Diseño de un sistema de detección de somnolencia utilizando redes neuronales.

Nicolás Lenis Sánchez y Omar Alfonso Galvis Camaron

Director

Jaime Guillermo Barrero Pérez

Magíster en Potencia Eléctrica

Universidad Industrial de Santander Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Ingeniería Electrónica

Bucaramanga

2024

Dedicatoria

En la finalización de mi etapa académica de pregrado para convertirme en ingeniero electrónico, representa el cumplimiento de una meta que es el conjunto de esfuerzo, resiliencia y perseverancia. Quiero dedicar este triunfo a los 3 pilares fundamentales de mi vida.

Para iniciar el agradecimiento a Dios quien me ha permitido estar con vida, guiando cada uno de mis pasos a lo largo de este recorrido brindándome la fortaleza necesaria, sin permitir desvíos en las metas que ya cumplí y las que me faltan. Segundo a mis padres quienes siempre han estado apoyándome incondicionalmente a lo largo de los desafíos, derrotas y triunfos, por medio de consejos y enseñanzas aportándome día a día conocimientos e ideas que construyeron la esencia, carácter y el ser humano que actualmente soy y por tercero y no menos importante mi tío Ilbar Sánchez quien desde la niñez ha sido mi mentor y apoyo para la programación. Agradezco inmensamente tanto a los mencionados anteriormente como a cada persona que en este trayecto de vida me brindo un poco de su tiempo, conocimiento y carisma, impulsando a la formación de un ser humano integro.

Nicolas Lenis Sánchez.

Primeramente, mi más profundo reconocimiento se eleva en gratitud hacia Dios, quién ha sido mi fuente de sabiduría y apoyo durante toda mi vida. Un homenaje eterno reside en mi corazón para mi querida madre, cuyo sueño de verme convertido en

profesional fue incluso más claro que el mío propio, y aunque no está físicamente presente, su visión y amor trascienden el tiempo y el espacio, querida madre, este logro es tuyo tanto como mío. Así mismo quiero dedicar este triunfo a mi familia, gracias por la fe incondicional que depositaron en mí, incluso cuando las circunstancias parecían adversas, su apoyo y amor constante fueron el combustible que alimentó mi determinación. A mis amigos más cercanos, compañeros de innumerables aventuras y desafíos, gracias por estar cuando la carga parecía demasiado pesada, su apoyo hizo de este viaje una experiencia más llevadera y divertida. Un reconocimiento especial a la Universidad Industrial de Santander, no solo por ser la institución que respaldó mi formación académica, sino también por ser un hogar que me acogió en sus hermosos espacios. Gracias por moldearme con rigor y humanidad, permitiéndome crecer no solo como profesional sino también como un ser humano íntegro. Este triunfo no es un logro solitario, sino el resultado de un constante trabajo en equipo y fe compartida. Gracias por ser parte de mi historia. Este proyecto de grado, y los frutos que de él surjan, están dedicados a cada uno de ustedes.

Omar Alfonso Galvis Camaron.

Agradecimientos

Quiero expresar un inmenso agradecimiento a mi alma mater, la UNIVERSIDAD INDUSTRIAL DE SANTANDER, escenario de mi crecimiento profesional y cuyas instalaciones han sido testigo de los momentos más importantes en el trayecto de mi formación. agradezco también a la empresa Colombiana de Petróleos Ecopetrol. Que financió parte de los gastos de mi educación y salud. A mis padres cuyo apoyo me motivó a no rendirme en los momentos más difíciles. a mis hermanos que estuvieron siempre al tanto de mis avances, en especial a mi hermana Carolina con la que siempre compartí conocimiento relacionado con proyectos y semilleros base de su trabajo como docente. una mención muy especial de agradecimiento a mis compañeros y amigos: Jonathan, Adriana Patricia y Natalia con quienes compartí conocimiento, amistad, dificultades y momentos felices. Por último y no menos importante, mi agradecimiento a Dios, que me guio, me dio salud, fortaleza y sabiduría para seguir adelante.

Nicolas Lenis Sánchez.

En este momento trascendental de mi vida, cuando miro hacia atrás en el viaje que ha sido mi carrera universitaria, mi corazón se desborda de gratitud. Primero y, antes que nada, quiero expresar mi más sincero agradecimiento a Dios por su bondad infinita y su presencia en cada desafío y en cada triunfo, mi guía durante todo mi proceso. No puedo dejar de mencionar la enorme deuda de gratitud que tengo con la Universidad Industrial de Santander. Esta prestigiosa institución no solo me ha brindado una educación de primer

nivel, sino que también ha sido un entorno en el que he crecido, aprendido y descubierto mi pasión. A cada profesor y personal administrativo, gracias por sembrar en mí las semillas del conocimiento y la integridad. Mi agradecimiento más especial lo ocupa mi madre, cuyo amor y sacrificios han sido la verdadera razón detrás de mi viaje que, a pesar de los innumerables obstáculos, nunca flaqueó en su convicción de que la educación era la inversión más valiosa. Su determinación inquebrantable y su fe en mis habilidades me sostuvieron cuando las dificultades financieras creaban una barrera en mi camino. A mis queridos amigos, ustedes son la familia que elegí. En los días sombríos, fueron mi luz, y en los días de alegría, fueron mi celebración. Su apoyo ha ido más allá de lo académico, la solidaridad y la comprensión que me brindaron convirtieron los retos en aventuras. A todos, les extiendo mi agradecimiento más profundo. Cada palabra de aliento, cada gesto de apoyo, han sido los hilos que tejieron este éxito. Continuaré adelante llevando las lecciones y los recuerdos que hemos compartido, sabiendo que mi gratitud va más allá de las palabras.

Omar Alfonso Galvis Camaron.

Tabla de Contenidos

Lista de figuras

Lista de tablas

Lista de apéndices

Resumen

Título: Diseño de un sistema de detección de somnolencia utilizando redes neuronales. *

Autores: Nicolás Lenis Sánchez, Omar Alfonso Galvis Camaron. **

Director: Jaime Guillermo Barrero Pérez. Mg. Ingeniería Eléctrica.

Palabras Clave: Detección, Somnolencia, Red neuronal, Deep learning, Algoritmo, Dataset, Procesamiento de imágenes, Prototipo, Raspberry pi 4

Descripción: La somnolencia puede ser especialmente peligrosa cuando se requiere concentración absoluta en momentos críticos del ser humano. Las debilidades momentáneas en el enfoque durante tareas esenciales, como la conducción o la manipulación de equipos complejos, pueden desencadenar consecuencias trágicas, incluidos accidentes graves o incluso pérdidas humanas. La clave para mitigar tales riesgos radica en la detección precoz de ésta. Este trabajo de grado presenta un método sofisticado que emplea inteligencia artificial y procesamiento de imágenes para reconocer los primeros signos de somnolencia. A través de un sistema embebido, la Raspberry pi 4, se implementaron dos soluciones, en la primera solución se implementó una red neuronal meticulosamente entrenada que identifica la somnolencia o cansancio mediante el tiempo del parpadeo. En la segunda solución se implementó un sistema con Dlib, el cual hace la medición del EAR, también con tiempo de parpadeo más bostezos prolongados.

---

*Trabajo de grado

** Facultad de Ingenierías Fisicomecánicas, Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director Jaime Guillermo Barrero Pérez.

Abstract

Title: Design of a drowsiness detection system using neural networks. *

Authors: Nicolás Lenis Sánchez, Omar Alfonso Galvis Camaron. **

Director: Jaime Guillermo Barrero Pérez. Mg. Electric engineering.

Keywords: Detection, Drowsiness, Neural network, Deep learning, Algorithm, Dataset, Image processing, Prototype, Raspberry pi 4

Description: Drowsiness can be especially dangerous when absolute concentration is required at critical human moments. Momentary weaknesses in focus during essential tasks, such as driving or handling complex equipment, can trigger tragic consequences, including serious accidents or even loss of life. The key to mitigating such risks lies in their early detection. This graduate work presents a sophisticated method that employs artificial intelligence and image processing to recognize the early signs of drowsiness. Through an embedded system, the Raspberry pi 4, two solutions were implemented, in the first solution a meticulously trained neural network was implemented that identifies drowsiness or tiredness through the blink time. In the second solution, a system was implemented with Dlib, which measures EAR, also with blink time plus prolonged yawning.

---

*Bachelor Thesis

** Faculty of Physical and Mechanical Engineering, School of Electrical, Electronic and Telecommunications Engi- neering. Director Jaime Guillermo Barrero Pérez.

1. Objetivo General

1.1 Objetivo General

Diseñar y evaluar un sistema de detección de somnolencia utilizando técnicas de procesamiento de imágenes y redes neuronales.

1.2 Objetivos Específicos

- Desarrollar una base de datos que permita realizar el entrenamiento para la detección de somnolencia.
- Seleccionar una arquitectura de red neuronal que junto con procesamiento de imágenes permita realizar la detección de somnolencia de forma rápida.
- Implementar el sistema seleccionado en un sistema embebido como una Raspberry pi 4 para la detección de somnolencia en tiempo real.

