UNIVERSIDAD TÉCNICA DEL NORTE

Faculta de Ingeniería en Ciencias Aplicadas

Carrera de Software

IMPLEMENTACIÓN DE UNA RED NEURONAL TRANSFORMER PARA DETECCIÓN DE SOMNOLENCIA EN CONDUCTORES DE VEHÍCULOS DURANTE LA CONDUCCIÓN DIURNA Y NOCTURNA.

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

a la obtención del título de Ingeniero de Software.

Autor:

Henry Patricio Meza Morales

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:	1004192371		
APELLIDOS Y NOMBRES:	Meza Morales Henry Patricio		
DIRECCIÓN:	Otavalo – Vía Selva Alegre – Quinde Km 18		
EMAIL:	hpmezam@utn.edu.ec / mezahenry38@gmail.com		
TELÉFONO FIJO:		**TELÉFONO MÓVIL:**	0994867536

DATOS DE LA OBRA	
TÍTULO:	Implementación de una red neuronal Transformer para detección de somnolencia en conductores de vehículos durante la conducción diurna y nocturna.
AUTOR (ES):	Meza Morales Henry Patricio
FECHA DE APROBACIÓN: DD/MM/AAAA	28/02/2025
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniero en Software
DIRECTOR:	PhD. Iván Danilo García Santillán
ASESOR:	MsC. Cosme MacArthur Ortega Bustamante

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 28 días del mes de febrero de 2025

EL AUTOR:

(Firma).....
Nombre: Meza Morales Henry Patricio
CI: 1004192371

CERTIFICACIÓN DIRECTOR

Ibarra 28 de febrero del 2025

CERTIFICACIÓN DIRECTOR DEL TRABAJO DE TITULACIÓN

Por medio del presente yo PhD. Iván Danilo García Santillán, certifico que el Sr. Meza Morales Henry Patricio portador de la cédula de ciudadanía número 100419237-1, ha trabajado en el desarrollo del proyecto de grado "**Implementación de una red neuronal Transformer para detección de somnolencia en conductores de vehículos durante la conducción diurna y nocturna",** previo a la obtención del Título de Ingeniero en Software. Este trabajo se ha realizado con responsabilidad, lo cual certifico con honor a la verdad.

Atentamente

IVAN DANILO GARCIA SANTILLAN

Digitally signed by IVAN DANILO GARCIA SANTILLAN
Date: 2025.02.28 14:05:21 -05'00'

PhD. Iván Danilo García Santillán

DIRECTOR DE TRABAJO DE GRADO

Dedicatoria

Con todo mi amor y emoción, dedico esta tesis:

A Dios, por darme la oportunidad de vivir esta experiencia, por brindarme fortaleza y resiliencia en cada etapa del camino, y por permitirme disfrutar verdaderamente este proceso.

A mis padres, Edgar y Laura, por su incondicional apoyo en cada decisión que tomé, por enseñarme que siempre es posible mejorar cuando se tiene determinación y, sobre todo, por motivarme a seguir adelante. Su confianza y aliento fueron fundamentales para llegar hasta aquí.

A mis abuelitos, Rafael y Josefina, por haber puesto ese granito de arena que me permitió continuar con mis estudios. Su apoyo y dedicación significaron mucho para mí, y siempre lo apreciaré

A mi familia, por su amor y confianza, por ser ese refugio al que siempre deseo regresar. Gracias por estar siempre ahí.

Meza Morales Henry Patricio

Agradecimientos

A mis padres, por dedicarme su tiempo, brindarme su confianza y permitirme alcanzar esta meta. Su apoyo incondicional, especialmente en los momentos más difíciles, ha sido fundamental para superar cada desafío y continuar avanzando con determinación.

A mis profesores, quienes con su paciencia, conocimiento y guía me han ayudado a crecer tanto académica como personalmente. Gracias por compartir su experiencia y enseñanzas, por inspirarme a seguir aprendiendo y por motivarme a dar siempre lo mejor de mí.

A mis compañeros Edwin, Carlos, Oliver, Kevin y Theo, por acompañarme en esta etapa. Compartir este camino con ustedes ha sido una experiencia invaluable, llena de aprendizajes, retos y momentos inolvidables que hicieron de este proceso algo más llevadero y enriquecedor.

A mi tutor, por su orientación y apoyo durante el desarrollo de esta investigación. Su compromiso, consejos y conocimiento han sido clave para llevar este proyecto a buen término. Gracias por su paciencia y por guiarme en cada paso de este proceso.

A todas las personas que, de una u otra manera, contribuyeron a que esta meta se hiciera realidad, mi más sincero agradecimiento.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Resumen

La detección de somnolencia durante la conducción, tanto diurna como nocturna, es un problema crítico debido al aumento significativo de accidentes de tránsito relacionados con la fatiga. Este incremento puede atribuirse al desgaste físico generado por las jornadas laborales, a condiciones de salud que afectan la concentración o incluso a factores ambientales como temperaturas elevadas en el entorno del conductor. En este contexto, el presente trabajo propone un modelo basado en la arquitectura Transformers RT-DETR para la detección de somnolencia, centrándose en la identificación de la zona de los ojos y la evaluación de su estado (despierto o dormido). El modelo será implementado en un sistema embebido con una Raspberry Pi 3 Modelo B+, diseñado para su integración en vehículos y el monitoreo en tiempo real del conductor.

El trabajo se divide en tres capítulos: el primero se centra en la investigación de la problemática de la somnolencia, revisando artículos y trabajos previos sobre modelos para la detección de somnolencia; el segundo capítulo aborda la creación y balanceo del dataset para el entrenamiento del modelo, que incluye datos específicos de somnolencia y otras fuentes de datos relevantes, además del desarrollo y entrenamiento del modelo RT-DETR para detección de objetos; en el tercer capítulo, se evalúan los resultados obtenidos mediante pruebas con diferentes sets, incluyendo rostros sin accesorios faciales, con lentes, con gafas, y en condiciones diurnas y nocturnas. Se comparan las métricas obtenidas con las de trabajos previos en la literatura. Además, se realizan pruebas en entornos simulados y reales con el sistema embebido, con la finalidad de evaluar su rendimiento.

Abstract

Drowsiness detection during daytime and nighttime driving is a critical problem due to the significant increase in fatigue-related traffic accidents. This increase can be attributed to physical wear and tear caused by working hours, health conditions that affect concentration, or even environmental factors such as high temperatures in the driver's environment. In this context, this work proposes a model based on the Transformers RT-DETR architecture for drowsiness detection, focusing on the identification of the eye area and the assessment of its state (awake or asleep). The model will be implemented in an embedded system with a Raspberry Pi 3 Model B+, designed for integration in vehicles and real-time monitoring of the driver.

The work is divided into three chapters: the first focuses on the investigation of the problem of drowsiness, reviewing previous articles and works on models for drowsiness detection; The second chapter addresses the creation and balancing of the dataset for model training, which includes drowsiness-specific data and other relevant data sources, as well as the development and training of the RT-DETR model for object detection; in the third chapter, the results obtained through tests with different sets are evaluated, including faces without facial accessories, with glasses, with lenses, and in daytime and nighttime conditions. The obtained metrics are compared with those of previous works in the literature. In addition, tests are performed in simulated and real environments with the embedded system, in order to evaluate its performance.

Introducción

Tema

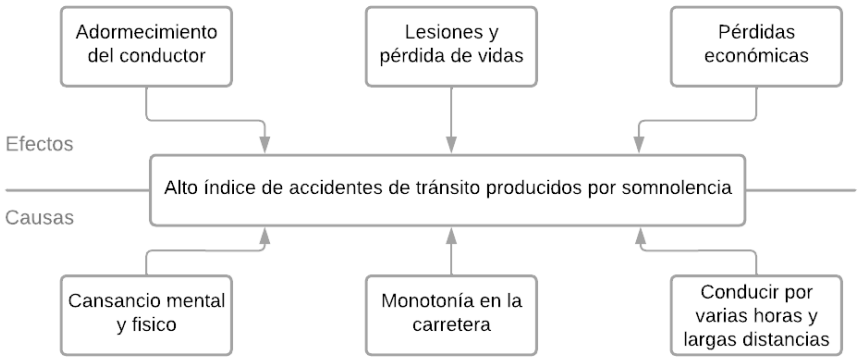
Implementación de una red neuronal Transformer para detección de somnolencia en conductores de vehículos durante la conducción diurna y nocturna.

Problema

La somnolencia es un factor de riesgo significativo que afecta a los conductores de vehículos en Ecuador y todo el mundo. Este riesgo se ve agravado por las condiciones a las que están expuestos los conductores, que incluyen el sedentarismo, las largas jornadas de conducción, síndrome del sueño insuficiente y apnea obstructiva del sueño (Gottlieb & Punjabi, 2020). Según un informe reciente publicado en el diario El Universo, se estima que hasta un 18.5 % de los conductores ecuatorianos pueden estar en riesgo debido al síndrome de apnea (El UNIVERSO, 2022), por lo cual existe la alta probabilidad de generar un accidente en las carreteras del Ecuador. Dormir lo suficiente no es solo una cuestión de salud y bienestar sino también una cuestión de responsabilidad social que aqueja a los conductores de vehículos que padecen de somnolencia diurna excesiva, esto se relaciona por la mala calidad de sueño que es considerada como un problema de salud pública en los países occidentales (Rodríguez González et al., 2018), el no enfatizar la importancia del sueño adecuado y de calidad puede producir disminución de la eficiencia laboral y mayor riesgo de accidentes de tránsito. Por lo mencionado, es necesario la implementación de un sistema embebido que ayude a la detección y alerta de somnolencia en conductores tanto en el día como en la noche el cual ayudará a evitar accidentes no deseados, proteger la integridad de usuarios de la carretera, prevenir pérdidas humanas, al tiempo que colabora en la mejora de la seguridad en las carreteras en su totalidad. La Figura 1, muestra el árbol de problemas.

Figura 1

Árbol de problema.



Fuente: Elaboración propia

Objetivos

Objetivo General

Desarrollar un sistema embebido mediante el uso de visión artificial para detección y alerta de somnolencia en conductores durante la conducción diurna y nocturna.

Objetivos Específicos

- Elaborar un marco teórico con respecto a la alerta de somnolencia en conductores aplicando la red neuronal Transformer.
- Desarrollar un sistema embebido que integre la red neuronal Transformer para la detección de somnolencia.
- Validar pruebas y resultados para detectar el estado de somnolencia en los conductores.

Alcance

Este trabajo tiene la finalidad de desarrollar e implementar un sistema embebido para la detección y alerta de somnolencia en conductores, utilizando una arquitectura de detección de objetos optimizada para imágenes en tiempo real. Para ello, se integrará un backbone convolucional con mecanismos de atención deformable, garantizando un procesamiento eficiente y preciso en entornos de conducción diurna y nocturna.

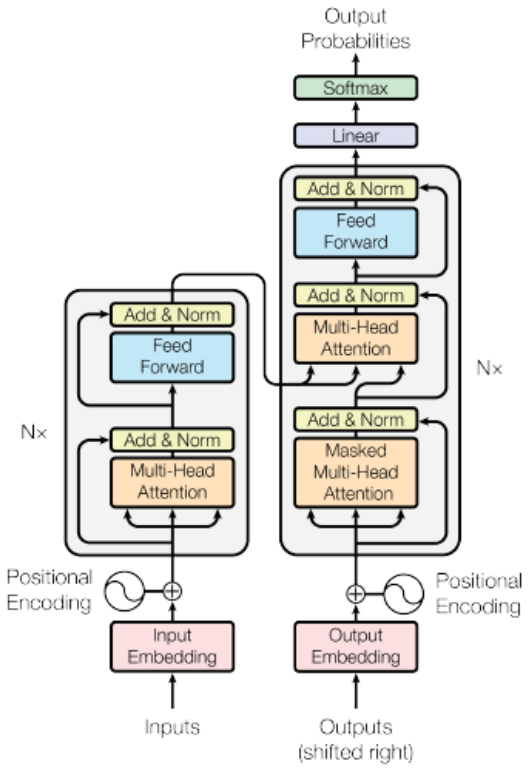
El sistema empleará el modelo RT-DETR, el cual constituye una adaptación clave de la arquitectura Transformer para tareas de detección de objetos. Es importante destacar que los

modelos Transformer sirven como base para múltiples aplicaciones de aprendizaje profundo, de las cuales RT-DETR deriva para la tarea específica de detección de objetos. La arquitectura general de los modelos Transformer se ilustra en la Figura 2.

El desarrollo del proyecto seguirá la metodología Knowledge Discovery in Databases (KDD) (Fayyad, 1996), la cual proporciona un marco estructurado para la recopilación, preprocesamiento, modelado y análisis de datos.

Figura 2

Arquitectura de modelo Transformer.



Fuente: (Vaswani et al., 2017)

El modelo entrenado de red neuronal Transformer resultante se integrará en un sistema embebido de hardware y software libre Jetson Nano y/o Raspberry Pi, esto garantizará que el sistema sea portátil y fácilmente instalable en cualquier vehículo, además, el sistema estará equipado con una cámara de video que capturará las imágenes que se someterán a un análisis para detectar el rostro del conductor y evaluar la apertura de los ojos, también, se agrega a este

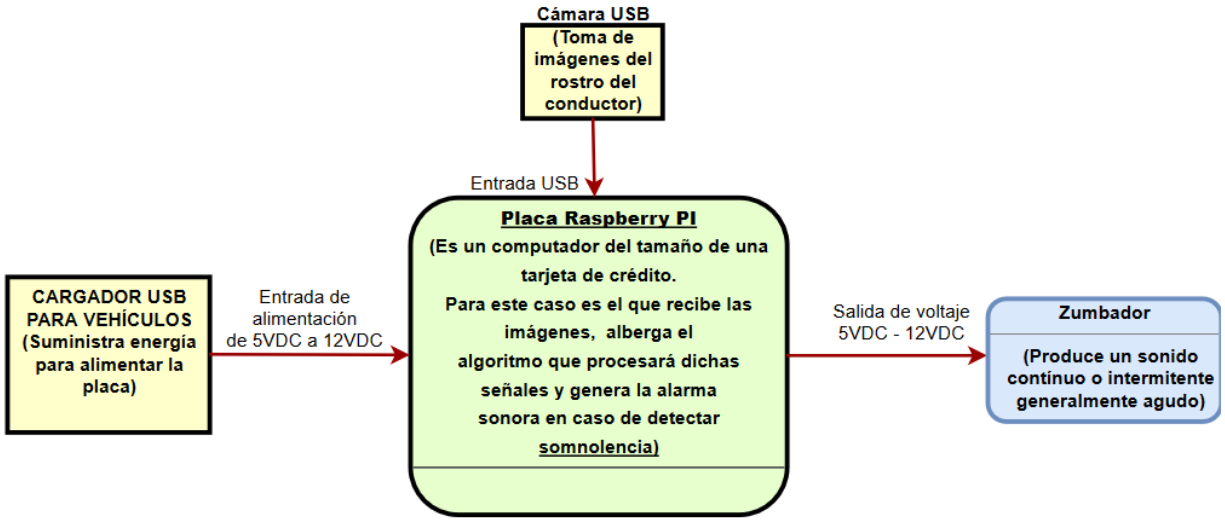
sistema un bocina para emitir una alarma y alertar al conductor. La Figura 3, muestra la estructura del sistema embebido.

