Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Sistema de Detección de Somnolencia

Drowsiness Detection System

Hugo Fernández Solís

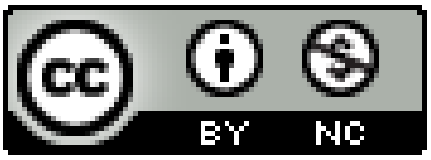
La Laguna, 13 de septiembre de 2022

Agradecimientos

A mis amigos, por haberme acompañado durante este largo viaje y haberme sacado una sonrisa cuando más lo he necesitado.

Y a mi familia, pilar fundamental en mi vida que siempre me ha apoyado en todo lo que hago y me ha dado fuerzas para seguir luchando.

Licencia



Abstract

La detección y clasificación de imágenes mediante el uso de Redes Neuronales y modelos de Deep Learning, es una de las ramas de la Inteligencia Artificial que más tiempo lleva en desarrollo y de las que mayor crecimiento ha experimentado. Esta clasificación de imágenes según su contenido, proporciona una herramienta muy potente a la hora de detectar objetos, personas, diferenciar animales... Y también para detectar acciones o situaciones que se estén produciendo en tiempo real.

En este proyecto estudiaremos el uso de estos modelos de Redes Neuronales para la clasificación de imágenes. Enfocando estos modelos en la resolución de un problema grave de la vida cotidiana, la somnolencia en los conductores.

El objetivo será desarrollar un sistema de detección de somnolencia que haga uso de redes neuronales para la clasificación de imágenes, y realizar un estudio posterior sobre el sistema y sus posibles mejoras, así como decidir cuáles parámetros consiguen que el sistema funcione mejor.

Para este trabajo se ha utilizado el lenguaje de programación Python, junto con diversas librerías que facilitan la interacción con el modelo, y otras que nos ayudan en la captura de las imágenes.

2 ***Palabras clave:*** *Redes Neuronales, Deep Learning, Inteligencia Artificial, Clasificación de Imágenes, Machine Learning*

Abstract

The detection and classification of images through the use of Neural Networks and Deep Learning models is one of the branches of Artificial Intelligence that has been in development for the longest time and has experienced the greatest growth. This classification of images according to their content provides a very powerful tool when it comes to detecting objects, people, differentiating animals... And also to detect actions or situations that are taking place in real time.

In this project we will study the use of these Neural Network models for image classification. Focusing these models on solving a serious problem in everyday life, drowsiness in drivers.

The objective will be to develop a drowsiness detection system that makes use of neural networks for image classification, and to carry out a subsequent study on the system and its possible improvements, as well as to decide which parameters make the system work better.

For this work, the Python programming language has been used, together with various libraries that facilitate interaction with the model, and others that help us in capturing the images.

2 ***Palabras clave:*** *Neural Networks, Deep Learning, Artificial Intelligence, Image Classification, Machine Learning*

Índice general

1. Introducción **1**
1.1. Motivación . . . 1
1.1.1. Motivaciones tecnológicas . . . 1
1.1.2. Motivaciones sociales . . . 1
1.2. Objetivos . . . 2
1.3. Fases del proyecto . . . 2

2. Antecedentes y estado del arte **4**
2.1. Clasificación de imágenes . . . 4
2.1.1. Modelos de Machine Learning . . . 4
2.2. Sistema ADAS . . . 6

3. Tecnologías utilizadas **7**
3.1. Hardware . . . 7
3.2. Software . . . 8

4. Estudio y desarrollo del sistema **9**
4.1. Estudio y primeras ideas . . . 9
4.1.1. Redes Neuronales . . . 9
4.1.2. Otros sistemas de detección de somnolencia . . . 17
4.1.3. Pre-procesamiento de las imágenes . . . 18
4.1.4. Imágenes durante la noche . . . 19
4.1.5. Selección de las áreas de interés para ojos y rostro . . . 19
4.2. Desarrollo del sistema . . . 20
4.2.1. Número de modelos . . . 20
4.2.2. Creación de los modelos . . . 20
4.2.3. Entrenamiento de los modelos . . . 22
4.2.4. Obtención y tratamiento de imágenes . . . 23
4.2.5. Clasificación de las imágenes . . . 26
4.2.6. Sistema de alarma . . . 28
4.2.7. Muestreo de los resultados . . . 29
4.2.8. Funciones de salida y control . . . 33
4.2.9. Componentes de streamlit . . . 34
4.3. Corrección de problemas . . . 36
4.3.1. Corrección de los datasets . . . 37

4.3.2. Adecuación al nuevo dataset 38

5. Informe técnico y análisis de los resultados 40

5.1. Ventajas y desventajas 40

5.1.1. Ventajas 40

5.1.2. Desventajas 41

6. Conclusiones y líneas futuras 42

6.1. Conclusiones 42

6.2. Líneas futuras 43

6.2.1. Desarrollo del sistema de detección de somnolencia . 43

6.2.2. Desarrollo del sistema de prueba del modelo 43

7. Summary and Conclusions 44

7.1. Conclusions 44

7.2. Future lines 45

7.2.1. Development of the drowsiness detection system . . . 45

7.2.2. Model test system development 45

8. Presupuesto 46

8.1. Costes materiales 46

8.2. Costes de horas de trabajo 46

Índice de Figuras

2.1. Localización + Clasificación de un elemento. 5
2.2. Elementos de un sistema ADAS. 6

4.1. Redes neuronales por número de capas. 9
4.2. Redes neuronales por tipo de conexiones. 9
4.3. Forma de una función sigmoidea. 10
4.4. Arquitectura de una CNN 13
4.5. Capas CNN dispuestas en 3 dimensiones. 14
4.6. Neuronas de una capa convolucional, conectadas a su campo receptivo. 15
4.7. Agrupación máxima con un filtro 2x2. 15
4.8. Causas de la somnolencia. 17
4.9. Dataset de ojos mediante cámara de visión nocturna. 19
4.10. Estructura de la red neuronal. 21
4.11. Algunas constantes del programa. 21
4.12. Imágenes de ejemplo del dataset. 22
4.13. Obtención del dataset. 22
4.14. Estructura del dataset. 23
4.15. Almacenamiento del dataset. 23
4.16. Entrenamiento del modelo. 23
4.17. Apertura de la cámara web. 24
4.18. Archivos haarcascade de opencv. 24
4.19. Lectura de imágenes de la cámara a modo de vídeo. 24
4.20. Proceso de clasificación del ojo izquierdo. 25
4.21. Lectura de resultados de la clasificación. 26
4.22. Análisis de la clasificación. 27
4.23. Inicialización de pygame. 29
4.24. Uso de la alarma. 29
4.25. Frame con el resultado final. 30
4.26. Código de muestreo del frame. 30
4.27. Resultado en formato texto. 31
4.28. Cantidad de frames despierto y dormido. 32
4.29. Establecer valores en st.empty. 33
4.30. Función de retardo. 33

4.31. Código del botón de salida. 33
4.32. Botón de salida. 34
4.33. Función de salida. 34
4.34. User inputs. 35
4.35. Subida y elección de modelos. 36
4.36. Elementos de streamlit 36
4.37. Ejemplos de los nuevos datasets. 37
4.38. Ejemplos de los nuevos datasets. 38
4.39. Xml local para la boca. 38
4.40. Cascade classifier con modelo de boca. 39
4.41. Cascade classifier de la boca. 39

Índice de Tablas

8.1. Costes materiales 46
8.2. Costes de horas de trabajo 46

- Capítulo 1 -

Introducción

1.1. Motivación

1.1.1. Motivaciones tecnológicas

El Machine Learning (ML) [1] o Aprendizaje Automático, es un campo de la Inteligencia Artificial, cuyo objetivo es desarrollar técnicas que, como su propio nombre indica, permitan que las computadoras aprendan. Se dice que un agente aprende cuando su desempeño mejora con la experiencia y mediante el uso de datos. De esta manera, un modelo informático bien entrenado, podría ser capaz de analizar una serie de datos y hacer clasificaciones o predicciones sobre estos datos con una fiabilidad muy alta.

La clasificación de imágenes mediante Redes Neuronales [2] y Deep Learning [3] o aprendizaje profundo, es una de las ramas de este campo de aprendizaje automático. En esta se intenta que un sistema sea capaz de obtener una imagen, analizarla, entenderla y actuar en consecuencia. Los usos de esta tecnología pueden ser muy variados. Desde conducción automática hasta clasificación de alimentos, pasando por cualquier actividad que requiera analizar y entender una imagen.

Es la idea de poder usar estas técnicas innovadoras para resolver un problema cotidiano lo que da motivación a este proyecto. Es una tecnología en constante evolución y que cada vez se va haciendo más común verla en el día a día.

1.1.2. Motivaciones sociales

En la actualidad, se producen muchos accidentes debido a errores humanos, lo que ocasiona muchas muertes y lesiones graves. Estos errores humanos se podrían reducir drásticamente con la ayuda de las tecnologías adecuadas.

Nosotros nos centraremos en la Somnolencia en los Conductores [4], lo

que produce entre el 15 % y el 30 % de los accidentes de tráfico en España. Nuestro objetivo será el desarrollo y estudio de un sistema que permita alertar al conductor de que está en estado de somnolencia y avisarle así del peligro. Pudiendo reducir drásticamente el número de accidentes causados por este efecto.

1.2. Objetivos

El principal objetivo de este TFG es el estudio de los sistemas de detección de somnolencia y su implementación. Así como analizar posibles mejoras que se puedan realizar a los mismos, ya sea mejorando el propio modelo, realizando un tratamiento previo a las imágenes que nos permita mejorar el análisis de las mismas.

Para realizar esto se partirá de unos sistemas ya creados y se estudiará su funcionamiento para, posteriormente, realizar mejoras en el modelo y el sistema de captura y procesamiento de imágenes.

1.3. Fases del proyecto

El proyecto se dividirá en las siguientes fases:

- La primera parte consistirá en un estudio de los distintos tipos de redes neuronales y la elección de una de estas para la implementación en el sistema.
- En la segunda parte se realizará un estudio de otros sistemas de detección de somnolencia ya existentes para poder observar ventajas e inconvenientes de los mismos y así poder tener una idea más clara del sistema a desarrollar.
- Una vez finalizadas las etapas de estudio, se procederá a la obtención de los datasets que se usarán para el entrenamiento de los modelos.
- Durante la cuarta etapa, se crearán las redes neuronales necesarias y se procederá con el entrenamiento.
- Posterior al desarrollo de las redes, se procederá a desarrollar el software de obtención y procesamiento de imágenes y se unirán a las redes neuronales para la clasificación de imágenes.

- En esta sexta etapa se procederá a desarrollar la interfaz de usuario para que este pueda modificar y experimentar con los valores, además de visualizar los resultados.
- Como penúltima etapa, se realizará un informe técnico donde se establezcan las características del sistema y sus ventajas e inconvenientes.
- Para finalizar, se desarrollará una memoria que abarque el desarrollo de este proyecto y la investigación realizada para llevarlo a cabo.

- Capítulo 2 -

Antecedentes y estado del arte

2.1. Clasificación de imágenes

La clasificación de imágenes consiste en conseguir que un ordenador la analice una imagen e identifique su composición, es decir, los objetos que hay en la misma. Este área fue una de las primeras en explorarse dentro del campo de la Inteligencia Artificial debido a la gran cantidad de tareas que pueden realizar y/o automatizar estos modelos y su gran impacto en la sociedad.

En sus orígenes, estos modelos se utilizaban principalmente para diferenciar objetos, por ejemplo tipos de flores, o saber qué animal hay en la imagen, aunque en los inicios ya existían modelos algo complejos que eran capaces de identificar caminos en una imagen y así poder conducir un vehículo siguiendo este camino.

Hoy en día estos sistemas están mucho más avanzados y ya son capaces de realizar tareas mucho más complejas, como conducir un vehículo por una ciudad estando atento a los peligros, señales, semáforos, demás vehículos... Detectar ciertas enfermedades viendo escáneres y radiografías realizadas a pacientes, servir de árbitros en diversos deportes como el tenis... Como podemos ver, el uso de estos modelos, puede ser llevado para una infinidad de tareas.