2. Somnolencia

La somnolencia se define como una pesadez y torpeza de los sentidos que son motivadas por el sueño, aunque también se puede definir como la tendencia o habilidad de transición de la vigilia al sueño y está determinada por la calidad de sueño, cantidad de sueño y el ritmo circadiano (Escobar, 2020) (Rosales Mayor, & Rey De Castro Mujica, 2010). El reloj biológico del ser humano regula el ciclo de dormir y despertar. El punto cuando se está más somnoliento es entre las 2 y 5 de la mañana y el siguiente es entre las 2 y 5 de la tarde. Existen dos tipos de somnolencia: La somnolencia objetiva y la subjetiva. La primera es la medición o evaluación externa del grado de somnolencia de una persona. Esto implica la observación de signos objetivos que indican la presencia de somnolencia, como la disminución del rendimiento cognitivo, la falta de atención, los microsueños (breves episodios de sueño involuntario), y otros indicadores medibles. La segunda se refiere a la percepción individual de la propia somnolencia. Es la sensación subjetiva de estar adormecido o con sueño, según la experiencia personal del individuo. (Rosales Mayor, & Rey De Castro Mujica, 2010) (Shen, Barbera, & Shapiro, 2004). Para poder reconocer la somnolencia se deben tener en cuenta los comportamientos del individuo, estos pueden ser el bostezo, tiempo de parpadeo, apertura de los ojos y movimiento de cabeza conocido como “cabeceo” (Rosales Mayor, & Rey De Castro Mujica, 2010).

El enfoque principal de este proyecto fue la evaluación de la somnolencia objetiva utilizando la medición del tiempo de parpadeo. Además, se empleó Dlib, un sistema

orientado a detectar la apertura de los ojos mediante la relación de aspecto del ojo (EAR) e integrando tambien una función de alerta para identificar bostezos prolongados.

La somnolencia puede tener efectos críticos en diversas áreas importantes de la vida humana, y este proyecto se enfoca especialmente en aquellas que son comúnmente afectadas, como el rendimiento académico durante el estudio y la seguridad en la conducción. Estas áreas son particularmente sensibles a la somnolencia, ya que puede comprometer la concentración, la atención y la toma de decisiones, aumentando así el riesgo de errores y accidentes.

2.1 Riesgos de la conducción

La atención de un conductor afectado por el cansancio se reduce gradualmente al volante y su capacidad de reacción ante situaciones que requieren respuestas rápidas disminuye mientras transita por las vías. El tiempo de parpadeo o quedarse dormido al conducir son ejemplos críticos de la disminución de la atención (Rey De Castro Mujica, 2013).

La conducción somnolienta es uno de los principales factores que contribuyen a los accidentes de tráfico, según la National Transportation safety board. En Europa, alrededor del 10-20% de todos los accidentes de tráfico parecen estar relacionados a la fatiga o somnolencia del conductor (System for effective Assessment of driver vigilance and Warning According to traffic risk Estimation). En los EE. UU., quedarse dormido mientras se conduce causa al menos 100.000 accidentes mortales. Hasta el 28% de los conductores

estadounidenses encuestados admiten haberse quedado dormidos al volante al menos una vez (Forsman, Vila, Short, Mott, & Van Dongen, 2013). Según un informe reciente, la Organización Mundial de la Salud de 2013 que afirma que: 1,24 millones de personas mueren en la carretera cada año; aproximadamente el 6 % de todos los accidentes son causados por conductores que conducen en estado de somnolencia; Y la mayoría de los accidentes de este tipo resultan en muertes.

A partir de la información proporcionada, resulta evidente la importancia de reconocer el riesgo y las posibles consecuencias peligrosas cuando un conductor, afectado por la fatiga, decide manejar en estado de somnolencia. Este escenario pone en peligro no solo la vida del conductor, sino también la de otros usuarios de la vía.

2.2. Problemas de desempeño

En un estudio llevado a cabo para evaluar la calidad del sueño y la presencia de somnolencia diurna excesiva entre estudiantes universitarios de diversas disciplinas en una universidad pública de Manizales durante el primer semestre de 2016, los investigadores examinaron a 258 mujeres (47,2%) y 289 hombres (65,8%). La edad promedio de las mujeres fue de 20,9 años con una desviación estándar de 2,7, mientras que la de los hombres fue de 22,9 años con una desviación estándar de 3,8. Contrariamente a hallazgos reportados en otros estudios, como el de Tlatoa-Ramírez, donde los hombres presentaban un riesgo cuatro veces mayor de experimentar somnolencia diurna excesiva, este estudio no reveló diferencias significativas entre los géneros. Se constató que el 77,1% de los

estudiantes muestra una calidad de sueño que requiere atención médica, y el 70,3% sufre de somnolencia diurna leve o moderada. Estos resultados concuerdan con una investigación realizada en Arabia Saudita, donde se encontró que, aunque los estudiantes de medicina dormían en promedio 5,8 horas por noche y se acostaban alrededor de la 1:53 a. m., al menos el 8,0% experimentaba somnolencia diurna, y el 30,0% reportaba una mala calidad de sueño. (Portilla, Dussán, Montoya, Taborda, & Nieto, 2019).

Es importante que las personas que presentan baja calidad del sueño o dificultades de SDE reciban atención y tratamiento médico puesto que diferentes estudios han demostrado la asociación entre la baja calidad del sueño con diferentes trastornos de salud mental tales como fallas cognitivas, atencionales y ejecutivas, dificultades emocionales relacionadas con ansiedad, depresión, irritabilidad, así como riesgo físico y cardiovascular entre otros (Portilla, Dussán, Montoya, Taborda, & Nieto, 2019).

Según estudio realizado en la universidad Tecnológica de Pereira sobre los estudiantes de medicina, se encontró una relación entre la baja calidad de sueño determinada por baja eficiencia y menor rendimiento académico al final del semestre, mientras que la buena calidad subjetiva de sueño se comportó como factor protector del desempeño (Duque, Echeverri, & Machado, 2015).

3. Selección estratégica para la resolución del problema

Este capítulo muestra el desarrollo y la implementación de tecnologías avanzadas para la identificación del estado de los ojos, específicamente centrado en determinar si están abiertos o cerrados, aplicaciones cruciales para la detección de la somnolencia y otras condiciones. Se investigó información para algoritmos de procesamiento de imágenes y aprendizaje automático, con cámaras que se pueden integrar tanto en computadoras como en sistemas embebidos. Se destacaron las técnicas de procesamiento de imágenes, el uso de redes neuronales para el reconocimiento de patrones y las innovaciones en hardware que permiten la captura y análisis en tiempo real del estado en tiempo real de los ojos.

3.1. Redes neuronales.

Estos algoritmos de inteligencia artificial, conocidos como redes CNN (Convolutional Neural Networks), encuentran su inspiración en la organización y funcionamiento del cerebro. Su objetivo principal se concentra en llevar a cabo eficientemente tareas de clasificación, dando lugar a un proceso de aprendizaje autónomo. Este aprendizaje adquirido se presenta como una habilidad adaptable y aplicable en diversas situaciones, brindando soluciones versátiles y eficaces en una amplia gama de sistemas (Belmar, 2021).

3.1.1. Redes CNN

Las Convolutional Neural Networks (CNN), en el contexto del Deep Learning, se presentan como un modelo arraigado en la capacidad de discernir patrones y características, destacándose, sobre todo, en la tarea de clasificación de imágenes. Su característica más distintiva radica en la sorprendente habilidad de mantener la invarianza; en términos simples, una CNN entrenada puede identificar patrones en cualquier ubicación de la imagen o marco de referencia. El funcionamiento de estas redes se desenvuelve en un proceso organizado y jerárquico. Navegando a través de diversas capas, cada una orientada a patrones específicos, las CNN desarrollan progresivamente jerarquías espaciales de patrones. En otras palabras, ciertas capas se especializan en la detección de patrones más pequeños, mientras que otras se dedican a patrones de mayor envergadura. Este proceso implica la extracción y fusión de características recopiladas de capas anteriores (Rangel, Ruiz, García, & Cervantes, 2019).

En términos sencillos, podríamos decir que las CNN realizan una exploración minuciosa de la información visual, aprendiendo desde características simples hasta aquellas de mayor complejidad a medida que avanzan a través de sus capas. Este enfoque escalonado les permite capturar la esencia de los patrones visuales de manera eficaz, otorgándoles una capacidad sólida y eficiente para llevar a cabo la clasificación de imágenes (Rangel, Ruiz, García, & Cervantes, 2019).

3.1.2. Transfer learning.

El Transfer learning es un método empleado en el Deep learning mejorar el rendimiento en una nueva tarea transfiriendo el conocimiento adquirido en una tarea previamente aprendida. Esta técnica consiste en aplicar los pesos de un modelo ya entrenados para una tarea específica a un nuevo modelo destinado a resolver una tarea diferente, pero que está relacionada el problema anterior. De esta manera, se aprovechan las soluciones desarrolladas para un problema anterior para abordar un nuevo desafío (Vrbancic, & Podgorelec, 2020).

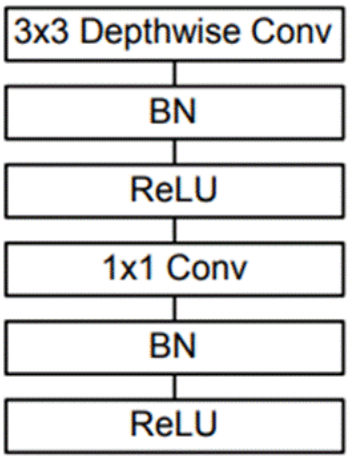
El Transfer learning resulta especialmente útil en tareas que requieren un extenso tiempo de entrenamiento de modelos. Mediante esta técnica, es posible enfocarse en entrenar únicamente las últimas capas del modelo, lo que representa un significativo beneficio en términos de complejidad temporal y reduce la necesidad de grandes conjuntos de datos. Esto hace que el proceso sea más eficiente y accesible, aprovechando el conocimiento previo para acelerar y mejorar el aprendizaje en nuevas tareas (Vrbancic, & Podgorelec, 2020).

3.1.2.1. Mobilenet.

Existen diversas arquitecturas de redes neuronales convolucionales, entre las cuales se incluyen modelos como VGG, ResNet y Inception, por mencionar algunas. No obstante, para este proyecto se eligió MobileNet debido a su eficiencia en términos de recursos computacionales y a su efectividad en el análisis en tiempo real.