La validación del sistema propuesto se realizará mediante el análisis de métricas numéricas que evaluarán la precisión. El objetivo principal de esta validación es asegurar que el sistema sea eficaz y pueda confiarse en su capacidad para detectar la somnolencia del conductor, lo que en última instancia contribuirá a elevar los niveles de seguridad en las vías.

La incorporación del modelo Transformers se realizará utilizando Python como lenguaje de programación y aprovechando bibliotecas especializadas en el aprendizaje profundo como PyTorch. Se empleará el modelo RT-DETR, el cual permite realizar detección de objetos optimizada para imágenes en tiempo real. El modelo se entrenará utilizando conjuntos de imágenes propias y disponibles en Internet, previamente elaboradas y etiquetadas, con el propósito de optimizar su desempeño y precisión en la detección de síntomas de somnolencia.

Figura 3

Estructura del sistema embebido.



Fuente: (Alba Neppas, 2020)

Metodología

El proyecto se centra en la investigación aplicada, que busca generar conocimiento con aplicaciones directas en la sociedad o en el sector productivo. El objetivo es llevar los hallazgos tecnológicos de la investigación básica y vincularlos con la creación de un producto o sistema (Duoc UC, 2023).

Se utilizarán varias técnicas para recopilar datos y llevar a cabo la investigación. Estas técnicas incluyen la utilización de Internet para obtener información y materiales, el fichaje para registrar datos de video, la observación del entorno donde funcionará el sistema embebido y pruebas de campo para evaluar el sistema en situaciones reales.

Se mencionan varios instrumentos y recursos que se utilizarán en el proyecto, como Mendeley para la gestión de citas bibliográficas, PyTorch y Ultralytics para el desarrollo del modelo de red neuronal, Python como lenguaje de programación, placas Jetson Nano y/o Raspberry Pi para la integración del sistema embebido, una cámara de video para la captura de imágenes, Google Colab y HPC CEDIA como entornos para el entrenamiento del modelo.

Justificación

La seguridad vial es una cuestión crítica en todo el mundo, y la somnolencia al volante representa un peligro significativo que conlleva riesgos de accidentes graves y, en algunos casos, fatales. La implementación de una red neuronal Transformer para detectar la somnolencia en conductores es una innovación tecnológica que aborda directamente el Objetivo de Desarrollo Sostenible (ODS) número 9, que promueve la "Industria, Innovación e Infraestructura" (Naciones Unidas, 22).

La seguridad ciudadana es un aspecto fundamental para garantizar la convivencia pacífica y segura de los ciudadanos. La somnolencia al volante es una preocupación seria en términos de seguridad vial y puede tener un impacto significativo en la seguridad de las personas en las carreteras. La implementación de una red neuronal Transformers para detectar la somnolencia en conductores de vehículos contribuye a la creación de oportunidades en el ámbito de la seguridad integral (Secretaria Nacional de Planificación, 2021).

Justificación Tecnológica

La implementación de una red neuronal Transformers radica en su capacidad para aprovechar avances tecnológicos para abordar un problema crítico de seguridad vial. Esta tecnología es precisa, adaptable y escalable.

Justificación Social

Su implementación promete mejorar la seguridad en las carreteras y salvar vidas a través de la detección oportuna de la somnolencia en conductores.

Capítulo 1

Marco Teórico

1.1. Sueño y somnolencia

El sueño es un proceso fisiológico cíclico que experimentan los seres vivos, especialmente los animales, incluyendo a los seres humanos. Durante este proceso el cuerpo experimenta cambios neurológicos esenciales que fortalecen a la recuperación de energías, consolidación de la memoria, regulación del estado de ánimo y colaboración en el buen funcionamiento del organismo.

En los seres humanos el sueño no es el valor proporcional por dormir menos horas de la cantidad necesaria, ya que esto puede variar por cada persona dependiendo la edad, estilo de vida y otros factores que inciten a realizar una actividad que requiera esfuerzo físico o mental. Al realizar una actividad existe un desgaste de energía lo cual es el indicio principal que conduce a la somnolencia en una persona. Algunas personas optan por consumir estimulantes para aplazar el sueño que al pasar los años lo único que han logrado es terminar afectando su salud, cuando la única forma de combatir el sueño es descansando, de aquí radica la importancia de tener buenos hábitos de sueño para evitar enfermedades (Castillo et al., 2020).

La somnolencia es la necesidad convertida en el “deseo de estar dormido” y de “estar despierto”, generalmente se presenta como interrupción para ejercer una actividad. Un estado subjetivo de sueño que puede afectar relaciones interpersonales, rendimiento académico y laboral (Zarza García et al., 2023). Este estado de adormecimiento puede manifestarse en diversas horas del día, incluyendo la mañana, la tarde y, de manera notable, durante la noche. Está considerablemente influenciado por diversos factores como el género, el peso, las condiciones de salud y, de manera destacada, los patrones y hábitos relacionados con el sueño.

Existen varios tipos de somnolencia, y las causas de esta condición pueden variar. Sin embargo, hay un tipo específico de somnolencia que es más frecuente entre los conductores de vehículos, conocida como Somnolencia Diurna Excesiva (SDE).

La SDE se caracteriza por la incapacidad de mantenerse alerta (Andreu & Castresana, 2023), lo que representa un riesgo significativo y puede resultar en accidentes de tránsito. Esta condición se considera transitoria y puede asociarse con malos hábitos o interrupciones en los patrones de sueño habituales. Las posibles causas en este tipo de somnolencia incluyen la

privación del sueño, trastornos del sueño, consumo de sustancias, factores ambientales, entre otros.

1.1.1. Causas del estado de somnolencia

Es fundamental aumentar la conciencia pública sobre la importancia de una buena higiene del sueño y promover prácticas de conducción seguras, como evitar manejar cuando se está extremadamente cansado, para abordar el problema de la somnolencia de los conductores. Los conductores también deben tomar medidas proactivas, como tomar descansos regulares y, en caso necesario, buscar un lugar seguro para descansar antes de continuar conduciendo si sienten somnolencia.

La somnolencia en una persona puede deberse a una variedad de causas las cuales pueden terminar afectando su salud de la persona que lo padece. Algunas de las causas más habituales de la somnolencia son:

- **Privación del sueño:** De acuerdo con (Castillo et al., 2020), "es la condición de insuficiencia del sueño", que se refiere a la falta de descanso antes de realizar una actividad. Esto puede ser el resultado de participar en actividades que limitan la cantidad y calidad del descanso o no dormir las horas necesarias durante la noche. La persona que lo padezca normalmente tiene consecuencias a corto plazo.
- **Trastornos del sueño:** Las dificultades para conciliar el sueño, conocidas como trastornos del sueño, pueden tener consecuencias duraderas, a diferencia de la privación del sueño. Estos trastornos son comunes en la población adulta, por lo que es crucial diagnosticarlos y tratarlos oportunamente, ya que pueden causar alteraciones significativas en la calidad de vida del individuo y de las personas que lo rodean (Guadamuz et al., 2022).
- **Consumo de sustancias:** Según (García et al., 2023) menciona que a mayor cansancio mayor es la implicación de alcohol. El consumo de alcohol aumenta los síntomas de somnolencia y reduce la alerta inmediata del conductor hacia cualquier accidente en la vía, lo que aumenta el riesgo de lesiones, problemas económicos e incluso la pérdida de vidas en las vías públicas.
- **Factores Ambientales:** Tiene un impacto significativo en la calidad del sueño y contribuye significativamente a los problemas de somnolencia, por lo que debe tener con

frecuencia las siguientes características: oscuridad adecuada, ausencia de ruido, temperatura adecuada y evitar la exposición a dispositivos electrónicos (Bernat et al., 2021).

1.1.2. Somnolencia en conductores

Muchos conductores pasan por alta la importancia de la somnolencia, lo que afecta negativamente su capacidad para conducir de manera segura y aumenta la probabilidad de accidentes. Estos accidentes pueden causar salidas de carretera o desviaciones de las vías, poniendo en peligro tanto a los conductores como a las personas que se encuentran cerca del accidente (Gillin, 2023).

Entrar en un estado de privación de sueño o somnolencia afecta directamente los sentidos de percepción, disminuyendo la capacidad de reacción auditiva y mermando los reflejos del individuo. Esto incrementa de manera notable el riesgo de sufrir un accidente de tráfico al encontrarse al volante en dicho estado.

Si una persona duerme menos de 6 horas al día, equivale a tener un nivel de alcohol en sangre del 0,08%, lo que se considera un estado de embriaguez legalmente y tiene la misma probabilidad de causar un accidente. Este problema afecta principalmente a los hombres jóvenes, que en su mayoría lo experimentan debido a la realización de turnos laborales prolongados o la asunción de largas rutas.

Según (UNASEV, 2020), los efectos de la somnolencia al volante son más frecuentes durante el día que durante la noche. Algunos de los efectos más comunes entre conductores son:

- El tiempo de respuesta se ve prolongado.
- Reduce la capacidad de enfocarse.
- La toma de decisiones se ralentiza, propiciando la ocurrencia de errores.
- Incrementa la probabilidad de tomar decisiones incorrectas.
- Presenta desafíos en las habilidades motoras.
- Se manifiestas movimientos más automáticos.
- Se producen alteraciones en las funciones sensoriales.

- Pueden surgir episodios de micro sueños.
- Se observan cambios en la percepción.
- Se experimentan modificaciones en el comportamiento.

No es redundante tener en cuenta recomendaciones preventivas para mantenerse alerta y evitar la somnolencia durante la conducción. Según (El Telégrafo, 2022), nos incluyen las siguientes recomendaciones como las más influyentes:

- Realizar paradas estratégicas cada 2 horas.
- No comer en exceso.
- Masticar de manera activa.
- Mantener el ambiente fresco.
- Hidratarse de manera correcta.
- Descansar.

1.1.3. Accidentes por somnolencia al conducir

Los accidentes de tránsito siguen siendo la causa más frecuente de muerte en Ecuador, según las estadísticas proporcionadas por la ANT, se revela que, en el año 2023, en Ecuador, se registraron 17.257 siniestros de tráfico, una cifra significativamente menor en comparación con el año anterior, como en 2022, donde se reportaron 21.739 siniestros, con un total de 2.202 muertes. A pesar de la disminución en la cantidad de siniestros, el número de muertes ha experimentado un aumento, llegando a un total de 1.942 personas, lo que representa un 11.25%. En 2022, este porcentaje fue del 10.13% con respecto a la cantidad de accidentes (ANT, 2023).

De acuerdo con los datos de la (DGT, 2023), la somnolencia es señalada como responsable del 7% de todos los accidentes de tráfico reportados por la ANT. La falta de conciencia de los conductores es la causa de esta situación, ya que el 45% de los adultos en el grupo de 20 a 29 años duerme menos de 7 horas al día, lo que los hace particularmente vulnerables a la somnolencia. Además, este grupo es altamente susceptible al consumo de drogas, alcohol u otros estupefacientes.

No solo la falta de conciencia ha sido la causa de los accidentes de tráfico en las carreteras ecuatorianas, sino que también hay conductores que sufren de trastornos del sueño.

Según un reporte del diario El Universo, la apnea obstructiva del sueño es un trastorno que empeora la somnolencia diurna y se convierte en un problema de salud ocupacional en la industria del transporte, afectando al 18.5% de los conductores ecuatorianos. Este trastorno contribuye significativamente a los accidentes, causando más lesionados y fallecidos (El UNIVERSO, 2022).

1.1.4. Sistemas de detección de somnolencia

Actualmente, contamos con varios sistemas de detección de somnolencia diseñados para detectar y alertar al conductor en caso de que esté agotado, lo que le permite tomar las medidas necesarias para prevenir accidentes. A largo plazo, estos eventos causan costosos daños materiales, lesiones y, en el peor de los casos, la pérdida de vidas humanas.

- **Sistemas por parámetros:** Los sistemas de medición de parámetros durante la conducción se concentran en crear perfiles específicos que describen la conducta del conductor. Estos perfiles se crean observando el nivel de actividad del conductor, incluido el uso de los controles del vehículo y su influencia en la conducción. Se observa el entorno del automóvil, incluidos los cambios de velocidad, aceleración, movimientos del volante y el uso del acelerador y el freno. Además, para determinar la posición del vehículo, se examinan elementos externos como el estado del semáforo, las condiciones del viento y la identificación de límites de carril.
- **Sistemas de Monitoreo de Actividad Cerebral (EEG):** Este sistema utiliza sensores EEG específicos para monitorear la actividad cerebral del conductor. Estos sensores recopilan datos y los analizan para identificar signos de fatiga o somnolencia. La información se transmite a una placa de procesamiento, que controla el estado del conductor. El sensor EEG puede adaptarse a cualquier entorno de gracias a su ergonomía, lo que reduce las interferencias con las actividades del conductor. Para hacer predicciones, la efectividad depende de recopilar datos en tiempo real y compararlos con bases de datos anteriores.
- **Sistemas por reconocimiento facial:** Este sistema se centra en analizar los rasgos faciales del conductor en tiempo real para detectar signos de fatiga o somnolencia. La detección se realiza mediante el seguimiento de características como:
    - Parpadeo lento y prolongado.

- Frecuencia de parpadeo disminuida o aumentada.
- Duración de los ojos cerrados.
- Asimetría en la apertura de los ojos.
- Mirada fija y falta de parpadeo

Estas señales pueden indicar una disminución en el nivel de alerta del conductor y, por lo tanto, un mayor riesgo de somnolencia.

Para el ejercicio de nuestra tesis, se implementará un sistema de detección de objetos basado en la arquitectura RT-DETR, el cual utiliza mecanismos de atención deformable para procesar imágenes capturadas por una cámara en tiempo real. Estas imágenes servirán como entrada del modelo entrenado, el cual clasificará si los ojos están abiertos o cerrados y proporcionará una medida de confianza en la predicción.

El sistema estará diseñado para emitir alertas en caso de que el conductor presente señales de somnolencia, ayudando a prevenir accidentes de tránsito y garantizando su seguridad.

El modelo de red neuronal será entrenado utilizando un conjunto de datos etiquetado con imágenes de rostros en diversas condiciones de iluminación y posiciones, con etiquetas que especifican el estado de los ojos (abiertos o cerrados). Además, se utilizarán técnicas de aumento de datos para mejorar la generalización del modelo en entornos reales.

1.2. Red neuronal Transformers

1.2.1. Red neuronal

El aprendizaje profundo y la inteligencia artificial (IA) dependen de las redes neuronales. Se utilizan para realizar tareas que requieren reconocimiento de patrones, clasificación, regresión y toma de decisiones, entre otras, simulando la estructura y el funcionamiento del cerebro humano (Ponce et al., 2023).

- **Arquitectura de una red neuronal:** Las neuronas se encuentran organizadas en capas y son la base de una red neuronal artificial (RNA), estas capas son las siguientes:
    - *Capa de entrada*, recibe las primeras señales o datos y las transmite a través de la red.

- *Capas ocultas*, las conexiones ponderadas procesan la información que reciben.
- *Capa de salida*, crea el resultado final o la predicción de la red.

Las arquitecturas y topologías de las redes neuronales varían. Se destaca la idea de redes neuronales profundas, las cuales son redes con un número mayor de capas ocultas entre la capa de entrada y la capa de salida. La profundidad de esta red aumenta a medida que aumenta la cantidad de capas ocultas. Este método ha demostrado ser particularmente efectivo en el procesamiento de grandes conjuntos de datos y en la resolución de tareas complejas.