Como podemos ver, el uso de estos modelos, puede ser llevado para una infinidad de tareas

Es la idea de poder usar estas técnicas innovadoras para resolver un problema cotidiano lo que da motivación a este proyecto. Es una tecnología en constante evolución y que cada vez se va haciendo más común verla en el día a día.

2.1.1. Modelos de Machine Learning

Para poder realizar esta clasificación de imágenes, se utilizan los ya mencionados modelos de Machine Learning.

Estos modelos implementan unas redes neuronales cuya entrada sean los píxeles de una imagen y cuya salida sea un vector que nos indicará la información obtenida de la imagen.

Es muy complicado crear una red neuronal que sea capaz de identificar múltiples objetos dentro de una imagen y clasificarlos correctamente, es por ello que se suele utilizar la estrategia de Divide y Vencerás [5].

Este proceso consiste en crear varios modelos, cada uno especializado en la detección de un tipo de objeto en concreto, por ejemplo, personas, animales, señales de tráfico... Luego se envía la imagen a cada uno de estos modelos y así se obtiene cada uno de los objetos encontrados.

En nuestro caso, una aproximación a esta estrategia, podría ser la creación de un modelo que sea capaz de extraer la cara del sujeto del resto de la imagen, y luego usar otro modelo que sea capaz de analizar si dicho sujeto está en estado de somnolencia. Esto corresponde a un patrón de localización + clasificación [6].

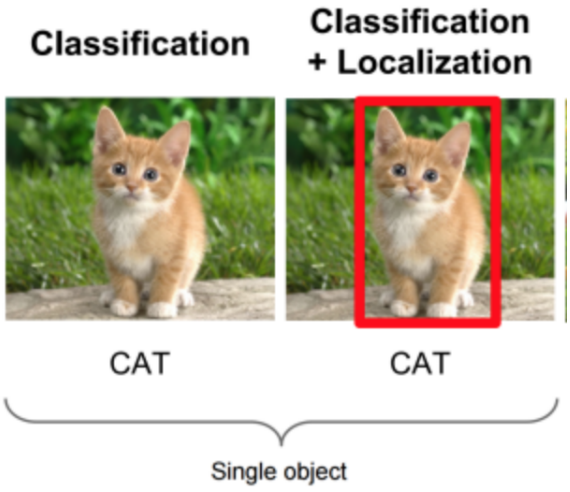


Figura 2.1: Localización + Clasificación de un elemento.

Uno de los modelos más famosos que existen es el modelo de conducción autónoma de Tesla [7]. Este sistema es capaz de reconocer una gran variedad de objetos del entorno y analizar los posibles peligros, al mismo tiempo que analiza las señales de tráfico y el camino que el vehículo debe seguir.

2.2. Sistema ADAS

Uno de los sistemas que implementan estos modelos de detección de somnolencia, son los sistemas ADAS (Advanced Driver Assistance System) [8]. Estos sistemas consisten en unos asistentes para la conducción que incorporan muchas funcionalidades, entre las que podemos encontrar:

- Sensor de ángulo de volante para examinar los micro-movimientos que realiza el conductor a lo largo del trayecto.
- Centralita del motor para conocer el tiempo que lleva el conductor sin detener el vehículo.
- Cámara incluida en los sistemas más sofisticados monitoriza las facciones del conductor y puede detectar si los ojos no están abiertos o si la cabeza no está erguida mirando hacia la calzada.
- Display de información al conductor (consola central).
- Y otros elementos mas...

Figura 2.2: Elementos de un sistema ADAS.

Este tipo de sistemas empezará a ser obligatorio en los nuevos vehículos a partir de 2022 según informa la DGT [9]. Lo que indica la importancia y utilidad de este tipo de servicios en la nueva era de la conducción.

Nosotros nos centraremos en un elemento en concreto, el sistema de detección de fatiga, y su utilidad y funcionamiento en estos sistemas ADAS mediante la clasificación de imágenes.

- Capítulo 3 -

Tecnologías utilizadas

3.1. Hardware

La realización de este proyecto ha sido realizada enteramente por tecnología hardware propia del alumno, gracias a que los modelos de Machine Learning [1] utilizados no requieren una gran complejidad para funcionar y/o ser entrenados.

Para ello hemos contado con un sobremesa montado por piezas con las siguientes características:

- CPU: 12th Gen Intel(R) Core(TM) i5-12600K 3.70 GHz
- GPU: NVIDIA GeForce GTX 1060 6GB
- RAM: 16GB
- SS.OO.: Windows 11 Pro

Este hardware ha sido más que suficiente para llevar a cabo el correcto desarrollo del sistema y el entrenamiento de los modelos de aprendizaje automático [10].

Aun así, cabe destacar, que la correcta implementación de un sistema de este estilo, debería ser realizada en un microprocesador que pueda ser integrado con facilidad en el vehículo.

3.2. Software

En cuanto al software utilizado en el desarrollo, ha sido variado para adaptarse a cada una de las etapas del proyecto.

- En cuanto a las herramientas de programación, hemos usado las siguientes:
  - El lenguaje de programación **Python 3.10** [11] con las siguientes librerías:
    - **numpy:** para el tratamiento de las imágenes [12].
    - **streamlit:** para la creación de la aplicación web [13].
    - **tensorflow:** para la creación de los modelos de Machine Learning [14].
    - **opencv-python:** para la captura y manipulación de imágenes, así como para la detección de ojos y rostro [15].
    - **scipy:** para obtener estadísticas sobre el funcionamiento del modelo [16].
    - **pygame:** para la función de alarma [17].
  - El editor **Visual Studio Code** [18], con la extensión de **python** [19], **github** [20] y **latex** [21].

- En cuanto a las herramientas para el control de versiones y almacenamiento del proyecto, hemos usado:
  - **Google Drive** [22].
  - **GitHub** [23].
  - **Git** [24].

- Por último, para la redacción de la memoria:
  - El editor de texto en formato LaTeX **Overleaf** [25].

- Capítulo 4 -

Estudio y desarrollo del sistema

4.1. Estudio y primeras ideas

4.1.1. Redes Neuronales

La primera fase del proyecto consistió en el estudio de las redes neuronales y los modelos de ML. Se estudiaron los tipos de redes más comunes en la clasificación de imágenes y sus ventajas y desventajas. Y se analizó otros proyectos, para observar cuáles eran las más utilizadas.

El primer paso fue analizar los tipos de redes neuronales existentes [26]:

- **Por número de capas:** Monocapas (capa de entrada y salida sin conexiones intermedias) o multicapas (múltiples capas intermedias entre la de entrada y la de salida).

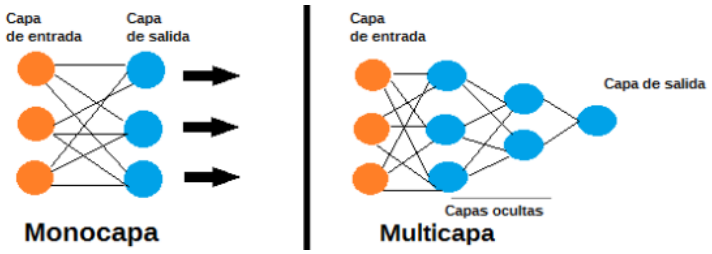


Figura 4.1: Redes neuronales por número de capas.

- **Por tipo de conexiones:** Recurrentes (La información avanza en un solo sentido) o no recurrentes (realizan conexiones de realimentación).

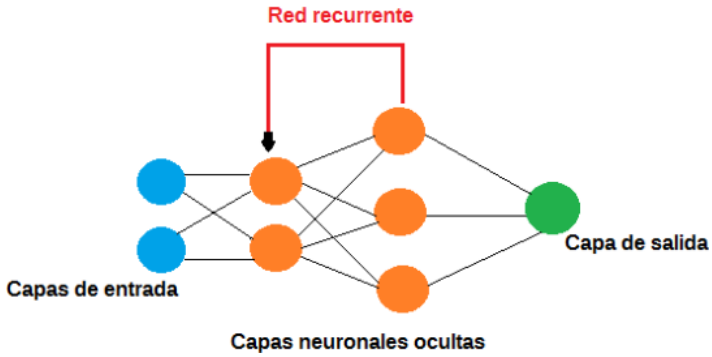


Figura 4.2: Redes neuronales por tipo de conexiones.

- **Por grado de conexiones:** Totalmente conectadas (las capas anteriores y posteriores están totalmente conectadas entre sí) o parcialmente conectadas (no todas las neuronas están conectadas entre ellas).

Aunque también hay otras clasificaciones más complejas como:

- Perceptrón multicapa - MLP
- Redes de base radial - RBF
- Redes neuronales recurrentes - RNN
- Redes neuronales convolucionales- CNN

Tras haber estudiado todas estas redes neuronales, se optó por utilizar una red CNN, la cuál es un tipo más complejo de las redes MLP.

Perceptrón multicapa - MLP

Estas son redes neuronales multicapas capaces de resolver problemas lineales no separables, lo cuál es a la vez su mayor limitación [27]. En estas redes, cada neurona utiliza una función de activación no lineal, y aprendizaje supervisado mediante retropropagación [28] para el entrenamiento.

Si un perceptrón multicapa tiene una función de activación lineal en todas las neuronas, es decir, una función lineal que asigna las entradas ponderadas a la salida de cada neurona, entonces, cualquier número de capas se puede reducir a una entrada de dos capas.

Las dos funciones de activación históricamente comunes son ambas sigmoideas [29] y están descritas por:

$$y(v_i) = tanh(v_i) \quad \text{y} \quad y(v_i i) = (1 + e^{-v})^{-1}$$

Estas funciones tienen una característica curva en forma de S o curva sigmoidea.

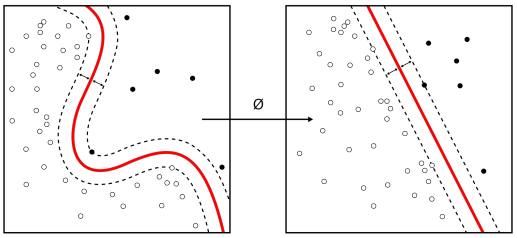


Figura 4.3: Forma de una función sigmoidea.

Sin embargo, la función de unidad lineal rectificador (ReLU) [30], se está usando con más frecuencia debido a su capacidad para superar los problemas numéricos relacionados con los sigmoides. Y se define como una función de activación definida como la parte positiva de su argumento:

$$f(x) = x^+ = max(0, x)$$

Esto también se conoce como función de rampa y es análogo a la rectificación de media onda en ingeniería electrónica.

En cuanto al aprendizaje, se utiliza el aprendizaje supervisado [31]. Una generalización del algoritmo de mínimos cuadrados medios [32] en el perceptrón lineal. Se puede representar el grado de error en un nodo de salida j en el n punto de datos (ejemplo de entrenamiento) por $e_j(n) = d_j(n) - y_j(n)$, dónde d es el valor objetivo e y es el valor producido por el perceptrón. Luego, los pesos de los nodos se pueden ajustar en función de las correcciones que minimizan el error en toda la salida, dada por:

$$\epsilon(n) = \frac{1}{2}\sum_j e_j^2(n)$$

Usando un descenso de gradiente, encontramos que nuestro cambio en cada peso es:

$$\triangle w_{ij}(n) = -\eta\frac{\partial\epsilon(n)}{\partial v_j(n)}$$

Dónde y_j es la salida de la neurona anterior y η es la tasa de aprendizaje, que se selecciona para garantizar que los pesos convergen rápidamente a una respuesta, sin oscilaciones.

