UNIVERSIDAD TÉCNICA DEL NORTE

Faculta de Ingeniería en Ciencias Aplicadas

Carrera de Software

DETECCIÓN DE SOMNOLENCIA EN CONDUCTORES DURANTE LA CONDUCCIÓN DIURNA Y NOCTURNA UTILIZANDO UN SISTEMA EMBEBIDO BASADO EN LA RED NEURONAL CONVOLUCIONAL (CNN) RESNEXT-50.

Trabajo de grado presentado ante la Ilustre Universidad Técnica del Norte previo

a la obtención del título de Ingeniero de Software.

Autor:

Marco Vinicio Inlago Cuascota

Director:

PhD. Iván Danilo García Santillán

Ibarra, 2025

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1727679688		
APELLIDOS Y NOMBRES:	Inlago Cuascota Marco Vinicio		
DIRECCIÓN:	Cayambe		
EMAIL:	mvinlagoc@utn.edu.ec		
TELÉFONO FIJO:		**TELÉFONO MÓVIL:**	0978878036

DATOS DE LA OBRA	
TÍTULO:	DETECCIÓN DE SOMNOLENCIA EN CONDUCTORES DURANTE LA CONDUCCIÓN DIURNA Y NOCTURNA UTILIZANDO UN SISTEMA EMBEBIDO BASADO EN LA RED NEURONAL CONVOLUCIONAL (CNN) RESNEXT-50.
AUTOR (ES):	Inlago Cuascota Marco Vinicio
FECHA DE APROBACIÓN: DD/MM/AAAA	31/01/2025
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniero en Software
DIRECTOR:	PhD. Iván Danilo García Santillán
ASESOR:	Ing. Cosme MacArthur Ortega Bustamante MSc.

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 31 días del mes de enero de 2025

EL AUTOR:

ESTUDIANTE
Inlago Cuascota Marco Vinicio
C.I: 1727679688

CERTIFICACIÓN DIRECTOR

Ibarra, 31 de enero del 2025

CERTIFICACIÓN DIRECTOR DEL TRABAJO DE TITULACIÓN

Por medio del presente yo PhD. Iván García, certifico que el sr. Marco Vinicio Inlago Cuascota portador de la cédula de ciudadanía número 1727679688, ha trabajado en el desarrollo del proyecto de grado "Detección de somnolencia en conductores durante la conducción diurna y nocturna utilizando un sistema embebido basado en la red neuronal convolucional (CNN) ResNeXt-50.", previo a la obtención del Título de Ingeniero en software realizado con interés profesional y responsabilidad que certifico con honor de verdad.

Es todo en cuanto puedo certificar a la verdad.

Atentamente

IVAN DANILO GARCIA SANTILLAN

Digitally signed by IVAN DANILO GARCIA SANTILLAN
Date: 2025.01.31 11:36:53 -05'00'

PhD. Iván Danilo García Santillán

DIRECTOR DE TRABAJO DE GRADO

Dedicatoria

Con todo mi amor y emoción, dedico esta tesis:

A mi madre Alicia, la persona que siempre me ha apoyado incondicionalmente para alcanzar este logro, que es tanto mío como suyo, gracias por todo su esfuerzo, por su sacrificio, por sus ganas de querer ver a su hijo lograrlo. Sus palabras de aliento, su fortaleza y todo su amor que puso en mí, hoy dan sus resultados. Mamá, estoy infinitamente agradecido por todo.

A mis abuelitos Josefina y Julián, por sus enseñanzas, historias y bendiciones que me han acompañado durante toda mi vida.

A toda mi familia, que siempre me han apoyado a seguir adelante durante toda esta etapa universitaria. Gracias por su comprensión y su confianza durante todo este trayecto.

Todos y cada uno de ustedes, con su amor, apoyo, comprensión y confianza en mí, han sido los pilares que me han ayudado a lograr esta meta. Gracias.

Inlago Marco.

Agradecimientos

En primer lugar, agradezco con todo mi corazón a mi madre. Su esfuerzo, dedicación y confianza en mí me han permitido alcanzar esta meta. Gracias por ser la persona que siempre estuvo ahí en los momentos buenos y difíciles, haciéndome ver que, si le pones esfuerzo dedicación y amor, todo se puede lograr.

A mis profesores, por su valiosa orientación, confianza y tiempo dedicado. Su experiencia y guía han sido los pilares fundamentares para desarrollar y culminar este trabajo.

A mi familia, por su paciencia, comprensión y apoyo incondicional durante todo este proceso. Su aliento y compañía han sido fundamentales para superar los retos y alcanzar esta meta.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Resumen

La detección de somnolencia durante la conducción diurna y nocturna actualmente es un tema muy importante debido a que el número de accidentes de tránsito ocasionados por cansancio o somnolencia ha tenido un aumento significativamente alto. La causa del aumento puede deberse al aumento del trabajo que puede tener los conductores o algún tipo de enfermedad que cause la disminución de la concentración durante la conducción. Este trabajo propone el modelo de Red Neurona Convolucional (CNN) ReNext-50 para la detección de estado de somnolencia en conductores durante la conducción diurna y nocturna. Este modelo se lo integrará en el sistema embebido Raspberry Pi 3 Modelo B+ el cual será instalado en vehículos. El trabajo consta de tres capítulos que abarca el tema de investigación, desarrollo del modelo y sistema embebido y la evaluación de los resultados obtenidos. En el primer capítulo se realiza una búsqueda literaria de artículos y trabajos relacionados en base a la problemática de la somnolencia, modelos utilizados para la detección de somnolencia. En el segundo capítulo se abarca la creación del Dataset propio, la creación del Dataset para el entrenamiento que incluye los datos de nuestro Dataset y de otros enfocados en este problema, la creación y entrenamiento del modelo ResNeXt-50 para las tareas de clasificación y regresión, y por último la creación e instalación del modelo en el sistema embebido. En el tercer capítulo se evalúan los modelos con 5 diferentes sets de pruebas, rostros sin accesorios faciales, con lentes, con gafas, durante el día y durante la noche. Se obtiene métricas de estas distintas evaluaciones y se los compara con trabajos de la literatura. Además, se realizan pruebas en entornos simulados y reales al sistema embebido, y se evalúa la característica de portabilidad.

Abstract

The detection of drowsiness during daytime and nighttime driving is currently a principal issue because the number of traffic accidents caused by tiredness or drowsiness has increased significantly. The cause of the increase may be due to the increased workload that drivers may have or some kind of illness that causes decreased concentration while driving. This work proposes the ReNext-50 Convolutional Neuron Network (CNN) model for drowsiness state detection in drivers during daytime and nighttime driving. This model will be integrated into the Raspberry Pi 3 Model B+ embedded system which will be installed in vehicles. The work consists of three chapters that cover the research topic, development of the model and embedded system and the evaluation of the results obtained. In the first chapter, a literary search of articles and related works based on the problem of drowsiness, models used for drowsiness detection is conducted. The second chapter covers the creation of our own Dataset, the creation of the Dataset for training that includes data from our Dataset and others focused on this problem, the creation and training of the ResNeXt-50 model for the classification and regression tasks, and finally the creation and installation of the model in the embedded system. In the third chapter the models are evaluated with five different test sets, faces without facial accessories, with glasses, with glasses, during the day and during the night. Metrics are obtained from these different evaluations and compared with works in literature. In addition, the embedded system is evaluated in simulated and real environments, and the portability feature is evaluated.

Introducción

Tema

Detección de somnolencia en conductores durante la conducción diurna y nocturna utilizando un sistema embebido basado en la red neuronal convolucional (CNN) ResNeXt-50.

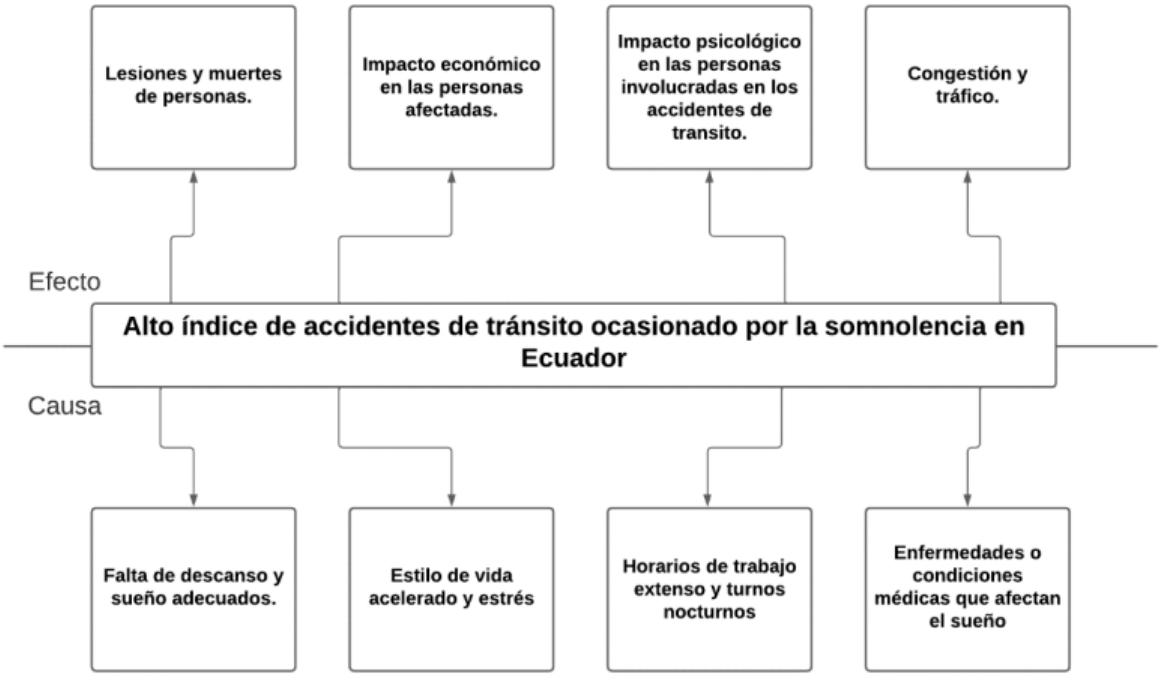
Problema

Según los Centros de Control y Prevención de Enfermedades, la conducción somnolienta es la combinación de conducción y somnolencia. Esto ocurre cuando un conductor de un vehículo no ha dormido lo suficiente, tiene trastornos del sueño sin tratar, está siendo medicado o se enfrenta a otros problemas que le causan somnolencia (Damashek, 2017). De acuerdo con la Organización Mundial de la Salud (OMS), el cansancio continúa siendo una de las principales causas de los siniestros, incluso presentó un Plan Mundial para el Segundo Decenio de Acción, con la finalidad de reducir en un 50% los siniestros de tránsito para el 2030 (El Telégrafo, 2022). Con respecto al ámbito nacional, el último reporte emitido por la Agencia Nacional de Tránsito (ANT), en lo que va del año se han registrado 121 incidentes de tránsito causados por conducir en estado de somnolencia o malas condiciones físicas, existiendo un aumento significativo de un 175% de incidentes de tránsito de este tipo en comparación con el año 2020 (ANT, 2023).

Los accidentes de tránsito causados por somnolencia representan un persistente problema social, y aunque no representa la causa principal de los accidentes de tránsito en el Ecuador, son accidentes que se pueden prevenir ya sea tomando recomendaciones como el descanso adecuado antes de conducir o utilizar la tecnología actual para crear sistemas que alerten a los conductores si se presentan síntomas de somnolencia en el mismo. La Figura 1, se muestra el árbol de problemas.

Figura 1

Árbol de problemas sobre el índice de accidentes de tránsito en Ecuador.



Fuente: Elaboración propia

Objetivos

Objetivo General

Detectar síntomas de somnolencia en conductores durante la conducción diurna y nocturna utilizando un sistema embebido basado en la red neuronal convolucional (CNN) ResNeXt-50.

Objetivos Específicos

- Elaborar un marco teórico sobre las bases de la somnolencia y las redes neuronales convolucionales: ResNeXt-50.
- Implementar un sistema embebido que integre la red neuronal convolucional ResNeXt-50 para la detección de somnolencia utilizando la metodología KDD.
- Implementar un sistema embebido que integre la red neuronal convolucional ResNeXt-50 para la detección de somnolencia utilizando la metodología KDD.

Alcance

Este trabajo tiene como propósito principal el desarrollo y la implementación de un sistema embebido que integre la arquitectura de red neuronal convolucional (CNN) ResNeXt-50 (Xie et al., 2016), para detectar y alertar síntomas de somnolencia en conductores durante la conducción diurna y nocturna. La arquitectura se muestra en la Figura 2. Se aplicará la metodología KDD para el desarrollo del proyecto (Fayyad, 1996).

Figura 2

Arquitectura ReNet-50 y ResNeXt-50.

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 128 3×3, 128, C=32 1×1, 256] ×3
conv3	28×28	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 256 3×3, 256, C=32 1×1, 512] ×4
conv4	14×14	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 512 3×3, 512, C=32 1×1, 1024] ×6
conv5	7×7	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 1024 3×3, 1024, C=32 1×1, 2048] ×3
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

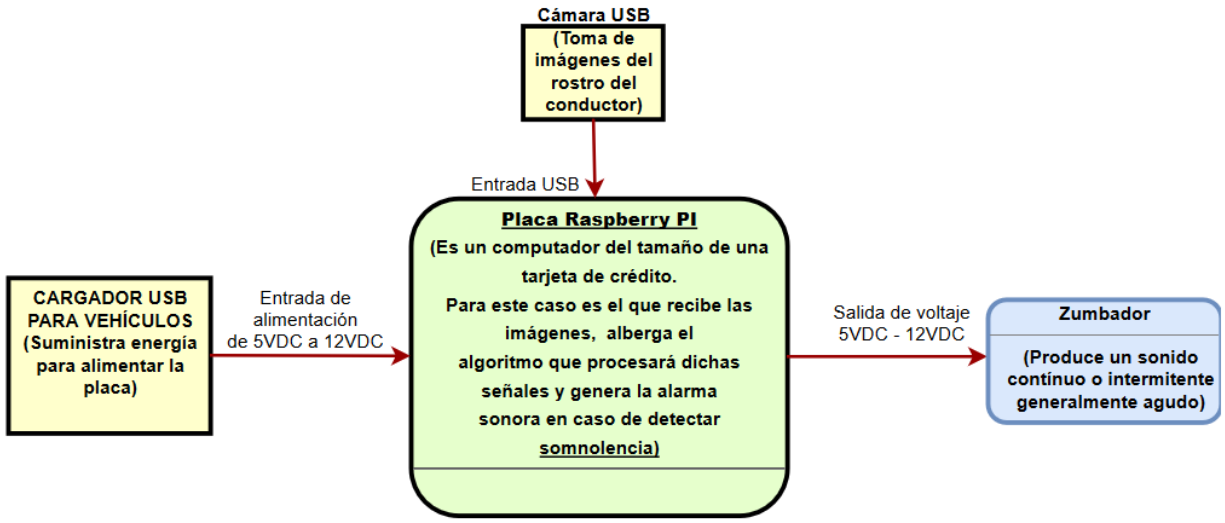
Fuente: (Xie et al., 2016)

La implementación del modelo ResNeXt-50, se llevará a cabo utilizando el lenguaje de programación Python y las bibliotecas TensorFlow y Keras, que son muy eficientes en el ámbito del aprendizaje profundo. El modelo se entrenará utilizando conjuntos de imágenes propias y disponibles en Internet, previamente elaborados y etiquetados, con el propósito de optimizar su desempeño y precisión en la detección de síntoma de somnolencia. El modelo entrenado se

integrará en un sistema embebido de hardware y software libre Jetson Nano y/o Raspberry Pi, además, el sistema integrará una cámara de video para la obtención de las imágenes y un zumbador para emitir una alarma. La estructura del sistema embebido se muestra en la Figura 3.

Figura 3

Estructura del sistema embebido.



Fuente: (Alba Neppas, 2020)

Se utilizarán métrica estándar en el campo de la inteligencia artificial: exactitud (accuracy), precisión, sensibilidad (recall), AUC-ROC, loss para evaluar el rendimiento del modelo ResNeXt-50 CNN, se tomará en cuenta el tiempo de respuesta del sistema como un indicador crítico para evaluar la eficiencia en situaciones simuladas y reales de conducción. Además, se utilizarán técnicas de estadística descriptiva para la representación de los datos.

Metodología

Se aplicará la investigación aplicada, que busca la generación de conocimiento con aplicación directa a los problemas presentes en la sociedad o el sector productivo. Se basa en los hallazgos tecnológicos de la investigación básica, ocupándose del proceso de enlace entre teoría y el producto (Lozada, 2014).

Técnicas de investigación empleadas para la recolección de datos:

- **Internet:** Obtención de información y materiales de trabajo (Dataset).
- **Fichaje:** Para registrar la información adquirida a través de la cámara de video que usaremos en el sistema.

- **Observación:** Realizar la observación del entorno donde funcionará el sistema embebido.
- **Pruebas de campo:** Evaluar el sistema en situaciones reales.

Usaremos los siguientes instrumentos de investigación:

- Mendeley para las citas bibliográficas que se realizará en todo el trabajo.
- Las bibliotecas TensorFlow y Keras.
- El lenguaje de programación Python en cualquier entorno de desarrollo.
- Placa Jetson Nano y/o Raspberry PI, para la integración del sistema embebido.
- Cámara de video para la captura de imágenes.
- Google Colab, entorno para el entrenamiento del modelo.

Se seleccionará la información más relevante de las diferentes bases de datos bibliográficas proporcionadas por la UTN, para crear un marco teórico detallado que incluirá los aspectos relacionados con el tema de investigación. Se usará el lenguaje de programación Python y las bibliotecas TensorFlow y Keras para desarrollar y entrenar el modelo de red neuronal ResNeXt-50. Posteriormente, integraremos el modelo en un sistema embebido para la captura y el procesamiento en tiempo real de las imágenes. Para las evaluaciones se usará métricas enfocadas al área de la inteligencia artificial, y pruebas simuladas y reales en entornos controlados. Además, se evaluará la característica de portabilidad, que es una de las métricas para la calidad interna y externa del software establecidas en la norma ISO 25023.

Justificación

Este trabajo pretende contribuir con el objetivo 9 de Desarrollo Sostenible (ODS): Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación. El objetivo tiene varias metas, y este trabajo se enfoca en la siguiente, “Apoyar el desarrollo de tecnologías, la investigación y la innovación nacionales en los países en desarrollo, incluso garantizando un entorno normativo propicio a la diversificación industrial y la adición de valor a los productos básicos, entre otras cosas” (Naciones Unidas, 2023).

Se pretende también, contribuir en el Plan de Creación de Oportunidades 2021-2025 de la República del Ecuador, en el objetivo 9 del Eje Seguridad Integral, donde se menciona: “Garantizar la seguridad ciudadana, orden público y gestión de riesgos”, y que trata en una de

sus políticas, el fortalecimiento de la seguridad de los sistemas de transporte terrestre y aéreo, promoviendo ambientes seguros (Secretaría Nacional de Planificación, 2021).

Justificación Tecnológica

El desarrollo de este sistema busca abordar los problemas y consecuencias causadas por accidentes relacionados con la conducción en estado de somnolencia.

Justificación Social

La implementación de este sistema en los vehículos podría contribuir a prevenir pérdidas económicas tanto como las personas que podrían verse afectadas como para los servicios de emergencia, limpieza y de reparación que podría requerirse en el lugar del accidente. Además, se podrían evitar traumas psicológicos, problemas de salud futuras a largo plazo y, en última instancia, salvar vidas de las personas que podrían verse involucradas en accidentes debido a este tipo de problema.

Capítulo 1

Marco Teórico

1.1. Somnolencia

La somnolencia, se cómo una sensación de pesadez y torpeza de los sentidos causada por el sueño, se trata de un fenómeno fisiológico que ocurre con frecuencia en determinadas situaciones. Un ejemplo común es su aparición durante los períodos de relajación tras una comida en las horas de la tarde. Distinguir entre un estado patológico, es decir, asociado a una enfermedad, y una inclinación natural hacia el sueño excesivo puede resultar complicado en muchos casos (Barbanoj et al., 2007).

Por el contrario, la somnolencia diurna excesiva se define como la tendencia a sentirse anormalmente somnoliento durante el día, que puede manifestarse en situaciones o momentos inoportunos. En consecuencia, una persona que experimenta somnolencia diurna excesiva es aquella que experimenta un deseo incontrolable de dormir, incluso en medio de actividades cotidianas o situaciones que demandan una elevada concentración (Escobar Cordoba, 2020).

Las principales causas de la somnolencia y la conducción somnolienta son la restricción y la fragmentación del sueño, sobre todo en personas que no padecen trastornos del sueño. La corta duración del sueño afecta negativamente la vigilancia y el rendimiento. El rendimiento en tareas relacionadas con la vigilancia se ve afectado si se duerme menos de 4 horas consecutivas cada noche, e incluso una sola noche de privación aguda del sueño provoca una somnolencia extrema (NHTSA, 1998).

La pérdida regular de sueño de 1-2 horas por noche acumula una "deuda de sueño" que, con el tiempo, conduce a una somnolencia crónica. Factores externos, como los horarios de trabajo, las obligaciones familiares y los compromisos escolares, así como factores internos, como las elecciones de estilo de vida, pueden limitar el tiempo disponible para dormir. Además, las exigencias de la presión económica y laboral en la sociedad contemporánea de 24 horas contribuyen a la privación del sueño. La limitación del sueño por motivos laborales, como trabajar muchas horas o varios turnos, provoca con frecuencia incidentes de conducción con somnolencia (NHTSA, 1998).

También existen situaciones medicas que pueden estar relacionados con la somnolencia, y se los puede clasificar en dos grupos: primarias y secundarias. El primer grupo, conocido como

de origen central, incluye la narcolepsia, la hipersomnia idiopática y otras hipersomnias primarias. Sin embargo, las causas secundarias se dividen en dos categorías diferentes: condiciones médicas y trastornos (Rosales Mayor & Rey De Castro Mujica, 2010).

El primer subgrupo se ocupa de trastornos relacionados con el sueño, como el Síndrome de Apneas-Hipopneas del Sueño (SAHS), así como de comportamientos que reducen el tiempo normal de sueño, como trabajar demasiado durante la noche. Por otro lado, el segundo subgrupo incluye una variedad de afecciones médicas, como traumas encefálicos, accidentes cerebrovasculares, enfermedades inflamatorias, enfermedades neurodegenerativas o trastornos psiquiátricos. También incluyen los efectos de algunos medicamentos, como las benzodiacepinas (Rosales Mayor & Rey De Castro Mujica, 2010).

1.1.1. Clasificación de la somnolencia

La somnolencia puede clasificarse como fisiológica o normal. Sin embargo, para considerarla como patológica o somnolencia diurna excesiva, debe persistir durante un periodo determinado, que puede variar desde semanas hasta meses. La clasificación de la gravedad de esta somnolencia se realiza de la siguiente manera:

- **Somnolencia leve:** Cuando no existen momentos de sueños involuntarios, sin embargo, la persona bosteza frecuentemente y su atención y concentración se ven alterados.
- **Somnolencia moderada:** Cuando el sueño se presenta en la persona sin que esta pueda controlarlo, y cuando la persona realiza actividades que no requieren demasiado trabajo físico.
- **Somnolencia grave:** Cuando la persona no se da cuenta de que se quedó dormida, incluso puede dormirse en situaciones donde se está realizando actividad física (Escobar Cordoba, 2020).

1.1.2. Riesgos de la somnolencia durante la conducción

La somnolencia o fatiga se erige como una amenaza significativa para la seguridad vial, desencadenando lesiones graves, pérdida de vidas y costos económicos considerables. El aumento de la somnolencia disminuye el rendimiento al conducir y aumenta la probabilidad de accidentes viales graves. La Administración Nacional de Seguridad del Tráfico en Carreteras de

los Estados Unidos (NHTSA) ha informado que la conducción somnolienta es responsable de casi 100,000 accidentes de tráfico y más de 1,500 muertes por año (Ramzan et al., 2019).

Así también, en Ecuador según los informes más reciente de la Agencia Nacional de Tránsito (ANT) revela que, hasta la fecha presente, se han documentado 121 incidentes de tránsito atribuibles a la conducción en estado de somnolencia o condiciones físicas deficientes durante el año en curso. Este dato refleja un marcado incremento del 175% en comparación con los incidentes registrados en el año 2020. Además, durante el año 2023, se han reportado 238 siniestros de tránsito por conducir en estado de somnolencia, representando el 1.1% de los siniestros totales, de los cuales, 138 personas resultaron heridas, que representa un 0.78% del total de personas heridas, y 25 personas resultaron fallecidas, que representa el 1.1% del total de personas fallecidas. Las provincias con más siniestros de tránsito por conducir en estado de somnolencia es Pichincha con 97 siniestros, seguido por Guayas y Manabí con 49 y 31 siniestros respectivamente (ANT, 2023).

Los accidentes de tránsito causados por la somnolencia son un problema social que persiste. Aunque no se consideran la principal causa de los accidentes de tránsito en Ecuador, sí representan una parte importante de ellos. Estos accidentes podrían evitarse mediante la adopción de prácticas como descansar adecuadamente antes de ponerse al volante o emplear la tecnología actual para implementar sistemas que alerten a los conductores ante posibles síntomas de somnolencia (ANT, 2023).

1.1.3. Como detectar la somnolencia

Existen diferentes maneras de poder detectar la somnolencia en un conductor, puede ser mediante características faciales, debido a que están directamente relacionado con su estado corporal, es decir, si el conductor se encuentra cansado o con sueño, se va a ver reflejado en los rasgos faciales, y no van a ser los mismo que cuando el conductor se encuentra en un estado normal (activo, sin sueño). También puede ser detectado mediante su fisiología, debido a que cambia según el estado en que se encuentre, y mediante parámetros fisiológicos se puede determinar si un conductor se encuentra en estado de somnolencia o fatiga. Todos estos parámetros pueden se obtenidos mediante sensores como cámaras, electroencefalograma,

electromiografía, electrocardiograma, sensores de pulso y frecuencia respiratoria (Shi et al., 2017).

- **Indicadores oculares:** Son una opción muy eficiente actualmente y se han realizado diferentes trabajos utilizando estos indicadores, como son la frecuencia y la duración que permanecen abiertos los parpados, así como el incremento en el porcentaje de cierre de los ojos. Estos parámetros pueden se registrados mediante cámaras, así como rasgos faciales típicos de un conductor somnoliento, como bostezos o movimientos de cabeza. Debido a que estos cambios son consistentes, la detección de somnolencia puede ser muy precisa (Perrotte et al., 2024).
- **Indicadores fisiológicos:** Los indicadores fisiológicos como la frecuencia cardiaca (FC) y la variabilidad en la frecuencia cardiaca (VFC) son importantes para identificar etapas iniciales del sueño, debido a que disminuye la frecuencia cardiaca y disminuye la ventilación. La variabilidad que surge de las fluctuaciones en los intervalos entre latidos refleja la actividad del sistema nervioso simpático y el sistema nervioso parasimpático. Estos indicadores fisiológicos proporcionan información valiosa sobre el estado de alerta y somnolencia de una persona, lo que es especialmente relevante en contextos como la detección de somnolencia al volante (Perrotte et al., 2024).
- **Indicadores posturales:** Son otra forma de detectar la somnolencia mientras se conduce. Los conductores con síntomas de somnolencia presentan cambios de comportamiento como bostezos, gestos egocéntricos y cambios de posición, es decir, los movimientos, la posición y la postura en el asiento son indicadores de somnolencia. De esta manera, al implementar sensores en los asientos como los de presión, podemos obtener un monitoreo no invasivo del conductor y evaluando estos sensores se puede detectar la somnolencia de forma efectiva (Perrotte et al., 2024).

La detección de la somnolencia en los conductores con gafas presenta algunos desafíos y consideraciones particulares. El uso de gafas puede afectar el método de detección de somnolencia que trabaja con indicadores oculares. Se han realizado avances significativos en el campo de la visión por computadora y el procesamiento de imágenes para abordar el desafío de reconocer características faciales a través de obstáculos como gafas o reflejos. Se ha creado un método para eliminar los reflejos de las imágenes faciales. A pesar de la presencia de reflejos,

este proceso implica estimaciones y refinamientos para preservar las características faciales importantes. Para compensar la falta de información visual directa causada por reflejos o gafas, este método utiliza una combinación de técnicas de redes neuronales y algoritmos especializados (Wan et al., 2021).

1.2. Redes neuronales convolucionales

1.2.1. Redes neuronales convolucionales (CNN)

Los modelos informáticos que contienen diferentes capas de procesamiento pueden aprender representaciones de datos con diferentes niveles de abstracción gracias al Aprendizaje Profundo (DL) Se distingue por descubrir estructuras complejas en grandes conjuntos de datos mediante algoritmos de retro propagación. Estos algoritmos indican cómo una máquina debe ajustar sus parámetros internos basándose en la representación de la capa anterior (LeCun et al., 2015).

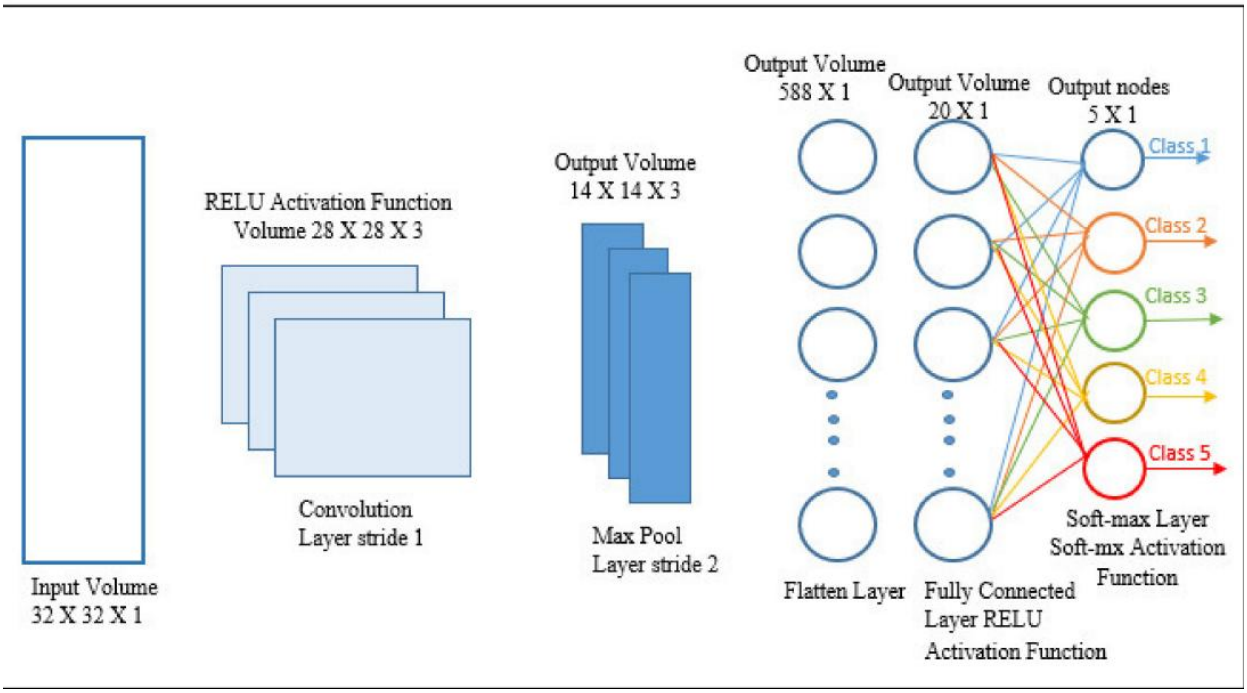
Las Redes Neuronales Convolucionales (CNN) son modelos de aprendizaje profundo utilizados en visión por computadora, diseñados para reconocer y clasificar imágenes. Las Redes Neuronales Convolucionales (CNN) Rede se utilizan ampliamente en tareas de clasificación de imágenes porque pueden procesar patrones visuales complejos (Bhatt et al., 2021).

En la actualidad, las CNN son consideradas como técnicas de aprendizaje automático ampliamente utilizadas en aplicaciones relacionadas con la visión por computadora. Su funcionamiento se basa en el proceso visual del cerebro humano: cada neurona en nuestro cerebro se encarga de procesar una parte específica de lo que observamos. Al estar interconectadas, estas neuronas generan colectivamente nuestro campo visual. De manera análoga, las capas de una CNN analizan datos en sus áreas respectivas y están programadas para detectar patrones que varían desde los más simples, como líneas y curvas, hasta los más complejos, como rostros y objetos. Este enfoque se inspira en la estructura y funcionamiento del sistema visual humano, permitiendo a las CNN destacarse en tareas de reconocimiento visual y clasificación de imágenes (A. Khan et al., 2020).

A continuación, se explicarán los componentes generales que integran las CNN, tal como se muestra en la Figura 4, así como su funcionalidad.

