B.append(np.array([df12_temp["pos_x"][row]/df12_temp["ancho_px"][row], d

        j += 1
        clear_output(wait=True)
        print(f"({row + 1}/{len(df12_temp.index)})")
        print(f"Errores de imagen: {image_error_count}")
        print(f"Errores de detección facial: {face_detection_error_count}")
        print(f"Total de imágenes procesadas: {j}/{len(df12_temp)}")

    clear_output(wait=True)
    print("Finalizado")
    print(f"Errores de imagen: {image_error_count}")
    print(f"Errores de detección facial: {face_detection_error_count}")
    print(f"Total de imágenes procesadas: {j}/{len(I_l)}")

    return I_l, I_r, F, D, G, B
```

In [131…

```python
I_l_train, I_r_train, F_train, D_train, G_train, B_train = generate_IFDGB(df12_t
I_l_valid, I_r_valid, F_valid, D_valid, G_valid, B_valid = generate_IFDGB(df12_v
I_l_test, I_r_test, F_test, D_test, G_test, B_test = generate_IFDGB(df12_test)
```

```
Finalizado
Errores de imagen: 1
Errores de detección facial: 15
Total de imágenes procesadas: 89/89
```

Backup para desarrollo

Se salvan los valores para recuperarlos como estaban en este punto para no tener que regenerarlos necesariamente durante la depuración

In [132…

```
I_l_train_o = copy.deepcopy(I_l_train)
I_r_train_o = copy.deepcopy(I_r_train)
F_train_o = copy.deepcopy(F_train)
D_train_o = copy.deepcopy(D_train)
G_train_o = copy.deepcopy(G_train)
B_train_o = copy.deepcopy(B_train)

I_l_valid_o = copy.deepcopy(I_l_valid)
I_r_valid_o = copy.deepcopy(I_r_valid)
F_valid_o = copy.deepcopy(F_valid)
D_valid_o = copy.deepcopy(D_valid)
G_valid_o = copy.deepcopy(G_valid)
B_valid_o = copy.deepcopy(B_valid)

I_l_test_o = copy.deepcopy(I_l_test)
I_r_test_o = copy.deepcopy(I_r_test)
F_test_o = copy.deepcopy(F_test)
D_test_o = copy.deepcopy(D_test)
G_test_o = copy.deepcopy(G_test)
B_test_o = copy.deepcopy(B_test)
```

Ejecutar esta celda para recuperar los valores

In [133…

```
I_l_train, I_r_train, F_train, D_train, G_train, B_train = I_l_train_o, I_r_trai
I_l_valid, I_r_valid, F_valid, D_valid, G_valid, B_valid = I_l_valid_o, I_r_vali
I_l_test, I_r_test, F_test, D_test, G_test, B_test = I_l_test_o, I_r_test_o, F_t
```

Aumentacion de datos tras detección facial

Este es un apartado que requiere un especial cuidado, no pudiendo seguir los pasos que se suelen hacer en otras redes neuronales con datos de otros tipos. En el presente proyecto, la posición de los ojos está en las imágenes está relacionada unívocamente con sus etiquetas. Alterar las imágenes variando su posición o rotándolas debería alterar las coordenadas de la pantalla a las que se dirige la mirada, de una forma que no es posible predecir (ya que ese es el cometido del modelo, y en el momento del entrenamiento no contamos con él). En otras palabras, si se mueven los ojos en la imagen, sus etiquetas asociadas deben ser modificadas en consecuencia o dejan de ser válidas. No sería un problema si supieramos cómo alterar los valores de esas etiquetas para que sigan siendo correctas.

Por suerte hay un caso en el que se puede hacer aumentación alterando imágenes y etiquetas: Reflejando ambas. Para las imágenes se puede aplicar un método de `opencv`, para reflejar una coordenada horizontal se aplica esta fórmula:

$$x_{reflejada} = x_{max} - x_{original}$$

Siendo `i` un índice valido cualquiera dentro de las listas. Para este caso, el valor máximo de cualquier coordenada, al estar normalizadas, es 1. Un array de la lista B tiene este formato:

`B[i] = [x, y]`

Su reflejo sería:

`B_mirrored[i] = [1 - x, y]`

Con los arrays de la lista `F` hay que tener alguna consideración a mayores:

`F[i] = [x1, x2 , y1, y2]`

[POSIBLE IMAGEN] Las coordenadas de esa se pueden definir como las rectas que describen un recorte facial, que es una imagen cuadrada que contiene una cara detectada. Las dos rectas verticales, al reflejarlas, resultan intercambiadas, de forma que el array reflejado queda así:

`F_mirrored[i] = [1 - x2, 1 - x1, y1, y2]`

In [134…
```python
def augmentation_flip_v2(I_l, I_r, F, D, G, B):
    I_l_mirrored, I_r_mirrored, F_mirrored, B_mirrored = [], [], [], []
    for i in range(len(D)):
        I_l_mirrored.append(cv2.flip(I_r[i], 1))
        I_r_mirrored.append(cv2.flip(I_l[i], 1))
        F_mirrored.append([1 - F[i][1], 1 - F[i][0], F[i][2], F[i][3]])
        B_mirrored.append([1 - B[i][0], B[i][1]])

    return I_l_mirrored, I_r_mirrored, F_mirrored, D, G, B_mirrored
```

Se generan las listas I, F, D, G y B con aumentación por reflejo

In [135…
```python
I_l_train_mirror, I_r_train_mirror, F_train_mirror, D_train_mirror, G_train_mirr
I_l_valid_mirror, I_r_valid_mirror, F_valid_mirror, D_valid_mirror, G_valid_mirr
I_l_test_mirror, I_r_test_mirror, F_test_mirror, D_test_mirror, G_test_mirror, B
```

Se repite la aumentación, de forma muy similar, pero esta vez añadiendo ruido Gaussiano a las imágenes. Esto suele mejorar la generalización a costa de algo de precisión, lo que se traduce en que las métricas de precisión en entrenamiento serán algo peores, pero en validación y test, probablemente mejores y menos dispersas. El ruido Gaussiano no es relacionable para el modelo, la red neuronal no puede aprenderlo

In [136…
```python
def augmentation_add_noise(I_l, I_r, F, D, G, B, noise_std=0.01):

    I_l_noisy, I_r_noisy, F_noisy, B_noisy = [], [], [], []
    D_noisy, G_noisy = [], []

    for i in range(len(D)):

        noisy_left = np.clip(I_l[i] + np.random.normal(0, noise_std, I_l[i].shap
        noisy_right = np.clip(I_r[i] + np.random.normal(0, noise_std, I_r[i].sha


        F_noisy.append(F[i])
        D_noisy.append(D[i])
        G_noisy.append(G[i])
        B_noisy.append(B[i])

        I_l_noisy.append(noisy_left)
        I_r_noisy.append(noisy_right)
```

```
    return I_l_noisy, I_r_noisy, F_noisy, D_noisy, G_noisy, B_noisy
```

In [137…
```
I_l_train_noisy, I_r_train_noisy, F_train_noisy, D_train_noisy, G_train_noisy, B
I_l_valid_noisy, I_r_valid_noisy, F_valid_noisy, D_valid_noisy, G_valid_noisy, B
```

Con el siguiente método se concatena una fracción aleatoria de las muestras aumentadas

In [138…
```
def append_random_samples(I_l, I_r, F, D, G, B, I_l_aug, I_r_aug, F_aug, D_aug,

    num_samples = int(len(D) * fraction)
    sampled_indices = random.sample(range(len(D_aug)-1), num_samples)


    I_l.extend([I_l_aug[i] for i in sampled_indices])
    I_r.extend([I_r_aug[i] for i in sampled_indices])
    F.extend([F_aug[i] for i in sampled_indices])
    D.extend([D_aug[i] for i in sampled_indices])
    G.extend([G_aug[i] for i in sampled_indices])
    B.extend([B_aug[i] for i in sampled_indices])

    return I_l, I_r, F, D, G, B
```

In [139…
```
#TEST
len(I_l_train)
```

Out[139… 918

In [140…
```
I_l_train, I_r_train, F_train, D_train, G_train, B_train = append_random_samples
    I_l_train, I_r_train, F_train, D_train, G_train, B_train,
    I_l_train_mirror, I_r_train_mirror, F_train_mirror, D_train_mirror, G_train_
    fraction=0.6
)
I_l_valid, I_r_valid, F_valid, D_valid, G_valid, B_valid = append_random_samples
    I_l_valid, I_r_valid, F_valid, D_valid, G_valid, B_valid,
    I_l_valid_mirror, I_r_valid_mirror, F_valid_mirror, D_valid_mirror, G_valid_
    fraction=0.6
)
I_l_test, I_r_test, F_test, D_test, G_test, B_test = append_random_samples(
    I_l_test, I_r_test, F_test, D_test, G_test, B_test,
    I_l_test_mirror, I_r_test_mirror, F_test_mirror, D_test_mirror, G_test_mirro
    fraction=0.6
)
```

In [141…
```
I_l_train, I_r_train, F_train, D_train, G_train, B_train = append_random_samples
    I_l_train, I_r_train, F_train, D_train, G_train, B_train,
    I_l_train_noisy, I_r_train_noisy, F_train_noisy, D_train_noisy, G_train_nois
    fraction=0.4
)

I_l_valid, I_r_valid, F_valid, D_valid, G_valid, B_valid = append_random_samples
    I_l_valid, I_r_valid, F_valid, D_valid, G_valid, B_valid,
    I_l_valid_noisy, I_r_valid_noisy, F_valid_noisy, D_valid_noisy, G_valid_nois
    fraction=0.4
)
```

In [142…

```
print(f"Shape of I_l_train[0]: {np.array(I_l_train[0]).shape}")
print(f"Shape of I_r_train[0]: {np.array(I_r_train[0]).shape}")
print(f"Shape of F_train[0]: {np.array(F_train[0]).shape}")
print(f"Shape of D_train[0]: {np.array(D_train[0]).shape}")
print(f"Shape of G_train[0]: {np.array(G_train[0]).shape}")
print(f"Shape of B_train[0]: {np.array(B_train[0]).shape}")


print(f"Length of I_l_train: {len(I_l_train)}")
print(f"Length of I_r_train: {len(I_r_train)}")
print(f"Length of F_train: {len(F_train)}")
print(f"Length of D_train: {len(D_train)}")
print(f"Length of G_train: {len(G_train)}")
print(f"Length of B_train: {len(B_train)}")
```

```
Shape of I_l_train[0]: (96, 96, 3)
Shape of I_r_train[0]: (96, 96, 3)
Shape of F_train[0]: (4,)
Shape of D_train[0]: (2,)
Shape of G_train[0]: (1,)
Shape of B_train[0]: (2,)
Length of I_l_train: 2055
Length of I_r_train: 2055
Length of F_train: 2055
Length of D_train: 2055
Length of G_train: 2055
Length of B_train: 2055
```

A continuación las listas se convierten a tensores. Este es un formato de matriz de Tensorflow. Al ejecutar el método, se almacenan en la memoria de la GPU, lo que agiliza mucho el proceso de lectura durante el entrenamiento.

In [143…

```
I_l_train = tf.convert_to_tensor(I_l_train, dtype=tf.float32)
I_r_train = tf.convert_to_tensor(I_r_train, dtype=tf.float32)
F_train = tf.convert_to_tensor(F_train, dtype=tf.float32)
D_train = tf.convert_to_tensor(D_train, dtype=tf.float32)
G_train = tf.convert_to_tensor(G_train, dtype=tf.float32)
B_train = tf.convert_to_tensor(B_train, dtype=tf.float32)

I_l_valid = tf.convert_to_tensor(I_l_valid, dtype=tf.float32)
I_r_valid = tf.convert_to_tensor(I_r_valid, dtype=tf.float32)
F_valid = tf.convert_to_tensor(F_valid, dtype=tf.float32)
D_valid = tf.convert_to_tensor(D_valid, dtype=tf.float32)
G_valid = tf.convert_to_tensor(G_valid, dtype=tf.float32)
B_valid = tf.convert_to_tensor(B_valid, dtype=tf.float32)

I_l_test = tf.convert_to_tensor(I_l_test, dtype=tf.float32)
I_r_test = tf.convert_to_tensor(I_r_test, dtype=tf.float32)
F_test = tf.convert_to_tensor(F_test, dtype=tf.float32)
D_test = tf.convert_to_tensor(D_test, dtype=tf.float32)
G_test = tf.convert_to_tensor(G_test, dtype=tf.float32)
B_test = tf.convert_to_tensor(B_test, dtype=tf.float32)
```

Definición y entrenamiento del modelo

El tamaño del input es directamente proporcional al tamaño del recorte facial (almacenado en `EYE_CROP_SIZE`), ya que el método heurístico que devuelve los recortes oculares devuelve una determinada fracción del recorte facial. Al tener todos ellos las mismas dimensiones (Al ser tensores, no puede ser de otra manera), se define la forma de la capa de entrada a partir de la forma del tensor de esta manera:

El número de épocas, el tamaño del bloque y la tasa de aprendizaje son los hiperparámetros directamente asociados al entrenamiento, y los más suceptibles de ser alterados en caso de inconveniencias en el entrenamiento.

In [144…
```
BATCH_SIZE = 16
LEARNING_RATE = 0.001
```

In [145…
```
# Función para encapsular MobileNet en un submodelo con nombre único
def build_mobilenet_branch(name):
    global EYE_CROP_SIZE
    input_tensor = layers.Input(shape=(EYE_CROP_SIZE, EYE_CROP_SIZE, 3))
    mobilenet = MobileNetV2(input_shape=(EYE_CROP_SIZE, EYE_CROP_SIZE, 3), inclu
    mobilenet.trainable = False
    x = mobilenet(input_tensor)
    return models.Model(inputs=input_tensor, outputs=x, name=name)
```

In [146…
```
def build_model():
    global EYE_CROP_SIZE
    # Crear submodelos MobileNet con nombres únicos
    mobilenet_0 = build_mobilenet_branch("mobilenet_left")
    mobilenet_1 = build_mobilenet_branch("mobilenet_right")

    # Inputs
    input_image_l = layers.Input(shape=(EYE_CROP_SIZE, EYE_CROP_SIZE, 3), name='
    input_image_r = layers.Input(shape=(EYE_CROP_SIZE, EYE_CROP_SIZE, 3), name='
    input_face_pos = layers.Input(shape=(4,), name='face_pos_input')
    input_screen_size = layers.Input(shape=(2,), name='screen_size_input')
    input_glasses = layers.Input(shape=(1,), name='glasses_input')