El modelo MobileNet se basa en convoluciones separables en profundidad, que es una forma de convoluciones factorizadas que descomponen una convolución estándar en una convolución en profundidad y una convolución 1×1 llamada convolución de punto. En el caso de MobileNet, la convolución en profundidad aplica un solo filtro a cada canal de entrada. Luego, la convolución de punto aplica una convolución 1×1 para combinar las salidas de la convolución en profundidad. Mientras que una convolución estándar filtra y combina las entradas en un solo paso, la convolución separable en profundidad divide este proceso en dos capas separadas, una para filtrar y otra para combinar. Esta factorización tiene el efecto de reducir drásticamente el cálculo y el tamaño del modelo. Esta estructura particular está ilustrada y explicada con más detalle en la **Figura *1***. (Howard, Zhu, Chen, Kalenichenko, Wang, Weyand, Andreetto, & Hartwig, 2017).

Figura 1
Arquitectura MobileNet.



Nota: tomado de (Howard, Zhu, Chen, Kalenichenko, Wang, Weyand, Andreetto, & Hartwig, 2017).

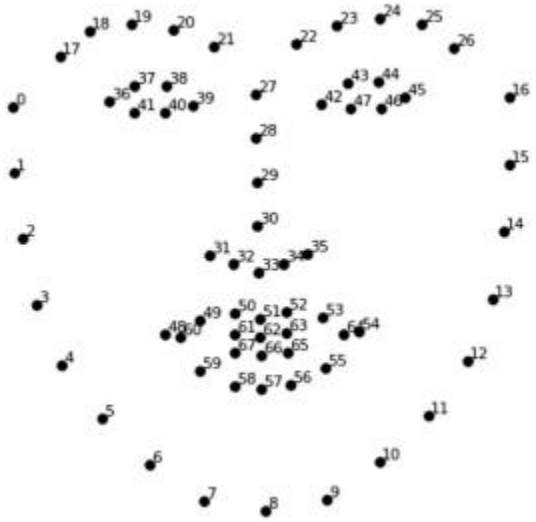
3.1.3. Dlib.

Dlib es una biblioteca de *Machine learning* que facilita la detección de rostros, la localización de puntos clave y el reconocimiento facial mediante la función frontal_face_detector(). Esta función específicamente emplea el método de Histogramas de Gradientes Orientados (HOG) para identificar rostros (Yang, & Fan, 2023).

3.1.3.1. Facial landmark

En este proyecto haremos uso del modelo pre-entrenado *face_landmark_detection.py* que ofrece Dlib. Este modelo se emplea para identificar la ubicación de 68 puntos de referencia específicos en el rostro, permitiendo una detección detallada de las características faciales como en la **Figura *2*** (Yang, & Fan, 2023).

Figura 2
Facial landmark con Dlib.

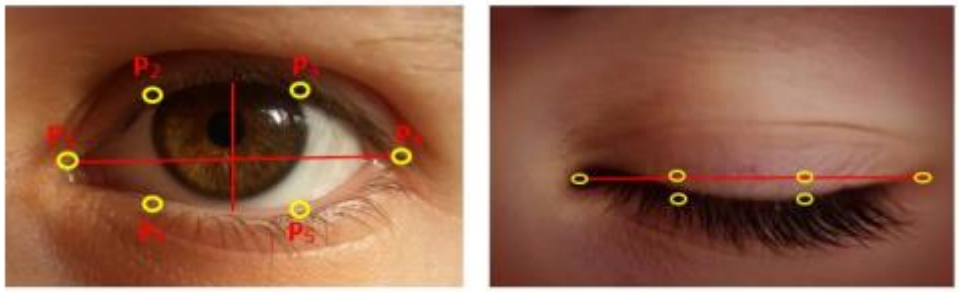


Nota: tomado de (Yang, & Fan, 2023).

3.1.3.2. Eye Aspect Ratio (EAR)

Utilizando los 68 puntos de referencia facial, es posible enfocarse exclusivamente en los puntos correspondientes a los ojos y la boca para realizar su detección y seguimiento. Mediante la medición de la distancia euclidiana entre los párpados, conocida como Eye Aspect Ratio (EAR), se puede determinar la apertura ocular ajustando la relación de aspecto en función de la anchura y altura del ojo según la **Figura *3*** (Yang, & Fan, 2023).

Figura 3
Eye Aspect Ratio (EAR).



Nota: tomado de (Yang, & Fan, 2023)

Ecuación 1
Fórmula del Eye Aspect Ratio (EAR).

$$EAR = \frac{\|P_2 - P_6\| + \|P_3 - P_5\|}{2\|P_1 - P_4\|}$$

Nota: tomado de (Yang, & Fan, 2023).

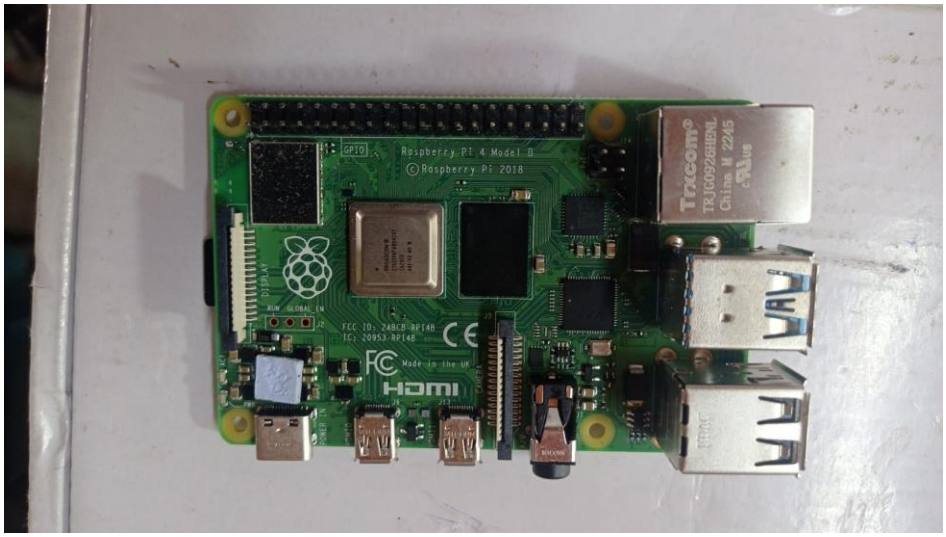
3.1.3.3. Open computer visión (OpenCV)

OpenCV es una biblioteca ampliamente utilizada en el campo de la visión por computadora y aprendizaje automático, aplicable en múltiples áreas incluyendo las ciencias de la vida. Esta biblioteca es reconocida por su extenso repertorio de más de 2500 algoritmos tanto clásicos como avanzados en visión artificial. Ofrece una amplia gama de funcionalidades, incluyendo el procesamiento de imágenes, detección de características, reconocimiento de objetos, aprendizaje automático y análisis de video (Dom, Heras, & Pascual, 2017).

3.2. Raspberry pi 4.

La Raspberry Pi se destaca actualmente como uno de los dispositivos más versátiles y prometedores en el campo del desarrollo de proyectos de electrónica recreativa, robótica y la comunidad en general. Además, desempeña un papel fundamental en el ámbito educativo, siendo una herramienta esencial para la enseñanza. La Raspberry Pi, junto con placas como Arduino, ha sido parte integral de la transformación del panorama tecnológico en los campos de la educación y la creación de prototipos para proyectos de electrónica y programación, **Figura *4***.

Figura 4
Raspberry pi 4.



4.Metodología.

Para avanzar en el proyecto de detección de somnolencia mediante el uso de redes neuronales y procesamiento de imágenes, se seleccionó y estructuró un conjunto de datos de acceso público proporcionado por la organización MRL (Media Research Lab, 2018). Este conjunto se organizó en subcarpetas y categorías específicas para poder realizar el proceso de entrenamiento. Se implementó el aprendizaje por transfer learning utilizando modelos ya pre-entrenados, lo cual incrementó notablemente la precisión del sistema. Adicionalmente, se exploró una alternativa utilizando la biblioteca de machine learning Dlib, reconocida por su capacidad de identificar 68 puntos característicos en el rostro, incluyendo áreas de gran importancia para este proyecto como lo son los ojos y la boca.

Para tener un buen rendimiento de las soluciones y basándonos en investigaciones previas, se optó por centrarse específicamente en el análisis de los ojos como indicador clave para la detección de somnolencia. Además, se agregó un análisis para la boca al sistema realizado con Dlib. Con este enfoque, las estrategias propuestas se dirigieron a medir y analizar la duración del cierre de los ojos, y en el caso con Dlib bostezos prolongados, entendiendo que estos son factores significativos para identificar signos tempranos de somnolencia.

4.1. Arquitectura de la base de datos.

Con el objetivo de detectar la somnolencia, se implementó un sistema basado en el análisis de la duración del parpadeo, tal como se mencionó anteriormente. Para llevar a cabo este análisis, se eligió una base de datos pública de la organización MRL, que incluye más de 70,000 imágenes de 37 personas distintas, clasificados y etiquetadas. Esta colección está compuesta por imágenes que representan estados de ojos abiertos y cerrados, proporcionando una amplia gama de datos para entrenar y validar nuestro sistema de detección la cantidad de imágenes en la **Tabla *1***.

A partir de esta colección de datos, se procedió a seleccionar y reestructurar el contenido, organizándolo en dos subcarpetas distintas: una para imágenes de ojos abiertos y otra para ojos cerrados, como se puede ver en la Tabla 1. Esta división facilita el acceso y la identificación de los diferentes estados oculares, permitiendo un proceso más eficiente

y estructurado tanto en la etapa de entrenamiento como en la validación del sistema de detección de somnolencia.

Tabla 1
Cantidad de imágenes.

