E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Cuantificación y optimización de la robustez de la calibración de un sistema de seguimiento de mirada

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

José María Armendáriz Armenteros

Directores: Rafael Cabeza Laguna

Gonzalo Garde Lecumberri

Pamplona, 9 de junio de 2023

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Índice

1. Resumen

Este proyecto pretende presentar un nuevo algoritmo de estimación de la mirada, basado en una red neuronal entrenada con dos bases de datos: la base de datos sintética U2Eyes y una base de datos específica del proyecto.

A lo largo del trabajo, se recoge todo el procedimiento seguido, desde la adquisición de vídeos hasta el entrenamiento de la red, pasando por todo el procesamiento de datos correspondiente. Por último, se realizarán varios experimentos con la red neuronal, para conocer mejor su funcionamiento y determinar los parámetros que más pueden afectar a su rendimiento. Una vez se ha hecho esta valoración, estaremos en posición de elaborar unas conclusiones que permitan establecer un buen uso del algoritmo y las condiciones en que debe ser aplicado para obtener unos buenos resultados.

2. Motivaciones y objetivos

Los sistemas de seguimiento de la mirada son sistemas que permiten detectar la presencia de una persona y calcular el punto de una pantalla al que dicha persona está mirando. Aplicados al uso de un ordenador, estos sistemas permiten estimar el punto del monitor al que el usuario está mirando utilizando una cámara. Estos sistemas permiten, por tanto, un control total de un ordenador con la utilización exclusiva de los ojos, lo que permite su uso por parte de usuarios con dificultades motoras o del habla, tales como enfermos de ELA, síndrome de enclaustramiento o personas que han sufrido un derrame.

En los últimos tiempos, se han producido muchos avances en este campo. Los sistemas de seguimiento de la mirada se han ido haciendo cada vez más precisos y compactos. Sin embargo, el coste de implementar estos sistemas puede llegar a ser, en muchas ocasiones, prohibitivo, pudiendo variar de unos pocos cientos de euros –los sistemas más baratos– a decenas de miles de euros –los dispositivos de gama alta–.

Dada esta situación en el mercado de los sistemas de detección de la mirada, este Trabajo Fin de Grado pretende presentar un algoritmo de bajo coste y sencillo de implementar, que pueda operar con los vídeos que pueda extraer la *webcam* incorporada en muchos ordenadores. Por esto, el algoritmo no tendrá un rendimiento puntero, pero esperamos que la precisión sea suficiente para la mayoría de aplicaciones que el detector pueda tener.

3. Introducción a las redes neuronales

Como se ha mencionado anteriormente, el algoritmo de detección de la mirada ha sido implementado a través de una red neuronal. Para entender mejor el proceso de entrenamiento de la red y la adquisición de datos, parece adecuado hacer primero una introducción a las redes neuronales.

3.1. Estructura de una red neuronal

Una red neuronal es un clasificador extremadamente complejo que intenta imitar la manera de funcionar de la mente humana. De esta manera, una red neuronal es capaz de hacer clasificaciones imposibles para otros algoritmos, adaptándose para cada problema en concreto.

Todo esto es posible gracias a la estructura de las redes neuronales, formada por una serie de neuronas agrupadas en diferentes capas, tal y como se muestra en la figura. En primer lugar, está la capa de entrada, donde las neuronas obtienen la información del ambiente, de los datos de entrada. Tras ser procesados por esta capa, los datos pasan a las capas intermedias –o capas ocultas–, hasta alcanzar la capa de salida. Las neuronas de la capa de salida hacen el último procesamiento y ofrecen el resultado de la clasificación.

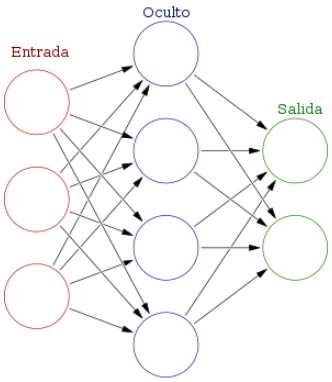

Figura 1 – Estructura de una red neuronal

En cada capa, el axón de una neurona alimenta a las dendritas de todas las neuronas de la capa siguiente. Esto añade más complejidad a la hora de calcular los parámetros de la red neuronal, pero permite hacer clasificaciones complejas, teniendo todos los resultados intermedios en cuenta.

Cada neurona tiene –a semejanza de las células nerviosas– unas dendritas, un núcleo y uno o varios axones. Las dendritas son las entradas a la neurona, son los datos que la neurona utilizará en sus cálculos. Estas dendritas pueden provenir del entorno o de otra neurona de la capa anterior.

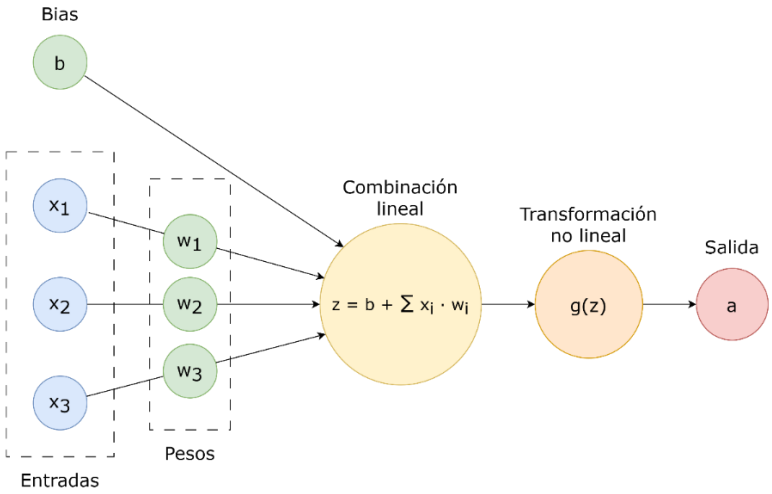

Figura 2 – Diagrama de bloques de una neurona

Una vez recibidos todos los datos de entrada, la neurona aplica una función lineal para obtener un primer parámetro de salida. Es decir, cada entrada es ponderada por un peso w_i con el que contribuirá a la salida de la neurona. Además, la función lineal se ve afectada por un cierto *bias* que la neurona introduce. El resultado de esta transformación es el parámetro z. En total, tenemos la expresión:

$$z = b + \sum_{i=1}^{M} x_i \cdot w_i$$

A continuación, se aplica una función no lineal que transforma el parámetro z en la salida de la neurona, denominada función de activación. De no existir la función de activación, la cascada de neuronas se convertiría en una cascada de transformaciones lineales, que se podrían resumir en una única función lineal, por lo que la red neuronal no tendría sentido. La definición de la función de activación depende del entrenamiento al que se someterá la red neuronal, por lo que conviene volver a este tema más adelante.

El último elemento de la neurona son los axones, las salidas de la neurona. Si es una neurona de la primera capa o de las capas intermedias, el axón se conectará directamente a las dendritas de las neuronas de la capa siguiente. En estos casos, la salida podrá tener un valor cualquiera, que las siguientes neuronas pueden ponderar por un peso.

Sin embargo, si se trata de una neurona de la capa de salida, pueden darse dos casos, dependiendo del número de clases de la clasificación. Si la red neuronal se ocupa de clasificar el dato de entrada en solamente dos clases, la capa de salida tendrá una única neurona de salida binaria. Por otra parte, si la red neuronal es capaz de clasificar un dato de entrada en una de varias clases a la salida, podemos tener tantas neuronas en la capa de salida como clases. En estos casos, suele utilizarse la función de activación *softmax*, de la que hablaremos en los siguientes apartados.

En resumen, una neurona es capaz de tomar unos datos de entrada, ponderarlos, y obtener una salida. Es decir, el funcionamiento de una neurona por sí misma es extremadamente sencillo y no sería capaz de realizar clasificaciones complejas. Sin embargo, la combinación de neuronas permite incrementar exponencialmente la complejidad de la clasificación de la red, por lo que podemos llegar a obtener resultados muy satisfactorios con arquitecturas relativamente sencillas.

3.2. Funciones de activación

Llegados a este punto, es conveniente retomar el tema de la función de activación de las neuronas. Como se ha dicho antes, la función de activación es una función no lineal que transforma el resultado z de la combinación lineal de las variables de entrada y el *bias* de la neurona para obtener el resultado que la neurona transmitirá por sus axones.

En un primer momento, podemos pensar en la función de activación más sencilla: el escalón. Una función escalón dará un resultado de 1 si el parámetro z es mayor que 0, y -1 –simplemente por convención– en caso contrario. Es la función de activación más evidente y la que se usa en las neuronas de la capa de salida si el resultado de la red neuronal debe ser binario.

Visto de otra manera, una neurona con función de activación escalón genera una frontera en el hiperplano

$$x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_n \cdot w_n + b = 0$$

Donde x_i es la entrada asociada a la dendrita i-ésima, y w_i es el peso correspondiente que le asigna la neurona. Los puntos que se sitúen por encima del hiperplano se clasificarán dentro de la clase A –con una salida de 1– y los que se sitúen por debajo, en la clase B –con una salida de -1–.

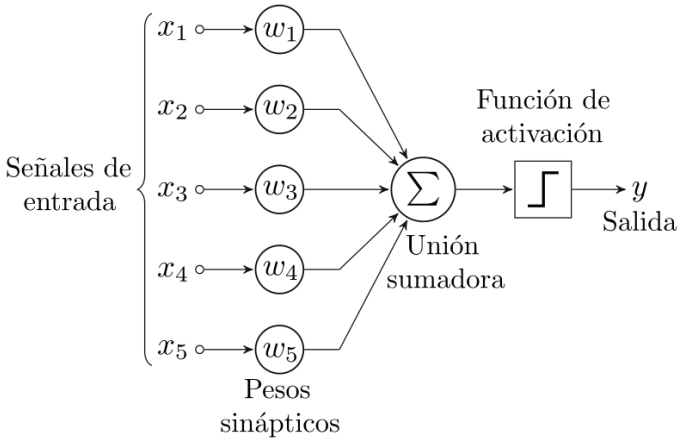

Figura 3 – Esquema de una neurona

En el caso sencillo de una única neurona con esta función de activación, el aprendizaje de los pesos sigue la siguiente ecuación:

$$w_{i,2} = w_{i,1} + (y_i - \hat{y}_i) \cdot x_i$$

Donde x_i es la entrada correspondiente a la dendrita i-ésima, $w_{i,1}$ es el peso que se le daba a dicha entrada en el paso anterior, y_i es la clase real del dato, $\hat{y}_i$ es la clase que la neurona ha predicho, y $w_{i,2}$ es el peso que se le asignará a la entrada en la siguiente iteración, una vez ha sido corregido.

Como se puede observar, este es un claro ejemplo de aprendizaje online, en el que utilizamos un único ejemplo de la base de datos de entrenamiento para corregir los pesos de la red neuronal. Es decir, los pesos varían con cada ejemplo que se procesa, hasta conseguir que todos estén correctamente clasificados. El proceso de entrenamiento es como el descrito en la figura.

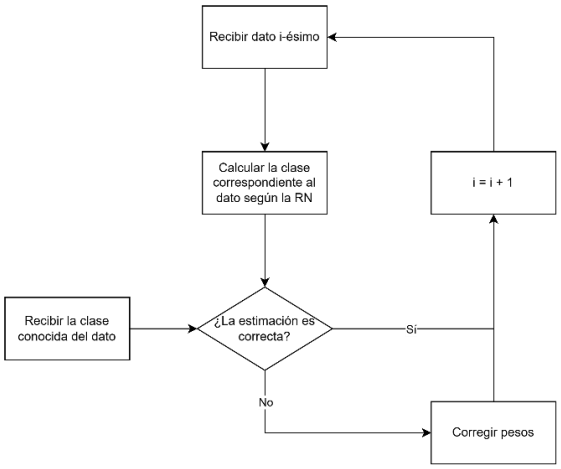

Figura 4 – Sistema de optimización de una única neurona

Sin embargo, para que la solución converja a ciertos valores de pesos, el problema de partida debe ser linealmente separable. Si no es así, nunca se logrará clasificar correctamente todos los ejemplos, y la solución se quedará oscilando entre los pesos necesarios para clasificar correctamente un par de datos incompatibles entre ellos.

Para evitar esta falta de convergencia, hay que cambiar ligeramente la filosofía a seguir para determinar los parámetros de una red neuronal. En primer lugar, la función de activación escalón no es cómoda a la hora de optimizar los pesos y *bias* de la red, porque un pequeño cambio en los parámetros puede causar un cambio muy grande a la salida. Además, como explicaremos más adelante, necesitamos poder computar la derivada de

la función de activación para modificar los parámetros, lo que es imposible en el escalón. Por ello, se suele utilizar otras funciones de activación más suaves, tales como la tangente hiperbólica, la *relu* o funciones sigmoides.

Cabe destacar la excepción de las neuronas de la capa de salida, que no sufren estos problemas. En el caso de redes neuronales que realicen clasificaciones binarias, la neurona de la capa de salida tendrá una función de activación escalón, ya que debe indicar si el dato de entrada pertenece, efectivamente, a una clase. Sin embargo, en el caso de clasificaciones multiclase, la estrategia es un poco diferente. En la capa de salida, tendremos tantas neuronas como clases existan en la clasificación, y cada neurona obtendrá la probabilidad de que el dato a la entrada pertenezca a su clase. Esto se consigue a través de la función de activación *softmax*, con la expresión $g(z) = \frac{e^z}{\sum_i e^{z_i}}$. La función transforma la salida exponencialmente, y, a continuación, la normaliza para obtener, entre todas las neuronas, un valor total de 1.