    # Procesamiento de imágenes
    l = mobilenet_0(input_image_l)
    l = layers.Conv2D(32, (3, 3), padding='same', activation='relu')(l)
    l = layers.BatchNormalization()(l)
    l = layers.Flatten()(l)

    r = mobilenet_1(input_image_r)
    r = layers.Conv2D(32, (3, 3), padding='same', activation='relu')(r)
    r = layers.BatchNormalization()(r)
    r = layers.Flatten()(r)

    # Concatenación con entradas numéricas
    x = layers.Concatenate()([l, input_face_pos, input_screen_size, input_glasse

    # Capas densas finales
    x = layers.Dense(64, activation='relu', kernel_regularizer=tf.keras.regulari
    x = layers.BatchNormalization()(x)
    x = layers.Dropout(0.1)(x)
    x = layers.Dense(32, activation='relu')(x)
    x = layers.BatchNormalization()(x)
    output = layers.Dense(2, activation='sigmoid', name='coordinates_output')(x)
```

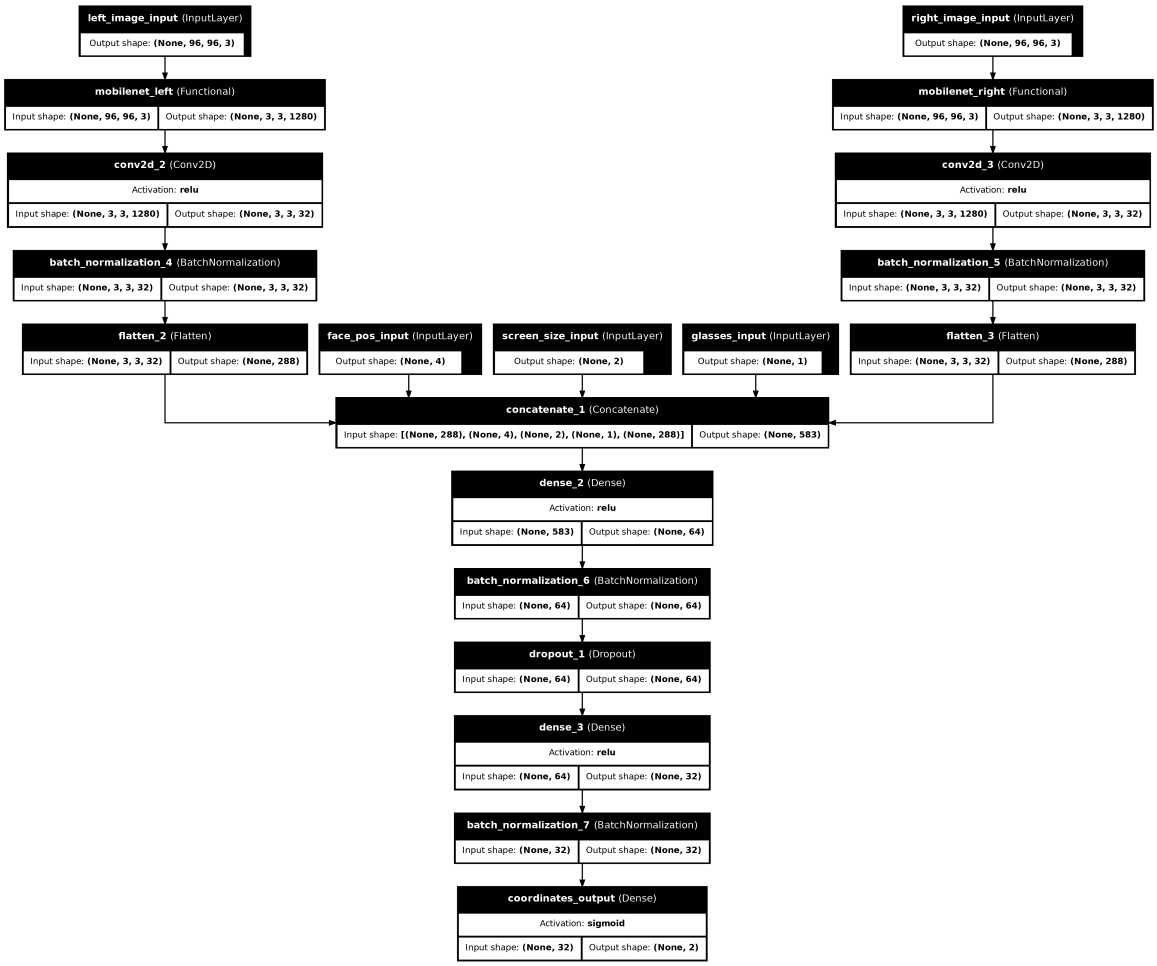
```
    # Modelo final
    return models.Model(
        inputs=[input_image_l, input_image_r, input_face_pos, input_screen_size,
        outputs=output
    )
```

In [147…
```
model = build_model()
```

In [148…
```
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss=tf.keras.losses.Huber(delta=1.0),
    metrics=['mae']
)
```

In [149…
```
tf.keras.utils.plot_model(model, dpi=200, show_shapes=True, show_layer_names=Tru
```

Out[149…



El mecanismo de EarlyStopping sirve para detectar puntos de inflexión en alguna de las métricas, que generalmente suele ser la función de pérdida en validación. El parámetro `patience` contiene el número de épocas que se esperará para confirmar el punto de inflexión. Cuando esto ocurra, el mecanismo emite un callback que detiene el entrenamiento y recupera los pesos inmediatamente anteriores al punto de inflexión.

In [150…
```
# Configurar EarlyStopping
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, re
```

Este es otro callback que reduce la tasa de aprendizaje cuando el factor por el que se reduce la pérdida tras cada época es inferior a un cierto umbral

In [151…

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.3,
    patience=4,
    min_lr=1e-6
)
```

Se incia el entrenamiento almacenando en una variable el historial que el método `fit()` devuelve, para su posterior análisis:

In [152…

```
train_history = model.fit(
    x=[I_l_train, I_r_train, F_train, D_train, G_train],
    y=B_train,
    validation_data=([I_l_valid, I_r_valid, F_valid, D_valid, G_valid], B_valid)
    epochs=50,
    batch_size=BATCH_SIZE,
    callbacks=[early_stop , reduce_lr],
    verbose=1
    )
```

```
Epoch 1/50
```

```
129/129 ━━━━━━━━━━━━━━━━━━━━ 19s 78ms/step - loss: 0.1358 - mae: 0.2366 - val_los
s: 0.0817 - val_mae: 0.1897 - learning_rate: 0.0010
Epoch 2/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0740 - mae: 0.1920 - val_los
s: 0.0585 - val_mae: 0.1924 - learning_rate: 0.0010
Epoch 3/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0486 - mae: 0.1663 - val_los
s: 0.0470 - val_mae: 0.1897 - learning_rate: 0.0010
Epoch 4/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0358 - mae: 0.1564 - val_los
s: 0.0311 - val_mae: 0.1569 - learning_rate: 0.0010
Epoch 5/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0266 - mae: 0.1406 - val_los
s: 0.0261 - val_mae: 0.1508 - learning_rate: 0.0010
Epoch 6/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0215 - mae: 0.1319 - val_los
s: 0.0217 - val_mae: 0.1396 - learning_rate: 0.0010
Epoch 7/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0187 - mae: 0.1278 - val_los
s: 0.0212 - val_mae: 0.1453 - learning_rate: 0.0010
Epoch 8/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0157 - mae: 0.1213 - val_los
s: 0.0194 - val_mae: 0.1409 - learning_rate: 0.0010
Epoch 9/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0137 - mae: 0.1141 - val_los
s: 0.0149 - val_mae: 0.1218 - learning_rate: 0.0010
Epoch 10/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0124 - mae: 0.1094 - val_los
s: 0.0125 - val_mae: 0.1093 - learning_rate: 0.0010
Epoch 11/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0123 - mae: 0.1106 - val_los
s: 0.0139 - val_mae: 0.1200 - learning_rate: 0.0010
Epoch 12/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - loss: 0.0111 - mae: 0.1057 - val_los
s: 0.0124 - val_mae: 0.1126 - learning_rate: 0.0010
Epoch 13/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - loss: 0.0103 - mae: 0.1035 - val_los
s: 0.0118 - val_mae: 0.1101 - learning_rate: 0.0010
Epoch 14/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0100 - mae: 0.1028 - val_los
s: 0.0125 - val_mae: 0.1179 - learning_rate: 0.0010
Epoch 15/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0098 - mae: 0.1018 - val_los
s: 0.0123 - val_mae: 0.1143 - learning_rate: 0.0010
Epoch 16/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0082 - mae: 0.0923 - val_los
s: 0.0121 - val_mae: 0.1134 - learning_rate: 0.0010
Epoch 17/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0095 - mae: 0.1011 - val_los
s: 0.0105 - val_mae: 0.1013 - learning_rate: 0.0010
Epoch 18/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0090 - mae: 0.0970 - val_los
s: 0.0101 - val_mae: 0.1025 - learning_rate: 0.0010
Epoch 19/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0074 - mae: 0.0880 - val_los
s: 0.0096 - val_mae: 0.0985 - learning_rate: 0.0010
Epoch 20/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0075 - mae: 0.0898 - val_los
s: 0.0126 - val_mae: 0.1177 - learning_rate: 0.0010
Epoch 21/50
```

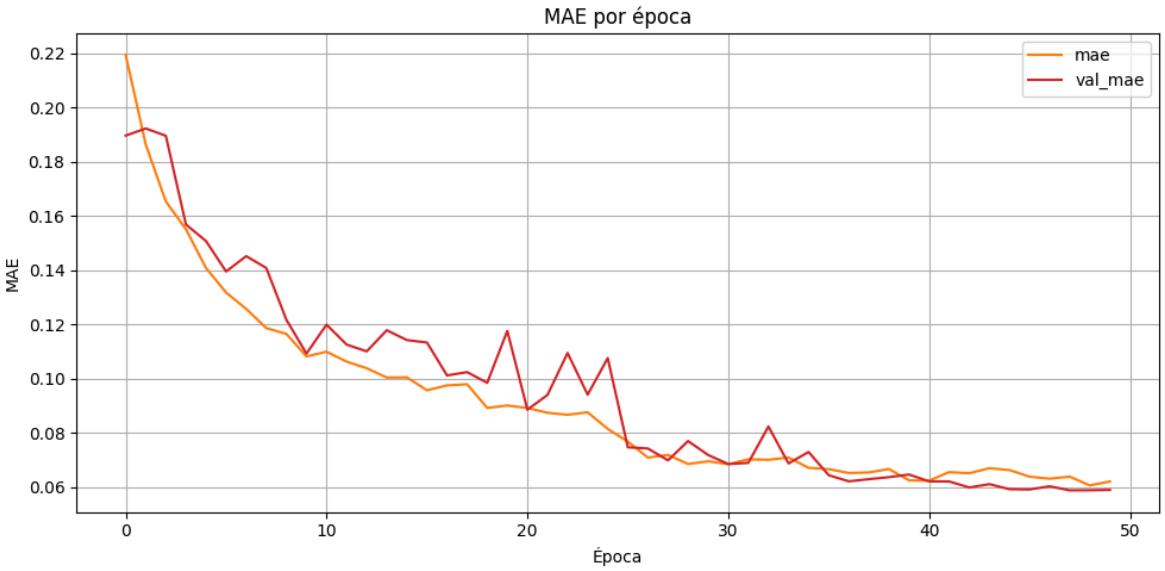
```
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0075 - mae: 0.0898 - val_los
s: 0.0082 - val_mae: 0.0886 - learning_rate: 0.0010
Epoch 22/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0069 - mae: 0.0861 - val_los
s: 0.0088 - val_mae: 0.0941 - learning_rate: 0.0010
Epoch 23/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0076 - mae: 0.0891 - val_los
s: 0.0108 - val_mae: 0.1095 - learning_rate: 0.0010
Epoch 24/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0072 - mae: 0.0892 - val_los
s: 0.0086 - val_mae: 0.0942 - learning_rate: 0.0010
Epoch 25/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0063 - mae: 0.0824 - val_los
s: 0.0103 - val_mae: 0.1077 - learning_rate: 0.0010
Epoch 26/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0057 - mae: 0.0800 - val_los
s: 0.0060 - val_mae: 0.0748 - learning_rate: 3.0000e-04
Epoch 27/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0046 - mae: 0.0711 - val_los
s: 0.0059 - val_mae: 0.0743 - learning_rate: 3.0000e-04
Epoch 28/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0043 - mae: 0.0690 - val_los
s: 0.0054 - val_mae: 0.0699 - learning_rate: 3.0000e-04
Epoch 29/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0041 - mae: 0.0679 - val_los
s: 0.0061 - val_mae: 0.0771 - learning_rate: 3.0000e-04
Epoch 30/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0045 - mae: 0.0706 - val_los
s: 0.0056 - val_mae: 0.0719 - learning_rate: 3.0000e-04
Epoch 31/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0041 - mae: 0.0683 - val_los
s: 0.0052 - val_mae: 0.0686 - learning_rate: 3.0000e-04
Epoch 32/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0043 - mae: 0.0701 - val_los
s: 0.0053 - val_mae: 0.0690 - learning_rate: 3.0000e-04
Epoch 33/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0041 - mae: 0.0681 - val_los
s: 0.0065 - val_mae: 0.0824 - learning_rate: 3.0000e-04
Epoch 34/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0041 - mae: 0.0690 - val_los
s: 0.0052 - val_mae: 0.0688 - learning_rate: 3.0000e-04
Epoch 35/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0036 - mae: 0.0647 - val_los
s: 0.0056 - val_mae: 0.0730 - learning_rate: 3.0000e-04
Epoch 36/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0037 - mae: 0.0650 - val_los
s: 0.0049 - val_mae: 0.0645 - learning_rate: 9.0000e-05
Epoch 37/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0036 - mae: 0.0639 - val_los
s: 0.0048 - val_mae: 0.0622 - learning_rate: 9.0000e-05
Epoch 38/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0038 - mae: 0.0658 - val_los
s: 0.0047 - val_mae: 0.0630 - learning_rate: 9.0000e-05
Epoch 39/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0037 - mae: 0.0658 - val_los
s: 0.0049 - val_mae: 0.0637 - learning_rate: 9.0000e-05
Epoch 40/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0032 - mae: 0.0604 - val_los
s: 0.0049 - val_mae: 0.0647 - learning_rate: 9.0000e-05
Epoch 41/50
```

```
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0033 - mae: 0.0617 - val_los
s: 0.0048 - val_mae: 0.0622 - learning_rate: 9.0000e-05
Epoch 42/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0036 - mae: 0.0635 - val_los
s: 0.0047 - val_mae: 0.0621 - learning_rate: 2.7000e-05
Epoch 43/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0032 - mae: 0.0628 - val_los
s: 0.0046 - val_mae: 0.0599 - learning_rate: 2.7000e-05
Epoch 44/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0038 - mae: 0.0683 - val_los
s: 0.0046 - val_mae: 0.0612 - learning_rate: 2.7000e-05
Epoch 45/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0037 - mae: 0.0663 - val_los
s: 0.0045 - val_mae: 0.0593 - learning_rate: 2.7000e-05
Epoch 46/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0032 - mae: 0.0612 - val_los
s: 0.0045 - val_mae: 0.0592 - learning_rate: 2.7000e-05
Epoch 47/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0033 - mae: 0.0626 - val_los
s: 0.0045 - val_mae: 0.0604 - learning_rate: 2.7000e-05
Epoch 48/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0034 - mae: 0.0642 - val_los
s: 0.0045 - val_mae: 0.0589 - learning_rate: 8.1000e-06
Epoch 49/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0031 - mae: 0.0601 - val_los
s: 0.0045 - val_mae: 0.0589 - learning_rate: 8.1000e-06
Epoch 50/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0033 - mae: 0.0628 - val_los
s: 0.0045 - val_mae: 0.0590 - learning_rate: 8.1000e-06
```