Categorías	**Imágenes**
Ojos abiertos	38518
Ojos cerrados	35946

Para el proceso de entrenamiento, se seleccionó una parte de los datos disponibles, limitada por la capacidad de procesamiento de la máquina utilizada. Por lo tanto, se trabajó con un conjunto reducido de datos, cuyos detalles y cantidades específicas están detallados en la **Tabla *2***. Esta selección permitió optimizar los recursos computacionales disponibles y asegurar un entrenamiento efectivo dentro de las limitaciones existentes.

Tabla 2
Selección de imágenes.

Subconjunto	**Categoría**	**Imágenes**
Entrenamiento	Ojos abiertos	900
	Ojos cerrados	900
Validación	Ojos abiertos	100
	Ojos cerrados	100

4.2 Entrenamiento de la red neuronal con mobilenet

Para iniciar el proceso de entrenamiento de la red neuronal destinada al proyecto de detección de somnolencia, fue esencial realizar un estudio detallado sobre la arquitectura de red más adecuada. Basándonos en este análisis, concluimos que la utilización de una Red Neuronal Convolucional (CNN) era la opción más acertada. Específicamente, optamos por emplear el modelo MobileNet, ya pre-entrenado, a través de transfer learning. Esta decisión se debe a la necesidad de contar con un modelo que no solo sea ligero y eficiente, sino que también tenga la capacidad de procesar datos en tiempo real. Para el entrenamiento del modelo, optamos por utilizar *Google Colaboratory* debido a sus capacidades de procesamiento superiores en comparación con las computadoras personales convencionales.

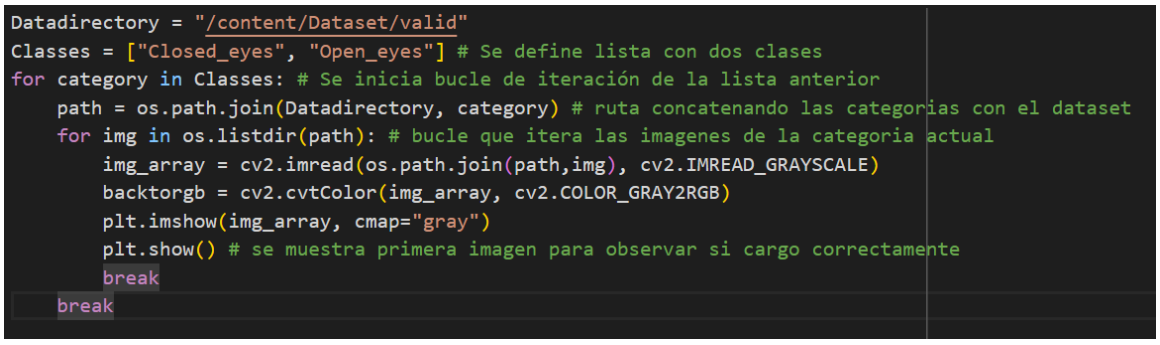
Para iniciar con la programación se importaron las librerías y funciones necesarias como se puede observar en la **Figura *5***.

Figura 5
Importación de librerías para entrenamiento con Transfer Learning.

```
import tensorflow as tf
import cv2
import os
import matplotlib.pyplot as plt
import numpy as np
import PIL
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
```

Para integrar el conjunto de datos en el código, se organizó de manera que los datos se dividieran en dos carpetas claramente marcadas, "Closed_eyes" y "Open_eyes", para facilitar la identificación y el análisis de las imágenes según si los ojos están abiertos o cerrados. Esta estructuración de los datos garantiza una mayor eficiencia y orden en el proceso de entrenamiento del modelo. Posteriormente, los datos se transfirieron del almacenamiento local al servidor, lo que permitió su incorporación al código, como se ilustra en la **Figura *6***. Para garantizar la correcta correspondencia, se unieron las etiquetas de las imágenes con el directorio, teniendo en cuenta que el nombre de las clases debe coincidir con el nombre de las carpetas. Además, se realizó una conversión de las imágenes de escala de grises a RGB para mantener la consistencia de los datos. Finalmente, se visualizó la primera imagen del conjunto para confirmar que la carga se había realizado correctamente.

Figura 6
Importación de datos de entrenamiento.



```
Datadirectory = "/content/Dataset/valid"
Classes = ["Closed_eyes", "Open_eyes"] # Se define lista con dos clases
for category in Classes: # Se inicia bucle de iteración de la lista anterior
    path = os.path.join(Datadirectory, category) # ruta concatenando las categorias con el dataset
    for img in os.listdir(path): # bucle que itera las imagenes de la categoria actual
        img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
        backtorgb = cv2.cvtColor(img_array, cv2.COLOR_GRAY2RGB)
        plt.imshow(img_array, cmap="gray")
        plt.show() # se muestra primera imagen para observar si cargo correctamente
        break
    break
```

Dado que el modelo MobileNet requiere imágenes de 224x224 píxeles y las imágenes en nuestra base de datos son de 86x86 píxeles, fue necesario redimensionarlas

para cumplir con esta especificación. Tras ajustar el tamaño de las imágenes, procedimos a visualizar una de ellas para asegurarnos de que todas las imágenes se hayan ajustado al tamaño requerido del modelo como se puede observar en la **Figura *7***.

Figura 7
Configuración de dimensiones acordes a MobileNet.

```
img_size =224
new_array = cv2.resize(backtorgb, (img_size, img_size)) #Se redimensionan las imágenes a 224*224
plt.imshow(new_array, cmap="gray")
plt.show()
```

Antes de proceder con el entrenamiento, inicialmente se generó una lista vacía destinada a almacenar todos los elementos de nuestra base de datos gestionada. En este proceso, cada imagen fue redimensionada para ajustarse a las especificaciones requeridas y etiquetada binariamente, utilizando "1" o "0" como marcadores. Estas imágenes redimensionadas y etiquetadas se almacenaron en la lista vacía creada al principio, preparando así el conjunto de datos para el siguiente paso del proceso de entrenamiento **Figura *8***.

Figura 8
Configuración de la lista de entrenamiento.

```
training_Data = [] #Se inicia una lista vacía para almacenar los datos del entrenamiento
def create_training_Data(): # Se define una función para los datos del entrenamiento
    for category in Classes:
        path = os.path.join(Datadirectory, category)
        class_num = Classes.index(category) #se obtiene el indice de la categoria actual
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
                backtorgb = cv2.cvtColor(img_array, cv2.COLOR_GRAY2RGB)
                new_array = cv2.resize(backtorgb, (img_size, img_size)) #Se redimensiona la imagen al tamaño 224*224
                training_Data.append([new_array, class_num]) #Se añade la imagen con su respectiva etiquta a la lista de entrenamiento
            except Exception as e:
                pass
```

Una vez establecida la función mencionada anteriormente, se procedió a su ejecución. Con el objetivo de prevenir el sobreajuste o el aprendizaje de patrones, se optó por mezclar aleatoriamente los datos en la lista de entrenamiento. Esta estrategia de aleatorización es fundamental para asegurar que el modelo aprenda de manera más efectiva y pueda generalizar mejor a partir de los datos proporcionados, evitando así sesgos en el aprendizaje que podrían surgir de un orden específico en los datos **Figura *9***.

Figura 9
Procesamiento y aleatorización de los datos de entrenamiento.

```
create_training_Data() # Se ejecuta la función definida anteriormente
random.shuffle(training_Data) # Se mezclan de manera aleatoria los datos para evitar sobreajustes
print(len(training_Data)) # se muestra la cantidad total de datos en la lista
```

Como se puede observar en la **Figura *10***, después de ejecutar la función previamente mencionada, se procedió a la creación de listas vacías. Estas listas se utilizaron para almacenar de forma organizada las etiquetas y las imágenes. Posteriormente, las listas, denominadas “X” para las imágenes y “Y” para las etiquetas, se transformaron en arreglos utilizando la biblioteca Numpy. Este paso es crucial para asegurar que los datos estén en el formato adecuado y sean compatibles con el modelo. La conversión a arreglos Numpy y los ajustes subsiguientes facilitan la integración eficiente de los datos con la arquitectura de la red neuronal que se va a entrenar.

Figura 10
División de datos y etiquetas.

```
X = [] # Se inician listas vacías "X" y "y"
y = []
for features,label in training_Data:
    X.append(features) # se agregan las imagenes de la lista de entrenamiento a la lista "X"
    y.append(label) # se agregan las etiquetas de la lista de entrenamiento a la lista "y"


X = np.array(X).reshape(-1, img_size, img_size, 3) # Se guarda en arreglo Numpy información todos los datos en la lista X
                    #(Longitud X, 224, 224, RGB)
Y = np.array(y)
```

Posteriormente, se realizó el proceso de normalización de los datos. Este paso facilita el aprendizaje del modelo y permite una convergencia más rápida durante el entrenamiento **Figura *11***.

Figura 11
Normalización.

```
X.shape
X= X/255.0;
```

Se procedió a inicializar una instancia del modelo MobileNet, utilizando los pesos que han sido previamente entrenados con el conjunto de datos ImageNet, **Figura *12***.

Figura 12
Activación de MobileNet ya entrenado previamente.

```
model = tf.keras.applications.mobilenet.MobileNet()
```

Con el modelo Mobilenet ya cargado, se procedió a utilizar la primera capa de este modelo pre-entrenado como la capa de entrada de nuestro nuevo modelo, asegurando que acepte los mismos formatos de datos de entrada. Posteriormente, seleccionamos como salida del modelo una de sus capas avanzadas, específicamente la cuarta última capa. Luego, se implementó una operación de aplanamiento mediante una capa 'Flatten', transformando la salida en un formato unidimensional. Esta transformación es necesaria para poder conectar adecuadamente la salida con una capa densa, la cual contará con una sola neurona. Esto se debe a que el modelo tiene como objetivo clasificar entre dos categorías posibles. Finalmente, en la capa de activación se evalúa el valor procedente de la capa densa, buscando el número más cercano para determinar si corresponde a un “1” o un “0”, **Figura *13***.