Figura 4

Arquitectura de las Redes Neuronales Convolucionales (CNN).



Fuente: (Bhatt et al., 2021).

- **Imagen de entrada (Data Input):** Una imagen está conformada por píxeles que son la representación binaria de los datos visuales. Estos píxeles se forman en una matriz y cada uno tiene una representación de 0 a 255; este valor especifica su brillo y tono (Bhatt et al., 2021).
- **Capa de convolución (Convolution Layer):** Está compuesta por grupos convolucionales y cada neurona actúa como un núcleo. El núcleo convolucional divide la imagen en sectores pequeños sectores conocidos como campos receptivos; esta división ayuda a la extracción de las características.
- **Capa de agrupación (Pooling Layer):** Reduce la dimensionalidad de las características extraídas en la capa convolucional, reduce la cantidad de información innecesaria y el procesamiento de datos se vuelve más rápido.
- **Función de activación (Activation Function):** La función de activación se encarga de concluir la salida de las redes neuronales, como si o no

- **Capa totalmente conectada:** Operación global que utiliza una red neuronal feed-forward al final de la red para la clasificación. Esta capa recibe como entrada la salida de la capa convolucional o agrupación, y debe ser aplanada antes de ser entregada como entrada. El aplanamiento más conocido como Flattening se trata de pasar la matriz obtenida de la capa de convolución o agrupamiento a un vector (A. Khan et al., 2020)

1.2.2. Redes de memoria a largo plazo (LSTM)

Las Redes de Memoria a Largo Plazo (LSTM), son un tipo de Redes Neuronales Recurrentes (RNN) que soluciona la degradación de rendimiento de los RNN y tiene la capacidad de manejar dependencias a largo plazo, es decir, toma en cuenta neuronas anteriores y recibe su información. Este tipo de redes neuronales, constan de tres partes principales: a) El modelo es entrenado con el fin de olvidar información irrelevante, b) El modelo aprende la nueva información, y c) El modelo devuelve nueva información actualizada del paso del tiempo actual al siguiente. Se los usa para modelar datos secuenciales y aprender comportamientos humanos (Goswami et al., 2023; Hochreiter & Schmidhuber, 1997).

1.2.3. Vision Transformer (ViT)

Vision Transformer, es un modelo que adapta la arquitectura Transformer para procesar datos de imágenes. La arquitectura divide la imagen en parches, estos parches se transforman en un vector 1D y son enviados a un conjunto de bloques Transformer, donde cada uno contienen múltiples mecanismos de atención. Finalmente, en la capa de salida se realiza el proceso de clasificación de la imagen. Este modelo es utilizado normalmente para problemas de clasificación de imágenes (Dosovitskiy et al., 2020).

1.2.4. VGG-16

Se trata de una Red Neuronal Convolucional (CNN), desarrollado por el grupo Visual Geometry Group (VGG) en la Universidad de Oxford. Consta de filtros convolucionales que contiene un campo receptivo de 3x3, tiene múltiples capas convolucionales seguidas de capas de agrupamiento (max-pooling), y tres capas totalmente conectadas al final de la red para realizar la tarea de clasificación. Es una arquitectura muy utilizada en el área de visión por computadora

que extrae características de imágenes para tareas de clasificación y regresión (Sarker et al., 2023; Simonyan Karen & Zisserman Andrew, 2014).

1.2.5. Inception-V3

Inception-V3, es un modelo de CNN diseñado para tareas de reconocimiento de imágenes, este modelo se caracteriza por utilizar módulos iniciales que realiza múltiples convoluciones de distintos tamaños de forma paralela seguida de la concatenación de sus salidas, de esta manera se aprovecha información local y global, la factorización reemplaza convoluciones grandes de 7x7 en tres convoluciones de 3x3, lo que reduce el costo computacional, y clasificadores auxiliares. La red cuenta con 42 capas de profundidad y se ha demostrado que es más eficiente que VGGNet (M. N. Khan et al., 2024; Szegedy Christian et al., 2016).

1.2.6. DenseNet

Es una arquitectura CNN profunda, que crea bloques densamente conectados para resolver el problema del gradiente que desaparece en redes neuronales muy profundas. De esta manera la arquitectura está formada por varios bloques densamente conectados, y en cada bloque se encuentran capas convolucionales apiladas con la misma cantidad de filtros de tal forma que la salida de cada capa es enviada como entrada para las capas posteriores. La capa final de cada bloque se conecta a una capa de transición que está compuesta por una capa de normalización por lotes, una capa convolucional 1x1 y una capa de agrupación. Con las conexiones densas y el uso eficaz de las funciones, la propagación de información y el uso del gradiente mejoran (Saleh et al., 2023).

1.2.7. ResNet

Con el paso del tiempo, los modelos de Deep Learning han ido mejorando con la implementación de más redes neuronales profundas, lo que permite la extracción de características más complejas, y así obtenemos un modelo muy potente con una gran precisión. Sin embargo, mientras se entrenan modelos con redes más profundas, aparece un problema llamado gradiente desvaneciente; este problema provoca la disminución de la exactitud del modelo (V. & Bhattacharyya, 2022). Como resultado, se creó una nueva red llamada ResNet

(Residual Network) en 2015. Esta red introdujo el término "aprendizaje residual profundo" para abordar el problema de degradación. Esto significa que, mientras otras redes esperan que algunas capas apiladas ajusten un mapeo subyacente deseado, ResNet permite que estas capas ajusten un mapeo residual. Utilizan conexiones denominadas “shortcut”, que se encarga de omitir una o más números de capas, además, no añaden parámetros ni complejidad a los modelos (He et al., 2015). En la Figura 5, podemos observar diferentes modelos de ResNet.

Figura 5

Arquitecturas ResNet.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fuente: (He et al., 2015)

Las redes residuales implementan la conexión shortcut, lo que la convierte en este tipo de red, esta conexiones se aplica de diferente forma dependiendo de si las dimensiones de entrada son iguales o si las dimensiones aumenta, en el primer caso estas conexiones se los puede usar directamente, sin embargo, en el segundo caso se considera dos opciones: a) La conexión shortcut se mantiene realizando mapeos de identidad con entradas adicionales, se aplica para dimensiones crecientes, además, esta opción no aumenta el número de parámetros, b) Se aplica shortcut de proyección aplicando convoluciones de 1x1 (He et al., 2015).

1.2.8. ResNeXt-50

ResNeXt-50 (Redes Residuales con Cardinalidad) es una arquitectura de red neuronal basada en ResNet (Redes Residuales), diseñada para mejorar el rendimiento de las redes neuronales convolucionales, especialmente en tareas de clasificación de imágenes. Esta arquitectura introduce un nuevo bloque denominado "cardinalidad", que se refiere al tamaño del conjunto de transformaciones aplicadas en paralelo. Al aumentar la cardinalidad, la arquitectura se vuelve capaz de capturar un conjunto más diverso de características, mejorando así el rendimiento general de la red (Xie et al., 2016). En la Figura 6, podemos observar la arquitectura de la ResNeXt-50.

Figura 6

(izquierda) arquitectura ResNet-50. (Derecha) ResNeXt-50.

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix}\times6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

Fuente: (Xie et al., 2016).

Se destaca por su diseño modularizado, adoptando la estrategia de VGG/ResNet. Esta red se compone de una serie de bloques residuales, y su estructura sigue dos reglas basadas en

VGG/ResNet. La primera regla establece que, si los bloques están produciendo mapas espaciales del mismo tamaño, comparten los mismos hiperparámetros, como el ancho y los tamaños de filtro. La segunda regla asegura que, cada vez que el tamaño del mapa espacial se reduce a la mitad, el ancho de los bloques se multiplica por un factor de 2, manteniendo así la complejidad computacional, medida en FLOPs, de manera aproximadamente constante para todos los bloques (Xie et al., 2016).

Ventajas

- Es una arquitectura CNN que está estructurada de manera modular, de manera que cada capa puede ser modificada o reemplazada fácilmente sin afectar su funcionamiento (Hira et al., 2021).
- Esta arquitectura tiene la capacidad de mejorar el rendimiento de la red neuronal al mismo tiempo que reduce el número de hiperparámetros (C. Deng et al., 2023).
- La arquitectura ResNext-50 reduce la perdida de información, la exposición de gradiente y la perdida de gradiente, y a su vez aumenta la precisión de la red sin cambios significativo en los parámetros y la complejidad computacional (Cheng et al., 2024).

Desventajas

- ResNeXt presenta varias desventajas, como la creciente dificultad de diseñar arquitecturas con un número cada vez mayor de hiperparámetros, los factores de complicación asociados a los modelos Inception y la saturación de la precisión al aumentar la cardinalidad a costa de reducir la amplitud. Además, preocupan la conservación aproximada de la complejidad y las posibles limitaciones debido a la complejidad del conjunto de datos (Xie et al., 2016).

1.2.9. Cuando y en que tipos de problemas es recomendable aplicar la arquitectura ResNeXt

La arquitectura en sus variantes ResNeXt-50 y ResNeXt-101 son muy efectivas en problemas relacionados con la visión por computadora y el aprendizaje profundo, se distinguen principalmente por el número de capas, siendo 50 y 101 respectivamente, además, esta última al tener más capas es más potente en el aprendizaje, sin embargo, requiere mayores recursos computacionales a diferencia del modelo que tiene 50 capas, que es más equilibrado entre eficiencia computacional y precisión. Su aplicación es variada, como por ejemplo clasificación de imágenes, detección de objetos, segmentación semántica, transferencia de aprendizaje,

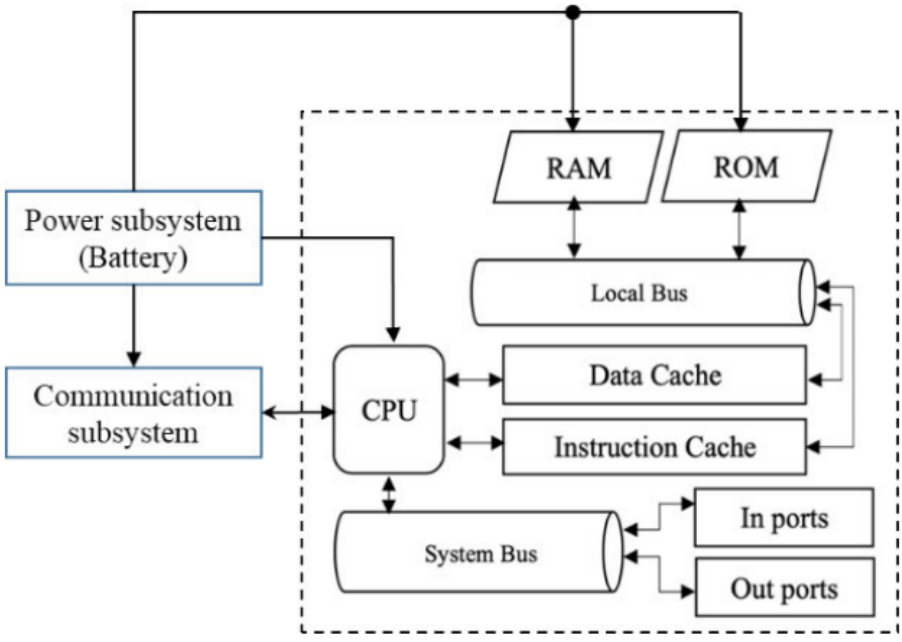
reconocimiento de patrones en datos complejos y análisis de video. Además, se recomienda usar ResNeXt cuando se disponga de Datasets con grandes números de datos y que se busque mejoras de precisión en tareas específicas (Xie et al., 2016).

1.3. Sistemas embebidos

Se trata de sistemas informáticos integrado y diseñado para funciones específicas. Son una combinación de software y hardware y, de ser necesario, piezas mecánicas, por ende, usar el termino sistema embebido hace referencia a cualquier sistema informático que no sea PC de uso general u ordenadores centrales. Su arquitectura puede variar dependiendo del fin que se requiera, por lo general estos sistemas cuentan con CPU, RAM, ROM y algunos puertos de entrada y de salida dependiendo del modelo. En la Figura 7, podemos observar la arquitectura típica de los sistemas embebidos (Aloseel et al., 2021).

Figura 7

Arquitectura común presentes en los sistemas embebidos.



Fuente: (Aloseel et al., 2021).

El diseño abarca desde pequeños circuitos electrónicos, pasando por microcontroladores con un pequeño número de transistores y una capacidad de 8 bits, hasta microprocesadores de 64

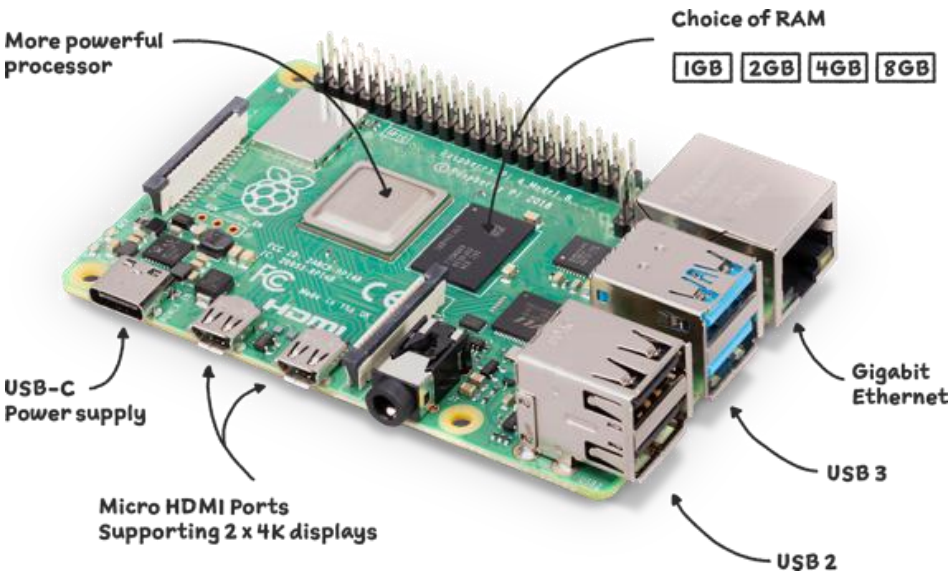
bits con múltiples núcleos y velocidades superiores a 1 GHz. También se utilizan diversas implementaciones y arquitecturas de CPU específicas para aplicaciones, como núcleos FPGA blandos o duros, procesadores de señal digital (DSP) e incluso núcleos recientes optimizados para aprendizaje automático. Un sistema embebido puede intercambiar información con dispositivos externos a través de puertos de entrada/salida (I/O). El subsistema de energía suministra la potencia a los componentes del sistema integrado, siendo muchos de ellos alimentados por baterías (Aloseel et al., 2021).

1.3.1. Raspberry Pi

La Fundación Raspberry Pi es una organización benéfica del Reino Unido que desarrolló una computadora de placa única (SBC) económica. Desde su lanzamiento en 2012, está disponible en varias generaciones, incluyendo modelos como Raspberry Pi A, B y Zero. Estos modelos comparten un sistema en un chip (SoC) que incluye una CPU integrada, GPU, memoria a bordo y una entrada de energía de 5 V DC. Con puertos para cámara, pines GPIO para la interacción con dispositivos electrónicos, conectividad Ethernet y Wi-Fi/Bluetooth, la Raspberry Pi puede funcionar como una computadora estándar con sistemas operativos como Linux, Windows 10 IoT y Android. A diferencia de un microcontrolador, la Raspberry Pi es versátil y puede ser utilizada tanto con dispositivos periféricos como de manera remota, proporcionando una amplia gama de aplicaciones (Jolles, 2021). En la Figura 8, se observa el modelo 4 que se trata de la versión más reciente de Raspberry Pi.

Figura 8

Raspberry Pi Modelo 4.



Fuente: (Raspberry Pi Foundation, 2024)

1.3.2. Jetson Nano

El Jetson Nano de NVIDIA emerge como una plataforma altamente efectiva en la fase inicial de visión artificial y computacional debido a su destacada capacidad de procesamiento de bajo consumo, consolidándose como uno de los líderes en el desarrollo de hardware para inteligencia artificial en el ámbito de la visión por computadora. La arquitectura de unidad de procesamiento central (CPU) y unidad de procesamiento gráfico (GPU) del Jetson Nano, posibilita una carga más rápida en la CPU, mientras que la GPU ejecuta sin contratiempos las complejas técnicas de aprendizaje automático. Su diseño elegante, portabilidad y eficiencia energética lo convierten en la elección ideal para aplicaciones que presentan limitaciones en cuanto a peso y consumo de energía, e la Figura 9, se observa el modelo Jetson Nano más reciente publicado en su página oficial (Al-Selwi et al., 2023).

Figura 9

Placa Jetson Nano.

Fuente: (NVIDIA Corporation, 2024)

1.4. ISO 25023 y Metodología KDD

1.4.1. Norma ISO/IEC 25000

La conformidad con las normas ISO/IEC 9126 y ISO/IEC 14598, también conocida como SQuaRE (Requisitos de calidad de sistema y software y evaluación). La norma ISO/IEC 2500n es una familia de normas para evaluar productos de software que incluyen modelos, métricas, procesos y herramientas (ISO 25000, 2022; Roa et al., 2015). En la Figura 10, observamos la familia de la norma ISO/IEC 2500n.

Figura 10

Familia Norma ISO/IEC 2500n.

Fuente: (ISO 25000, 2022)

1.4.2. ISO/IEC 2502n: Medición de la calidad

Este conjunto incluye un modelo de referencia para la calidad de los productos de software, definiciones matemáticas de las métricas de calidad y una guía para su aplicación adecuada. Además, ofrece instrucciones detalladas sobre cómo usar estas métricas para evaluar la calidad interna, externa y de uso del producto de software (Balseca, 2014).

1.4.3. Modelo de calidad del producto de software (Portabilidad)

La norma ISO/IEC 25023 establece medidas de calidad para evaluar la calidad de sistemas y productos de software de manera cuantitativa, utilizando características y subcaracterísticas definidas en ISO/IEC 25010. Es diseñado para trabajar con ISO/IEC 25010, pero también puede trabajar con ISO/IEC 2503n e ISO/IEC 2504n para satisfacer las necesidades del usuario en cuanto a la calidad del producto o sistema de software (ISO, 2022). Este trabajo utilizará la característica de portabilidad, que evalúa la capacidad de un componente de software para cambiar de entorno sin afectar la funcionalidad de cada sistema. Esta característica se subdivide como se muestra en la Tabla 1.

Tabla 1

Característica de portabilidad ISO 25023.

Portabilidad	**Adaptabilidad**	• Adaptabilidad en entorno hardware. • Adaptabilidad en entorno de software. • Adaptabilidad en entorno organizacional.
	Capacidad de ser Instalado	• Eficiencia en el tiempo de instalación. • Facilidad de instalación.
	Capacidad de ser Reemplazado	• Consistencia en la función de soporte al usuario. • Inclusividad funcional. • Uso continuo de datos.

Fuente: (Balseca, 2014)

1.4.4. Metodología KDD

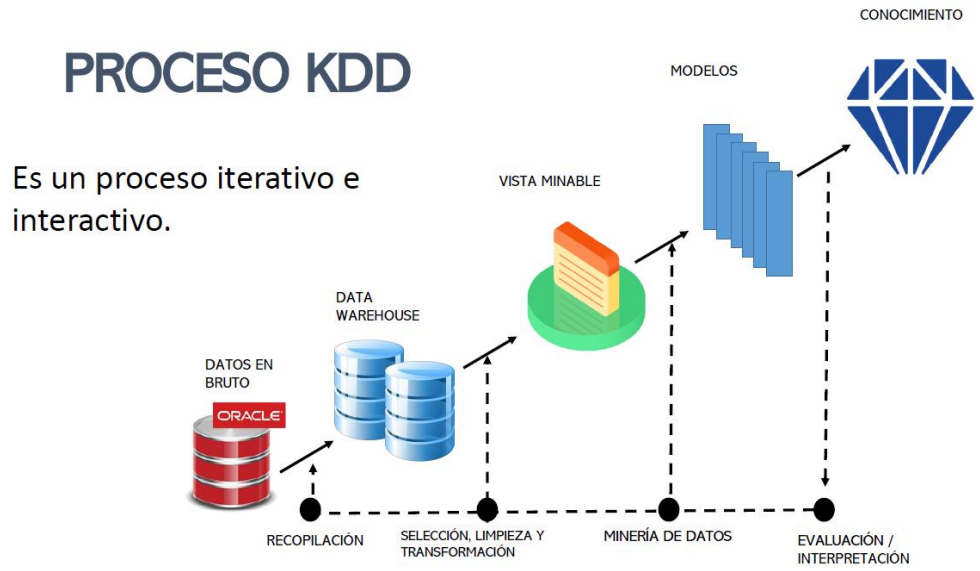
El proceso de minería de datos consiste en el análisis de grandes conjuntos de datos con el propósito de encontrar patrones y relaciones que son utilizados para la innovación y el beneficio para quien posee los datos. La abundante disponibilidad de una gran cantidad de datos y la necesidad de transformarlos en información y conocimientos valiosos han llevado a este método como un resultado natural del avance de las tecnologías de la información. La metodología de Descubrimiento de Conocimientos en Bases de Datos (KDD) es un proceso interactivo e iterativo que combina la experiencia en problemas con varias técnicas de análisis de datos y tecnologías de aprendizaje automático avanzadas. Para crear predicciones confiables, su principal objetivo es extraer patrones y asociaciones de los datos. (Fayyad, 1996; Gervilla García et al., 2009).

Esta metodología está dividida en varias fases muy importantes, las cuales ayudarán a obtener una investigación de calidad y una síntesis adecuada de conceptos. A continuación, se

definirá cada una de las fases que se muestran en la Figura 11, donde se muestra el proceso iterativo e interactivo de la metodología.

Figura 11

Fases de la metodología KDD.



Fuente: (Fayyad, 1996)

- **Fase de selección:** Con los conocimientos relevantes identificados, se crean conjuntos de datos objetivos y se selecciona una muestra representativa para realizar el trabajo de descubrimiento.
- **Fase de preprocesamiento:** Se realiza un análisis de los datos que fueron previamente seleccionados en la etapa anterior. Se aplican operaciones de eliminación de datos no válidos, estrategias para datos desconocido y estadística para la sustitución de datos faltantes en caso de ser necesario. En el proceso de limpieza, se eliminan o reemplazan datos incompletos o inconsistentes, debido a que pueden generar resultaos incorrectos durante el análisis.
- **Etapa de transformación:** Los datos obtenidos pasan por un proceso llamado reducción de datos, donde se buscan características importantes para la empresa. Se lo debe realizar en una tabla, ya sea de forma horizontal o vertical. La forma horizontal elimina datos

comparables y la forma vertical elimina datos innecesarios, datos de baja importancia o redundantes.

- **Fase de extracción de datos:** En esta etapa el objetivo principal establecido en la metodología KDD debe establecer como un objeto de minería de datos. Pero antes se debe realizar un análisis preliminar para establecer hipótesis y posteriormente seleccionar la secuencia o algoritmo para la minería de datos.
- **Interpretación:** Terminado la fase anterior, se pueden volver a evaluar las fases anteriores con el fin de identificar una iteración que obtenga mejores resultados. Aquí también se evalúa la correcta funcionalidad de los algoritmos (Gervilla García et al., 2009).

1.5. Trabajos relacionados

(Phan et al., 2023) propusieron un método para la detección de somnolencia utilizando datos extraídos de cámaras de vigilancia, considera redes neuronales LSTM, VGG-16, Inception-V3 y DenseNet. Los modelos se integran en un sistema IoT inteligente. Utilizaron videos de cámaras de vigilancia para extraer los datos, luego realizaron la detección de rostros con la red Single Shot MultiBox Detector, normalizaron las imágenes a un tamaño de 244x244, y finalmente dividieron los datos en subconjuntos (somnolientos y no somnolientos), obteniendo el conjunto de datos con el que se entrenaron los modelos. La red neuronal profunda que obtuvo la mejor precisión en el entrenamiento y las pruebas fue DenseNet con un 98% de precisión. Los fallos que surgen en el dispositivo IoT debido al entorno pueden suponer una grave limitación.

(Li & Li, 2024) propusieron el desarrollo de un modelo de aprendizaje profundo multigranular Practical Facial Landmark Detector (PFLD) y LSTM para la detección de somnolencia en conductores. Las características de los ojos, la boca y la cabeza se obtienen mediante el Detector Práctico de Puntos de Referencia Faciales (PFLD), la información semántica se obtiene con un modelo Transformador de Visión (ViT) y, a continuación, una red de Memoria Larga a Corto Plazo (LSMT) utiliza las características de granularidad obtenidas como entrada para profundizar aún más en las pistas de somnolencia. El modelo se entrena y evalúa utilizando el conjunto de datos NTHU-DDD (National Tsuing Hua University, 2016). El modelo LSMT obtuvo una precisión del 93,15% en comparación con otros modelos como RF (83,49%), SVM (85,19%) y

MLP (85,34%). Existen limitaciones como: dificultad para obtener datos y el coste elevado en la etiquetación de datos.

(Ahmed et al., 2023) propusieron desarrollar un sistema de detección de somnolencia utilizando técnicas de visión artificial que identifican la cara del conductor y técnicas de aprendizaje profundo para clasificar si el conductor está somnoliento o no. Utilizaron conjunto de datos (Perumandla Dheeraj, 2020) de 2900 imágenes disponible en Kaggel, utilizaron el clasificador Haar Cascade para detectar la cara del conductor, entrenaron un modelo CNN desarrollado por los propios autores para detectar el estado de somnolencia del conductor y la arquitectura VGG16. Se obtuvo una tasa de precisión del 97% para el modelo desarrollado por los autores y del 74% para el modelo VGG16 personalizado. La limitación de este estudio se debe a que el conjunto de datos público no contiene imágenes de calidad y sólo contiene caras descubiertas.

(Pandey & Muppalaneni, 2023) propusieron un modelo para la detección de la somnolencia desde sus primeras fases, de modo que el conductor tenga tiempo de recuperarse. Se utilizó el detector RIO (versión modificada de YOLOv3) para capturar los ojos y la boca de la cara, se utilizó el conjunto de datos UTA-RLDD (Ghoddoosian et al., 2019), que contiene conjuntos de datos masivos sobre somnolencia, y se realizó un aumento de datos (DA) mediante redes adversariales generativas (GAN) para evitar el sobreajuste del modelo. El estudio propone dos modelos, uno basado en características temporales (TCBR) que se centra en la visión por ordenador y extrae características faciales que van a la LSTM para identificar el nivel de somnolencia, y otro en características espaciotemporales (CNN-LSTM) donde la CNN extrae características espaciales y luego las envía a la LSTM para su clasificación. El modelo CNN-LSTM obtuvo una precisión del 97,50%, siendo superior al modelo TCBR (86,00%). La escasa cantidad de datos para el entrenamiento y las pruebas, la variabilidad en el tamaño de los ojos y la boca, y los problemas de detección en condiciones específicas son algunas de las limitaciones.

Capítulo 2

Desarrollo

2.1. Recopilación y tratamiento de imágenes

2.1.1. Planificación

El equipo que integran este trabajo de investigación se muestra en la Tabla 2.

Tabla 2

Integrantes del trabajo.

Miembro	Grupo	Rol
Marco Inlago	Tesista	Desarrollador
PhD. Iván García	Director	Testeador

Fuente: Propia

2.1.2. Creación de Datasets

En el desarrollo de modelos de aprendizaje profundo, especialmente en aquellos que están diseñados para el procesamiento de imágenes, requieren accesos a grandes volúmenes de datos. Por lo tanto, la creación de un Dataset adecuado es importante para entrenar y ajustar el modelo al problema que se desea resolver.

Para este trabajo se ha creado un Dataset específico para la detección de 12 puntos de referencia facial pertenecientes a los puntos de los dos ojos, y para clasificación de los rostros en dos clases despierto (awake) y dormido (drowsiness), que lo utilizaremos para el entrenamiento del modelo ResNeXt-50. El modelo obtenido posteriormente nos ayudara a la detección de somnolencia, en la Tabla 3, se listan las herramientas que se utilizó, cada uno con sus respectivos nombres y su funcionalidad.

Tabla 3

Materiales utilizados para la creación del Dataset.

Imagen	Nombre y función
	**Teléfono** Teléfono para la captura de los videos en el día.

	**Cable USB a USB c** Para la conexión entre el teléfono y la computadora portátil.
	**Computadora portátil** Para el uso de la aplicación de captura de video y para el almacenamiento de estos.
	**Placa Raspberry Pi** Para la captura de videos con la cámara NIR.
	**Cámara OV5647** Para la captura de video NIR en el día y noche.
	**Memoria microSD** Para el almacenamiento del Sistema Operativo Raspberry.
	**Fuente de 5V** Para alimentar a la Raspberry en vehículos.

Fuente: Elaboración propia

2.1.2.1. Grabación de los videos

Para la creación del Dataset, la grabación de videos fue muy importante. Utilizando las herramientas especificadas en la Tabla 3, se procedió a capturar videos de 20 personas en un entorno controlado. Las grabaciones se realizaron dentro de un vehículo. Esta actividad se realizó tanto en condiciones diurnas como nocturnas para captar variaciones lumínicas. A continuación, se especifican los detalles de las grabaciones:

- **Duración:** Cada sesión de grabación tuvo una duración de 60 segundos, capturando el video a una tasa de 25 fotogramas por segundo (FPS). Este intervalo de 60 segundos era suficiente para recoger las expresiones faciales necesarias para nuestro Dataset.

- **Condiciones de iluminación y equipamiento:** Para el trabajo se necesitó diferentes condiciones de iluminación y se optó por realizar las grabaciones tanto en el día y en la noche para tener una variedad de iluminación en las imágenes.

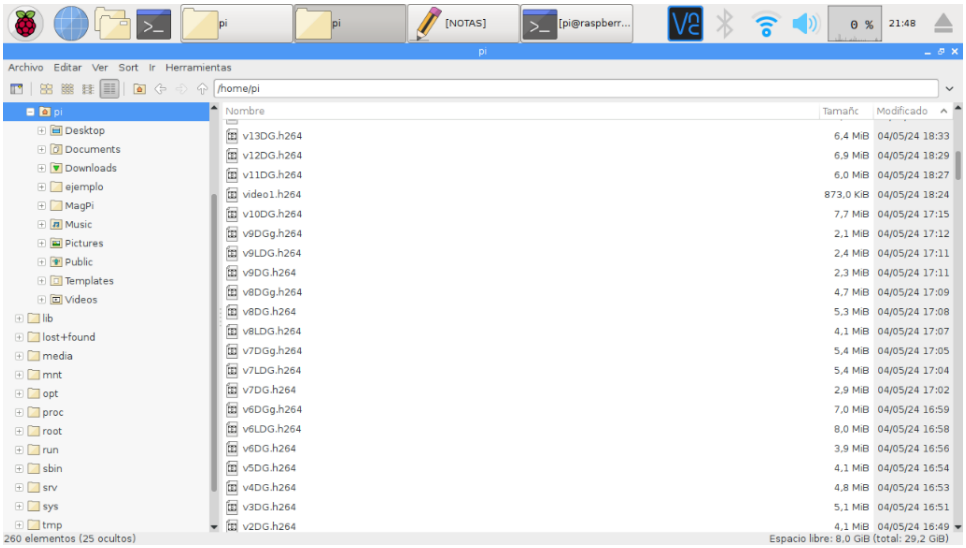
  Día: Durante las grabaciones diurnas, se utilizó la cámara de un teléfono móvil de 12 megapíxeles a una resolución de 1280x720, así también como la cámara NIR (infrarroja cercana).

  Noche: Durante las grabaciones nocturnas, se utilizó una cámara NIR de 12 megapíxeles a una resolución de 720x460, la cual es muy efectiva en condiciones de baja iluminación, asegurando así la captura de imágenes claras y detalladas del rostro.
- **Variaciones en la apariencia:** Cada persona fue grabada bajo tres condiciones distintas de accesorios faciales: sin accesorios (bareface), con lentes (glasses) y con gafas (sunglasses). Es importante mencionar que las gafas de sol utilizadas eran lo suficientemente claras para permitir la visibilidad de los ojos, aspecto importante para la detección de puntos de referencia facial.

Las grabaciones se realizaron con la cámara de un teléfono y la cámara NIR durante el día y solo con la cámara NIR durante la noche. No obstante, al grabar con la cámara NIR los videos se guardaron en formato h264 como se observa en la Figura 12, el cual no era compatible para reproducirse en el sistema operativo de Windows. Para solucionar esto, fue necesario convertir los archivos en formato h264 a mp4. Para eso se utilizó el comando '*MP4Box -add video1.h264 video1.mp4*' como se muestra en la Figura 13. '*MP4Box*' es el nombre de la herramienta diseñada para el procesamiento de archivos tipo ISOBMF, tales como MP4 y 3GP. Utilizado también para importar y exportar medios desde archivos contenedores como AVI, MPG y MKV. La opción '*-add*' es un comando de '*MP4Box*' que indica la adición de contenido a un archivo contenedor. '*video1.h264*' se refiere al archivo de entrada que se quiere empaquetar, en este caso, un archivo codificado en formato h264. '*video1.mp4*' es el archivo de salida; en otras palabras, '*MP4Box*' debe tomar el flujo de video en '*video1.h264*' y empaquetarlo en un archivo MP4 denominado '*video1.mp4*'.