Volviendo a los problemas de la red neuronal propuesta, necesitamos evitar el problema de convergencia mencionado anteriormente. Para solucionarlo, la estrategia utilizada consiste en definir una función de error, que nos permita aproximar cómo de cerca está la salida de la red neuronal de los resultados que debería obtener. A lo largo del entrenamiento, se buscará minimizar esta función de error, buscando el conjunto de parámetros que permita obtener una diferencia mínima entre resultados observados y teóricos.

Por tanto, las neuronas utilizadas en redes neuronales tendrán una función de activación no lineal distinta del escalón –salvo en el caso de las neuronas de salida–, y se optimizarán según una función de error que mida lo diferentes que son los datos obtenidos y los esperados.

3.3. Redes convolucionales

Sin embargo, esta filosofía de funcionamiento es muy inefectiva a la hora de tratar imágenes. El número de variables de entrada a la red neuronal se dispara, junto al número de pesos y *bias* que es necesario calcular a lo largo de la red. Además, el algoritmo presenta problemas si la imagen se desplaza o se recorta, por lo que utilizar la estructura de neuronas es muy poco eficiente en estos casos.

En su lugar, es necesario un cambio de paradigma, consistente en el trabajo con redes convolucionales. Estas redes extraen una serie de características de la imagen, que después son clasificadas según una red neuronal como las estudiadas.

En primer lugar, es necesario explicar en qué consiste la operación convolución en dos dimensiones. Esta operación se define entre una imagen y una máscara de convolución, y consiste en la sustitución de los valores de píxel de la imagen por el resultado del sumatorio de los productos de cada píxel de la imagen por los de la máscara centrada en el píxel tratado.

En otras palabras, para calcular el resultado de la convolución en un píxel, hay que desplazar, en primer lugar, la máscara para que quede centrada en el píxel. A continuación, se hace el producto entre los valores de los píxeles de la imagen y de la máscara que quedan superpuestos, se calcula el sumatorio y se asigna al píxel de salida.

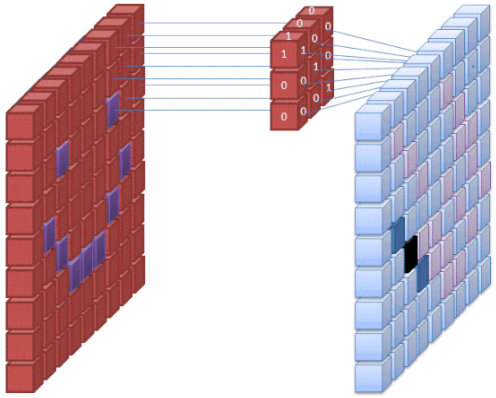

Figura 5 – Convolución en dos dimensiones

Una red convolucional dispondrá de una serie de máscaras de convolución, cuyos valores deben ser entrenados, al igual que el resto de parámetros de la red. Al aplicar una máscara a una imagen, se obtiene otra matriz de dos dimensiones, de un tamaño ligeramente menor –ya que la máscara no se puede centrar en los bordes, sus píxeles se "saldrían" de la imagen subyacente–. Por tanto, si una capa de una red convolucional consta de N máscaras de convolución, se obtendrán, a la salida, N imágenes reducidas, producto de la convolución de la imagen de entrada por cada una de estas máscaras. Por ejemplo, si a la entrada se tiene una imagen 32x32 y se usan 16 máscaras 3x3, se obtendrán 16 imágenes 30x30 a la salida de la primera capa convolucional, es decir, un volumen 30x30x16.

En serie, pueden realizarse más operaciones convolución, que pueden usar máscaras en tres dimensiones, por ejemplo, 3x3xN. De esta manera, se tienen en cuenta, a la vez, todos los resultados de la capa anterior, y no escala el número de imágenes dentro de la red.

Sin embargo, las capas convolucionales no reducen las dimensiones de la imagen de entrada de manera suficientemente rápida, y serían necesarias demasiadas capas para extraer características de una imagen con buena resolución. Por ello, se emplean capas *pooling*, que permiten reducir rápidamente las dimensiones de las entradas. Estas capas realizan una operación entre vecindarios de píxeles –típicamente, el máximo o la media–, de forma que la información de un vecindario de 2x2 o 3x3 píxeles se condensa en un solo valor.

Por último, es necesario convertir la información en forma matricial a un vector, que sirva como entrada a las últimas capas de la red neuronal, que serán similares a las estudiadas en los apartados anteriores. Para ello, se implementa una capa de aplanamiento, que ordene los datos de la imagen de entrada –ya sea por filas o por columnas– en un solo vector de salida. A partir de entonces, se utilizarán neuronas para procesar las características extraídas y obtener la clasificación final.

3.4. Entrenamiento de una red neuronal

El proceso de entrenamiento de una red neuronal es, obviamente, mucho más complejo que el mostrado anteriormente, que sólo sirve para una única neurona. En este caso, la red debe ir adaptando los pesos y *bias* de todas las neuronas de la red, adaptándose

al error medido. Por ello, el proceso de corrección debe ser un proceso de toda la red y no la corrección individual de los parámetros de una neurona.

Para minimizar el error, es necesario saber cómo varía el error en función de los pesos asignados y los *bias* de la red, es decir, la derivada del error respecto a cada parámetro de la red.

Hallar esta derivada –o, más generalmente, gradiente– es la función del algoritmo denominado propagación hacia atrás. Este algoritmo comienza por calcular el error según la función que se haya definido, de forma que, a partir de este punto, sólo se trate con este dato. A continuación, el algoritmo va recorriendo la red neuronal desde la salida hasta la primera capa, para calcular la dependencia del error respecto a cada uno de los parámetros de la red. Con este gradiente, la red estará en condiciones de aplicar el algoritmo de optimización, de forma que se vayan cambiando los parámetros de la red hasta llegar a una configuración que ofrezca resultados óptimos. Entre todos los algoritmos de optimización, el de descenso de gradiente es el más popular.

El algoritmo de descenso de gradiente tiene en cuenta el valor de la derivada del error con respecto a cada parámetro de la red neuronal para calcular cuánto debería variar dicho parámetro. El principio consiste en suponer que la función de error presenta un mínimo para cada parámetro, de forma que, al alcanzar el mínimo, los resultados de la red neuronal sean los óptimos. Para avanzar en dirección al mínimo, se varía el parámetro en la dirección descendente del gradiente, tal y como se muestra en la figura.

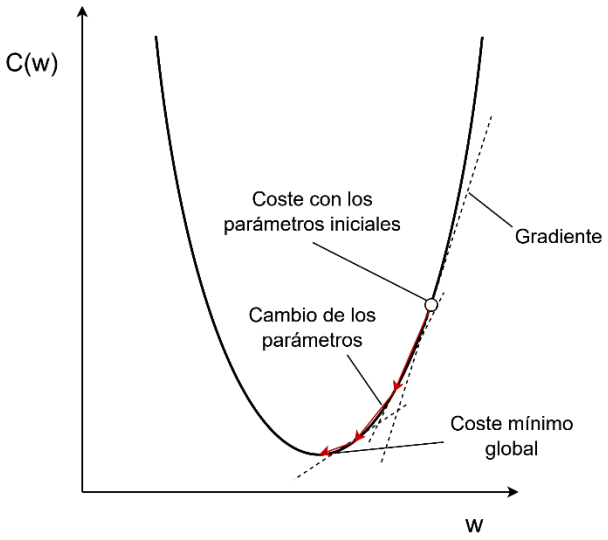

Figura 6 – Algoritmo de descenso de gradiente

La variación de los parámetros es proporcional al gradiente calculado en el algoritmo de propagación hacia atrás: si, en un punto, el gradiente tiene una gran magnitud, esperamos hacer una variación grande del parámetro, ya que estaremos relativamente lejos del mínimo. Sin embargo, conforme nos vamos acercando al mínimo, el gradiente se va haciendo cada vez más pequeño y los saltos van siendo cada vez más cortos, consiguiendo un ajuste más fino.

La constante de proporcionalidad entre el gradiente calculado y la variación del parámetro es el llamado *learning rate*. Si el *learning rate* es mayor, observaremos una mayor variación de los parámetros para el mismo gradiente, es decir, los saltos en los valores de los pesos y *bias* serán más grandes.

El valor óptimo del *learning rate* es un compromiso entre el tiempo de entrenamiento de la red neuronal y la precisión del ajuste de los parámetros. Si el *learning rate* es demasiado grande, los valores de los parámetros variarán demasiado de una iteración a otra, y puede haber problemas para detectar mínimos relativamente "estrechos". Además, el algoritmo puede tener problemas para converger a un punto o incluso divergir. Por otro lado, si el *learning rate* es demasiado pequeño, el algoritmo tardará mucho tiempo en converger a un punto y aprender los valores óptimos de los parámetros de la red, además de poder quedarse atascado en mínimos locales por ir explorando la función de error a demasiada poca escala.

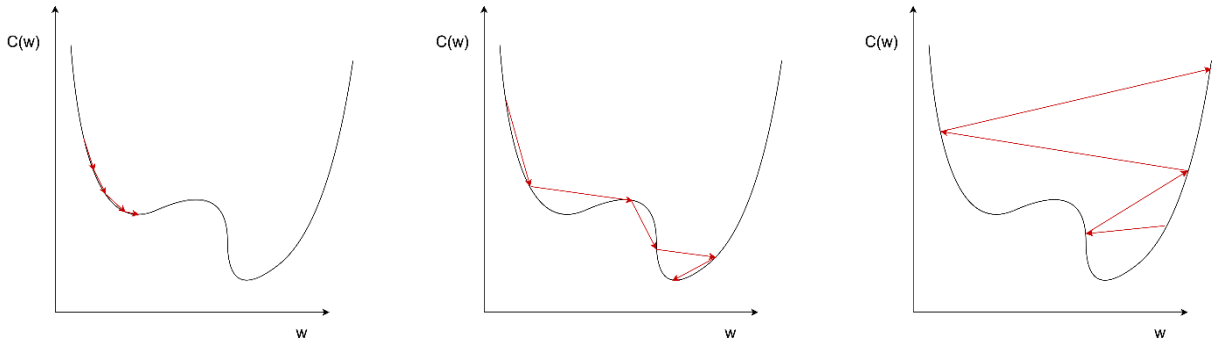

Figura 7 – Ejemplos de learning rate *demasiado pequeño (izquierda), óptimo (medio) y demasiado grande (derecha)*

Por último, también es de interés el parámetro del *batch size*. Cuando el conjunto de datos de entrenamiento es muy grande, es muy costoso computacionalmente utilizar todos los datos en cada iteración del proceso de optimización de parámetros. Por ello, se suele usar el método del descenso de gradiente estocástico, que consiste en dividir el conjunto de datos de entrenamiento en grupos más reducidos. En cada iteración del algoritmo de entrenamiento, se utilizará un grupo de datos diferente, de forma que se vaya aprendiendo de todos los datos, pero sin sobrecargar al ordenador. El tamaño de cada uno de estos grupos es el *batch size*. Con mayores *batch sizes*, la optimización en cada iteración será más precisa –al contar con mayor número de datos–, pero el entrenamiento será computacionalmente más costoso, por lo que hay que alcanzar un compromiso.

En conclusión, el proceso de entrenamiento de una red neuronal consiste en repetir muchas veces unos pequeños ajustes de los parámetros de la red, hasta conseguir que los valores produzcan un resultado óptimo. En cada iteración, se clasifican los datos de entrada según los parámetros de la red en ese instante y se calcula el error respecto a los datos teóricos. A continuación, se calcula la derivada de dicho error respecto a cada parámetro de la red, y se modifican los parámetros de acuerdo al gradiente. A este proceso seguiría la siguiente iteración, con una nueva clasificación de todos los datos, etc.

3.5. Conjuntos de entrenamiento y *test*

Para medir la calidad de una red neuronal, pueden utilizarse los mismos datos que se han empleado en el entrenamiento, ya que son datos ya etiquetados. Sin embargo, es una medición muy optimista del rendimiento de la red, ya que se están utilizando los mismos datos a los que el algoritmo se ha adaptado. Si la red se adaptara demasiado a los datos de entrenamiento, hasta el punto de descuidar los datos que se podrían dar en un escenario

real, y utilizáramos dichos datos de entrenamiento para medir el rendimiento de la red, podríamos obtener resultados que no se corresponden con el rendimiento real del algoritmo.

Por ello, la estrategia más común suele consistir en subdividir los datos etiquetados en dos conjuntos: el conjunto de entrenamiento y el conjunto de *test*. El conjunto de entrenamiento estará formado por los datos que se utilizarán para entrenar la red neuronal y asignar los diferentes pesos y *bias* de la arquitectura; mientras que el conjunto de *test* se utilizará para medir el rendimiento de la red con datos diferentes a los que la red ya ha procesado. La subdivisión de la base de datos disponible en conjuntos de entrenamiento y *test* depende de cada proyecto y del ámbito de aplicación de la red neuronal.

3.6. Ámbito de aplicación de una red neuronal

Como hemos revisado en este apartado, las redes neuronales constituyen un algoritmo que permite obtener clasificadores de manera automática. El hecho de que no necesiten teoría que respalde la clasificación que se intenta acometer provoca que las redes neuronales sean frecuentemente utilizadas en problemas en que la teoría subyacente no esté demasiado desarrollada, y en los que lo único que se puede aportar para intuir la solución son muchos ejemplos de datos clasificados correctamente.