La derivada a calcular depende del campo local inducido v_j, que en sí mismo varía. Es fácil probar que para un nodo de salida esta derivada se puede simplificar a:

$$-\frac{\partial\epsilon(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n))$$

Dónde ϕ' es la derivada de la función de activación descrita anteriormente, que en sí misma no varía. El análisis es más difícil para el cambio de pesos a un nodo oculto, pero se puede demostrar que la derivada relevante es:

$$-\frac{\partial\epsilon(n)}{\partial v_j(n)} = \phi'(v_j(n))\sum_k \frac{\partial\epsilon(n)}{\partial v_k(n)}w_{kj}(n)$$

Esto depende del cambio en los pesos de los k nodos, que representan la capa de salida. Entonces, para cambiar los pesos de la capa oculta, los pesos de la capa de salida cambian de acuerdo con la derivada de la función de activación, por lo que este algoritmo representa una propagación hacia atrás de la función de activación.

Estas redes son útiles en la investigación por su capacidad para resolver problemas estocásticamente. También son muy utilizadas para crear modelos matemáticos mediante análisis de regresión. Como la clasificación es un caso particular de regresión cuando la variable de respuesta es categórica , los MLP son buenos algoritmos clasificadores.

Redes neuronales convolucionales- CNN

Las redes neuronales convolucionales son un tipo especializado de redes MLP, que utilizan una operación matemática llamada convolución en lugar de la multiplicación general de matrices en al menos una de sus capas [13]. Están diseñados específicamente para procesar datos de píxeles. Este tipo de redes intenta recrear, de una manera muy similar, el funcionamiento de nuestro cerebro para reconocer imágenes.

Las CNN son versiones regularizadas de perceptrones multicapa . Los perceptrones multicapa generalmente significan redes completamente conectadas, pero la çonectividad total"de estas redes las hace propensas al sobreajuste. Las formas típicas de regularización, incluyen: penalizar los parámetros durante el entrenamiento o recortar la conectividad. Las CNN adoptan un enfoque diferente: aprovechan el patrón jerárquico en los datos y ensamble de patrones de complejidad creciente utilizando patrones más pequeños y simples grabados en sus filtros.

Una CNN consta de una capa de entrada, capas ocultas y una capa de salida. Las capas intermedias se denominan ocultas porque sus entradas y salidas están enmascaradas por la función de activación y la convolución [33] final. En una CNN, las capas ocultas realizan convoluciones. Por lo general, esto incluye una capa que realiza un producto escalar del kernel [34] de convolución con la matriz de entrada. Este producto suele ser el producto interno de Frobenius [35], y su función de activación es comúnmente ReLU. A medida que el kernel de convolución se desliza a lo largo de la matriz de entrada, la operación de convolución genera un mapa de características, que a su vez contribuye a la entrada de la siguiente capa.

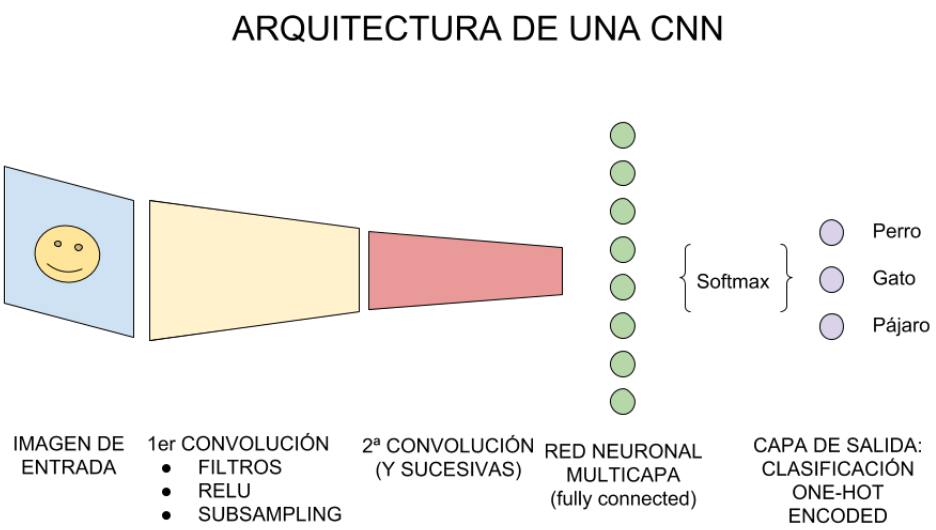


Figura 4.4: Arquitectura de una CNN

Después de pasar por una capa convolucional, la imagen se abstrae en un mapa de características, también llamado mapa de activación, con forma: $(entradas)\ x\ (altura)\ x\ (ancho)\ x\ (canales)$.

Las capas convolucionales convolucionan la entrada y pasan su resultado a la siguiente capa. Esto es similar a la respuesta de una neurona en la corteza visual a un estímulo específico.

Entre los rasgos distintivos de una CNN podemos encontrar:

- **Volúmenes 3D de neuronas:** Las capas de una CNN tienen neuronas dispuestas en 3 dimensiones: ancho, alto y profundidad. Donde cada neurona dentro de una capa convolucional está conectada solo a una pequeña región de la capa anterior, llamada campo receptivo.
- **Conectividad local:** Las CNN explotan la localidad espacial imponiendo un patrón de conectividad local entre neuronas de capas adyacentes. Por lo tanto, la arquitectura asegura que los "filtros.ªprendidos produzcan la respuesta más fuerte a un patrón de entrada espacialmente local. Apilar muchas de estas capas conduce a filtros no lineales que se vuelven cada vez más globales (es decir, responden a una región más grande del espacio de píxeles) de modo que la red primero crea representaciones de pequeñas partes de la entrada y luego a partir de ellas ensambla representaciones de áreas más grandes.
- **Pesos compartidos:** Cada filtro se replica en todo el campo visual. Estas unidades replicadas comparten la misma parametrización (vector

de peso y sesgo) y forman un mapa de características. Esto significa que todas las neuronas en una capa convolucional determinada responden a la misma característica dentro de su campo de respuesta específico.

- **Agrupación:** Los mapas de características se dividen en subregiones rectangulares, y las características en cada rectángulo se muestrean de forma independiente a un solo valor. Además de reducir los tamaños de los mapas de características, la operación de agrupación otorga un grado de invariancia traslacional local a las características contenidas en ellos, lo que permite que la CNN sea más robusta a las variaciones en sus posiciones.

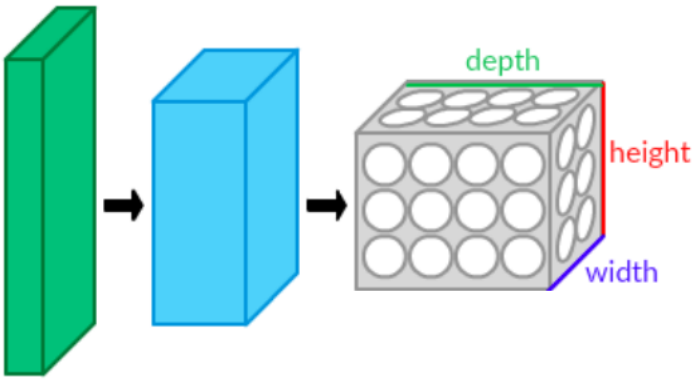


Figura 4.5: Capas CNN dispuestas en 3 dimensiones.

Estas propiedades permiten que las CNN logren una mejor generalización de los problemas de visión.

Tres hiperparámetros controlan el tamaño del volumen de salida de la capa convolucional: la profundidad, la zancada y el tamaño del relleno.

- La profundidad del volumen de salida controla el número de neuronas en una capa que se conectan a la misma región del volumen de entrada. Estas neuronas aprenden a activarse para diferentes funciones en la entrada.

- Stride controla cómo se asignan las columnas de profundidad alrededor del ancho y la altura. Para cualquier número entero $S > 0$,un paso S significa que el filtro se traduce S unidades a la vez por salida. En la práctica, $S > 3$ es raro. Una mayor zancada significa una menor superposición de campos receptivos y menores dimensiones espaciales del volumen de salida.

- El relleno proporciona control del tamaño espacial del volumen de salida, esto se conoce comúnmente como relleno "mismo".

El tamaño espacial del volumen de salida es una función del tamaño del volumen de entrada W, el tamaño del campo del kernel K de las neuronas de la capa convolucional, la zancada S, y la cantidad de relleno cero P en el borde. El número de neuronas que "encajan"en un volumen dado es entonces:

$$\frac{W - K + 2P}{S} + 1$$

Si se analizan las capas de una CNN, las primeras y más distintivas son las capas convolucionales. Estas son el bloque de construcción central de una CNN.

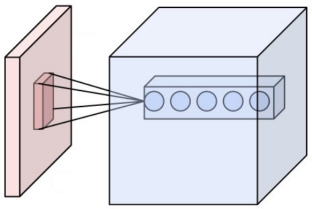


Figura 4.6: Neuronas de una capa convolucional, conectadas a su campo receptivo.

Los parámetros de la capa consisten en un conjunto de filtros aprendibles, que tienen un campo receptivo pequeño, pero se extienden a lo largo de toda la profundidad del volumen de entrada. Cada filtro se convoluciona a lo ancho y alto del volumen de entrada, calculando el producto escalar entre la entrada y el filtro, produciendo un mapa bidimensional de ese filtro. Como resultado, la red aprende filtros que se activan cuando detecta alguna característica en alguna posición en la entrada.

La capa de agrupación divide la imagen de entrada en un conjunto de rectángulos y, para cada subregión, genera el máximo

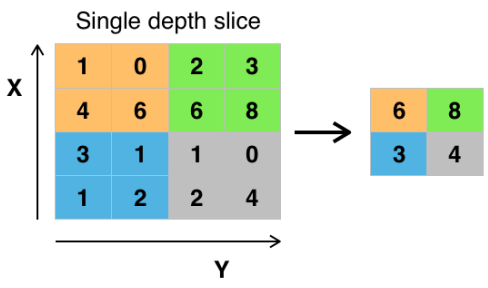


Figura 4.7: Agrupación máxima con un filtro 2x2.

La ubicación exacta de una característica es menos importante que su ubicación aproximada con otras características. Esta es la idea detrás del uso de la agrupación en redes neuronales convolucionales. La capa de agrupación sirve para reducir progresivamente el tamaño espacial de la representación, para reducir la cantidad de parámetros, la huella de memoria y la cantidad de cómputo en la red y, por lo tanto, también para controlar el sobreajuste [36].

La capa de agrupación normalmente opera de forma independiente en cada profundidad, o corte, de la entrada y la redimensiona espacialmente.

$$f_{x,y}(S) = \max_{a,b=0}^{1} S_{2x+a2y+b}$$

En cuanto a las funciones de activación, la más usada es ReLU, que aplica una función no saturada y elimina los valores negativos de un mapa de activación al establecerlos en cero e introduce no linealidades en la función de decisión sin afectar los campos receptivos de las capas de convolución.

$$f(x) = max(0, x)$$

En la última capa se suele emplear otro tipo de función para la clasificación final. La función softmax por ejemplo, que convierte un vector de K números en una distribución de probabilidad de K resultados posibles.

La función softmax $\sigma : \mathbb{R}^K \to (0,1)^K$ cuando K es mayor que uno es:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K}} e^{z_i}$$

Otro elemento fundamental en las CNN son las funciones $loss$. Estas especifican cómo el entrenamiento penaliza la desviación entre la salida prevista de la red y las etiquetas de datos reales. Se pueden utilizar varias funciones de pérdida , dependiendo de la tarea específica. Por ejemplo la función de entropía binaria, que se define como la entropía de un proceso de Bernoulli con probabilidad p de uno de dos valores. Esta muy bien al sistema gracias a su clasificación binaria.

$$H(x) = H_b(p) = -p \log_2 p - (1-p) \log_2 (1-p)$$

Elección de red neuronal

Una vez analizados los tipos más comunes de redes neuronales, el siguiente paso era escoger cuál sería la utilizada para el sistema. Llegados a este punto, no quedaba ninguna duda que se usaría una CNN. Estas redes están diseñadas específicamente para el reconocimiento y la clasificación de imágenes y son muchas las ventajas que ofrecen para ello respecto a las demás redes.