Figura 12

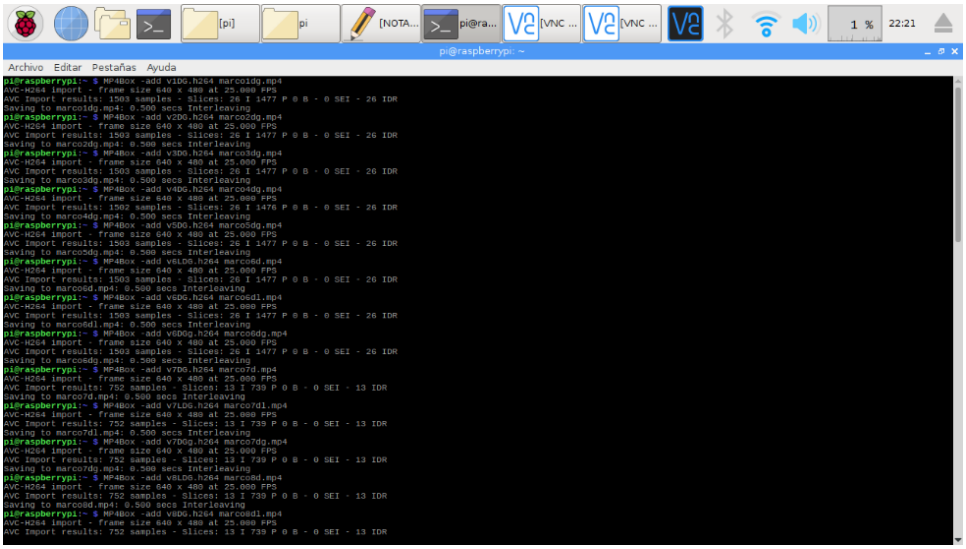
Videos en formato h264.



Fuente: Elaboración propia

Figura 13

Transformación de videos de h264 a mp4.

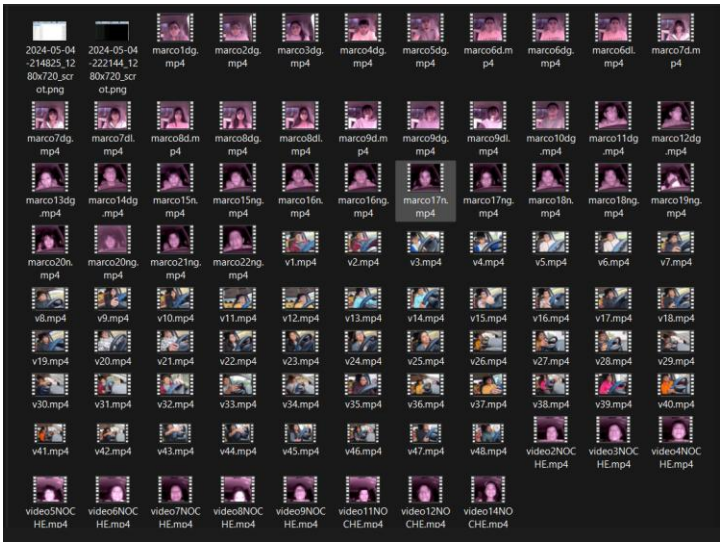


Fuente: Elaboración propia

Con las grabaciones realizadas tanto en el día como en la noche y con las herramientas mencionadas en la Tabla 3, aumentamos el Dataset para mejorar el entrenamiento del modelo ResNeXt-50. En la Figura 14, se muestran todos los videos realizados.

Figura 14

Lista de grabaciones realizadas.



Fuente: Elaboración propia

2.1.2.2. Extracción y selección de fotogramas

Una vez realizado las grabaciones, el siguiente paso consistió en extraer los fotogramas de cada video. Esto se realizó utilizando un script en Python utilizando la biblioteca OpenCV, una herramienta muy robusta en el procesamiento de imágenes y videos lo que nos ayudó a convertir cada video en un conjunto de imágenes. A continuación, se muestra el proceso de extracción de fotogramas:

- **Directorio de salida de los fotogramas:** Antes de realizar la extracción de los fotogramas, se verifica si existe el directorio que se especifica, en caso de no existir se crea uno nuevo usando '*os.makedirs()*'. Con esto se asegura que exista un directorio destino para los fotogramas.
- **Carga de video:** El script utiliza OpenCV para abrir y leer cada uno de los videos. Para cargar el video en memoria se necesita de la ruta que haga referencia al video, luego con el método '*cv2.VideoCapture()*', cargamos el video, de esta manera se tiene acceso a los fotogramas.
- **Lectura y extracción de fotogramas:** Mediante un bucle '*while true*', el script lee cada fotograma del video de manera secuencial. Con el método '*cap.read()*' obtenemos dos valores, un valor booleano que nos indica que el fotograma se leyó correctamente, y el

fotograma. En caso de que el método devuelva un valor '*false*', significa que el video se terminó o que ocurrió un error en la lectura y el bucle se termina.

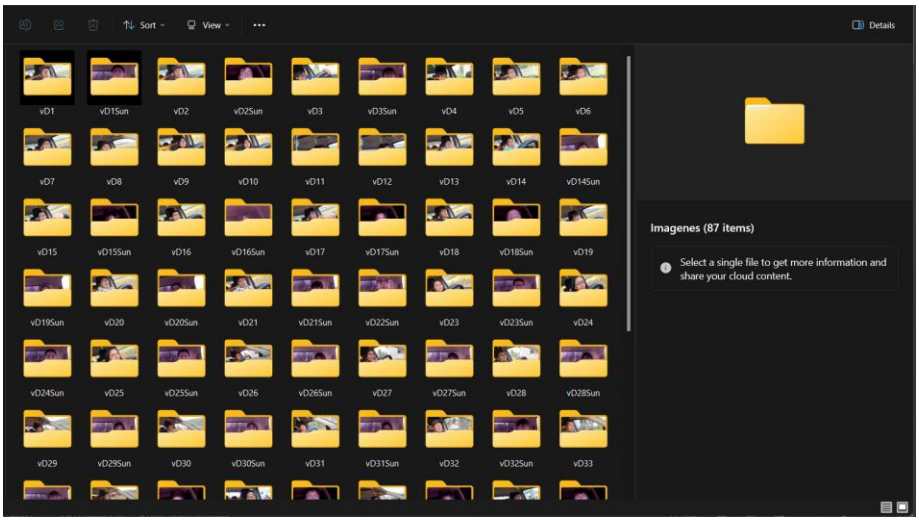
- **Guardado de fotogramas:** Cada fotograma leído correctamente es guardado en el directorio que se especificó. Esto se lo realiza con el método '*cv2.imwrite()*', que crea un nombre único para cada fotograma y lo guarda, de esta manera se controla que no existan conflictos por los nombres de las imágenes.
- **Finalización del proceso:** Se finaliza la ejecución del script una vez que todos los fotogramas del video se han guardado correctamente en el directorio especificado, y usando el método '*cap.release()*' se libera los recursos del sistema que se estaban utilizando.

  De cada video se obtienen 1500 imágenes, esto debido a que cada video tiene una duración de 60 segundos a 25 fotogramas por segundo.

En la Figura 15, se muestran las carpetas con las imágenes extraídas de cada uno de los videos y en la Figura 16, se muestra un ejemplo de las imágenes que contiene cada carpeta.

Figura 15

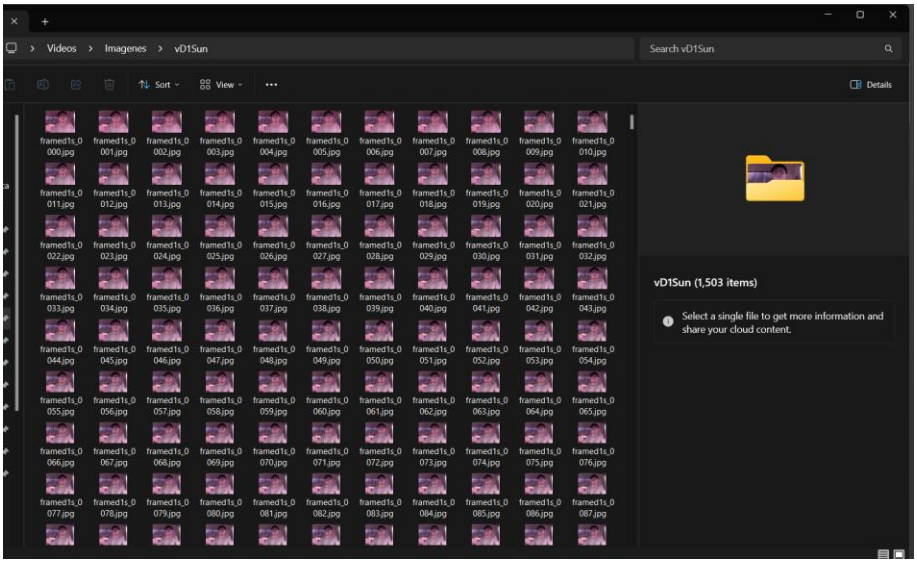
Carpetas con imágenes extraídas de los videos.



Fuente: Elaboración propia

Figura 16

Imágenes que contienen las carpetas.



Fuente: Elaboración propia

Una vez realizado la extracción de los fotogramas de los videos, se continuo con el proceso de la selección de las imágenes. Para esto se tomó en cuenta ciertos criterios de selección:

- **Claridad y calidad de las imágenes:** Se seleccionaron las imágenes que muestren una mejor claridad y que se encuentren libres de distorsiones o desenfoques, esto es muy importante para garantizar que el modelo sea entrenado con datos precisos y detallados.
- **Representación de diversas condiciones:** La diversidad de las imágenes es otro aspecto importante, por lo tanto, se seleccionaron imágenes con una variedad de condiciones de iluminación en el día y en la noche, así también diferentes rostros que tienen diferentes accesorios faciales como los lentes y gafas de sol.
- **Expresiones faciales:** Se seleccionaron imágenes donde se muestre el rostro con los ojos cerrado, entrecerrados y abiertos, de esta manera se asegura que el modelo reconozca estas expresiones oculares, lo que es muy importante para detectar la somnolencia.

Con base en los criterios mencionados se procedió a la selección de las imágenes de forma manual. Cada persona fue grabada tanto en el día como en la noche, con los siguientes criterios faciales: el rostro sin ningún accesorio, el rostro con lentes y el rostro sin lentes. Por lo

tanto, de cada persona se seleccionó las siguientes cantidades de imágenes, en el día: 2 solo el rostro, 2 con lentes y 2 con gafas de sol, esto se aplica con ojos abiertos y cerrados; en la noche: 2 solo el rostro, 2 con lentes y 2 con gafas de sol, esto se aplica con ojos abiertos y cerrado, como se muestra en la Tabla 4.

Tabla 4

Detalle de imágenes seleccionadas de cada persona.

			Num. Imágenes con ojos abiertos	**Num. Imágenes con ojos Cerrados**
Conductor n	**En el día**	Solo el rostro	2	2
		Con lentes	2	2
		Con gafas de sol	2	2
	En la noche	Solo el rostro	2	2
		Con lentes	2	2
		Con gafas de sol	2	2
		Total, imágenes por categoría	**12**	**12**
		Total, imágenes por persona	**24**	

Fuente: Elaboración propia

Es importante considerar que las grabaciones no se realizaron bajo las condiciones previamente mencionadas para todas las personas. Esto se hizo con el propósito de conseguir una mayor diversidad de personas y como resultado obtener una mayor variedad de imágenes.

En la Tabla 5, se presenta un resumen de las imágenes seleccionadas.

Tabla 5

Resumen de imágenes seleccionadas.

Tipo de imagen	**Num. Imágenes**
Expresiones faciales	

Ojos abiertos	132
Ojos cerrados	130
Accesorios faciales	
Solo el rostro	97
Con lentes	81
Con gafas	84
Iluminación	
En el día	138
En la noche	124
Total de imágenes	**262**

Fuente: Elaboración propia

2.2. Etiquetación de imágenes

Para este trabajo de detección de somnolencia, se utilizó dos enfoques que se describen a continuación:

- **Clasificación:** En este enfoque se detecta si una persona presenta somnolencia mediante la clasificación del estado de los ojos, en este enfoque, el modelo será entrenado para esta tarea de clasificación, donde me prediga un valor entre 0 y 1, lo que significa que si el valor se acerca más al uno indica que los ojos cerrados, y si se acerca más al cero indica que los ojos se encuentran abiertos. Los ojos cerrados se han asignado a la clase 1, ya que es el principal indicador que queremos detectar. De esta manera el modelo ResNeXt-50 utilizara imágenes etiquetadas que pertenecerán a una de estas dos clases, y realizará la clasificación.
- **Eye Aspect Ratio (EAR):** Relación de Aspecto del Ojo por sus siglas en español, es una métrica utilizada para monitorear el estado de los ojos, es decir que tan abiertos o cerrados se encuentran. En este enfoque se entrenará el modelo ResNeXt-50 para un problema de regresión, el cual nos devolverá 12 coordenadas que representan los 12 puntos de los ojos, 6 del ojo izquierdo y 6 del ojo derecho para calcular el EAR de cada uno. Una vez obtenido los dos valores se utiliza la métrica PERCLOS (Percentage of Eye Closure) que representa el porcentaje en el que se encuentra cerrado el ojo en una secuencia de imágenes y de esta manera detectar la somnolencia.

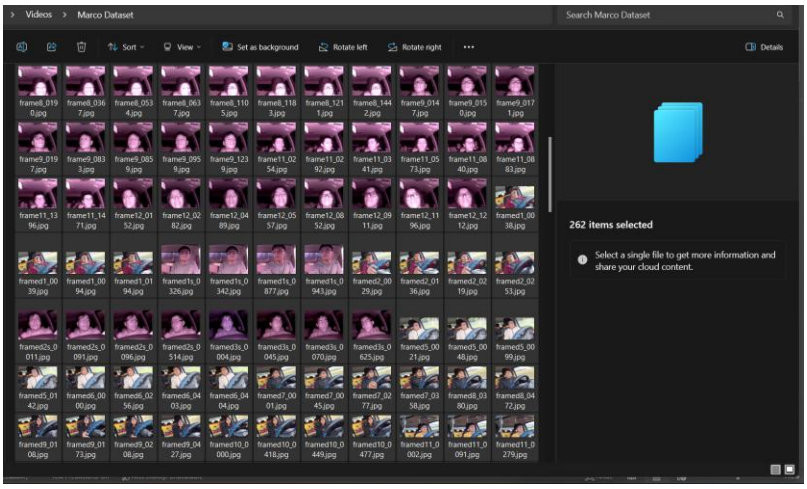
2.2.1. Etiquetación de imágenes para regresión

Se debe mencionara que a pesar de que en este trabajo se utilizará únicamente los 12 puntos mencionados, el algoritmo que se utilizará para la etiquetación de los puntos nos devuelve 68 puntos de referencia facial, así que cada imagen estará etiquetada con los 68 puntos, y es durante la preparación de los datos para el entrenamiento que los puntos se los limitaran solo a 12.

Primero se organizó las imágenes seleccionadas en una sola carpeta como se muestra en la Figura 17, de esta manera el proceso se realizó de forma más rápida.

Figura 17

Carpeta de imágenes para etiquetar.



Fuente: Elaboración propia

Para la etiquetación de las imágenes con los 68 puntos de referencia faciales utilizaron las siguientes librerías, OpenCV y Dlib, los cuales nos ayudaron en el proceso de obtención de los 68 puntos.

Dlib se trata de una librería con herramientas de C++ que cuenta con algoritmos de aprendizaje automático que ayuda a resolver problemas del mundo real. Se lo utiliza en diferentes campos de la industria para áreas como robótica, teléfonos móviles, dispositivos integrados, entre otros. En su librería se encuentra la herramienta '*shape_predictor*', que nos genera un conjunto de ubicaciones de puntos que se encuentran en una imagen, este detector utiliza el Histograma de Orientación Función de Gradientes (HOG) que se combina una pirámide

de imágenes, un clasificador lineal y un esquema de detección de ventana deslizante. En este caso enviaremos imágenes de rostros y nos devolverá las 68 coordenadas de referencia facial los cuales serán guardados en un archivo '*.pts*' con un formato especifico.

El uso de estas dos librerías nos ayudó a etiquetar las imágenes de forma más rápida. A continuación, se detalla el proceso:

- **Cargar el modelo:** Como se muestra en la Figura 18, se importan las librerías '*cv2, dlib y os*', además, cargamos el modelo que predice los puntos de referencia facial, cabe mencionar que el modelo fue descargado de la página oficial de Dlib. Por último, iniciamos el detector de puntos de referencia facial y establecemos la ruta en donde se encuentran las imágenes.

Figura 18

Carga del modelo para detección de 68 puntos de referencia facial.

```python
import cv2
import dlib
import os

# Cargar el modelo de landmarks
predictor_path = 'shape_predictor_68_face_landmarks.dat'
predictor = dlib.shape_predictor(predictor_path)

# Iniciar el detector de caras de Dlib
detector = dlib.get_frontal_face_detector()

# Ruta a la carpeta de imágenes
folder_path = 'C:/Users/Marco/Videos/DATASET TESIS/Corregir'
```

Fuente: Elaboración propia

- **Recorrer y convertir las imágenes:** Como se muestra en la Figura 19, recorremos toda la carpeta que contiene las imágenes y verificamos que los archivos sean de tipo imagen ya sea 'png', 'jpg' o 'jpeg'. Luego, desde la librería de OpenCV utilizamos el método '*cv2.imread(ruta_imagen)*' para cargar la imagen que se encuentra en una ruta especifica, convertimos la imagen a escala de grises con '*cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)*', el cual nos pide como entrada una imagen cargada y el formato de color al que quiere que se convierta. Se convirtió a escala de grises debido a que ese es el formato que acepta el modelo para predecir los 68 puntos de referencia facial.

Figura 19

Preprocesamiento de las imágenes.

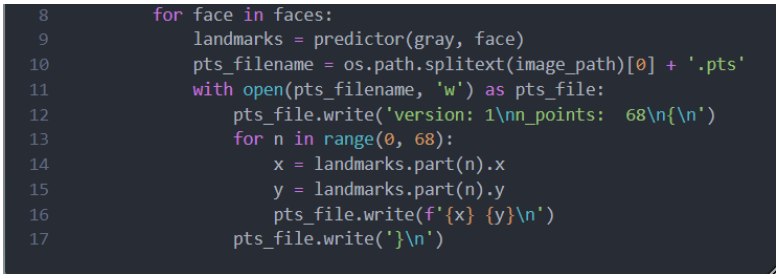
```python
# Recorrer todos los archivos en la carpeta
for filename in os.listdir(folder_path):
    if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
        image_path = os.path.join(folder_path, filename)
        image = cv2.imread(image_path)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        faces = detector(gray)
```

Fuente: Elaboración propia

- **Guarda los 68 puntos:** En la Figura 20, el modelo nos devuelve las 68 coordenadas de los puntos de referencia facial por cada rostro detectado en la imagen, por lo que para cada imagen se crea un archivo con extensión '*.pts*' y el mismo nombre de la imagen donde se guardaran las coordenadas de los puntos, además, el archivo sigue un formato donde se incluye la versión del archivo, el número de puntos y las coordenadas dentro de llaves. En la Figura 21, se muestra cómo queda el formato mencionado.

Figura 20

Guardar los 68 puntos en el archivo ".pts".

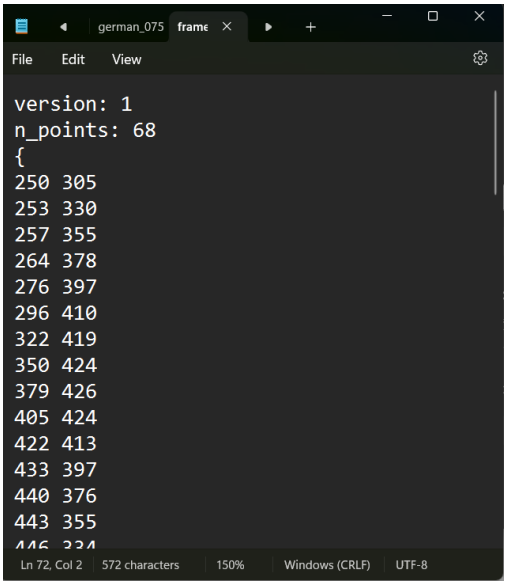


```python
        for face in faces:
            landmarks = predictor(gray, face)
            pts_filename = os.path.splitext(image_path)[0] + '.pts'
            with open(pts_filename, 'w') as pts_file:
                pts_file.write('version: 1\nn_points:  68\n{\n')
                for n in range(0, 68):
                    x = landmarks.part(n).x
                    y = landmarks.part(n).y
                    pts_file.write(f'{x} {y}\n')
                pts_file.write('}\n')
```

Fuente: Elaboración propia

Figura 21

Formato del archivo ".pts" de cada imagen.



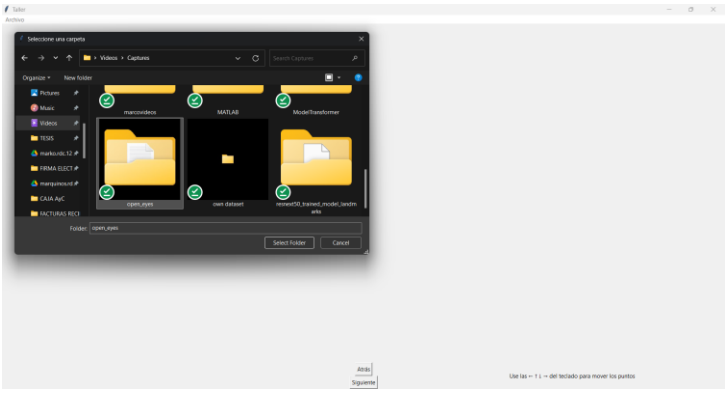
Fuente: Elaboración propia

Para asegurar que las coordenadas guardadas sean las correctas, se utilizó un programa desarrollado en Python el cual nos permite visualizar las imágenes con sus respectivos archivos '*.pts*' y modificar cada coordenada que se encuentra en el archivo en caso de ser necesario, asegurando así que los datos para el entrenamiento sean de mayor calidad y de esta manera obtener mejores resultados en el modelo. A continuación, se detalla el proceso de corrección de las coordenadas:

- Seleccionamos la carpeta donde se encuentran las imágenes con sus respectivos archivos '*.pts*' en la interfaz que se muestra en la Figura 22.

Figura 22

Selección de carpeta para edición de los puntos faciales.

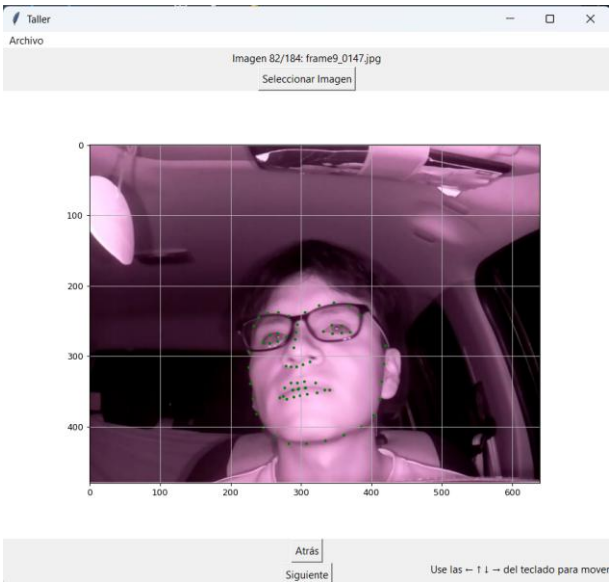


Fuente: Elaboración propia

- **Corrección de puntos y guardado:** Al abrir el directorio obtenemos la interfaz que se muestra en la Figura 23, donde observamos la imagen y los puntos de color verde que representan los 68 puntos de referencia facial. Además, podemos navegar entre las imágenes y mover los puntos a donde deseemos.

Figura 23

Imagen con sus respectivos puntos faciales.

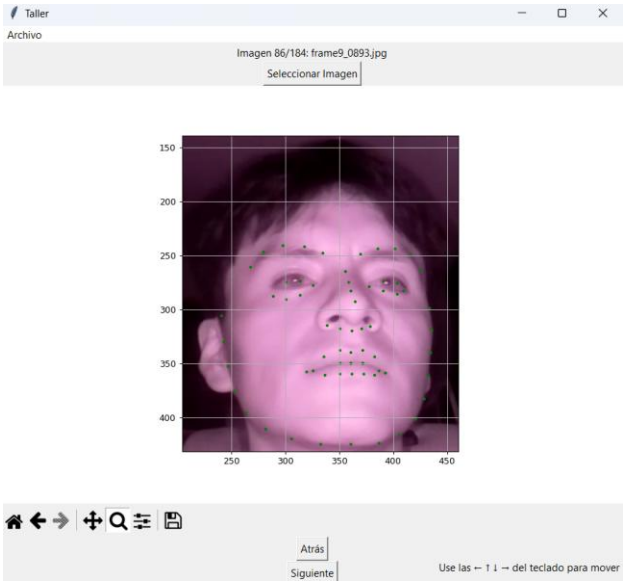


Fuente: Elaboración propia

Corregiremos las imágenes que contengan errores en la ubicación de los 12 puntos perteneciente a los 2 ojos como se observa en la Figura 24, donde dichos puntos se encuentran en ubicaciones incorrectas, la finalidad es corregirlos y obtener coordenadas más precisas. Solo se corregirán los 12 puntos debido a que son los que se utilizaran en el entrenamiento. Para poder mover los puntos damos clic en el punto que deseemos modificar y movemos con las teclas de navegación.

Figura 24

Puntos de los ojos mal ubicados.

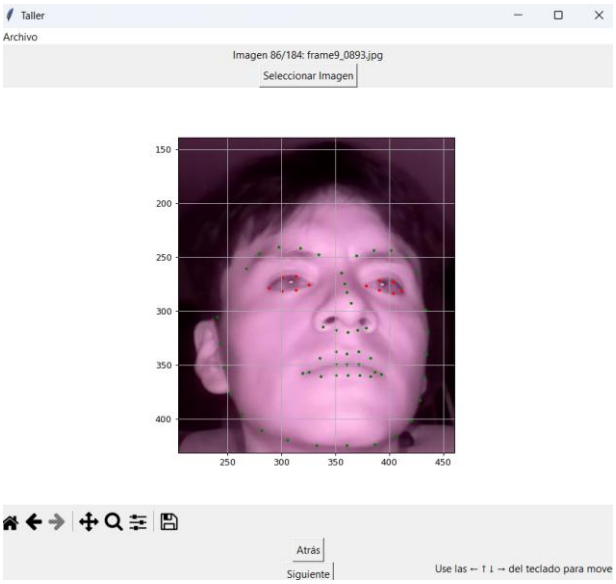


Fuente: Elaboración propia

Corregimos los puntos, que son aquellos que se encuentra de color rojo y guardamos las correcciones. De esta manera se actualizan los archivos con las coordenadas corregidas. Esta corrección lo observamos en la Figura 25.

Figura 25

Puntos de los ojos corregidos.



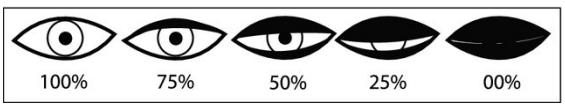
Fuente: Elaboración propia

2.2.2. Etiquetación de imágenes para clasificación

Para el tema de clasificación, las imágenes se dividieron en dos grupos, ojos abiertos y ojos cerrados. Para poder separar las imágenes en estos dos grupos, nos ayudamos de las etiquetas de los puntos de referencia facial de cada imagen que se realizó anteriormente en la etiquetación para la parte de regresión, nos centramos en los 12 puntos que representan los ojos, seis puntos para el ojo izquierdo y seis puntos para el ojo derecho. Una vez determinado cuales son estos puntos, se calculó el Eyes Aspect Ratio (EAR). Se realizaron varias experimentaciones con diferentes personas, y en base al a Figura 26, se observa que del 25% al 00% de apertura del ojo ya es muy difícil que una persona pueda ver. Debido a esto el valor del umbral para determinar que los ojos se encuentran cerrados se estableció en 0.20 que llega a representar este rango de porcentaje, y para los ojos abiertos los valores que sean mayor a este.

Figura 26

Porcentaje de apertura de los ojos.

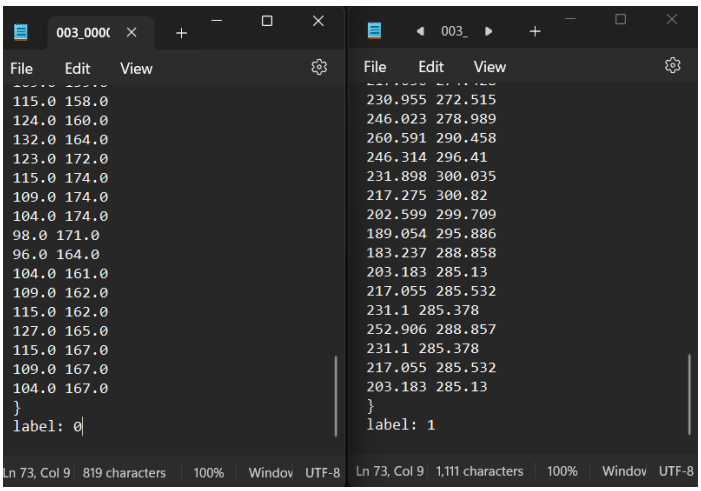


Fuente: (Akrout & Mahdi, 2023)

Una vez establecido el valor del umbral, nos ayudamos de un script en Python para realizar la división de las imágenes con sus respectivos archivos '*.pts*'. Luego se añadió la etiqueta para la clasificación al final del mismo archivo '*.pts*' de cada imagen, estableciendo el valor de 1 para los ojos cerrados y 0 para los ojos abiertos, de esta manera el archivo quedo con una última línea adicional para esta etiqueta como se muestra en la Figura 27.

Figura 27

Archivos ".pts" con la etiqueta de clasificación.



Fuente: Elaboración propia

2.2.3. Aplicación filtro de gafas de sol

Una vez etiquetadas las imágenes, se aplicó filtros de gafas de sol para aumentar el número de imágenes donde los rostros tengan este accesorio. Para ello se utilizó un script en Matlab. El proceso para realizar la tarea fue el siguiente:

a) Selección de imágenes con los siguientes criterios aplicados a cada persona, durante el día, la noche, ojos abiertos y ojos cerrados. Es decir, de cada persona se obtuvo 4 imágenes a las cuales se le aplicaría el filtro, obteniendo así un total de 66 imágenes. Además, se creó un repertorio con un total de 20 diferentes modelos de gafas que se muestra en la Figura 28, lo cual nos permitirá tener imágenes variadas.

Figura 28

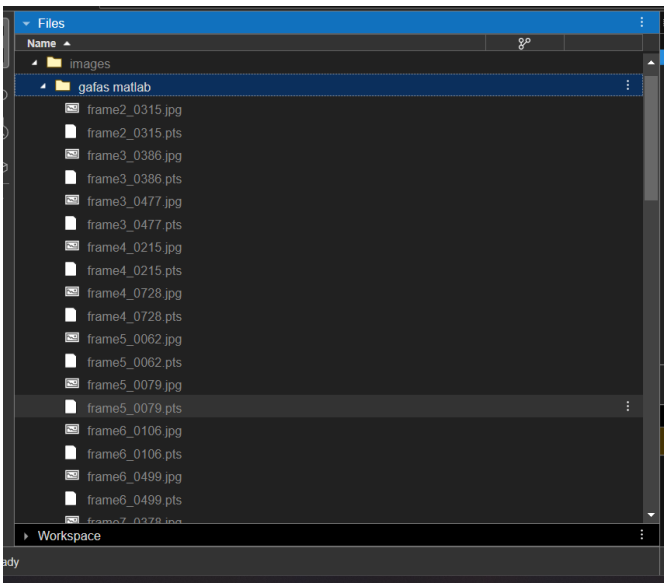
Modelos de gafas seleccionadas.

Fuente: Elaboración propia

b) Una vez seleccionadas, las imágenes junto con sus respectivos archivos '*.pts*' de los 68 puntos del faciales se cargaron en el entorno de ejecución de Matlab como se muestra en la Figura 29. Esto se realizó utilizando Matlab online, que nos permite el acceso desde cualquier navegador lo que nos ahorra el tiempo de instalación.

Figura 29

Datos en el entorno de Matlab Web.