Y a continuación se dibujan gráficas de las métricas registradas durante el proceso

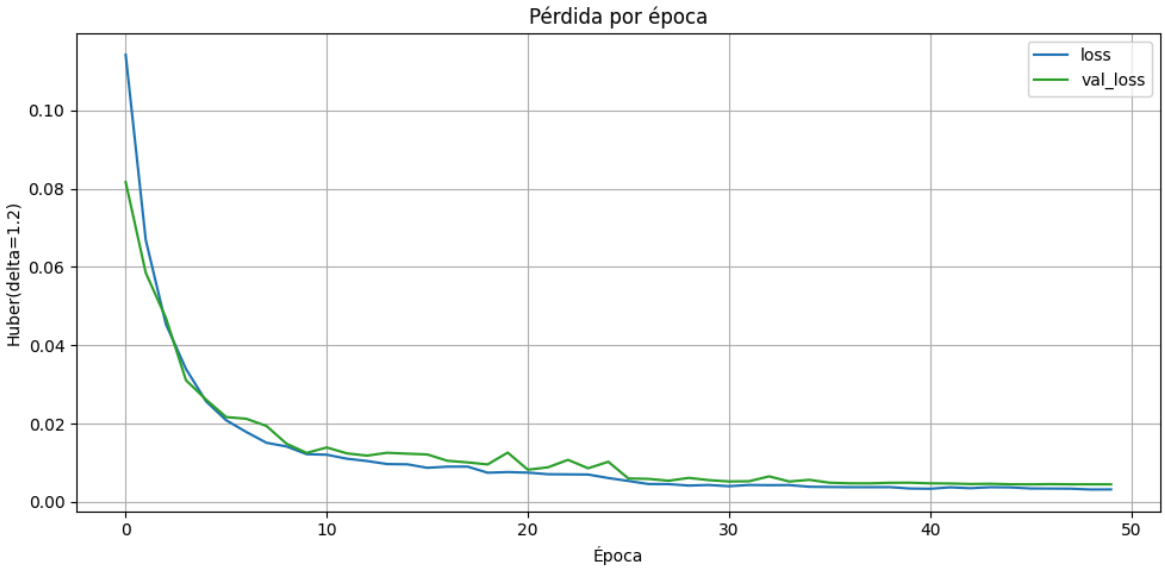
In [153…

```python
pd.DataFrame({
    'mae': train_history.history['mae'],
    'val_mae': train_history.history['val_mae']
}).plot(figsize=(10,5), color={'mae': '#ff7f0e', 'val_mae': '#d62728'})
plt.grid(True)
plt.title('MAE por época')
plt.xlabel('Época')
plt.ylabel('MAE')
plt.tight_layout()
plt.legend()
plt.show()
```



In [154…

```
pd.DataFrame({
    'loss': train_history.history['loss'],
    'val_loss': train_history.history['val_loss']
}).plot(figsize=(10,5), color={'loss': '#1f77b4', 'val_loss': '#2ca02c'})
plt.grid(True)
plt.title('Pérdida por época')
plt.xlabel('Época')
plt.ylabel('Huber(delta=1.2)')
plt.legend()
plt.tight_layout()
plt.show()
```

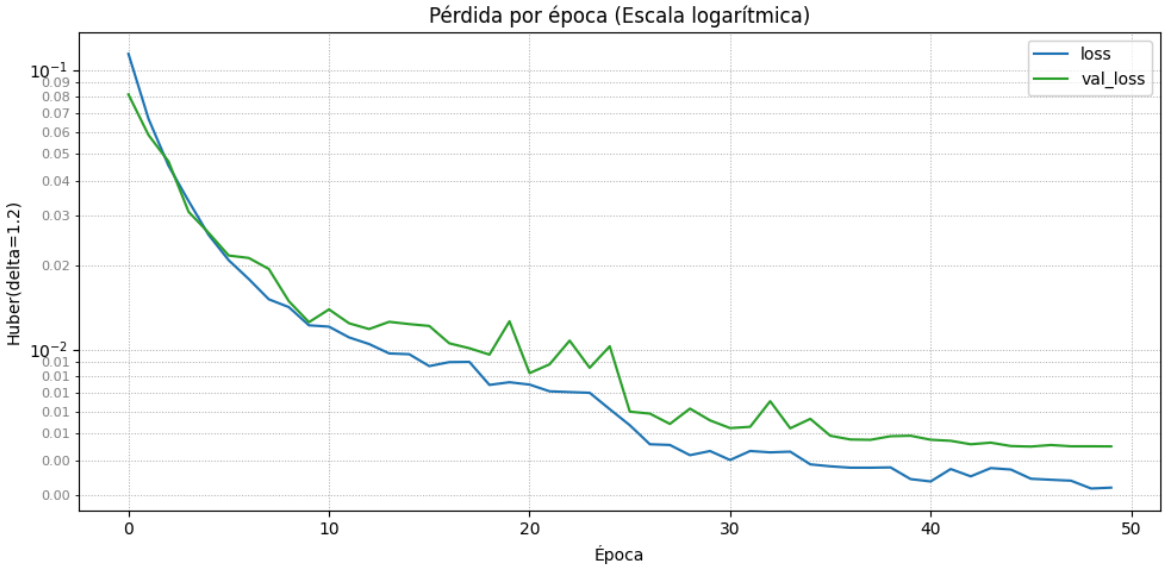


Dada la naturaleza de estas funciones, las escalas logarítmicas en el eje vertical permiten visualizar mucho mejor la variación.

In [155…

```
pd.DataFrame({
    'loss': train_history.history['loss'],
    'val_loss': train_history.history['val_loss']
}).plot(figsize=(10,5), color={'loss': '#1f77b4', 'val_loss': '#2ca02c'})
plt.grid(which='both', axis='both', linestyle=':', linewidth=0.7)
plt.title('Pérdida por época (Escala logarítmica)')
plt.xlabel('Época')
plt.ylabel('Huber(delta=1.2)')
```

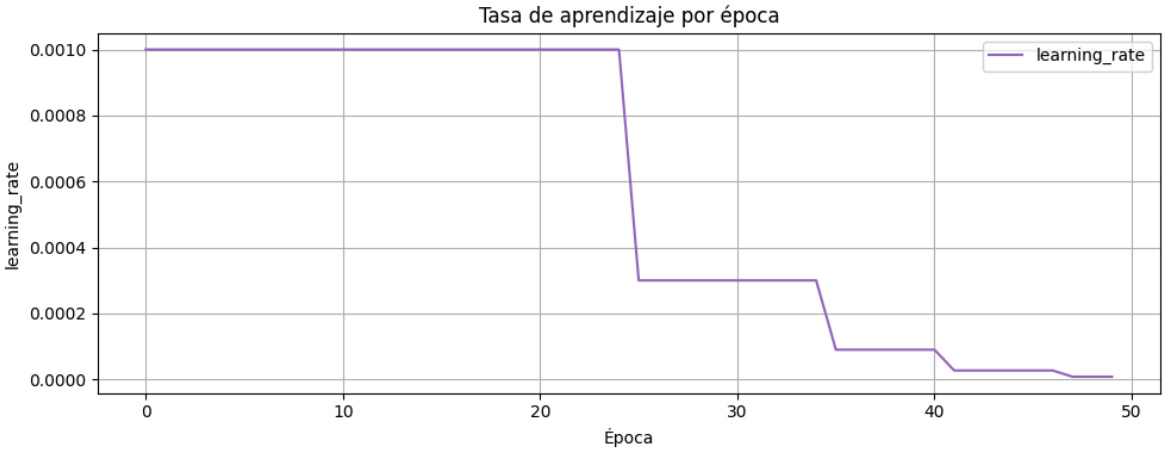
```
plt.legend()
plt.gca().set_yscale('log')

plt.gca().yaxis.set_minor_locator(mticker.LogLocator(base=10.0, subs=np.arange(1
plt.gca().yaxis.set_minor_formatter(mticker.ScalarFormatter())
plt.tick_params(axis='y', which='minor', labelsize=8, labelcolor='gray', labelle

plt.tight_layout()
plt.show()
```



La visualización conjunta de estas dos gráficas evidencia el efecto de la tasa de aprendizaje en el entrenamiento.

In [156…

```
pd.DataFrame({'learning_rate':train_history.history['learning_rate']}).plot(figs
plt.title('Tasa de aprendizaje por época')
plt.xlabel('Época')
plt.ylabel('learning_rate')
plt.legend(['learning_rate'])
plt.grid(True)
plt.tight_layout()

plt.show()
```



Evaluación estándar para usuarios conocidos

Lo primero es graficar las estadísticas del entrenamiento en función de la época

Aquí es donde se puede discernir si el entrenamiento ha sido correcto o no. Estos son los tres posibles escenarios:

- Underfitting: El entrenamiento es insuficiente. El modelo no es capaz de inferir buenas predicciones. Los valores de precisión en entrenamiento y validación son bajos y próximos. Se puede solucionar ajustando el número de épocas (hacia arriba), los parámetros del modelo o utilizando datasets más grandes.
- Appropiate-fitting: El entrenamiento es correcto. Los valores de precisión y pérdida presentan forma de asíntota horizontal. En entrenamiento, ligeramente más próximas a los extremos que en validación.
- Overfitting: El entrenamiento es excesivo. El modelo "memoriza" el set de entrenamiento y no es capaz de generalizar. Los valores de pérdida en validación se alejan de los de entrenamiento en el momento en el que se empieza a producir. Se puede solucionar ajustando el número de épocas (hacia abajo), los parámetros del modelo, o utilizando datasts más grandes.

In [157…
```
# Evaluar el modelo
eval_results = model.evaluate([I_l_test, I_r_test, F_test, D_test, G_test], B_te
```
```
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 414ms/step - loss: 0.0036 - mae: 0.0496
```

In [158…
```
predictions = model.predict([I_l_test, I_r_test, F_test, D_test, G_test])
```
```
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 743ms/step
```

A continuación vamos a definir un método que reciba un input (I F D G) o un conjunto de ellos para inferir con el modelo recién entrenado las coordenadas de la pantalla a las que se orienta la mirada de los usuarios.

In [159…
```
def predict_single(image_l, image_r, face_pos, diagonal, glasses):
    return model.predict([np.expand_dims(image_l, axis=0), np.expand_dims(image_
```

En el contexto de este modelo, no basta con utilizar las métricas de precisión (accuracy) y pérdida (loss) utilizadas por `tensorflow` para evaluar el rendimiento del modelo durante entrenamiento, validación y test, pues aunque sirven para comparar y comprobar si el modelo está/fue entrenado correctamente, no son útiles para cuantificar el error cometido por el modelo al realizar predicciones con el conjunto de test, o al menos de forma simple

Se necesita una métrica que mida dicha distancia y resulte sencilla de comprender. La que mejor se adapta a estas circunstancias es la media del error absoluto (MAE), que se define por la siguiente expresión:

$$\sum_{i=1}^{D} |y_i - \hat{y}_i| = 0$$

Donde y e $\hat{y}$ son vectores de dimensión D y uno de ellos contiene los valores reales mientras que el otro, los inferidos. La fórmula calcula el valor medio de las diferencias absolutas.

La implementación del método `tf.keras.metrics.MeanAbsoluteError():` de Keras es `loss = mean(abs(y_true - y_pred))`

La red cuenta con dos neuronas en la capa de salida. La primera devuelve la coordenada horizontal. La segunda, la vertical. Si se calcula el MAE para cada una de ellas, resulta obvio que el resultado es el promedio de las diferencias absolutas de los elementos del tensor de test y los del array de predicciones

In [160…
```
# MAE horizontal
tf.keras.metrics.MAE(B_test[:,0], predictions[:,0])
```

Out[160…
```
<tf.Tensor: shape=(), dtype=float32, numpy=0.05574736>
```

In [161…
```
# MAE vertical
tf.keras.metrics.MAE(B_test[:,1], predictions[:,1])
```

Out[161…
```
<tf.Tensor: shape=(), dtype=float32, numpy=0.04351123>
```

Lo que no resulta tan obvio es concluir el formato del valor que devuelve este método en caso de recibir como argumento `B_test` y `predictions` al completo

In [162…
```
MAEs = tf.keras.metrics.MAE(B_test, predictions).numpy()
```

Es un arreglo unidimensional, con el mismo número de elementos que el set de test. Cada uno de ellos es el promedio de los MAEs horizontal y vertical de cada elemento de `B_test` y `predictions`