Por otra parte, las redes neuronales también son capaces de resolver problemas no lineales, y de encontrar patrones absolutamente nada evidentes a otros clasificadores. Sin embargo, hay que tener en cuenta que las soluciones ofrecidas por las redes neuronales son únicamente aproximaciones a la solución perfecta. Por tanto, debemos asumir cierto grado de error en los resultados, sabiendo que la solución obtenida no es la óptima, ni tenemos certeza de lograr clasificar correctamente todos los datos de entrada.

Además, para construir una red neuronal, es necesario tener disponible una gran base de datos que sirva para entrenar y poner a prueba el algoritmo. Hay ocasiones en que la adquisición de datos está muy restringida, y no habrá información suficiente para entrenar una red neuronal con un rendimiento satisfactorio. En estos casos, cabe destacar el uso del *transfer learning* para lograr construir un buen clasificador.

El *transfer learning* consiste en la utilización de la información aprendida por otro modelo en el entrenamiento de nuestra red neuronal. Esta información puede implementarse de dos maneras distintas: con modelos preentrenados de extracción de características, y con modelos base que permitan un mejor inicio del algoritmo de optimización.

La extracción de características es una técnica por la cual se toma un algoritmo de una aplicación similar a la de la red neuronal propia y se utiliza para conseguir resultados intermedios, que permitan facilitar el trabajo de nuestra red neuronal. Por ejemplo, en la detección de la mirada, pueden utilizarse algoritmos que detecten la posición de la cara, ojos, etc. en una imagen, de forma que la red neuronal sólo necesite trabajar con las coordenadas de estos puntos, y no con la imagen entera. Por otra parte, los modelos preentrenados consisten en aproximaciones al modelo final, que se pueden realizar con bases de datos de otros proyectos o tomando directamente sus algoritmos. Estos modelos permiten no partir de cero, sino de un punto mucho más cercano al algoritmo óptimo, de

forma que el tiempo de procesamiento del algoritmo de optimización se acorte en gran medida.

Pese a las restricciones que pueden presentar, las redes neuronales son ampliamente utilizadas en multitud de ámbitos hoy en día: desde las redes sociales, hasta la atención sanitaria, Defensa, reconocimiento facial, el mercado en Bolsa… La cada vez mayor disponibilidad de información masiva sobre muchos de estos ámbitos provoca que entrenar una red neuronal sea extremadamente sencillo y dé resultados satisfactorios.

4. Adquisición de datos

Como se ha mencionado anteriormente, es necesaria una amplia base de datos para poder entrenar una red neuronal. En este proyecto, la red neuronal utilizará información disponible a través de la *webcam* de un ordenador para deducir el punto del monitor al que el usuario está mirando. Por tanto, el proceso de adquisición de los datos incluirá la obtención del vídeo en crudo de la cámara, el procesado para obtener los cuadros de interés, la obtención de características faciales a partir dichos cuadros y la depuración de datos erróneos. En todo este proceso, hay que tener en cuenta que los datos de los conjuntos de entrenamiento o *test* deben estar etiquetados, por lo que es necesario recoger información sobre el punto al que el usuario está mirando en cada instante.

4.1. Obtención del vídeo de las cámaras

El primer paso en la obtención de datos con los que alimentar la red neuronal es grabar a los usuarios mirando distintos puntos en la pantalla. Para grabar, se ha partido de la configuración mostrada en la figura.

Ilustración 1 – Sistema de grabación del proyecto

Un total de seis cámaras se ocupan de grabar al usuario desde puntos de vista diferentes. Es importante para nuestro sistema intentar hacer la detección desde diferentes puntos de vista, ya que el usuario debería poder situar la cámara donde prefiera. Si se construyese la red neuronal a partir de una sola cámara, la red no se podría adaptar a diferentes ángulos y distancias a la cara del usuario, por lo que perdería flexibilidad. Para evitar esto, se han dispuesto dos cámaras encima del monitor, una a cada lado, para disponer de varios ángulos laterales. Se ha procedido de igual manera con otras dos cámaras debajo de la pantalla. Por último, se ha colocado una cámara a cada lado del monitor, cada una a una altura diferente, con el fin de poder grabar al usuario desde ángulos y alturas diferentes.

Durante cada uno de los vídeos, las cámaras recogen la información mientras se le va mostrando al usuario un punto en la pantalla que cambia de posición un total de 16 veces. Los seis vídeos capturados simultáneamente por todas las cámaras forman una subsesión. Siempre se graban dos subsesiones en condiciones similares, de forma que una sirva para entrenar a la red neuronal y la segunda sirva para comprobar su rendimiento. Al grupo de dos subsesiones se le denomina sesión.

Se han grabado un total de 10 sesiones para cada usuario. Si tenemos en cuenta únicamente la subsesión dedicada al entrenamiento de la red neuronal, cada sesión es equivalente a una sesión de calibración de un sistema comercial. El resto de datos capturados en la sesión formarán parte del conjunto de *test*. La ventaja de grabar varias

sesiones con cada usuario consiste en la obtención de vídeos con diferentes condiciones de luz en la habitación, distancia a la pantalla, etc. Sin embargo, no siempre se han podido grabar diez sesiones independientes, ya sea por falta de tiempo o porque los usuarios no han podido acudir al laboratorio diez días diferentes. Por ello, a menudo, se han grabado varias sesiones secuencialmente el mismo día.

Para hacer las grabaciones, se ha hecho uso de un script elaborado por Gonzalo Garde, utilizado también en otros experimentos de detección facial multicámara.

En primer lugar, el script explora los directorios que hay en la carpeta base para componer una lista de usuarios de los que se han podido hacer grabaciones. De esta manera, se puede acceder a posteriori a los vídeos grabados de un usuario.

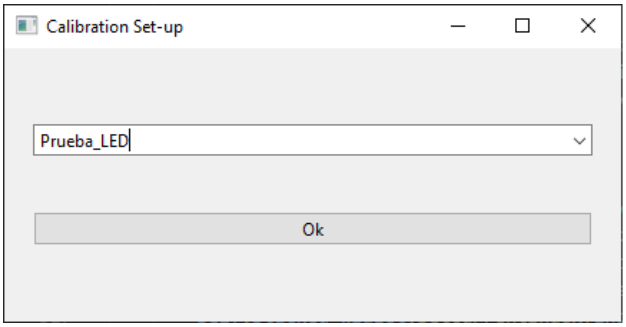

Ilustración 2 – Menú de selección de usuario

Tras seleccionar un usuario, se abre una nueva pantalla donde podemos elegir una de las sesiones que el usuario ya ha grabado, o grabar una nueva sesión. Seleccionando cualquiera de las opciones, accederemos a las subsesiones de la sesión escogida. En este menú, podremos elegir si queremos grabar una nueva subsesión o si queremos rehacer la sesión entera. La primera opción nos mostrará la secuencia de grabación, añadiendo los vídeos recogidos como una nueva subsesión dentro de la sesión escogida; mientras que la segunda borrará todas las subsesiones que había grabadas, para comenzar a grabar, acto seguido, la primera subsesión.

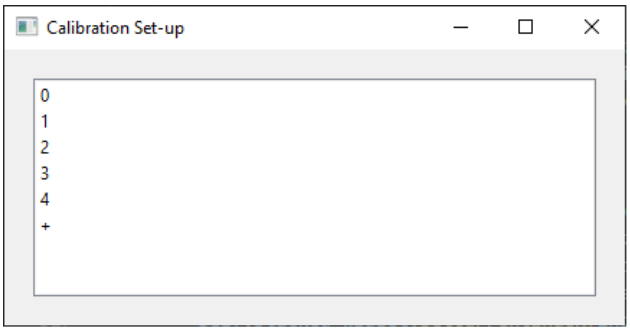

Ilustración 3 – Menú de selección de sesión

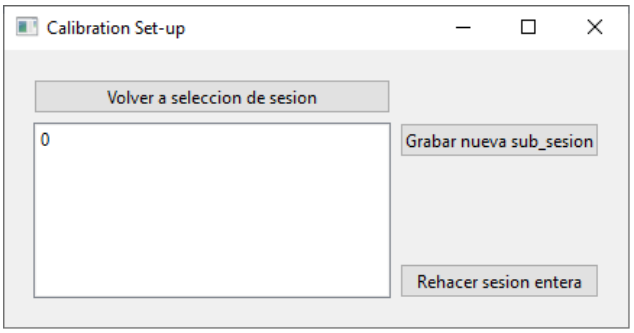

Ilustración 4 – Menú de selección de subsesión

Si se decide grabar una nueva subsesión, el programa muestra la secuencia de grabación. La secuencia de grabación tiene dos fases: una pequeña cuenta atrás y el seguimiento del punto. Durante el seguimiento, se mostrará un punto en la pantalla, que cambiará de posición cada dos segundos. Durante este periodo, el usuario debe fijar la mirada en el punto y, una vez lo ha hecho, pinchar en cualquier sitio con el ratón, para provocar el parpadeo de una bombilla LED situada a la derecha del usuario. Una vez se han sucedido los 16 puntos, la ventana se cierra y volvemos al explorador de subsesiones.

Al mostrar la secuencia de grabación, el script no simula en tiempo real el movimiento del punto, sino que reproduce un vídeo pregrabado en el que se muestra la cuenta atrás y el punto cambiando de posición. Cada sesión tiene asignada un vídeo

diferente, de forma que, a ojos del usuario, el movimiento del punto sea aleatorio. Sin embargo, podemos recuperar cuál ha sido la secuencia de posiciones del punto accediendo a este vídeo.

Por otra parte, tampoco se hace una detección, por parte del código, de los instantes en que el usuario hace un *click* de ratón. En su lugar, el botón del ratón enciende una bombilla LED, visible por las cámaras. Este parpadeo sirve para que, manualmente, se pueda identificar el instante en que el usuario está mirando al punto. Dado que el punto cambia de posición bruscamente, solo nos interesan los instantes precisos en que sabemos dónde está mirando el usuario, y no el movimiento de transición intermedio. Gracias a ello, podemos tratar solamente los instantes de interés de cada vídeo, usando el brillo del LED como referencia para la extracción de imágenes.

En conclusión, a través de este script, podemos obtener todos los vídeos correspondientes a las sesiones de calibración. El programa hace uso de secuencias determinadas para cada sesión y el LED actúa como guía para indicar cuándo el usuario está mirando a cada punto. De este modo, podremos obtener, a posteriori, el punto en la pantalla al que el usuario está mirando, es decir, el *ground truth* del experimento.

4.2. Extracción de cuadros

La manera más eficiente de acometer este problema sería con otro script Python, que detectara los momentos en los que la bombilla se iluminara y extrajera los cuadros correspondientes. Sin embargo, el script programado no ofreció buenos resultados, detectando tan solo 15 parpadeos del LED en muchos vídeos y 17 en otros tantos.

Debido a ello, se optó por extraer los cuadros de manera manual. Haciendo uso del editor de vídeos *DaVinci Resolve*, se marcó individualmente cada uno de los momentos en los que el LED se encendía, siguiendo el proceso descrito a continuación.

Al extraer los cuadros de los vídeos, nos apoyamos en el hecho de que, para la misma subsesión, los vídeos de todas las cámaras reflejan los mismos eventos. Sin embargo, los vídeos de salida del programa de captura no tienen la misma duración ni están sincronizados.

Por eso, el primer paso para extraer los cuadros de un vídeo consiste en alinear y cortar todos los vídeos correspondientes a una subsesión. Para ello, se cargan los seis vídeos en el editor y se busca, manualmente, el momento en el que el LED se enciende por primera y por última vez en cada vídeo. Muy a menudo, el brillo de la bombilla dura más de un cuadro, por lo que, en este caso, optamos siempre por marcar el primer cuadro en el que el LED aparece encendido. A continuación, se alinean todos los vídeos y se recortan para que tengan la misma duración.

El siguiente paso consiste en buscar, manualmente, cada uno de los 16 instantes en que el LED se ilumina. Dado que todos los vídeos están sincronizados, basta con observar uno solo y establecer marcadores de tiempo globales.

Hacer este marcado de manera manual tiene la gran ventaja de poder evitar los momentos que puedan dar un error en la extracción de características. Por ejemplo, hay usuarios que, por acto reflejo, parpadean siempre que pinchan con el ratón. Si obtenemos el cuadro en el que el LED está encendido, capturaremos al usuario con los ojos cerrados,

por lo que el dato de entrada a la red neuronal será inválido. En este caso, si se observa que ocurre un fenómeno similar, se puede desplazar el marcador hasta un instante en el que el usuario no esté parpadeando, pero en el que siga mirando al mismo punto. Se ha observado que la mayoría de los usuarios siguen con la mirada fija en el punto después de pinchar con el ratón, por lo que este proceso no es demasiado complicado.

Una vez se han obtenido todos los marcadores, se exportan en formato EDL, de cara a utilizarlos en otro script. De manera similar, se exporta cada uno de los vídeos sincronizados y recortados.

A continuación, se hace uso de otro script, cuya finalidad es obtener los cuadros marcados. El script obtiene como entrada un vídeo y el archivo EDL con los marcadores, y exporta una carpeta con las imágenes correspondientes a las marcas temporales en el vídeo indicado. De esta manera, se recorren todos los vídeos y se extraen los cuadros que serán utilizados para alimentar la red neuronal. Además, el programa exporta otro par de cuadros, uno, dos frames anterior y otro, dos frames posterior. Estos cuadros se emplearán en la corrección de errores y servirán para detectar posibles fallos en la extracción de características.

Hay que señalar que, al tratarse de un proceso manual, es susceptible a fallos humanos. Uno de los fallos más comunes ha consistido en la obtención de marcas desplazadas un par de segundos, provocada por un desorden en los pasos del proceso. Otra fuente de errores ha sido omitir, inadvertidamente, uno de los parpadeos del LED, obteniendo solo 15 marcas temporales en el vídeo. Cualquiera de los dos fallos ha producido que el script de extracción de imágenes no haya podido obtener todos los cuadros necesarios a la salida, lo que se ha corregido en la posterior detección de errores.