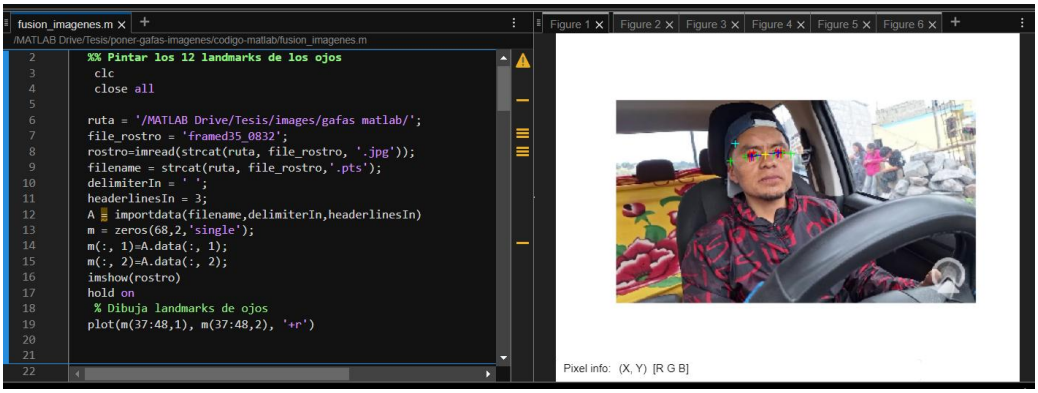
Fuente: Elaboración propia

c) El script comienza definiendo la ubicación de los archivos tanto de la imagen como de los puntos en una ruta específica para leerlos. Solo se toman en cuenta los puntos de los

ojos para poder superponerlos sobre la imagen como se muestra en la Figura 30, esto es muy importante para ubicar correctamente las gafas.

Figura 30

Carga de imágenes.

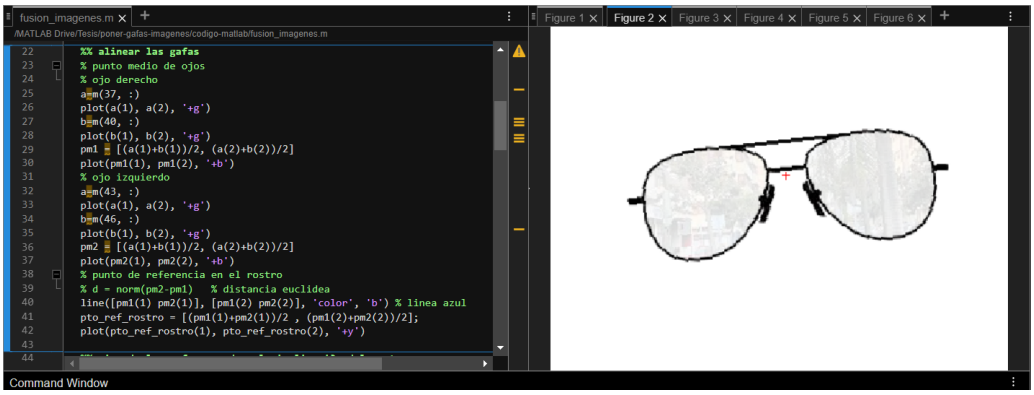


Fuente: Elaboración propia

d) Se calculó el ángulo de inclinación necesario entre los puntos que se encuentran alrededor del ojo para obtener el ajuste correcto de la rotación que deben tener la gafas. Además, se lee la imagen de las gafas y se roto de acuerdo con el ángulo calculado como se muestra en la Figura 31.

Figura 31

Rotación de las gafas de acuerdo con la ubicación de los puntos de los ojos.

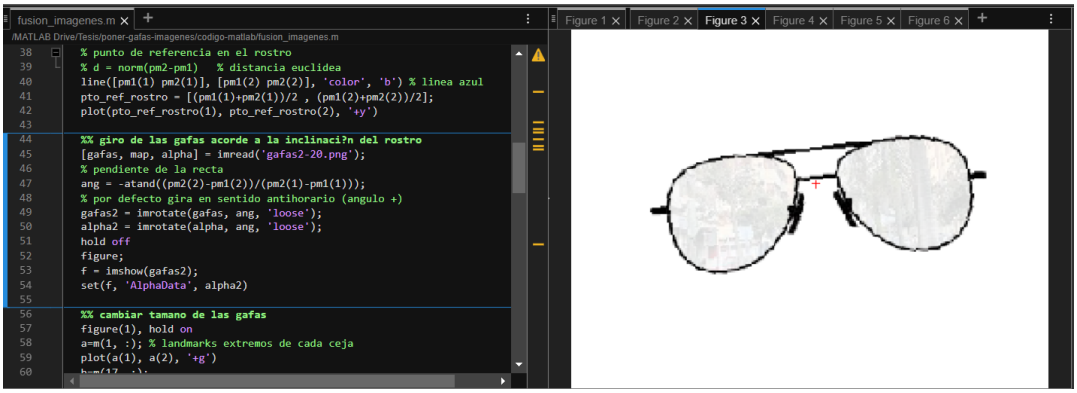


Fuente: Elaboración propia

e) La Figura 32, muestra el cálculo de la distancia entre los puntos medios de los ojos para poder ajustar el tamaño de las gafas, de esta manera se asegura que las gafas tengan el tamaño correcto para el rostro de la imagen.

Figura 32

Ajuste del tamaño de las gafas.

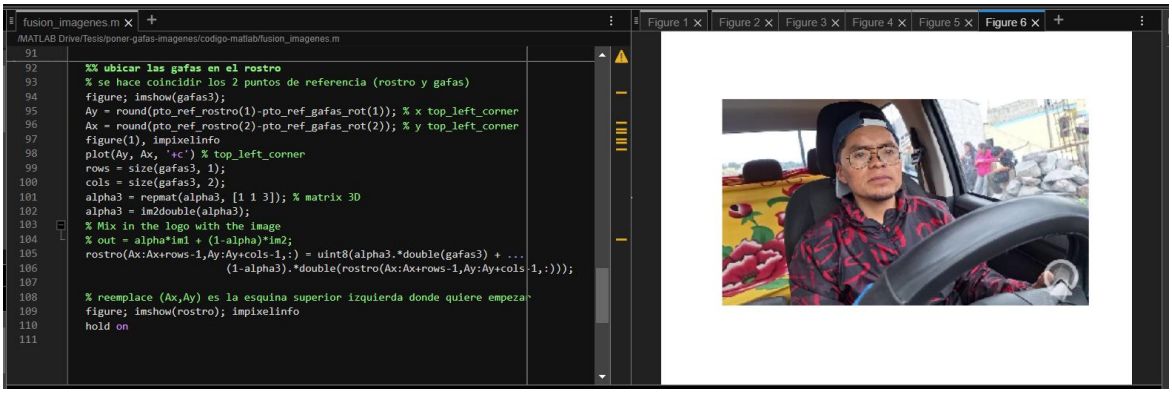


Fuente: Elaboración propia

f) En la Figura 33, se realiza la superposición de las gafas en la imagen aplicando el canal alfa para manejar transparencia y que de esta manera se obtenga un resultado más natural y precisa de las gafas sobre el rostro de la imagen.

Figura 33

Superposición de las gafas en el rostro.



Fuente: Elaboración propia

2.3. Entrenamiento y prueba de la ResNeXt-50 CNN

2.3.1. Dataset

Se utilizó el Dataset de creación propia detallado en la sección 2.4.1. "*Creación de un Dataset*", además, Datasets populares de Internet para la detección de 68 puntos de referencia facial que son los siguientes: 300VW, 300W, afw, CEW, COFW, frgc, Helen, ibug, lfpw,

Mempo2D y xm2vts, todos estos se encuentran etiquetados con los 68 puntos, creando así una fusión con una gran cantidad de datos.

Se crearon dos Datasets diferentes que se detallan a continuación:

- **Dataset balanceado:** En este Dataset balanceamos tanto la clase de ojos abiertos y ojos cerrados con una cantidad de 5871 datos para cada clase. En la Tabla 6, se muestran la cantidad de datos que se tomó de cada Dataset.

Tabla 6

Distribución de datos para el Dataset balanceado.

Dataset	**Ojos abiertos**		**Ojos cerrados**	
	Num. Datos	**%**	**Num. Datos**	**%**
300W	201	3%	409	7%
300WV	0	0%	2295	39%
afw	110	2%	152	3%
CEW	0	0%	555	9%
COFW	80	1%	15	0%
frgc	907	15%	48	1%
Helen	1396	24%	693	12%
Ibug	44	1%	53	1%
lfpw	522	9%	76	1%
Mempo	687	12%	366	6%
Xm2vts	461	8%	40	1%
Propio	1463	25%	1169	20%
Total	5871	100%	5871	100%
Porcentaje	50%		50%	

Fuente: Elaboración propia

- **Dataset desbalanceado:** A este Dataset se amento un 30% de datos a la clase de ojos abiertos con un total de 13703 datos en comparación con la cantidad de datos del Dataset balanceado, mientras que la clase de ojos cerrados se mantuvo con 5871 datos. La Tabla 7, muestra la cantidad de datos que se tomó de cada Dataset.

Tabla 7

Distribución de datos para el Dataset desbalanceado.

Dataset	**Ojos abiertos**		**Ojos cerrados**	
	Num. Datos	**%**	**Num. Datos**	**%**
300W	551	4%	409	7%
300WV	0	0%	2295	39%
afw	298	2%	152	3%
CEW	0	0%	555	9%
COFW	217	2%	15	0%
frgc	2489	18%	48	1%
Helen	3980	29%	693	12%
Ibug	121	1%	53	1%
lfpw	1433	10%	76	1%
Mempo	1887	14%	366	6%
Xm2vts	1264	9%	40	1%
Propio	1463	11%	1169	20%
Total	13703	100%	5871	100%
Porcentaje	70%		30%	

Fuente: Elaboración propia

Se organizó el Dataset de la siguiente manera. Una carpeta principal con el nombre del Dataset, en este caso se nombró 'Dataset Landmarks', dentro de la carpeta principal se crearon tres carpetas, uno de entrenamiento, validación y pruebas, donde se almacenaron las imágenes y sus respectivos archivos "*.pts*", con una distribución de datos del 80%, 10% y 10% respectivamente.

2.3.1. Desarrollo del modelo ResNeXt-50

Se utilizó TensorFlow y Keras para el desarrollo de este modelo. Normalmente, en TensorFlow y Keras ya se encuentran desarrollados y preentrenados algunos modelos como DenseNet, EfficientNet, MobileNet, ResNet, entre otros; sin embargo, el modelo ResNeXt-50 no se encuentra disponible. Debido a esto se buscó en Internet el modelo que esté construido desde cero con la biblioteca de TensorFlow, y se encontró el modelo en el siguiente repositorio de

GitHub *'https://github.com/calmiLovesAI/ResNeXt_TensorFlow2.git'*. El modelo ResNext50 encontrado está enfocado para temas de clasificación, y debido a que nuestro trabajo se va a centrar tanto en clasificación y regresión, el modelo se adaptó para el problema de clasificación binaria, y para regresión que predecirá 24 valores que representan los 12 puntos (x, y) de los ojos tanto derecho como izquierdo.

En la Figura 34, se muestra la salida que tenía el modelo ResNext-50 que está diseñado para tareas de clasificación de 1000 clases que se muestra en la línea 31 que es la capa de salida, además, está configurada con la función de activación '*softmax'* utilizada en problemas de clasificación multiclase, y ayuda a convertir los valores generados por el modelo en una distribución de probabilidad.

Figura 34

Salida original del modelo ResNeXt-50.

```
1  class ResNeXt(tf.keras.Model):
2      def __init__(self, repeat_num_list, cardinality):
3          if len(repeat_num_list) != 4:
4              raise ValueError("The length of repeat_num_list must be four.")
5          super(ResNeXt, self).__init__()
6          self.conv1 = tf.keras.layers.Conv2D(filters=64,
7                                              kernel_size=(7, 7),
8                                              strides=2,
9                                              padding="same")
10         self.bn1 = tf.keras.layers.BatchNormalization()
11         self.pool1 = tf.keras.layers.MaxPool2D(pool_size=(3, 3),
12                                                strides=2,
13                                                padding="same")
14         self.block1 = build_ResNeXt_block(filters=128,
15                                           strides=1,
16                                           groups=cardinality,
17                                           repeat_num=repeat_num_list[0])
18         self.block2 = build_ResNeXt_block(filters=256,
19                                           strides=2,
20                                           groups=cardinality,
21                                           repeat_num=repeat_num_list[1])
22         self.block3 = build_ResNeXt_block(filters=512,
23                                           strides=2,
24                                           groups=cardinality,
25                                           repeat_num=repeat_num_list[2])
26         self.block4 = build_ResNeXt_block(filters=1024,
27                                           strides=2,
28                                           groups=cardinality,
29                                           repeat_num=repeat_num_list[3])
30         self.pool2 = tf.keras.layers.GlobalAveragePooling2D()
31         self.fc = tf.keras.layers.Dense(1000, activation=tf.keras.activations.softmax)
32
```

Fuente: Elaboración propia

El modelo ResNeXt-50 se configuro para realizar tareas de clasificación y regresión al mismo tiempo como se muestra en la Figura 35. Una salida de clasificación formada por una capa densa de una salida y la función de activación '*sigmoid'* utilizada para clasificación binaria,

y una salida de regresión con una capa densa de 24 salidas que representan los 12 puntos (x, y) de los ojos. Esta configuración permite que el modelo determine si los ojos se encuentran cerrados mientras la probabilidad se acerque más al 1 o abiertos si se acerca al 0, y prediga los 12 puntos de los ojos.

Figura 35

Configuración de las salidas de clasificación y regresión del modelo ResNeXt-50.

```
class ResNeXt(tf.keras.Model):
    def __init__(self, repeat_num_list=[3, 4, 6, 3], cardinality=32, num_classes=2, num_landmarks=12, kernel_initializer='he_normal'):
        super(ResNeXt, self).__init__()
        self.conv1 = layers.Conv2D(64, kernel_size=7, strides=2, padding='same', kernel_initializer=kernel_initializer)
        self.bn1 = layers.BatchNormalization()
        self.pool1 = layers.MaxPooling2D(pool_size=3, strides=2, padding='same')

        self.block1 = build_resnext_block(128, 1, cardinality, repeat_num_list[0], kernel_initializer=kernel_initializer)
        self.block2 = build_resnext_block(256, 2, cardinality, repeat_num_list[1], kernel_initializer=kernel_initializer)
        self.block3 = build_resnext_block(512, 2, cardinality, repeat_num_list[2], kernel_initializer=kernel_initializer)
        self.block4 = build_resnext_block(1024, 2, cardinality, repeat_num_list[3], kernel_initializer=kernel_initializer)

        self.pool2 = layers.GlobalAveragePooling2D()

        self.classifier = layers.Dense(1, activation='sigmoid', name='output_classifier')
        self.regressor = layers.Dense(num_landmarks * 2, activation='linear', name='output_regressor')
```

Fuente: Elaboración propia

Una vez realizado las modificaciones, se creó y compiló el modelo como se muestra en la Figura 36. Se define la función ‘ResNeXt50()’ que nos devuelve el modelo. Se construye el modelo creando una instancia de este utilizando la función ‘ResNeXt50()’ y continua con la especificación de la forma de entrada utilizando el método ‘*modelo.build()*’, donde se establece el tamaño de entrada de las imágenes a 128x128 píxeles y en escala de grises representada por el número 1, quedando la entrada del modelo en una forma de ‘*(None, 128, 128, 1)*’.

Figura 36

Compilación del modelo ResNeXt-50.

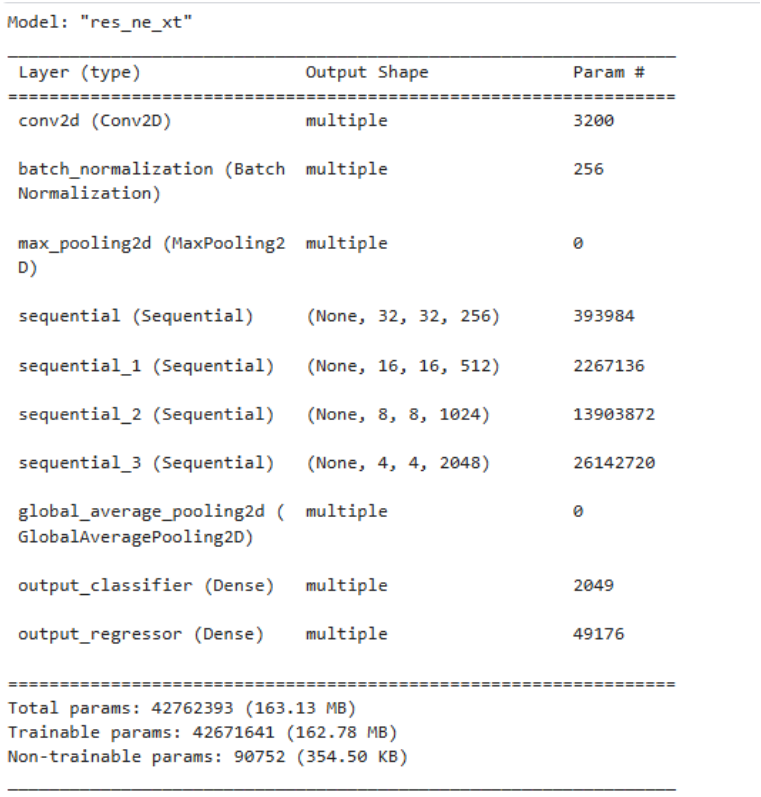
```
model = ResNeXt()
model.build(input_shape=(None, 128, 128, 1))
model.summary()
```

Fuente: Elaboración propia

Finalmente, con el método ‘*model.summary()*’ observamos la arquitectura detallada del modelo como se observa en la Figura 37.

Figura 37

Arquitectura modificada del modelo ResNeXt-50.



```
Model: "res_ne_xt"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             multiple                  3200

 batch_normalization (Batch  multiple                  256
 Normalization)

 max_pooling2d (MaxPooling2  multiple                  0
 D)

 sequential (Sequential)     (None, 32, 32, 256)       393984

 sequential_1 (Sequential)   (None, 16, 16, 512)       2267136

 sequential_2 (Sequential)   (None, 8, 8, 1024)        13903872

 sequential_3 (Sequential)   (None, 4, 4, 2048)        26142720

 global_average_pooling2d (  multiple                  0
 GlobalAveragePooling2D)

 output_classifier (Dense)   multiple                  2049

 output_regressor (Dense)    multiple                  49176

=================================================================
Total params: 42762393 (163.13 MB)
Trainable params: 42671641 (162.78 MB)
Non-trainable params: 90752 (354.50 KB)
_________________________________________________________________
```

Fuente: Elaboración propia

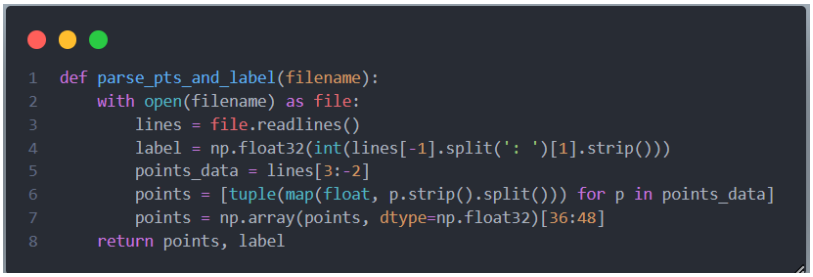
2.3.2. Carga y preprocesamiento de datos

Es un paso muy importante donde preparamos los datos (imágenes y archivos con las coordenadas) para que tengan el formato que el modelo necesita como entrada para el entrenamiento. Se definieron tres métodos para el procesamiento de datos que se detalla a continuación:

- **‘*parse_pts_and_label(filename)*’:** este método recibe un archivo en formato ‘*.pts*’, extrae la etiqueta de clasificación ubicada en la última línea de los archivos, limita para que se tomen únicamente los puntos desde el 36 al 47 que representan los puntos de los ojos, y convierte estos puntos seleccionados en un arreglo Numpy. Finalmente, retorna la etiqueta y los 12 puntos de los ojos. En la Figura 38, se muestra el proceso.

Figura 38

Procesamiento de puntos y etiquetas.



```python
def parse_pts_and_label(filename):
    with open(filename) as file:
        lines = file.readlines()
        label = np.float32(int(lines[-1].split(': ')[1].strip()))
        points_data = lines[3:-2]
        points = [tuple(map(float, p.strip().split())) for p in points_data]
        points = np.array(points, dtype=np.float32)[36:48]
    return points, label
```

Fuente: Elaboración propia

- **'*load_image_and_landmarks(image_path, pts_path)*':** La Figura 39, muestra el proceso del método. Se encarga del procesamiento de la imagen y su respectivo archivo de coordenadas. La imagen se redimensiona a un tamaño de 128x128 pixeles, se convierte a escala de grises, y se normaliza dividiendo por 255. Los puntos y la etiqueta se obtienen con el método '*parse_pts_and_label()*'. Estos puntos se normalizan al nuevo tamaño de la imagen. Finalmente, el método devuelve la imagen procesada, los puntos normalizados y la etiqueta de clasificación.

Figura 39

Carga y normalización de imágenes y puntos.

```python
def load_image_and_landmarks(filepath):

    image_path = filepath
    pts_path = filepath.replace('.jpg', '.pts')

    image = Image.open(image_path)
    original_width, original_height = image.size
    image = image.resize((128, 128)).convert('L')
    image = np.array(image, dtype=np.float32) / 255.0
    image = np.expand_dims(image, axis=-1)

    points, label = parse_pts_and_label(pts_path)
    points[:, 0] /= original_width
    points[:, 1] /= original_height
    points = points.flatten()

    return image, points, label
```

Fuente: Elaboración propia

- **‘*load_data(image_dir, pts_dir)*’:** Carga todas las imágenes y sus respectivos archivos ‘*.pts*’, desde los directorios especificados como se muestra en la Figura 40, los cuales son procesados utilizando los métodos definidos anteriormente. Estos datos procesados son almacenados en tres listas: para imágenes, para los puntos, y para las etiquetas. Finalmente, el método devuelve tres arreglos conteniendo todas las imágenes, puntos y etiquetas.

Figura 40

Carga general de datos.

```python
def load_data(directory):
    file_paths = [os.path.join(directory, f) for f in os.listdir(directory) if f.endswith('.jpg')]
    images, points, labels = [], [], []
    count=0
    for filepath in file_paths:
        try:
            img, pts, label = load_image_and_landmarks(filepath)
            images.append(img)
            points.append(pts)
            labels.append(label)
            count+=1
            print(f"Procesando: {count} - {os.path.basename(filepath)}")
        except Exception as e:
            print(f"Error al procesar {filepath}: {str(e)}")

    return np.array(images), np.array(points), np.array(labels)

```

Fuente: Elaboración propia

En la Figura 41, se muestra cómo se establecen las rutas donde se encuentran los datos de entrenamiento, validación y prueba para realizar el preprocesamiento de datos con los métodos explicados anteriormente. Al terminar el preprocesamiento, se crean Datasets de TensorFlow.

Figura 41

Creación de los datasets en TensorFlow.

```python
# Rutas de datos
train_dir = '/kaggle/input/dataset-301/Dataset 30/train'
val_dir = '/kaggle/input/dataset-301/Dataset 30/validation'
test_dir = '/kaggle/input/dataset-301/Dataset 30/test 30'

# Cargar datos
train_images, train_points, train_labels = load_data(train_dir)
val_images, val_points, val_labels = load_data(val_dir)
test_images, test_points, test_labels = load_data(test_dir)


# Preparar datasets de TensorFlow
train_dataset = tf.data.Dataset.from_tensor_slices((train_images, {'output_regressor': train_points, 'output_classifier': train_labels}))
val_dataset = tf.data.Dataset.from_tensor_slices((val_images, {'output_regressor': val_points, 'output_classifier': val_labels}))
test_dataset = tf.data.Dataset.from_tensor_slices((test_images, {'output_regressor': test_points, 'output_classifier': test_labels}))
```

Fuente: Elaboración propia

2.3.3. Entrenamiento del modelo ResNeXt-50

Se utilizaron dos plataformas para el entrenamiento del modelo que se detallan a continuación:

Google Colab: Es una plataforma basada en la nube proporcionada por Google que nos permite ejecutar notebooks Jupyter en Python, no se necesita de configuraciones adicionales para su uso y proporciona acceso gratuito a recursos informáticos, como GPU y TPU. Esta plataforma está enfocada para su uso en las áreas de aprendizaje automático, la ciencia de datos y educación (Google, 2024).

Para el entrenamiento del modelo ResNeXt-50 en Google Colab, se utilizó un entorno con la siguiente configuración:

- **Tipo de Runtime:** Python 3
- **Acelerador de hardware:** GPU L4
- **Capacidad de la GPU:** Memoria dedicada de 22.5 GB
- **Memoria RAM:** 53 GB
- **Almacenamiento:** 112.6 GB

Versiones de erramientas y librerias claves:

- **Python:** 3.10.12
- **Tensorflow y Keras:** 2.15.0

HPC CEDIA: El High Performance Computing (HPC) de CEDIA (Corporación Ecuatoriana para el Desarrollo de la Investigación y la Academia) es una infraestructura con un conjunto de equipos conectados mediante redes de alta velocidad para el procesamiento de grandes cantidades de datos, realizar operaciones matemáticas complejas, Inteligencia Artificial, Machine Learning, entre otros (CEDIA, 2024).

Para el entrenamiento del modelo ResNeXt-50 en HPC CEDIA se utilizó un entorno con la siguiente configuración:

- **Tipo de Runtime:** Python 3
- **Cores de CPU:** 40 cores
- **Acelerador de hardware:** GPU NVIDIA A100 SMX4

- **Capacidad de la GPU:** Memoria dedicada de 40 GB
- **Memoria RAM:** 32 GB

Versiones de herramientas y librerías claves:

- **Python:** 3.10.0
- **TensorFlow y Keras**: 2.10.0

Para realizar la creación y configuración del entorno, así también como la carga de datos y la instalación de librerías véase el Anexo 1.

Se preparan los datos de entrenamiento, validación y prueba, estableciendo el tamaño de lote (batch size) en 8. Además, se aplica una mezcla a los datos de entrenamiento con la función '*shufle()*' estableciendo un buffer de 4048 como se muestra en la Figura 42.

Figura 42

Preparación de los datos de entrenamiento, validación y prueba.

```
size=8
train_dataset = train_dataset.shuffle(buffer_size=4048).batch(size).prefetch(tf.data.experimental.AUTOTUNE)
val_dataset= val_dataset.batch(size).prefetch(tf.data.experimental.AUTOTUNE)
test_dataset = test_dataset.batch(size).prefetch(tf.data.experimental.AUTOTUNE)
```

Fuente: Elaboración propia

A continuación, se compila el modelo en TensorFlow, donde se establece el optimizador Adam con una tasa de aprendizaje inicial de 0.005. Se utiliza una perdida combinada entre clasificación y regresión, asignando '*binary_crossentropy*' y '*mean_squared_error*' como funciones de perdida respectivamente. Se asigna un peso de 0.4 para la tarea de clasificación y 0.6 para regresión, indicando una mayor importancia en la tarea de regresión. Además, se especifican métricas para evaluar el rendimiento del modelo durante y después del entrenamiento: exactitud, precisión, recall y F1 para la clasificación, y Error Medio Cuadrático (MSE), Error Absoluto Medio (MAE) y la Raíz Cuadrada del Error Cuadrático Medio (RMSE) para la regresión. Todo esto se muestra en la Figura 43.

Figura 43

Compilación del modelo ResNeXt-50.

```
initial_learning_rate = 0.005
optimizer = tf.keras.optimizers.AdamW(learning_rate=initial_learning_rate)

model.compile(
    optimizer=optimizer,
    loss={
        'output_classifier': 'binary_crossentropy',
        'output_regressor': 'mean_squared_error'
    },
    loss_weights={
        'output_classifier': 0.4,
        'output_regressor': 0.6
    },
    metrics={
        'output_classifier': [
            'accuracy',
            Precision(name='precision'),
            Recall(name='recall'),
            F1Score(name='f1_score')
        ],
        'output_regressor': [
            'mse',
            tf.keras.metrics.MeanAbsoluteError(name='mae'),
            tf.keras.metrics.RootMeanSquaredError(name='rmse')
        ]
    }
)
```

Fuente: Elaboración propia

Se aplicó funciones Callbacks, que se tratan de objetos los cuales pueden realizar diferentes acciones tales como escribir registros en TensorBoard, guardar periódicamente el modelo, realizar paradas tempranas, reducir la taza de aprendizaje, entre otros, durante el entrenamiento.

Se configuró la función callback '*lr_scheduler*' utilizando la clase de TensorFlow '*ReduceLROnPlateau()*', el cual nos permite reducir la taza de aprendizaje cuando una métrica de validación ha dejado de mejorar, en este caso monitoreamos la perdida de validación (val_loss). Se estableció la siguiente configuración representado en la Figura 44:

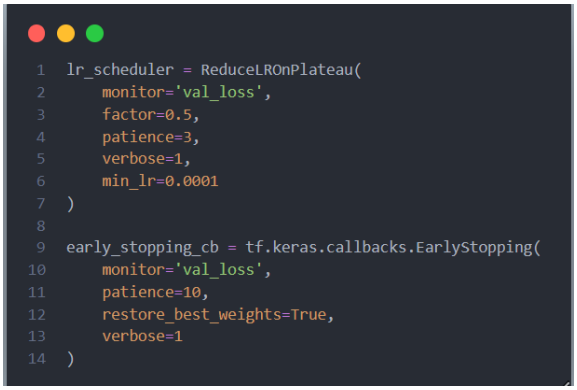
- **'*factor*':** Se estableció con el valor de 0.5, es decir el valor de la taza de aprendizaje se reduce a la mitad.
- **'*patience*':** Se estableció el valor en 3, lo que significa que esperará 3 épocas sin mejoras antes de reducir la taza de aprendizaje.
- '*verbose*': Establecido en 1, lo que permite imprimir un mensaje cuando se hace un cambio en la taza de aprendizaje.
- **'*min_lr*':** Se estableció el valor en 0.001, que representa el valor mínimo que podrá tomar la taza de aprendizaje.

También se establece una segunda función callback '*early_stop_cb*' con la clase '*EarlyStopping()*', que permite detener el entrenamiento después de un cierto número de épocas establecidas en las cuales no existe mejora en la métrica de pérdida. La configuración como se muestra en la Figura 44, se detalla a continuación:

- **'*patience*':** Se estableció en 10, es decir, esperara 10 épocas sin mejoras antes de detenerse.
- **'*restore_best_weights*':** Se lo estableció en True, lo que revertirá a los pesos donde se observó la mejor métrica.

Figura 44

Configuración de Callbacks para el entrenamiento.



```
lr_scheduler = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=3,
    verbose=1,
    min_lr=0.0001
)

early_stopping_cb = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True,
    verbose=1
)
```

Fuente: Elaboración propia

A continuación, se configuró el entrenamiento estableciendo el número máximo de épocas en 50. Como se observa en la Figura 45, con el método '*fit()*' se llevó a cabo el entrenamiento, que recibió como entradas el Dataset de entrenamiento y el de validación, además de las dos funciones callbacks establecidas anteriormente (lr_scheduler y early_stop_cb). Al finalizar el entrenamiento el modelo se evaluó con el Dataset de prueba. También se utilizó una función de la librería Time, que nos permitió conocer la duración que tomó el entrenamiento en minutos.

Figura 45

Configuración para el entrenamiento del modelo ResNeXt-50.

```
epochs =50
start_time = time.time()
history = model.fit(
    train_dataset,
    epochs=epochs,
    validation_data=val_dataset,
    callbacks=[lr_scheduler, early_stopping_cb],
    verbose=1
)

results = model.evaluate(test_dataset)
for name, value in zip(model.metrics_names, results):
    print(f"{name}: {value}")

end_time = time.time()
total_time = (end_time - start_time) / 60
```

Fuente: Elaboración propia

Finalmente, el modelo se guardó en dos formatos, SavedModel y TensorFlow Lite, siendo este último el más recomendado para dispositivos móviles y sistemas embebidos debido a su capacidad de ejecutarse con recursos limitados. En la Figura 46, se observa el proceso de guardado.