In [163…
```
MAEs
```

Out[163…
```
array([0.02967508, 0.13751918, 0.01620687, 0.00324573, 0.0156599 ,
       0.03564537, 0.04380201, 0.19548771, 0.00352234, 0.13051626,
       0.04399434, 0.02937308, 0.05023499, 0.01454972, 0.03778261,
       0.01106051, 0.08102173, 0.01739189, 0.07849722, 0.0187556 ,
       0.06103064, 0.01342887, 0.02553582, 0.01542051, 0.01011398,
       0.11286314, 0.02930507, 0.02767587, 0.10938647, 0.01156536,
       0.00946957, 0.02687818, 0.03175502, 0.01192062, 0.00582755,
       0.05325415, 0.12367894, 0.03234719, 0.01451439, 0.02680722,
       0.03136254, 0.14599662, 0.02272727, 0.01633268, 0.01221481,
       0.03896774, 0.04416677, 0.06149392, 0.01545513, 0.09445317,
       0.17959405, 0.02317278, 0.02174898, 0.01210794, 0.1629885 ,
       0.03198931, 0.00250785, 0.05372122, 0.05498388, 0.02720968,
       0.04755668, 0.02979681, 0.03211042, 0.00957744, 0.03137076,
       0.03279413, 0.04988388, 0.09524786, 0.02846768, 0.0055252 ,
       0.02177717, 0.04786634, 0.03366615, 0.01574714, 0.01217693,
       0.12258378, 0.01180981, 0.03550868, 0.04280686, 0.04382814,
       0.02521978, 0.0154958 , 0.04353681, 0.3240503 , 0.01516049,
       0.01915318, 0.00805034, 0.01197948, 0.02676234, 0.0400942 ,
       0.02800453, 0.05063267, 0.01164755, 0.07668385, 0.14499983,
       0.0108763 , 0.06102923, 0.05855574, 0.02711682, 0.05243629,
       0.01864845, 0.02795774, 0.04506108, 0.12588073, 0.02503096,
       0.01062194, 0.1638279 , 0.02409866, 0.07317069, 0.02280092,
       0.02211905, 0.13354361, 0.02643129, 0.03500596, 0.02435519,
       0.02596302, 0.05676524, 0.01552821, 0.0541044 , 0.02563041,
       0.02602162, 0.02633066, 0.18646418, 0.20137565, 0.04347028,
       0.03013832, 0.08531754, 0.19046739, 0.36364996, 0.04559183,
       0.06098118, 0.15766338, 0.06489713, 0.22833103, 0.08633213,
       0.02883276, 0.07998736, 0.2071101 , 0.09334354, 0.10381678,
       0.09099317, 0.01147935], dtype=float32)
```

La media de todos ellos coincide con el resultado que devuelve la función de pérdidas `mean_absolute_error` de Keras

In [164…
```
MAEs.mean()
```

Out[164…
```
0.049613129
```

In [165…
```
tf.keras.losses.MeanAbsoluteError()(B_test, predictions).numpy()
```

Out[165…
```
0.049613129
```

De cara a observar los resultados, no es la implementación más adecuada, pues las medias de las desviaciones absolutas horizontales y verticales no son necesariamente iguales a las distancias absolutas. Se define a continuación un método para calcular las distancias en su valor absoluto, que implementa esta fórmula:

$$d_i = \sqrt{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2}$$

In [166…
```
def calc_deviation(predictions, reals, normalize=False):
    difference = predictions - reals
    squared = difference**2
    squared = tf.transpose(squared)
    deviations = np.sqrt(squared[0] + squared[1]) if not normalize else np.sqrt(
    return deviations
```

In [167…
```
deviations = calc_deviation(predictions=model.predict([I_l_test, I_r_test, F_tes
print(np.mean(deviations))
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
0.07571215
```

Esta forma de calcular el error devuelve un valor notablemente más alto. Sin embargo, no son comparables, ya que las dimensiones de ambas medidas son diferentes. La primera medida resulta del promedio de los errores absolutos medios de inferir dos coordenadas normalizadas. Es decir, acotadas entre 0 y 1; por lo que al promedio de ambas le ocurre lo mismo.

Por otro lado, la distancia máxima que se puede tener en un espacio bidimensional con coordenadas normalizadas (entre 0 y 1) es $\sqrt{1^2 + 1^2} = \sqrt{2}$. Para que sea equiparable, hay que normalizar el resultado del método dividiéndolo entre $\sqrt{2}$. Esto ya se ha contemplado al implementar el método, devolviendo el resultado normalizado si el argumento `normalize` es verdadero

In [168…
```
norm_deviations = calc_deviation(predictions=model.predict([I_l_test, I_r_test,
print(np.mean(norm_deviations))
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
0.05353655
```

In [ ]:
```
# TEST
index = random.randint(0, len(I_r_test) - 1)


prediction = predict_single(I_l_test[index], I_r_test[index], F_test[index], D_t
print(F_test[index])
print(f"Inferencia: {prediction}")
print(f"Real: {B_test[index]}")
print(f"Desviación: {abs(prediction - B_test[index])}")
print(f"Desviación (normalizada): {calc_deviation(prediction, B_test[index], nor
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
tf.Tensor([0.4296875  0.58515626 0.42083332 0.7916667 ], shape=(4,), dtype=float3
2)
Inferencia: [0.8435137 0.4444585]
Real: [0.8255287  0.40962672]
Desviación: [0.01798505 0.03483179]
Desviación (normalizada): 0.02771926840336835
```

Aprovechando el método `describe()` del *dataframe* de `pandas` se va a declarar un método que convierta los tipos para aplicar `describe()` a los arrays de desviaciones. De esta forma, se pueden visualizar varias características de los arrays.

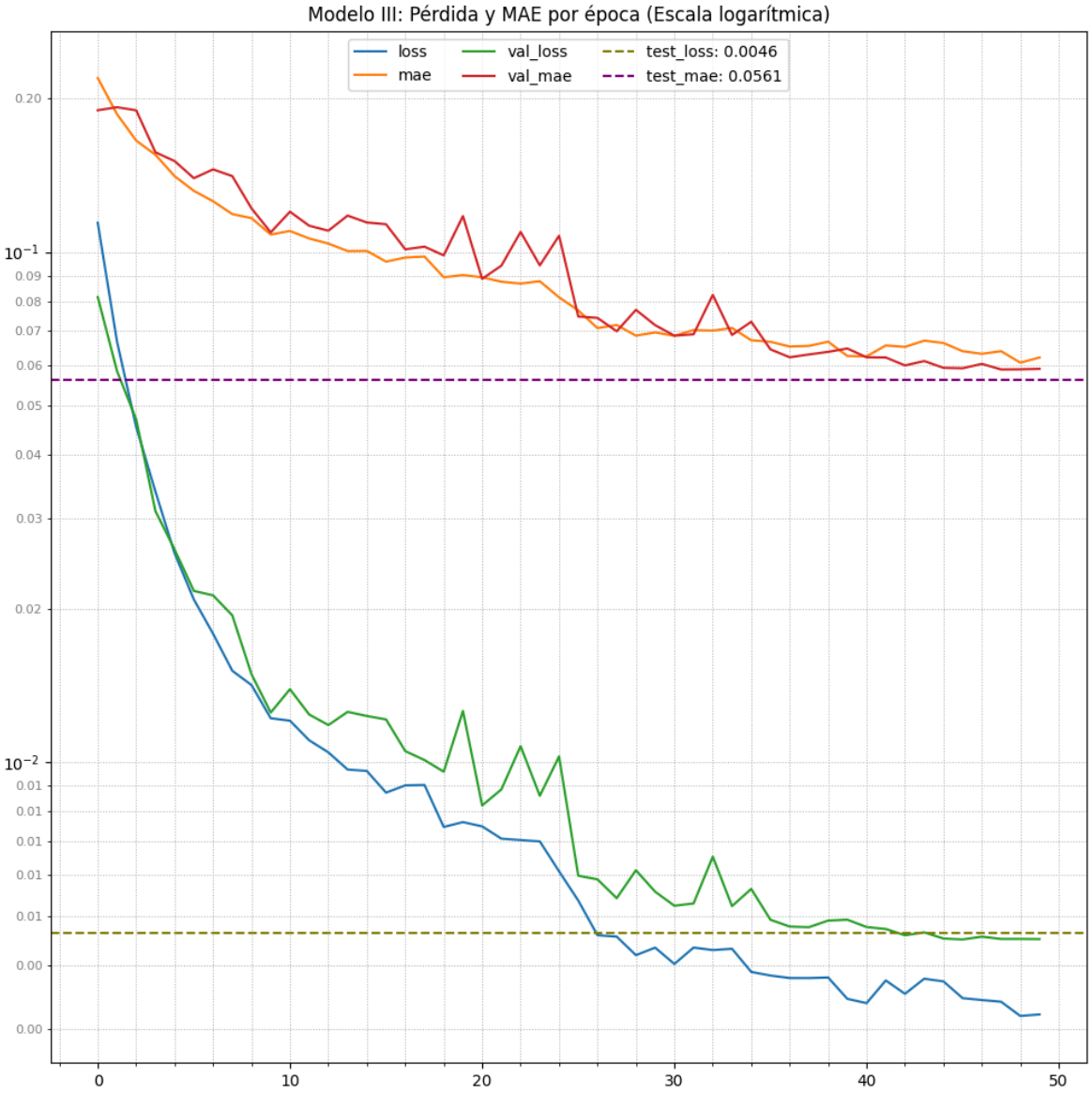
In [170…
```
def descript_array(array):
    return pd.DataFrame(array).describe()
```

Muestra de resultados

Se vuelve a renderizar el gráfico de la historia del entrenamiento junto con los valores de la evaluación

In [173…

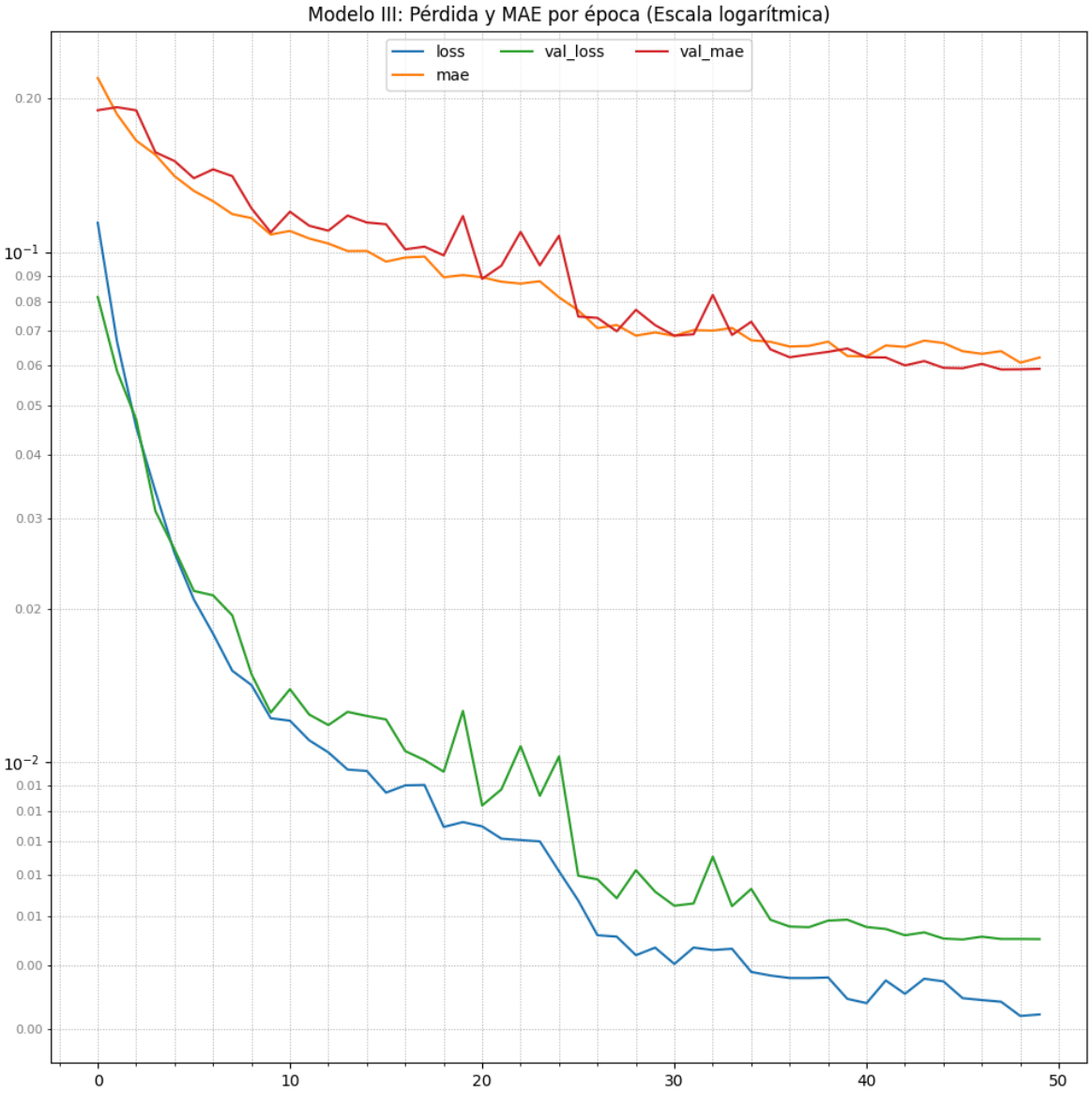
```
pd.DataFrame({
    'loss': train_history.history['loss'],
    'mae': train_history.history['mae'],
    'val_loss': train_history.history['val_loss'],
    'val_mae': train_history.history['val_mae']
}).plot(figsize=(10, 10))
plt.grid(which='both', axis='both', linestyle=':', linewidth=0.7)
plt.minorticks_on()
plt.gca().set_yscale('log')

# Set minor locator for log scale
plt.gca().yaxis.set_minor_locator(mticker.LogLocator(base=10.0, subs=np.arange(1
plt.gca().yaxis.set_minor_formatter(mticker.ScalarFormatter())
plt.tick_params(axis='y', which='minor', labelsize=8, labelcolor='gray', labelle

plt.axhline(y=eval_results[0], color='olive', linestyle='--', label=f'test_loss:
plt.axhline(y=eval_results[1], color='purple', linestyle='--', label=f'test_mae:
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1), ncol=3)
plt.title('Modelo III: Pérdida y MAE por época (Escala logarítmica)')
plt.tight_layout()
plt.show()
```



In [174…

```
pd.DataFrame({
    'loss': train_history.history['loss'],
```

```
    'mae': train_history.history['mae'],
    'val_loss': train_history.history['val_loss'],
    'val_mae': train_history.history['val_mae']
}).plot(figsize=(10, 10))
plt.grid(which='both', axis='both', linestyle=':', linewidth=0.7)
plt.minorticks_on()
plt.gca().set_yscale('log')

# Set minor locator for log scale
plt.gca().yaxis.set_minor_locator(mticker.LogLocator(base=10.0, subs=np.arange(1
plt.gca().yaxis.set_minor_formatter(mticker.ScalarFormatter())
plt.tick_params(axis='y', which='minor', labelsize=8, labelcolor='gray', labelle

# plt.axhline(y=eval_results[0], color='olive', linestyle='--', label=f'test_los
# plt.axhline(y=eval_results[1], color='purple', linestyle='--', label=f'test_ma
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1), ncol=3)
plt.title('Modelo III: Pérdida y MAE por época (Escala logarítmica)')
plt.tight_layout()
plt.show()
```