En la actualidad, sobresalen diversas arquitecturas de aprendizaje profundo, las cuales abarcan:

- Redes Neuronales Feedforward Profundas (DNN)
- Redes Neuronales Convolucionales (CNN)
- Redes Neuronales Residuales (ResNet)
- Redes Neuronales Generativas Adversarias (GAN)
- Redes Neuronales Recurrentes (RNN)
- Transformers

Estos grupos de redes neuronales fueron creados con propósitos específicos, pero su versatilidad les ha permitido ser aplicadas en diversas áreas de la inteligencia artificial, y en muchos casos, se han adaptado para su uso en tareas más allá de sus propósitos originales.

A medida que la tecnología se ha integrado en nuestra cotidianidad, nuestra dependencia de las redes neuronales ha aumentado. Esto se debe a que cada vez las utilizamos más en una diversidad de ámbitos, aprovechando su capacidad para aprender patrones complejos y desempeñar funciones específicas, como en medicina, finanzas, robótica, visión por computadora y procesamiento del lenguaje natural.

Las Redes Neuronales Recurrentes (RNN) son ampliamente reconocidas por su capacidad para gestionar secuencias, que se utilizan en tareas como la traducción

automática y la creación de texto. Sin embargo, a medida que las RNN procesan más elementos, enfrentan dificultades para recordar información pasada, lo cual es una limitación inherente. Este desafío surge debido al algoritmo “back-propagation through time” (Ronald J & Zipser, 1995).

Por otro lado, el campo del Procesamiento del Lenguaje Natural (PLN) ha experimentado un cambio significativo gracias a los Transformers, que se pueden ver en modelos como BERT y GPT. Su ingenioso uso de mecanismos de atención de ha transformado la eficiencia del procesamiento, permitiendo la captura de relaciones contextuales más efectiva en textos extensos.

1.2.2. Transformer

Transformer es una arquitectura de red neuronal que se introdujo por primera vez en el contexto del procesamiento de lenguaje natural (NLP) pero que ha demostrado ser versátil en diversas tareas, incluyendo la visión por computadora. Esta arquitectura se fundamenta exclusivamente en mecanismos de atención, eliminando por completo la dependencia de convoluciones y recurrencia (Vaswani et al., 2017).

- **Mecanismos de atención:** Introducido por (Bahdanau et al., 2014) en el aprendizaje profundo con el propósito de brindar soluciones importantes a la traducción automática y mejorar la capacidad de los modelos, incluidos los Encoder y Decoder.

  Mecanismo de atención. es una técnica de aprendizaje profundo que permite que el modelo se concentre en partes específicas de la entrada, asignando pesos a los elementos de la secuencia. Este enfoque supera la limitación de representar toda la información de entrada con un solo vector fijo, utilizando conjuntos de vectores para seleccionar subconjuntos específicos en cada paso, mejorando la capacidad del modelo para manejar secuencias más largas (Arana, 2021).

  Los modelos de atención son cruciales para la creación de secuencias persuasivas y tareas de transducción, incorporar la poderosa técnica de autoatención, que conecta diferentes posiciones dentro de una secuencia, ha demostrado ser útil en una variedad de aplicaciones. Estos incluyen la comprensión lectora, los resúmenes abstractivos, la

vinculación textual y el aprendizaje de representaciones de oraciones independientes de la tarea.

- **Arquitectura del modelo:** La utilización de mecanismos de atención difiere significativamente entre las redes neuronales recurrentes (RNN) y las redes neuronales Transformers. A diferencia de las RNN, las redes Transformers no imponen requisitos de orden en los datos y, por lo tanto, emplean un mecanismo de atención llamado "self-attention" para identificar patrones en los datos (Vaswani et al., 2017). La arquitectura "Transformer" se compone de capas de codificación y decodificación, y aunque suele asociarse con las Redes Neuronales Recurrentes, las capas recurrentes son sustituidas por un modelo de "self-attention".

  A continuación, se definen los componentes que integran un Transformer, tal como lo muestra la Figura 4.

  - ***Encoder,*** el codificador está compuesto por una pila de N = 6 capas idénticas con 2 subcapas cada una, la primera subcapa es un mecanismo de autoatención de múltiples cabezales, mientras que la segunda es una simple red de alimentación directa, completamente conectada en función de la posición. Para mejorar el flujo de información, se implementa una conexión residual y normalización de capa alrededor de ambas subcapas.
  - ***Decoder,*** el decodificador tiene una composición similar a la del Encoder con una pila de N = 6 capas idénticas y sigue desempeñando un papel importante en la generación de secuencias de salida basadas en la información del

Para lograr una comprensión completa de esta arquitectura, no es suficiente con definir e identificar los roles tanto del codificador como el decodificador; también es esencial familiarizarse con los componentes y conceptos que constituyen la estructura de un Transformer.

- ***Input Embedding,*** etapa inicial de transformación, donde las oraciones se convierten en vectores mediante un proceso de reconversión.
- ***Positional Encoding,*** incorpora información sobre la posición de cada palabra en la representación vectorial de entrada. Este token posicional informa a la red sobre el orden de los diversos tokens en una secuencia, ya que una

secuencia sin un orden claro puede resultar en un caos total. Se utiliza una formulación basada en funciones sen y cos, para dotar a la red neuronal de la capacidad de entender el orden de las palabras proporcionadas como entrada.

$$\mathrm{PE}_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

Dónde "*pos"* es la posición y "*i*" es la dimensión. Por tanto, cada dimensión de la codificación posicional corresponde a una sinusoide.

- ***Attention,*** corresponde al mapeo de una consulta en la cual sus salidas están representadas en vectores consulta, las claves y valores. El resultado se obtiene sumando los valores de manera ponderada y utilizando una función que evalúa la compatibilidad entre la consulta y la clave correspondiente para determinar el peso asignado a cada valor.

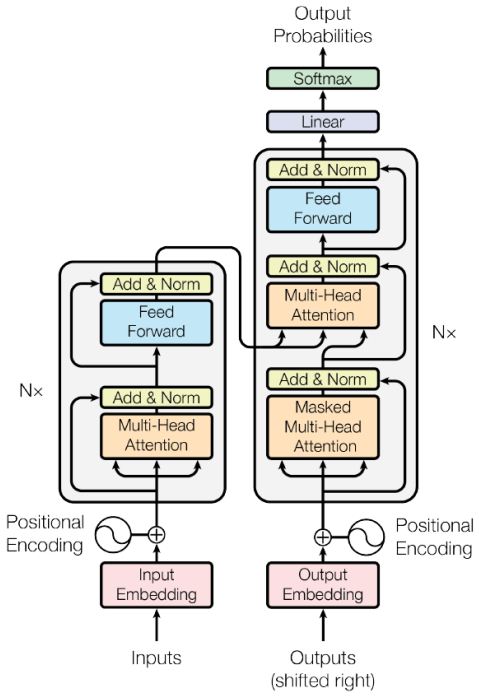
  La función de atención representa un conjunto de consultas simultáneamente, empaquetadas en una matriz Q. Igualmente K para la matriz claves y V para la matriz valores. Calculamos la matriz de resultado como:

$$\mathrm{Attention(Q,K,V)} = \mathrm{softmax}\left(\frac{\mathrm{QKT}}{\sqrt{\mathrm{d_k}}}\right)\mathrm{V}$$

- ***Scaled Dot-Product Attention,*** valor númerico que aumentara proporcionalmente a la alineación de las direcciones de los vectores, Figura 5 (izquierda).
- ***Multi-Head Attention,*** compuesta por múltiples capas de atención que se ejecutan en paralelo. Cada cabeza de atención realiza su propio cálculo de atención, brindando al modelo una representación más rica, diversa en información y en comportamiento relacionado con la estructura sintáctica y semántica de las oraciones, Figura 5 (derecha).

Figura 4

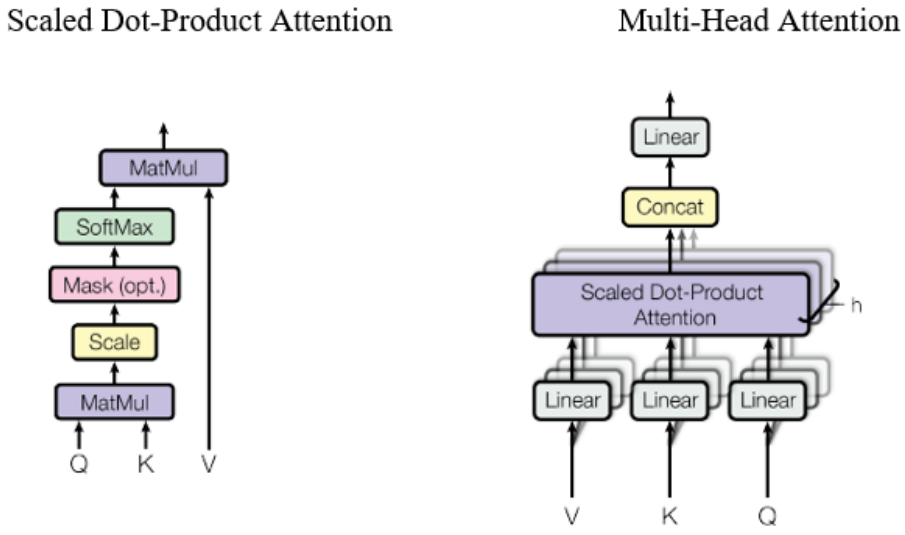
Arquitectura del modelo Transformer.



Fuente: (Vaswani et al., 2017)

Figura 5

Producto escalar y atención multicabezal.



Fuente: (Vaswani et al., 2017)

Ventajas

- *Procesamiento en paralelo,* debido a su arquitectura, posibilitan el uso eficiente de todos los recursos de hardware. Esto facilita el procesamiento de conjuntos de entrenamiento considerablemente más extensos en un período de tiempo más corto (Garcias & Lucero, 2023).
- *Memoria de largo plazo,* al implementar paralelismo también les permite tener una memoria de mucho más largo plazo.

Desventajas

- Necesita un grande conjunto de datos para un aprendizaje efectivo.
- Sobreajuste en conjuntos de datos pequeños.
- Requiere recursos computacionales significativos.

1.2.3. ¿Cuándo y en qué tipos de problemas aplicar la arquitectura Transformer?

La arquitectura Transformer responde efectivamente a una variedad de tareas, pero especialmente cuando aquellas tareas involucren datos secuenciales o estructurados.

Aquí hay algunos escenarios en los que la arquitectura Transformer puede ser aplicada con éxito:

- **Texto:** Clasificación de texto, extracción de información, respuesta a preguntas, resumir, traducción.
- **Imágenes:** Clasificación de imágenes, detección de objetos y segmentación.
- **Audio:** Reconocimiento de voz y clasificación de audio.
- **Multimodal:** Respuesta a preguntas en tablas, reconocimiento óptico de caracteres, extracción de información de documentos escaneados, clasificación de videos y respuesta visual a preguntas.

1.2.4. Visión artificial

La visión artificial se presenta como una rama científica que aborda técnicas para capturar, procesar, analizar y comprender imágenes del entorno real, con la meta de generar datos numéricos o simbólicos para la interpretación por sistemas informáticos. Su propósito central es la creación de enfoques automatizados para identificar patrones complejos presentes en imágenes

provenientes de diversos contextos. Se utiliza en campos como la robótica, la medicina, y en sistemas de seguridad, entre otros. Además, emplea técnicas de aprendizaje automático y aprendizaje profundo, con componentes como cámaras y sensores para la captación de datos, y procesadores de imágenes para el análisis.

- **Visión Transformer (ViT):** Es el algoritmo principal que utilizó la arquitectura Transformers para la visión artificial. ViT divide una imagen en parches y las transforma en secuencias para que un modelo Transformer pueda procesarlas en lugar de usar capas convolucionales (Mangalam et al., 2022). Esto se ha demostrado efectivo en las tareas de clasificación de imágenes.

  Según, (Ruan et al., 2022) ViT se compone de tres partes: 1) parche e incrustación posicional, 2) codificador transformador y 3) cabezal de perceptrón multicapa (MLP).
- **Swin Transformer (Swin-T):** Shifted Window Transformer es una variante mejorada del Vision Transformer (ViT) diseñada para mejorar la eficiencia y escalabilidad en tareas de visión artificial como detección de objetos, segmentación y clasificación de imágenes (Liu et al., 2021). A diferencia de ViT, que aplica autoatención global, Swin Transformer introduce un mecanismo de autoatención por ventanas deslizantes (shifted windows), lo que reduce significativamente el costo computacional sin perder información contextual importante.

  Según Liu et al. (2021), Swin Transformer se compone de tres partes principales:
  - *División jerárquica de imágenes,* donde la imagen se procesa en múltiples resoluciones para capturar detalles locales y globales.
  - *Autoatención en ventanas deslizantes,* que permite aplicar autoatención dentro de regiones locales de la imagen, mejorando la eficiencia en comparación con la autoatención global de ViT.
  - *Estructura piramidal,* que ayuda a Swin Transformer a manejar imágenes de diferentes escalas y mejorar su capacidad de generalización en tareas complejas.
- **Data- efficient Image Transformer (DeiT):** Utiliza el aprendizaje transferido para mejorar el uso de datos en Visión por Computadora. Incluso con conjuntos de datos más pequeños, este modelo ha obtenido resultados competitivos en la clasificación de

imágenes, aprovechando la capacidad de generalización de modelos preentrenados en conjuntos de datos más grandes (Touvron et al., 2021).

- **Detection Transformer (DETR):** Es un modelo innovador para la detección de objetos basado en la arquitectura Transformer. Su diseño simplifica significativamente el pipeline de detección al eliminar la necesidad de mecanismos tradicionales como las propuestas de regiones y la supresión de no máximos (NMS).

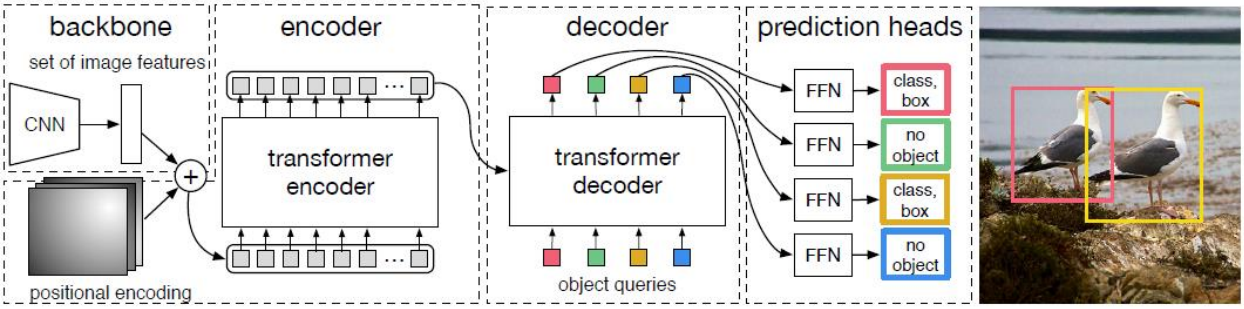
  La arquitectura general de DETR es sorprendentemente sencilla y consta de tres componentes principales (Carion et al., 2020):

  - *Backbone CNN*, se encarga de extraer una representación compacta de características a partir de la imagen de entrada. Se puede emplear cualquier red troncal convolucional estándar, como ResNet.
  - *Transformador Encoder-Decoder,* procesa las características extraídas y modela relaciones espaciales de largo alcance mediante mecanismos de atención global.