Figura 46

Guardado del modelo ResNeXt-50.

```
model.save('/working//ResNeXt_50_Modelo_128x128_vf-31')

converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
tflite_model_path = '/working/ResNeXt_50_Modelo_TFL.tflite'
with open(tflite_model_path, 'wb') as f:
    f.write(tflite_model)
```

Fuente: Elaboración propia

2.4. Desarrollo del sistema embebido

2.4.1. Especificaciones de hardware

El modelo seleccionado, se lo integro en un sistema embebido, en este caso una Raspberry Pi 3 Modelo B+ con las siguientes especificaciones de hardware:

- **Placa base Raspberry Pi 3 tipo B**+: Sistema embebido desde donde se alojará todo el proceso de detección de somnolencia.
- **Memoria RAM de 1GB**
- **MicroSD de 32GB:** Utilizado para instar el Sistema Operativo, así como para alojar diferentes archivos.
- **GPIO (General Purpose Input/Output) de 40 pines:** Entradas y salidas para propósito general utilizado para realizar la conexión del zumbador a la placa principal.
- **Puerto de cámara CSI:** Entrada serial para conexión con la cámara.
- **Cámara NIR OV5647:** Cámara de Infrarrojo Cercano (NIR) para capturar las imágenes que se enviaran al modelo.
- **Zumbador 3V:** Utilizado para emitir la alarma cuando se detecte somnolencia.
- **Alimentación 5V/ 2.5 A:** Utilizado para proporcionar energía a la placa principal y todos sus periféricos.

2.4.2. Especificaciones de software

Para realizar la programación y la integración del modelo con el sistema embebido se utilizaron las siguientes herramientas y librerías:

- **Raspberry Pi OS 32-bit (2021-12-02):** Sistema operativo instalado en la microSD con el que funcionara la placa principal.
- **IDE Geany versión 1.33:** Utilizado para realizar el script de Python donde se encuentra toda la lógica de detección de somnolencia.
- **Python versión 3.7.3:** Lenguaje de programación utilizado para programar la lógica.
- **Numpy versión 1.21.6**
- **OpenCV versión 4.1.0:** Utilizado para el manejo de imágenes.
- **TensorFlow Lite Runtime versión 2.11.0:** Utilizado para cargar y ejecutar el modelo ResNeXt-50. Es una versión optimizada para sistemas embebidos.

- **Imutils versión 0.5.4:** Utilizado para poder acceder a la cámara.
- **RPi.GPIO versión 0.7.1:** Librería que nos da acceso y control sobre los pines de la placa principal.

2.4.3. Integración del modelo ResNeXt-50 en la placa base

Para la integración del modelo con el sistema embebido se creó un entorno con las herramientas y librerías detalladas en la sección 2.4.2 “Especificaciones de software”. De esta manera se facilita la instalación y eliminación de librerías sin afectar otras librerías del sistema.

Se creo un script en Python donde se colocó toda la lógica para el funcionamiento del detector de somnolencia en el sistema embebido. El script se encarga de ejecutar diferentes funciones para poder obtener las predicciones de regresión del modelo ResNeXt-50 y utilizarlas para poder detectar somnolencias durante la conducción. A continuación, se detalla el proceso:

- **Importación de librerías y carga del modelo:** Se importaron las siguientes librerías necesarias para el proceso: numpy, cv2, tflite_runtime, imutils, y RPi.GPIO. Se creo un método para la carga del modelo seleccionado ResNeXt-50 y del detector de rostros Haard Cascade que utiliza OpenCV, como se muestra en la Figura 47.

Figura 47

Método para cargar el modelo ResNeXt-50 y el detector de rostro.

```
import numpy as np
import cv2
import tflite_runtime.interpreter as tflite
from imutils.video import VideoStream
import time
import RPi.GPIO as GPIO

def load_model_and_cascade():
    print("Cargando modelo ResNeXt-50 y detector de rostros...")
    tflite_model_path = '/home/pi/Marco/ResNeXt_50_Modelo_128x128-L_vf-28.2(5)-TFL.tflite'
    face_cascade = cv2.CascadeClassifier('/home/pi/Marco/haarcascade_frontalface_default.xml')

    interpreter = tflite.Interpreter(model_path=tflite_model_path)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    time.sleep(2.0)
    print("Modelo y detector de rostros cargados")
    return interpreter, face_cascade, input_details, output_details

```

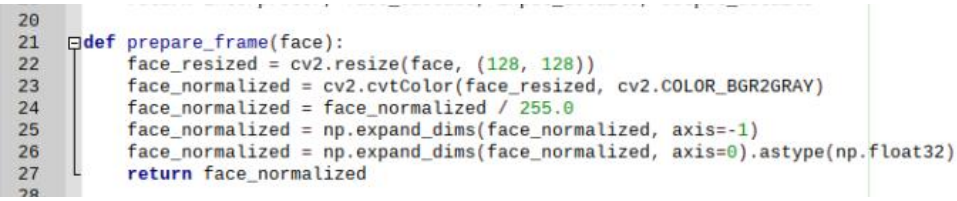
Fuente: Elaboración propia

- **Preprocesamiento de imágenes:** Función para preparar cada imagen capturada por la cámara y enviarla al modelo ResNeXt-50. En la Figura 48, se muestra el método para realizar la redimensión de imágenes a 128x128 pixeles y se lo transforma a escala de

grises, que es el tamaño y el canal de colores que acepta el modelo para realizar las predicciones.

Figura 48

Preprocesamiento de la imagen antes de enviase al modelo.



```
20
21  def prepare_frame(face):
22      face_resized = cv2.resize(face, (128, 128))
23      face_normalized = cv2.cvtColor(face_resized, cv2.COLOR_BGR2GRAY)
24      face_normalized = face_normalized / 255.0
25      face_normalized = np.expand_dims(face_normalized, axis=-1)
26      face_normalized = np.expand_dims(face_normalized, axis=0).astype(np.float32)
27      return face_normalized
```

Fuente: Elaboración propia

- **Cálculo del EAR (Eye Aspec Ratio):** Una función para calcular la Relación de Aspectos del Ojo (EAR) que representa un valor escalar que proporciona el estado de los ojos mientras se encuentran abiertos y cerrados. Para el cálculo de este valor se utiliza la Ecuación 1.

$$EAR = \frac{\|p2 - p6\| + \|p3 - p5\|}{2\|p1 - p4\|}$$

Ecuación 1: *Ecuación para el cálculo de la relación de aspecto ocular (EAR)*

Fuente: (Yu et al., 2024)

Donde se debe calcular la distancia euclidiana que se trata la distancia que existe entre dos puntos (x, y) en un espacio bidimensional, por ejemplo, la siguiente expresión representa la distancia euclidiana entre los puntos p2 y p6 $\|p2 - p6\|$". El modelo ResNeXt-50 en las predicciones de regresión, al enviar imágenes de un rostro nos devuelve 24 valores, que representan 12 coordenadas (x, y) de los puntos de los ojos, 6 para el ojo izquierdo y 6 para el derecho. Para calcular la distancia euclidiana utilizamos la función "*np.linalg.norm(p1-p2)*", que realiza este cálculo. Según esto se obtienen las diferentes distancias euclidianas según la Tabla 8

Tabla 8

Distancias euclidianas obtenidas de cada ojo.

Distancia euclidiana	Ojo izquierdo	Ojo derecho
Vertical 1	(p2, p6)	(p8, p12)

Vertical 2	(p3, p5)	(p9, p11)
Horizontal	(p1, p4)	(p7, p10)

Fuente: Elaboración propia

A continuación, para cado ojo lo reemplazamos en la formula y nos quedaría como se muestra en la Ecuación 2:

$$EAR = \frac{Vertical\ 1 + Vertical\ 2}{2 * Horizontal}$$

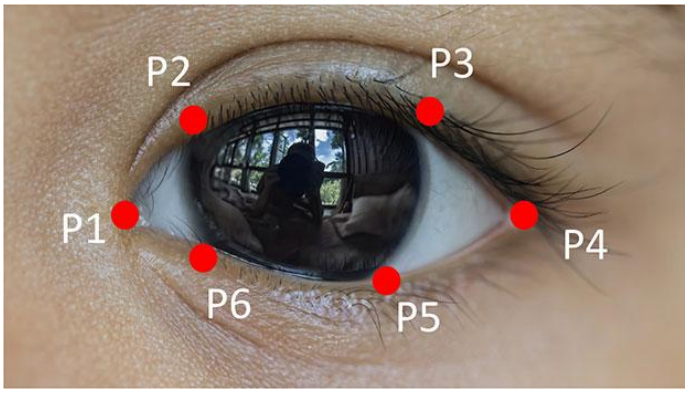
Ecuación 2: *Calculo EAR con las distancias euclidianas.*

Fuente: Elaboración propia

El cálculo del EAR se utilizar en dos funciones diferentes, de tal manera que se calcule el EAR de cada ojo para luego sacar un promedio de estos dos y usar el valor obtenido. Como observamos en la Figura 49, son los puntos que se utilizan para el cálculo del EAR de cada ojo.

Figura 49

Puntos de los ojos.



Fuente: (Thapa & Sarkar, 2024)

El método para realizar el cálculo EAR se muestra en la Figura 50, donde con la función mencionada de Numpy podemos calcular las tres distancias euclidianas. Al final se retorna el EAR aplicando la Ecuación 2.

Figura 50

Método para calcular el EAR.

```
29  def calculate_ear(eye_points):
30      vertical1 = np.linalg.norm(eye_points[1] - eye_points[5])
31      vertical2 = np.linalg.norm(eye_points[2] - eye_points[4])
32      horizontal = np.linalg.norm(eye_points[0] - eye_points[3])
33      return (vertical1 + vertical2) / (2.0 * horizontal)
34
```

Fuente: Elaboración propia

- **Configuración de pines GPIO para el zumbador:** Se creó un método para activar el pin 7 que permite emitir el sonido a través del zumbador a una frecuencia de 500 Hz como se muestra en la Figura 51.

Figura 51

Método que configura el GPIO.

```
35  def initialize_buzzer():
36      pin = 7
37      GPIO.setmode(GPIO.BOARD)
38      GPIO.setup(pin, GPIO.OUT)
39      pwm = GPIO.PWM(pin, 500)
40      pwm.start(0)
41      return pwm
```

Fuente: Elaboración propia

- **Cálculo del umbral EAR:** Debido a que los ojos de las personas son diferentes unos de otros, establecer un umbral fijo de EAR puede causar problemas el momento de detectar la somnolencia. Para esto se creó un método el cual calcula un umbral EAR para cada persona. Al iniciar el script de detección de somnolencia, como se muestra en la Figura 52, se establece el número de imágenes a 16 que son alrededor de 2 segundos, donde mediante el método "*calculate_ear()*" se calcula el EAR de cada ojo y luego se calcula el promedio de estos. Esto se realiza con los ojos abiertos o en un estado normal. El promedio obtenido en cada fotograma se almacena en un arreglo y posteriormente mediante la función "*np.mean()*" de Numpy obtendremos el promedio de esto valores. Este promedio se multiplica por 0.7 denominado factor de multiplicación o factor de reducción, el cual permite reducir el valor del promedio y así obtenemos un umbral que se encuentra por debajo del promedio de apertura normal de los ojos, en este caso nos quiere decir que los ojos se considerarán cerrados cuando el EAR actual sea menor al 70% del EAR promedio calculado.

Figura 52

Método para el cálculo del umbral EAR.

```python
def calibrate_ear_threshold(cap, face_cascade, input_details, output_details, interpreter):
    print("Calculando umbral EAR...")
    calibration_frames = 16
    calibration_data = []
    while len(calibration_data) < calibration_frames:
        frame = cap.read()
        if frame is None:
            continue

        gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(150, 150))

        if len(faces) > 0:
            for (x, y, w, h) in faces:
                expand_top = 0.28
                expand_bottom = 0.20
                expand_width = 0.45

                expand_width_value = int(expand_width * w)
                expand_height_top = int(expand_top * h)
                expand_height_bottom = int(expand_bottom * h)

                new_x = max(0, x - expand_width_value // 2)
                new_y = max(0, y - expand_height_top)
                new_w = w + expand_width_value
                new_h = h + expand_height_top + expand_height_bottom
                cv2.rectangle(frame, (new_x, new_y), (new_x + new_w, new_y + new_h), (255, 0, 0), 1)

                face = frame[new_y:new_y + new_h, new_x:new_x + new_w]
                prepared_face = prepare_frame(face)

                interpreter.set_tensor(input_details[0]['index'], prepared_face)
                interpreter.invoke()
                regression_output = interpreter.get_tensor(output_details[1]['index'])[0]
                landmarks = np.reshape(regression_output, (-1, 2)) * [new_w, new_h]
                landmarks += [new_x, new_y]

                ear_left = calculate_ear(landmarks[0:6])
                ear_right = calculate_ear(landmarks[6:12])
                ear_avg = (ear_left + ear_right) / 2.0

                calibration_data.append(ear_avg)
                break

        cv2.imshow("Calibración de Umbral EAR", frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    ear_threshold = np.mean(calibration_data) * 0.7
    print(f"umbral EAR calculado: {ear_threshold}")

```

Fuente: Elaboración propia

Para establecer este factor de multiplicación en el cálculo del umbral EAR se realizaron una serie de pruebas en 5 conjuntos de datos, cada conjunto consta de 40 imágenes pertenecientes a una sola persona, dividido en una proporción de 50:50 para ojos abiertos y cerrados. El proceso inicia con el cálculo del umbral EAR donde se utilizan 10 imágenes de ojos abiertos para obtener el promedio de estos. Este promedio lo vamos multiplicando por el factor de multiplicación que va desde 0.45 a 0.7. Finalmente, con el umbral obtenido calculamos el EAR de las imágenes y determinamos cuantas imágenes de ojos abiertos y cerrados fueron correctas para sacar un promedio. Todos esto utilizando el modelo ResNeXt-50 seleccionado.

Los resultados se muestran en la Tabla 9, donde con el factor de multiplicación 0.70 se obtiene un mayor promedio, con un 85.74% de precisión, por lo que en este trabajo se

estableció este valor para el factor de multiplicación en el cálculo del umbral EAR para cada persona.

Tabla 9
Cálculo de promedios en base a diferentes factores de multiplicación.

Factor de multiplicación	**Precisión (%)**					**Promedio %**
	Persona 1	**Persona 2**	**Persona 3**	**Persona 4**	**Persona 5**	
0.50	72.41	51.11	75.00	51.72	63.63	62.77
0.55	75.86	60.00	80.00	65.51	66.66	69.61
0.60	79.31	64.87	88.33	62.06	72.72	73.46
0.65	82.75	71.14	95.00	68.96	84.84	80.54
0.70	89.65	82.22	93.10	75.86	87.87	85.74
0.75	87.93	80.98	90.54	80.72	86.86	85.40
0.80	86.20	79.04	87.44	76.02	81.81	82.10

- **Proceso de detección de somnolencia:** El método de detección de somnolencia utiliza el detector de rostros y el modelo seleccionado ResNeXt-50. Donde se detecta el rostro en cada fotograma, este se envía el método "*prepare_frame()*" para el preprocesamiento del fotograma, posteriormente se envía al modelo ResNeXt-50 que realizara las predicciones de clasificación y regresión. Del modelo obtenemos las predicciones de regresión y mediante el método "*calculate_ear()*" obtenemos el valor de cada ojo para luego calcular es promedio de estos. En cada fotograma, este promedio es comparado con el umbral establecido, y en caso de que los ojos permanezcan cerrados por más de 1.5 segundos, se activara el zumbador hasta que el conductor abra los ojos. Este proceso se muestra en la Figura 53.