4.1.2. Otros sistemas de detección de somnolencia

La continuación del estudio se centró en otros sistemas de detección de somnolencia ya existentes y su funcionamiento. Durante este proceso, se observó que la mayoría de sistemas estudiados empleaban redes CNN. Reivindicando esto que, se había escogido el modelo adecuado. Además, se detectó un paso no considerado, el preprocesamiento de las imágenes.

También ayudó a determinar cómo detectar si un conductor estaba en estado de somnolencia o no, y a analizar cuáles son las principales causas de la somnolencia. Tras analizar todos los sistemas previos, se optó por analizar los ojos y los bostezos del conductor para detectar este estado.

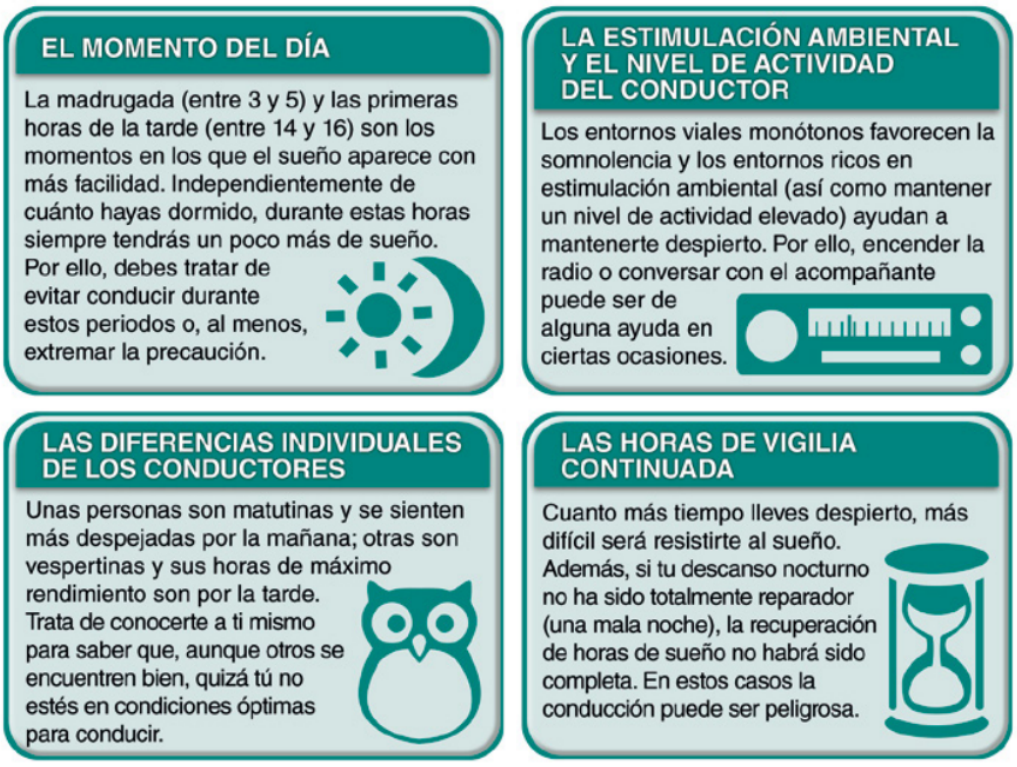


Figura 4.8: Causas de la somnolencia.

4.1.3. Pre-procesamiento de las imágenes

En este paso se realizó un estudio sobre las posibles operaciones de preprocesamiento de imágenes y cuales eran sus ventajas y desventajas.

Existen una gran multitud de operaciones que se pueden realizar sobre una imagen para "mejorarla". Sin embargo, encontramos relevantes 3 de estas operaciones:

- **Redimensión de una imagen:** Este es el proceso de modificar el tamaño, intentando perder la menor cantidad de información posible.

  Esta operación puede ayudar a mejorar el rendimiento reduciendo significativamente el número de operaciones que la red debe realizar. Pero también hay que tener en cuenta que una reducción drástica del tamaño de la imagen puede llevar a una pérdida considerable de información y, por lo tanto, reducir la eficacia del modelo.

  Se realizó una búsqueda sobre los mejores tamaños para el procesamiento de imágenes y se encontró opiniones muy variadas, desde 255x255 hasta 22x22. Variando en función del tamaño original de la imagen, el contenido de la misma, la velocidad de procesamiento disponible... Así que, se decidió que lo mejor sería realizar pruebas con el modelo para decidir el tamaño idóneo.

- **Área de interés:** Consiste en seleccionar una zona de la imagen en la que nos centraremos y descartar el resto.

  El sistema se centrará en detectar si el conductor está en estado de somnolencia mediante los ojos y los bostezos. Esta operación consistirá en seleccionar las áreas del ojo y la cara para analizar únicamente esa porción de imagen, mejorando la eficacia y rendimiento del modelo.

- **Escala de grises:** Este filtro consiste en modificar una imagen a color para transformarla en una imagen en tonos de gris.

  Una imagen digital está compuesta, generalmente, por 3 canales, uno para cada uno de los colores primarios, rojo, verde y azul. Es decir, cada píxel tendrá 3 valores de color. Juntando estos 3 colores se pueden obtener todos los demás.

  Si se tiene una imagen de 255x255 supondría un total de 65.025 píxeles, que al tener 3 canales supondrían un total de 195.075 entradas que tendría que evaluar la red neuronal. La operación de pasar a escala de grises esta imagen dejaría tan solo las 65.025 entradas, lo

que supone reducir la cantidad de operaciones en un 75 %. Es por ello que se decidió trabajar con imágenes en escala de grises.

Gracias a este estudio surgió otro problema a tener en cuenta. Los conductores suelen tener más somnolencia por la noche, cuando se acerca o ha pasado la hora de dormir. Esto supone un problema, pues en la noche es muy difícil capturar una imagen mediante una cámara normal sin deslumbrar al conductor, lo que dejaba una pregunta en el aire.

¿Cómo obtener imágenes durante la noche? y ¿Cómo procesarlas?

4.1.4. Imágenes durante la noche

Tras investigar varias soluciones, se encontró que la más fiable y que mejor se había asentado en el mercado, eran las cámaras de visión nocturna.

Estas cámaras funcionan mediante luz infrarroja, que es invisible al ojo humano, pero que es capaz de detectar escenas nocturnas con bastante claridad. Además, este tipo de imágenes son devueltas en escalas de grises.

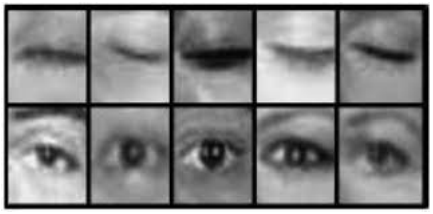


Figura 4.9: Dataset de ojos mediante cámara de visión nocturna.

4.1.5. Selección de las áreas de interés para ojos y rostro

Se va a implementar un sistema que trabaje únicamente con el rostro y los ojos del conductor, por lo que es necesario ser capaces de diferenciar estos del resto de la imagen.

Se descubrió que Opencv ya viene con unos modelos pre-entrenados capaces de realizar esta tarea con un gran rendimiento. Debido a que ya se iba a utilizar dicha librería, se optó por utilizar este sistema.

4.2. Desarrollo del sistema

Una vez que se realizó el estudio necesario y se disponía de una idea clara del sistema a desarrollar, el siguiente paso fue comenzar con el desarrollo del mismo.

4.2.1. Número de modelos

Como vimos en el capítulo 2, los modelos de clasificación de imágenes pueden ser muy complejos. Es por ello que se optó por una estrategia de Divide y Vencerás. Como el sistema trabaja detectando si los ojos estaban abiertos o cerrados y con los bostezos del conductor, se optó por dividir este problema en dos. Creando así dos modelos, uno para detectar los bostezos y otro para los ojos.

A pesar de que ambos modelos vayan a detectar cosas distintas, ambos son modelos de clasificación de imágenes, por lo que se construyeron dos CNN con las mismas capas y funciones. La única diferencia fue cómo se entrenaron.

4.2.2. Creación de los modelos

Para ello se usó la librería tensorflow con keras, y se inició un modelo secuencial con las capas de la red:

- **3 capas convolucionales:** Estas capas son capaces de identificar patrones en la imagen mediante un kernel y proporcionan una salida acentuando ciertas características de la imagen. Cuantas más capas haya, se podrán reconocer características más complejas.
- **3 capas max pooling:** Ayuda a reducir el tamaño de una entrada seleccionando el valor máximo en una pequeña región.
- **2 capas dropout:** De forma aleatoria, establece algunos valores a 0 para evitar un sobre-ajuste.
- **2 capas dense:** Esta se encarga de llamar a la función de activación sobre la entrada y devolvernos el resultado como salida.
- **1 capa flatten:** Convierte una matriz en un vector unidimensional.

```
model: tf.keras.models.Sequential = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), input_shape=\
    (IMAGE_SIZE, IMAGE_SIZE, 1), activation="relu"),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation="relu"),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation="relu"),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation="relu"),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(units=2, activation="sigmoid")
])
```

Figura 4.10: Estructura de la red neuronal.

Funciones de activación

La función de activación será relu para todas las capas menos para la final, que usaremos la función sigmoid. La función relu analiza todos los valores y pone a 0 todos aquellos que sean negativos.

$$f(x) = max(0, x)$$

Para la función de activación de la capa final, se estudió usar las funciones softmax y sigmoid. Ambas muy ampliamente utilizadas en la clasificación de imágenes. La diferencia entre ambas funciones, es que la sigmoid está diseñada para realizar una regresión sobre 2 clases, mientras que la softmax es multiclase. Debido a que en el proyecto se plantean únicamente dos clases, se optó por usar la función sigmoid.

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} = 1 - S(-x)$$

Algunas variables del modelo

El valor de IMAGE_SIZE corresponde al tamaño de la imagen de entrada. Se decidió trabajar con imágenes de 24x24, ya que mejoraba mucho el rendimiento del modelo y los resultados eran muy fiables.

```
TARGET_SIZE: tuple[int, int] = (24, 24)
COLOR_MODE: str = "grayscale"
ACCURACY: float = 0.95
IMAGE_SIZE: int = 24
BATCH_SIZE: int = 32
EPOCHS: int = 99999
```

Figura 4.11: Algunas constantes del programa.

El resto de variables corresponden al tamaño de los kernels, filtros, etc. Estos valores se han establecido de forma arbitraria siguiendo el estándar proporcionado por la página oficial de keras para la creación de modelos de clasificación de imágenes [37].

4.2.3. Entrenamiento de los modelos

Datasets

Para el entrenamiento de los modelos, el primer paso debía ser la obtención de un dataset. Tras realizar una búsqueda en internet, se encontró un dataset en data-flair [38], con 2900 imágenes de ojos (abiertos y cerrados) y de rostros (bostezando y sin bostezar).

(a) Bostezo del dataset.

(b) Ojo del dataset.

Figura 4.12: Imágenes de ejemplo del dataset.

El siguiente paso consistió en preparar el dataset para entrenar el modelo. Para ello se utilizó la clase ImageDataGenerator de Keras, incluida en la librería tensorflow. Esta clase recibe un directorio como parámetro que posea varias subcarpetas y en cada una de estas, varias imágenes. Luego creará un dataset con todas las imágenes y a cada una de ellas se le asignará una clase, la cual consiste en un índice numérico que corresponderá con el nombre de la subcarpeta en la que se encuentre la imagen.

```
def generator(dir, shuffle=True, batch_size=1, target_size=TARGET_SIZE,
              class_mode='categorical'):
    return tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)\
            .flow_from_directory(dir, batch_size=batch_size, shuffle=shuffle,
            color_mode=COLOR_MODE, class_mode=class_mode, target_size=target_size)
```

Figura 4.13: Obtención del dataset.