    En la Figura 6, podemos observar la arquitectura de DETR, donde se visualiza la interacción entre el backbone convolucional, el transformador y la red de salida.

Figura 6

Arquitectura DETR.



Fuente: (Carion et al., 2020)

- **Deformable Detection Transformer (Deformable DETR):** Mejora la eficiencia y la velocidad de convergencia del modelo DETR original al introducir un módulo de atención deformable multiescala. Este módulo permite que el modelo se enfoque en un

conjunto reducido de puntos clave alrededor de una referencia, en lugar de considerar toda la imagen, lo que optimiza el proceso de detección (Zhu et al., 2020).

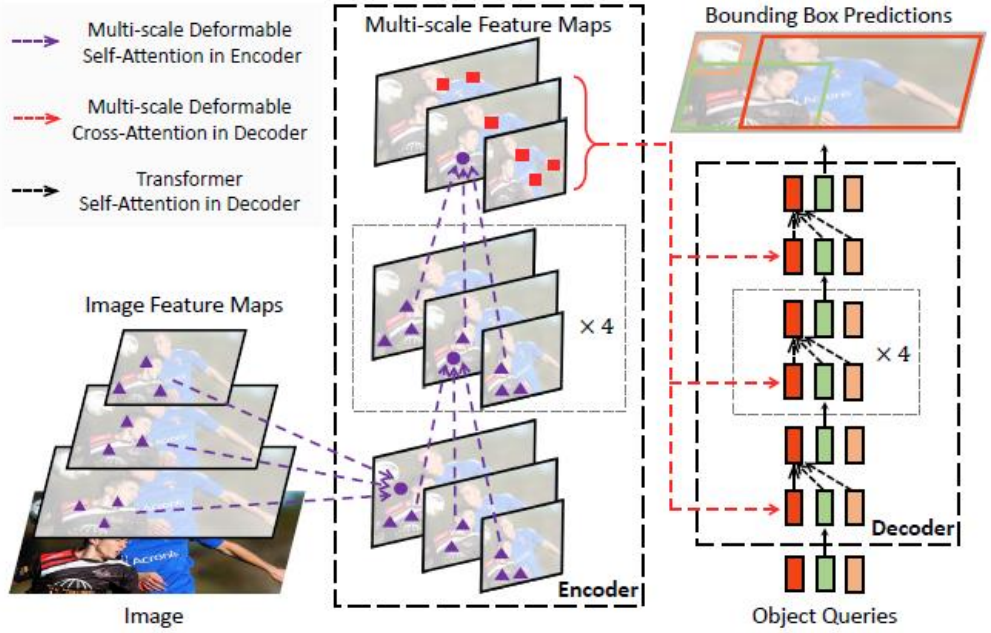
Los componentes principales de Deformable DETR son:

- Backbone CNN, una red neuronal convolucional (por ejemplo, ResNet-50 o ResNet-101) que extrae mapas de características de la imagen de entrada.
- Módulo de Atención Deformable Multiescala, este módulo aplica atención a un conjunto limitado de puntos de muestreo alrededor de una referencia en múltiples escalas, permitiendo al modelo capturar información relevante de diferentes niveles de resolución sin procesar toda la imagen.
- Codificador-Decoder Transformer, el codificador procesa las características extraídas por la CNN, mientras que el decodificador utiliza consultas de objetos para predecir las posiciones y clases de los objetos en la imagen.
- Red Feed-Forward (FFN), genera las predicciones finales de las cajas delimitadoras y las categorías de los objetos detectados.

Para una representación visual detallada, se puede consultar la Figura 7, donde se muestra la arquitectura general de Deformable DETR.

Figura 7

Arquitectura Deformable DETR.



Fuente: (Zhu et al., 2020)

Se propuso Deformable DETR mitigar los problemas de convergencia lenta y dificultad en la detección de objetos pequeños en DETR, cuyos módulos de atención solo atienden a un pequeño conjunto de puntos de muestreo clave alrededor de una referencia. Gracias a esta optimización, Deformable DETR puede lograr un mejor rendimiento que DETR (especialmente en la detección de objetos pequeños) con 10 veces menos épocas de entrenamiento.

1.2.4.1. Real-Time Detection Transformer (RT-DETR)

Es un modelo avanzado de detección de objetos en tiempo real basado en la arquitectura Vision Transformer (ViT), desarrollado por Baidu. Se diferencia de modelos previos como DETR y Deformable DETR al integrar un codificador híbrido eficiente, selección de consultas basada en IoU y ajuste dinámico de la velocidad de inferencia, lo que le permite lograr un rendimiento superior en tareas de detección de objetos en tiempo real (Zhao et al., 2024).

En la Figura 8, se muestra la arquitectura de RT-DETR, donde se observa como las tres últimas etapas de la estructura principal (backbone convolucional) sirven como entrada para el codificador híbrido eficiente. Este codificador optimiza el procesamiento de características

multiescala a través de dos mecanismos clave: la interacción de características intraescala basada en atención (AIFI) y la fusión de características entre escalas basada en CNN (CCFM).

Luego, la selección de consultas se realiza mediante un enfoque de incertidumbre mínima, que permite elegir un conjunto fijo de características generadas por el codificador y utilizarlas como consultas iniciales de objetos para el decodificador. Finalmente, el decodificador, que incorpora cabezas de predicción auxiliares, refina de manera iterativa estas consultas para mejorar la precisión de las detecciones, asignando correctamente categorías y cuadros delimitadores a los objetos detectados en la imagen.

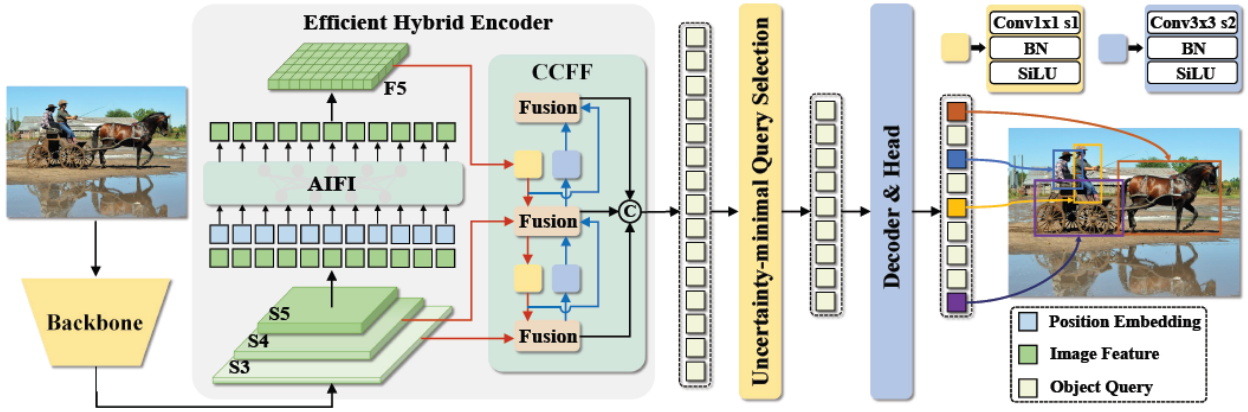
Esta combinación de estrategias permite que RT-DETR logre un equilibrio entre eficiencia computacional y precisión en tiempo real, siendo capaz de procesar imágenes con menor costo computacional y mayor capacidad de adaptación a diferentes escenarios sin la necesidad de reentrenamiento.

- **Arquitectura RT-DETR:** RT-DETR introduce mejoras clave sobre sus predecesores, combinando una estructura eficiente basada en atención deformable con optimización para acelerar la inferencia y mejorar la detección de objetos pequeños. Su arquitectura se compone de los siguientes elementos:
    - *Backbone Convolucional,* extrae representaciones multiescala de la imagen de entrada. Utiliza las tres últimas etapas {S3, S4, S5} como entrada para el codificador híbrido.
    - *Codificador Híbrido Eficiente,* transforma las características multiescala en una representación optimizada para la detección de objetos. Se compone de dos módulos principales:
        - Interacción de Características Intraescala (AIFI), procesa información dentro de una misma escala sin afectar la estructura global.
        - Módulo de Fusión de Características de Escala Cruzada (CCFM), integra información entre diferentes escalas, mejorando la detección de objetos pequeños.
    - *Selección de Consultas Basada en IoU,* optimiza la inicialización de los objetos detectados mediante el mecanismo IoU-aware Query Selection, permitiendo que el modelo se enfoque en los objetos más relevantes y mejore la precisión sin incrementar el costo computacional.

- *Decodificador con Cabezas de Predicción Auxiliares*, refina iterativamente las consultas de objetos para generar cajas delimitadoras y puntuaciones de confianza, optimizando el proceso de detección y reduciendo la cantidad de épocas necesarias para la convergencia.
- *Velocidad de Inferencia Adaptable*, facilita la optimización para diferentes dispositivos y escenarios de detección en tiempo real.

Figura 8

Arquitectura RT-DETR.



Fuente: (Zhao et al., 2024)

Ventajas

- Más rápido que DETR y Deformable DETR en detección en tiempo real.
- Mejor rendimiento en objetos pequeños, ideal para aplicaciones de precisión.
- Optimización eficiente de la selección de consultas, reduciendo cálculos innecesarios.
- Escalabilidad y adaptabilidad, permitiendo ajuste sin reentrenamiento.

Desventajas

- Mayor complejidad arquitectónica, lo puede dificultar su implementación.
- Uso de memoria más elevado que modelos más simples como YOLO.
- Dependencia de hardware especializado si se busca inferencia en tiempo real en dispositivos de baja potencia

1.2.5. Herramientas de visión artificial

- **OpenCV:** Es una biblioteca de código abierto muy demandada en la visión artificial, sus algoritmos son tratados para el procesamiento de imágenes, tiene aproximadamente 2500 algoritmos se alojan en su biblioteca, además, se mantiene en constante evolución debido a que tiene colaboración con empresa como Google, Microsoft, Toyota y otros. OpenCv tiene funciones importantes como detección de objetos, rostros y seguimiento de objetos. Desde su versión alfa en 1999, ha evolucionado y se utiliza ampliamente en diversas aplicaciones, siendo multiplataforma y desarrollada originalmente por Intel (OpenCV, 2020).
- **Python:** Lenguaje de programación muy popular aprendizaje automático y la ciencia de datos. Su popularidad se debe a su facilidad de aprendizaje, eficiencia y capacidad para ejecutarse en una variedad de plataformas, así como a su amplia gama de bibliotecas y a la capacidad de brindar recursos para el aprendizaje automático (Raschka et al., 2020). Python es popular entre los desarrolladores porque es gratuito, se integra bien con varios sistemas y contribuye al desarrollo de software rápido.
- **Google Colab:** Plataforma en línea que busca impulsar la educación y la investigación en aprendizaje automático. Esta herramienta acelera el proceso de entrenamiento de modelos al proporcionar recursos de GPU. Con su entorno basado en la nube, Colab facilita la implementación y ajuste de modelos de aprendizaje automático al simplificar la colaboración y la ejecución de código Python (Google Colab, 2024).
- **PyTorch:** Es una biblioteca de código abierto para inteligencia artificial y aprendizaje profundo desarrollada por Meta AI. Se ha consolidado como una de las herramientas más utilizadas en el campo del aprendizaje automático, debido a su enfoque basado en tensores dinámicos, su facilidad de implementación y su compatibilidad con aceleración en GPU. PyTorch permite la creación, entrenamiento e implementación de modelos de redes neuronales con una sintaxis más intuitiva y flexible en comparación con otras bibliotecas, lo que lo hace ideal tanto para investigación como para aplicaciones en producción (Pytorch.org., 2023).
- **Ultralytics:** Es una plataforma de visión por computadora especializada en el desarrollo y optimización de modelos de detección de objetos, segmentación, clasificación y

estimación de poses (Ultralytics, 2024). Se ha destacado principalmente por el desarrollo de la familia de modelos YOLO (You Only Look Once) y por su enfoque en la accesibilidad y facilidad de uso para entrenar e implementar modelos de aprendizaje profundo en diversas aplicaciones.

Desde su lanzamiento, Ultralytics ha evolucionado hacia una solución más versátil, permitiendo el uso de modelos avanzados como RT-DETR, combinando velocidad y precisión en detección de objetos en tiempo real. Su compatibilidad con PyTorch y su capacidad de optimización para GPU y entornos de alto rendimiento la han convertido en una opción popular en la industria y en la investigación.

Algunas características clave de Ultralytics incluyen:

- Facilidad de uso: Proporciona una API intuitiva para entrenar, validar y realizar inferencias con modelos de detección.
- Compatibilidad con modelos avanzados: Soporta modelos como YOLO, RT-DETR y SAM (Segment Anything Model).
- Optimización para hardware acelerado: Compatible con GPU, TPU y entornos HPC CEDIA.
- Flexibilidad y modularidad: Permite ajustes en hiperparámetros y configuraciones de entrenamiento de manera sencilla.
- Soporte para múltiples tareas: Detección de objetos, segmentación, clasificación y seguimiento en tiempo real.
- Comunidad y documentación extensa: Ofrece guías detalladas, notebooks de demostración y soporte en su foro y GitHub.

1.3. Sistemas embebidos

Un sistema embebido, también conocido como minicomputadora, combina software y hardware para el procesamiento de código. Estos sistemas, que son comunes en dispositivos móviles y otros, con frecuencia no tienen una interfaz de usuario visible, aunque algunos tienen interfaces gráficas complejas. Su tamaño pequeño y simplicidad lo hacen fácilmente integrados en dispositivos más grandes, programándolos para funciones específicas. En áreas como automóviles, electrodomésticos y dispositivos médicos, se destacan por su bajo consumo de energía y recursos limitados. Algunos incorporan sistemas operativos RTOS limitados, pero

pueden ejecutar programas en una variedad de lenguajes, como ensamblador, ANSI C, C++, Basic o Python (Rebound, 2022).

1.3.1. Comparativa de sistemas embebidos

A continuación, la tabla Tabla 1 muestra una comparativa de los dispositivos embebidos más utilizados:

Tabla 1

Comparativa de dispositivos embebidos más populares.