Figura 53

Método para la detección de somnolencia.

```
            for (x, y, w, h) in faces:

                expand_top=0.28
                expand_bottom=0.20
                expand_width=0.45

                expand_width_value=int(expand_width*w)
                expand_height_top=int(expand_top*h)
                expand_height_bottom=int(expand_bottom*h)

                new_x=max(0,x-expand_width_value//2)
                new_y=max(0,y-expand_height_top)
                new_w=w+expand_width_value
                new_h=h+expand_height_top+expand_height_bottom

                cv2.rectangle(frame,(new_x,new_y),(new_x+new_w,new_y+new_h),(255,0,0),1)

                face=frame[new_y:new_y+new_h,new_x:new_x+new_w]
                prepared_face=prepare_frame(face)

                interpreter.set_tensor(input_details[0]['index'],prepared_face)
                interpreter.invoke()
                regression_output=interpreter.get_tensor(output_details[1]['index'])[0]

                landmarks=np.reshape(regression_output,(-1,2))*[new_w,new_h]
                landmarks+=[new_x,new_y]

                ear_left=calculate_ear(landmarks[0:6])
                ear_right=calculate_ear(landmarks[6:12])
                ear_avg=(ear_left+ear_right/2.0)

                for (lx,ly) in landmarks:
                    cv2.putText(frame, f"Umbral EAR: {ear_threshold:.3f}",(10,30),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,255,0),1)
                    cv2.putText(frame, f"EAR Actual: {ear_avg:.3f}",(10,50),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,255,0),1)
                    cv2.circle(frame,(int(lx),int(ly)),1,(0,255,0),-1)

                if ear_avg < ear_threshold:
                    if eye_closure_start is None:
                        eye_closure_start=time.time()
                else:
                    eye_closure_start=None

                if eye_closure_start and (time.time()-eye_closure_start)>=1.5:
                    cv2.putText(frame, "SOMNOLENCIA DETECTADA",(10,70),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)
                    pwm.ChangeDutyCycle(50)
                    time.sleep(2)
                    pwm.changeDutyCycle(0)

                pass
        cv2.imshow('Detección de Somnolencia', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

Fuente: Elaboración propia

Capítulo 3

Resultados

La evaluación de los modelos se va a realizar por medio de las métricas de clasificación y regresión. Además, se realizarán pruebas al modelo integrados en el sistema embebido, estas pruebas serán simuladas y reales, de esta manera podremos conocer cuál es el rendimiento del modelo en diferentes entornos. Por último, se aplicará al sistema embebido la evaluación sobre la característica de portabilidad de la norma ISO/IEC 25023.

3.1. Evaluación de las métricas de Inteligencia Artificial

Los modelos entrenados presentaron resultados en métricas de clasificación, que incluye la perdida (Loss), exactitud (Accuracy), precisión, sensibilidad (Recall) y F1 Score, así como en métricas de regresión como el Error Cuadrático Medio (MSE), Error Absoluto Medio (MAE) y la Raíz del Error Cuadrático Medio (RMSE).

La Tabla 10, muestra las métricas de los dos modelos entrenados en el entorno de Google Colab. Se observa que el modelo entrenado con el Dataset balanceado obtiene una Perdida Combinada de 9.79% frente al 10.29% del desbalanceado, mientras que en términos clasificación el modelo entrenado con el Dataset balanceado alcanza una exactitud del 91.15% en comparación con el 90.47% del modelo entrenado con el Dataset desbalanceado. Las métricas de regresión entre los dos modelos muestran resultados similares con un MSE de 0.28% y 0.29%, MAE de 3.74% y 3.76%, y RMSE de 5.35% y 5.46% respectivamente. El tiempo de entrenamiento con el Dataset balanceado fue menor con un total de 3 horas y 50 minutos en comparación con el Dataset desbalanceado con un total de 4 horas y 37 minutos, esto debido a que el Dataset desbalanceado tiene una mayor cantidad de datos.

Tabla 10

Métricas y tiempos de entrenamiento de los modelos en Google Colab.

Dataset	Combine Loss (%)	Clasificación					Regresión			Tiempo de entrenamiento
		Loss (%)	Exactitud (%)	Precisión (%)	Recall (%)	F1 Score (%)	MSE (%)	MAE (%)	RMSE (%)	
Balanceado	9.79	24.08	91.15	91.47	90.85	91.15	0.28	3.74	5.35	3hrs 50min
Desbalanceado	10.29	25.28	90.47	91.63	89.15	90.30	0.29	3.76	5.46	4hrs 37min

Fuente: Elaboración propia

Por otro lado, al entrenar los modelos en el entorno HPC-CEDIA obtenemos una Perdida Combinada de 9.02% para el modelo entrenado con el Dataset balanceado y 7.21% para el modelo entrenado con el Dataset desbalanceado. Una exactitud del 93.06% y 91.74% respectivamente. Las métricas de regresión muestran resultados similares tanto para el modelo balanceado y el desbalanceado con un MSE de 0.295 y 0.31%, MAE de 3.78% y 3.96%, y RMSE de 5.40% y 5.57% respectivamente. El tiempo de entrenamiento para el modelo con el Dataset balanceado es de 1 hora y 37 minutos en comparación con el otro modelo que tomo 2 horas y 40 minutos. Esta diferencia de tiempo es debido a una mayor cantidad de datos en el Dataset desbalanceado. Los resultados se muestran en la Tabla 11.

Tabla 11

Métricas y tiempo de entrenamiento de los modelos en HPC-CEDIA.

Dataset	**Combine Loss (%)**	**Clasificación**					**Regresión**			**Tiempo de entrenamiento**
		Loss (%)	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1 Score (%)**	**MSE (%)**	**MAE (%)**	**RMSE (%)**	
Balanceado	9.02	22.11	93.06	92.93	93.02	92.97	0.29	3.78	5.40	1hr 37min
Desbalanceado	7.21	17.56	91.74	92.60	85.39	88.85	0.31	3.96	5.57	2hrs 40min

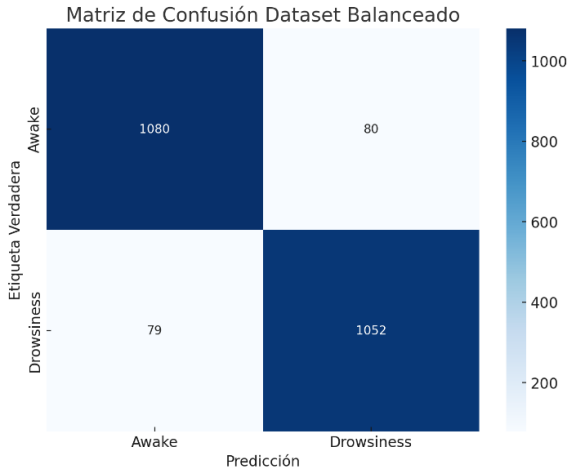
Fuente: Elaboración propia

En base a estos dos entornos utilizados para el entrenamiento, observamos que se obtienen mejores métricas y mejores tiempos de entrenamiento al usar el HPC CEDIA debido a que cuenta con hardware más potente. Por lo tanto, se eligieron los modelos entrenados en este entorno para continuar con las evaluaciones.

Para cada modelo seleccionado se obtuvo su matriz de confusión y sus respectivos gráficos de la evolución de métricas durante el entrenamiento. En la Figura 54, observamos la matriz de confusión del modelo entrenado con el Dataset balanceado donde las etiquetas verdades “Awake” (Despierto) y “Drowsiness” (Dormido) se muestra en el eje vertical y las predicciones del modelo en el eje horizontal. No indica 1052 verdaderos positivos para “Awake”, 80 falsos positivos de “Drowsiness” clasificados como “Awake”, 79 falsos negativos de “Awake” clasificados como “Drowsiness” y 1052 verdaderos positivos para “Drowsiness”.

Figura 54

Matriz de confusión del modelo con el Dataset balanceado.

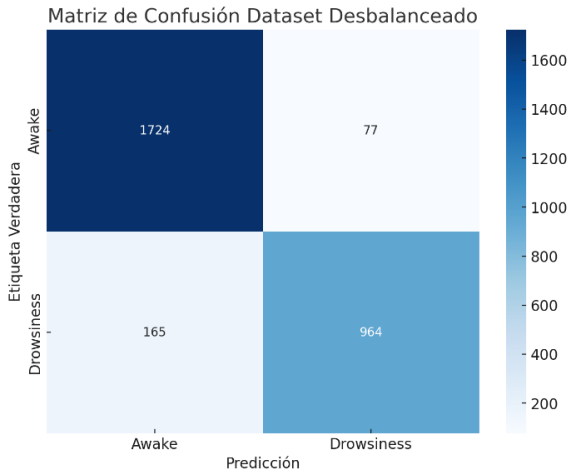


Fuente: Elaboración propia

Se obtuvo la matriz de confusión para el modelo entrenado con el Dataset desbalanceado, como se muestra en la Figura 55. Se observa que el modelo ha clasificado correctamente 1724 instancias como “Awake” y 964 como “Drowsiness” que son verdaderos positivos. Hay 165 falsos negativos donde “Awake” fue clasificado como “Drowsiness” y 77 falsos positivos donde “Drowsiness” fue clasificado como “Awake”

Figura 55

Matriz de confusión del modelo con el Dataset desbalanceado.

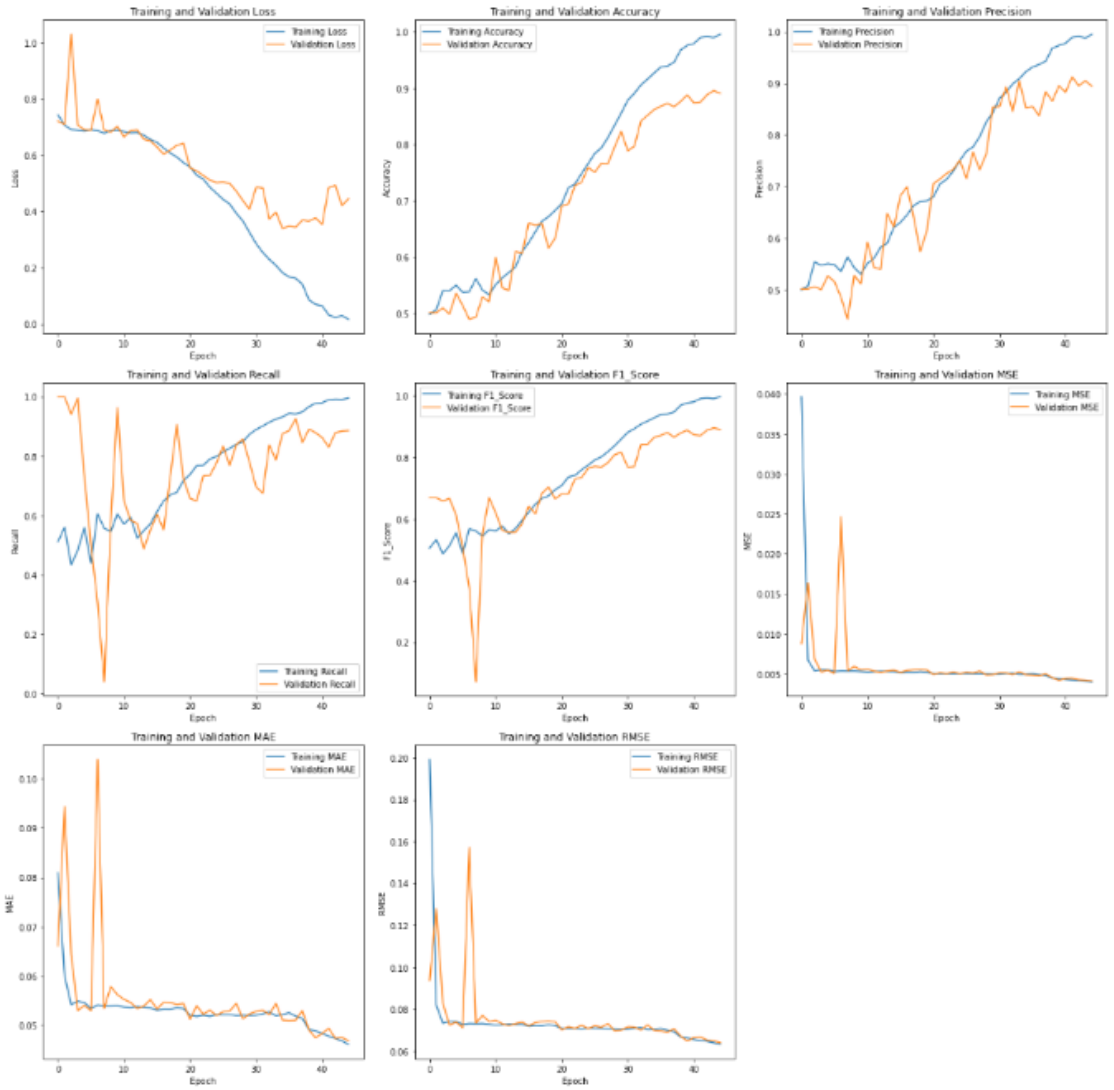


Fuente: Elaboración propia

Durante las fases de entrenamiento y validación, se monitorearon las métricas de clasificación y regresión mencionadas anteriormente. La Figura 56, muestra las gráficas del modelo entrenado con el Dataset balanceado, donde las líneas azules indican el rendimiento durante el entrenamiento y las naranjas durante la validación.

Figura 56

Métricas de clasificación y regresión durante el entrenamiento y validación del modelo balanceado.

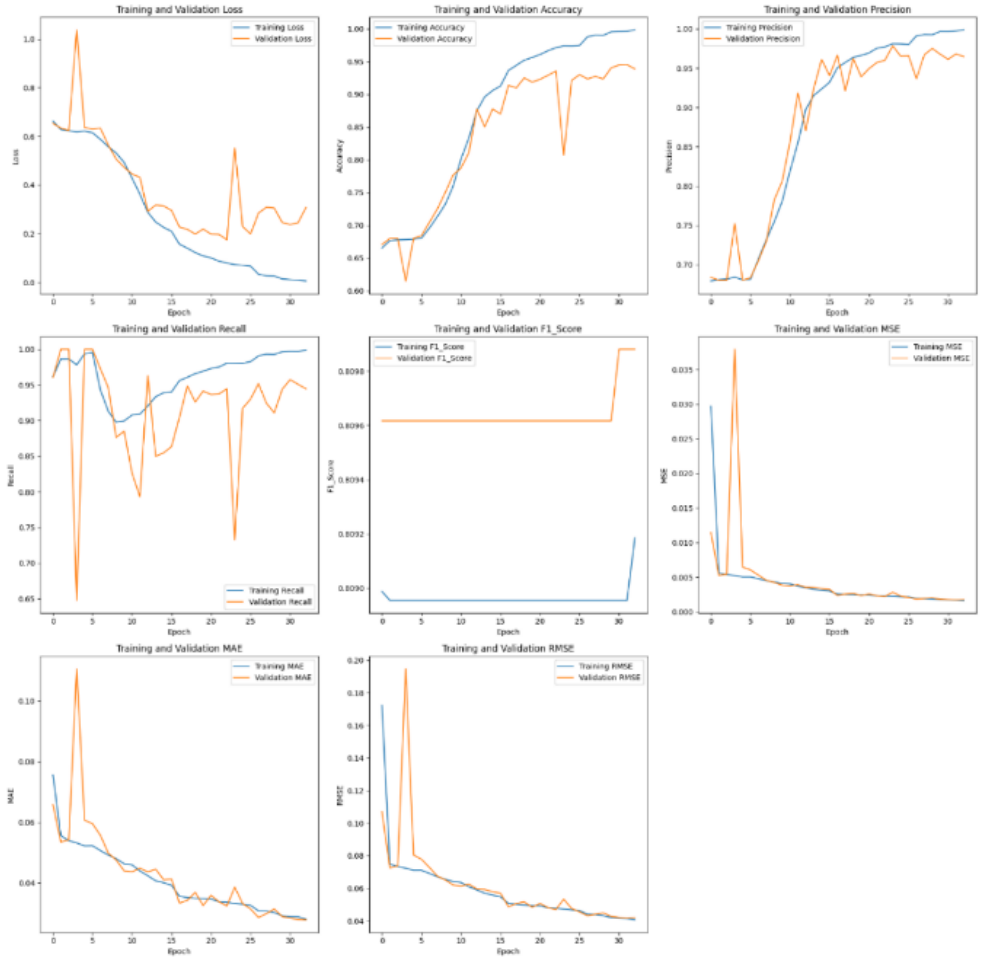


Fuente: Elaboración propia

La Figura 57, muestra las gráficas de las métricas de clasificación y regresión del modelo entrenado con el Dataset desbalanceado.

Figura 57

Métricas de clasificación y regresión durante el entrenamiento y validación del modelo desbalanceado.



Fuente: Elaboración propia

Estas graficas son importantes para interpretar como los modelos van evolucionando durante cada época, lo que nos ayudara a evaluar como nuestro modelo se comporta con los datos y la identificación de sobreajustes. Esto permite tomar decisiones sobre modificaciones en hiperparámetros y conjunto de datos para el mejoramiento de estas.

3.1.1. Evaluación con diferentes conjuntos de pruebas

Se evaluaron los dos modelos seleccionados con cinco conjuntos de datos diferentes: rostro sin accesorios (bareface), rostro con lentes (glasses), rostro con gafas de sol (sunglasses), condiciones diurnas y nocturnas. Las métricas que se obtendrán ayudarán a la selección del

mejor modelo, ya sea el entrenado con el Dataset balanceado o con el desbalanceado, dependiendo de las mejores métricas tanto de clasificación como de regresión.

3.1.1.1. Métricas con accesorios faciales

Para determinar qué modelo es mejor con respecto a los accesorios que puede llevar una persona en el rostro (como lentes o gafas), se evaluó el modelo utilizando tres conjuntos de datos distintos. Un conjunto con imágenes de rostros sin ningún accesorio facial, otro conjunto con imágenes de rostros con lentes, y un tercer conjunto de rostros con gafas. Cada conjunto con imágenes tomadas tanto en el día como en la noche.

En la Tabla 12, se muestra la comparación de métricas entre estos tres conjuntos de datos obtenidas con el modelo entrenado en el Dataset balanceado. En la Perdida Combinada, se observa un mejor resultado con el conjunto de prueba de rostros sin accesorios con un 20.58%, seguido por el conjunto de rostros con lentes con un 23.45% y finalmente el conjunto de rostros con gafas con un 28.74%. en la clasificación, para cada conjunto de prueba se obtiene una exactitud del 93.50%, 92.25% y 88.68% respectivamente. Esto nos indica que el modelo es mejor cuando el rostro esta sin accesorios faciales.

Tabla 12

Métricas de accesorios faciales obtenidos con el modelo entrenado en el Dataset balanceado.

Modelo con Dataset Balanceado	**Combine Loss (%)**	**Clasificación**					**Regresión**		
		Loss (%)	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1Score (%)**	**MSE (%)**	**MAE (%)**	**RMSE (%)**
Bareface	20.58	20.34	93.50	92.23	95.00	93.60	0.24	3.67	4.98
Glasses	23.45	23.24	92.25	91.90	92.67	92.28	0.20	3.41	4.56
Sunglasses	28.74	28.41	88.68	88.83	94.67	91.65	0.32	4.27	5.68
Promedio	24.26	24.00	91.48	90.98	94.11	92.51	0.25	3.78	5.07

Fuente: Elaboración propia

La Tabla 13, muestra los resultados que obtuvo el modelo entrenado con el Dataset desbalanceado. La pérdida combinada en el conjunto de prueba de rostros sin accesorios fue del 21.72%, para conjunto de rostros con lentes fue de 27.32% y para el conjunto de rostros con gafas fue del 27.51%. Además, para estos tres conjuntos se obtuvieron una exactitud del 91.91%, 89.74% y 89.26% respectivamente.

Tabla 13

Métrica de accesorios faciales obtenidos con el modelo entrenado en el Dataset desbalanceado

Modelo con Dataset Desbalanceado	Combine Loss (%)	Clasificación					Regresión		
		Loss (%)	Exactitud (%)	Precisión (%)	Recall (%)	F1Score (%)	MSE (%)	MAE (%)	RMSE (%)
Bareface	21.72	21.42	91.91	95.32	88.17	91.60	0.30	4.03	5.49
Glasses	27.32	27.09	89.74	84.09	84.83	89.22	0.23	3.61	4.87
Sunglasses	27.51	27.23	89.26	85.11	77.82	81.30	0.28	3.94	5.33
Promedio	25.52	25.25	90.30	88.17	83.60	87.37	0.27	3.86	5.23

Fuente: Elaboración propia

3.1.1.2. Métricas en condiciones de luz diurna y nocturna

Para determinar qué modelo es mejor según las condiciones de luz, diurna y nocturna, se evaluó el modelo con dos conjuntos de datos, uno con imágenes de rostros durante el día, y otro con imágenes de rostros tomadas durante la noche.

La Tabla 14, muestra los resultados obtenidos del modelo entrenado con el Dataset balanceado. La pérdida combinada con imágenes de rostros durante el día y la noche son del 27.87% y 16.96% respectivamente, obteniendo una menor perdida con el segundo conjunto. Se alcanzó una exactitud del 89.39% con las imágenes diurnas y 94.42% con las nocturnas. Los datos indican que el modelo es mejor con imágenes capturadas en condiciones nocturnas.

Tabla 14

Métricas de condiciones de luz obtenidos con el modelo entrenado en el Dataset balanceado.

Modelo con Dataset Balanceado	Combine Loss (%)	Clasificación					Regresión		
		Loss (%)	Exactitud (%)	Precisión (%)	Recall (%)	F1Score (%)	MSE (%)	MAE (%)	RMSE (%)
Día	27.87	27.62	89.39	93.46	89.28	91.32	0.25	3.71	5.04
Noche	16.96	16.65	94.42	96.65	94.82	95.73	0.30	3.91	5.52
Promedio	22.42	22.14	91.91	95.06	92.05	93.53	0.28	3.81	5.28

Fuente: Elaboración propia

La Tabla 15, muestra los resultados que obtuvo el modelo entrenado con el Dataset desbalanceado. La pérdida combinada para el conjunto con imágenes diurnas es del 25.91% y para el conjunto con imágenes nocturnas es del 17.86%. La exactitud para el primer conjunto alcanza el 89.18% y el segundo alcanza el 91.83%. De igual manera, este modelo presenta un mejor desempeño con imágenes nocturnas.

Tabla 15

Métricas de condiciones de luz obtenidas con el modelo entrenado en el Dataset desbalanceado.

Modelo con Dataset Desbalanceado	Combine Loss (%)	Clasificación					Regresión		
		Loss (%)	Exactitud (%)	Precisión (%)	Recall (%)	F1Score (%)	MSE (%)	MAE (%)	RMSE (%)
Día	25.91	25.63	89.18	94.04	83.67	88.55	0.27	3.81	5.20
Noche	17.86	17.51	91.83	97.69	85.71	91.31	0.34	4.17	5.91
Promedio	21.89	21.57	90.50	95.86	84.69	89.93	0.31	3.99	5.56

Fuente: Elaboración propia

3.1.1.3. Error Medio Normalizado (NME)

Se opto por el Error Medio Normalizado (NME) para evaluar los modelos en términos de regresión. El Error Medio Normalizado se define como la distancia euclidiana media entre las ubicaciones de los puntos de referencia faciales predichos por el modelo *PPredicho (Punto Predicho)* y su correspondiente valor real *PReal (Punto Real)*, dividido por un factor de normalización *d* utilizando la siguiente formula (Xu et al., 2020).

$$NME = \frac{1}{N}\sum_{i=1}^{N}\frac{\|PPredicho - PReal\|2}{d}$$

Esta métrica es común utilizada para evaluar la precisión de algoritmos enfocados en la detección de puntos de referencia faciales. Donde *N* es el número de imágenes y *d* es el factor de normalización. Con nuestro Dataset se usará la distancia '*inter-ocular*' como factor de normalización.

Los resultados que obtuvo cada modelo se muestran en la Tabla 16, donde para el modelo balanceado se obtienen un NME de 07.16% y un AUC de 0.62, mientras que para el modelo desbalanceado se obtiene un NME de 07.03% y un AUC de 0.68. Esto quiere decir que en promedio los 12 puntos de los ojos predichos por el modelo balanceado se desvían 07.16% de la distancia '*inter-ocular*' con respecto a sus ubicaciones reales. Mientras que las predicciones obtenidas por el modelo desbalanceado se desvían un 07.03%, es decir que mientras este valor se acerque más al cero la desviación de los puntos es menor.

Tabla 16

Error Medio Normalizado (NME) y el Área Bajo la Curva (AUC) de los modelos.

Modelo	NME (%)	AUC
Dataset balanceado	07.16	0.62
Datase desbalanceado	07.03	0.68

Fuente: Elaboración propia

3.1.1.4. Tasa de Fallo (FR)

La Tasa de Fallo (FR) es una métrica que evalúa la eficacia del modelo en términos de NME, donde nos indica el porcentaje de imágenes que obtienen un NME por encima de un umbral establecido (Prados-Torreblanca et al., 2022). Para este trabajo el umbral fue establecido en 0.10, lo que nos indica que cualquier valor de NME que exceda dicho umbral será considerado como fallo o como resultados no aceptables. Se eligió este valor ya que, en varios artículos utilizados en este trabajo, los autores lo establecen como un estándar dentro de esta tarea de predecir puntos de referencias faciales. Se estableció ese valor al umbral ya que es utilizado en artículos como lo hace (Dang et al., 2024), para realizar comparaciones con otros modelos.

Para el cálculo de esta métrica se utilizó un conjunto de datos con 2000 imágenes respectivamente etiquetadas. La Table 17, muestra los resultados de la métrica obtenidos en los dos modelos balanceado y desbalanceado, obteniendo una tasa de fallos de 17.87% y 15.97% respectivamente. Esto nos quiere decir con el primer modelo el 17.87% de las imágenes obtuvieron un NME superior al umbral 0.10, mientras que con el segundo modelo el 15.97% de las imágenes obtuvieron un NME superior al umbral. Esto nos indica que el modelo balanceado es mejor manteniendo la precisión dentro del límite aceptable reduciendo la cantidad de errores con respecto al modelo desbalanceado.

Tabla 17

Tasa de Error obtenidos en los modelos.

Modelo	FR (%)
Dataset balanceado	17.87

Dataset desbalanceado	15.97

Fuente: Elaboración propia

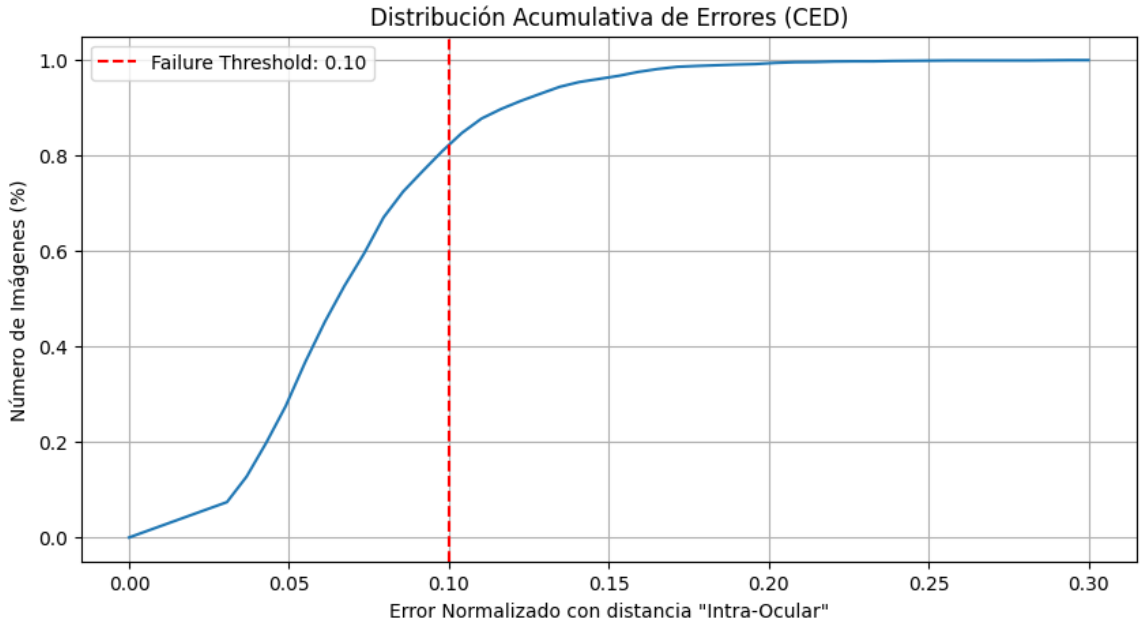
3.1.1.5. Distribución Acumulativa de Errores (CED)

La curva de Distribución Acumulativa de Errores (CED) es una representación gráfica que muestra el error acumulativo en la detección de puntos faciales a partir de un conjunto de imágenes. Cada imagen se analiza individualmente para determinar el error de predicción de los puntos faciales, y estos errores se acumulan para generar la gráfica.

La Figura 58, muestra la curva CED obtenida con el modelo balanceado, así también como una línea roja entrecortada que nos indica el Umbral de Fallo establecido en 0.10, este nos indica el límite aceptable de NME, por lo tanto, las imágenes que obtenga un NME superior al límite se los considera como predicciones no aceptables. El porcentaje de fallo que se obtuvo para este modelo se lo muestra en la Tabla 16.

Figura 58

Distribución Acumulativa de Errores obtenida por el modelo balanceado.

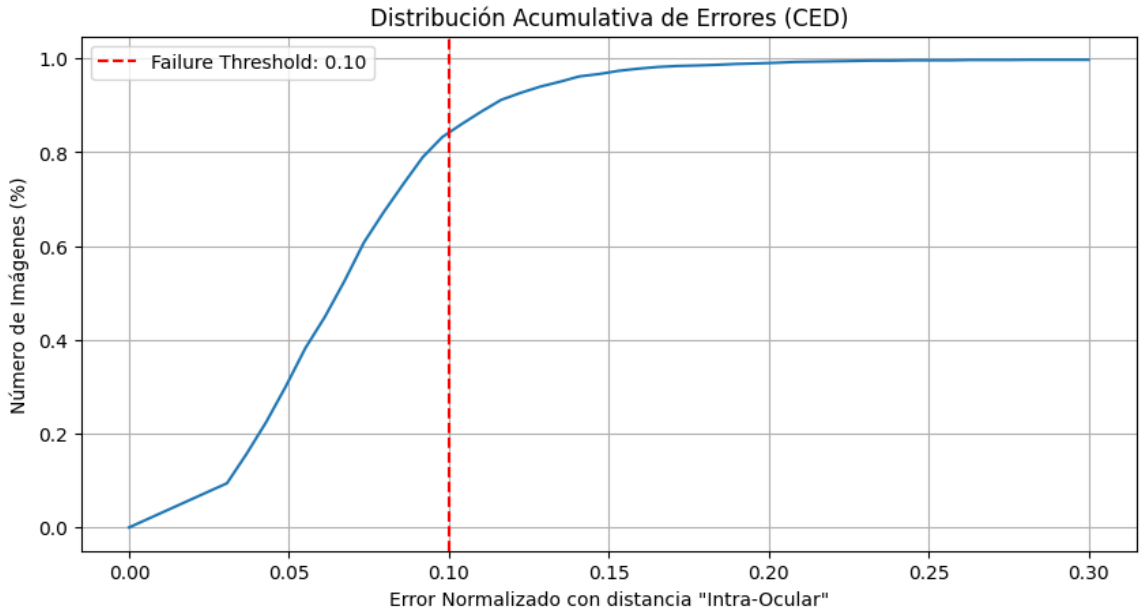


Fuente: Elaboración propia

La grafica obtenida para el modelo desbalanceado se muestra en la Figura 59, donde se observa la curva CED y el Umbral de Fallo.

Figura 59

Distribución Acumulativa de Errores obtenida por el modelo desbalanceado.



Fuente: Elaboración propia

3.1.2. Selección del mejor modelo

De acuerdo con las evaluaciones realizados a ambos modelos se obtienen los siguientes resultados que se muestran en la Tabla 18, donde los resultados del modelo balanceado indican un rendimiento más alto y uniforme sin tanta diferencia en los promedios de las diferentes métricas obtenidas que son las siguientes: exactitud del 92.15%, precisión del 92.99%, recall del 93.06% y f1 score del 93.00%, lo que nos muestra una capacidad muy buena para generalizar datos. Por otra parte, los resultados del modelo desbalanceado muestran resultados casi uniformes en las métricas de exactitud y precisión con 90.84% y 92.21% respectivamente, sin embargo, muestra una caída en las métricas recall y f1 score con 84.56% y 88.71% respectivamente, lo que nos indica que el modelo tiene dificultades para identificar correctamente los casos positivos, es decir tiene problemas para detectar somnolencia (drowsiness).

Tabla 18

Comparativa de métricas de los modelos balanceado y desbalanceado.

Conjunto de datos	Modelo entrenado con el Dataset balanceado	Modelo entrenado con el Dataset desbalanceado

	Exactitud (%)	Precisión (%)	Recall (%)	F1 Score (%)	Exactitud (%)	Precisión (%)	Recall (%)	F1 Score (%)
General	93.06	92.93	93.02	92.97	91.74	92.60	85.39	88.85
Accesorios faciales	91.48	90.98	94.11	92.51	90.30	88.17	83.60	87.37
Condiciones de luz	91.91	95.06	92.05	93.53	90.50	95.86	84.69	89.93
Promedio	**92.15**	**92.99**	**93.06**	**93.00**	**90.84**	**92.21**	**84.56**	**88.71**

Fuente: Elaboración propia

En base a este análisis, se seleccionó el modelo entrenado en el Dataset balanceado para continuar con la comparación con modelos de la literatura.

3.1.3. Comparación de métricas con la literatura

El modelo seleccionado fue comparado con artículos relacionados con nuestro trabajo que estén enfocados ya sea en clasificación o regresión. Esto permitió evaluar la eficacia de nuestro modelo, el cual está diseñado para realizar ambas tareas.

La Tabla 19, muestra la comparativa de la exactitud (accuracy) de diferentes modelos aplicadas en un contexto general de detección de somnolencia aplicando clasificación de imágenes. Nuestro modelo ResNeXt-50, alcanza una exactitud de 93.06%. El modelo ResNet-50 en el estudio de (Minhas et al., 2022) alcanza el 93.69%. Además, los estudios de (Ahmed et al., 2023) y (Pandey & Muppalaneni, 2023) alcanzan exactitudes más altas de 97.00% y 97.50% respectivamente. Siendo estes modelos mencionados los más destacados, sin embargo, existen más modelos con resultados bajos que se detallan en la misma tabla.

Tabla 19

Comparación general de resultados de clasificación (rostro completo) con otros modelos.

Referencia	Modelo	Loss (%)	Exactitud (%)	Precisión (%)	Recall (%)	F1 Score (%)
Nuestro modelo	ResNext-50	22.11	93.06	92.93	93.02	92.97
(Minhas et al.,	InceptionV3	69.31	90.70	-	-	-
2022)	ResNet-50	69.31	93.69	-	-	-
	VGG16	69.31	39.87	-	-	-
	Facial Landmark with LightGBM	-	85.90	-	-	-

(Dua et al., 2021)	AlexNet	-	76.48	-	-	-
	VGG-FaceNet	-	87.09	-	-	-
	FlowImageNet	-	83.11	-	-	-
	ResNet	-	81.34	-	-	-
(El Zein et al., 2024)	VAS-3D	-	95.50	-	-	-
(Lee et al., 2024)	3D-CNN basado en GoogleNet Inception	-	85.00	-	-	-
(Ahmed et al., 2023)	CNN	-	97.00	96.00	96.25	96.25
	VGG16	-	74.00	68.75	67.65	61.25
(Pandey & Muppalaneni, 2023)	TCBR-LSTM	-	86.00	85.29	87.00	-
	CNN-LSTM	-	97.50	97.97	97.00	-

Fuente: Elaboración propia

Otros estudios evaluaron métricas específicas para la clasificación de somnolencia en imágenes de rostros sin accesorios faciales. Nuestro modelo obtiene una exactitud de 93.50%, el estudio de (Li & Li, 2024) con el modelo PFLD – ViT – LSTM obtiene 95.26%, siendo estos dos los modelos con mejores resultados en esta categoría. Además, estudios como (Jamshidi et al., 2021) con el modelo ResNet-18 - VGG16 - LSTM, y (W. Deng & Wu, 2019) con el modelo MC-KCF, obtienen exactitudes de 92.73% yo 92.10%, que son resultados muy buenos. La comparativa se muestra en la Tabla 20.

Tabla 20

Comparación de resultados obtenidos en imágenes de rostros sin accesorios faciales.

Referencia	**Modelo**	**Loss (%)**	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1 Score (%)**
Nuestro modelo	ResNext-50	20.34	93.50	94.84	92.00	93.40
(Jamshidi et al., 2021)	ResNet-18 - VGG16 - LSTM	-	92.73	-	-	-

(W. Deng & Wu, 2019)	MC-KCF	-	92.10	-	-	-
(Li & Li, 2024)	PFLD – ViT - LSTM	-	95.26	-	-	-

Fuente: Elaboración propia

Evaluando el modelo en imágenes de rostros con lentes, nuestro modelo alcanzo una exactitud del 92.25%, y junto con los estudios de (W. Deng & Wu, 2019) y (Li & Li, 2024), con 91.55% y 91.6% respectivamente, son modelos con mejores resultados. Modelos como 3DDGAN, 3DDGAN-LA de (Bearly & Chitra, 2024) y ResNet-18 - VGG16 – LSTM de (Jamshidi et al., 2021) obtiene exactitudes de 80.95%, 85.50% y 84.19%, indicando resultados más bajos. La Tabla 21, muestra la comparativa.

Tabla 21

Comparación de resultados obtenidos en imágenes de rostros con lentes.

Referencia	**Modelo**	**Loss (%)**	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1 Score (%)**
Nuestro modelo	ResNext-50	23.24	92.25	92.60	91.83	92.21
(Bearly & Chitra, 2024)	3DDGAN	-	80.95	-	-	-
	3DDGAN - LA	-	85.50	-	-	-
(Jamshidi et al., 2021)	ResNet-18 - VGG16 - LSTM	-	84.19	-	-	-
(W. Deng & Wu, 2019)	MC-KCF	-	91.55	-	-	-
(Li & Li, 2024)	PFLD – ViT - LSTM	-	91.61	-	-	-

Fuente: Elaboración propia

Al evaluar nuestro modelo con imágenes de rostros con gafas, nuestro modelo obtuvo una exactitud del 88.68%, y el modelo de (Li & Li, 2024) obtuvo 91.10%, siendo estas dos los que tuvieron mejores resultados. Por otra parte, se obtienen loa siguientes resultados de exactitud con estos modelos: (Bearly & Chitra, 2024) y sus modelos 3DDGAN y 3DDGAN + LA con 80.10% y

86.30%, (Jamshidi et al., 2021) y el modelo ResNet-18 - VGG16 – LSTM con 82.09%, y (Lee et al., 2024) y el modelo 3D-CNN basado en GoogleNet Inception con 75.00%. Estos resultados se muestran en la Tabla 22.

Tabla 22

Comparación de resultados obtenidos en imágenes de rostros con gafas.

Referencia	**Modelo**	**Loss (%)**	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1 Score (%)**
Nuestro modelo	ResNext-50	28.41	88.68	94.67	88.83	91.65
(Bearly & Chitra, 2024)	3DDGAN	-	80.10	-	-	-
	3DDGAN - LA	-	86.30	-	-	-
(Jamshidi et al., 2021)	ResNet-18 - VGG16 - LSTM	-	82.09	-	-	-
(Lee et al., 2024)	3D-CNN basado en GoogleNet Inception	-	75.00	-	-	-
(Li & Li, 2024)	PFLD – ViT - LSTM	-	91.10	-	-	-

Fuente: Elaboración propia