En este caso, para los ojos por ejemplo, se tendrían dos subcarpetas, una con los ojos abiertos que se llamará *open* y otra con los ojos cerrados que se llamará *close*. A la carpeta *close* se le asignaría el índice 0 y a la carpeta

open el índice 1 (por orden alfabético). Así pues, a cada clase de la carpeta *close* se le asignará la clase 0 y a las de la carpeta *open* la clase 1.

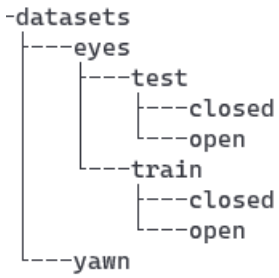


Figura 4.14: Estructura del dataset.

Estos datasets se almacenan en unas variables para después entrenar el modelo.

```
trainBatch = generator(dir, batch_size=BATCH_SIZE, target_size=TARGET_SIZE)
validBatch = generator(dir, batch_size=BATCH_SIZE, target_size=TARGET_SIZE)
```

Figura 4.15: Almacenamiento del dataset.

Una vez que los datasets están preparados, se inicia la fase de entrenamiento. Para ello, la librería tensorflow, ofrece una función *fit* a la que le se le pasa como parámetro ambos datasets y entrenará al modelo con ellos.

```
model.fit(
    trainBatch,
    validation_data=validBatch,
    epochs=epochs,
    callbacks=[Callback()],
    shuffle = True
)
```

Figura 4.16: Entrenamiento del modelo.

La variable epochs indica la cantidad de veces que ejecutará el proceso de entrenamiento, mientras que Callback, es una clase creada para que cuando el dataset supere un cierto umbral de aciertos, detenga el entrenamiento.

Una vez que termine este proceso, el modelo estará listo para realizar predicciones.

4.2.4. Obtención y tratamiento de imágenes

Una vez que el modelo está listo para realizar las predicciones, el siguiente paso era la obtención de las imágenes para probar el funcionamiento del modelo.

Para ello se utilizó la librería OpenCV [15], especializada en el tratamiento de imágenes, y su clase VideoCapture [39], que nos permite abrir una cámara del sistema.

```
cap: cv.VideoCapture = cv.VideoCapture(0 + cv.CAP_DSHOW)
```

Figura 4.17: Apertura de la cámara web.

El índice 0 indica que debe abrir la cámara web por defecto del sistema. Mientras que CAP_DSHOW le indica que va a trabajar con otros hilos. Esto es necesario para que pueda sincronizarse con la librería streamlit.

El siguiente paso es leer la imagen, pero antes, debemos detectar el rostro y los ojos del conductor. Para ello se utilizan los archivos de opencv haarcascade. Estos utilizan un algoritmo de detección de elementos. En este caso se usarán 3 de ellos: frontalface, lefteye y righteye.

```
faceCascade = cv.CascadeClassifier(f"{cv.data.haarcascades}haarcascade_frontalface_alt2.xml")
leftEyeCascade = cv.CascadeClassifier(f"{cv.data.haarcascades}haarcascade_lefteye_2splits.xml")
rightEyeCascade = cv.CascadeClassifier(f"{cv.data.haarcascades}haarcascade_righteye_2splits.xml")
```

Figura 4.18: Archivos haarcascade de opencv.

Una vez que la cámara está abierta y los clasificadores haarcascade preparados, se leerá de esta con la función read. En nuestro caso, como se quiere que la cámara esté siempre leyendo a modo de vídeo, se crea un bucle while que continúe leyendo mientras tenga acceso a la cámara.

```
while(cap.isOpened()):
    ret, frame = cap.read()
```

Figura 4.19: Lectura de imágenes de la cámara a modo de vídeo.

Una vez establecidos los clasificadores y obtenida la imagen, se procede a detectar el rostro y ojos del conductor y a almacenar los resultados en unos arrays. Estos tres arrays, contendrán las coordenadas de cada uno de los elementos dentro del frame original. Además, podemos observar como previamente se ha pasado la imagen a escala de grises usando la funcionalidad de opencv para ello (por las mejoras en el rendimiento mencionadas anteriormente).

Una vez que ya se dispone de todos los elementos diferenciados, se procede a detectar si el conductor está en estado de somnolencia o no. Para ello se recorta de la imagen los elementos dichos y se les envía al modelo para que este devuelva su predicción.

El proceso es el siguiente:

- Se recogen los valores de la primera coordenada dentro de la imagen, y su ancho y alto, siendo x e y las coordenadas y w y h el ancho y alto
- Se recorta ese segmento de la imagen para trabajar con ella por separado.
- Se normaliza la imagen. Esto es dividir todos los valores por 255 para así trabajar con números entre 0 y 1, lo que mejora enormemente el rendimiento del modelo.
- Se modifica el tamaño de la imagen para que coincida con el tamaño de entrada del modelo.
- Se añade un nuevo eje a la imagen con la función expand_dims para que coincida con el tamaño de entrada del modelo.
- Se pasa esta nueva imagen procesada al modelo para que realice su predicción.
- Se colorea un recuadro alrededor del elemento analizado en la imagen en color para dar feedback al usuario sobre lo que el sistema ha detectado como ese elemento.

En la siguiente imagen se puede ver el proceso para el ojo izquierdo (para el ojo derecho y el rostro, el proceso sería exactamente el mismo):

```
for (x, y, w, h) in leftEyeShape:
    leftEye: np.ndarray = grayFrame[y:y+h,x:x+w]
    leftEye: np.ndarray = cv.resize(leftEye, (mg.IMAGE_SIZE, mg.IMAGE_SIZE))
    leftEye: np.ndarray = leftEye / 255
    leftEye: np.ndarray = leftEye.reshape(mg.IMAGE_SIZE, mg.IMAGE_SIZE, -1)
    leftEye: np.ndarray = np.expand_dims(leftEye,axis=0)
    leftEye: np.ndarray = model.predict(leftEye, verbose=0)[0]
    cv.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)
```

Figura 4.20: Proceso de clasificación del ojo izquierdo.

Una vez finalizada la clasificación de imágenes. Se dispone de 3 vectores con el resultado de la clasificación. Y dichos vectores tendrán dos elementos. La primera posición nos indica la probabilidad de que el ojo esté abierto, mientras que la segunda, la probabilidad de que este esté cerrado. Es decir, el modelo indica que está un 92.76 % seguro de que el ojo está cerrado.

Una vez que se tienen los 3 vectores comienza el proceso de interpretarlos. La primera idea que se planteó fue encontrar el elemento más grande dentro de cada uno de los vectores y obtener su índice. Si el índice era el 0, indicaba que estaba despierto, de lo contrario, estaría somnoliento. Pero esto conlleva realizar primero una búsqueda del elemento más grande y luego una búsqueda de la posición de dicho elemento en el vector. Así que tras una breve reflexión, se decidió establecer un método más sencillo.

Para saber qué elemento es más grande, basta con comparar el primero y analizar si este es mayor que ***0.5***. Esto es gracias a que la suma de las probabilidades del modelo, siempre va a ser 100 % (con un pequeño margen de error en décimas). Es decir, si un elemento del modelo tiene un 60 % de probabilidad, el otro va a tener una probabilidad del 40 %. Por lo tanto, aquel que supere el 50 %, se podrá decir casi con total seguridad, que es el mayor de los elementos del vector.

4.2.5. Clasificación de las imágenes

En la siguiente imagen se encuentra la lectura de esta clasificación como se menciona anteriormente.

```
# Analizamos los ojos.
if leftEye[0] < 0.5 or rightEye[0] < 0.5:
    eyesTag: str = "Open"
else:
    eyesTag: str = "Close"

# Analizamos el rostro.
if face[0] < 0.5:
    yawnTag: str = "No"
else:
    yawnTag: str = "Yes"
```

Figura 4.21: Lectura de resultados de la clasificación.

Si uno de los ojos está abierto, se sume que el conductor está despierto y que tan solo estará guiñando un ojo o algo parecido, pues en estado de somnolencia lo más común es cerrar ambos ojos. En cuanto al rostro, basta con que el modelo detecte un bostezo que ya estará en estado de somnolencia. ***eyesTag*** y ***yawnTag*** almacenarán el resultado.

Una vez que se tiene la clasificación del rostro y los ojos, se procede a analizar este resultado y a actuar en consecuencia. El análisis de los resultado sigue las siguientes reglas:

- Si uno de los ojos está abierto, no se contará como estado de somnolencia (Esto se puede observar en la imagen anterior).

- Basta con que ambos ojos estén cerrados o el conductor bostezando para establecer un estado de somnolencia, no hace falta que sean los dos a la vez.
- En caso de que haya un error o ambigüedad en la interpretación de los resultados, ese frame no se tendrá en cuenta y se pasará al siguiente.

En la siguiente imagen se puede ver el proceso de análisis:

```
if yawnTag == "No" and eyesTag == "Open":
    score -= scoreDecrease
    thicc -= thiccDecrease
    if score < 0:
        score = 0
    if thicc < 0:
        thicc = 0
    label: str = "Awake"
    framesAwake += 1
else:
    score += scoreIncrease
    if score > warningScore:
        thicc += thiccIncrease
    if score > maxScore:
        score = maxScore
    if thicc > maxThicc:
        thicc = maxThicc
    label: str = "Sleep"
    framesSleep += 1
```

Figura 4.22: Análisis de la clasificación.

La aplicación lleva un sistema de puntuaciones incorporado para no hacer sonar la alarma desde que el conductor cierre los ojos o bostece, pues recordemos que una persona puede pestañear y no significa que esté en estado de somnolencia. En el primer ***if*** se comprueba si el conductor no bosteza y tiene los ojos abiertos. Si se cumplen ambas condiciones, se asume que el conductor está despierto y se reducen las puntuaciones. En el caso contrario, se asume que el conductor está en estado de somnolencia y se aumentan.

Este sistema de puntuaciones lleva una cuenta de la cantidad de frames consecutivos que lleva el conductor en estado de somnolencia y, cuando se supere un umbral establecido por el usuario, comenzará a sonar la alarma.

Para lograr esto se establecen dos variables fundamentales. ***score*** y ***thicc***. ***Score*** irá aumentando a medida que detecte el estado de somnolencia, a la vez que disminuirá en caso contrario. ***Thicc*** actúa de una manera muy similar, pero esta solo crecerá cuando se detecté somnolencia y, además, se haya superado el umbral establecido por el usuario.

Esto ayuda de varias maneras. Para empezar, permite no activar el sistema de alarma a la mínima que el usuario pestañee o bostece, si no que se puede definir un tiempo para detectar la somnolencia. Además, gracias a thicc se puede controlar el volumen de la alarma, haciendo que esta se acentúe cuanto más tiempo lleve dormido el usuario.

También permite activar el sistema de alarma de forma inmediata si el cierre de ojos o bostezos del conductor son muy constantes. Esto sucedería si el usuario lleva unos segundos despierto pero todavía sigue por encima del umbral.

Si desglosamos la imagen anterior podemos ver:

- **Reducción de las puntuaciones:** En caso de que el conductor tenga los ojos abiertos y no bostece, se reduce el valor de ***score*** y ***thicc*** mediante unos valores arbitrarios que también podrá modificar el usuario.

  Además, se comprueba que el valor de estos no sea inferior a 0. En caso de serlo, se establecen a 0.

- **Aumento de las puntuaciones:** En caso de que el conductor tenga los ojos cerrados o bostece, se aumenta el valor de ***score*** y ***thicc*** mediante unos valores arbitrarios que también podrá modificar el usuario.

  Además, se comprueba que el valor de estos no sea superior al límite también establecido por el usuario. En caso de serlo, se establecen a l valor máximo.

Todos estos valores arbitrarios mencionados y que puede modificar el usuario, se detallarán más adelante.