	RASPERRY PI 5	NVIDIA JETSON NANO
CPU	• Broadcom BCM2712 • Quad-core Arm Cortex-A76 a 2,4 GHz	• Procesador ARM A57 de cuatro núcleos a 1,43 GHz
GPU	• VideoCore VII • Soporta OpenGL ES 3.1 y Vulkan 1.2	• 128-core Maxwell
Almacenamiento	• Tarjeta microSD • Opción para unidades SSD M.2 (vía HAT opcional)	• Tarjeta microSD
RAM	• 4 / 8 GB LPDDR4X	• 4 GB 64-bit LPDDR4 25.6 GB/s
Conectividad	• Wi-Fi 5, Bluetooth 5.0 / BLE	• USB, GPIO
Sistema Operativo	• Linux, Windows 10 IOT	• Linux
Puertos	• 2 x micro HDMI (hasta 2 x 4K 60Hz simultáneas) • 2 x USB 3.0 • 2 x USB 2.0 • 1 x Gigabit Ethernet con PoE opcional • 2 x MIPI de 4 pistas • 1 x PCIe 2.0 x1 • 1 x GPIO 40 pines	• Cámara 1x MIPI CSI-2 DPHY • Connectivity Gigabit Ethernet, M.2 Key E • Display HDMI 2.0 • USB 4x USB 3.0, USB 2.0 Micro-B
Dimensiones	• 85 mm x 56 mm x	• 100 mm x 80 mm x 29

	17 mm	mm
Precio en Amazon sin costos de envío	• Raspberry Pi 5 (4 BG RAM): $88 • Raspberry Pi 5 (8 BG RAM): $120	• $149

Fuente:

1.3.2. Jetson Nano

Es una plataforma de inteligencia artificial (IA) creada por NVIDIA que está destinada a aplicaciones de visión por computadora y aprendizaje profundo. Una GPU Maxwell de 128 núcleos CUDA, un procesador de cuatro núcleos ARM Cortex-A57 y 4 GB de RAM forman parte de su arquitectura. Esta pequeña pero poderosa placa ha ganado relevancia en la inteligencia artificial porque proporciona a los desarrolladores un entorno de desarrollo accesible que les permite implementar modelos de aprendizaje profundo en dispositivos embebidos. Su bajo consumo de energía, la capacidad de ejecutar redes neuronales complejas y su importancia en aplicaciones de robótica, automatización y sistemas de visión son algunas de sus características destacadas (NVIDIA, 2024).

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).

1.4. Metodología KDD

1.4.1. KDD

KDD es el acrónimo de "Knowledge Discovery in Databases". Esta metodología tiende su practica en la minería de datos en diferentes campos, como negocios, investigación científica y salud (Fayyad, 1996).

El proceso KDD se lleva a cabo de forma secuencial y requiere que se complete una serie de pasos para automatizar la adquisición de conocimiento. Este proceso comienza con una gran cantidad de datos sin procesar y puede ser iterativo, lo que permite repetir los pasos según sea necesario hasta obtener la información deseada. Es importante destacar que el objetivo de este proceso es convertir datos crudos en conocimientos útiles y significativos. Este método es importante porque puede identificar patrones y relaciones ocultas en grandes conjuntos de datos, lo que ayuda a tomar decisiones inteligentes y generar valor a partir de la información disponible.

1.4.1.1. Etapas del proceso KDD

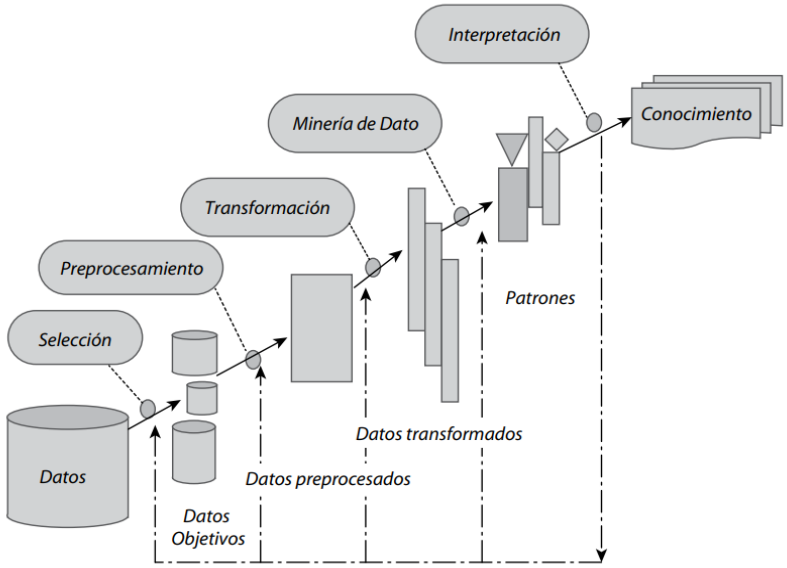
De acuerdo con los autores (Timarán et al., 2016), se abordan las distintas fases de la metodología KDD como se detalla en la Figura 9.

- **Etapa de selección:** Se crea un conjunto de datos objetivo después de identificar el conocimiento prioritario y las metas desde la perspectiva del usuario final. Este conjunto puede incluir una muestra representativa de la totalidad de los datos actualmente disponibles, y la elección se adapta a los objetivos comerciales particulares. Es fundamental tener en cuenta los requisitos del usuario final para garantizar que la selección de datos cumpla con las metas de KDD y contribuya significativamente al logro de resultados valiosos.
- **Etapa de preprocesamiento:** Este proceso establece las bases para un análisis de datos sólidos y precisos en las etapas posteriores del ciclo KDD. Además, se utilizan estrategias estadísticas como la moda, el mínimo y el máximo para reemplazar datos desconocidos y valores faltantes, lo que garantiza que los datos estén limpios y preparados para la fase de descubrimiento de patrones y conocimiento.

- **Etapa de transformación:** Corresponde a un proceso estratégico para identificar las características esenciales que mejor representan los datos utilizando técnicas de reducción como agregaciones, comprensión de datos, entre otras lo que permite una adaptación precisa a las características específicas de los datos en cada contexto de aplicación.
- **Etapa de minería de datos:** Implica el uso de algoritmos y técnicas para encontrar patrones, tendencias y conocimientos importantes en grandes conjuntos de datos. Esta etapa se enfoca en descubrir las complejas interacciones entre las variables y proporcionar datos útiles para la toma de decisiones. Además, contribuye a la generación de conocimiento en una variedad de campos, desde la investigación científica hasta el negocio, al buscar tanto patrones conocidos como descubrimientos inesperados.
- **Etapa de interpretación**: Muestra la visualización de patrones significativos en términos fáciles de entender para el usuario, además, de consolidarse como base crucial para su incorporación en otros sistemas con finos operativos o para su documentación y presentación a las partes interesadas. Este proceso no solo ayuda a verificar y resolver posibles conflictos con conocimientos previos, sino que también promueve la toma de decisiones informadas a partir de los patrones identificados.

Figura 9

Etapas del proceso KDD.



Fuente:

1.5. Trabajos Relacionados

(Krishna et al., 2022) propusieron, un marco de método conductual utilizando YOLO V5 (CNN) y Vision Transformers (ViT). Se uso dos Datasets, uno de la Universidad de Texas en Arlington (UTARLDD 640x480) y otros datos recopilados con una cámara DSLR (3840x2160) al que se le aplico Aumento de Datos para mejorar la precisión del modelo ViT. Se personaliza el modelo YoloV5 preentrenado que detecta la región ROI del rostro, y el modelo ViT se lo usa para realizar clasificación, donde la imagen se redimensiona y lo convierte en N parches y de cada parche se sacan las características para detectar la somnolencia. Se obtuvo una precisión de entrenamiento de 96.2% y validación de 97.4%. La falta de un mayor conjunto de datos con diferentes condiciones y que se encuentren etiquetados es una de las limitaciones relevantes.

(Hashemi et al., 2020) propusieron desarrollar un sistema para la detección de somnolencia en tiempo real, realiza la comparación con tres modelos: el primero es un modelo creado por los autores (FD-NN), el segundo un modelo basado en TL-VGG16 con características da bajo y alto nivel, y la tercera un modelo TL-VGG19. Se usa el algoritmo de Viola y Jones para detectar la caja del rostro y puntos faciales para el recorte de la región ROI. Los modelos

fueron entrenados con dos Datasets, el primero es el Dataset ZJU (Pan et al., 2007), y el segundo el Dataset creada por los autores, cabe recalcar que después de la normalización de los datos, la resolución de la imagen es de 24x24. Para los tres modelos TL-VGG19, TL-VGG16 y FD-NN, se obtuvo una precisión de 95%, 95.45% y 98.15% respectivamente. Las condiciones ambientales, el tamaño del Dataset, requisitos elevados del hardware, son las limitaciones presentes en el trabajo.

(Fernández, 2022) propuso, desarrollar un sistema para la detección de somnolencia usando 2 modelos de Redes Neuronales Convolucionales (CNN). Los modelos fueron entrenados con un Dataset de data-flair con la cantidad de 2900 imágenes al que se le aplico aumento de datos para mejorar la precisión de los modelos, los modelos tienen una entrada de 24x24, una vez entrenado se realizó la integración del sistema el cual tiene una precisión del 96% que demuestra una fiabilidad bastante alta para la detección de somnolencia. El sistema tiene ciertas limitaciones como el que la precisión de la detección puede verse afectado debido a la variación de la luminosidad, y que solo puede detectar a un individuo en el vehículo.

(Qingyun et al., 2022) propuso un sistema de detección de objetos que emplea imágenes por teledetección, fusionando las arquitecturas CNN y Transformer. Para la identificación de objetos, se optó por la arquitectura CNN, ya que ha demostrado brindar resultados destacados en la extracción de características de imágenes. Por otro lado, se incorporó la arquitectura Transformer para establecer relaciones en imágenes por teledetección a larga distancia. Para validar los resultados, se llevó a cabo una comparación con la arquitectura YOLO v3, la cual tiende a pasar por alto detalles pequeños en una imagen, a diferencia de la propuesta T-TRD-DA, que logra un rendimiento de detección sobresaliente con un mAP de 0.879 en todas las categorías. No obstante, se identifica una limitación en la velocidad de inferencia, atribuida al elevado costo computacional asociado a la implementación de un Transformer.

(Lakhani, 2022) propuso una estrategia pionera para abordar la conducción distraída y somnolienta mediante la atención espaciotemporal con transformadores de visión. Utilizando la arquitectura Swin Transformer, se entrenaron modelos separados para detectar la somnolencia y la distracción, utilizando conjuntos de datos de conducción con sueño de la Universidad Nacional Tsing-Hua (NTHU-DDD) y para distracción se utilizó el conjunto de datos de

monitoreo del conductor (DMD) .Aunque el modelo de somnolencia alcanzó una precisión del 44%, se observó sobreajuste y rendimiento deficiente, destacando la necesidad de un conjunto de datos más amplio y ajustes en la arquitectura. El modelo de distracción superó las arquitecturas de última generación, logrando una impresionante precisión del 97.5%, subrayando la idoneidad de los transformadores para la detección de conducción no apta. Las limitaciones identificadas incluyen la falta de parámetros cuantificables en la arquitectura y la importancia de futuras investigaciones con modelos más avanzados, como TokenLearner.

(Khoramdel et al., 2024) propusieron un método innovador llamado YOLO-Former, la cual fusiona la eficacia del Transformer y YOLOv4 para desarrollar un sistema altamente preciso y eficiente en la detección de objetos. Este enfoque utiliza módulos de atención convolucional y Transformer para mejorar la capacidad de generalización del modelo, elevando así su precisión en la identificación de objetos. La arquitectura del YOLO-Former se basa en YOLOv4 y consta de tres subredes: el "backbone", el "neck" y el "head". El "backbone", conocido como CPS-Darknet-53, extrae características de la imagen de entrada, generando salidas en tres niveles distintos. Utilizaron el conjunto de datos Pascal VOC, el YOLO-Former en el cual sus resultados alcanzaron una precisión media promedio (mAP) del 85.76%, manteniendo una rápida velocidad de predicción de 10.85 cuadros por segundo. Aunque el artículo no especifica las limitaciones del YOLO-Former, es crucial considerar que, como cualquier enfoque de investigación, podría tener áreas para mejora o limitaciones no abordadas explícitamente.

(Adewopo et al., 2024) propusieron la elaboración de un conjunto de datos completo destinado a la detección de accidentes de tráfico mediante sistemas de visión por computadora y reconocimiento de acciones. Su enfoque implica la implementación del modelo I3D-CONVLSTM2D, que fusiona redes neuronales convolucionales tridimensionales (I3D) con redes neuronales de memoria convolucional de larga duración (CONVLSTM2D). A pesar de la ausencia de resultados específicos en la detección de accidentes de tráfico, el documento se centra en detallar la creación del conjunto de datos y en describir la arquitectura de la red neuronal propuesta. Entre las limitaciones se destacan la escasez de conjuntos de datos anotados, la variabilidad en las condiciones ambientales y preocupaciones éticas y de privacidad asociadas con la implementación de estos sistemas.

Capítulo 2

Desarrollo

En este capítulo se detallarán las etapas del desarrollo del sistema embebido, utilizando la metodología KDD (Knowledge Discovery in Databases). Estas etapas abarcan desde la recolección y tratamiento de imágenes hasta la implementación de la red neuronal RT-DETR en el sistema embebido.

Con la combinación de estas etapas y el uso de la metodología KDD, se pretende desarrollar un sistema embebido robusto y eficiente, capaz de cumplir con los objetivos planteados y ofrecer resultados confiables y precisos en su aplicación.

2.1. Recopilación y tratamiento de imágenes

Contar con un dataset adecuado es fundamental para el entrenamiento de un modelo de detección de somnolencia, ya que la calidad y la diversidad de los datos influyen directamente en la precisión y eficacia del modelo. Un conjunto de datos bien etiquetado y representativo permite que el modelo identifique patrones relevantes asociados con la somnolencia, lo que mejora significativamente su capacidad de generalización en escenarios reales.

En Deep Learning, los datos son el núcleo del éxito de cualquier proyecto: datos de mala calidad generan modelos ineficaces y resultados poco fiables, mientras que datos de alta calidad permiten construir modelos sólidos y realizar predicciones precisas. Sin embargo, obtener datos de calidad no es un proceso simple; no se trata únicamente de acumular grandes volúmenes de información, sino de garantizar que estos datos sean relevantes, representativos y estén correctamente organizados.

2.1.1. Selección

Para llevar a cabo el entrenamiento del modelo RT-DETR dedicado a la detección de somnolencia, se emplearán conjuntos de datos proporcionados por el tutor de tesis. Estos datasets no solo contienen las imágenes necesarias, sino también sus respectivos archivos ".pts", los cuales comparten el mismo nombre que las imágenes y almacenan la información de los puntos faciales correspondientes.

Los archivos “.pts” son fundamentales en tareas de regresión, ya que permiten entrenar modelos que predicen con precisión la ubicación exacta de puntos faciales específicos, como los

landmarks oculares. Sin embargo, en el caso de detección de objetos, como se plantea con RT-DETR, no es necesario predecir cada punto facial individualmente. En su lugar, se emplean cajas delimitadoras (bounding boxes) para ubicar y etiquetar regiones específicas de interés, como la zona de los ojos, clasificándolas como (0 para ojos abiertos y 1 para ojos cerrados). Este enfoque permite optimizar la detección de patrones de somnolencia sin depender de la localización exacta de cada punto facial.

Esta estructura organizada de datos facilita la asociación entre las imágenes y sus etiquetas, asegurando que el modelo aprenda a identificar y localizar de manera eficiente las señales visuales asociadas con la somnolencia.

2.1.1.1. Datasets proporcionados por el tutor

Los datasets seleccionados abarcan una amplia variedad de imágenes que incluyen sujetos de ambos géneros, diferentes edades, etnias y colores de piel, capturados en diversos escenarios y contextos. Estas imágenes fueron tomadas en condiciones diurnas y nocturnas, en distintos lugares y situaciones, lo que representa un desafío significativo para la extracción de características y permite al modelo realizar un análisis más exhaustivo de la somnolencia. Los datasets fueron elegidos por su alta calidad y relevancia para el problema de investigación. A continuación, se presenta una descripción de los datasets seleccionados en la Tabla 2.

Tabla 2

Datasets seleccionados.

Dataset	**Cantidad de Imágenes**	**Particularidades**
300VW	2442	Imágenes en condiciones diurnas y nocturnas, capturadas desde diversos ángulos y con distintas condiciones de iluminación.
300W	1760	Rostros de revistas y celebridades.
afw	570	Imágenes con fondos variados y múltiples personas, etiquetadas tanto en estados de sueño como de alerta.