Además, utilizando el Error Medio Normalizado (NME) y el Área Bajo la Curva (AUC) como métricas para evaluar nuestro modelo en términos de regresión, lo comparamos con otros trabajos de la literatura. En la Tabla 23, se muestra el valor obtenido por nuestro modelo que obtiene un NME del 07.16%, siendo un valor significativamente alto con respecto a otros trabajos, donde (Dang et al., 2024) con el modelo D-ViT obtiene 3.75%, (Kar et al., 2024) con el modelo FiFA obtiene 3.89% y (Prados-Torreblanca et al., 2022) con el modelo SPIGA obtiene 4.06% de NME. Con respecto al área bajo la curva (AUC) obtenemos un valor de 0.62, mientras que los modelo de (Dang et al., 2024) y (Prados-Torreblanca et al., 2022) obtienen 0.63 y 0.60 de AUC respectivamente. Por otra parte, se realiza la comparación sobre la métrica Tasa de Fallo (FR), donde nuestro modelo obtiene un FR de 17.87%, siendo este el que más taza de fallos obtiene, el trabajo de (Dang et al., 2024), (Kar et al., 2024) y (Prados-Torreblanca et al., 2022) obtienen un FR 1.76%, 1.60% y 2.08% respectivamente, donde obtienen resultados similares con

taza de fallos bajos lo que indica un mayor rendimiento de cada modelo al momento de predecir los puntos de referencia facial.

Tabla 23

Comparación del Error Medio Normalizado con trabajos relacionados.

Referencia	**Modelo**	**NME (%)**	**AUC**	**FR (%)**
Nuestro modelo	ResNeXt-50	07.16	0.62	17.87
(Dang et al., 2024)	D-ViT	03.75	0.63	1.76
(Kar et al., 2024)	FiFA	03.89	-	1.60
(Prados-Torreblanca et al., 2022)	SPIGA	04.06	0.60	2.08

Fuente: Elaboración propia

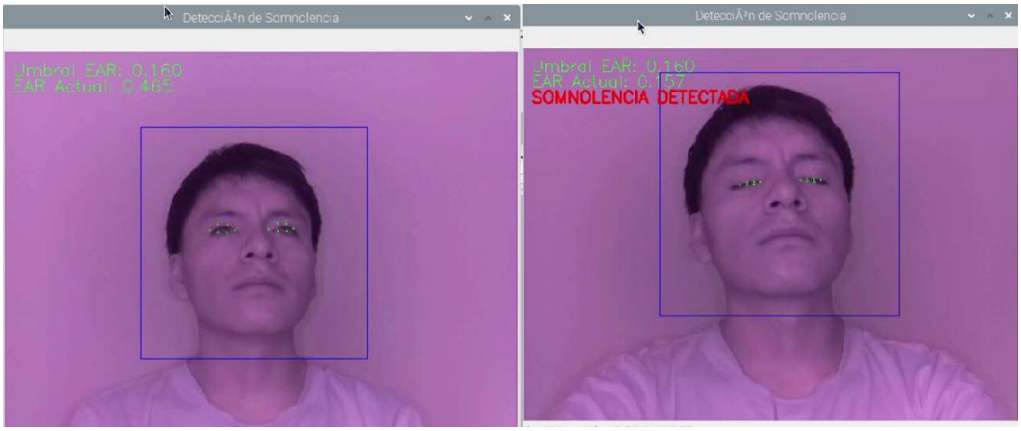
3.2. Diseño y ejecución de experimentos

El sistema embebido se colocó en un entorno que simula la cabina del conductor donde se realizaron pruebas de detección de somnolencia. Los experimentos se lo realizaron en dos sesiones: uno durante el día y otro durante la noche, de esta manera evaluamos la eficiencia del modelo en estas condiciones.

En la Figura 60, se observa la prueba realizada durante el día. El umbral EAR de la persona se estableció en 0.160, lo que significa que si el EAR Actual se encuentra por debajo del umbral por más de 1.5 segundos, se activara la alarma. A la izquierda se muestra a la persona con el estado de los ojos normales y un EAR Actual de 0.465, mientras que a la derecha se encuentra la persona con el estado de los ojos dormidos y un EAR Actual de 0.157 lo que significa que la alarma se activó y muestra un mensaje de alerta.

Figura 60

Pruebas de simulación realizadas durante el día.

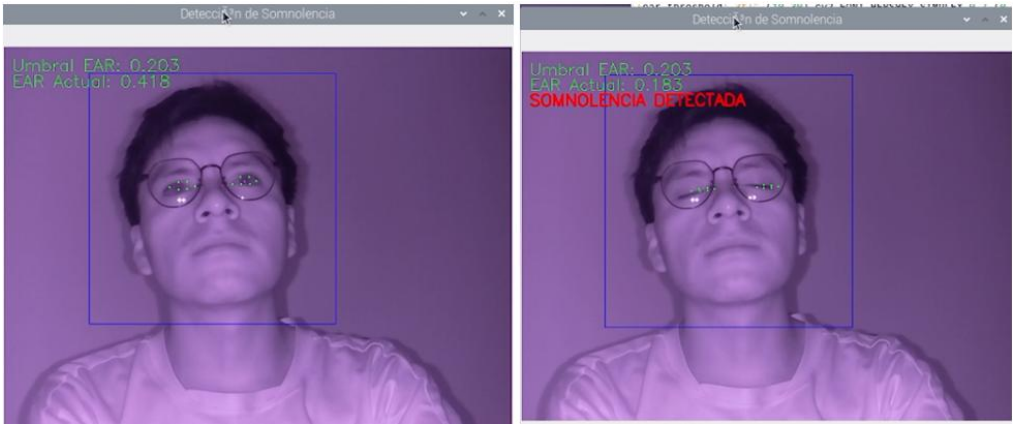


Fuente: Elaboración propia

Durante la prueba nocturna y con el uso de la cámara NIR, se obtienen resultados bastantes precisos como se muestran en la Figura 61, se obtiene un umbral EAR de 0.203. Durante el estado normal de los ojos el EAR es de 0.418, mientras que durante el estado somnoliento el EAR es de 0.183.

Figura 61

Pruebas de simulación realizadas durante la noche.



Fuente: Elaboración propia

3.3. Implementación de pruebas

Las pruebas se realizaron en dos vehículos diferentes para poder evaluar su adaptabilidad en diferentes entornos. Los modelos para la implementación fueron una camioneta Chevrolet D-MAX y un camino Isuzu que se muestran en la Figura 62.

Figura 62

Camión y camioneta utilizados para las pruebas.

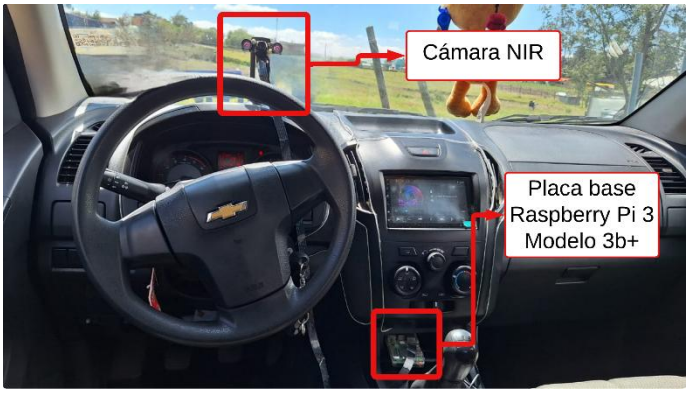
Fuente: Elaboración propia

3.3.1. Instalación del sistema embebido

En la camioneta Chevrolet la cámara se colocó sobre el tablero, justo delante del volante debido a que en esta ubicación se obtienen imágenes más claras y centradas del rostro. La placa base se colocó en un compartimento justo al lado del encendedor junto con la fuente de alimentación, como se muestra en la Figura 63. Esta ubicación de la cámara y de la palca base no obstruyen la visión del conductor.

Figura 63

Ubicación de la cámara y placa base en la camioneta.

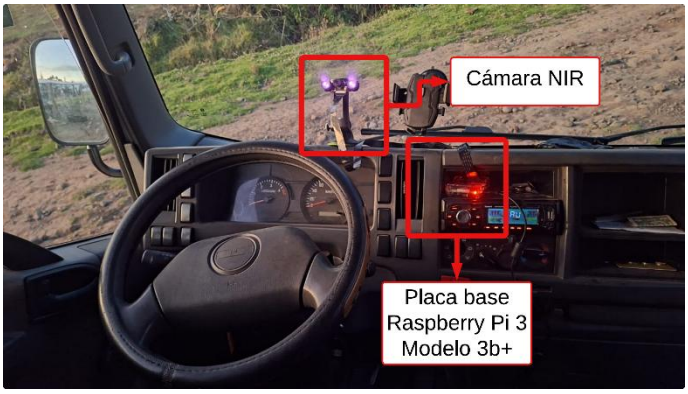


Fuente: Elaboración propia

En el camión, la cámara fue colocada sobre el tablero frente al volante, y la placa base sobre un compartimento disponible sobre la radio como se muestra en la Figura 64. Estas ubicaciones no incomodaron y no obstruyeron la visibilidad del conductor.

Figura 64

Ubicación de la cámara y placa base en el camión.



Fuente: Elaboración propia

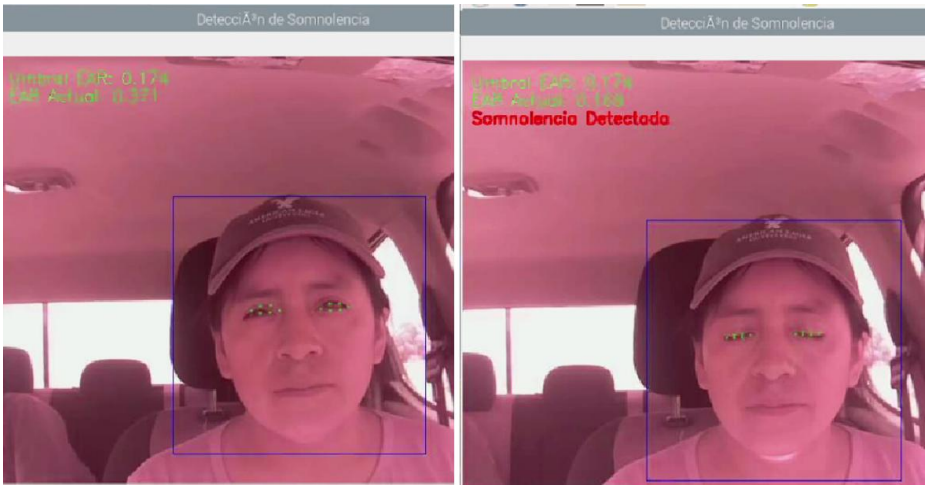
3.3.2. Pruebas del sistema embebido

En la camioneta, el sistema embebido funciono correctamente con el modelo ResNeXt-50 para la detección de somnolencia mencionado anteriormente. En la Figura 65, observamos el funcionamiento. El umbral para la conductora se establece en 0.174. los ojos en estado normal tienen un EAR de 0.371, mientras que los ojos en estado somnoliento tienen un EAR de 0.168.

Esta prueba se lo realizo durante el día, y se observa que el modelo detecta correctamente los puntos de los ojos a pesar de que existe mucha luz de sol de fondo.

Figura 65

Resultados obtenidos en las pruebas de día en la camioneta.

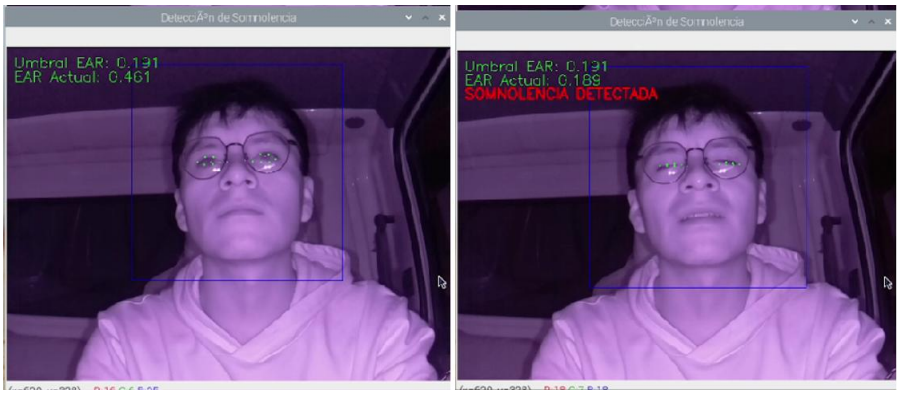


Fuente: Elaboración propia

La prueba en el camion se lo realizo durante la noche como se observa en la Figura 66. Al contar con una camara de Infrarrojo Cercano (NIR) obtenemos imágenes bastante claras, lo que permite al modelo predecir mejor los puntos de los ojos. Al condutor se le establece un umbral EAR de 0.191. Con los ojos en estado normal el EAR es de 0.461 mientras que en estado somnoliento es de 0.189.

Figura 66

Resultados obtenidos en las pruebas de noche en el camión.



Fuente: Elaboración propia

Se evaluaron estas pruebas utilizando la matriz de confusión y obtener la exactitud que obtiene el modelo para detectar somnolencia mediante la predicción de los 12 puntos de los ojos, así también se debe tomar en cuenta que para el cálculo del umbral EAR se utilizó el factor de multiplicación 0.7 descrito en la sección 2.4.3, “Integración del modelo ResNeXt-50 en la placa base”. El formato de la matriz de confusión utilizada se presenta en la Tabla 24, donde somnolencia se representa por ‘Drowsiness’ y no somnolencia por ‘Awake’.

Tabla 24

Formato de la matriz de confusión para cada persona.

Persona (n)		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	Verdaderos Negativos (TN)	Falsos Positivos (FP)
Reales	Drowsiness	Falsos Negativos (FN)	Verdaderos Positivos (TP)

Fuente: Elaboración propia

Para estas evaluaciones se utilizaron 10 conjuntos de datos, donde cada uno contiene 100 imágenes pertenecientes a una sola persona, además, el conjunto se dividió en una proporción de 50:50 de ojos abiertos y cerrados respectivamente. Cada clase contiene imágenes en el día con rostros sin accesorios fáciles, con lentes y con gafas, y en la noche con rostros sin accesorios fáciles y lentes. Los resultados obtenidos se presentan en las Tablas 25 a 34, donde se detallan los valores TP, FP, TN y FN, que obtuvo el modelo con cada persona.

Tabla 25

Matriz de confusión persona 1.

Persona 1		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	41	9
Reales	Drowsiness	7	43

Fuente: Elaboración propia

Tabla 26

Matriz de confusión persona 2.

Persona 2		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	37	13
Reales	Drowsiness	13	37

Fuente: Elaboración propia

Tabla 27

Matriz de confusión persona 3.

Persona 3		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	40	10
Reales	Drowsiness	11	39

Fuente: Elaboración propia

Tabla 28

Matriz de confusión persona 4.

Persona 4		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	39	11
Reales	Drowsiness	9	41

Fuente: Elaboración propia

Tabla 29

Matriz de confusión persona 5.

Persona 5		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	42	8
Reales	Drowsiness	9	41

Fuente: Elaboración propia

Tabla 30

Matriz de confusión persona 6.

Persona 6		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	40	10
Reales	Drowsiness	8	42

Fuente: Elaboración propia

Tabla 31

Matriz de confusión persona 7.

Persona 7		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	40	10
Reales	Drowsiness	13	37

Fuente: Elaboración propia

Tabla 32

Matriz de confusión persona 8.

Persona 8		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	44	6
Reales	Drowsiness	6	44

Fuente: Elaboración propia

Tabla 33

Matriz de confusión persona 9.

Persona 9		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	39	11
Reales	Drowsiness	10	40

Fuente: Elaboración propia

Tabla 34

Matriz de confusión persona 10.

Persona 10		**Valores Predichos**	
		Awake	Drowsiness
Valores	Awake	42	8
Reales	Drowsiness	9	41

Fuente: Elaboración propia

Con las matrices de confusión obtenidas, se calculó las métricas de exactitud, precisión, recall y F1 Score, que nos permitirá evaluar las robustes del modelo al detectar somnolencia utilizando los 12 puntos de los ojos que este nos predice mediante la tarea de regresión. En la Tabla 35, se muestran las métricas obtenidas de cada persona en base a los resultados de su respectiva matriz de confusión.

Tabla 35

Métricas obtenidas mediante el enfoque del cálculo EAR (landmarks faciales).

Persona	**TP**	**FN**	**FP**	**TN**	**Total**	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1 Score (%)**
1	43	7	9	41	100	84.00	82.69	86.00	84.31
2	37	13	13	37	100	74.00	74.00	74.00	74.00
3	39	11	10	40	100	79.00	79.59	78.00	78.79
4	41	9	11	39	100	80.00	78.85	82.00	80.39
5	41	9	8	42	100	83.00	83.67	82.00	82.83
6	42	8	10	40	100	82.00	80.77	84.00	82.35
7	37	13	10	40	100	77.00	78.72	74.00	76.29
8	44	6	6	44	100	88.00	88.00	88.00	88.00
9	40	10	11	39	100	79.00	78.43	80.00	79.21
10	41	9	8	42	100	83.00	83.67	82.00	82.83
				Promedio		**80.90**	**80.84**	**81.00**	**80.90**

Fuente: Elaboración propia

Por otra parte, con los mismos conjuntos de datos de cada persona se evaluó el enfoque de clasificación. La Tabla 36, muestra los resultados obtenidos con este enfoque.

Tabla 36

Métricas obtenidas mediante el enfoque de clasificación de imágenes (rostro completo).

Persona	TP	FN	FP	TN	Total	Exactitud (%)	Precisión (%)	Recall (%)	F1 Score (%)
1	48	2	3	47	100	95.00	94.12	96.00	95.05
2	50	0	2	48	100	98.00	96.15	100	98.04
3	40	10	0	50	100	90.00	100	80.00	88.89
4	45	5	0	50	100	95.00	100	90.00	94.74
5	44	6	5	45	100	89.00	89.80	88.00	88.89
6	45	5	5	45	100	90.00	90.00	90.00	90.00
7	50	0	0	50	100	100	100	100	100
8	50	0	0	50	100	100	100	100	100
9	46	4	4	46	100	92.00	92.00	92.00	92.00
10	47	3	2	48	100	95.00	94.12	96.00	95.05
				Promedio		**94.40**	**95.62**	**93.20**	**94.27**

Fuente: Elaboración propia

3.4. Análisis y reporte de resultados

3.4.1. Sistema embebido

El modelo ResNeXt-50 integrado en el sistema embebido Raspberry Pi 3 Modelo B+ para la detección de somnolencia presento el siguiente rendimiento durante la ejecución del script. Mientras el script de detección de somnolencia se encontraba en ejecución, el uso promedio de CPU estaba en 80%. Mientras el modelo ResNeXt-50 está en ejecución, logra un promedio de 5 fotogramas por segundo. Esta tasa de fotogramas baja se debe principalmente a la complejidad que presenta el modelo, que cuenta con más de 27 millones de parámetros. A pesar de su bajo rendimiento en términos de velocidad, el modelo es muy preciso en la detección de puntos de los ojos, importantes para la detección de somnolencia. Además, es importante mencionar que la tasa de fotogramas se ve afectado debido al hardware que utilizamos, ya que, si

se utilizaría un hardware con mejores especificaciones que las que usamos en este trabajo, la tasa de fotogramas tendría un aumento muy notable.

Al evaluar la detección de somnolencia en 10 personas, del modelo integrado en el sistema embebido en base al enfoque del cálculo EAR con 12 puntos de los ojos que el modelo predice mediante la tarea de regresión, se obtienen las métricas que se muestran en la Tabla 35, donde en promedio, se obtiene una exactitud del 80.90%, precisión del 80.84%, recall del 81.00% y f1 score del 80.90%. Esto nos demuestra que el modelo tiene una alta efectividad al momento de detectar somnolencia. Por otra parte, al evaluar nuestro modelo en base al enfoque de clasificación de imágenes mediante la tarea de clasificación del modelo, se obtienen los siguientes resultados: una exactitud del 94.40%, precisión del 95.62%, recall del 93.20% y f1 score del 94.27%. Los resultados son bastante estables, indicándonos un balance entre la clase positiva (drowsiness) y la negativa (awake).

3.4.2. Característica de portabilidad ISO 25023

3.4.2.1. Adaptabilidad

Se evaluó el tiempo que se tarda en realizar la configuración del sistema embebido en un nuevo entorno. Esta métrica es importante y necesaria para tomar en cuenta el tiempo necesario para configurar el sistema embebido, esto implica la configuración del entorno de Python que requiere el script, configuración de la ubicación de la cámara y configuración de inicio del script que contienen la lógica de detección de somnolencia.

Para esto se usó un cronometro como instrumento que ayudo a realizar esta medición de tiempo. En la Tabla 37, se detallan los tiempos obtenidos en cada configuración mencionada.

Tabla 37

Tiempo para la configuración del sistema embebido en nuevos entornos.

Configuración	**Tiempo (min: seg)**	**Observaciones**
Configuración del entorno Python	10:35	Se debe tener un archivo “requirements.txt” que contienen todas las librerías necesarias, así como acceso a internet.
Configuración de	2:28	Para ajustar la correcta ubicación de la cámara, es

la ubicación de la cámara		necesario ocupar un script que ejecute solo la cámara para poder obtener la mejor ubicación.
Configuración de inicio del script	5:10	Ninguna
Tiempo total	18.13	

Fuente: Elaboración propia

Es importante tomar en cuenta que estas configuraciones fueron realizadas por una persona que conoce del tema. Por lo tanto, los tiempos reflejados representan las duraciones estimadas para una persona con conocimientos en esta área.

3.4.2.2. Capacidad de ser Instalado

Se evaluó que tan fácil es instalar el sistema embebido en el vehículo. Se realizo un conteo de pasos que toman completar la instalación del sistema en el vehículo. En la Table 38, se muestran los pasos necesarios para instalación.

Tabla 38

Pasos para la instalación del sistema embebido.

Nro.	Paso	Observaciones
1	Conectar la placa base a la fuente de alimentación 5V.	Es recomendable conectar a la placa base entes de encender el vehículo.
2	Colocar la cámara en un lugar adecuado.	Ubicarlo en un lugar estable, además, no debe obstruir la visión del conductor.
3	Colocar la placa base en un lugar adecuado.	Evitar colocarlo en área cerradas y con humedad. Esto podría provocar sobrecalentamiento y daños en la placa base
4	Verificación de conexiones	Se debe revisar que las conexiones de los periféricos como la cámara y el zumbador se encuentren correctamente conectados para un correcto funcionamiento.

Fuente: Elaboración propia

Los pasos de la instalación son sencillos y pueden ser realizados por cualquier persona con un mínimo de instrucciones técnicas. Es importante tomar en cuenta las observaciones para obtener una instalación correcta.

Limitaciones

El presente trabajo se desarrolló utilizando conjuntos de datos de creación propia y otros 12 conjuntos de datos, todos para tareas de regresión en predicción de 68 puntos de referencia facial. A pesar de que con todos estos conjuntos de datos se obtiene más de 30 mil imágenes, cada uno con su respectivo archivo '*.pts*', se observó que existe un desbalanceo de datos entre lo que se categorizó como imágenes de rostros con ojos abiertos e imágenes de rostros con ojos cerrados, ya que se contabilizo solo 5871 imágenes de ojos cerrados, y el resto de las imágenes presentan ojos abiertos. Esta desproporcionalidad de datos nos limita a utilizar solo 5871 datos de las dos clases para el entrenamiento, con un total de 11742 datos, ya que si al entrenar el modelo con un conjunto de datos desbalanceado, en este caso, amentar el número de imágenes a la clase de ojos abiertos, se pueden obtener mejores métricas, pero el modelo obtenido va a tener mejores resultados si se le presentan datos similares a los que predomino durante el entrenamiento. Es decir, va a clasificar mejor y obtener mejores predicciones de los puntos si son imágenes con ojos abiertos.

Esta limitación reduce la robustes del modelo, así como la capacidad de generalizar los resultados en más entornos, en condiciones de luz variada y en rostros con accesorios faciales. Ya que, a pesar de utilizar el aumento de datos como técnica para aumentar el número de imágenes de rostros con gafas, esta técnica se va a ver limitado si no podemos utilizar más cantidad de datos en ambas clases.

Si embargo, incluso después de presentarse esta limitación, el modelo logro alcanzar resultados positivos tanto en la tarea de clasificación con una alta exactitud, así como en la tarea de regresión con la predicción de los 12 puntos de los ojos.

Discusiones

El presente trabajo se enfoca en la detección de somnolencia utilizando la Red Neurona Convolucional ResNeXt-50 el sistema embebido RaspBerry Pi Modelo 3b+. El modelo fue adaptado para realizar las tareas de clasificación y regresión. En la primera tarea, nos enfocamos en clasificar imágenes que presenten rostros y determinar si el rostro presenta los ojos abiertos o cerrados, donde el modelo nos retorna un valor entre cero y uno respectivamente. Durante el entrenamiento se realizó el seguimiento de las siguientes métricas de clasificación: pérdida, exactitud (accuracy), precisión, sensibilidad (recall), F1 Score. En la tarea de regresión, nos enfocamos en la predicción de 24 valores que representan los 12 puntos (x, y) de los ojos, siendo monitoreado durante el entrenamiento con las siguientes métricas: Error Cuadrático Medio (MSE, tomándolo también como pérdida), Error Absoluto Medio (MAE), y la Raíz del Error Cuadrático Medio (RMSE). Además, se utilizó dos diferentes Dataset denominados Dataset Balanceado y Dataset Desbalanceado, por lo que se obtuvieron dos modelos.

Estos modelos fueron entrenados en dos diferentes entornos, el de Google Colab y el de HPC CEDIA, de las cuales se seleccionaron los modelos entrenados en el segundo entorno debido a que obtuvieron mejores tiempos de entrenamiento. El modelo entrenado con el Dataset Balanceado obtuvo las siguientes métricas de clasificación: perdida combinada (9.02%), exactitud (93.06%), precisión (92.93%), sensibilidad (93.02%) y f1 score (92.97%), y las siguientes métricas de regresión: MSE (0.29%), MAE (3.78%) y RMSE (5.40%). El modelo entrenado con el Dataset Desbalanceado obtuvo las siguientes métricas de clasificación: perdida combinada (7.21%), exactitud (91.74%), precisión (92.60%), sensibilidad (85.39%) y F1Score (88.85%), y las siguientes métricas de regresión: MSE (0.31%), MAE (3.96%) y RMSE (5.57%).

Estos resultados nos muestran que el modelo entrenado con le Dataset balanceado obtiene resultados superiores con respecto al modelo entrenado con el Dataset desbalanceado, sin embargo, además, debe tomar en cuenta que este modelo fue entrenado con más datos de imágenes que presentan ojos abiertos, lo que significa que va a funcionar mejor si se le presenta imágenes con características de ojos abiertos, por el contrario la precisión va a verse reducida si al modelo se le presentan imágenes con características de ojos cerrados, algo que no se espera que suceda ya que nos centramos en detectar somnolencia y esto implica que en esa situación los ojos van a encontrarse cerrados. Esto se muestra en la Tabla 18. Con base en ese análisis se

decidió seleccionar el modelo entrenado con el Dataset balanceado para realizar pruebas y comparaciones con trabajos relacionados.

Se realizaron 5 diferentes pruebas donde evaluamos el rendimiento del modelo en las siguientes condiciones y características que presentan las imágenes: rostros sin accesorios faciales (bareface), con lentes (glasses) y con gafas (sunglasses), condiciones de luz diurna y nocturna. Centrándonos en la métrica de exactitud para la tarea de clasificación, el modelo obtuvo los siguientes resultados: bareface (93.50%), glasses (92.25%), sunglasses (88.68%), durante el día (89.39%), y durante la noche (94.42%). Estos resultados nos muestran mayor exactitud si se presentan imágenes donde el rostro este sin accesorio faciales, así también si se encuentra durante condiciones de luz nocturna. Para las evaluar su rendimiento con la tarea de regresión se optó por la métrica Erro Medio Normalizado (NME) y el Área Bajo la Curva (AUC), obteniendo los siguientes resultados: NME (07.16%) y AUC (0.62).

En la Tabla 39, se muestra la comparación realizada con trabajos relacionados en tareas de clasificación en las siguientes categorías: general (donde se incluyen imágenes de rostros sin accesorios faciales, con lentes y con gafas), imágenes de rostros sin accesorios faciales, imágenes de rostros con lentes, e imágenes de rostros con gafas.

Tabla 39

Comparación con trabajos relacionados en tareas de clasificación.

Referencia	**Modelo**	**Exactitud (%)**
General		
Nuestro modelo	ResNext-50	93.06
(Pandey & Muppalaneni, 2023)	CNN-LSTM	97.50
(Ahmed et al., 2023)	CNN	97.00
(Minhas et al., 2022)	ResNet-50	93.69
(El Zein et al., 2024)	VAS-3D	95.50
Imágenes de rostros sin accesorios faciales		
Nuestro modelo	ResNext-50	93.50
(Jamshidi et al., 2021)	ResNet-18 - VGG16 - LSTM	92.73
(W. Deng & Wu, 2019)	MC-KCF	92.10

(Li & Li, 2024)	PFLD – ViT - LSTM	95.26
Imágenes de rostros con lentes		
Nuestro modelo	ResNext-50	92.25
(Li & Li, 2024)	PFLD – ViT - LSTM	91.61
(W. Deng & Wu, 2019)	MC-KCF	91.55
Imágenes de rostros con gafas		
Nuestro modelo	ResNext-50	88.68
(Li & Li, 2024)	PFLD – ViT - LSTM	91.10
(Bearly & Chitra, 2024)	3DDGAN - LA	86.30
(Jamshidi et al., 2021)	ResNet-18 - VGG16 - LSTM	82.09

Fuente: Elaboración propia

Por otra parte, se realizó la comparación de detección de somnolencia por medio de la tarea de regresión, es decir utilizando puntos de referencias faciales. En nuestro enfoque utilizamos el cálculo EAR para la detección de somnolencia, por lo tanto, se lo comparara con trabajos que tengan el mismo enfoque. La Tabla 40, muestra la comparación realizada, donde nuestro modelo ResNeXt-50 obtiene una exactitud del 80.90%, los trabajos de (M. A. Khan et al., 2023), (Theivadas & Ponnan, 2024), y (Asdyo et al., 2023) obtienen una exactitud del 96.00%, 94.00%, y 85.92% respectivamente. Esta comparación nos muestra que nuestro modelo es menos eficiente al momento de detectar somnolencia mediante el enfoque del cálculo EAR.

Tabla 40

Comparación en base al enfoque del cálculo EAR.

Referencia	**Modelo**	**Exactitud (%)**
Nuestro modelo	ResNeXt-50	80.90
(M. A. Khan et al., 2023)	Detector de puntos de referencias faciales Dlib	96.00
(Theivadas & Ponnan, 2024)	Detector de puntos de referencias faciales Dlib	94.00
(Asdyo et al., 2023)	LightGBM - Dlib	85.92

Fuente: Elaboración propia

Además, se realizaron las siguientes dos comparaciones: una sobre el tiempo de inferencia del modelo y la segunda sobre el número de fotogramas por segundo (FPS). El tiempo de inferencia se refiere al tiempo que el modelo tarda en procesar una entrada y generar una salida después de haber sido entrenado. Por otra parte, los FPS indican la cantidad de imágenes que el modelo puede procesar por segundo. Estos resultados pueden variar dependiendo del hardware utilizado durante las pruebas, donde generalmente, mientras más potente sea el hardware mayor serán los FPS menor el tiempo de inferencia. La Tabla 41, muestra la comparativa realizada, donde nuestro modelo ResNeXt-50 obtiene un tiempo de inferencia de 200 ms/f a un promedio de 5 fps, el modelo CNN-LSTM de (Pandey & Muppalaneni, 2023) obtiene 79.14 ms/f a 12 fps respectivamente, por otra parte, los modelos VAS-3D de (El Zein et al., 2024) y ResNet-18 - VGG16 – LSTM de (Jamshidi et al., 2021) obtiene en tiempo de inferencia y FPS los siguientes resultados, 25 ms/f a 40 fps, y 33.33 ms/f a 30 fps respectivamente. Se observa que nuestro modelo al contar con un hardware de recursos limitados ya que es un sistema embebido, obtiene un menor tiempo de inferencia, mientras tanto, los otros modelos al realizarse en sistemas con hardware más potentes obtienen mejores resultados.

Tabla 41

Comparación de tiempo de inferencia por fotograma y FPS.

Referencia	**Modelo**	**Tiempo de inferencia (milisegundos por fotograma)**	**Fotogramas por segundo**	**Especificaciones Hardware**	**Tipo Hardware**
Nuestro modelo	ResNeXt-50	666.67 ms/f	1.5 fps	Raspberry Pi 3 modelo B+ con CPU a 1.4 GHz y 1 GB RAM	Sistema embebido
		50 ms/f	20 fps	Laptop con CPU RYZEN 7, GPU RTX 3060, 16GB RAM	Laptop
(Pandey & Muppalaneni, 2023)	CNN-LSTM	79.14 ms/f	12 fps	No especificado	No especificado
(El Zein et al., 2024)	VAS-3D	25 ms/f	40 fps	Servidor HPE ProLiant 1.7 GHz, 64 GB RAM, 240 GB SSD	Servidor
(M. A. Khan et al., 2023)	Dlib	370 ms/f	2.70 fps	RaspBerry Pi 4	Sistema embebido
	Dlib	230 ms/f	4.35 fps	Nvidia Jetson Nano	Sistema embebido
(Jamshidi et al., 2021)	ResNet-18 - VGG16 - LSTM	33.33 ms/f	30 fps	Sistema con CPU Core i7 y GPU GTX 1080	Ordenador

Fuente: Elaboración propia

La tabla comparativa incluye los modelos cuyos autores has reportado el tiempo de inferencia o los FPS. Los modelos cuyos autores no incluyeron esta información no están incluidos.