4.3. Extracción de características

Una vez se han obtenido los cuadros de interés, el siguiente paso consiste en extraer las características de dichas imágenes, es decir, extraer la posición de los ojos, la pupila, la orientación de la cabeza, etc. La posición de estos puntos de interés se recogerá en archivos JSON, que constituirán la entrada a la red neuronal.

Todos estos puntos de interés se extraen de las imágenes gracias a dos detectores: el propuesto por Kaipeng Zhang en su trabajo *Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks* [1] y el algoritmo de Viola-Jones [2].

El script usado en el proyecto integra estos dos detectores faciales en el mismo proceso: se van explorando los directorios buscando las imágenes extraídas por el script anterior y se va segmentando la cara del usuario en cada frame. En primer lugar, el script usa el detector Viola, ya que es mucho más rápido y permite ofrecer buenos resultados. En el caso de que el algoritmo falle y no detecte correctamente la cara del usuario, el script usa el detector MTCNN, que es más robusto.

Para construir un detector extremadamente rápido y suficientemente preciso, el detector de Viola-Jones –o, simplemente, detector Viola– hace uso de las características de Haar, de una representación “integral” de las imágenes, de un algoritmo AdaBoost y de un clasificador de cascada.

Las características de Haar son máscaras que se pueden aplicar a una imagen para capturar puntos de interés. Por ejemplo, si sabemos que el puente de la nariz se suele ver con más brillo que la zona de las mejillas, puede usarse la quinta característica de Haar para hallarlo. A continuación, habrá que hacer un sumatorio de los valores de brillo una vez se ha aplicado la máscara, por lo que resulta de interés usar una imagen "integral", en la que es más eficiente hacer el sumatorio. Una representación integral de una imagen consiste en sustituir el valor de brillo de cada píxel por la suma del valor de brillo de todos los píxeles que tengan coordenadas x e y iguales o inferiores. Además, para lograr una mayor velocidad, no se opera con todas las características extraídas de la imagen, sino que, gracias al algoritmo AdaBoost, se computan solamente las características más significativas. Por último, el detector utiliza un clasificador en cascada, compuesto por una serie de clasificadores denominados débiles. Si una entrada es rechazada por un clasificador débil, no pasa a los posteriores, sino que es desechada. Esto provoca que cada clasificador tenga que operar con un rango de entradas mucho menor, y solamente llegan hasta el final los datos realmente prometedores.

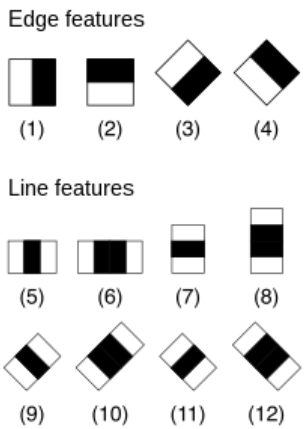

Figura 8 – Características de Haar

Por otro lado, el algoritmo MTCNN sigue una filosofía completamente distinta. Se utiliza una arquitectura compuesta por una cascada de redes neuronales, en la que la salida de cada una alimenta las siguientes. Resumiendo, la imagen es tratada por un total de tres redes neuronales convolucionales: las redes P, R y O. La función de la red P es la de conseguir posibles regiones que contengan los puntos de interés, la red R filtra estos candidatos para quedarse con los más prometedores, y la red O extrae los puntos con sus coordenadas.

Después de detectar la cara del usuario con cualquiera de estos detectores, el script extrae las características de la imagen, para exportar un archivo JSON con los puntos de interés, mediante el código escrito por Rubén Beruete.

4.4. Comprobación de errores

Una vez se han exportado todas las características de las imágenes de interés, conviene hacer un filtrado para detectar los posibles errores que el algoritmo haya podido cometer.

Para detectar estos errores, se han construido dos mecanismos de comprobación. El primero es automático y presenta una altísima sensibilidad ante errores, a cambio de una baja especificidad. De esta manera, el script es capaz de detectar la inmensa mayoría de los errores que se han podido producir durante la extracción de características. Sin embargo, dada la gran cantidad de falsos positivos generados por el detector, es necesaria una segunda etapa manual que filtre el resultado.

El programa automático comienza su ejecución pidiendo al usuario la carpeta donde están contenidos los archivos JSON y cuántos cuadros se han extraído por cada instante de interés. De esta manera, lee, simultáneamente, las características extraídas de las tres imágenes correspondientes a cada marcador temporal.

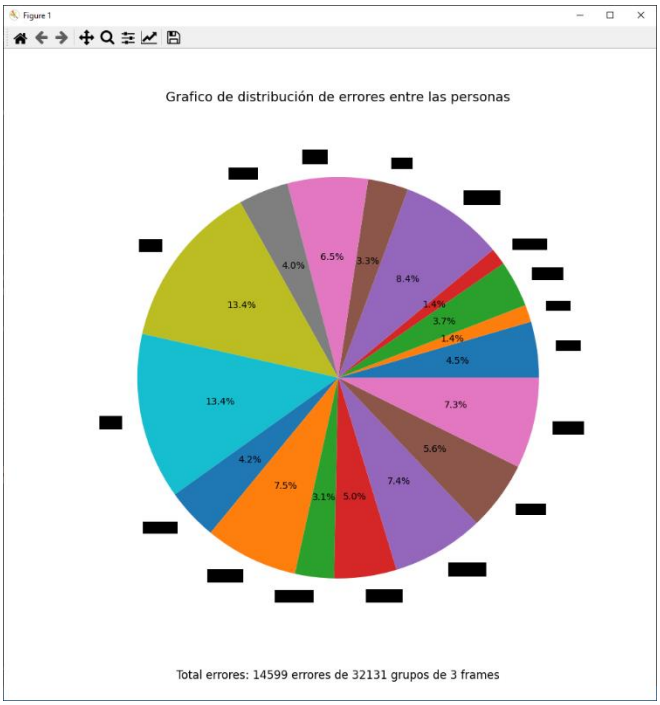

Figura 9 – Gráfico de sectores, indicando la porción de errores detectados en cada usuario respecto al total

Los primeros errores que el programa busca son los errores que se producen con discordancias entre los datos de los tres archivos de características. En primer lugar, puede haber un error debido a que uno de los archivos no exista, por errores en la extracción de cuadros o porque ni el detector Viola ni el MTCNN hayan podido detectar correctamente la cara del usuario.

Por otra parte, el programa también marca error en los casos en los que las marcas no coincidan entre los tres archivos. Dado que las imágenes corresponden a instantes muy cercanos, es de esperar que el usuario mantenga la misma postura, y las características detectadas estén en, prácticamente, las mismas posiciones. Sin embargo, si una cara no ha sido detectada correctamente –sino que los puntos se marcan en un espacio vacío de la imagen–, las coordenadas de las características no coincidirán con las de otras imágenes en que sí se haya detectado correctamente al usuario.

A continuación, el programa revisa los errores que se pueden producir en un único archivo JSON. El primer error consiste en que el archivo JSON esté vacío y no indique la posición de los puntos de interés. Acto seguido, se comprueba que las proporciones entre los puntos indicados sean las esperables: se compara el tamaño de cada ojo y la distancia entre ellos, la distancia entre las orejas, la distancia de cada oreja a la barbilla y la posición relativa de las cejas y de las pupilas. Una vez se han hecho todas estas medidas con cierto umbral de tolerancia, el script devuelve un informe con todos los errores detectados.

A continuación, se ejecuta un último script que presenta los errores para que sean legibles por un usuario, y tenerlos en un formato que permita operar con ellos. Además, el programa ofrece la posibilidad de hacer un segundo barrido de detección de errores para filtrar algunos falsos positivos. Al terminar, todos los datos de los errores se guardan en un archivo JSON, y se muestran unos gráficos con las estadísticas de errores detectados.

Como se ha mencionado anteriormente, este primer filtro tiene una muy baja especificidad, y la tasa de falsos positivos indicados por el programa, aún después de la segunda etapa, es demasiado alta. Por ello, en el proyecto se ha implementado otro programa de Rubén Beruete, que consiste en una detección manual de errores. Este segundo programa actúa como un último filtro que permita obtener solamente los verdaderos positivos, las instancias en las que exista, efectivamente, un error en la extracción de características de las imágenes.

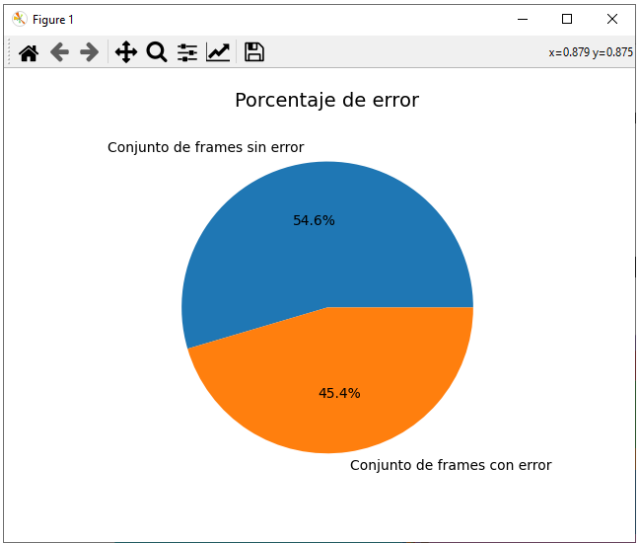

Figura 10 – Gráfico de sectores, indicando la porción de imágenes en las que se ha detectado un error

Para poder clasificar los errores, el programa muestra el cuadro, superpuesto con los puntos de interés, además de una breve descripción del error detectado. De esta manera, el usuario tendrá toda la información necesaria para etiquetar el error como un verdadero fallo en la detección de características o como un falso positivo. Una vez se ha clasificado correctamente una imagen, se puede avanzar hasta el siguiente error detectado, hasta recorrer todas las imágenes. Al final del proceso, se habrán filtrado todos los resultados, para obtener solamente las situaciones en las que las características estén situadas erróneamente. Estos datos no podrán ser utilizados para el entrenamiento o *test* de la red neuronal, y deberán ser desechados.

Ilustración 5 – Interfaz del programa de detección manual de errores

5. Entrenamiento de la red neuronal

Terminada la fase de adquisición de los datos, se puede proceder al entrenamiento de la red neuronal. En este proyecto, nos hemos querido centrar en el efecto que tiene, en el rendimiento de una red neuronal, aumentar el número de sesiones utilizadas para el entrenamiento de la red. Sin embargo, antes de acometer este problema, se ha decidido abordar otras cuestiones de menor calado para entender mejor el funcionamiento del algoritmo, tales como la importancia del *learning rate* y el *batch size*, la importancia de incluir varios usuarios en el entrenamiento y el efecto de variar la cámara utilizada en entrenamiento y en *test*. Para realizar estos experimentos, se parte de la configuración de la red neuronal que se describe a continuación.

5.1. Configuración del entrenamiento

Todo el proceso para entrenar una red neuronal comienza con un acondicionamiento de los datos. Por cada usuario, se genera un único archivo *feather* que contiene, de manera más accesible, toda la información recogida en los JSON. Por cada entrada, el *feather* indica las características faciales detectadas, el *ground truth*, el número de sesión y subsesión, la cámara utilizada, etc. De esta manera, los datos se recogen en un formato que el resto de código puede utilizar de manera eficiente y robusta.

Además, se operará con una segunda base de datos: U2Eyes [3]. Se trata de una base de datos sintética, creada a partir de un motor UnityEyes. Por defecto, la herramienta no implementa todos los parámetros necesarios para la construcción de un buen algoritmo de estimación de la mirada, y tampoco crea imágenes binoculares. Por ello, fueron necesarios varios cambios en la aplicación del motor para la creación de una base de datos sintética, que se reflejan en el proyecto de Sonia Porta *et al.*, *U2Eyes: a binocular dataset for eye tracking and gaze estimation*. Esta base de datos se utilizará para la construcción de un modelo base del que se partirá en cada experimento, de forma que el punto de partida del algoritmo es relativamente cercano al óptimo.

El modelo base cuenta con una serie de capas densas –es decir, cuyas neuronas están conectadas a todas las neuronas de la capa anterior– y capas de *dropout*, siguiendo la estructura de la Tabla 1.

El funcionamiento de las capas de *dropout* es eliminar, aleatoriamente, algunas de las salidas de la capa anterior, para evitar el sobreentrenamiento. Si en una capa se producen errores, las capas posteriores pueden entrenarse para corregir dichos errores, de tal forma que, para las entradas del conjunto de entrenamiento, no se visualizan los fallos de la primera capa. Sin embargo, a la hora de tratar con datos nuevos, las correcciones no serán tan precisas y se pueden producir fallos en la clasificación. La capa de *dropout* provoca que se tengan en cuenta, en cada iteración, salidas diferentes de cada capa, por lo que se evita que los errores de una capa sean corregidos por las posteriores: los errores serán visibles sólo en ciertas ocasiones. Esto fuerza a la red a optimizar los parámetros capa a capa y evita el sobreentrenamiento que se puede producir.

Tipo de capa	Longitud del vector de salida	Número de parámetros a entrenar
Densa	512	8192
Densa	128	65664
Dropout	128	0
Densa	128	16512
Dropout	128	0
Densa	32	4128
Dropout	32	0
Densa	32	1056
Dropout	32	0
Densa	16	528
Dropout	16	0
Densa	16	272
Dropout	16	0
Densa	2	34

Tabla 1 – Estructura del modelo base

Por último, en cuanto al modelo base, cabe destacar que no se ha utilizado una red convolucional, sino una red neuronal normal. Esto se debe a que la red no trata con imágenes, sino con las coordenadas de los puntos de interés extraídos anteriormente. Por ello, el modelo consta exclusivamente de capas densas y de *dropout*.