CEW	560	Rostros de adultos y niños con ojos cerrados.
COFW	432	Imágenes de celebridades en diferentes resoluciones.
frgc	4950	Rostros de adultos.
Helen	8320	Imágenes de adultos y niños; algunas contienen múltiples rostros, pero solo uno está etiquetado.
ibug	568	Rostros de celebridades; incluye aumento de datos con la colocación de gafas.
lfpw	3434	Imágenes de adultos; incluye aumento de datos mediante la colocación de gafas.
Menpo2D	3979	Imágenes de adultos y niños, capturadas desde diferentes ángulos.
ownNIR	1876	Imágenes de personas conduciendo en condiciones diurnas y nocturnas, con ojos abiertos y cerrados.
xm2vts	2360	Rostros de personas adultas con ojos abiertos.
Total	30689	

Fuente: Propia

Aunque los datasets proporcionan una amplia variedad de rostros, hay una mayor cantidad de imágenes con los ojos abiertos, lo que podría generar un sesgo en el modelo. Para evitar que el modelo aprenda solo a identificar ojos abiertos, sería importante incluir un dataset adicional que balancee mejor las imágenes de ojos abiertos y cerrados.

Asimismo, si bien algunos datasets incluyen imágenes de personas conduciendo, la mayoría se enfocan solo en los rostros, por lo que sería útil incorporar más imágenes de conducción en entornos reales. Esto enriquecería el análisis y permitiría una evaluación más contextualizada del comportamiento del conductor y su nivel de somnolencia.

2.1.1.2. Creación de propio dataset

Para complementar los datasets proporcionados por el tutor y garantizar una mayor representatividad de los datos en el entrenamiento del modelo, se procedió a la creación de un dataset propio enfocado específicamente en la detección de somnolencia en conductores. A diferencia de los datasets previamente mencionados, que en muchos casos incluyen imágenes capturadas en entornos simulados o controlados, este nuevo conjunto de datos se generó utilizando capturas de fotogramas de video tomados en vehículos reales, pero variados, para reflejar diferentes condiciones y escenarios de conducción.

Para la recolección de los fotogramas, se utilizó una Raspberry Pi en conjunto con una cámara OV5647 equipada con infrarrojos, lo que permitió capturar datos tanto en condiciones diurnas como nocturnas. Las imágenes tienen una resolución de 640x480 píxeles y se grabaron en escala de grises, lo que optimiza el enfoque en las características relevantes para la detección de somnolencia. Además, la cámara operó con una tasa de refresco promedio de 25 fotogramas por segundo, asegurando la captura de suficientes detalles para el análisis y entrenamiento del modelo.

- **Configuración del Hardware:**
    - Raspberry Pi 4: Se utilizó un Raspberry Pi 4 debido a su capacidad de procesamiento y flexibilidad.
    - Cámara: Para la captura de fotogramas, se empleó esta cámara equipada con infrarrojos, lo que hace eficiente para captura en condiciones nocturnas.
    - Pantalla LCD: Se utilizó para visualizar la imagen capturada por la cámara, lo que permite ajustar el enfoque y la dirección de la misma.
    - Laptop Ideapad 330s: Se usó para manipular el Rapsberry Pi y almacenar los datos obtenidos después de grabar los videos.
    - Cable HDMI: Utilizado para conectar la laptop al Raspberry Pi.
    - Cable USB a USB c: Conectó el Raspberry Pi a la pantalla LCD.
    - Fuente de poder: Suministro de energía necesaria para el funcionamiento de todos los dispositivos.
- **Proceso de Captura de Imágenes:**

- Participantes: Se reclutaron 20 personas de diversas edades y géneros con el objetivo de garantizar un dataset completamente diversificado, que permita al modelo aprender de una amplia variedad de características faciales y contextos.
- Escenarios: El dataset incluye 3 clases de escenarios distintos que cubren tanto condiciones diurnas como nocturnas, para asegurar la variabilidad en las imágenes:
  - Bareface-. Los videos se grabaron tanto de día como de noche, con los participantes sin gafas ni lentes, proporcionando imágenes claras del rostro sin obstrucciones.
  - Glasses-. Los videos fueron grabados tanto en el día como en la noche, con los participantes usando gafas, lo que introduce variabilidad en las imágenes debido a la presencia de lentes.
  - Sunglasses-. Los videos se grabaron tanto de día como de noche, con los participantes usando gafas de sol. Aunque las gafas de sol pueden cubrir parcialmente los ojos, estas no tienen lupas concentradas, lo que permite distinguir y ver los ojos.

- **Sesiones de Captura:**

  Se grabaron videos de 1 minuto para cada participante, simulando movimientos de conducción real. Durante la grabación, los participantes realizaron diferentes movimientos, como girar el volante, ajustar su postura y cambiar la distancia entre ellos y la cámara, con el objetivo de generar una variedad de ángulos y perspectivas. Además, se buscó capturar diversos estados de ánimo y variaciones en la expresión facial, creando un ambiente que simula condiciones de conducción reales.

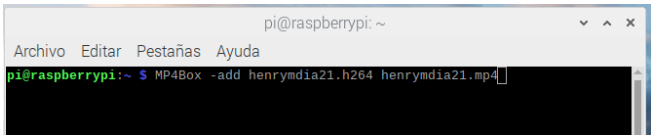
  Las grabaciones se realizaron con 20 personas durante el día y la noche utilizando la cámara OV5647, la cual está conectada directamente a la Raspberry Pi. El sistema de la Raspberry Pi gestiona la grabación en el formato h264, seleccionado por su alta eficiencia de compresión y baja latencia, lo que permite capturar videos de manera óptima para su procesamiento en tiempo real. Sin embargo, debido a que el formato h264 no es compatible de forma nativa con el sistema Windows fue necesario convertir los videos a mp4 para su posterior análisis y etiquetado de datos.

  Para convertir el formato de video, se utilizó el siguiente comando en la terminal de la Raspberry Pi *"MP4Box -add henrymdia21.h264 henrymdia21.mp4"* tal como se muestra

en la Figura 10, este proceso permitió transformar los archivos de video para garantizar su compatibilidad con el sistema operativo. En la Figura 11, se presentan ejemplos de los videos capturados en distintas condiciones de iluminación, reflejando las variaciones entre las grabaciones realizadas durante el día y la noche.

Figura 10

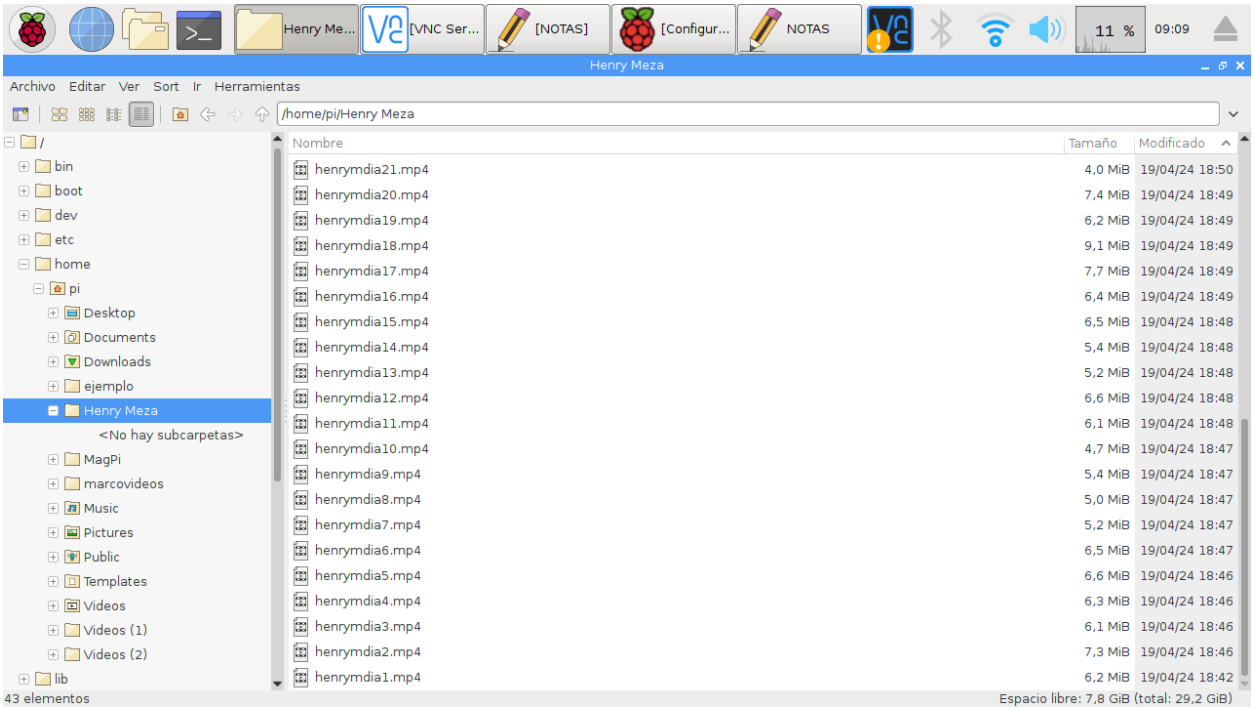
Conversión de formato h264 a mp4.



Fuente: Elaboración propia

Figura 11

Ejemplo de videos capturados.



Fuente: Elaboración propia

2.1.2. Preprocesamiento

2.1.2.1. Extracción de frames

Para extraer los frames de cada video, se implementó un script en Python, el cual fue ejecutado en Google Colab. Google Colab ofrece un entorno similar a Jupyter Notebook, permitiendo ejecutar código de manera interactiva con acceso a recursos computacionales en la nube. Aunque el plan gratuito presenta ciertas limitaciones en cuanto a la disponibilidad de recursos para entrenamiento, sigue siendo una opción eficiente para la extracción de frames en equipos con capacidades limitadas. En la Figura 12, se muestra el código implementado para este proceso.

Figura 12

Código para extraer los frames del video.

```python
# Creamos un método para extraer los frames del video
def extraerFrames(video_path, output_folder):
  # Abre el video
  capture = cv2.VideoCapture(video_path)

  # Verifica si el video se abrio correctamente
  if not capture.isOpened():
    print('Error al abrir el video.')
    return

  # Crea el directorio de salida si no existe
  os.makedirs(output_folder, exist_ok = True)

  # Inicializa el contador de frames
  frame_count = 0

  # Itera sobre los frames del video
  while True:
    ret, frame = capture.read()

    # Verifica si se alcanzó el final del video
    if not ret:
      break

    # Redimensiona el frame a 640x480
    frame_resized = cv2.resize(frame, (640, 480))

    nombre = os.path.basename(output_folder)
    # Guarda el frame como una imagen jpg en el directorio de salida
    frame_name = f"{nombre}_{frame_count:04d}.jpg"
    frame_path = os.path.join(output_folder, frame_name)
    cv2.imwrite(frame_path, frame_resized)

    # Incrementa el contador de frames
    frame_count += 1

  # Libera el objeto capture
  capture.release()

  # Imprime un mensaje de completado
  print("¡Frames completados!")
```

Fuente: Elaboración propia

2.1.2.2. Selección de frames

En esta etapa, se seleccionaron los frames de cada video priorizando aquellos de mejor calidad y asegurando una mayor variabilidad entre ellos. Se consideraron factores como los gestos del conductor, la posición de la cabeza, la presencia de accesorios (gafas, lentes) y las expresiones faciales. Además, se prestó especial atención a la captura de imágenes donde los ojos se encuentren tanto abiertos como cerrados, con el fin de proporcionar un conjunto de datos balanceado y representativo. Este proceso tiene como objetivo mejorar la capacidad del modelo para identificar patrones asociados a la somnolencia.

En la Figura 13, se muestra un ejemplo de los frames seleccionados. La selección de los frames por cada persona se realizó bajo el siguiente criterio, el cual se detalla en la Tabla 2:

- **Durante el día:** 2 imágenes del rostro sin accesorios, 2 con lentes y 2 con gafas de sol, tanto con los ojos abiertos como cerrados.
- **Durante la noche:** Se replico la misma metodología.

Figura 13

Frames seleccionados de un video.

Fuente: Elaboración propia

Tabla 3

Criterio de selección de frames para cada persona.

Condición	Solo el rostro	Con lentes	Con gafas	Total, de frames
Día – Ojos Abiertos				
Día – Ojos Cerrados	2	2	2	6
Noche - Ojos Abiertos	2	2	2	6
Noche - Ojos Cerrados	2	2	2	6
Total, de frames por persona	**8**	**8**	**8**	**24**

Fuente: Elaboración propia

2.2. Etiquetación de imágenes

2.2.1. Transformación

En esta etapa, se procesaron y ajustaron los datos del conjunto de imágenes para su uso en el entrenamiento del modelo RT-DETR. La transformación de los datos incluyó la adaptación de las etiquetas y estructuras de anotación, asegurando que fueran compatibles con el enfoque más adecuado para la detección de somnolencia.

Inicialmente, se intentó un enfoque basado en regresión, en el que el modelo predecía directamente los 12 landmarks de los ojos. Posteriormente, se exploró un enfoque de clasificación, etiquetando las imágenes como ojos abiertos o cerrados. Finalmente, se adoptó un esquema basado en detección de objetos, utilizando cajas delimitadoras (bounding boxes) generadas a partir de los landmarks para localizar los ojos en cada imagen.

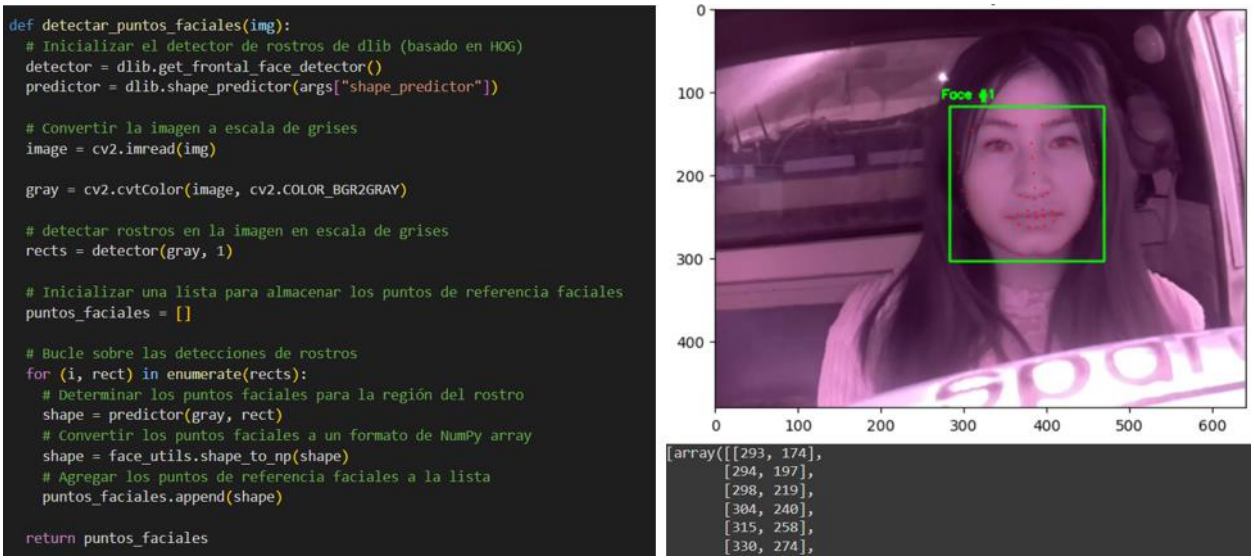
2.2.2.1. Etiquetado para Regresión

El objetivo era entrenar un modelo para predecir directamente las coordenadas de los 12 landmarks correspondientes a los ojos, es decir, 6 puntos por cada ojo. Para ello, se implementó un proceso de etiquetado basado en regresión, asegurando la precisión de los puntos faciales que posteriormente serían utilizados en el entrenamiento del modelo RT-DETR.