4.2.6. Sistema de alarma

Para el sistema de alarma, al principio de todo, antes de iniciar streamlit o la captura de imágenes, se debe inicializar pygame para poder reproducir sonidos cuando sea necesario. Para ello se pueden usar las siguientes líneas de código, que inicializan el mixer de pygame y guarda el sonido de alarma en la variable sound para usarlo cuando se requiera.

```
pg.mixer.init()
sound = pg.mixer.Sound('alarm.wav')
```

Figura 4.23: Inicialización de pygame.

Una vez que se ha guardado el sonido en una variable y se ha realizado la clasificación de las imágenes, se procede a utilizar el sonido.

```
sound.set_volume(thicc/maxThicc)
if label == "Sleep" and score > warningScore:
    try:
        sound.play()
    except:
        pass
if score < warningScore:
    try:
        sound.stop()
    except:
        pass
```

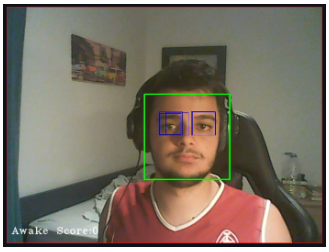
Figura 4.24: Uso de la alarma.

Primero se establece el volumen de la alarma. Esta empezará al mínimo cuando ***thicc*** esté a 0 e irá aumentando hasta llegar al máximo cuando ***thicc*** valga el máximo. Como esta función sólo acepta valores entre 0 y 1, se normaliza dividendo el valor actual de ***thicc*** por su valor máximo. Luego se comrpueba en qué estado se encuentra el sistema. Si la puntuación se encuentra por encima del umbral de peligro, y además el sistema predice en ese frame que el conductor está en estado de somnolencia, comenzará a sonar la alarma. En caso contrario, mandará la alarma a parar. Aquí se puede observar como, aunque la puntuación haya pasado el umbral, si el sistema no detecta somnolencia en ese frame, la alarma no sonará.

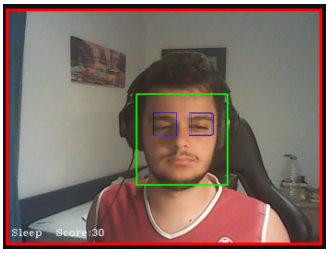
4.2.7. Muestreo de los resultados

El principal elemento que nos servirá para mostrar los resultados, será el propio vídeo. En este se podrá ver un recuadro alrededor de lo que el sistema detecta como el rostro y los ojos del conductor, lo que servirá para saber si se está seleccionando las áreas correctas. También se muestra un recuadro rojo que irá aumentando de tamaño junto con la variable ***thicc***, por lo que servirá como indicador de peligro en caso de no disponer de dispositivo de audio para la alarma o como refuerzo a la misma.

La imagen también mostrará un texto con la puntuación y cuál ha sido la clasificación del modelo respecto al frame mostrado.



(a) Frame con los ojos abiertos.



(b) Frame con los ojos cerrados.

Figura 4.25: Frame con el resultado final.

Para ello se coge el frame original y se escribe sobre el utilizando las herramientas proporcionadas por OpenCV.

```
cv.putText(frame, label, (10, frame.shape[0] - 20), cv.FONT_HERSHEY_COMPLEX_SMALL, 1, (255, 255, 255),
    1, cv.LINE_AA)
cv.rectangle(frame, (0,0), (frame.shape[1]-1, frame.shape[0]-1), (0,0,255), thicc)
cv.putText(frame, 'Score:'+str(score), (100, frame.shape[0]-20), cv.FONT_HERSHEY_COMPLEX_SMALL, 1,
    (255, 255, 255), 1, cv.LINE_AA)
imagePlaceHolder.image(cv.cvtColor(frame, cv.COLOR_BGR2RGB), caption='Video')
```

Figura 4.26: Código de muestreo del frame.

En el código anterior, se puede ver como se escribe en el frame el recuadro la etiqueta de la predicción, el recuadro de peligrosidad y la puntuación respectivamente. Y luego se utiliza la utilidad de streamlit para mostrarla.

Los recuadros de los ojos y el rostro no se muestran aquí pues el proceso de pintado del recuadro se realiza en la clasificación de los elementos mostrada con anterioridad. Como formas secundarias de feedback, se escribe el valor de las variables y la clasificación en la aplicación. Para que así el usuario pueda tener toda la información a mano y ver las modificaciones que él mismo ha realizado a las variables y los resultados que se han ido obteniendo.

Esto es muy útil, puesto que en la imagen se puede mostrar que el conductor se encuentra dormido. Pero no especifica si es por tener los ojos cerrados o por estar bostezando, o en caso de tener los ojos abiertos no indica si se tienen los dos o solo uno, ni cuál de ellos es el abierto.

Para ello se utilizan estos cuadros de texto, para mostrar toda la información que no se puede mostrar en la imagen pues la saturaría.

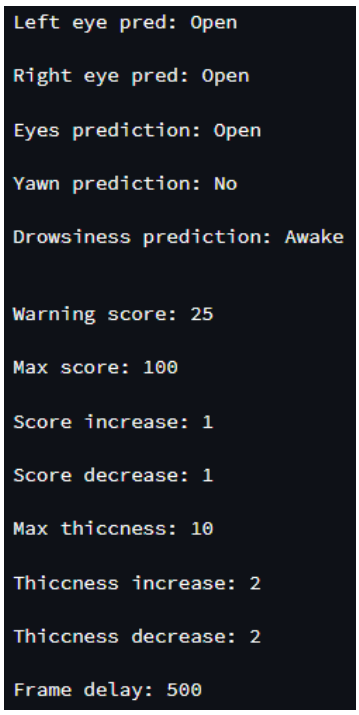


Figura 4.27: Resultado en formato texto.

Como se puede observar en la imagen, hay dos secciones: la superior y la inferior.

Sección superior: Aquí se muestran los resultados de la clasificación y la predicción de los elementos, los cuales son por orden:

- **Left eye pred:** Clasificación del ojo izquierdo.
- **Right eye pred:** Clasificación del ojo derecho.
- **Eyes prediction:** Predicción sobre el conjunto de los ojos.
- **Yawn prediction:** Clasificación del rostro sobre el bostezo.
- **Drowsiness prediction:** Predicción sobre el estado de somnolencia del conductor.

Sección inferior: Aquí se muestran las variables y parámetros que puede modificar el usuario.

- **Warning score:** Umbral de puntuación a partir del cuál se empieza a considerar peligroso el estado del conductor.
- **Max score:** Puntuación máxima que puede alcanzar la variable score.
- **Score increase:** Factor de crecimiento de la puntuación cuando se esté en estado de somnolencia.

- **Score decrease:** Factor de decrecimiento de la puntuación cuando no se esté en estado de somnolencia.
- **Max thiccness:** Máximo valor que puede alcanzar el ancho de peligrosidad que controla el volumen de la alarma y el recuadro de peligro.
- **Thiccness increase:** Factor de crecimiento del ancho de peligrosidad cuando se esté en estado de somnolencia y se supere el umbral de puntuación.
- **Thiccness decrease:** Factor de decrecimiento del ancho de peligrosidad cuando no se esté en estado de somnolencia.
- **Frame delay:** Tiempo de espera en ms entre captura y captura de frames..

Toda esta es la información directa que ofrece el sistema y las variables que se pueden utilizar para modificar el mismo. Sin embargo, también hay otra información más global pero que puede servir también de utilidad. Esta consiste en un conteo de la cantidad de frames en los que se ha detectado somnolencia frente en los que no.

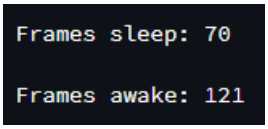


Figura 4.28: Cantidad de frames despierto y dormido.

Esto puede ser utilizado para analizar la cantidad de tiempo que el conductor ha permanecido en estado de somnolencia frente al tiempo total que lleva el sistema activo.

Todos estos datos anteriores, se muestran utilizando la herramienta empty() de streamlit. Esta permite reservar un espacio dentro de la app que luego se podrá rellenar con el contenido que se requiera más adelante.

Se crea un elemento empty() para cada uno de los datos que se planean mostrar más adelante. Esto se realiza antes de iniciar la captura de los frames para así ya tener los elementos preparados para escribir en ellos. Destacar que, aunque los elementos ya se encuentren en la app, estos no muestran ningún tipo de información al inicio y por lo tanto parecerá como si no hubiese nada hasta que se escriba en ellos.

Una vez que se han inicializado los elementos empty() y se ha leído el frame y realizado todas las operaciones de clasificación y predicción, se inicializa los elementos empty asignándoles los valores correspondientes.

```
leftEyeTagText.text(f"Left eye pred: {leftEyeTag}")
rightEyeTagText.text(f"Right eye pred: {rightEyeTag}")
eyesTagText.text(f"Eyes prediction: {eyesTag}")
faceTagText.text(f"Yawn prediction: {yawnTag}")
labelText.text(f"Drowsiness prediction: {label}")
spaceText.text(f"")
warningScoreText.text(f"Warning score: {warningScore}")
maxScoreText.text(f"Max score: {maxScore}")
scoreIncreaseText.text(f"Score increase: {scoreIncrease}")
scoreDecreaseText.text(f"Score decrease: {scoreDecrease}")
maxThiccText.text(f"Max thiccness: {maxThicc}")
thiccIncreaseText.text(f"Thiccness increase: {thiccIncrease}")
thiccDecreaseText.text(f"Thiccness decrease: {thiccDecrease}")
framesSleepText.text(f"Frames sleep: {framesSleep}")
framesAwakeText.text(f"Frames awake: {framesAwake}")
frameDelayText.text(f"Frame delay: {frameDelay}")
```

Figura 4.29: Establecer valores en st.empty.

Este es el último paso que se realiza dentro del bucle while de captura y análisis de los frames. Fuera de este tan solo quedan las funciones de salida y algún elemento de streamlit.

4.2.8. Funciones de salida y control

En cuanto al control de la ejecución, tan solo se ha establecido un pequeño delay entre frame y frame que puede ser aumentado para modificar la velocidad a la que son obtenidos los frames. Para ello se ha utilizado la función de OpenCV waitKey() a la que se le establece un tiempo en ms. Esta función fue diseñada para esperar a que el usuario pulse una tecla y establece el tiempo que se espera para dicha acción. Pero sirve a modo de retardo dentro del programa, ya que las imágenes se muestran en streamlit y esta función tan solo es capaz de leer la entrada de teclado si hay alguna ventana de OpenCV activa, lo cuál no es el caso.

```
cv.waitKey(frameDelay)
```

Figura 4.30: Función de retardo.

Para el control de la salida el programa se ha creado un botón exit mediante las herramientas de streamlit, que nos permita finalizar la ejecución.

```
exitButton: st.button = st.button(label="Exit", on_click=exit_sys)
```

Figura 4.31: Código del botón de salida.

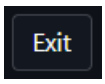


Figura 4.32: Botón de salida.

Al pulsar este botón, se llama automáticamente a una función que se encarga de cerrar la cámara web y parar la ejecución de streamlit.

```
def exit_sys():
    cap.release()
    st.stop()
```

Figura 4.33: Función de salida.

4.2.9. Componentes de streamlit

El despliegue de la interfaz gráfica del sistema, se ha desarrollado completamente con streamlit. Esta herramienta, como ya se menciona en apartados anteriores, es una librería de python enfocada en el despliegue web de aplicaciones de inteligencia artificial, ciencia de datos, etc.

Se pueden dividir los componentes en dos tipo, salida y entrada de datos:

- **Salida de datos:** Estos son los componentes que se utilizan para mostrar la información que obtiene y procesa el sistema.

  Son todos los elementos de streamlit ya mencionados con anterioridad en este capítulo, por lo que no se reincidirá en ellos.

- **Entrada de datos:** Estos componentes son los que podrá utilizar el usuario para manejar las variables del sistema y realizar las distintas pruebas con el sistema.

  Como todos las variables que puede modificar el usuario son números, se ha optado por utilizar una slidebar que permita elegir un número dentro de un rango predefinido. Streamlit también permite crear entradas para introducir números directamente o elementos que permiten aumentar o disminuir el valor con un botón como los number input. Pero se ha elegido esta opción entre todas las demás porque es la que mejor muestra el rango en el que ese puede mover el usuario y en qué posición está actualmente dentro de ese rango.