Métricas absolutas:

$$X_{AbsDevMean} = \frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} |(x_i - \hat{x}_i)|$$

$$Y_{AbsDevMean} = \frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} |(y_i - \hat{y}_i)|$$

$$XY_{NormEucldAbsDevMean} = \frac{1}{2\sqrt{2}} \sqrt{X_{AbsDevMean}^2 + Y_{AbsDevMean}^2}$$

Métricas no absolutas:

$$X_{DevMean} = \frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} (x_i - \hat{x}_i)$$

$$X_{DevStd} = \sqrt{\frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} ((x_i - \hat{x}_i) - X_{DevMean})^2}$$

$$Y_{DevMean} = \frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} (y_i - \hat{y}_i)$$

$$Y_{DevStd} = \sqrt{\frac{1}{U.S.C.} \sum_{i=1}^{U.S.C.} ((y_i - \hat{y}_i) - Y_{DevMean})^2}$$

$$XY_{NormEucldDevMean} = \frac{1}{2\sqrt{2}} \sum_{i=1}^{U.S.C.} \sqrt{X_{DevMean}^2 + Y_{DevMean}^2}$$

$$XY_{NormEucldDevStd} = \frac{1}{2\sqrt{2}} \sum_{i=1}^{U.S.C.} \sqrt{X_{DevStd}^2 + Y_{DevStd}^2}$$

$$XY_{DevMean} = \frac{1}{2\sqrt{2}} \sum_{i=1}^{U.S.C.} \sqrt{X_{DevMean}^2 + Y_{DevMean}^2}$$

$$XY_{DevStd} = \frac{1}{2}(X_{DevStd}^2 + Y_{DevStd}^2)$$

Con el método `descript_array()` y `pandas` se muestra un análisis de los arrays de desviaciones tanto en vertical y horizontal como en diagonal normalizadas

In [175…

```
df_error_metrics = pd.concat([
        descript_array(norm_deviations),
        descript_array(abs(np.array(B_test-predictions))),
        descript_array(np.array(B_test-predictions))
        ], axis=1)
df_error_metrics.columns = ['norm_abs_xy_dev', 'x_abs_dev', 'y_abs_dev', 'x_dev'

test_accuracy = eval_results[0]
norm_abs_xy_dev_mean = df_error_metrics['norm_abs_xy_dev']['mean']
norm_abs_xy_dev_std = df_error_metrics['norm_abs_xy_dev']['std']
x_abs_dev_mean = df_error_metrics['x_abs_dev']['mean']
x_abs_dev_std = df_error_metrics['x_abs_dev']['std']
y_abs_dev_mean = df_error_metrics['y_abs_dev']['mean']
y_abs_dev_std = df_error_metrics['y_abs_dev']['std']
x_dev_mean = abs(df_error_metrics['x_dev']['mean'])
x_dev_std = df_error_metrics['x_dev']['std']
y_dev_mean = abs(df_error_metrics['y_dev']['mean'])
y_dev_std = df_error_metrics['y_dev']['std']
xy_euclidean_mean = math.sqrt(df_error_metrics['x_dev']['mean']**2 + df_error_me
xy_euclidean_std = math.sqrt(df_error_metrics['x_dev']['std']**2 + df_error_metr
xy_mean = (abs(df_error_metrics['x_dev']['mean']) + abs(df_error_metrics['y_dev'
xy_std = (df_error_metrics['x_dev']['std'] + df_error_metrics['y_dev']['std']) /

data = {
    'TestOverallAcc': test_accuracy,
    'XYNormEucldAbsDevMean': norm_abs_xy_dev_mean,
    'XYNormEucldAbsDevStd': norm_abs_xy_dev_std,
    'XAbsDevMean': x_abs_dev_mean,
```

```
    'XAbsDevStd': x_abs_dev_std,
    'YAbsDevMean': y_abs_dev_mean,
    'YAbsDevStd': y_abs_dev_std,
    'XDevMean': x_dev_mean,
    'XDevStd': x_dev_std,
    'YDevMean': y_dev_mean,
    'YDevStd': y_dev_std,
    'XYNormEucldDevMean': xy_euclidean_mean,
    'XYNormEucldDevStd': xy_euclidean_std,
    'XYDevMean': xy_mean,
    'XYDevStd': xy_std,
}

df_cv_metrics = pd.DataFrame([data])
#df_cv_metrics = df_cv_metrics.set_index('UserCode')
#df_cv_metrics.drop(6, inplace=True)

correlation = df_cv_metrics.select_dtypes(include=[np.number]).corr()
```

In [176…

```
df_cv_metrics_to_show = df_cv_metrics.copy()
df_cv_metrics_to_show = df_cv_metrics_to_show.transpose()
df_cv_metrics_to_show.columns = [NOTEBOOK_VERSION]
df_cv_metrics_to_show
```

Out[176…

	AI_E_V2_2DD_YN96
TestOverallAcc	0.049625
XYNormEucldAbsDevMean	0.053536
XYNormEucldAbsDevStd	0.068969
XAbsDevMean	0.055747
XAbsDevStd	0.079666
YAbsDevMean	0.043515
YAbsDevStd	0.079814
XDevMean	0.001175
XDevStd	0.091292
YDevMean	0.010084
YDevStd	0.080202
XYNormEucldDevMean	0.008281
XYNormEucldDevStd	0.074593
XYDevMean	0.008129
XYDevStd	0.073247

In [177…

```
df_cv_metrics.to_json(f'results/{NOTEBOOK_VERSION}.json', index=False)
```

Validación cruzada

En conjuntos de datos muy pequeños, como el de este proyecto, el procedimiento común de separar una pequeña muestra del conjunto de datos para conformar el conjunto de test, no sirve para hacer una evaluación justa:

- Las muestras provienen de una población demasiado pequeña inevitablemente, ya que hacerla más grande supone una disminución perjudicial del conjunto de entrenamiento. En el caso del proyecto, uno o dos sujetos como mucho
- Al ser tan pequeña, es probable que el caso de uso sea demasiado específico. El rendimiento del modelo en este conjunto no es representativo de un caso de uso genérico. Puede ocurrir que con el sujeto o los sujetos de la muestra funcione mejor o peor de lo habitual.
- Actualmente, la separación de conjuntos se hace en base a los sujetos. Tomar una muestra aleatoria, sin tener en cuenta los sujetos pueden solucionar el problema de la poca generalidad del conjunto de evaluación, pero produciría un sesgo en la evaluación, ya que compartirían sujetos, y habría registros no iguales pero muy similares, que en caso de sobreentrenamiento mostrarían una precisión muy elevada a pesar de que el modelo esté memorizando el set de entrenamiento y sea incapaz de generalizar.

Durante el desarrollo se puede utilizar la evaluación tradicional como referencia, eligiendo con criterio los sujetos para el set de evaluación. Para plasmar los resultados en un artículo o en la memoria de un Trabajo de Fin de Grado, hay que recurrir a métodos más rigurosos en estas circunstancias, como la validación cruzada:

La técnica consiste en dividir el conjunto de datos en múltiples porciones. Una parte grande de ellas se dedican al conjunto de entrenamiento y, el remanente, al conjunto de evaluación o test. Esto se debe repetir varias veces, entrenando, evaluando y variando la repartición de las porciones en cada una. Al terminar, se calcula el promedio de las métricas de todas las iteraciones. El resultado es una estimación muy robusta sobre el rendimiento del modelo.

En este caso, la segmentación en porciones es muy evidente: Mediante sujetos, tal y como se ha hecho anteriormente.

Para hacerlo eficientemente, se van a mantener una relación con las etiquetas de usuario tras el procesado

In [178…
```
# Este cuaderno ocupa una barbaridad de memoria. Imprescincible borrarla antes d
del model

del I_l_train, I_r_train, F_train, D_train, G_train, B_train
del I_l_valid, I_r_valid, F_valid, D_valid, G_valid, B_valid
del I_l_test, I_r_test, F_test, D_test, G_test, B_test

tf.keras.backend.clear_session()

gc.collect()
```

Out[178…
```
0
```

In [179…
```
df12_cv = df_shuffle(df12)
```

In [180…
```
def get_user_code(row, df):
    user_code = df['codigoUsuario'][row]
    return user_code
```

In [181…
```
get_user_code(676, df12_cv)
```

Out[181…
```
'QXYT'
```

In [182…
```
j, i = 0, 0
image_error_count, face_detection_error_count, eyes_detection_error_count = 0, 0

def generate_labeled_IFDGB(df12_temp):
    global EYE_CROP_SIZE, j, i, image_error_count, face_detection_error_count, e
    I_l, I_r, F, D, G, B, users = [], [], [], [], [], [], []
    j, i = 0, 0
    image_error_count, face_detection_error_count = 0, 0

    for row in df12_temp.index:
        try:
            bgr_img, rgb_img, gray_img = get_image(row, df12_temp)
        except:
            image_error_count += 1
            continue

        try:
            x1, y1, x2, y2, left_eye_img, right_eye_img = detect_face_crop_eyes_
        except:
            face_detection_error_count += 1
            continue

        left_eye_img = square_crop_resize(left_eye_img, EYE_CROP_SIZE)
        right_eye_img = square_crop_resize(right_eye_img, EYE_CROP_SIZE)

        # ERROR DE FORMATO: En algunos datasets antiguos, la columna 'anchoImage

        I_l.append(left_eye_img/255)
        I_r.append(right_eye_img/255)
        F.append(np.array([x1/df12_temp['anchoImagenCam'][row], x2/df12_temp['an
        D.append(np.array([df12_temp['ancho_cm'][row], df12_temp['alto_cm'][row]
        G.append(np.array([1] if df12_temp['gafas'][row] == 'Si' else [0]))
        B.append(np.array([df12_temp["pos_x"][row]/df12_temp["ancho_px"][row], d
        users.append(get_user_code(row, df12_temp))

        j += 1
        clear_output(wait=True)
        print(f"({row + 1}/{len(df12_temp.index)})")
        print(f"Errores de imagen: {image_error_count}")
        print(f"Errores de detección facial: {face_detection_error_count}")
        print(f"Total de imágenes procesadas: {j}/{len(df12_temp)}")

    clear_output(wait=True)
    print("Finalizado")

    return I_l, I_r, F, D, G, B, users
```

In [183…
```
I_l_full, I_r_full, F_full, D_full, G_full, B_full, users_full  = generate_label
```

```
Finalizado
```

Para distinguir las muestras por sujeto, se definen diccionarios, que tienen claves en vez de índices como las listas, lo que permite identificar los valores mediante objetos. En este caso, mediante una cadena que contiene el código de sujeto.

In [184…
```python
I_l = {}
I_r = {}
F = {}
D = {}
G = {}
B = {}

for user in df12_cv['codigoUsuario'].unique():
    I_l[user] = []
    I_r[user] = []
    F[user] = []
    D[user] = []
    G[user] = []
    B[user] = []
    for i in range(len(users_full)):
        if users_full[i] == user:
            I_l[user].append(I_l_full[i])
            I_r[user].append(I_r_full[i])
            F[user].append(F_full[i])
            D[user].append(D_full[i])
            G[user].append(G_full[i])
            B[user].append(B_full[i])
```

In [185…
```python
for user in df12_cv['codigoUsuario'].unique():
    I_l_mirror, I_r_mirror, F_mirror, D_mirror, G_mirror, B_mirror = augmentatio
    I_l[user], I_r[user], F[user], D[user], G[user], B[user] = append_random_sam
        I_l[user], I_r[user], F[user], D[user], G[user], B[user],
        I_l_mirror, I_r_mirror, F_mirror, D_mirror, G_mirror, B_mirror,
        fraction=0.5
    )
```

In [186…
```python
for user in df12_cv['codigoUsuario'].unique():
    I_l_noisy, I_r_noisy, F_noisy, D_noisy, G_noisy, B_noisy = augmentation_add_
    I_l[user], I_r[user], F[user], D[user], G[user], B[user] = append_random_sam
        I_l[user], I_r[user], F[user], D[user], G[user], B[user],
        I_l_noisy, I_r_noisy, F_noisy, D_noisy, G_noisy, B_noisy,
        fraction=0.3
    )
```

In [187…
```python
for user in df12_cv['codigoUsuario'].unique():
    I_l[user] = tf.convert_to_tensor(I_l[user])
    I_r[user] = tf.convert_to_tensor(I_r[user])
    F[user] = tf.convert_to_tensor(F[user])
    D[user] = tf.convert_to_tensor(D[user])
    G[user] = tf.convert_to_tensor(G[user])
    B[user] = tf.convert_to_tensor(B[user])
```

In [188…
```python
EPOCHS = 30
```

Dado que el número de épocas es bajo, no hacen falta callbacks en esta fase.