- **Software y Herramientas Utilizadas:** Para la detección de landmarks faciales, se utilizó el modelo preentrenado "shape_predictor_68_face_landmarks.dat" de la biblioteca Dlib, el cual permite identificar 68 puntos faciales como lo muestra en la Figura 14. La implementación del modelo y su ejecución se realizaron en Google Colab, lo que permitió procesar grandes volúmenes de datos sin depender de recursos locales.

Figura 14

Identificación de 68 landmarks faciales.

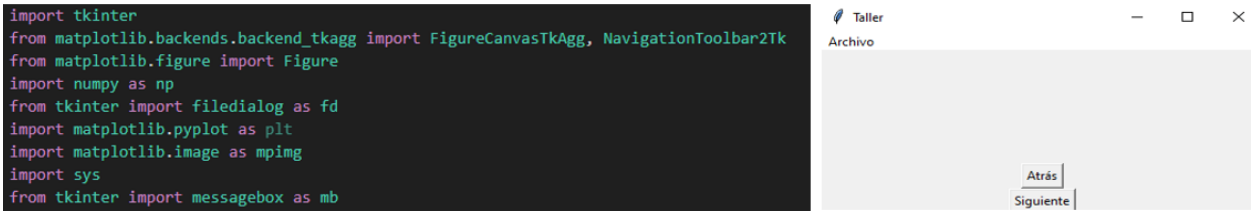


Fuente: Elaboración propia

Además, se empleó una aplicación desarrollada en Python con Tkinter para la interfaz gráfica y Matplotlib para la visualización y edición de los landmarks detectados. Esta herramienta permitió corregir manualmente los puntos en caso de que la predicción automática no fuera precisa. En la Figura 15, se muestra la importación de las bibliotecas y la aplicación en funcionamiento.

Figura 15

Importación de bibliotecas (izquierda), aplicación(derecha).



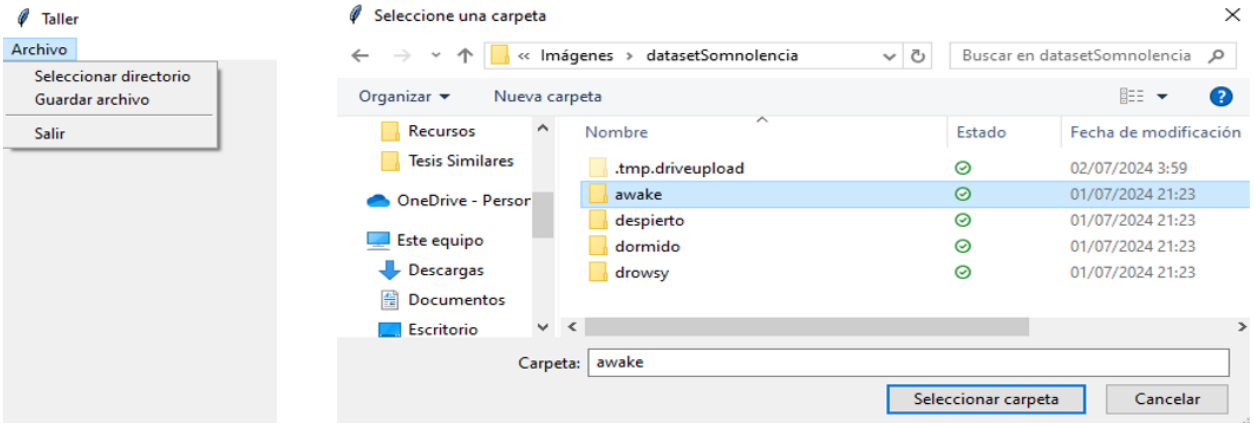
Fuente: Elaboración propia

- **Proceso de Etiquetado:**
  - Cargar las imágenes en la aplicación de anotación (Figura 16).
  - Ejecutar aplicación de anotación desarrollada en Python para visualización y edición de landmarks.

- Identificar los 12 landmarks correspondientes a los ojos.
- Revisar y corregir manualmente los landmarks en caso de detecciones incorrectas, utilizando la herramienta de anotación (Figura 17).
- Guardar los cambios en el archivo ".pts" en caso de haber realizado modificaciones.

Figura 16

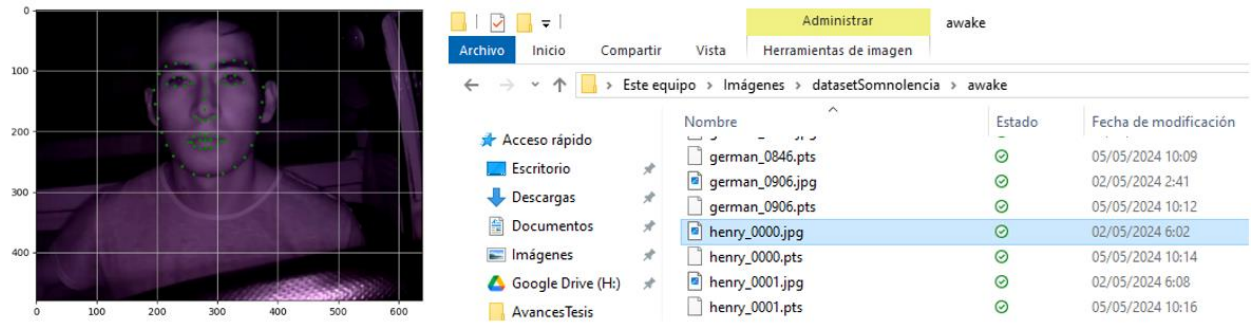
Selección de carpeta de imágenes.



Fuente: Elaboración propia

Figura 17

Revisión manual de landmarks.



Fuente: Elaboración propia

Este proceso se repitió para todas las imágenes del conjunto de datos, garantizando un etiquetado uniforme y preciso. Es importante destacar que los archivos ".pts" almacenan la información clave sobre la posición de los landmarks, lo que facilita el aprendizaje del modelo en la detección de somnolencia.

Sin embargo, la implementación de los landmarks para crear un modelo de regresión no se llevó a cabo. Tras realizar diversas pruebas, se identificó que la arquitectura RT-DETR no estaba optimizada para este tipo de tarea. Esto llevó a la necesidad de explorar enfoques alternativos, como la clasificación y la detección de objetos.

2.2.2.2. Etiquetado para Clasificación

Se evaluó la posibilidad de adaptar la red a un enfoque mixto de regresión y clasificación, separando las imágenes en dos clases:

- Ojos abiertos (label: 0)
- Ojos cerrados: (label: 1)

Para facilitar la clasificación, las imágenes fueron organizadas en carpetas separadas, donde cada carpeta contenía todas las imágenes correspondientes a su respectiva clase junto con sus archivos ".pts". Este procedimiento garantizó que cada imagen tuviera su archivo asociado, evitando problemas al momento de asignar la etiqueta label a cada archivo.

La Figura 18 muestra el código utilizado para agregar las etiquetas label a cada archivo ".pts", mientras que la Figura 19 ilustra cómo se insertó la etiqueta dentro del archivo ".pts".

Figura 18

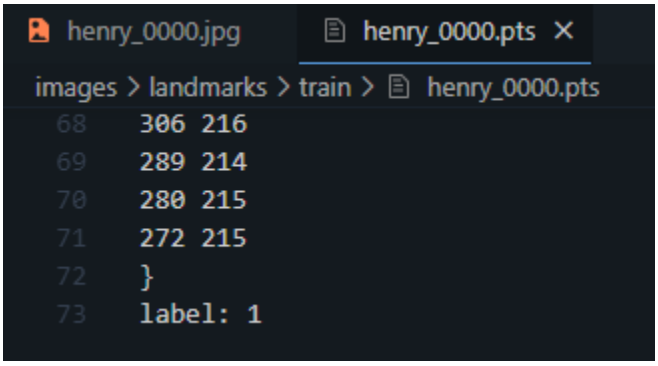
Agregar etiquetas label.

```
def add_label_after_braces(directory, label):
    for filename in os.listdir(directory):
        if filename.endswith('.pts'):
            filepath = os.path.join(directory, filename)
            try:
                with open(filepath, 'r') as file:
                    content = file.readlines()
                # Verificar si la última línea es una llave de cierre
                if content[-1].strip() == '}':
                    content[-1] = '}\n'  # Asegurar que haya un salto de línea después de la llave
                    content.append(f'label: {label}\n')  # Añadir la etiqueta en una nueva línea
                else:
                    # Si no, corregir la última línea y agregar la llave y la etiqueta correctamente
                    content[-1] = content[-1].rstrip() + '\n'
                    content.append('}\n')
                    content.append(f'label: {label}\n')
                # Escribe el contenido modificado de vuelta al archivo
                with open(filepath, 'w') as file:
                    file.writelines(content)
            except PermissionError:
                print(f"No se pudo abrir el archivo: {filepath} - Compruebe los permisos.")
```

Fuente: Elaboración propia

Figura 19

Creación de etiqueta label.



Fuente: Elaboración propia

El objetivo de esta estrategia era que el modelo aprendiera simultáneamente a detectar los landmarks y a clasificar el estado de los ojos. Sin embargo, este enfoque generó confusión en el modelo en lugar de mejorar la precisión en la detección. Como resultado, se optó por adaptar la red a detección de objetos, lo que permitió estructurar mejor la tarea y obtener un desempeño más adecuado utilizando la arquitectura RT-DETR.

2.2.2.3. Etiquetado para Detección de Objetos

Dado que la arquitectura RT-DETR no se adaptó correctamente ni a la regresión, ni a una combinación entre regresión y clasificación, se decidió reformular el problema como una tarea de detección de objetos.

Para ello, se reutilizaron los archivos .”pts”, que originalmente contenían los landmarks faciales, transformándolos en cajas delimitadoras (bounding boxes). Esto permitió definir regiones de interés alrededor de los ojos, en lugar de trabajar con puntos individuales.

- **Extracción de Coordenadas para las Cajas Delimitadoras:** Inicialmente, se realizó la extracción de coordenadas a partir de los landmarks faciales, con el objetivo de delimitar correctamente la región de los ojos en cada imagen. Para ello, se calcularon los valores mínimos y máximos de los puntos correspondientes a cada ojo, generando así los parámetros necesarios para definir las cajas delimitadoras. Además, se aplicó un ajuste de padding, asegurando que las cajas cubrieran correctamente la región ocular sin excluir información relevante. La Figura 20 muestra el código utilizado para este proceso.

Figura 20

Extracción de bounding boxes.

```python
def get_bounding_box(points, padding, state = False):
    if isinstance(points, tf.Tensor):
        points = points.numpy()
    if points.size < 2:
        print(f"🚨 Error: El array de puntos tiene tamaño insuficiente: {points.shape}")
        return 0, 0, 0, 0  # Devolver un bounding box vacío
    points = np.reshape(points, (-1, 2))
    x_min, y_min = np.min(points, axis=0)
    x_max, y_max = np.max(points, axis=0)
    if not state:
        return int(x_min - padding), int(y_min - padding), int(x_max + padding), int(y_max + padding)
    else:
        return int(x_min - padding), int(y_min - padding), int(x_max + padding), int(y_max + ((padding*2)-4))

def extr_reg_of_each_eye(landmarks, state, padding=1):
    """Extrae las cajas delimitadoras de los ojos a partir de los landmarks."""
    landmarks = landmarks.reshape(-1, 2)
    left_eye_points = landmarks[:6]  # Puntos 36-41
    right_eye_points = landmarks[6:]  # Puntos 42-47
    left_eye_box = get_bounding_box(left_eye_points, padding, state)
    right_eye_box = get_bounding_box(right_eye_points, padding, state)
    return left_eye_box, right_eye_box

def extr_reg_of_each_eye_File(filepath, new_size = 128, state = False):
    if state:
        padding = 7
    else:
        padding = 2
    _, landmarks, label = redi.redim_img_and_land_File(filepath, new_size)
    left_eye_box, right_eye_box = extr_reg_of_each_eye(landmarks, state, padding)
    return left_eye_box, right_eye_box, label, new_size
```

Fuente: Elaboración propia

Este paso únicamente extrae las coordenadas; sin embargo, los datos aún no están en un formato compatible con RT-DETR, por lo que es necesario un procesamiento adicional para convertirlas en un archivo adecuado para la carga en el modelo.

- **Conversión a Formato Compatible con RT-DETR:** Una vez obtenidas las coordenadas de las cajas delimitadoras, se procede a estructurarlas en un formato adecuado para RT-DETR. Este modelo requiere que cada objeto detectado esté representado en un archivo con la siguiente estructura:

$$clase, x_{centro}, y_{centro}, ancho, alto$$

Donde:

- clase - indica si el objeto es un ojo abierto (0) o cerrado (1).
- x_{centro}, y_{centro} - coordenadas normalizadas del centro de la caja.

- ancho, alto – dimensiones normalizadas de la caja.

A diferencia de enfoques tradicionales que utilizan herramientas como Roboflow para la etiquetación de datos, en este caso se aprovechó la información de los archivos ".pts" generados previamente. Esto permitió una conversión automática de landmarks a bounding boxes sin necesidad de etiquetar manualmente cada imagen, reduciendo significativamente el tiempo de procesamiento.

En la Figura 21, se muestra el proceso implementado para generar los datos en formato .txt. Este archivo contiene la información estructurada requerida por RT-DETR, indicando claramente la clase del objeto detectado y las coordenadas normalizadas del centro, ancho y alto de cada caja delimitadora. En la Figura 22, se presenta un ejemplo del resultado final obtenido en los archivos generados en formato .txt.

Figura 21

Creación de datos para RT-DETR.

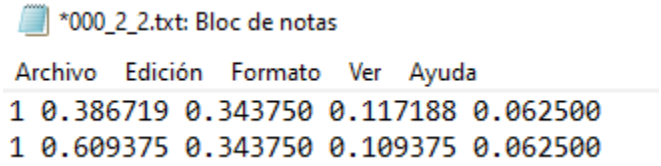
```python
def create_bounding_boxes(output_dir, name_image, left_eye_box, right_eye_box, label, new_size):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    txt_filename = os.path.join(output_dir, f'{os.path.splitext(name_image)[0]}.txt')
    with open(txt_filename, 'w') as file:
        for eye_box in [left_eye_box, right_eye_box]:
            x_min, y_min, x_max, y_max = eye_box
            x_center, y_center, width, height = convert_to_yolo_format(x_min, y_min, x_max, y_max, new_size, new_size)
            file.write(f'{label} {x_center:.6f} {y_center:.6f} {width:.6f} {height:.6f}\n')
    print(f"Archivo creado o reemplazado: {txt_filename}")

def create_txt(directory, output_dir):
    file_paths = [os.path.join(directory, f) for f in os.listdir(directory) if f.endswith('.jpg')]
    total_images = len(file_paths)
    for i, file_path in enumerate(file_paths):
        name_image = os.path.basename(file_path)
        left_eye_box, right_eye_box, label, new_size = extr_reg_of_each_eye_File(file_path, 128, True)
        print(f"Procesando {i+1}/{total_images}: {name_image}")
        create_bounding_boxes(output_dir, name_image, left_eye_box, right_eye_box, label, new_size)
```

Fuente: Elaboración propia

Figura 22

Bounding boxes de ojos cerrados.