El procedimiento es equivalente en todos los experimentos. Se parte del modelo base entrenado con la base de datos U2Eyes, en lugar de emplear una inicialización nula o aleatoria de los pesos y *bias* de la red neuronal. A continuación, se hace una calibración del modelo con la información de la base de datos propia, afinando los parámetros de la red. Al hacer la calibración, se podrán variar los valores de *learning rate*, *batch size* y el optimizador utilizado para el entrenamiento, para alcanzar una configuración que optimice los resultados. Por último, se podrán modificar aspectos como qué usuarios o cámaras se utilizan en entrenamiento o *test*, o el número de sesiones de vídeo utilizadas, etc.

La forma de medir el rendimiento de nuestro sistema de estimación de la mirada se basa en la diferencia entre las coordenadas provistas por el *ground truth* y las resultantes de la aplicación de la red neuronal. Esta diferencia se da en píxeles, y en los ejes horizontal y vertical. Por tanto, es necesario cierto procesamiento: el cálculo de la diferencia total –y no descompuesta en sus ejes– y la conversión de píxeles a centímetros. Sin embargo, esta medida depende de la distancia a la que el usuario se coloca de la pantalla. Para evitar este efecto, se estima que el usuario se coloca a una distancia de 60 cm de la pantalla y se calcula el ángulo que subtiende la diferencia calculada anteriormente. La precisión de las estimaciones de la red neuronal se dará, por tanto, en función del ángulo que contiene la diferencia entre el punto calculado por la red y el *ground truth*.

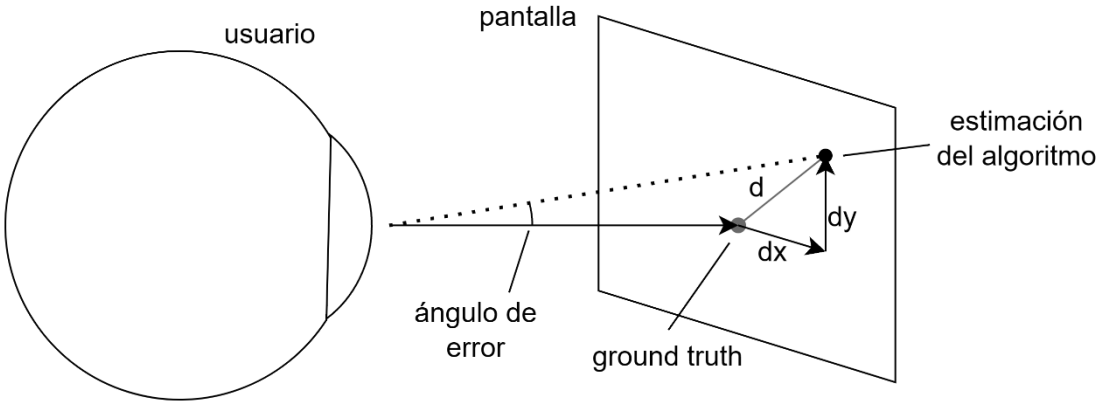

Figura 11 – Medición del ángulo de error

El algoritmo devuelve este resultado para cada ejemplo de test que se le haya pasado. Para obtener la salida, será necesario computar estadísticos de todos los ejemplos provistos, agrupando los datos en función de la cámara o el usuario utilizados en la calibración, el número de sesiones empleadas, etc.

5.2. Importancia de los parámetros de aprendizaje de la red neuronal

Ya que necesitamos especificar ciertos parámetros a la hora de entrenar una red neuronal, puede parecer que lo más sensible es determinar, en el primer experimento, la configuración necesaria para obtener unos resultados óptimos. Para ello, este experimento consistirá en variar el optimizador y los valores de *learning rate* y *batch size* utilizados en una red neuronal, hasta obtener los valores que optimizan el algoritmo.

Para evitar el efecto de otras variables, en este experimento se realiza la denominada calibración clásica. Esta calibración consiste en utilizar los datos provenientes de la misma cámara y del mismo usuario tanto en el entrenamiento de la red neuronal como en la evaluación de su rendimiento. De esta manera, se cancelan los efectos de variabilidad entre usuarios, puntos de vista de la cámara, condiciones de luz en las que se hicieron las grabaciones, etc., para apreciar mejor los efectos de la variación de parámetros estudiados.

El entrenamiento de la red neuronal se ha hecho para dos de los 19 usuarios disponibles, elegidos al azar, de forma que el proceso sea más rápido. Se ha establecido que, para todos los casos, se usen las 10 sesiones disponibles para entrenamiento, ya que todavía no se ha cuantificado el efecto de usar menos sesiones, y podríamos acabar optimizando algunos resultados demasiado concretos, y no la función de error global. Además, el proceso se ha realizado con los dos optimizadores más prometedores: los optimizadores Adam y RMSprop. El optimizador SGD es también popular en otros proyectos, pero ha dado resultados pobres en pruebas a menor escala, de forma que no se ha utilizado en este experimento.

En concreto, se ha decidido variar el *learning rate* entre 0.05 y 0.001, obteniendo 50 puntos equidistantes. Este entrenamiento se ha repetido también para 5 valores diferentes de *batch size*: 8, 16, 24, 32 y 64.

En las siguientes gráficas, se muestra la evolución del rendimiento de la red neuronal en función del *learning rate* utilizado en el entrenamiento, sin discriminar según el *batch size*. La primera gráfica muestra la evolución del rendimiento en una red que utiliza el optimizador Adam, mientras que la segunda muestra los datos de una red con el optimizador RMSprop.

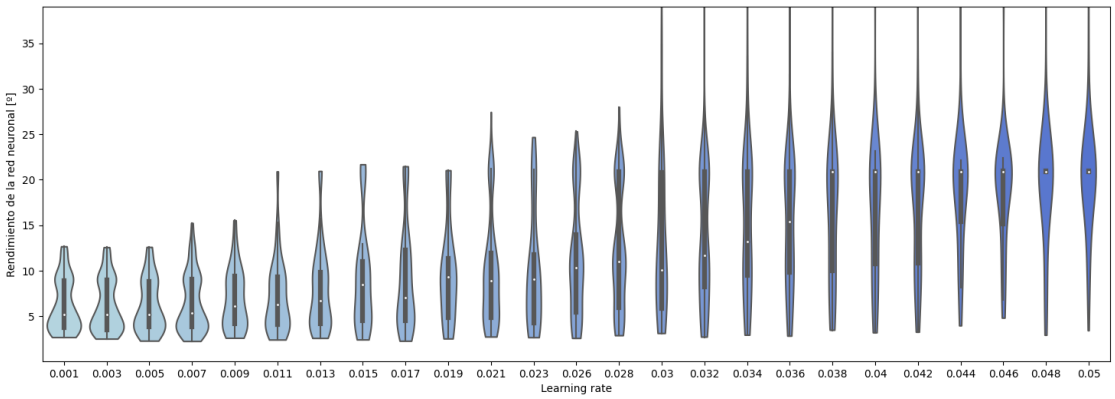

Figura 12 – Rendimiento de la red neuronal en función del learning rate *utilizado. El optimizador utilizado es Adam*

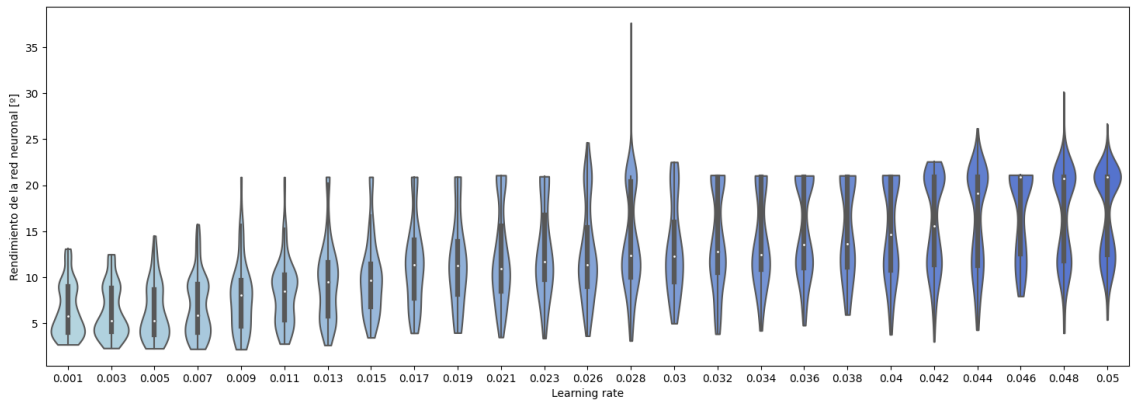

Figura 13 – Rendimiento de la red neuronal en función del learning rate *utilizado. El optimizador utilizado es RMSprop*

Si representamos ambos en la misma gráfica, obtenemos la figura:

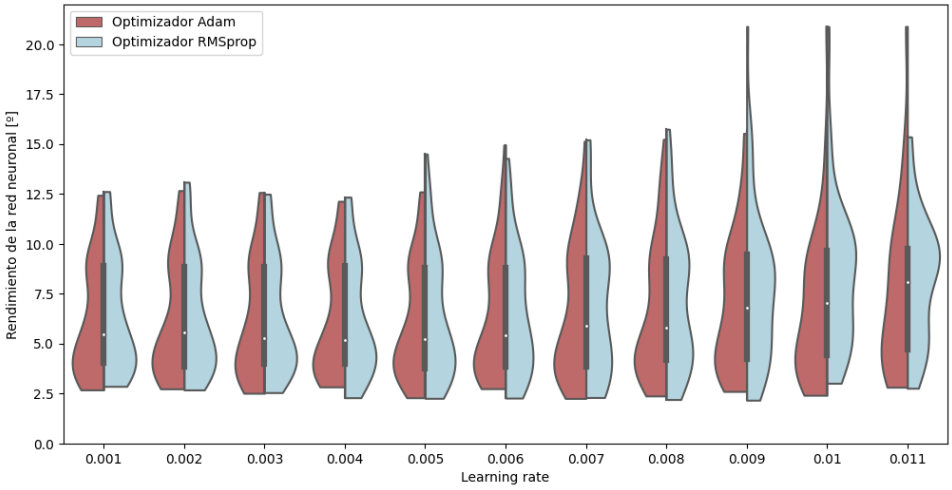

Figura 14 – Comparativa del rendimiento de ambas redes neuronales

Donde el rendimiento de la red con optimizador Adam se muestra en rojo, y la que utiliza el optimizador RMSprop, en azul. Se puede ver que ambas redes presentan unos resultados similares, aunque se puede intuir cómo, al aumentar ligeramente el *learning rate* por encima de 0.006, la red que utiliza RMSprop tiende a empeorar su rendimiento de manera más rápida que la red que usa el optimizador Adam. Sin embargo, en la zona de *learning rates* bajos –es decir, en la zona en que ambas redes presentan su mejor rendimiento–, apenas se pueden apreciar diferencias entre los optimizadores, por lo que se puede afirmar que la elección de uno u otro optimizador no afectará a los resultados obtenidos.

También puede apreciarse que resulta necesario optimizar el valor de *learning rate*. Si el *learning rate* es demasiado grande, se puede hacer una mala aproximación al mínimo de la función de error y, en el caso extremo, el algoritmo puede divergir y no optimizar los parámetros de la red neuronal. Por otra parte, si el valor escogido es demasiado pequeño, se producirá un acercamiento demasiado lento al mínimo global de la función y el algoritmo de optimización se puede quedar atrapado en mínimos locales. Además, el algoritmo de optimización será más lento y llevará más tiempo. Sin embargo, este efecto no puede llegar a apreciarse en las gráficas.

A continuación, hacemos detalle sobre las zonas de *learning rates* más bajos. En la siguiente gráfica, se usa el optimizador Adam, pero ambos son indistinguibles en estos valores del eje de abscisas. Si distinguimos el rendimiento según el *batch size* empleado en la evaluación, obtenemos la siguiente gráfica:

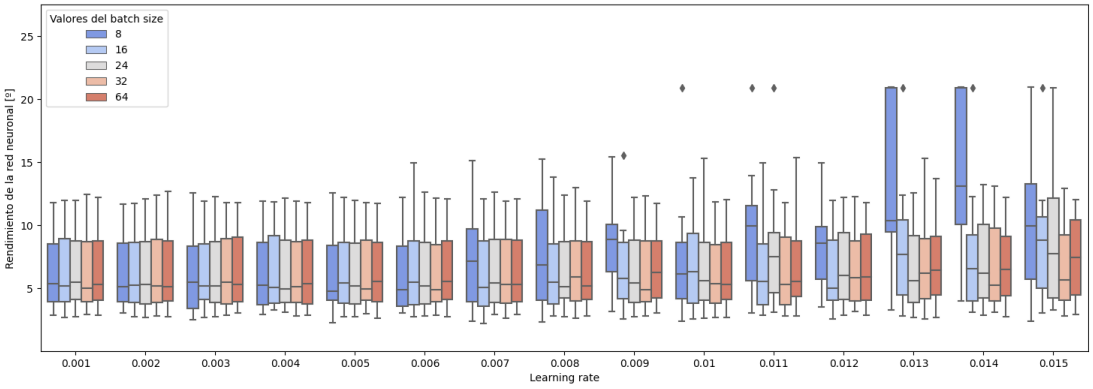

Figura 15 – Rendimiento de la red neuronal en función del learning rate*, discriminando según el* batch size

Como se puede apreciar, las redes entrenadas con diferentes *batch sizes* presentan un rendimiento extremadamente similar, salvo en el caso de usar un *batch size* de 8, que puede resultar demasiado pequeño para este entrenamiento. Un *batch size* demasiado pequeño implica que se toman demasiados pocos datos en cada iteración del algoritmo de optimización. Esto puede causar que la muestra no sea del todo representativa del conjunto de entrenamiento, por lo que la calibración de parámetros puede no ser adecuada. Al hacer la siguiente iteración, pueden tomarse datos muy diferentes y producirse oscilaciones muy grandes de los parámetros de la red, mientras intentan adaptarse a grupos de datos demasiado concretos. En la figura, se observa que, cuando el *learning rate* aumenta un poco, los resultados del algoritmo que usa un batch size pequeño se empobrecen mucho, ya que no es capaz de llegar, con suficiente precisión, al mínimo de la función de error. Al igual que en el caso anterior, podría elegirse un batch size demasiado grande, que tuviese demasiados datos en cuenta para cada iteración del algoritmo de optimización, saturando la memoria del ordenador; pero no se ha llegado a tal extremo y el efecto no se refleja en la gráfica.