Figura 13
Desarrollo de un modelo personalizado basado en uno pre-entenado.

```
base_input = model.layers[0].input
base_output = model.layers[-4].output
Flat_layer = layers.Flatten()(base_output)
final_output = layers.Dense(1)(Flat_layer)
final_output= layers.Activation('sigmoid')(final_output)
```

Con los parámetros del nuevo modelo ya configurados, se procede a definir la entrada y la salida que se establecieron anteriormente en el nuevo modelo. Posteriormente, se establece la función de pérdida, se selecciona 'Adam' como el optimizador y se definen las métricas relevantes que guiarán y evaluarán el proceso de entrenamiento. A

continuación, se inicia el entrenamiento del modelo, utilizando las listas preparadas de imágenes y sus correspondientes etiquetas. Durante este proceso, se reserva un 10% de los datos para la validación, asegurando así una evaluación del modelo. Este entrenamiento se lleva a cabo a lo largo de 30 épocas, permitiendo un ajuste progresivo y detallado del modelo a los datos, **Figura *14***.

Figura 14
Entrenamiento del nuevo modelo.

```
new_model = keras.Model(inputs = base_input, outputs= final_output)
new_model.compile(loss="binary_crossentropy", optimizer = "adam",  metrics = ["accuracy"])
history = new_model.fit(X,Y, epochs = 30, validation_split = 0.1)
```

4.3 Funcionamiento de la red neuronal con mobilenet

La puesta en marcha de nuestra red neuronal previamente entrenada demandó la creación de un código de detección. Este código utiliza la red para evaluar y definir resultados en tiempo real, permitiendo así el funcionamiento efectivo del sistema.

Para comenzar con el funcionamiento se procede a la importación de las librerías y funciones requeridas para el funcionamiento como se puede observar en la **Figura *15***.

Figura 15
Importación de las librerías y funciones.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import cv2
import os
import matplotlib.pyplot as plt
import numpy as np
import winsound
import cv2
```

Después se importó a la programación la red neuronal previamente entrenada como se puede ver en la **Figura *16***.

Figura 16
Importación de librerías.

```
new_model.save('ModeloSomnolencia.h5')
```

En el proceso de desarrollo, se agregor las variables de la alarma para su funcionamiento y se incorporó al código la esencial funcionalidad de detección de rostros mediante el uso del clasificador en cascada Haar. Este paso se materializó a través de la definición del camino hacia el archivo XML del clasificador y la inicialización de la variable faceCascade con dicho clasificador como se puede ver en la **Figura *17***.

Figura 17
Incorporación de las variables de la alarma y del clasificador Haar.

```
frequency=2500
duration=1000
path ="haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default.xml')
```

A continuación, se procedió en la programación a la configuración de la captura de video, inicializando el objeto cap. En un primer intento, se buscó activar la cámara secundaria (índice 1), y en caso de no estar disponible, se realizó un segundo intento con la cámara principal (índice 0). La verificación de la apertura de la cámara se llevó a cabo mediante una condición En situaciones de fallo en la apertura, se generó una excepción *IOError* indicando que la cámara no pudo ser abierta.

Se crea una variable *counter* que inicializa en 0. Este segmento de código establece la base para la captura de video y manejo de errores en el sistema, como se aprecia en la **Figura *18***.

Figura 18
Funcionamiento de cámara.

```
cap = cv2.VideoCapture(1)
if not cap.isOpened():
    cap=cv2.VideoCapture(0)
if not cap.isOpened():
    raise IOError("no se pudo abrir camara")
counter=0
```

El código captura continuamente fotogramas desde la cámara y utiliza un clasificador Haar para detectar ojos en cada fotograma. Para cada par de coordenadas que representan un ojo detectado, se extraen regiones de interés en escala de grises y color, se dibuja un rectángulo en el fotograma original y se realiza una segunda detección de ojos dentro de la región de interés. Se imprime un mensaje si no se detectan ojos en la región

de interés. Este proceso se repite de manera continua, permitiendo la detección en tiempo real de ojos en el flujo de video de la cámara como se puede observar en la **Figura *19***.

Figura 19
Captura de fotograma.

```
while True:
    ret,frame = cap.read()
    eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    eyes=eye_cascade.detectMultiScale(gray,1.1,4)
    for x,y,w,h in eyes:
        roi_gray=gray[y:y+h, x:x+w]
        roi_color=frame[y:y+h, x:x+w]
        cv2.rectangle(frame, (x,y), (x+w, y+h),(0,255,0),2)
        eyess = eye_cascade.detectMultiScale(roi_gray)
        if len(eyess)== 0:
            print("ojos no detectados")
        else:
            for(ex,ey,ew,eh) in eyess:
                eyes_roi= roi_color[ey:ey+eh, ex:ex+ew]
```

En este fragmento, se convierte el fotograma actual a escala de grises y se verifica si el clasificador de rostros (*faceCascade*) está vacío. Luego, se detectan rostros en la imagen en escala de grises y se dibujan rectángulos alrededor de los ojos detectados en el fotograma original. Es importante señalar que las coordenadas utilizadas para dibujar los rectángulos provienen de la detección de ojos y no de la detección de rostros, por lo que es necesario ajustar esta parte del código para garantizar una detección y marcado preciso de rostros como se puede observar en la **Figura *20***.

Figura 20
Conversión a escala de grises y dibujo de rectángulos.

```
gray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
print(faceCascade.empty())
faces=faceCascade.detectMultiScale(gray,1.1,4)
for(x,y,w,h) in eyes:
    cv2.rectangle(frame,(x,y),(x+w, y+h),(0,255,0),2)
```

Se selecciona un estilo de fuente para visualizar en tiempo real el estado de la persona, **Figura *21***.

Figura 21
Definición de fuente de letra.

```
font=cv2.FONT_HERSHEY_SIMPLEX
```

La imagen final se ajusta al tamaño de 224x224 píxeles, se expanden sus dimensiones, se normaliza y luego se realiza una predicción utilizando el modelo específico como se puede ver en la **Figura *22***.

Figura 22
Ajuste y predicción.

```
final_image=cv2.resize(eyes_roi,(224,224))
final_image= np.expand_dims(final_image,axis =0)
final_image=final_image/255.0
predictions = new_model.predict(final_image)
```

se evalúa si las predicciones del modelo son mayores que cero. Si es así, se establece el estado como "ojos abiertos". Luego, se añade un texto en el fotograma indicando este estado y se dibuja un rectángulo para resaltar la información visual. Además, se agrega el texto 'ACTIVO' en otra ubicación del fotograma. Estas operaciones buscan proporcionar una representación visual del estado "ojos abiertos" en el video capturado como se puede observar en la **Figura *23***.

Figura 23
Evaluación y representación visual.

```
if(predictions>0):
    status=""
    cv2.putText(frame,
                status,
                (150, 150),
                font,
                3,
                (0, 255, 0),
                2,
                cv2.LINE_4)


    x1,y1,w1,h1=0,0,175,75
    cv2.rectangle(frame,(x1, x1),(x1+w1, y1+h1),(0,0,0),-1)

    cv2.putText(frame,'Despierto',(x1 + int(w1/10),y1 + int(h1/2)),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,255,0),2)
```

Si las predicciones del modelo no son mayores que cero, se incrementa el contador. Se establece el estado como "ojos cerrados" y se añade un texto en el fotograma indicando este estado, Además se cambia el color de los cuadros a rojo alrededor del ojo como se puede ver en la **Figura *24***.

Figura 24
Detección de ojos cerrados.

```
else:
    counter=counter+1
    status=""
    cv2.putText(frame,
                status,
                (150, 150),
                font,
                3,
                (0, 255, 0),
                2,
                cv2.LINE_4)

    cv2.rectangle(frame,(x ,y),(x+w, y+h),(0,0,255),2)
    cv2.putText(frame,'Ojos cerrados',(x1 + int(w1/10),y1 + int(h1/2)),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)
```

se dibuja un rectángulo alrededor de los ojos cerrados en el fotograma utilizando cv2.rectangle. Luego, se verifica si el contador supera los 5 ciclos. En caso afirmativo, se ejecutan acciones adicionales: se dibuja un rectángulo para ocultar parte de la pantalla y se

agrega un texto de "alerta dormido" y se active la alarma Sonora, Estas operaciones buscan alertar sobre la somnolencia cuando se detectan ojos cerrados durante un período prolongado como se puede ver en la **Figura *25***.

Figura 25
Tiempo transcurrido con ojos cerrados.

```
if counter > 5:
    x1,y1,w1,h1=0,0,175,75
    cv2.rectangle(frame, (x1,x1),(x1+w1, y1+h1), (0,0,0), -1)
    cv2.putText(frame, 'alerta dormido', (x1 + int(w1/10), y1 + int(h1/2)), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    winsound.Beep(frequency,duration)
    counter=0
```

Por último, se mostrará el fotograma con la detección de los ojos en tiempo real y para lograr finalizar el programa se oprimirá la letra q, que permite salir del ciclo while y logrando cerrar correctamente la aplicación de manera adecuada usando “cap.release()” y “cv2.destroyAllWindows()” como se puede observar en la **Figura *26***.

Figura 26
Impresión de fotograma y finalización del sistema.

```
    cv2.imshow('deteccion de insomnio',frame)
    if cv2.waitKey(2) & 0xFF == ord('q'):
        break


cap.release()

cv2.destroyAllWindows()
```

4.4 Diseño del sistema con Dlib

Para trabajar sobre herramienta de "machine learning" y procesamiento de imágenes, "dlib" es una biblioteca abierta que nos permite realizarlo, y este modo, la captura de video en tiempo real pueda detectar la somnolencia a partir de unos valores establecidos. Este sistema no requiere la importación de Tensorflow debido a que no es necesario cargar una red neuronal previamente entrenada.

Para el inicio del programa se importan las librerías esenciales para su correcto funcionamiento, lo cual se refleja en la **Figura *27***.

Figura 27
Importación de librerías para el sistema hecho con Dlib.