  En la imagen siguiente, se pueden encontrar todas las variables que se habían mencionado en apartados anteriores y sus sliders para seleccionar el valor.

Figura 4.34: User inputs.

En cada uno de los sliders se pueden observar distintos componentes. El primero sería el nombre del elemento que se va a modificar, seguido por el slider, que es la barra roja que se podrá desplazar para modificar los valores. Por último, en la parte inferior, hay dos números a izquierda y derecha, que corresponden al valor mínimo y máximo que puede tomar esa variable respectivamente.

El primer detalle que resalta, es que el slider, no se inicializa directamente en streamlit, si no que se crea en una barra lateral, sidebar. Todos los sliders reciben 4 valores. El primero es el nombre del slider que representa la variable a la que referencia, el segundo es el valor mínimo que puede alcanzar ese slider, seguido por el valor máximo y por último un valor por defecto.

Los valores mínimo, máximo y por defecto se han establecido tras

realizar varias pruebas con el sistema modificando dichos valores, y se ha definido que esos valores son los óptimos.

Adicionalmente se agregó una característica que permitiese al usuario subir sus propios modelos y ejecutarlos, para así poder realizar sus propias pruebas.

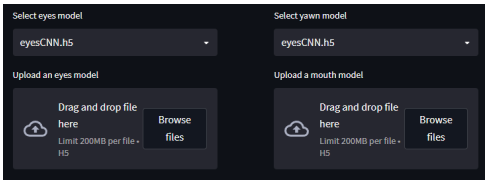


Figura 4.35: Subida y elección de modelos.

Una vez que se ha terminado de explicar cada uno de los componentes de streamlit, lo mejor sería mostrar una vista global del sistema para poder identificar cada uno de los elementos dentro de este.

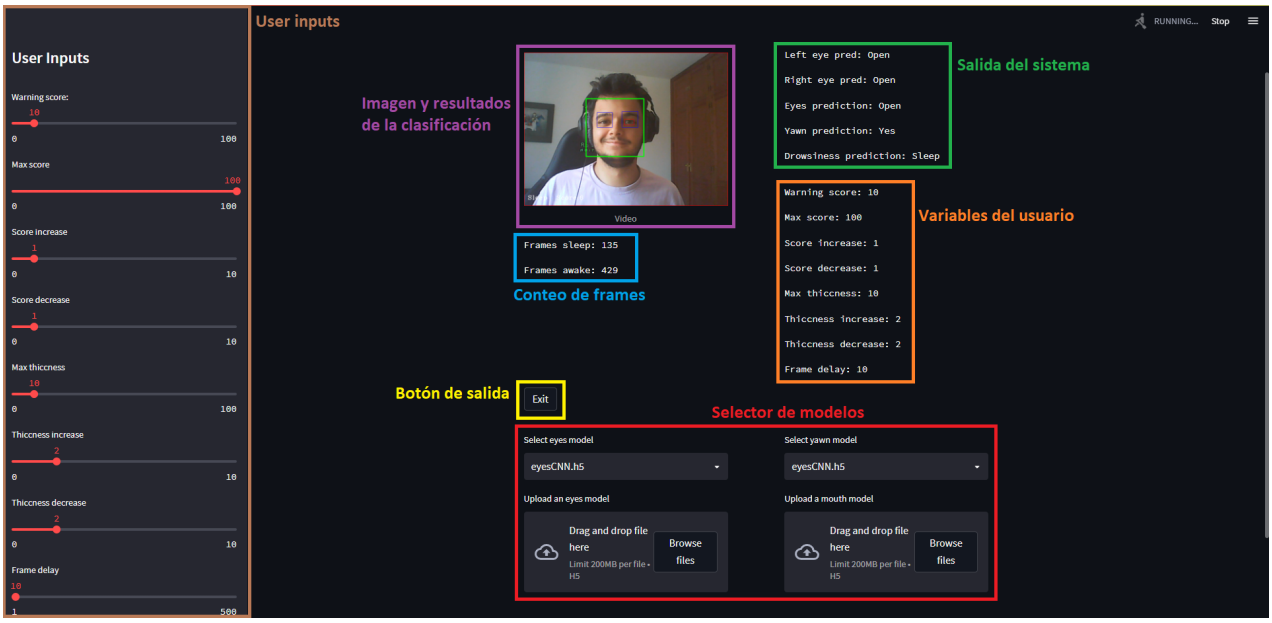


Figura 4.36: Elementos de streamlit

4.3. Corrección de problemas

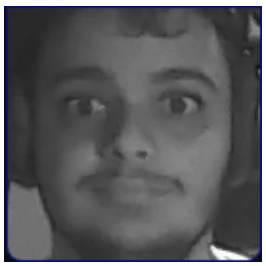
Sin embargo, tras entrenar los modelos y realizar varias pruebas, surgieron algunos problemas. El modelo de los ojos era bastante fiable y tenía una tasa de acierto muy buena, sin embargo, el modelo de detección de bostezos era poco fiable y solía fallar mucho pese a haber superado la fase de entrenamiento.

4.3.1. Corrección de los datasets

Tras realizar varias pruebas más sobre el modelo y analizar las situaciones en las que más fallaba el modelo, no se encontró ninguna respuesta clara a por qué no acertaba con sus predicciones. Lo que sugería que tal vez el problema no estaba en el modelo si no en el dataset, y, efectivamente, así fue.

El proceso de obtención de la imagen y la selección del área que se pasará al modelo se explica más adelante, pero básicamente se selecciona el rostro del conductor dentro de la imagen, se recorta y se envía ese pequeño fragmento. Es decir, se envía una subimagen que contiene únicamente el rostro del conductor. Sin embargo, las imágenes del dataset no son de rostros, si no de personas conduciendo donde se puede ver gran parte del entorno. Además, estas imágenes estaban tomadas con un ángulo lateral, mientras que en nuestro sistema, generalmente, las imágenes serán tomadas frontalmente.

(a) Imagen del dataset.



(b) Entrada esperada.

Figura 4.37: Ejemplos de los nuevos datasets.

Una vez encontrado el problema, había dos soluciones, modificar el modo de obtención de imágenes para ajustarse al dataset, o buscar otro dataset que se ajustase mejor a nuestras necesidades. Se optó por buscar otro dataset que mejor se adaptara a las necesidades del sistema.

Realizando una búsqueda de datasets de bostezos que pudiésemos usar bajo licencia, se encontró uno muy interesante, pero que no trabajaba con el rostro de las personas, si no con con la boca.

Al encontrar este dataset, nos dimos cuenta de otro problema con el tamaño de las imágenes. A la hora de pasarle las imágenes al modelo, estas se redimensionan para que todas tengan el mismo tamaño, 24x24 como se menciona en el apartado anterior. Esta era una medida enfocada a los ojos, que ya de por si eran imágenes pequeñas y no perdían mucha calidad

en el redimensionado. Sin embargo, las imágenes de rostros son bastante más grandes que las de un ojo, con muchos más elementos, y si perdían bastante información al reducir tanto su tamaño.

Por suerte, este nuevo dataset encontrado, al trabajar con la boca de las personas, trabajaba con imágenes mucho más pequeñas, de dimensiones similares a las de los ojos. Además es un dataset mucho mayor que el anterior, unas 5.047 imágenes, frente a las 1448 imágenes del anterior. Incluyendo imágenes de lado y de frente de bocas bostezando y sin bostezar.

Además, mientras se realizaba esta búsqueda del nuevo dataset de bostezos, se encontró un dataset aún mayor de ojos, los cuales parecían estar sacados con cámaras infrarrojas en la oscuridad, lo que sirvió para aumentar el dataset de los ojos aún más, pasando de 1.449 imágenes a 126.730 imágenes. Un cambio muy significativo.

(a) Ejemplo de ojo.

(b) Ejemplo de boca.

Figura 4.38: Ejemplos de los nuevos datasets.

Esto solucionaba el problema del tamaño y del perfil, ahora tan solo había que adecuar el modelo para ello.

4.3.2. Adecuación al nuevo dataset

La adecuación del sistema al nuevo modelo es un paso relativamente sencillo. Lo primero era ser capaces de obtener el recuadro de la boca del frame capturado. Para ello se usa un fichero .xml de opencv igual que para el resto de elementos, pero a diferencia de los otros, este no se encuentra en el repositorio de OpenCV, así que hubo que descargarlo y añadirlo localmente.

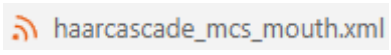


Figura 4.39: Xml local para la boca.

Después se crea otro cascade classifier y se añade a los demás.

```
faceCascade = cv.CascadeClassifier(f"{cv.data.haarcascades}haarcascade_frontalface_alt2.xml")
leftEyeCascade = cv.CascadeClassifier(f"{cv.data.haarcascades}haarcascade_lefteye_2splits.xml")
rightEyeCascade = cv.CascadeClassifier(f"{cv.data.haarcascades}haarcascade_righteye_2splits.xml")
mouthCascade = cv.CascadeClassifier(f"haarcascade_mcs_mouth.xml")
```

Figura 4.40: Cascade classifier con modelo de boca.

Por último, se envía la imagen a este clasificador para que nos devuelva la posición de la boca en el frame.

```
faceShape = faceCascade.detectMultiScale(grayFrame, minNeighbors=5, scaleFactor=1.1, minSize=(25,25))[:1]
leftEyeShape = leftEyeCascade.detectMultiScale(grayFrame, minNeighbors=4, scaleFactor=1.1)[:1]
rightEyeShape =  rightEyeCascade.detectMultiScale(grayFrame, minNeighbors=4, scaleFactor=1.1)[:1]
mouthShape =  mouthCascade.detectMultiScale(grayFrame, minNeighbors=4, scaleFactor=1.1)[:1]
```

Figura 4.41: Cascade classifier de la boca.

En el resto del sistema lo único que se cambiará será el envío del frame del rostro por el frame de la boca al modelo.

En la figura 4.49 también se puede observar otro cambio. Se ha añadido el código [:1] a la clasificación de los elementos del frame. Esto es una medida a la detección de múltiples de estos elementos. Estos clasificadores son muy potentes y capaces de detectar muchos rostros, ojos o bocas en una misma imagen, pero a veces pueden confundirse y detectar objetos que no son por sobras en la imagen, llevando a detectar múltiples rostros o bocas equivocadamente. En este sistema tan solo se va a analizar un conductor, por lo que es imposible detectar múltiples rostros o demás elementos.

Esa pieza extra de código lo que hace es eliminar todos los elementos del vector menos el primero, quedándonos así con tan solo un rostro. Sin embargo ¿Por qué quedarnos con el primero y no cualquier otro? ¿No podría ser este el equivocado? Efectivamente podría serlo, sin embargo, estos modelos clasifican los elementos y los ordenan según la certeza que tienen de estar acertados, así pues, el primero siempre tendrá más probabilidades que el resto de elementos de ser el elemento deseado.

- Capítulo 5 -

Informe técnico y análisis de los resultados

Una vez que se finalizó el desarrollo del sistema, se procedió a realizar un informe técnico sobre el mismo.

Este informe técnico puede obtenerse en el siguiente enlace:

https://drive.google.com/file/d/1cP2W7vA8dcq6yblnfNBOg6qJcTIeVKVy/view?usp=sharing

5.1. Ventajas y desventajas

Tras haber realizado el informe técnico posterior al desarrollo de sistema y a su estudio, se realizó un análisis de los resultados y se concluyó que el sistema tenía las siguientes ventajas y desventajas:

5.1.1. Ventajas

Fiabilidad

La principal ventaja del sistema es su fiabilidad. El sistema tiene una tasa de acierto entre el 99 % y el 96 %. Unos valores bastante altos y buenos. Lo que permite obtener resultados muy fiables y detectar de manera eficaz la somnolencia en el conductor.