A la vista de la figura, se puede establecer un valor óptimo para el *learning rate* de 0.005, mientras que el mejor *batch size* es de 16, utilizando cierto margen de seguridad en ambos valores. Si se produce una disminución del *learning rate* o un aumento del *batch size*, las mejoras de rendimiento de la red no son significativas, y no justifican el aumento en carga computacional que suponen. Por tanto, éstos serán los valores utilizados a la hora de calcular, con precisión, el rendimiento de la red neuronal en los siguientes experimentos.

5.3. Importancia del usuario utilizado en la calibración

En este segundo experimento, se ha intentado determinar el efecto que tiene, en el rendimiento de la red neuronal, la diferencia de usuarios entre los utilizados para el entrenamiento y los utilizados para la evaluación de la red neuronal.

Para ello, se han establecido unos valores arbitrarios de *learning rate*, *batch size* y optimizador utilizado. Al evaluar el rendimiento de la red, se han utilizado todas las entradas correspondientes a un solo usuario, mientras que las correspondientes a los demás se han utilizado en el entrenamiento. Por tanto, se realizan tantas calibraciones como usuarios hay en la lista. Cabe destacar que la cámara es la misma, tanto en entrenamiento como en *test*, para no introducir efectos dependientes de otros factores.

En primera instancia, se realizó un entrenamiento para 6 usuarios aleatorios de la base de datos, con un valor de *learning rate* de 0.01 y de *batch size* de 32. En todos los casos de este experimento, se ha usado el optimizador Adam. En la siguiente tabla, se recoge el rendimiento de la red neuronal cuando cada uno de los usuarios era utilizado en *test*. El rendimiento se ha calculado como la media del ángulo de diferencia entre todas las imágenes correspondientes al usuario.

Usuario	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6
Rendimiento de la red [º]	14.468	10.700	10.675	9.871	15.276	10.352

Tabla 2 – Rendimiento de la red neuronal para cada usuario

Como se podía suponer, el rendimiento es mucho peor que el obtenido cuando el usuario del entrenamiento es el mismo usuario que en el *test*. Dependiendo del usuario, se puede llegar a construir algoritmos con cierto grado de éxito, pero no se llega a comparar con los resultados obtenidos en el experimento anterior.

A continuación, se varían los valores de *learning rate* y *batch size* para dar lugar a otras dos configuraciones diferentes. En la configuración 2, el *learning rate* usado es de 0.0075, mientras que el *batch size* es de 8. En la configuración 3, se utiliza un *learning rate* de 0.005 y un *batch size* de 16. Por último, la configuración 1 es la empleada en el caso anterior. Los resultados se recogen en la siguiente tabla:

Usuario	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6
Rendimiento en la configuración 1 [º]	14.468	10.700	10.675	9.871	15.276	10.352
Rendimiento en la configuración 2 [º]	14.215	9.676	11.712	9.900	15.002	10.599
Rendimiento en la configuración 3 [º]	14.743	8.196	10.858	9.709	15.195	10.667

Tabla 3 – Rendimiento de la red neuronal para cada usuario, para tres configuraciones distintas

Cabe destacar la gran variación que se produce en el rendimiento para el usuario 2 si se cambian los parámetros de calibración: el *learning rate* y el *batch size*, llegando hasta dos grados y medio. Sin embargo, esta variación no se produce para el resto de usuarios. Esto puede deberse a que, para el usuario 2, la curva de rendimiento-*learning rate* está más desplazada hacia los valores bajos, mientras que, para los demás, todos los valores de *learning rate* están en la zona plana de la curva, donde los cambios entre configuraciones no producen variaciones en los resultados obtenidos.

5.4. Importancia de la cámara utilizada en la calibración

El tercer experimento es similar al anterior, con la salvedad de que la variación se introduce sobre la cámara utilizada, y no sobre el usuario. De forma similar, se establecen ciertos parámetros arbitrarios y se divide la base de datos en los conjuntos de entrenamiento y *test*.

En este caso, se utilizan los datos del mismo usuario tanto en el entrenamiento como en el *test*, y se discrimina según la cámara utilizada. Los datos provenientes a una cámara son utilizados para el *test* de la red, mientras que los correspondientes al resto de cámaras se utilizan para calibrar la red neuronal.

En un primer momento, se utilizó un *learning rate* de 0.01 y un *batch size* de 32 para la calibración del modelo base. Los resultados obtenidos se recogen, realizando la media del ángulo de error, en la siguiente tabla:

Cámara	Cámara 1	Cámara 2	Cámara 4	Cámara 5	Cámara 7	Cámara 8
Rendimiento de la red [º]	17.767	12.030	12.615	13.638	12.521	14.834

Tabla 4 – Rendimiento de la red neuronal para cada cámara

Observando la tabla, la red que mejores resultados ha dado es la que hace la evaluación con la cámara 2, situada en la parte superior derecha del monitor. La cámara que ve al usuario desde un punto de vista más parecido es la cámara 1, situada en el lado izquierdo del monitor. Sin embargo, el usuario se ve condicionado, por la presencia del segundo monitor a la izquierda del principal, a adoptar una postura más orientada hacia el lado izquierdo. Esto, añadido al ligeramente diferente pitch que presentan las dos cámaras, provoca que la cámara 2 capte la cara del usuario de manera más frontal, mientras que la cámara 1 se sitúa con un mayor ángulo. Esta diferencia de puntos de vista puede provocar la disparidad de rendimiento entre ambas redes.

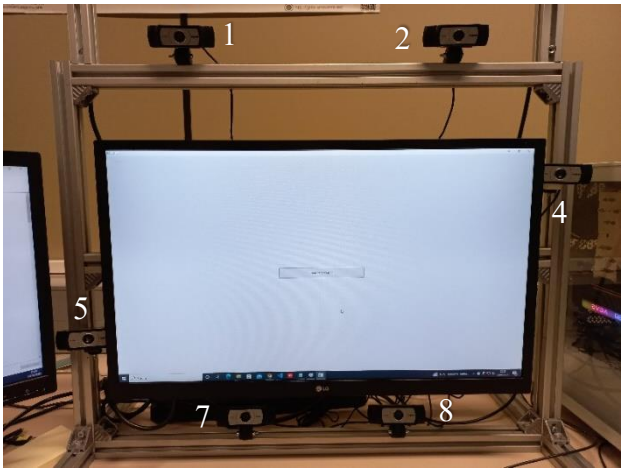

Ilustración 6 – Entorno de grabación utilizado. Cada cámara se muestra con su numeración correspondiente

Cabría esperar que el rendimiento de la red para las cámaras situadas encima y debajo del monitor fuera bueno, ya que estas cámaras tienen otra que obtiene imágenes aproximadamente simétricas del usuario. Sin embargo, se puede observar cómo esto no da resultado, llegando al punto de que la pareja de cámaras superiores ofrece la mayor disparidad de resultados de todo el conjunto.

La diferencia de rendimiento entre las cámaras puede deberse a que, si extraemos los vídeos de una de estas cámaras, sólo un quinto de los datos de entrenamiento de la red pertenece a datos grabados desde una perspectiva similar. Gracias a ello, la red neuronal

da más peso a los datos procedentes de cámaras con ángulos muy diferentes que a los datos que realmente provocarán un impacto positivo en el desempeño de la red.

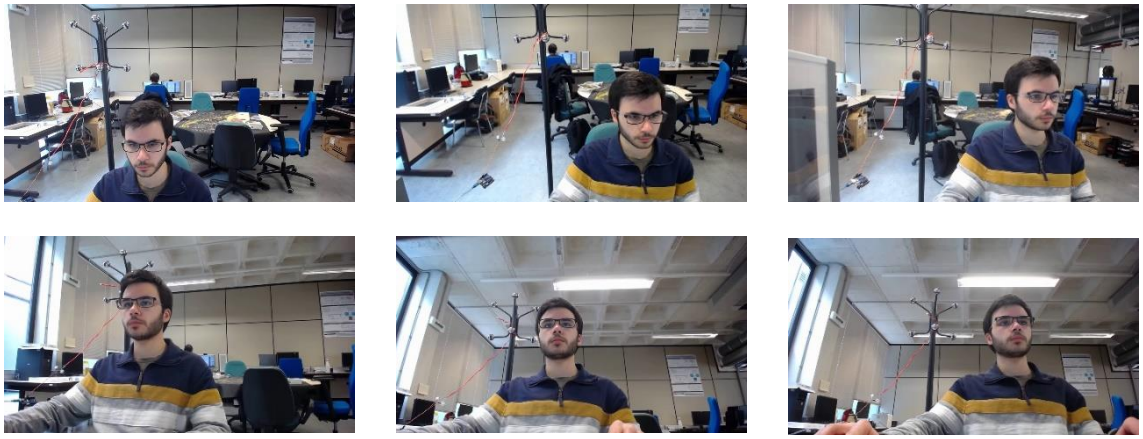

Ilustración 7 – Imágenes extraídas de las seis cámaras. En orden y por filas, las cámaras correspondientes a las imágenes son: 1, 2, 4, 5, 7 y 8.

Este efecto es más leve en otras cámaras, debido a que el enfoque de la cara del usuario es relativamente frontal, mientras que, en el caso de las cámaras 1 y 8, se presenta un ángulo muy pronunciado respecto al eje visual del usuario. Por tanto, la imagen de *test* es menos similar a las utilizadas en entrenamiento que en el caso del resto de cámaras, provocando unos peores resultados.

También podemos analizar, para este experimento, la dependencia del rendimiento de los parámetros de calibración. A continuación, se muestran los resultados para dos configuraciones diferentes:

Usuario	Cámara 1	Cámara 2	Cámara 4	Cámara 5	Cámara 7	Cámara 8
Rendimiento en la configuración 1 [º]	17.767	12.030	12.615	13.638	12.521	14.834
Rendimiento en la configuración 2 [º]	15.956	10.989	12.217	13.598	13.254	13.812

Tabla 5 – Rendimiento de la red neuronal para cada cámara, para dos configuraciones diferentes

La primera configuración es la empleada anteriormente, mientras que la segunda configuración cuenta con un *learning rate* de 0.005 y un *batch size* de 16. Las dos configuraciones utilizan el optimizador Adam. Al igual que como sucedía en el segundo experimento, existen configuraciones que favorecen el desempeño de determinadas cámaras antes que otras. Sin embargo, también en este experimento, todos los ensayos comparten algunos rasgos, como el mejor rendimiento de la cámara 2.

Por último, podemos comparar el rendimiento del algoritmo en estos dos últimos experimentos, cuando ambos se realizan con la misma configuración de los parámetros de calibración. En azul se representan los resultados del experimento 2, y en rojo, los del experimento 3. La comparativa entre ambos se ha hecho para dos configuraciones diferentes de *learning rate* y *batch size*.

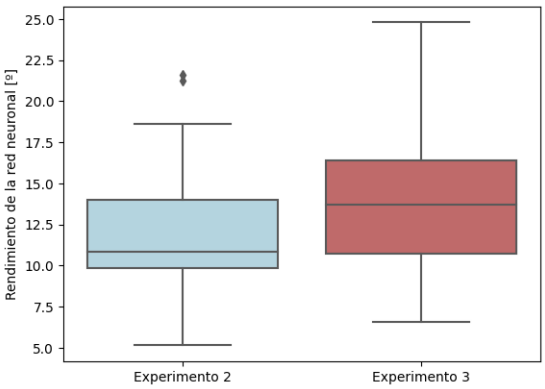

Figura 17 – Rendimiento de la red neuronal para los dos experimentos, con un learning rate *de 0.01 y un* batch size *de 32*

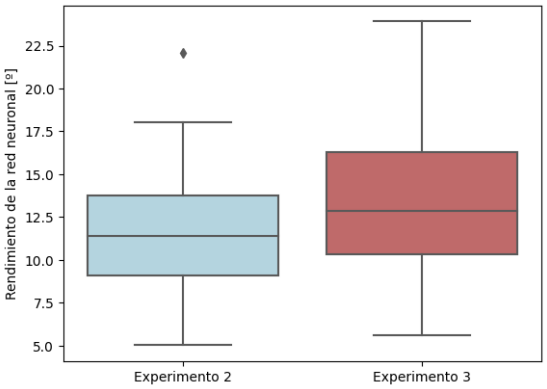

Figura 16 – Rendimiento de la red neuronal para los dos experimentos, con un learning rate *de 0.005 y un* batch size *de 16*

Tal y como se puede observar, el rendimiento de la red es mayor si es el usuario el que no coincide en entrenamiento y *test*. Esta circunstancia se repite tras probar varias configuraciones de *learning rate* y *batch size*, por lo que podemos afirmar que la cámara es más determinante que el usuario en el rendimiento de una red neuronal. Esto tiene sentido, ya que la variación entre usuarios –inclinación de la cabeza, distancia a la pantalla, apertura de los ojos, etc.– es mucho menor que la que presenta un cambio de cámara. El algoritmo está forzado a interpolar la información de que dispone para estimar el punto al que el usuario está mirando, cuando no se le ha entrenado con imágenes tomadas desde el mismo punto de vista que el que se le presenta en el *test*, por lo que la precisión disminuye.