```
#Librerias necesarias para el funcionamiento del código
from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
from pygame import mixer
import numpy as np
import argparse
import imutils
import time
import dlib
import cv2
import os
```

Se inicia la configuración de la alarma al incorporar el archivo “alarm.wav”, como se puede visualizar en la **Figura *28***.

Figura 28
Configuración de la alarma.

```
#Agregamos alarma para el aviso de la deteccion de somnolencia
mixer.init()
mixer.music.load('alarm.wav')
```

A través de una función que lleve a cabo la medición en tiempo real del EAR en ambos ojos, se calcula un valor promedio único del EAR para el par de ojos. De esta manera, se proporciona el valor de las coordenadas correspondientes, como se muestra en la **Figura *29***.

Figura 29
Función para obtener el EAR promedio de los ojos.

```
#Función para calcular el EAR promedio de ambos ojos
def final_ear(shape):
    (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
    (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]

    left_eye = shape[lStart:lEnd]
    right_Eye = shape[rStart:rEnd]

    leftEAR = aspecto_del_ojos( left_eye)
    rightEAR = aspecto_del_ojos(right_Eye)

    ear = (leftEAR + rightEAR) / 2.0
    return (ear, left_eye, right_Eye)
```

Para la detección de bostezo se requiere una función que permita calcular la distancia de los labios. de esta manera se obtiene eso datos a través de las coordenadas

definidas como se puede observar en la **¡Error! No se encuentra el origen de la r eferencia.**.

Figura 30
Función para obtener la distancia de los labios.

```
def lip_distancia(shape):
    top_lip = shape[50:53]
    top_lip = np.concatenate((top_lip, shape[61:64]))

    low_lip = shape[56:59]
    low_lip = np.concatenate((low_lip, shape[65:68]))

    top_mean = np.mean(top_lip, axis=0)
    low_mean = np.mean(low_lip, axis=0)

    distancia = abs(top_mean[1] - low_mean[1])
    return distancia
```

se requiere gestionar el índice de la cámara por lo cual se usa “argparse” como alternativa en caso de no especificar el índice asumirá por defecto “0” y después se establecen valores constantes con el objetivo de ser de referencia como se puede apreciar en la **Figura *31***.

Figura 31
Parámetros iniciales para el análisis.

```
#Se establecen argumentos y variables generales para la detección
ap = argparse.ArgumentParser()
ap.add_argument("-w", "--camara", type=int, default=0,
                help="index of webcam on system")
args = vars(ap.parse_args())

EYE_AR_THRESH = 0.2 #limite de EAR
EYE_AR_CONSEC_FRAMES = 5 #tiempo limite para ojos cerrados
estado_1 = False
contador = 0
```

Para que el sistema funcione correctamente, es esencial contar con el clasificador Haar y el predictor de 68 puntos. Se proporcionará una notificación cuando ambos estén siendo cargados para su funcionamiento, tal como se muestra en la **Figura *32***.

Figura 32
Carga del sistema.

```
#Se carga el clasificador Haar para la detección de rostros con OpenCV
print("-> cargando el sistema...")
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

#Se carga el predictor de los 68 puntos faciales que tiene la librería Dlib
predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
```

El sistema emite una notificación al decidir comenzar con la cámara, proporcionando un intervalo de un segundo para la activación, como se ilustra en la **Figura *33***.

Figura 33
Inicio del sistema.

```
#Se inicia la captura de video y se espera un 1 segundo para que la cámara se active
print("-> iniciando camara")
vs = VideoStream(src=args["camara"]).start()
time.sleep(1.0)
```

Se creó un bucle while diseñado para el funcionamiento en tiempo real, permitiendo el análisis de cada cuadro de manera continua. Se lleva a cabo la captura utilizando el método "vs.read" y se redimensiona a un tamaño de 450 para facilitar el procesamiento, como se evidencia en la **Figura *34***.

Figura 34
Captura y redimensionamiento del fotograma.

```
#Se captura y redimensiona el frame
frame = vs.read()
frame = imutils.resize(frame, width=450)
```

El cuadro previamente redimensionado se transforma a una representación en escala de grises, con el propósito de conservar recursos del sistema, tal como se visualiza en la **Figura *35***.

Figura 35
Conversión a escala de grises.

```
#Se convierte a escala de grises
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Empleando el clasificador Haar, se intenta identificar el rostro en el cuadro que ha sido previamente transformado. A través de los parámetros "minNeighbors" y "minSize",

se tiene la capacidad de ajustar la sensibilidad y precisión de la detección. Finalmente, la variable "rects" almacena las coordenadas de los rectángulos que delimitan los ojos, como se ilustra en la **Figura *36***.

Figura 36
Detección del rostro.

```
#Se Detecta el rostro
rects = detector.detectMultiScale(gray, scaleFactor=1.1,
    minNeighbors=5, minSize=(30, 30),
    flags=cv2.CASCADE_SCALE_IMAGE)
```

Después de adquirir esta información, se utiliza un bucle for para detectar y analizar los cuadros de manera consecutiva. Dentro de este bucle, se requiere la creación de un rectángulo para definir la región del rostro, para lo cual Dlib proporciona un comando específico. Luego, se obtienen los puntos faciales para calcular el EAR y se registran las coordenadas de los ojos y labios. Se aplica una envolvente convexa a los contornos de los ojos y labios, resultando en la creación y dibujo de cuadros que rodean cada ojo y también los labios como se puede ver en la **Figura *37***.

Figura 37
Inicio de bucle y cálculo del EAR.

```
for (x, y, w, h) in rects:
    #Se calcula el EAR, luego se calcula la distancia de labios y se dibuja los contornos
    rect = dlib.rectangle(int(x), int(y), int(x + w),int(y + h))

    shape = predictor(gray, rect)
    shape = face_utils.shape_to_np(shape)

    ojo = final_ear(shape)
    ear = ojo[0]
    leftEye = ojo[1]
    rightEye = ojo[2]
    distance = lip_distancia(shape)
    leftEyeHull = cv2.convexHull(leftEye)
    rightEyeHull = cv2.convexHull(rightEye)
    cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
    cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
```

Cuando el valor de EAR es inferior a la referencia establecida, se activa un contador que, después de transcurrir un cierto periodo de tiempo, cambia el estado_1. Esto desencadena la activación de la alarma y la visualización de un mensaje en tiempo real en el cuadro. La alarma y el mensaje en pantalla se mantienen hasta que la persona se despierta o se detecta que la medida del ojo vuelve a ser igual o superior a la referencia. En ese momento, el contador se reinicia y el estado_1 se desactiva, eliminando la alarma. Este ciclo se repite cuando el valor de EAR desciende por debajo de la referencia y transcurre el tiempo establecido, como se muestra en la **Figura *38***.

Figura 38
Control del EAR.

```
#Si el Ear es muy bajo, se inicia un conteo para tener en cuenta cuanto tiempo se tienen los ojos cerrados
if ear < EYE_AR_THRESH:
    contador += 1
    #si este tiempo supera el permitido se inicia el aviso de somnolencia
    if contador >= EYE_AR_CONSEC_FRAMES:
        if estado_1 == False:
            estado_1 = True
            mixer.music.play()
        #Se da alerta en pantalla
        cv2.putText(frame, "somnoliento!", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
else:
    contador = 0
    estado_1 = False
```

Para el control mediante el bostezo se aplicará el mismo comportamiento al control de los ojos donde cambiará el a que el valor no debe superar la referencia durante un intervalo de tiempo como se puede ver en la **¡Error! No se encuentra el origen de la referencia.**.

Figura 39
Control de labios.

```
# Si la distancia suele ser mas abierta a la referencia procede a activar un contador
if distance > bozteso_detectado:
        contador_2 += 1
        #si este tiempo supera el permitido se inicia el aviso de somnolencia
        if estado_2 == False:
            estado_2 = True
            mixer.music.play()
         #Se da alerta en pantalla
        cv2.putText(frame, "Alerta bozteso!", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

else:
    estado_2 = False
    contador_2 = 0
```

A lo largo de este procedimiento, se tiene la intención de visualizar el valor del EAR y la distancia de los labios medido. Este valor será mostrado en el cuadro mediante el uso de la función "cv2.putText", tal como se evidencia en la **Figura *40***.

Figura 40
Impresión de valores en tiempo real del EAR y distancia de labios.

```
#Se muestran en pantalla los valores en tiempo real del EAR y dsintacia de los labios
cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
cv2.putText(frame, "bostezo: {:.2f}".format(distance), (300, 60),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
```

Finalmente, para permitir que el usuario observe en tiempo real, se utiliza "cv2.imshow" para mostrar el cuadro resultante. Con la función "cv2.waitKey(1) & 0xFF", se espera la entrada de una tecla, específicamente 'q' en este caso, para cerrar el programa y finalizar la ventana junto con el video en vivo como se observa en la **Figura *41***.

Figura 41
Finalización del programa.

```
    cv2.imshow("Frame", frame)
    apagado = cv2.waitKey(1) & 0xFF
    #Se permite cerrar el programa con la tecla q si se desea
    if apagado == ord("q"):
        break

#Finalmente si se cierra el programa se detienen todas las ventanas y el stream
cv2.destroyAllWindows()
vs.stop()
```

5. Análisis de resultados

A continuación, se muestran los resultados del proceso de entrenamiento y validación, incluyendo gráficos y cifras específicas del modelo realizado por Transfer learning. Asimismo, se presentarán los datos estadísticos del desempeño del sistema creado con Dlib para el análisis de la apertura de los ojos mediante el EAR. También se muestra el funcionamiento de cada sistema completo para la detección de somnolencia.