In [189…

```
histories = []
test_accuracies = []
user_sample_counts = []
detect_fail_rates = []
wearing_glasses = []
training_set_sizes = []
camera_mpixs = []
screen_sizes = []
norm_abs_xy_dev_means = []
norm_abs_xy_dev_stds = []
x_abs_dev_means = []
x_abs_dev_stds = []
y_abs_dev_means = []
y_abs_dev_stds = []
x_dev_means = []
x_dev_stds = []
y_dev_means = []
y_dev_stds = []
xy_euclidean_means = []
xy_euclidean_stds = []
xy_means = []
xy_stds = []
i = 0

for user in I_l.keys():
    print(f"Training started. Testing with user {user} ({i+1}/{len(I_l.keys())})

    if i > 0:
        del model
        gc.collect()
        tf.keras.backend.clear_session()

    # Separar los datos del sujeto seleccionado en la iteración de los demás

    I_l_train = tf.concat([I_l[u] for u in I_l.keys() if u != user], axis=0)
    I_r_train = tf.concat([I_r[u] for u in I_r.keys() if u != user], axis=0)
    F_train = tf.concat([F[u] for u in F.keys() if u != user], axis=0)
    D_train = tf.concat([D[u] for u in D.keys() if u != user], axis=0)
    G_train = tf.concat([G[u] for u in G.keys() if u != user], axis=0)
    B_train = tf.concat([B[u] for u in B.keys() if u != user], axis=0)

    I_l_test = I_l[user]
    I_r_test = I_r[user]
    F_test = F[user]
    D_test = D[user]
    G_test = G[user]
    B_test = B[user]


    model = build_model()

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        loss=tf.keras.losses.Huber(delta=1.0),
        metrics=['mae']
    )


    history = model.fit(
```

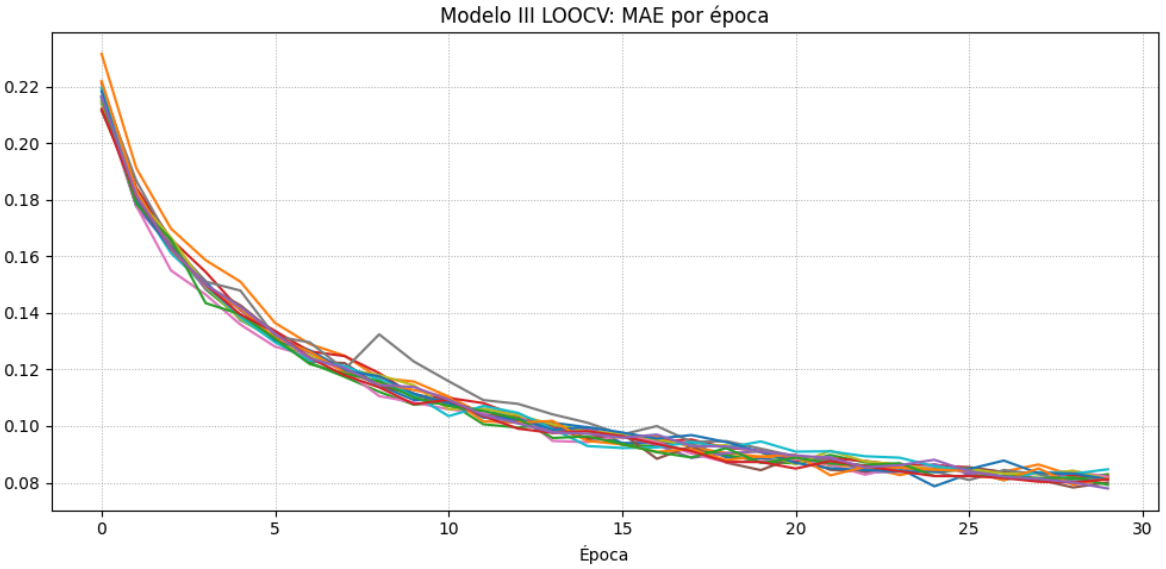
```
        x=[I_l_train, I_r_train, F_train, D_train, G_train],
        y=B_train,
        epochs=EPOCHS,
        batch_size=BATCH_SIZE,

        verbose=1
    )

    clear_output(wait=True)

    eval_result = model.evaluate([I_l_test, I_r_test, F_test, D_test, G_test], B
    predictions = model.predict([I_l_test, I_r_test, F_test, D_test, G_test])

    norm_deviations = calc_deviation(predictions=model.predict([I_l_test, I_r_te
    print(np.mean(norm_deviations))

    df_error_metrics = pd.concat([
        descript_array(norm_deviations),
        descript_array(abs(np.array(B_test-predictions))),
        descript_array(np.array(B_test-predictions))
        ], axis=1)
    df_error_metrics.columns = ['norm_abs_xy_dev', 'x_abs_dev', 'y_abs_dev', 'x_

    # Métricas del conjunto
    user_sample_counts.append(len(I_l_test))
    camera_mpixs.append(int(df12_cv[df12_cv['codigoUsuario'] == user]['anchoImag
    screen_sizes.append(df12_cv[df12_cv['codigoUsuario'] == user]['diagonal_Pulg
    detect_fail_rates.append(1 - len(F[user]) / (len(df12_cv[df12_cv['codigoUsua
    wearing_glasses.append(G_test.numpy()[0][0])
    # Métricas de desviación
    norm_abs_xy_dev_means.append(df_error_metrics['norm_abs_xy_dev']['mean'])
    norm_abs_xy_dev_stds.append(df_error_metrics['norm_abs_xy_dev']['std'])
    x_abs_dev_means.append(df_error_metrics['x_abs_dev']['mean'])
    x_abs_dev_stds.append(df_error_metrics['x_abs_dev']['std'])
    y_abs_dev_means.append(df_error_metrics['y_abs_dev']['mean'])
    y_abs_dev_stds.append(df_error_metrics['y_abs_dev']['std'])
    x_dev_means.append(abs(df_error_metrics['x_dev']['mean']))
    x_dev_stds.append(df_error_metrics['x_dev']['std'])
    y_dev_means.append(abs(df_error_metrics['y_dev']['mean']))
    y_dev_stds.append(df_error_metrics['y_dev']['std'])
    xy_euclidean_means.append(math.sqrt(df_error_metrics['x_dev']['mean']**2 + d
    xy_euclidean_stds.append(math.sqrt(df_error_metrics['x_dev']['std']**2 + df_
    xy_means.append((abs(df_error_metrics['x_dev']['mean']) + abs(df_error_metri
    xy_stds.append((df_error_metrics['x_dev']['std'] + df_error_metrics['y_dev']

    histories.append(history)
    test_accuracies.append(eval_result[1])
    training_set_sizes.append(len(I_l_train))

    clear_output(wait=True)

    i+=1
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 452ms/step - loss: 0.0312 - mae: 0.1985
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 934ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
0.21486896816511253
```

In [190…

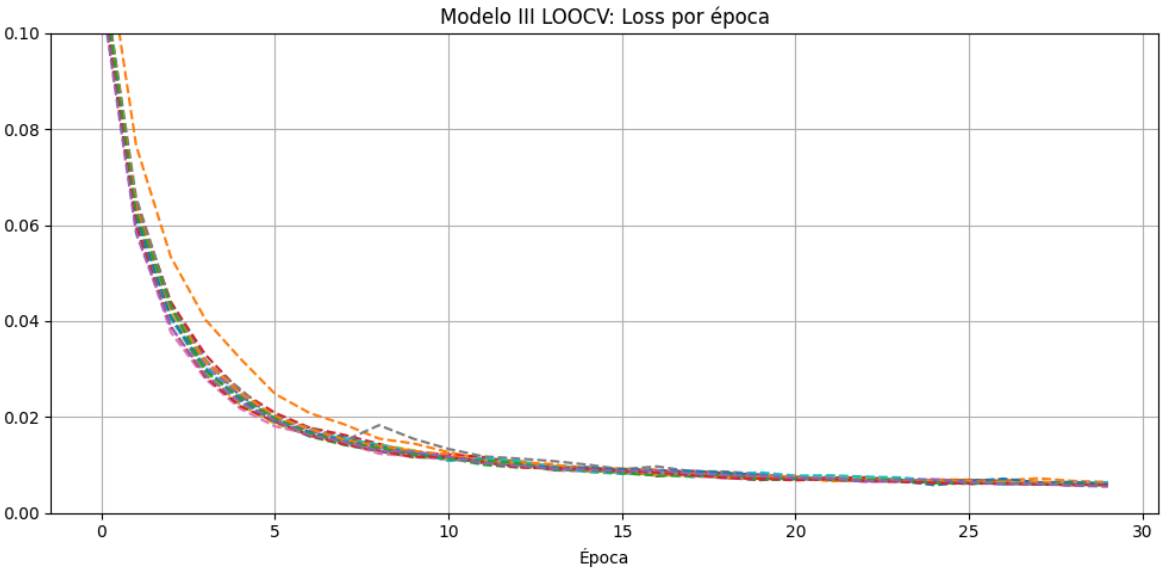
```
plt.figure(figsize=(10, 5))
```

```
for i, history in enumerate(histories):
    plt.plot(history.history['mae'], label=f'Model {i+1} Train')

plt.title('Modelo III LOOCV: MAE por época')
plt.xlabel('Época')
plt.tight_layout()
plt.grid(which='both', axis='both', linestyle=':', linewidth=0.7)
plt.show()
```



In [191…

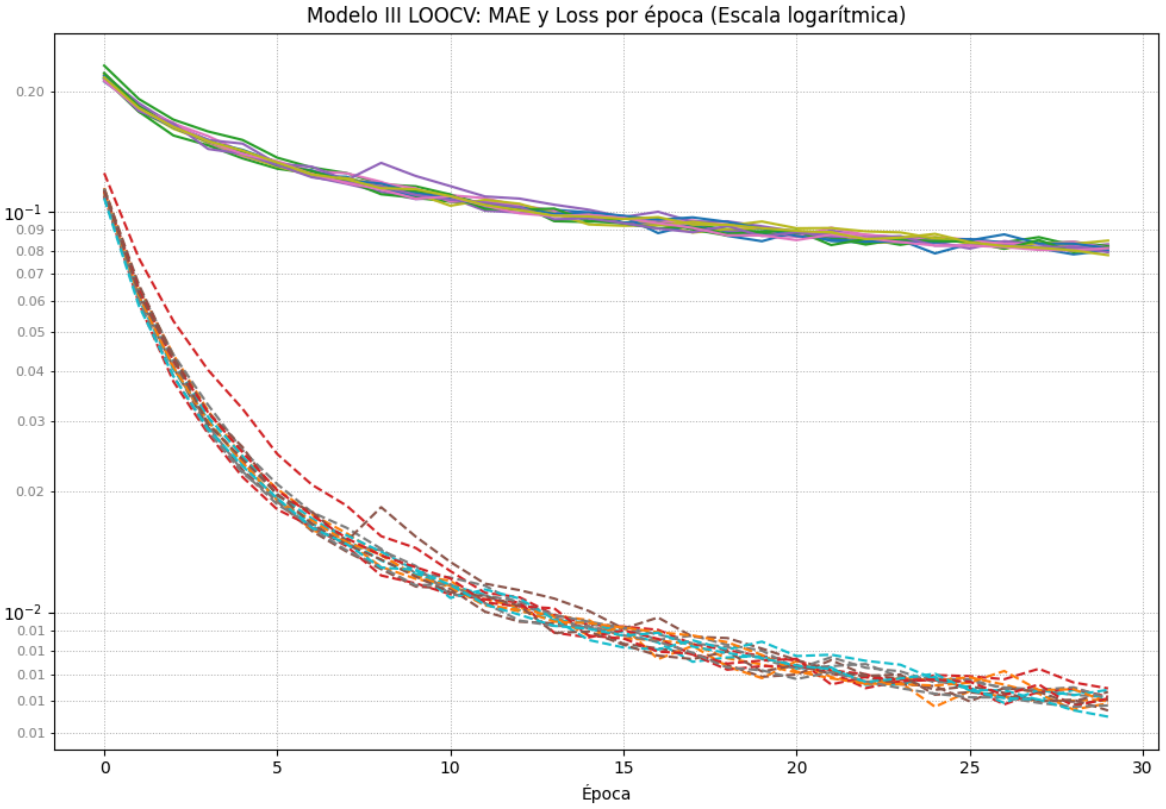
```
plt.figure(figsize=(10, 5))

for i, history in enumerate(histories):
    plt.plot(history.history['loss'], linestyle='--', label=f'Model {i+1} Val')

plt.title('Modelo III LOOCV: Loss por época')
plt.xlabel('Época')
plt.ylim(0, 0.1)
plt.tight_layout()
plt.grid(True)
plt.show()
```



In [192…

```
plt.figure(figsize=(10, 7))

for i, history in enumerate(histories):
```

```
    plt.plot(history.history['mae'], label=f'Model {i+1} Train')
    plt.plot(history.history['loss'], linestyle='--', label=f'Model {i+1} Val')

plt.title('Modelo III LOOCV: MAE y Loss por época (Escala logarítmica)')
plt.xlabel('Época')
plt.gca().set_yscale('log')
plt.gca().yaxis.set_minor_locator(mticker.LogLocator(base=10.0, subs=np.arange(1
plt.gca().yaxis.set_minor_formatter(mticker.ScalarFormatter())
plt.tick_params(axis='y', which='minor', labelsize=8, labelcolor='gray', labelle
plt.tight_layout()
plt.grid(which='both', axis='both', linestyle=':', linewidth=0.7)
plt.show()
```



In [193…

```
data = {
    'UserCode': F.keys(),
    'UserSampleCount': user_sample_counts,
    'WearingGlasses': wearing_glasses,
    'TrainingSetSize': training_set_sizes,
    'DetectionFailRate': detect_fail_rates,
    'CameraMPix': camera_mpixs,
    'ScreenSize': screen_sizes,
    'TestOverallAcc': test_accuracies,
    'XYNormEucldAbsDevMean': norm_abs_xy_dev_means,
    'XYNormEucldAbsDevStd': norm_abs_xy_dev_stds,
    'XAbsDevMean': x_abs_dev_means,
    'XAbsDevStd': x_abs_dev_stds,
    'YAbsDevMean': y_abs_dev_means,
    'YAbsDevStd': y_abs_dev_stds,
    'XDevMean': x_dev_means,
    'XDevStd': x_dev_stds,
    'YDevMean': y_dev_means,
    'YDevStd': y_dev_stds,
    'XYNormEucldDevMean': xy_euclidean_means,
    'XYNormEucldDevStd': xy_euclidean_stds,
    'XYDevMean': xy_means,
```

```
    'XYDevStd': xy_stds,
}

df_cv_metrics = pd.DataFrame(data)
#df_cv_metrics = df_cv_metrics.set_index('UserCode')
#df_cv_metrics.drop(6, inplace=True)

correlation = df_cv_metrics.select_dtypes(include=[np.number]).corr()

mean_row = df_cv_metrics.mean(numeric_only=True)
mean_row.name = 'Mean'
df_cv_metrics = pd.concat([df_cv_metrics, mean_row.to_frame().T])
```

Guardado en ficheros para su posterior análisis

In [194…
```
df_cv_metrics.to_json(f'results/{NOTEBOOK_VERSION}_LOOCV.json', index=False)
```

In [195…
```
# Valores promedios
print(f"XY Norm Euclidean Abs Dev Mean:\t {np.mean(norm_abs_xy_dev_means):.5f}")
print(f"XY Norm Euclidean Abs Dev Std:\t {np.mean(norm_abs_xy_dev_stds):.5f}")
print(f"X Abs Dev Mean:\t\t\t {np.mean(x_abs_dev_means):.5f}")
print(f"X Abs Dev Std:\t\t\t {np.mean(x_abs_dev_stds):.5f}")
print(f"Y Abs Dev Mean:\t\t\t {np.mean(y_abs_dev_means):.5f}")
print(f"Y Abs Dev Std:\t\t\t {np.mean(y_abs_dev_stds):.5f}")
print(f"X Dev Mean:\t\t\t {np.mean(x_dev_means):.5f}")
print(f"X Dev Std:\t\t\t {np.mean(x_dev_stds):.5f}")
print(f"Y Dev Mean:\t\t\t {np.mean(y_dev_means):.5f}")
print(f"Y Dev Std:\t\t\t {np.mean(y_dev_stds):.5f}")
print()
print(f"XY Norm Euclidean Dev Mean:\t {np.mean(xy_euclidean_means):.5f}")
print(f"XY Norm Euclidean Dev Std:\t {np.mean(xy_euclidean_stds):.5f}")
print()
print(f"XY Dev Mean (Accuracy):\t\t {np.mean(xy_means):.5f}")
print(f"XY Dev Std (Precision):\t\t {np.mean(xy_stds):.5f}")
```

```
XY Norm Euclidean Abs Dev Mean:  0.25808
XY Norm Euclidean Abs Dev Std:   0.12582
X Abs Dev Mean:                  0.23036
X Abs Dev Std:                   0.16019
Y Abs Dev Mean:                  0.23553
Y Abs Dev Std:                   0.16842
X Dev Mean:                      0.09578
X Dev Std:                       0.24011
Y Dev Mean:                      0.14046
Y Dev Std:                       0.23184

XY Norm Euclidean Dev Mean:      0.12842
XY Norm Euclidean Dev Std:       0.24625

XY Dev Mean:                     0.11812
XY Dev Std:                      0.24011
```

In [196…
```
print(f"Errores de imagen: {image_error_count}")
print(f"Errores de detección facial: {face_detection_error_count}")
print(f"Total de imágenes correctas: {j}/{len(df12_cv)}")
print(f"Error relativo: {((len(df12_cv)-j)/(len(df12_cv)))}")
```

Errores de imagen: 15
Errores de detección facial: 96
Total de imágenes correctas: 1206/1317
Error relativo: 0.08428246013667426

Apéndice C

Adaptación y mejoras a la aplicación de adquisición de datos de entrenamiento de *Eye-Tracking*

Raúl Barba Alonso desarrolló un sistema conceptual de adquisición de datos de entrenamiento pare *Eye-Tracking* como Trabajo de Fin de Grado. El mismo consistió en una aplicación web con un cliente desarrollado en HTML y JavaScript puros (es decir, sin *frameworks* ni librerías) con dos videojuegos interactivos en los que el usuario sigue con el ratón un punto que se desplaza por la pantalla mientras se toman imágenes con la cámara web del equipo; y un *backend* desarrollado en PHP que gestiona y almacena los datos del cliente en el sistema de ficheros del servidor y en una base de datos MySQL.