5.5. Experimento principal

Una vez hemos recopilado todos estos datos, tenemos las herramientas necesarias para poder determinar el efecto que tiene el número de sesiones utilizadas como entrenamiento en el rendimiento de la red neuronal.

Para poder conocer este efecto, partimos de una configuración similar a la del primer experimento. Se utilizarán los datos correspondientes al mismo usuario tanto en el entrenamiento de la red neuronal como en el *test*, ya que es el caso de los sistemas comerciales, donde las sesiones de calibración son grabadas por el usuario del sistema. De forma similar, la cámara utilizada también será la misma en las dos fases. De esta manera, evitamos los efectos de cambio de usuario y de punto de vista que se reflejan en los experimentos anteriores, y que suponen una disminución del rendimiento del algoritmo que no se corresponde con una aplicación real de la red neuronal.

Además, el entrenamiento se ha hecho con los parámetros que permiten optimizar el rendimiento de la red neuronal, tal y como se han calculado en el primer experimento. De esta forma, se utiliza un *learning rate* de 0,005, un *batch size* de 8 y el optimizador RMSprop.

En cada iteración, se ha variado el número de sesiones utilizadas en el entrenamiento, desde una sola hasta 10. Por tanto, se realiza una calibración para cada persona, cada cámara y cada número de sesiones utilizadas en entrenamiento.

En las siguientes gráficas, se muestran los resultados del experimento del proyecto, para un *learning rate* de 0.05 y un *batch size* de 16, con el optimizador RMSprop. El eje de ordenadas representa el rendimiento del algoritmo. Para calcular este rendimiento, se ha hecho la media del ángulo de diferencia en cada elemento de *test* del mismo usuario, incluyendo todas las cámaras.

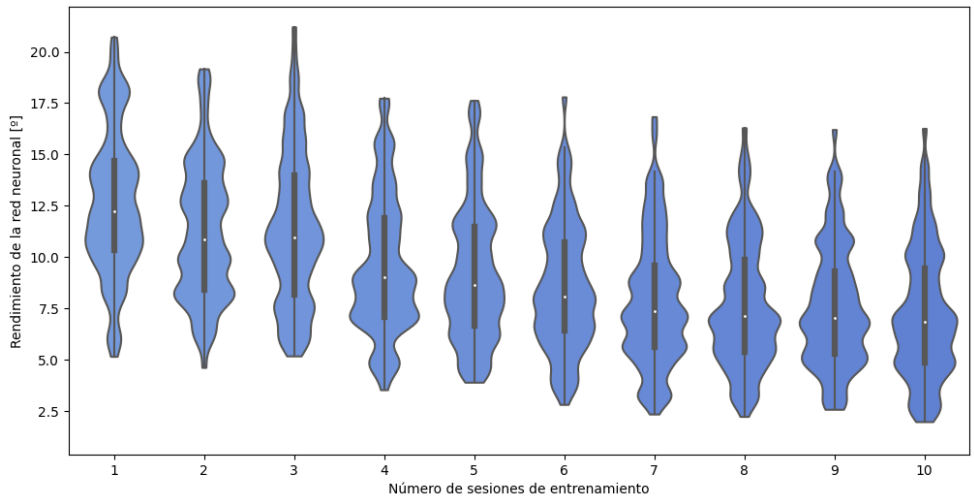

Figura 18 – Rendimiento de la red neuronal en función del número de sesiones de entrenamiento

En primer lugar, se puede apreciar que el rendimiento del algoritmo, de media, no es demasiado bueno. Esto puede deberse a una combinación de factores, como la baja calidad del sistema de grabación –aunque cada cámara sea de buena resolución, captan el ojo del usuario con tan sólo varias decenas de píxeles–, la determinación manual de los

marcadores en los vídeos o la clasificación manual de fotogramas erróneos –ya que cualquier proceso manual está sujeto a fallos humanos–. Todas estas circunstancias pueden contribuir a que el rendimiento del algoritmo se aleje del esperado en un sistema de seguimiento de la mirada de gama media-alta.

En la figura, también se puede apreciar que el rendimiento de la red neuronal es mejor si se utilizan más sesiones para entrenar el algoritmo, lo que concuerda con los resultados esperados. Si se alimenta la red neuronal con más datos, podrá caracterizar mejor cómo calcular el punto al que el usuario esté mirando y obtener mejores resultados.

También se puede apreciar que el rendimiento de la red es extremadamente dependiente del usuario del que se trate, ya que existe una dispersión entre los resultados de unos diez grados. Una posible explicación a esta diferencia podría consistir en el uso de gafas por algunos de los usuarios de la base de datos. Las gafas podrían distorsionar la zona de los ojos, de forma que las características extraídas no reflejan realmente el punto al que está mirando el usuario. Si se segmenta la población de usuarios, mostrando en azul a los que usan gafas y en rojo a los que no, obtenemos la siguiente gráfica:

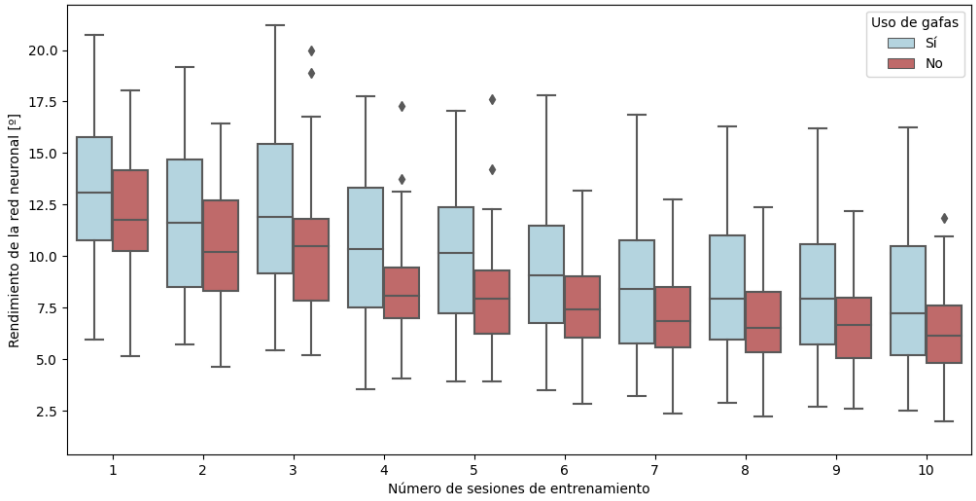

Figura 19 – Rendimiento de la red neuronal para la población de usuarios con y sin gafas

Como se puede observar, el rendimiento del algoritmo es sensiblemente peor en el caso de los usuarios que utilizan gafas. Además, existe cierta correlación entre la graduación de las gafas y el rendimiento de la red, ya que los dos usuarios que presentan peores resultados son de los usuarios cuyas gafas tienen mayor graduación. Esta dependencia puede deberse a que las gafas con mayor graduación distorsionan más los haces de luz que pasan a su través, por lo que la estimación del punto al que están mirando es más difícil.

También podemos discriminar los resultados según la cámara utilizada en entrenamiento y *test*:

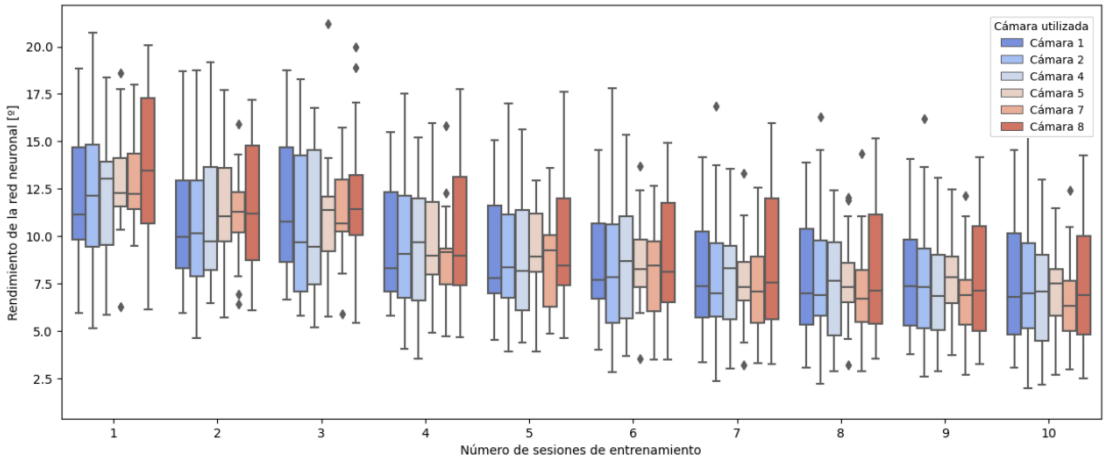

Figura 20 – Rendimiento de la red neuronal para cada cámara

En este caso, parece que la cámara 8 presenta unos resultados ligeramente más pobres, aunque la diferencia entre las medidas es demasiado insignificante como para llegar a alguna conclusión.

Parece que la tendencia a la baja puede llegar a cierto límite, provocando que el rendimiento se aproxime a una asíntota horizontal, si se aumenta indefinidamente el número de sesiones. Para intentar extender el rango del que disponemos, podemos utilizar algunas sesiones de *test* como sesiones de entrenamiento, aumentando así el número de sesiones con el que la red se ha calibrado. Estas sesiones que se añaden al conjunto de entrenamiento no podrán formar parte del conjunto de *test*, debido a la importancia que tiene separar ambos conjuntos de datos.

Si representamos el rendimiento de la red en función del número de sesiones, ampliando el eje hasta las 15 sesiones utilizadas en el entrenamiento, obtenemos la siguiente figura:

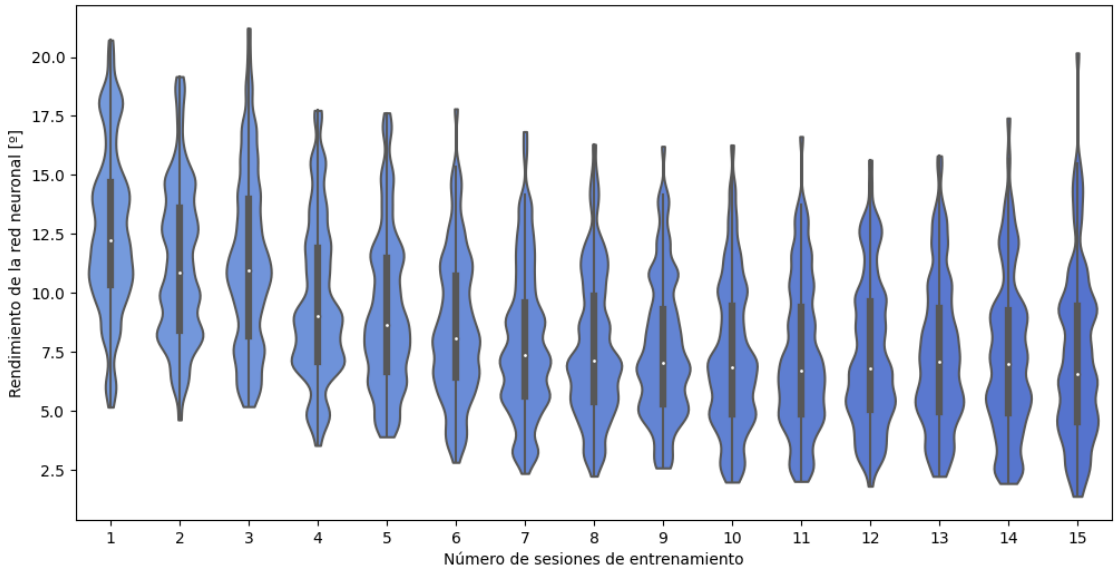

Figura 21 – Rendimiento de la red neuronal en función del número de sesiones de entrenamiento

Como se puede observar, el aumento de sesiones por encima del límite impuesto anteriormente no se traduce, necesariamente, en un mejor rendimiento del algoritmo. Para explicar esto, pueden distinguirse varios casos de usuarios.

Como se ha mencionado anteriormente, no siempre ha sido posible grabar cada sesión un día distinto. De hecho, en la mayoría de ocasiones, todas las sesiones han sido grabadas en dos días o incluso en uno solo. Esto provoca cierta correlación entre las sesiones, lo que llega a modular el resultado.