Portabilidad

Otra de sus ventajas es la portabilidad. Este sistema, al estar construido en python, puede funcionar en casi cualquier SS.OO. siempre que este soporte python, tenga un navegador web y acceso a una webcam.

Implementabilidad

Otra de sus principales ventajas es la posibilidad de exportar los modelos a otros lenguajes como C/C++ gracias a las utilidades de Keras. Y el código es fácilmente reproducible en este lenguaje ya que se utilizan librerías como OpenCV que también están disponibles. Esto hace que este sistema

se pueda implementar con facilidad en microprocesadores que se puedan incorporar en el coche.

Diversidad

El sistema se ha entrenado con ojos y boca de miles de personas de todo el mundo, por lo que es capaz de identificar la fatiga en conductores de todas las etnias, por lo que podría ser implementado en cualquier lugar del mundo y seguir funcionando con la misma fiabilidad.

Funcionamiento nocturno

Este sistema ha sido entrenado para trabajar con imágenes obtenidas por la noche con cámaras infrarrojas, por lo que es un sistema que no necesita de luz ambiente ni de ningún foco que deslumbre al conductor para funcionar.

5.1.2. Desventajas

Luminosidad

En caso de no disponer de una cámara de infrarrojos o similar, el sistema es incapaz de detectar correctamente los elementos que componen la imagen, es decir, puede confundir la boca con un ojo o establecer un elemento del fondo como el rostro del conductor. Y si el sistema es incapaz de pasarle los elementos correctos a los modelos, estos no son capaces de realizar predicciones fiables.

Es por ello que, en caso de no tener cámara infrarroja o similares, el sistema debe trabajar en espacios de luz bien iluminados.

Individualidad

El sistema tan solo es capaz de trabajar con un único individuo, por lo que, en caso de querer trabajar con múltiples usuarios, el sistema no sería capaz. También influye en que, si un rostro que no sea el del usuario, se cuela en la imagen, el sistema no tiene forma alguna de elegir cuál es el rostro del usuario que debe escoger, y por lo tanto puede equivocarse y escoger el equivocado.

- Capítulo 6 -

Conclusiones y líneas futuras

6.1. Conclusiones

El principal objetivo de este TFG era el desarrollo de un sistema detección de somnolencia capaz de alertar a conductores durante la conducción, y el estudio y análisis posterior de las métricas y parámetros utilizados para encontrar la solución óptima.

Tras desarrollar el sistema y realizar las pruebas pertinentes, hemos podido comprobar que dicho sistema funciona satisfactoriamente. Tras haber concluido este desarrollo, poder detectar la somnolencia en los conductores de manera eficaz se vuelve una realidad. Esto supone un avance muy importante en la seguridad en la conducción. Pues como se vio en la introducción, la somnolencia produce entre el 15 % y el 30 % de los accidentes de tráfico en España. Y poder desarrollar un sistema que ayude a reducir estos porcentajes, supone un gran avance social muy grande.

Sin embargo, no todo es un camino de rosas. Aunque este sistema ha demostrado su utilidad y eficacia, también ha demostrado que queda un largo camino por delante para lograr que este sea una medida realmente eficaz en la conducción. Pero, a pesar de ello, no nos desanimamos. Hemos dado el primer paso hacia este nuevo futuro que abre una abanico de incontables oportunidades en el mundo de los sistemas de seguridad, no solo viales, si no en cualquier ámbito que se nos ocurra.

Y demostrar que nosotros, tan solo unos estudiantes, somos capaces de crear y desarrollar estos sistemas tan importantes para nuestro futuro, no hace más que llenarme de orgullo y esperanza.

6.2. Líneas futuras

Este proyecto es capaz de ser desarrollado en el futuro siguiendo dos líneas completamente distintas.

6.2.1. Desarrollo del sistema de detección de somnolencia

En esta posible línea futura, el trabajo se centraría en implementar este sistema en algún tipo de microprocesador que se pueda incorporar en los vehículos y no necesita de un gran hardware para funcionar. Para esto sería necesario implementar este sistema en un lenguaje como C/C++, pero esto es una tarea sencilla pus keras permite exportar sus modelos a C/C++ y la librería usada para la obtención y tratamiento de imágenes, OpenCV, también se encuentra disponible en ese lenguaje.

Además, a parte de incorporar este sistema en un microprocesador, sería muy interesante incorporar otras tecnologías como el control de cabeceo, o el control de los movimientos del volante. Si se juntan todos estos sistemas al ya desarrollado, se podría lograr un sistema de detección de somnolencia realmente eficaz en cualquier entorno.

6.2.2. Desarrollo del sistema de prueba del modelo

La otra línea de desarrollo que se podría seguir, consiste en mejorar y ampliar el sistema de prueba de este modelo.

Por ejemplo, se podría añadir la opción para que el usuario añada sus propios modelos y pruebe con ellos, o que el propio usuario pueda crear y entrenar sus modelos desde la interfaz usando el dataset ya disponible en este sistema.

Además se podrían añadir varias operaciones sobre imágenes como la mejora del contraste o brillo y gamma, y que el usuario pudiese experimentar con estas operaciones y sus valores, a demás de con los ya existentes, para encontrar el ajuste idóneo para su modelo.

Como se puede observar, el futuro de este proyecto es muy amplio y diverso. Lo que brinda muchas oportunidades y una capacidad de desarrollo enorme.

- Capítulo 7 -

Summary and Conclusions

7.1. Conclusions

The main objective of this TFG was the development of a drowsiness detection system capable of alerting drivers while driving, and the subsequent study and analysis of the metrics and parameters used to find the optimal solution.

After developing the system and carrying out the pertinent tests, we were able to verify that this system works satisfactorily. After completing this development, being able to detect drowsiness in drivers effectively becomes a reality. This represents a very important advance in driving safety. Becouse, as was seen in the introduction, drowsiness produces between 15 % and 30 % of traffic accidents in Spain. And being able to develop a system that helps reduce these percentages represents a great social advance.

However, not everything is a bed of roses. Although this system has proven its usefulness and effectiveness, it has also shown that there is a long way to go to make it a truly effective driving measure. But despite this, we are not discouraged. We have taken the first step towards this new future that opens up a range of countless opportunities in the world of security systems, not only for roads, but in any area that comes to mind.

And showing that we, just a few students, are capable of creating and developing these systems that are so important for our future, only fills me with pride and hope.

7.2. Future lines

This project is capable of being developed in the future following two completely different lines.

7.2.1. Development of the drowsiness detection system

In this possible future line, the work would focus on implementing this system in some type of microprocessor that can be incorporated into vehicles and does not require large hardware to work. For this it would be necessary to implement this system in a language like C/C++, but this is a simple task , becouse keras allows you to export your models to C/C++ and the library used for obtaining and processing images, OpenCV, is also available at that language.

Furthermore, apart from incorporating this system into a microprocessor, it would be very interesting to incorporate other technologies such as pitch control, or steering wheel movement control. If all these systems are added to the one already developed, a truly effective drowsiness detection system could be achieved in any environment.

7.2.2. Model test system development

The other line of development that could be followed consists of improving and expanding the test system for this model.

For example, the option could be added for the user to add their own models and test with them, or for the user to create and train their models from the interface using the dataset already available in this system.

In addition, several operations on images could be added, such as the improvement of contrast or brightness and gamma, and the user could experiment with these operations and their values, in addition to the existing ones, to find the ideal adjustment for their model.

As can be seen, the future of this project is very broad and diverse. Which provides many opportunities and enormous development capacity.

- Capítulo 8 -

Presupuesto

En la elaboración de este trabajo, están incluidos los siguientes costes:

8.1. Costes materiales

Elementos	Coste
NVIDIA GTX 1060 6GB	289,89€
Intel i5-12600K 3.70 GHz	388,49€
Memoria RAM + SSD	88,95€
Placa base	86,99€
Carcasa y periféricos	201,70€
Total	**1056,02€**

Tabla 8.1: Costes materiales

8.2. Costes de horas de trabajo

Tipos	Horas	Coste	Total
Investigación	55	35	1925€
Desarrollo de los modelos	65	30	1950€
Desarrollo del preprocesamiento de las imágenes	70	35	2450€
Testeo del sistema	80	32	2560€
Documentación	30	30	900€
Total	**300**	**-**	**9785€**

Tabla 8.2: Costes de horas de trabajo

Bibliografía

[1] Ethem Alpaydin. Machine learning. 2021.

[2] Fernando Izaurieta and Carlos Saavedra. Redes neuronales artificiales. *Departamento de Física, Universidad de Concepción Chile*, 2000.

[3] F. Berzal. Redes neuronales and deep learning. 2018.

[4] Conducir con sueño o cansancio. hhttps://www.dgt.es/muevete-con-seguridad/evita-conductas-de-riesgo/Conducir-con-sueno-o-cansancio.

[5] A Dorta, L García, JR González, C León, and C Rodríguez. Aproximación paralela a la técnica divide y vencerás. 2004.

[6] Natalia Alzate González. *Influencia de la variación de parámetros del sistema de potencia en la localización de fallas con métodos de clasificación*. PhD thesis, Universidad Tecnológica de Pereira. Facultad de Ingenierías Eléctrica . . . , 2013.

[7] Piloto automático de tesla. https://www.tesla.com/es_ES/autopilot.

[8] Sistemas adas. https://www.fundacionmapfre.org/educacion-divulgacion/seguridad-vial/sistemas-adas/.

[9] Sistemas adas - dgt. https://revista.dgt.es/es/motor/tecnologia-seguridad/2021/0317-Como-funciona-ADA-detector-fatiga.shtml.

[10] Alan T Norman. *Aprendizaje automático en acción*. Litres, 2019.

[11] Python 3.10. https://www.python.org/.

[12] Librería numpy. https://pypi.org/project/numpy/.

[13] Librería streamlit. https://pypi.org/project/streamlit/.

[14] Librería tensorflow. https://pypi.org/project/tensorflow/.

[15] Librería opencv-python. https://pypi.org/project/opencv-python/.

[16] Librería scipy. https://pypi.org/project/scipy/.

[17] Librería pygame. https://pypi.org/project/pygame/.

[18] vscode. https://code.visualstudio.com/.

[19] vscode-python. https://marketplace.visualstudio.com/items?itemName=ms-python.python.

[20] vscode-github. https://marketplace.visualstudio.com/items?itemName=GitHub.vscode-pull-request-github.

[21] vscode-latex. https://marketplace.visualstudio.com/items?itemName=mathematic.vscode-latex.

[22] Google drive. https://www.google.com/drive/.

[23] Github. https://github.com/.

[24] Git. https://git-scm.com/.

[25] Overleaf. https://www.overleaf.com//.

[26] Tipos de redes neuronales. https://inteligencia-artificial.dev/tipos-redes-neuronales/.

[27] Perceptron multicapa. https://es.wikipedia.org/wiki/Perceptrón_multicapa.

[28] Backpropagation. https://es.wikipedia.org/wiki/Propagación_hacia_atrás.

[29] Función sigmodia. https://en.wikipedia.org/wiki/Sigmoid_function.

[30] Función relu. https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[31] Aprendizaje supervisado. https://es.wikipedia.org/wiki/Aprendizaje_supervisado.

[32] Cuadros medios. https://en.wikipedia.org/wiki/Least_mean_squares_filter.

[33] Convolución. https://en.wikipedia.org/wiki/Convolution.

[34] Producto escalar. https://en.wikipedia.org/wiki/Dot_product.

[35] Producto interno de frobenius. https://en.wikipedia.org/wiki/Frobenius_inner_product.

[36] Sobreajuste. https://en.wikipedia.org/wiki/Overfitting.

[37] Keras. https://www.tensorflow.org/tutorials/images/classification.

[38] Dataflair. https://data-flair.training/blogs/python-project-driver-drowsiness-detection-system/.

[39] Videocapture. https://docs.opencv.org/4.x/d8/dfe/classcv_1_1VideoCapture.html.