Una parte muy importante del presente proyecto consistió en recuperar el sistema de Raúl, adecuarlo a las nuevas necesidades y ponerlo a funcionar en un entorno real para recopilar datos para realizar su análisis y entrenar redes neuronales.

Estas son las funcionalidades de la versión de la aplicación utilizada en este proyecto, tras las mejoras practicadas [1]:

La interfaz, maquetada en HTML y estilizada con CSS, presenta una página de inicio con una barra de navegación que da acceso al resto de pantallas de la aplicación. Cuenta con formularios de registro de usuarios y de inicio de sesión, imprescindibles para asociar las capturas de los experimentos con rasgos almacenados de la persona que los realiza: género, edad, color de ojos, realización de experimentos con gafas o lentes de contacto:

[1] Para ver los detalles sobre la implementación original, consúltese el Trabajo de Raúl Barba [8]. Nótese que ciertos aspectos mostrados en su Memoria, como por ejemplo, el volcado de muestras a ficheros JSON, o las escrituras en base de datos, no eran funcionales en la versión de partida proporcionada para este Trabajo.

Figura C.1: Página principal de la aplicación de adquisición.

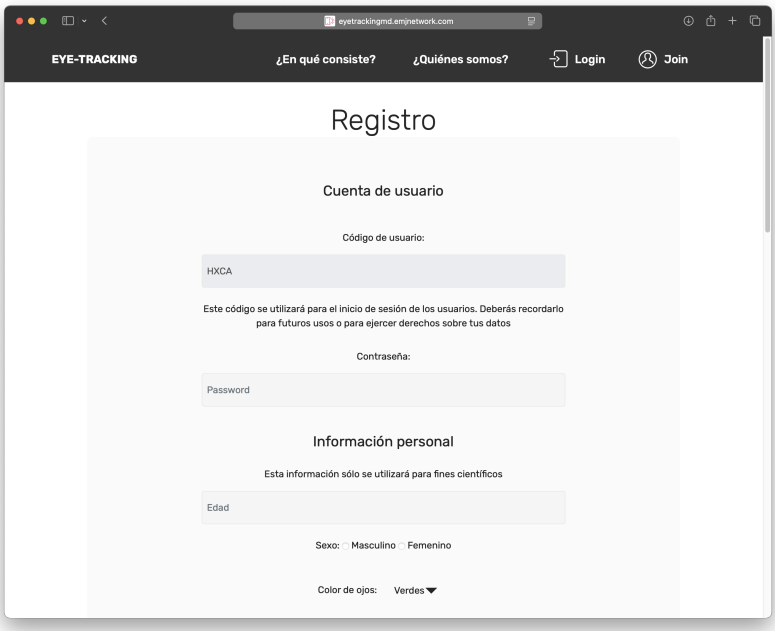


Figura C.2: Formulario de registro de la aplicación de adquisición.

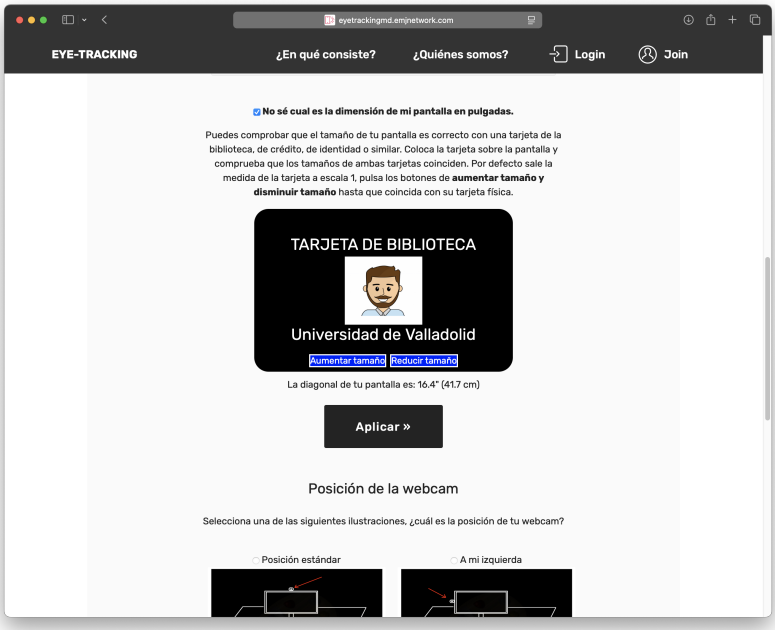


Figura C.3: Mecanismo de estimación del tamaño de la pantalla.

diagonal en pulgadas de la pantalla y posición de la cámara respecto a la pantalla. Todos ellos requeridos por el formulario de registro. A mayores, el mismo genera un código único aleatorio de cuatro letras que identifica al usuario. También, para estimar la diagonal de la pantalla en caso de que el usuario no la conozca, cuenta con un elemento que consiste en una ilustración de una tarjeta con controles para redimensionarla

Su funcionamiento es el siguiente: El usuario toma una tarjeta física y la superpone sobre la ilustración en pantalla. Ajusta el tamaño de esta con los controles que se proporcionan para que coincida con la física. Una vez esto suceda, el usuario hace clic en "Aplicar" para copiar el tamaño de diagonal calculado al formulario. En ordenadores portátiles, se ha comprobado que el mecanismo funciona correctamente [8].

Una vez se ha hecho el registro, se muestra por primera vez la página de selección de experimentos, término con el que la web (y esta memoria de ahora en adelante) se refiere a los juegos desarrollados en JavaScript que recopilan datos de entrenamiento y los envían al servidor web mediante Asynchronous JavaScript And XML (AJAX).

El primero de ellos muestra una circunferencia estática que a lo largo de un minuto que dura el experimento, aparece periódicamente en posiciones aleatorias de la pantalla. Se pide al usuario que haga clic sobre él. Cada vez que esto sucede, se toma una foto con la cámara del equipo y se envía mediante AJAX al servidor junto con las coordenadas de la circunferencia, el clic del usuario y otros parámetros ordinales y temporales internos del experimento. El formato con el que se guarda una captura en el servidor es: La imagen de

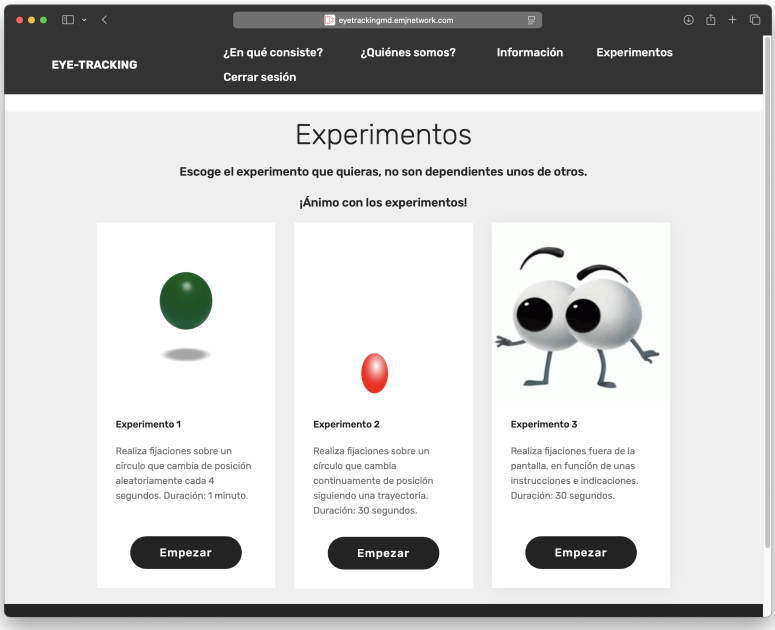


Figura C.4: Página de selección de Experimentos de la aplicación de adquisición.

la cámara en formato *png* en el sistema de ficheros y un registro en una tabla con todos los datos asociados, incluyendo la referencia a la imagen y las posiciones en pantalla del puntero del ratón, que son las potenciales etiquetas para los datos de entrenamiento. Como retroalimentación, la circunferencia muestra un aura rojo cuando este proceso se completa con éxito. Además, se comprueba y se almacena en cada iteración del ciclo la distancia del punto de la pantalla sobre el que se hizo clic al punto en el que se renderizó la circunferencia, a fin de implementar un sistema de gamificación que premia la exactitud del usuario a la hora de hacer clic sobre la circunferencia, mostrando la puntuación obtenida al concluir el experimento, junto a una valoración personal (de otras realizaciones del experimento para el usuario actual) y global (para todas las realizaciones del experimento registradas) de los 10 mejores registros.

El segundo experimento es muy similar. Gran parte del código del primero se reutiliza para este. Las diferencias radican en que la circunferencia deja de ser estática para moverse continuamente. Por esto, no se realizan capturas en función de los clics, sino periódicamente. Al usuario se le solicita que siga en la medida de lo posible la circunferencia con el ratón. Periódicamente también se comprueba la distancia del cursor del ratón a la circunferencia para calcular la puntuación para el sistema de gamificación. En este caso, cuenta con un umbral. Si la distancia lo sobrepasa, la iteración se marca como inválida. Si no lo alcanza, es válida. La puntuación total resulta de dividir el número de iteraciones correctas entre el total de iteraciones.

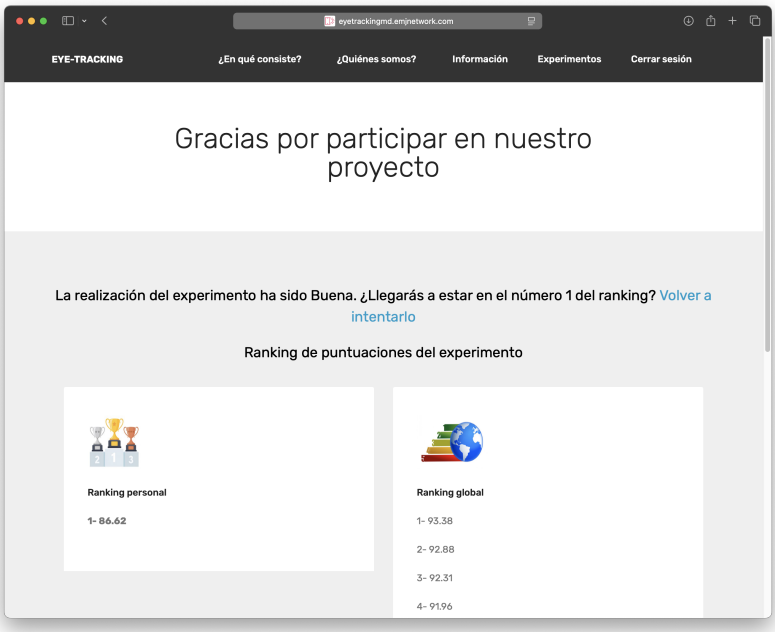


Figura C.5: Pantalla de gamificación de los Experimentos 1 y 2

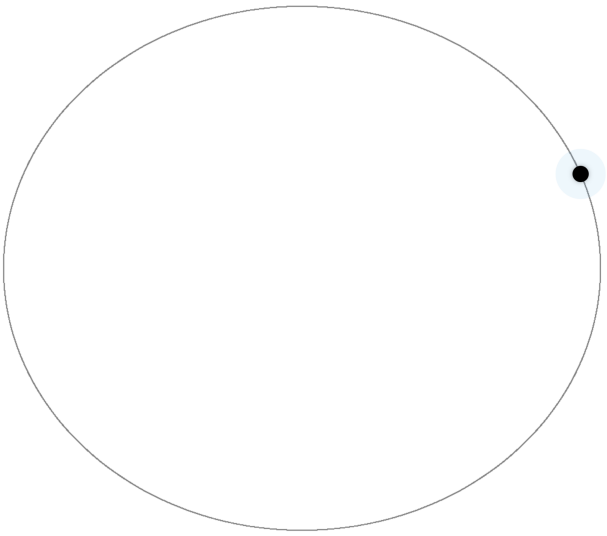


Figura C.6: Interfaz del Experimento 2.

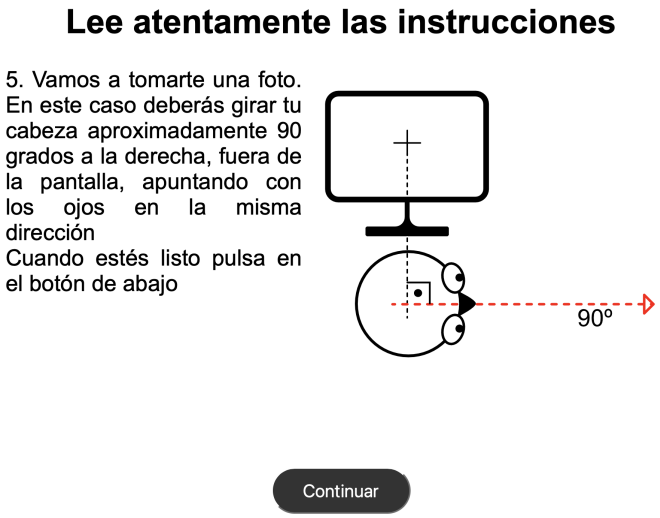


Figura C.7: Interfaz del Experimento 3.