Con las métricas evaluadas y comparadas con diferentes trabajos relacionados, observamos que el enfoque que obtuvo mejores resultados es el de clasificación como se muestra en la Tabla 36, de la sección 3.3.2. '*Pruebas del sistema embebido*' donde se obtiene una exactitud del 94.40%, una precisión del 95.62%, un recall del 93.20% y un F1 score del 94.27%. Por otra parte, el enfoque de regresión de detección de somnolencia mediante el cálculo EAR de los 12 puntos predichos por el modelo, obtiene resultados aproximadamente un 13% más bajos que los obtenidos en el primer enfoque, con una exactitud del 80.90%, una precisión del 80.84%, un recall del 81.00% y un F1 score del 80.90%, como se observan en la Tabla 35, de la misma sección mencionada. Esto nos indica que, aunque el enfoque de regresión es efectivo, no alcanza la misma precisión que el enfoque de clasificación.

Conclusiones

- Realizar una búsqueda exhaustiva sobre artículos relacionados en detección de somnolencia, ayudo a conocer varios parámetros utilizados para la preparación y el entrenamiento de modelos en este problema. Además, se conoció las métricas que se utilizan para evaluar los modelos tantos los en clasificación como los de regresión. Así también, se conoció correctamente el funcionamiento de la arquitectura ResNeXt-50.
- La creación de un buen Dataset es una parte muy importante durante el proceso de entrenamientos de modelos de Inteligencia Artificial, ya que de eso depende si el modelo obtiene buenos o malos resultados. Debido a esto, se creó un Dataset donde las clases estén balanceadas y un Dataset donde la clase de ojos abiertos predominen un 30% más. Además, se realizó aumento de datos aplicando filtros de gafas a diferentes imágenes, lo que permitió el aumento de imágenes de rostros con gafas, de tal manera que se obtengan mejores resultados con este tipo de imágenes. Esto nos ayudó a comprender la importancia de utilizar un balanceo de datos en las diferentes clases que pudiéramos tener, así como la importancia del aumento de datos.
- TensorFlow y Keras son herramientas muy utilizadas en el área de la Inteligencia Artificial, proporcionándonos modelos preentrenados y listos para ser utilizados en tareas que necesitemos tanto de clasificación como de regresión. Además, en caso de querer crear nuestro propio modelo o crear modelos con arquitecturas que aún no se encuentran en sus repositorios, estas herramientas nos dan esa posibilidad, de crear modelos acordes a nuestras necesidades. Por lo tanto, debido a que nuestro modelo aún no se encuentra en el repositorio de TensorFlow, se adaptó el modelo ResNeXt-50 perteneciente aun repositorio de GitHub, el cual no se encuentra preentrenados. La tarea que a la que se enfocaba era de clasificación de 1000 clases, lo cual fue cambiada para las tareas de clasificación y regresión acordes a nuestra necesidad.
- Los modelos se entrenaron en dos entonos diferentes, el entorno de Google Colab y el de HPC-CEDIA. En el entorno de Google Colab se obtuvieron los siguientes resultados: el modelo balanceado obtuvo una exactitud del 91.15%, con un tiempo de entrenamiento de 3 horas y 50 minutos. El modelo desbalanceado obtuvo una exactitud del 90.47%, con un tiempo de entrenamiento de 4 horas y 37 minutos. En el entorno de HPC-CEDIA se obtuvieron los siguientes resultados: el modelo balanceado obtuvo una exactitud del

93.06%, con un tiempo de entrenamiento de 1 hora y 37 minutos. El modelo desbalanceado obtuvo una exactitud del 91.74%, con un tiempo de entrenamiento de 2 horas y 40 minutos. Estos resultados nos indican que se obtienen mejores métricas y mejores tiempos de entrenamientos en los dos modelos al utilizar el entorno de HPC-CEDIA.

- Los modelos se evaluaron con 5 diferentes conjuntos de datos para conocer la efectividad de detección de somnolencia a través de la tarea de clasificación del modelo ResNeXt-50: rostros sin accesorios faciales, con lentes, con gafas, durante el día, y durante la noche. Esto nos ayudó a seleccionar el mejor modelo para ser comparado con trabajos relacionados. Con el modelo balanceado se obtuvo una exactitud del 92.15 % y para el modelo desbalanceado del 90.84%, donde se observa una diferencia mínima del 1.31%, sin embargo, en la métrica recall que mide la capacidad del modelo en identificar los casos positivos (somnolencia), el modelo balanceado obtiene un 93.06% y el modelo desbalanceado un 84.56%, con una diferencia de casi el 10% lo que nos indica que el modelo desbalanceado tiene más dificultad en identificar somnolencia.
- El modelo ResNeXt-50, integrado en el sistema embebido Raspberry Pi, mediante el enfoque de somnolencia a través de la predicción de los 12 puntos de los ojos utilizando la tarea de regresión y detectando el EAR para determinar el estado de los ojos, alcanzo una exactitud del 80.90%, una precisión del 80.84%, un recall del 81.00% y un f1 score del 80.90%. Por otra parte, el enfoque de clasificación que determina el estado de los ojos al enviarle al modelo la imagen del rostro completo obtuvo una exactitud del 94.40%, una precisión del 95.62%, un recall del 93.20% y un f1 score del 94.27%. Estos resultados nos indica que el segundo enfoque es más efectivo al momento de realizar la detección de somnolencia.

Recomendaciones

- Debido a la limitación presentada sobre la distribución de los datos, se recomienda aumentar el número de datos pertenecientes a la categoría de ojos cerrados para poder entrenar los modelos con un mayor número de datos de forma balanceada.
- El sistema embebido muestra una lata efectividad en la detección de somnolencia, sin embargo, este se encuentra limitado por la taza de fotogramas debido a la capacidad del hardware. Se recomienda explorar alternativas de hardware más actuales y enfocados en el campo de Inteligencia Artificial con la finalidad de aumentar la eficiencia en términos de fotogramas para el modelo ResNeXt-50 y para otros modelos que requieran mayor capacidad computacional.
- Se recomienda experimentar con otras arquitecturas actuales como YOLO 11 y Transformer RT-DETR.

Referencias

Ahmed, M. I. B., Alabdulkarem, H., Alomair, F., Aldossary, D., Alahmari, M., Alhumaidan, M., Alrassan, S., Rahman, A., Youldash, M., & Zaman, G. (2023). A Deep-Learning Approach to Driver Drowsiness Detection. *Safety*, *9*(3), 65. https://doi.org/10.3390/safety9030065

Akrout, B., & Mahdi, W. (2023). A novel approach for driver fatigue detection based on visual characteristics analysis. *Journal of Ambient Intelligence and Humanized Computing*, *14*(1), 527–552. https://doi.org/10.1007/s12652-021-03311-9

Alba Neppas, J. M. (2020). *Desarrollo de un sistema embebido mediante el uso de técnicas de visión artificial para detección y alerta de somnolencia en conducción diurna en tiempo real* [Tesis de pregrado, Universidad Técnica del Norte]. http://repositorio.utn.edu.ec/handle/123456789/10171

Aloseel, A., He, H., Shaw, C., & Khan, M. A. (2021). Analytical Review of Cybersecurity for Embedded Systems. In *IEEE Access* (Vol. 9). https://doi.org/10.1109/ACCESS.2020.3045972

Al-Selwi, H. F., Hassan, N., Ghani, H. B. A., Hamzah, N. A. B. A., & Aziz, A. B. A. (2023). Face mask detection and counting using you only look once algorithm with Jetson Nano and NVDIA giga texel shader extreme. *IAES International Journal of Artificial Intelligence*, *12*(3). https://doi.org/10.11591/ijai.v12.i3.pp1169-1177

ANT. (2023). *Histórico estadísticas siniestros de tránsito*. Agencia Nacional de Tránsito Del Ecuador. https://www.ant.gob.ec/historico-estadisticas-siniestros-de-transito/

Asdyo, B., Kanigoro, B., & Rojali. (2023). Drowsy Detection System by Facial Landmark and Light Gradient Boosting Machine Method. *Procedia Computer Science*, *227*, 500–507. https://doi.org/https://doi.org/10.1016/j.procs.2023.10.551

Balseca, E. (2014). *Evaluación de Calidad de Productos de Software en Empresas de Desasrrollo de Software Aplicando la Norma ISO/IEC 25000* [Trabajo de grado, Escuela Politécnica Nacional]. https://bibdigital.epn.edu.ec/bitstream/15000/9113/1/CD-6067.pdf

Barbanoj, M. J., Antonijoan, R. M., García-Gea, C., Clos, S., Grasa, E., & Giménez, S. (2007). Fármacos que pueden producir somnolencia excesiva. *Vigilia-Sueño*, *18*, 26–31. https://www.elsevier.es/es-revista-vigilia-sueno-270-articulo-farmacos-que-pueden-producir-somnolencia-13109171

Bearly, E. M., & Chitra, R. (2024). Automatic drowsiness detection for preventing road accidents via 3dgan and three-level attention. *Multimedia Tools and Applications*, *83*(16), 48261–48274. https://doi.org/10.1007/s11042-023-17272-y

Bhatt, D., Patel, C., Talsania, H., Patel, J., Vaghela, R., Pandya, S., Modi, K., & Ghayvat, H. (2021). Cnn variants for computer vision: History, architecture, application, challenges and future scope. In *Electronics (Switzerland)* (Vol. 10, Issue 20). https://doi.org/10.3390/electronics10202470

CEDIA. (2024). *cedia*. https://cedia.edu.ec/beneficio/supercomputador/

Cheng, Y., Lu, M., Gai, X., Guan, R., Zhou, S., & Xue, J. (2024). Research on multi-signal milling tool wear prediction method based on GAF-ResNext. *Robotics and Computer-Integrated Manufacturing*, *85*, 102634. https://doi.org/https://doi.org/10.1016/j.rcim.2023.102634

Damashek, J. (2017, March 21). *Las Consecuencias de Conducir Somnoliento.* Hecht, Kleeger & Damashek, P.C. https://lawyer1.com/espanol/blog/las-consecuencias-de-conducir-somnoliento/#:~:text=Un%20conductor%20somnoliento%20es%20menos,que%20trabajan%20durante%20la%20noche.

Dang, Z., Li, J., & Liu, L. (2024). *Cascaded Dual Vision Transformer for Accurate Facial Landmark Detection*. https://api.semanticscholar.org/CorpusID:273962779

Deng, C., Li, D., Feng, M., Han, D., & Huang, Q. (2023). The value of deep neural networks in the pathological classification of thyroid tumors. *Diagnostic Pathology*, *18*(1). https://doi.org/10.1186/s13000-023-01380-2

Deng, W., & Wu, R. (2019). Real-Time Driver-Drowsiness Detection System Using Facial Features. *IEEE Access*, *7*, 118727–118738. https://doi.org/10.1109/ACCESS.2019.2936663

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR*, *abs/2010.11929*. https://arxiv.org/abs/2010.11929

Dua, M., Shakshi, Singla, R., Raj, S., & Jangra, A. (2021). Deep CNN models-based ensemble approach to driver drowsiness detection. *Neural Computing and Applications*, *33*(8), 3155–3168. https://doi.org/10.1007/s00521-020-05209-7

El Telégrafo. (2022, March 17). *Medidas para combatir el sueño en la conducción*. El Telégrafo. https://www.eltelegrafo.com.ec/noticias/empresariales/172/medidas-para-combatir-el-sueno-en-la-conduccion

El Zein, H., Harb, H., Delmotte, F., Zahwe, O., & Haddad, S. (2024). VAS-3D: A Visual-Based Alerting System for Detecting Drowsy Drivers in Intelligent Transportation Systems. *World Electric Vehicle Journal*, *15*(12). https://doi.org/10.3390/wevj15120540

Escobar Cordoba, F. (2020). *Somnolencia diurna excesiva e insomnio: males de los tiempos actuales* (Vol. 0). Editorial Universidad Nacional de Colombia. https://elibro.net/es/lc/utnorte/titulos/189690

Fayyad, U. (1996). Advances in Knowledge Discovery and Data Mining. *MIT Press*.

Gervilla García, E., Jiménez López, R., Montaño Moreno, J. J., Sesé Abad, A., Cajal Blasco, B., & Palmer Pol, A. (2009). La metodología del Data Mining. Una aplicación al consumo de alcohol en adolescentes. *Adicciones*, *21*(1), 65–80. https://www.redalyc.org/articulo.oa?id=289122882009

Ghoddoosian, R., Galib, M., & Athitsos, V. (2019). A Realistic Dataset and Baseline Temporal Model for Early Drowsiness Detection. *CoRR*, *abs/1904.07312*. http://arxiv.org/abs/1904.07312

Google. (2024). *Google Colab*. https://colab.google/

Goswami, U., Rani, J., Kumar, D., Kodamana, H., & Ramteke, M. (2023). Energy Out-of-distribution Based Fault Detection of Multivariate Time-series Data. *Computer Aided Chemical Engineering*, *52*, 1885–1890. https://doi.org/10.1016/B978-0-443-15274-0.50299-7

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *CoRR*, *abs/1512.03385*. http://arxiv.org/abs/1512.03385

Hira, S., Bai, A., & Hira, S. (2021). An automatic approach based on CNN architecture to detect Covid-19 disease from chest X-ray images. *Applied Intelligence*, *51*(5), 2864–2889. https://doi.org/10.1007/s10489-020-02010-w

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

ISO. (2022). *ISO/CEI 25023:2016*. ISO. https://www.iso.org/standard/35747.html#:~:text=Abstract.%20ISO/IEC%2025023:2016%20defines%20quality%20measures%20for,product%20quality%20in%20terms%20of%20characteristics%20and

ISO 25000. (2022). *The ISO/IEC 25000 series of standards*. ISO 25000. https://iso25000.com/index.php/en/iso-25000-standards

Jamshidi, S., Azmi, R., Sharghi, M., & Soryani, M. (2021). Hierarchical deep neural networks to detect driver drowsiness. *Multimedia Tools and Applications*, *80*(10), 16045–16058. https://doi.org/10.1007/s11042-021-10542-7

Jolles, J. W. (2021). Broad-scale applications of the Raspberry Pi: A review and guide for biologists. In *Methods in Ecology and Evolution* (Vol. 12, Issue 9). https://doi.org/10.1111/2041-210X.13652

Kar, P., Chudasama, V. M., Onoe, N., Wasnik, P., & Balasubramanian, V. N. (2024). Fiducial Focus Augmentation for Facial Landmark Detection. *ArXiv*, *abs/2402.15044*. https://api.semanticscholar.org/CorpusID:267897568

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, *53*(8), 5455–5516. https://doi.org/10.1007/s10462-020-09825-6

Khan, M. A., Nawaz, T., Khan, U. S., Hamza, A., & Rashid, N. (2023). IoT-Based Non-Intrusive Automated Driver Drowsiness Monitoring Framework for Logistics and Public Transport Applications to Enhance Road Safety. *IEEE Access*, *11*, 14385–14397. https://doi.org/10.1109/ACCESS.2023.3244008

Khan, M. N., Das, S., & Liu, J. (2024). Predicting pedestrian-involved crash severity using inception-v3 deep learning model. *Accident Analysis & Prevention*, *197*, 107457. https://doi.org/https://doi.org/10.1016/j.aap.2024.107457

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lee, J., Woo, S., & Moon, C. (2024). 3D-CNN Method for Drowsy Driving Detection Based on Driving Pattern Recognition. *Electronics*, *13*(17). https://doi.org/10.3390/electronics13173388

Li, T., & Li, C. (2024). A Deep Learning Model Based On Multi-granularity Facial Features And LSTM Network For Driver Drowsiness Detection. *Journal of Applied Science and Engineering (Taiwan)*, *27*(7), 2799–2811. https://doi.org/10.6180/jase.202407_27(7).0011

Lozada, J. (2014). *Investigación Aplicada*. 47–50. https://dialnet.unirioja.es/servlet/articulo?codigo=6163749

Minhas, A. A., Jabbar, S., Farhan, M., & Najam ul Islam, M. (2022). A smart analysis of driver fatigue and drowsiness detection using convolutional neural networks. *Multimedia Tools and Applications*, *81*(19), 26969–26986. https://doi.org/10.1007/s11042-022-13193-4

Naciones Unidas. (2023). *Objetivos de Desarrollo Sostenible*. https://www.un.org/sustainabledevelopment/es/infrastructure/

National Tsuing Hua University. (2016). *The ACCV Workshop on Driver Drowsiness Detection from Video 2016*. National Tsing Hua University. https://cv.cs.nthu.edu.tw/php/callforpaper/2016_ACCVworkshop/#tab4

NHTSA. (1998). *DROWSY DRIVING AND AUTOMOBILE CRASHES* (Vol. 2). https://www.nhtsa.gov/document/drowsy-driving-and-automobile-crashes

NVIDIA Corporation. (2024). *Jetson Nano*. NVIDIA. https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/product-development/

Pandey, N. N., & Muppalaneni, N. B. (2023). Dumodds: Dual modeling approach for drowsiness detection based on spatial and spatio-temporal features. *Engineering Applications of Artificial Intelligence*, *119*, 105759. https://doi.org/https://doi.org/10.1016/j.engappai.2022.105759

Perrotte, G., Bougard, C., Portron, A., & Vercher, J.-L. (2024). Monitoring driver drowsiness in partially automated vehicles: Added value from combining postural and physiological indicators. *Transportation Research Part F: Traffic Psychology and Behaviour*, *100*, 458–474. https://doi.org/10.1016/J.TRF.2023.12.010

Perumandla Dheeraj. (2020). *Drowsiness_dataset*. Kaggle. https://www.kaggle.com/datasets/dheerajperumandla/drowsiness-dataset

Phan, A.-C., Trieu, T.-N., & Phan, T.-C. (2023). Driver drowsiness detection and smart alerting using deep learning and IoT. *Internet of Things*, *22*, 100705. https://doi.org/10.1016/j.iot.2023.100705

Prados-Torreblanca, A., Buenaposada, J. M., & Baumela, L. (2022). Shape Preserving Facial Landmarks with Graph Attention Networks. *ArXiv*, *abs/2210.07233*. https://api.semanticscholar.org/CorpusID:252873341

Ramzan, M., Khan, H. U., Awan, S. M., Ismail, A., Ilyas, M., & Mahmood, A. (2019). A Survey on State-of-the-Art Drowsiness Detection Techniques. In *IEEE Access* (Vol. 7). https://doi.org/10.1109/ACCESS.2019.2914373

Raspberry Pi Foundation. (2024). *Raspberry Pi 4*. Raspberry Pi Foundation. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

Roa, P., Morales, C., & Gutiérrez, P. (2015). Norma ISO/IEC 25000. *Tecnología Investigación y Academia*, *3*(2). https://revistas.udistrital.edu.co/index.php/tia/article/view/8373

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

Saleh, A. W., Gupta, G., Khan, S. B., Alkhaldi, N. A., & Verma, A. (2023). An Alzheimer's disease classification model using transfer learning Densenet with embedded healthcare decision support system. *Decision Analytics Journal*, *9*, 100348. https://doi.org/https://doi.org/10.1016/j.dajour.2023.100348

Sarker, S., Tushar, S. N. B., & Chen, H. (2023). High accuracy keyway angle identification using VGG16-based learning method. *Journal of Manufacturing Processes*, *98*, 223–233. https://doi.org/10.1016/J.JMAPRO.2023.04.019

Secretaría Nacional de Planificación. (2021). *Plan de Creación de Oportunidades 2021-2025*. https://www.planificacion.gob.ec/wp-content/uploads/2021/09/Plan-de-Creacio%CC%81n-de-Oportunidades-2021-2025-Aprobado.pdf

Shi, S.-Y., Tang, W.-Z., & Wang, Y.-Y. (2017). A Review on Fatigue Driving Detection. *ITM Web Conf.*, *12*. https://doi.org/10.1051/itmconf/20171201019

Simonyan Karen, & Zisserman Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv Preprint ArXiv:1409.1556*. https://doi.org/https://doi.org/10.48550/arXiv.1409.1556

Szegedy Christian, Vanhoucke Vincent, Ioffe Sergey, Shlens Jon, & Wojna Zbigniew. (2016). Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818–2826). https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.html

Thapa, S., & Sarkar, A. (2024). A deep dive into enhancing sharing of naturalistic driving data through face deidentification. *The Visual Computer*. https://doi.org/10.1007/s00371-024-03552-7

Theivadas, J. R., & Ponnan, S. (2024). VigilEye: Machine learning-powered driver fatigue recognition for safer roads. *Measurement: Sensors*, *33*, 101186. https://doi.org/https://doi.org/10.1016/j.measen.2024.101186

Wan, R., Shi, B., Li, H., Duan, L.-Y., & Kot, A. C. (2021). Face Image Reflection Removal. *International Journal of Computer Vision*, *129*(2), 385–399. https://doi.org/10.1007/s11263-020-01372-5

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2016). *Aggregated Residual Transformations for Deep Neural Networks*. http://arxiv.org/abs/1611.05431

Xu, Z., Li, B., Geng, M., & Yuan, Y. (2020). AnchorFace: An Anchor-based Facial Landmark Detector Across Large Poses. *ArXiv E-Prints*, arXiv:2007.03221. https://doi.org/10.48550/arXiv.2007.03221

Yu, L., Yang, X., Wei, H., Liu, J., & Li, B. (2024). Driver fatigue detection using PPG signal, facial features, head postures with an LSTM model. *Heliyon*, *10*(21), e39479. https://doi.org/10.1016/j.heliyon.2024.e39479

Anexos

Anexo 1: Manual HPC-CEDIA

Inicio de Sesión

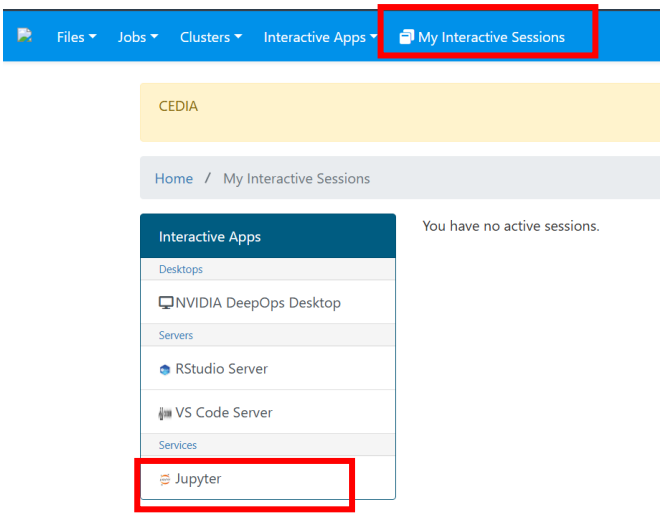
1. Ingresar a la página web de HPC-CEDIA que se encuentra en el siguiente enlace: https://hpc.cedia.edu.ec
2. Iniciar sesión con sus respectivas credenciales.

Creación del entorno Jupyter

1. Una vez iniciado sesión, nos dirigimos a la opción “My Interactive Sessions” y no llevara al menú que se muestra en la Figura 67, donde podemos acceder a diferentes aplicaciones interactivas que ofrece HPC-CEDIA que son Jupyter Notebook, RStudio Server, VS Code Server y NVIDIA DeepOps Desktop. En nuestro caso crearemos una sesión con Jupyter, así que seleccionamos esa opción.

Figura 67

Menú de aplicaciones interactivas ofrecidas por HPC-CEDIA.

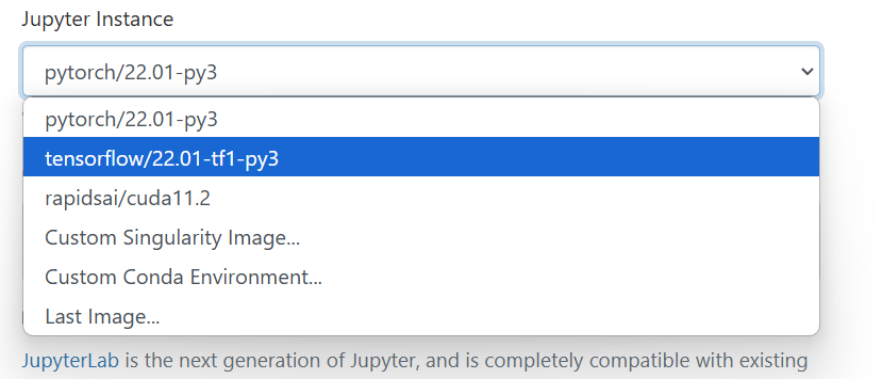


2. Deberemos personalizar los parámetros de nuestra sesión dependiendo de las necesidades requeridas como se muestra a continuación:
   - Elegimos la instancia de Jupyter a crear según lo que necesitemos, tenemos varias instancias como se muestra en la Figura 68. En nuestro caso seleccionamos la

instancia de TensorFlow, que integra Python 3 y TensorFlow 1 que puede ser actualizado a versiones más recientes.

Figura 68

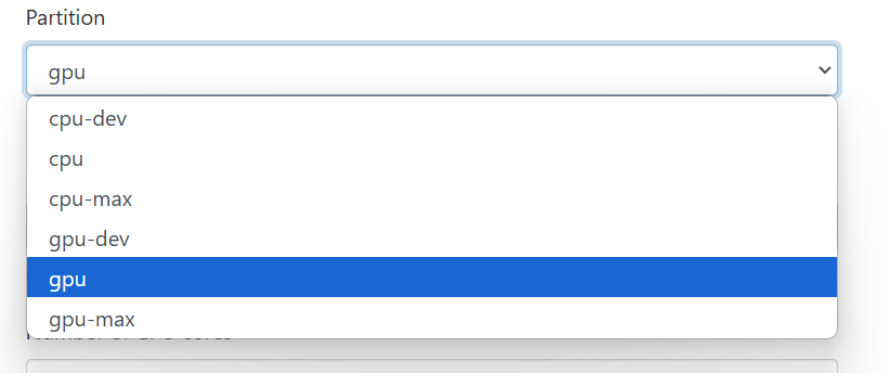
Selección de la instancia de Jupyter.



- A continuación, elegimos la partición que se necesite como se muestra en la Figura 69.

Figura 69

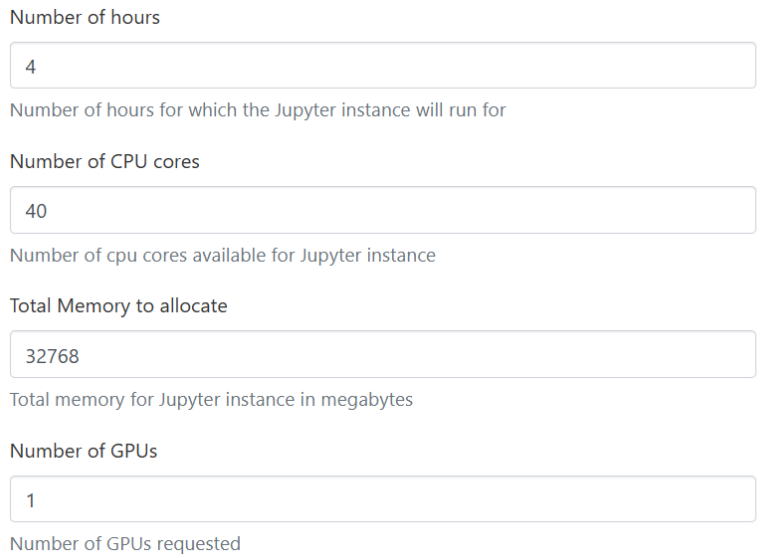
Configuración del tipo de partición.



- Estableceos el número de horas que se reservará los recursos, el número de núcleos de CPU, la cantidad de memoria en megabytes y el número de GPUs como se muestra en la Figura 70.

Figura 70

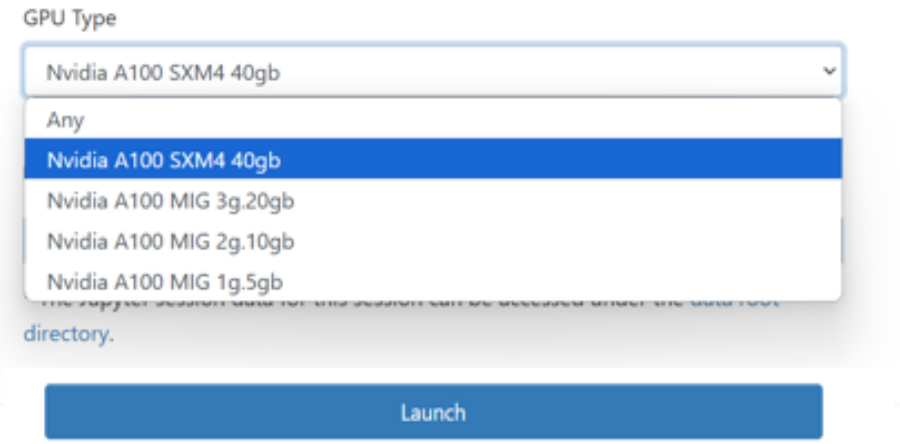
Configuración de recursos del entorno.



- Por último, seleccionamos el tipo de GPU que necesitemos y creamos el entorno. Se muestra en la Figura 71.

Figura 71

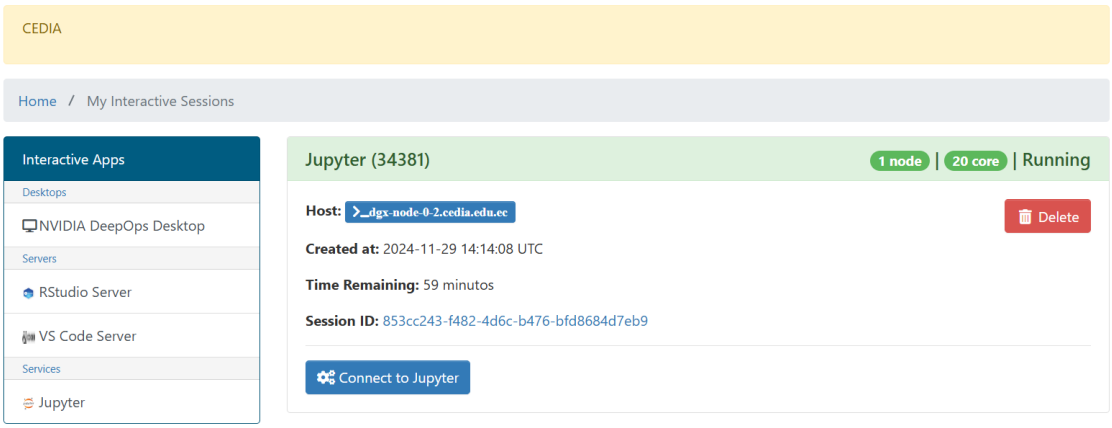
Selección de GPU y creación del entorno.



3. Al terminar la creación serán dirigidos a una nueva vista donde podrán visualizar las sesiones activas que tienen, así como el tiempo restante de cada sesión como se muestra en la Figura 72.

Figura 72

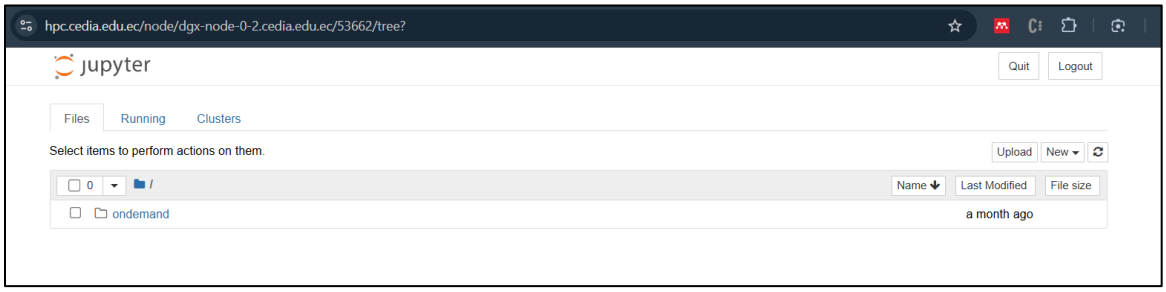
Lista de entornos creados



4. Para acceder a nuestro entorno seleccionamos la opción “Connect to Jupyter” y se nos abrirá una nueva ventana con el entorno listo para su uso como se muestra en la Figura 73.

Figura 73

Entorno Jupyter.



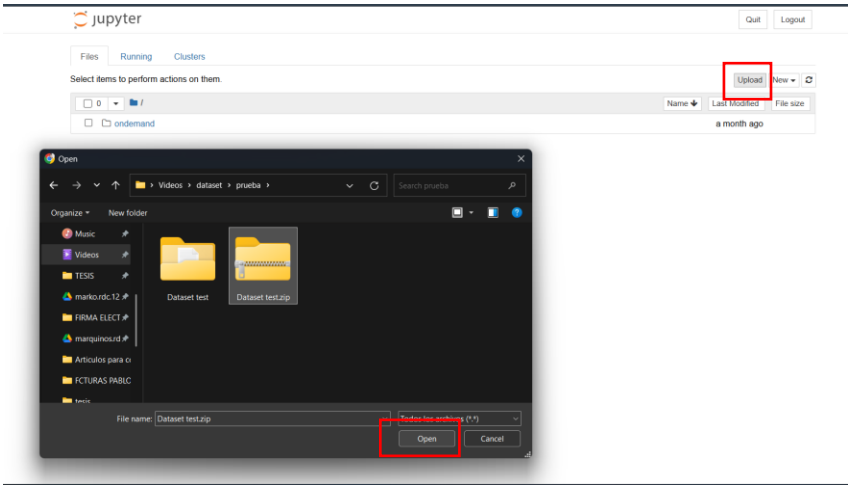
Subida de archivos al entorno Jupyter

Al entorno se pueden subir deferentes tipos de archivos, sin embargo, no permite subir carpetas. Para poder subir carpetas que contengan datos u otros archivos, se recomienda seguir los siguientes pasos.

1. Comprimir la carpeta en formato ‘*.zip*’.
2. En nuestro entorno Jupyter, seleccionar la opción “Upload” y dirigirnos a la ubicación que se encuentra nuestra carpeta comprimida y seleccionarla como se muestra en la Figura 74.

Figura 74

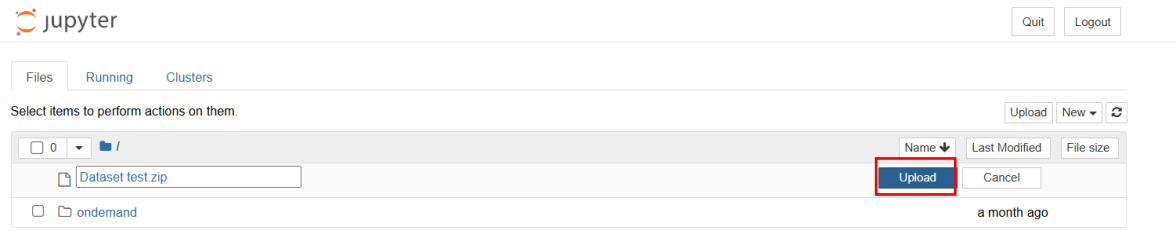
Selección de archivos para el entorno Jupyter.



3. A continuación, aparecerá la carpeta comprimida en el entorno y seleccionamos la opción "Upload" para que comience la carga como muestra la Figura 75.

Figura 75

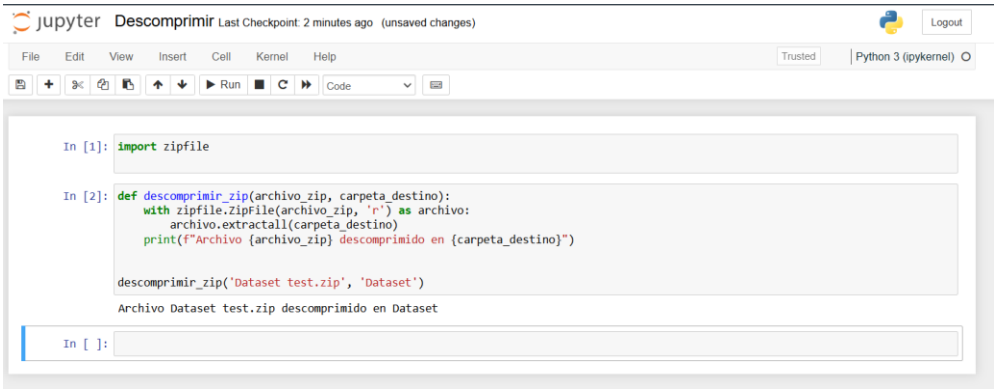
Subida de archivos al entrono Jupyter.



4. Finalmente tendremos disponible la carpeta comprimida en nuestro entorno. Para poder acceder a los datos, en el mismo entorno utilizaremos un código en Python que nos permite descomprimir archivos .zip, que se muestra en la Figura 76. Donde en la función "*descomprimir_zip()*" enviamos como parámetros la ubicación donde se encuentra el archivo o carpeta comprimida, y la ubicación en donde queremos que se descompriman.

Figura 76

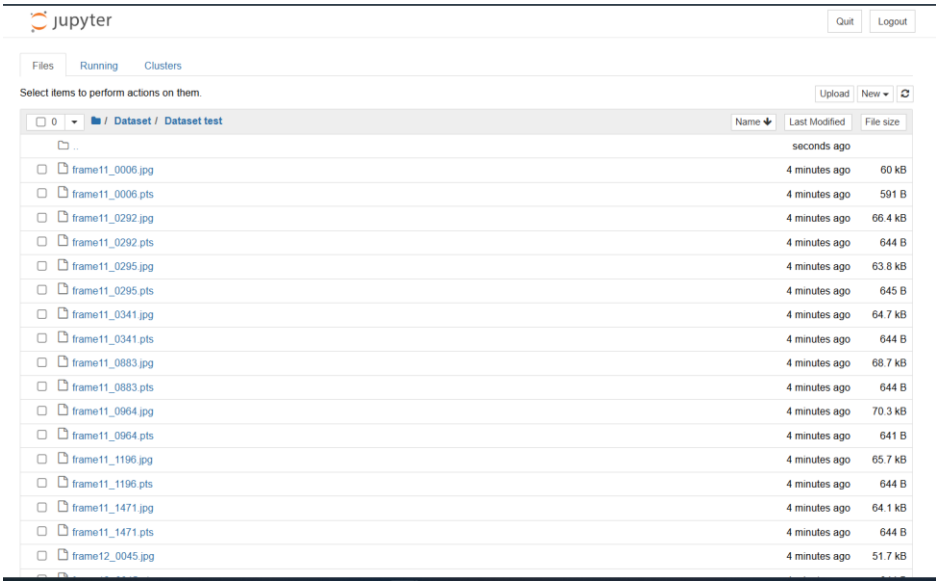
Método para descomprimir archivos en formato '.zip'.



5. De esta manera podremos acceder a grandes conjuntos de datos en caso de que se necesite, la Figura 77, muestra el ejemplo.

Figura 77

Datos en el entorno Jupyter.

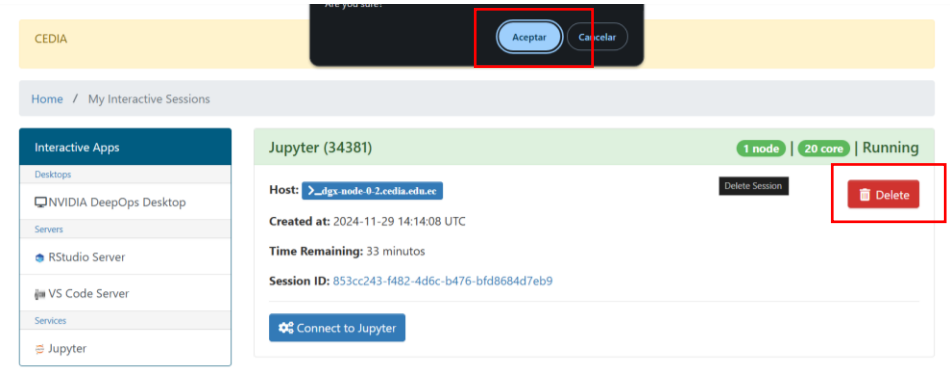


Eliminación de recursos

Cuando ya no se desee ocupar el entorno es necesario liberar los recursos reservados. Para eso vamos a la pantalla donde se muestran nuestras sesiones y seleccionamos la opción "Delete" y aceptamos como se muestra en la Figura 78.

Figura 78

Eliminación del entorno Jupyter.



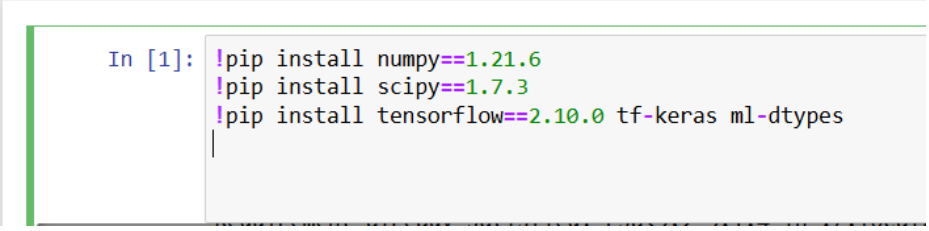
Instalación de librerías

Para instalar librerías en el entorno Jupyter utilizamos la herramienta pip, un gestor de paquetes que permite instalar y bibliotecas y dependencias en Python.

1. Creamos y abrimos un cuaderno de Jupyter en nuestro entorno.
2. Escribimos el comando de instalación. Utilizamos el prefijo '*!*', esto indica que se ejecutara un comando del sistema desde el cuaderno Jupyter, de la siguiente manera '*!pip install* ***libreria***'. En caso de querer instalar una versión especifica de librería, se debe especificarla después del nombre de la siguiente manera '*!pip install* ***librería*****==x.x.x**'. La Figura 79, muestra un ejemplo de instalación de librerías y sus respectivas versiones necesarias para el entrenamiento, donde se instalan las siguientes librerías: numpy versión 1.21.6, scipy versión 1.7.3, y TensorFlow y Keras versión 2.10.0.

Figura 79

Instalación de librerías en el entorno HPC-CEDIA.



Anexo 2: Entrenamiento del modelo ResNeXt-50

https://github.com/mvinlagoc/Tesis-Resnext50.git

Anexo 3: Cálculo de métricas NME, curva CED, AUC, Failure Rate

https://github.com/mvinlagoc/Tesis-Resnext50.git

Anexo 4: Código Python del sistema embebido

https://github.com/mvinlagoc/Tesis-Resnext50.git

Anexo 5: Datasets balanceado y desbalanceado

https://drive.google.com/drive/folders/16KqNw8C7f5_NeU3iU8PCverYTDXIvi2j?usp=drive_link