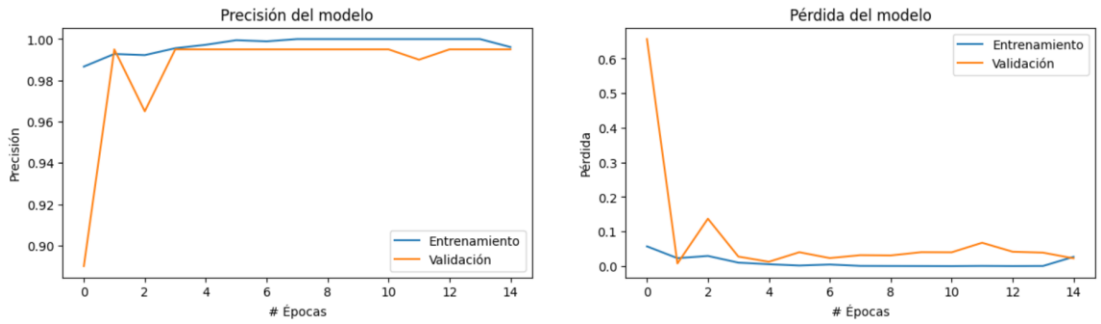
5.1. Modelo entrenado con Transfer learning

Para desarrollar el modelo encargado de clasificar el estado de los ojos, se empleó la metodología de transfer learning, utilizando como base el modelo MobileNet. En la **Tabla *3***, se detallan las estadísticas numéricas correspondientes al entrenamiento y la validación logrados con el conjunto de datos seleccionado. Por otro lado, en la **Figura *42***, se ilustran las trayectorias de precisión y error a lo largo de las etapas de entrenamiento y validación.

Tabla 3
Resultados numéricos de entrenamiento.

Entrenamiento		Validación	
Pérdida	Precisión	Pérdida	Precisión
0.0269	0.9961	0.0226	0.995

Figura 42
Gráficas de precisión y pérdida del entrenamiento.



Los resultados obtenidos en el entrenamiento, como se muestra en la figura 41, indican un desempeño satisfactorio en la detección y clasificación del estado de los ojos. Este Resultado es muy positivo, considerando la aplicación de la técnica de transfer learning, lo cual subraya la efectividad de la arquitectura de MobileNet en tareas de clasificación como esta. Estos hallazgos respaldan la viabilidad de MobileNet para la identificación del estado de los ojos, lo que es un aspecto clave en diversas aplicaciones prácticas.

5.2. Pruebas de detección de somnolencia con el modelo de Mobilenet.

Para la evaluación del rendimiento del programa, se procedió a iniciar su ejecución y se observó su comportamiento durante la detección facial en tiempo real. Se realizaron pruebas para verificar cómo cambia dinámicamente el tamaño de los rectángulos en función de la detección del rostro, como se muestra en la **Figura *43*** y la **Figura *44***. En una

de las imágenes, se evidencia que los cuadros tienen dimensiones uniformes, mientras que, en la otra, se aprecia un ligero aumento en el tamaño del cuadro izquierdo. Adicionalmente, se probó la activación de la alarma, la cual emite un sonido inmediato como alerta. Esta característica facilita despertar a la persona de manera efectiva en caso de que la medición del EAR indique somnolencia. Estas pruebas permitieron evaluar y ajustar la funcionalidad del programa en condiciones prácticas.

Figura 43
Persona despierta.

Figura 44
Persona con los ojos cerrados luego de un tiempo.

Las imágenes visualizadas nos permiten analizar un correcto funcionamiento del Sistema para determinar su aplicación en la Raspberry pi 4, y que factores puedan llegar a afectar de manera significative para la prevención.

5.3. Pruebas de detección de somnolencia con Dlib

En esta parte del estudio, se muestran las evaluaciones realizadas al sistema completo desarrollado con Dlib para la detección de somnolencia. El objetivo es examinar la precisión y el rendimiento del programa en la detección en tiempo real.

La selección de un valor óptimo para la proporción del Aspecto del Ojo (EAR) es crucial para detectar la somnolencia de manera efectiva y actuar rápidamente antes de un posible cierre completo de los ojos, situación que puede ser crítica en ciertos contextos. Para esto, Se consideró el estado de entrecerrado de los ojos con el fin de lograr una detección rápida de la somnolencia. Para determinar el valor más adecuado de EAR, se llevaron a cabo pruebas con 26 voluntarios que participaron como sujetos de prueba en este proyecto. Estas pruebas fueron fundamentales para establecer los valores de EAR más efectivos bajo diversas condiciones. Los resultados obtenidos a partir de estas pruebas son esenciales para comprender cómo diferentes valores de EAR pueden influir en la precisión y la rapidez de la detección de somnolencia en escenarios reales.

Figura 45
Parte del conjunto de datos de prueba.

Como se muestra en la **Figura *45***, el análisis tomó en cuenta tres estados diferentes de los ojos: abiertos, entrecerrados y cerrados, se realizó de esta manera para poner a prueba el EAR. Las imágenes fueron evaluadas mediante el código programado, que emitía un veredicto sobre si existía somnolencia o no, tal como se ilustra en la **Figura *46***. Con esta metodología, se lograron recolectar datos significativos que favorecieron el análisis y permitieron obtener resultados concluyentes, los cuales se presentan detalladamente en la **Tabla *4***. Estos resultados son fundamentales para comprender cómo el sistema responde a diferentes condiciones oculares y para validar la eficacia de este en la detección de somnolencia en tiempo real.

Figura 46
Imagen de prueba detectada con somnolencia con 0.25 de EAR.

Tabla 4
Porcentaje de precisión de diferentes valores de EAR.

EAR	Porcentaje de precisión
0.1	36,9%
0.2	82,5%
0.25	97,2%
0.3	87,8%
0.4	33,3%

Según se detalla en la Tabla 4, el valor óptimo identificado para la detección de somnolencia es 0.25. Una vez establecido este valor, el sistema fue sometido a pruebas en tiempo real para evaluar su rendimiento práctico, como se evidencia en la **Figura *47*** y la **Figura *48***. Estas pruebas en escenarios reales son cruciales para verificar la eficacia del sistema y asegurar su fiabilidad en la detección de somnolencia, proporcionando así una validación adicional del valor seleccionado para el EAR y su aplicabilidad en situaciones del mundo real.

Figura 47
Prueba con ojos abiertos con Dlib.

Figura 48
Prueba con ojos entrecerrados con Dlib.

Por otro lado, se implementó una notificación de advertencia para detectar bostezos prolongados, con el objetivo de identificar posibles signos de sueño como se pueden observar en la Figura 49.

Figura 49
Alerta luego de un bostezo prolongado

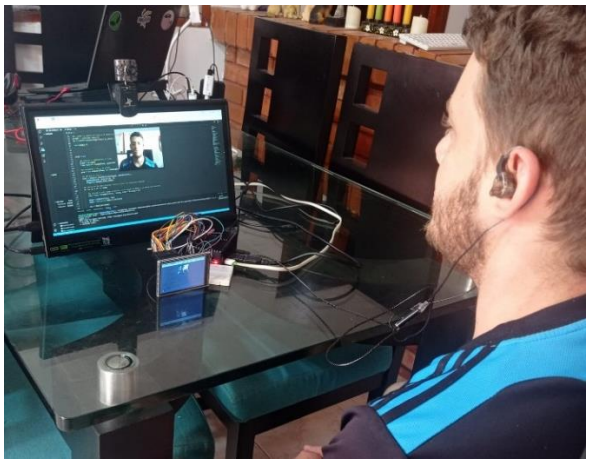
Los resultados obtenidos por el sistema en la detección oportuna de la somnolencia con Dlib fueron altamente positivos. Este sistema, al identificar signos de fatiga en una persona, activa una alarma como señal de advertencia, lo que demuestra su eficacia en emitir alertas y realizar la detección en tiempo real. Además, destaca por su fluidez de funcionamiento y por no requerir una gran capacidad de procesamiento, lo que contribuye a su óptimo desempeño en diversas condiciones de uso. Estas características lo hacen especialmente valioso en situaciones críticas donde la rapidez y la precisión en la detección de la somnolencia son esenciales.

5.4. Selección del sistema para la Raspberry pi 4

Se adaptaron los requisitos necesarios para habilitar el funcionamiento adecuado de ambos sistemas en la Raspberry Pi 4, un sistema embebido capaz de operar con sistemas operativos de 32 o 64 bits. La elección del sistema de transfer learning requería el uso del sistema de 64 bits debido a la compatibilidad con las versiones actuales de TensorFlow. Sin embargo, se observó que este sistema, al ponerse en marcha en la Raspberry Pi 4, generaba una carga significativa en la máquina, resultando en una detección poco fluida, a diferencia de su rendimiento en una computadora personal.

Por otro lado, el sistema implementado con Dlib demostró versatilidad al funcionar eficientemente en ambos sistemas operativos, requiriendo menos potencia de máquina. Durante las detecciones, este sistema demostró una fluidez notable, proporcionando alertas oportunas. Esta adaptabilidad y eficiencia hacen que el sistema basado en Dlib sea una elección favorable para la Raspberry Pi 4 en comparación con el sistema de transfer learning.

Figura 50
Sistema funcionando en Raspberry pi 4.