El Experimento 3 proporciona instrucciones al usuario para desviar la cabeza del centro de la pantalla o apartar la mirada de la misma. Se consideran situaciones anómalas porque se alejan de lo que sería objeto de inferencia de los modelos principales del sistema, aquellos entrenados con datos generados con los Experimentos 1 y 2. El objetivo de este es generar muestras para entrenar un modelo clasificador binario, que sirva para detectar si el sujeto dirige o no su mirada hacia la pantalla de su dispositivo.

El paso previo a la realización de cualquier experimento es una comprobación del tipo de dispositivo a partir de la información del agente de usuario incluida en las cabeceras HTTP, de forma transparente al usuario. Puesto que el sistema se ha concebido para ser utilizado en ordenadores personales de sobremesa y portátiles, si detecta que el navegador pertenece a un teléfono móvil o a una tableta, muestra un mensaje de error y no continúa al siguiente paso.

A mayores, existe una opción en el menú de navegación que se hace visible solo cuando la sesión está iniciada. Muestra información sobre el usuario en cuestión y permite modificarla o borrar su cuenta de la base de datos.

Por último, existe un panel de visualización de muestras disponible solo para los usuarios con privilegios de administrador, cuyo principal propósito es ayudar y agilizar el proceso de depuración.

El proceso tuvo las siguientes fases:

1. Puesta en marcha del servidor del grupo de investigación:

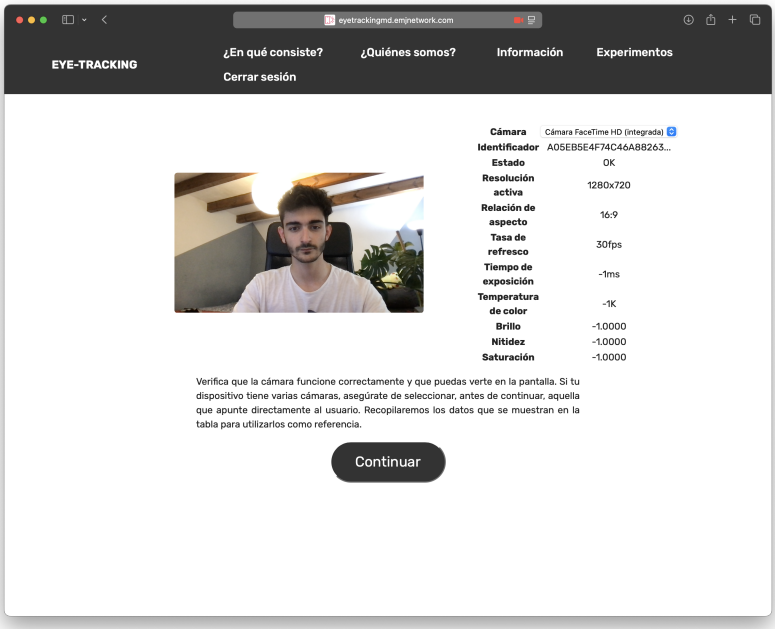


Figura C.8: Prueba inicial del dispositivo de captura.

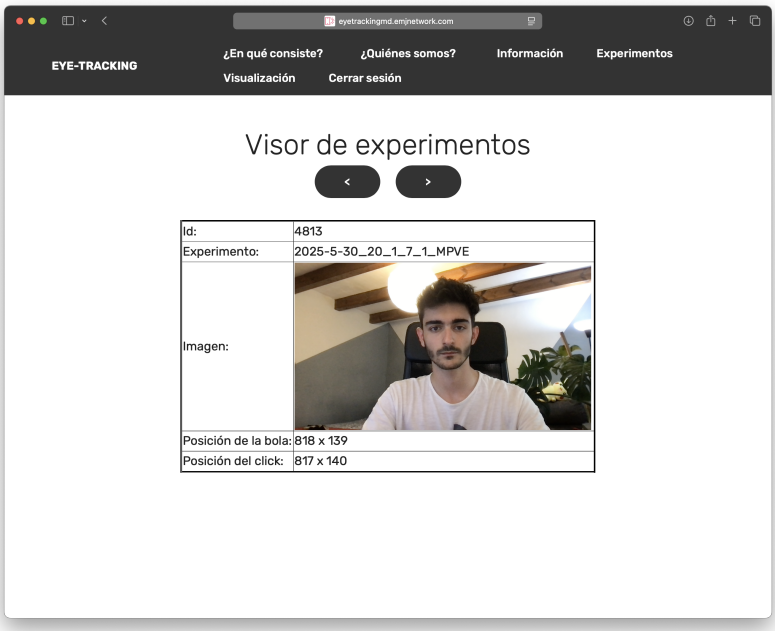


Figura C.9: Panel de visualización de muestras del administrador.

El *hostname* de esta máquina, y el nombre con el que se hará referencia a la misma de ahora en adelante, es `saqqara`. Es una máquina virtual situada en una red privada con traducción de direcciones, o NAT, dentro de la red del grupo. Esto tiene relevancia más adelante.

`saqqara` ejecuta sobre una distribución de Linux, Ubuntu, el servidor web Apache, el servidor de base de datos relacional MySQL y además proporciona el sistema de ficheros en el que se almacenan las imágenes generadas por los clientes.

Al arrancar la máquina y los servicios por primera vez, se hizo accesible la entidad HTTP del servidor y la página web del proyecto con su interfaz, el registro y control de sesiones de usuarios y los experimentos.

2. Arreglos comentados:

Para adaptar el proyecto que se encontró a las necesidades del trabajo actual, se efectuaron modificaciones y mejoras, pasando así de un concepto a un conjunto funcional adecuado al entorno de trabajo con el que se realizaría el análisis de los datos más adelante.

Por un lado, dado que la aplicación debía ser accesible a través de Internet para permitir la adquisición de datos de voluntarios que participaron en el proyecto, se hicieron las siguientes modificaciones:

- Se sustituyeron los términos y condiciones provisionales anteriores por otros definitivos basados en el Reglamento General de Protección de Datos.
- Para cumplir con dichos términos y condiciones, se modificó el mecanismo de eliminación de cuentas para que al ejecutarlo, además de borrar todos los registros relacionados con el usuario en cuestión de la base de datos, también se eliminen del sistema de ficheros del servidor las imágenes que hubiera generado, y no solo las referencias a las mismas.
- En ambos experimentos se añadió lógica para establecer un umbral máximo aceptable para la distancia entre la circunferencia y el ratón. Cuando este es superado, se asume que el usuario no fijó la mirada correctamente en la circunferencia y la imagen asociada a la iteración no se sube al servidor.
- En el experimento 1, para mejorar la usabilidad, se hizo que en los clics válidos (con desviación inferior al umbral) la circunferencia adicional que muestra la posición del clic se resalte en verde. En caso de no ser válido, se resalta en rojo como lo hacía anteriormente.
- En el experimento 2, debido a que Javascript es monohilo y los tiempos de ejecución varían inevitablemente de una máquina a otra (e incluso puede que de un agente a otro) no se puede garantizar la perfecta sincronía entre los tiempos que se almacenan en la base de datos y los tiempos de los fotogramas asociados en los

vídeos. Además, no es factible ni necesario almacenar todos los fotogramas junto con su registro individual asociado en la base de datos, ya que ello haría la tabla desproporcionadamente grande y los fotogramas contiguos presentan diferencias mínimas entre sí como para suponer una mejora en el entrenamiento frente a la implementación que se ha realizado finalmente, que es similar a la del experimento 1. Se realizan capturas periódicamente. Aquellas en las que la desviación del ratón respecto de la circunferencia dibujada en pantalla sea superior al umbral, se descartan. Aquellas en las que no se supere el umbral, se suben con el mismo formato que en el experimento anterior. Esto acarreó el inconveniente de que al guardar la imagen en memoria, el hilo se bloqueaba hasta completar el proceso, lo que producía irregularidades en el movimiento de la circunferencia. Para evitar bloqueos excesivamente largos que perjudicasen la experiencia, se bloqueó la resolución máxima a 1280x720, bajo la asunción de que no supondrá un problema dado que la mayoría de cámaras integradas de ordenadores portátiles también tiene este máximo.

- Al intentar acceder a la aplicación a través de internet pública vía HTTP (Hasta entonces solo se había accedido vía túnel SSH a `saqqara` mediante la VPN de la UVa) después de hacer accesible la máquina mediante reenvío de puertos en uno de los *firewalls* del grupo, se observó que no era posible ejecutar los experimentos porque la prueba de la cámara siempre concluía con un error relacionado con el navegador y no permitía continuar. Para el acceso a las cámaras, el script de `webcamtests.com` utiliza la API `Media Devices`, que solo funciona en contextos seguros [55] (Esto explica por qué no se había detectado el problema cuando se hizo el túnel SSH: La dirección del servicio para la máquina local es `localhost`, que se considera contexto seguro [56]). La única solución para este problema es obtener un certificado TLS válido y realizar la conexión a través de HTTPS. Let's Encrypt es una autoridad certificadora capaz de emitir certificados TLS válidos de forma gratuita. Proporcionan un script que llaman Certbot, para realizar la configuración en el servidor de forma automática. Los requisitos para el servidor son: Que tenga nombre en el DNS, que dicho nombre aparezca en la variable ServerName de la configuración del servidor y que tenga conexión a Internet para comunicarse con el servicio de Let's Encrypt (que verifica que la dirección que resuelve el nombre de dominio para el que se solicita el certificado corresponda a la máquina en cuestión). Subsanados ambos problemas, el servicio era accesible a través de `https://eyetracking.lpi.tel.uva.es` y la prueba de cámaras detectaba los dispositivos con éxito.

- La prueba de cámaras originalmente no es la que se ha mostrado, sino una adaptación del front-end de una aplicación alojada en `webcamtests.com`. Esta tenía un par de problemas que la inviabilizaban para el proyecto. El primero es que era demasiado lenta. Tardaba demasiado en realizar comprobaciones. Esto no sería un problema si fuera capaz de comprobar si una cámara ya era conocida y en caso afirmativo, recuperar su información desde la base de datos y no volver a extraerla

del dispositivo en cada ejecución. El segundo problema era que en caso de caída de `https://webcamtests.com` dejaba de funcionar en la web del servicio de adquisición, lo cuál es un claro indicador de que existe una dependencia incompatible con los términos y condiciones de la aplicación, que declaran que no se enviarán datos a terceros bajo ningún concepto. Probablemente hubiera sido posible subsanar estos inconvenientes depurando el script de `webcamtests.com` si no fuera porque estaba ofuscado (la obtención de parámetros del mismo se hacía por *scrapping* del documento HTML). No quedó de otra alternativa que prescindir de esta prueba de cámara y desarrollar una nueva desde cero. Esta utiliza igualmente la API Media Devices para acceder a las cámaras del equipo, permite seleccionar una en caso de contar con varias, extrae datos de la misma y los almacena en la base de datos. También informa al usuario de cuáles son estos datos y muestra en un cuadro de la interfaz lo que ve la cámara, para poder verificar su correcto funcionamiento.

- Al utilizar la aplicación accediendo a través de internet se observó que en ocasiones los registros de la base de datos llegaban completamente vacíos a pesar de que el cliente los enviaba correctamente. Esto se debía al que el código no estaba preparado para trabajar en un entorno en el que los datos no llegaran al servidor inmediatamente como ocurría en local. Cuando las condiciones de la red impiden que los datos lleguen antes de que el cliente pase a la siguiente iteración del experimento, se produce el error. Para subsanarlo, se modificó la subida de datos al servidor, para que en vez de ser síncrona al experimento, fuera asíncrona. Es decir: ahora según se adquieren los datos se almacenan en memoria durante la realización del experimento y cuando concluye, se suben al servidor. Mientras esto sucede, una pantalla de carga muestra un indicador porcentual del progreso al usuario. De esta forma se garantiza que los datos lleguen al servidor independientemente del estado de la red, si bien, influirá en la duración del proceso.

La motivación del Experimento 3 fue un caso de uso para el proyecto actual no contemplado por el sistema de adquisición. Las redes neuronales no tienen forma de comprobar por sí mismas la corrección de los datos de entrada. Reciban lo que reciban, siempre devuelven una predicción. Por supuesto, si a un modelo se le proporcionan datos para resolver un problema para el que no ha sido entrenado, la salida es potencialmente incorrecta y no puede ser tomada en cuenta. En el caso de los modelos entrenados con los datos de los experimentos 1 y 2, el problema a resolver es claramente inferir la posición de la pantalla a la que el usuario dirige la mirada cuando este orienta la cabeza al centro de la pantalla y por supuesto mira dentro de la misma. Sin embargo, en un entorno real, esto no tiene por qué ser así. Puede ser que el usuario desvíe su atención de la pantalla, en cuyo caso, cabe esperar que una arquitectura de *Eye-Tracking* completa deje de alimentar el modelo que hace las predicciones propiamente de *Eye-Tracking* para evitar realizar predicciones potencialmente incorrectas. La solución que se propone consiste en un tercer experimento completamente diferente a los anteriores que mediante instrucciones e ilustraciones requiera a los usuarios mirar a diferentes puntos de su alrededor, bien dirigiendo la mirada o tanto la mirada como la cabeza,

por medio de unos casos predefinidos seleccionados aleatoriamente, entendiendo como caso, una ilustración que indica a dónde debe mirar, y si la cabeza debe apuntar en la misma dirección o permanecer en el centro; acompañada de un texto que desarrolle la explicación.

Para poder utilizar la misma base de datos con la que ya se contaba, se planteó un sistema de coordenadas en el que se pueden describir todos los casos con los mismos campos que ya se utilizan para los experimentos 1 y 2. Ver tabla C.1 y figura 3.4

Campo	**Exp. 1 y 2**	**Exp. 3**
pos_x	Circunferencia objetivo	Orientación de la cabeza
pos_y	Circunferencia objetivo	Orientación de la cabeza
click_x	Posición del cursor del ratón	Orientación de la mirada
click_y	Posición del cursor del ratón	Orientación de la mirada

Tabla C.1: Descripción del contenido de los campos de la tabla `posicion_tiempo` de la base de datos en los Experimentos 1, 2 y 3

Para mejorar la usabilidad se han implementado animaciones que evidencian las actualizaciones en los botones o en los textos, y cuentas regresivas acompañadas de señales acústicas que indican al usuario cuándo se realizan las capturas.