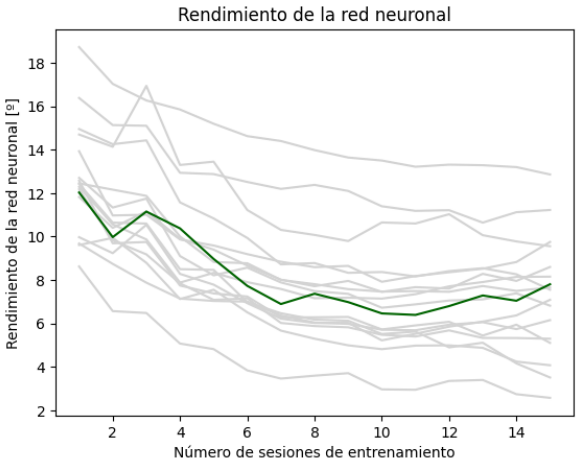

Figura 22 – Rendimiento de la red neuronal (media del ángulo de error) en función del número de sesiones de entrenamiento, para un usuario cuyos vídeos se grabaron en dos días diferentes

En el caso de un usuario cuyos vídeos se han grabado en dos días, las sesiones 0 a 4 se habrán grabado el primer día y las sesiones 5 a 9, el segundo. Al entrenar la red neuronal con 10 sesiones, se equilibra el *feedback* de los dos días. Sin embargo, si se calibra la red neuronal con 15 sesiones, se tomarán las 10 sesiones originalmente dedicadas al entrenamiento y otras 5 sesiones originalmente dedicadas al *test*. Si, al determinar el conjunto de entrenamiento, se toman las 5 sesiones adicionales del mismo día, el algoritmo se adaptará excesivamente a las condiciones específicas de dicho día, provocando un sobreentrenamiento y una disminución del rendimiento para el caso general.

Por otra parte, en el caso de un usuario que haya grabado todos los vídeos el mismo día, el sobreentrenamiento no se hará visible en estos resultados, dado que el rendimiento del algoritmo se computa en las mismas condiciones que aquellas en las que se ha producido esta sobreadaptación. En su lugar, la gráfica muestra cómo el rendimiento de la red neuronal sigue mejorando casi monótonamente, hasta llegar a cierta asíntota.

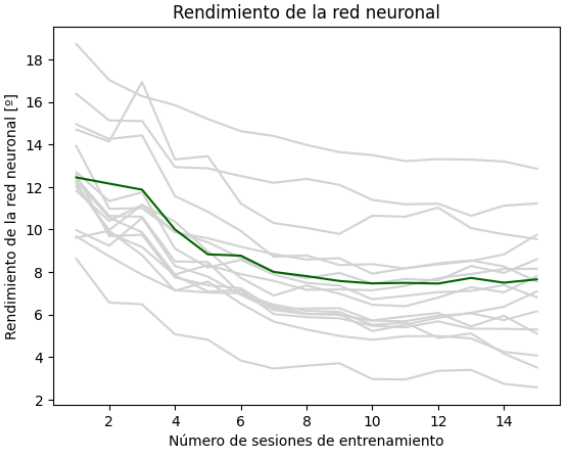

Figura 23 – Rendimiento de la red neuronal en función del número de sesiones de entrenamiento, para un usuario cuyos vídeos se grabaron en un solo día

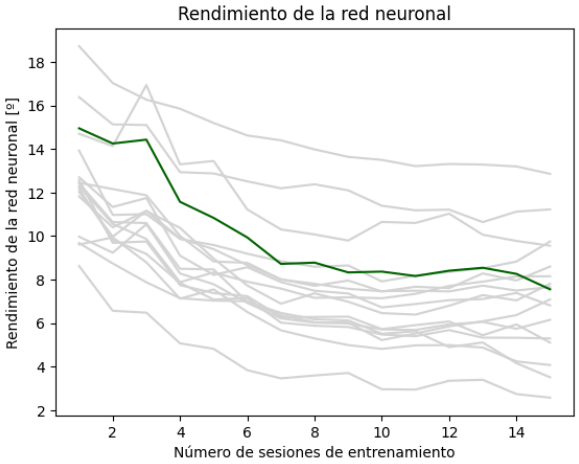

Figura 24 – Rendimiento de la red neuronal en función del número de sesiones de entrenamiento, para un usuario cuyos vídeos se grabaron a lo largo de 8 días diferentes

Por último, quedaría por evaluar el caso de un usuario cuyos vídeos se han grabado en diez días distintos. Aunque no se dé este caso concreto en la base de datos del proyecto, sí existen usuarios cuyos vídeos se han grabado repartidos a lo largo de muchos más días. Es esperable una leve mejora del rendimiento de la red neuronal una vez superado el umbral de las diez sesiones. pero la mejora del algoritmo debida a una mayor cantidad de datos de entrenamiento estaría reñida con cierto sobreentrenamiento de la red. El sobreentrenamiento no sería tan evidente como en el primer caso, ya que hay un mayor número de condiciones en las que se practica el experimento. No obstante, seguirán favoreciéndose algunas condiciones por encima de otras, ya que estarán presentes en el doble de datos de entrada que las demás. Y, por otro lado, las condiciones menos favorecidas serán aquellas presentes en los datos utilizados para el *test* de la red, por lo que el sobreentrenamiento se podrá apreciar en su máxima expresión.

Esta situación, sin embargo, no es específica para el entrenamiento con más de diez sesiones. Si las sesiones de grabación no se han distribuido de forma equitativa entre el número de días disponibles, el algoritmo se adaptará más, inevitablemente, a las condiciones del día en el que más sesiones se grabaron –y, por tanto, más datos de entrada a la red se extrajeron–.

De hecho, podemos extraer por separado el rendimiento que presenta la red neuronal para cada sesión. La gráfica siguiente compara el rendimiento de la red neuronal para los datos del primer día –en azul– y los del segundo –en rojo–, cuando se le ha entrenado solamente con las cinco sesiones grabadas el primer día:

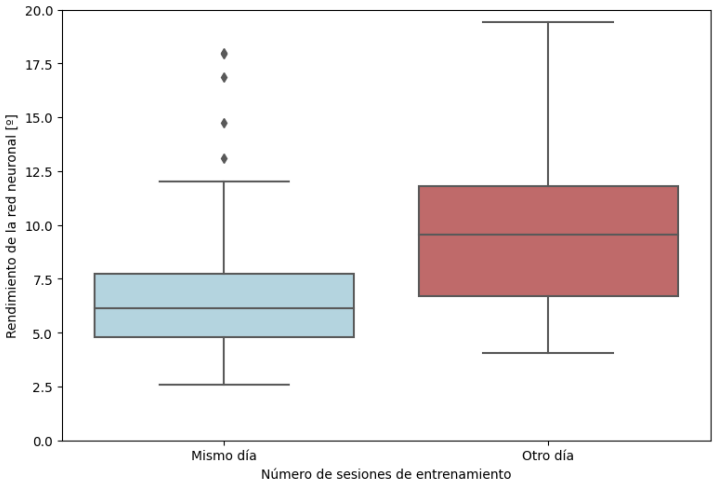

Figura 25 – Comparativa entre el rendimiento calculado en las sesiones de test del mismo u otro día que las sesiones de entrenamiento

Como se puede ver, el algoritmo funciona mejor cuando las condiciones son parecidas a aquellas de los datos con los que se ha entrenado. Podemos generalizar la figura, hasta conseguir la siguiente. En el eje de abscisas, se representa el número de sesiones utilizadas para el entrenamiento de la red neuronal. El rendimiento de la red en las sesiones equivalentes a las utilizadas en entrenamiento está representado en azul, mientras que el rendimiento en las demás sesiones figura en rojo.

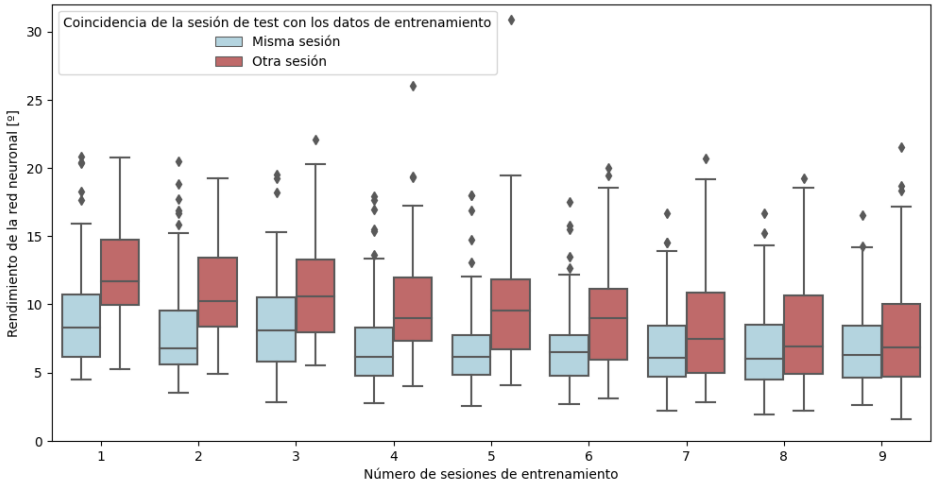

Figura 26 – Generalización de la figura anterior

Se puede apreciar cómo esta diferencia de rendimiento es notable, aunque disminuye a partir de la quinta sesión. Esto se puede deber a que el algoritmo empieza a tomar en cuenta datos del segundo día de grabaciones, por lo que la eficiencia a la hora de procesar las sesiones 5 a 10 mejorará.

Para concluir, se puede evaluar el límite del número de sesiones con que entrenar el algoritmo. En la mayoría de los casos, la mejora que se produce en el rendimiento del uso de siete sesiones es casi insignificante. Es decir, una vez se ha entrenado el algoritmo con siete sesiones de calibración, seguir aumentando el número de sesiones no mejorará, en gran medida, su desempeño. Sin embargo, es importante que las condiciones de grabación sean parecidas en el entrenamiento y en el *test*, por lo que será importante tener sesiones de calibración lo más actualizadas posible.

6. Conclusiones

En este proyecto, se ha construido un algoritmo que permite estimar, de manera aproximada, el punto al que el usuario está mirando en la pantalla. Para ello, ha sido preciso construir una base de datos propia, a partir de grabaciones hechas a distintos usuarios. De dichos vídeos, se han extraído, en primer lugar, los cuadros de interés, y finalmente, las características. Además, se han depurado los errores encontrados antes de utilizar los datos como entrada al algoritmo.

La red neuronal se ha construido a partir de un modelo base entrenado con la base de datos sintética U2Eyes. Se le han entregado los datos obtenidos anteriormente a este modelo base para calibrarlo y que se adapte mejor a su función con cada usuario, en una serie de experimentos en los que se ha definido el alcance que diversas modificaciones de la calibración podrían tener sobre el rendimiento del algoritmo.

En primer lugar, se han calculado los valores óptimos de *learning rate*, *batch size* y optimizador utilizado para obtener unos resultados óptimos sin sobrecargar el ordenador. A continuación, se ha verificado la importancia de entrenar la red neuronal con imágenes tomadas desde el mismo punto de vista y con el mismo usuario que los que serán utilizados en la aplicación real de la red; de hacer lo contrario, el algoritmo presenta un rendimiento mucho más pobre.

Por último, se ha definido el efecto que tiene el número de sesiones utilizadas en la calibración del modelo. A partir de siete sesiones, la mejora de los resultados es insignificante, por lo que no será preciso que, en la aplicación comercial, se le exija al usuario un número mayor de sesiones de calibración. Sin embargo, también se ha observado la importancia de usar sesiones de calibración grabadas en condiciones similares a las condiciones en las que el algoritmo se probará. Por ello, será indispensable tener vídeos de ajuste del algoritmo actualizados, y no permitir introducir demasiados cambios entre la grabación de los vídeos y la aplicación del algoritmo.

Aunque el algoritmo de este proyecto no presenta una precisión suficiente para la mayoría de las aplicaciones, sí puede constituir una buena base de partida para la elaboración de otros sistemas que sí puedan ofrecer una gran calidad. Algunas vías de mejora podrían ser un reconocimiento más preciso de los fotogramas erróneos o un reconocimiento automático de los instantes de interés en los vídeos de calibración. Estas mejoras podrían mejorar la calidad del sistema y permitir su implementación.

7. Bibliografía

[1] K. Zhang, Z. Zhang, Z. Li y Y. Qiao, *Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks* [En línea]. Disponible en: https://arxiv.org/abs/1604.02878

[2] P. Viola y M. Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features* [En línea]. Disponible en: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

[3] S. Porta *et al.*, *U2Eyes: a binocular dataset for eye tracking and gaze estimation* [En línea]. Disponible en: https://openaccess.thecvf.com/content_ICCVW_2019/papers/OpenEDS/Porta_U2Eyes_A_Binocular_Dataset_for_Eye_Tracking_and_Gaze_Estimation_ICCVW_2019_paper.pdf

[4] M. J. Islam, M. Ahmadi y M. A. Sid-Ahmed, *An Efficient Automatic Mass Classification Method in Digitized Mammograms Using Artificial Neural Network* [En línea]. Disponible en: https://aircconline.com/ijaia/V1N3/0710ijaia1.pdf

[5] E. Grossi y M. Buscema, *Introduction to artificial neural networks* [En línea]. Disponible en: https://www.researchgate.net/publication/5847739_Introduction_to_artificial_neural_networks

[6] M. Ryan, *How Neural Networks "Learn"*, Towards Data Science [En línea]. Disponible en: https://towardsdatascience.com/how-neural-network-learn-3b56c175b5ca

[7] M. Ryan, *How Neural Network Process Your Input (Trained Neural Network)*, Data Driven Investor [En línea]. Disponible en: https://medium.datadriveninvestor.com/how-neural-network-process-your-input-trained-neural-network-fd48f1bf310

[8] S. Lee, *Understanding Face Detection with the Viola-Jones Object Detection Framework*, Towards Data Science [En línea]. Disponible en: https://towardsdatascience.com/understanding-face-detection-with-the-viola-jones-object-detection-framework-c55cc2a9da14

[9] J. Brownlee, *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*, Machine Learning Mastery [En línea]. Disponible en:

https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/