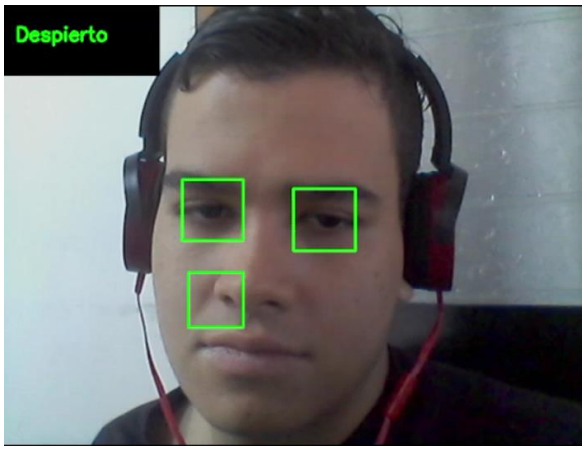


Como se puede apreciar en la Figura 50 , el sistema implementado con Dlib opera de manera efectiva en el dispositivo embebido Raspberry Pi 4, demostrando una adecuada capacidad para identificar la somnolencia mediante criterios precisos.

5.5. Error de detección del clasificador en cascada

Se empleó el archivo *haarcascade_eye.xml* para llevar a cabo la detección de ojos durante el funcionamiento del sistema, pero se experimentaron inconvenientes al realizar dicha detección. En esta situación específica, se identificaron otros rasgos erróneamente como ojos, como se evidencia en la Figura 51.

Figura 51
Detección de rasgos no correspondientes a ojos.



6. Conclusiones y recomendaciones

- Se desarrolló un sistema de detección de somnolencia que evalúa la apertura de los ojos, implementando dos enfoques distintos con resultados positivos. Uno de ellos se basa en la técnica de transfer learning utilizando un modelo de red neuronal convolucional (CNN) con la arquitectura de Mobilenet, mientras que el otro se construyó con la librería de machine learning Dlib. Además, añadiendo con Dlib, una alerta de bostezos prolongados. Tras comparar ambos sistemas, se optó por implementar en la Raspberry Pi 4 el desarrollado con Dlib. Esta elección se basó en su rendimiento en tiempo real, su capacidad de hacer análisis mediante el EAR y no esperar el cierre completo de los ojos, su bajo peso y su excelente desempeño, logrando una efectividad del 97.2% en la prueba de detección de somnolencia y una mayor fluidez en el sistema embebido.

- Al implementar el sistema de detección de somnolencia en la Raspberry Pi 4 utilizando el modelo entrenado mediante transfer learning, se observó un rendimiento poco fluido durante las pruebas en tiempo real. Este inconveniente se atribuye a la mayor demanda de potencia de procesamiento necesaria para el análisis en tiempo real. Asimismo, el requisito del sistema operativo embebido de 64 bits contribuye a tensionar al máximo los recursos de la máquina, afectando la eficiencia del sistema en el entorno de la Raspberry Pi 4. Estos hallazgos resaltan la importancia de considerar las limitaciones de recursos en sistemas embebidos para lograr un funcionamiento óptimo.
- La adición de un contador de bostezos al sistema implementado con Dlib, mediante la medición de la distancia entre los labios superior e inferior, representa una valiosa característica adicional. Esta funcionalidad puede ser fundamental para mejorar la capacidad del sistema en la detección signos tempranos de somnolencia y la fatiga. Al monitorear la distancia entre los labios, se puede capturar la frecuencia de bostezos, proporcionando así información adicional sobre el estado de alerta del individuo.
- Si se considera la reutilización de la arquitectura MobileNet para abordar problemas similares en el futuro, se recomienda contar con una capacidad de computación adecuada. Esto permitiría realizar entrenamientos más extensos con conjuntos de datos más grandes, abordando así las limitaciones previamente mencionadas que afectaron la calidad del entrenamiento en esta ocasión. La disponibilidad de

recursos computacionales suficientes facilitará un proceso de entrenamiento más robusto, mejorando la capacidad del modelo para generalizar y ofrecer resultados más precisos en la detección de patrones específicos relacionados con el problema en cuestión.

- Para la Raspberry pi 4, existen varios sistemas operativos, estos sistemas la selección afectan en la herramientas o librerías ofrecidas se recomienda instalar el sistema de 32 bits full, debido a que este se encuentra mejor distribuido para el correcto funcionamiento del sistema, y a su vez no genera complicaciones o demoras al momento de su instalación y configuración.
- La medición del EAR se referencio para Colombia obteniendo el mejor resultado para esa región, el valor puede verse afectado hacia la región deseada por lo cual es necesario un estudio o análisis de las personas para determinar el EAR óptimo de acuerdo con la región o las personas que habitan en esas zonas.
- La iluminación es un factor crucial al momento del funcionamiento, para esto se manejó una cámara que pudiera proporcionar luz cuando detecte el alrededor oscuro y pueda activarse sin necesidad de programar o agregar una función reduciendo la posibilidad de un dato erróneo durante el funcionamiento.
- Existen diversidad de formas de aplicar el sistema se recomienda diseñar varios modelos 3D que puedan implementarse y se adecuen a necesidades como el tamaño, lugar a operar y cambio de componentes a necesidad de la situación.

Lista de referencias

Belmar, A. (2021). *Deep Laerning Crea tu red neuronal*. RedUsers. https://www.google.com.co/books/edition/Deep_Laerning/VBxREAAAQBAJ?hl=es&gbpv=1&pg=PP1&printsec=frontcover

Čolić, Aleksandar., Marques, Oge., & Furht, Borko. (2014). Driver Drowsiness Detection Systems and Solutions (1st ed. 2014.). Springer International Publishing. https://doi.org/10.1007/978-3-319-11535-1

Dom, C., Heras, J., & Pascual, V. (2017). IJ-OpenCV: Combining ImageJ and OpenCV for Processing Images in Biomedicine. *Computers in Biology and Medicine*, *84*, 189–194. https://doi.org/10.1016/j.compbiomed.2017.03.027

Escobar Córdoba, F. (2020). Somnolencia diurna excesiva e insomnio: males de los tiempos actuales. Editorial Universidad Nacional de Colombia.

Forsman, P. M., Vila, B. J., Short, R. A., Mott, C. G., & Van Dongen, H. P. A. (2013). Efficient driver drowsiness detection at moderate levels of drowsiness. Accident Analysis and Prevention, 50, 341–350. https://doi.org/10.1016/j.aap.2012.05.005

Garcia Prats, J. (1998). Conductores...cuidado con los `microsuenos'! Dos o tres segundos son suficientes para una tragedia. *La Opinión (Los Angeles, Calif.)*, *72*(144).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Hartwig, A. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. ArXiv (Cornell University). https://doi.org/10.48550/ARXIV.1704.04861

https://doi.org/10.1109/ACCESS.2020.3034343

Machado-Duque, M. E., Echeverri Chabur, J. E., & Machado-Alba, J. E. (2015). Somnolencia diurna excesiva, mala calidad del sueño y bajo rendimiento académico en estudiantes de Medicina. *Revista colombiana de psiquiatría*, *44*(3), 137–142. https://doi.org/10.1016/j.rcp.2015.04.002

Media Research Lab. (2018), Conjunto de datos de ojos MRL, Recuperado de http://mrl.cs.vsb.cz/eyedataset

Mohammed, A. Z., Mohammed, E. A., & Aaref, A. M. (2020). Real-Time Driver Awareness Detection System. *IOP Conference Series. Materials Science and Engineering*, *745*(1), 12053-. https://doi.org/10.1088/1757-899X/745/1/012053

Pérez-Aguilar, D. A., Risco-Ramos, R. H., & Casaverde-Pacherrez, L. (2021). Transfer learning en la clasificación binaria de imágenes térmicas. *INGENIUS*, *26*, 71–86. https://doi.org/10.17163/ingenius.n26.2021.07

Portilla-Maya, S. de la, Dussán-Lubert, C., Montoya-Londoño, D. M., Taborda-Chaurra, J., & Nieto-Osorio, L. S. (2019). CALIDAD DE SUEÑO Y SOMNOLENCIA DIURNA EXCESIVA EN ESTUDIANTES UNIVERSITARIOS DE DIFERENTES DOMINIOS. *Hacia la promocíon de la salud*, *24*(1), 84–96. https://doi.org/10.17151/hpsal.2019.24.1.8

Rangel-Cortes, J. J., Ruiz-Castilla, J. S., García-Lamont, F., & Cervantes-Canales, J. (2019). Redes Neuronales Convolucionales en la identificación de melanomas benignos y malignos. *RISTI : Revista Ibérica de Sistemas e Tecnologias de Informação*, *E23*, 15–27.

Rey De Castro Mujica, J. (2013). Accidentes de tránsito en carreteras e hipersomnia durante la conducción. ¿Es frecuente en nuestro medio? La evidencia periodística. Revista Médica Herediana, 14(2), 69-. https://doi.org/10.20453/rmh.v14i2.758

Rosales Mayor, E., & Rey De Castro Mujica, J. (2010). Somnolencia: Qué es, qué la causa y cómo se mide. Acta médica peruana, 27(2), 137–143.

Shen, J., Barbera, J., & Shapiro, C. M. (2006). Distinguishing sleepiness and fatigue: focus on definition and measurement. Sleep Medicine Reviews, 10(1), 63–76. https://doi.org/10.1016/j.smrv.2005.05.004

System for effective Assessment of driver vigilance and Warning According to traffic risk Estimation. (s/f). https://cordis.europa.eu/project/id/IST-2000-28062

Vrbancic, G., & Podgorelec, V. (2020). Transfer Learning With Adaptive Fine-Tuning. *IEEE Access*, *8*, 196197–196211.

Yang, Y., & Fan, F. (2023). Ancient thangka Buddha face recognition based on the Dlib machine learning library and comparison with secular aesthetics. *Heritage Science*, *11*(1), 137–16. https://doi.org/10.1186/s40494-023-00983-8

Apéndices

Apéndice A. Repositorio GitHub DSS2024

En el siguiente enlace puede consultar de manera libre el código utilizado para el desarrollo de este proyecto el cual corresponde a un repositorio público de GitHub.

https://github.com/NicolasLenis/DSDSURN222