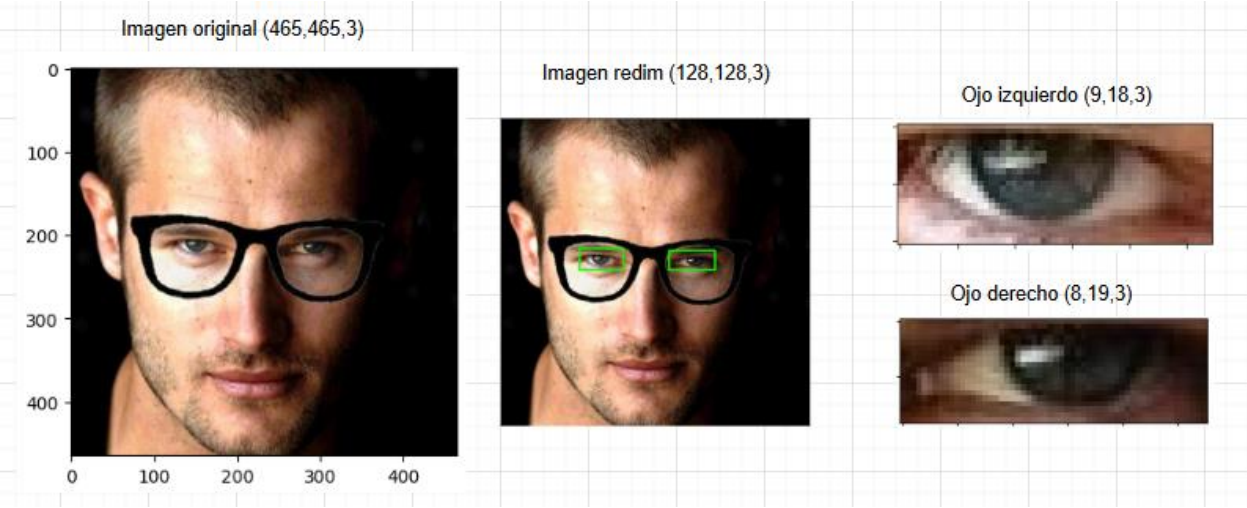
Fuente: Elaboración propia

- **Visualización de datos etiquetados:** Para asegurar la correcta interpretación de los datos por parte del modelo RT-DETR, se realizaron verificaciones visuales en varias etapas, como se muestra en la Figura 23. Inicialmente se parte de la imagen original, la cual puede tener diferentes dimensiones dependiendo de la fuente de captura. Luego, esta imagen es redimensionada a 128x128 píxeles, manteniendo los 3 canales RGB, dado que este es el formato requerido por la arquitectura del modelo RT-DETR.

  Posteriormente, sobre la imagen redimensionada, se realiza la detección de los ojos mediante cajas delimitadoras. Finalmente, se extraen las regiones de interés (ojos izquierdo y derecho) para su análisis individual, asegurando así que las cajas sean precisas y estén correctamente posicionadas.

Figura 23

Imagen con ojos etiquetados.



Fuente: Elaboración propia

2.2.2.4. Balanceo y desbalanceo de datos

Durante la etapa de transformación, se crearon dos versiones del dataset: una balanceada y otra desbalanceada, con el objetivo de evaluar el impacto de la distribución de clases en el desempeño del modelo RT-DETR.

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 4, se muestran la cantidad de datos que se tomó de cada Dataset.

Tabla 4

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 aumento 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 5, muestra la cantidad de datos que se tomó de cada Dataset.

Tabla 5

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 ‘Datasets’, dentro de la carpeta principal se crearon tres carpetas, uno de train, valid y test, donde se almacenaron las imágenes y sus respectivos archivos “.*txt*”, con una distribución de datos del 80%, 10% y 10% respectivamente.

2.3. Entrenamiento y pruebas del modelo RT-DETR

2.3.1. Minería de datos

En esta etapa se desarrolla la implementación y el entrenamiento de la red neuronal RT-DETR (Real-Time Detection Transformer), utilizando la biblioteca Ultralytics, basada en PyTorch. Ultralytics proporciona una API optimizada para tareas de detección de objetos en tiempo real, permitiendo realizar entrenamiento, validación, predicción, exportación, rastreo (tracking) y evaluación comparativa (benchmarking). Gracias a su estructura modular y

configurable, facilita el desarrollo de modelos de alto rendimiento con capacidad de ajuste según las necesidades del proyecto.

2.3.1.1. Cargar del modelo

En este proyecto, se empleó el modelo preentrenado RT-DETR-L, el cual ha sido previamente ajustado en el dataset COCO val2017, logrando 53.0% AP y una velocidad de 114 FPS en una GPU T4. Si bien existe una variante más potente, RT-DETR-X, que alcanza 54.8% AP, su velocidad de inferencia es menor (74 FPS en la misma GPU), por lo que se optó por RT-DETR-L al ofrecer un balance óptimo entre precisión y eficiencia para su implementación en tiempo real.

La Figura 24 muestra la carga del modelo preentrenado RT-DETR-L a través de la API de Ultralytics.

Figura 24

Carga del modelo preentrenado RTDETR-L.

```
# importar el modelo RTDETR
from ultralytics import RTDETR
# cargar el modelo RTDETR
model = RTDETR('rtdetr-l.pt')
✓ 0.6s
```

Fuente: Elaboración propia

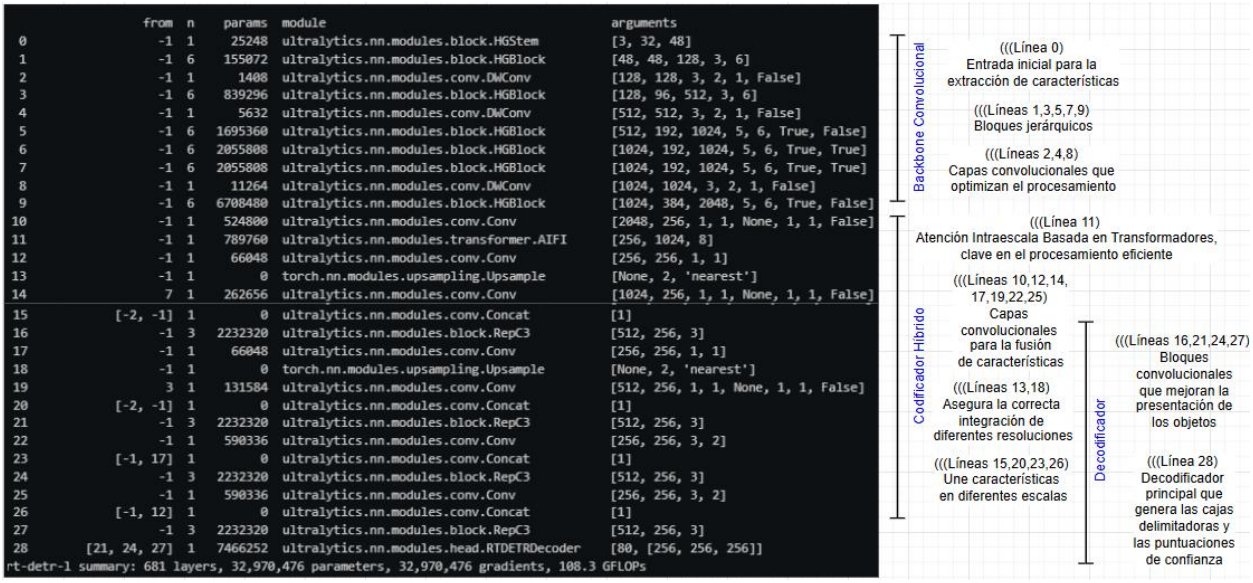
2.3.1.2. Visualización de la arquitectura del modelo preentrenado RT-DETR

Una vez cargado el modelo, se procedió a inspeccionar su arquitectura para verificar la estructura de capas y la cantidad de parámetros entrenables. En la Figura 25 se muestra un resumen del modelo, donde se pueden observar sus componentes principales:

- **Backbone convolucional:** Para la extracción de características multiescala.
- **Codificador híbrido:** Que transforma las características en una secuencia de atención eficiente.
- **Decodificador con cabezas de predicción:** que genera las cajas delimitadoras y sus respectivas puntuaciones de confianza.

Figura 25

Arquitectura del modelo preentrenado "rtdetr-l".



	from	n	params	module	arguments
0	-1	1	25248	ultralytics.nn.modules.block.HGStem	[3, 32, 48]
1	-1	6	155072	ultralytics.nn.modules.block.HGBlock	[48, 48, 128, 3, 6]
2	-1	1	1408	ultralytics.nn.modules.conv.DWConv	[128, 128, 3, 2, 1, False]
3	-1	6	839296	ultralytics.nn.modules.block.HGBlock	[128, 96, 512, 3, 6]
4	-1	1	5632	ultralytics.nn.modules.conv.DWConv	[512, 512, 3, 2, 1, False]
5	-1	6	1695360	ultralytics.nn.modules.block.HGBlock	[512, 192, 1024, 5, 6, True, False]
6	-1	6	2055808	ultralytics.nn.modules.block.HGBlock	[1024, 192, 1024, 5, 6, True, True]
7	-1	6	2055808	ultralytics.nn.modules.block.HGBlock	[1024, 192, 1024, 5, 6, True, True]
8	-1	1	11264	ultralytics.nn.modules.conv.DWConv	[1024, 1024, 3, 2, 1, False]
9	-1	6	6708480	ultralytics.nn.modules.block.HGBlock	[1024, 384, 2048, 5, 6, True, False]
10	-1	1	524800	ultralytics.nn.modules.conv.Conv	[2048, 256, 1, 1, None, 1, 1, False]
11	-1	1	789760	ultralytics.nn.modules.transformer.AIFI	[256, 1024, 8]
12	-1	1	66048	ultralytics.nn.modules.conv.Conv	[256, 256, 1, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	7	1	262656	ultralytics.nn.modules.conv.Conv	[1024, 256, 1, 1, None, 1, 1, False]
15	[-2, -1]	1	0	ultralytics.nn.modules.conv.Concat	[1]
16	-1	3	2232320	ultralytics.nn.modules.block.RepC3	[512, 256, 3]
17	-1	1	66048	ultralytics.nn.modules.conv.Conv	[256, 256, 1, 1]
18	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
19	3	1	131584	ultralytics.nn.modules.conv.Conv	[512, 256, 1, 1, None, 1, 1, False]
20	[-2, -1]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	3	2232320	ultralytics.nn.modules.block.RepC3	[512, 256, 3]
22	-1	1	590336	ultralytics.nn.modules.conv.Conv	[256, 256, 3, 2]
23	[-1, 17]	1	0	ultralytics.nn.modules.conv.Concat	[1]
24	-1	3	2232320	ultralytics.nn.modules.block.RepC3	[512, 256, 3]
25	-1	1	590336	ultralytics.nn.modules.conv.Conv	[256, 256, 3, 2]
26	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
27	-1	3	2232320	ultralytics.nn.modules.block.RepC3	[512, 256, 3]
28	[21, 24, 27]	1	7466252	ultralytics.nn.modules.head.RTDETRDecoder	[80, [256, 256, 256]]

rt-detr-l summary: 681 layers, 32,970,476 parameters, 32,970,476 gradients, 108.3 GFLOPs

Fuente: Elaboración propia

El modelo RT-DETR-L cuenta con aproximadamente 32.9 millones de parámetros, lo que permite un equilibrio entre capacidad de aprendizaje y eficiencia computacional. La Figura 26 muestra la cantidad exacta de parámetros del modelo RT-DETR-L.

Figura 26

Cantidad de parámetros del modelo preentrenado.

```
def count_parameters(model):
    trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
    non_trainable_params = sum(p.numel() for p in model.parameters() if not p.requires_grad)
    total_params = trainable_params + non_trainable_params
    print(f"Total de parámetros: {total_params}")
    print(f"Parámetros entrenables: {trainable_params}")
    print(f"Parámetros no entrenables: {non_trainable_params}")

count_parameters(model)
```

✓ 18.2s

```
Total de parámetros: 32970476
Parámetros entrenables: 0
Parámetros no entrenables: 32970476
```

Fuente: (Akrout & Mahdi, 2023)

2.3.1.3. Carga general de los dataset

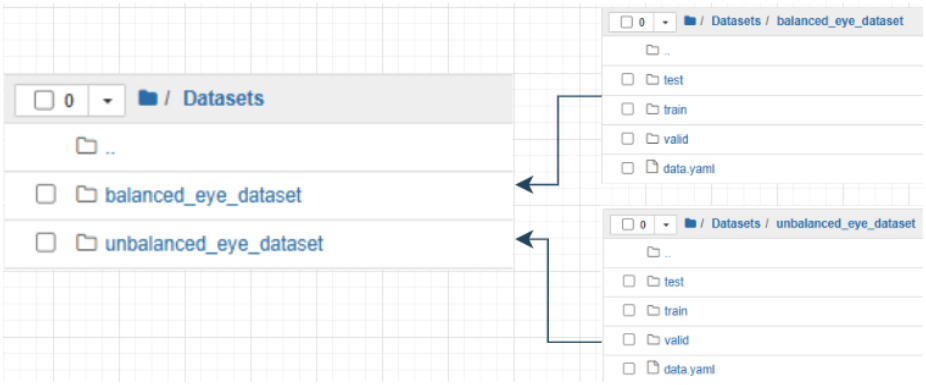
Para la carga de los datasets, se comprimieron en un archivo .zip, el cual posteriormente fue descomprimido para su uso en el entrenamiento. La referencia a los datos se realiza a través del archivo data.yaml, donde se especifica la ubicación de los conjuntos de datos.

El proceso de carga se llevó a cabo en HPC CEDIA, el entorno en el que se realizó el entrenamiento del modelo. Dentro de la carpeta Datasets, se encuentran los dos conjuntos de datos utilizados, tal como lo muestra en la Figura 27:

- Dataset balanceado: “balanced_eye_dataset”.
- Dataset desbalanceado: “unbalanced_eye_dataset”.

Figura 27

Dataset balanceado y desbalanceado.



Fuente: Elaboración propia

2.3.1.4. Carga y preparación de datos

Para entrenar el modelo, se utilizó un dataset propio, previamente creado y etiquetado específicamente para la detección de somnolencia en conductores. La API de Ultralytics permite una fácil integración de datasets personalizados mediante un archivo de configuración en formato YAML, el cual especifica:

- Nombre de las clases a detectar (open eye, closed eye).
- Número de clases.
- Rutas de los conjuntos de datos (test, train, valid).

La Figura 28 muestra el archivo de configuración utilizado para cargar el dataset.

Figura 28

Configuración para carga de dataset en el entorno de entrenamiento.

```
names:
- open eye
- closed eye
nc: 2
test: /home/ivan.garcia__utn.edu.ec/Datasets/balanced_eye_dataset/test/images
train: /home/ivan.garcia__utn.edu.ec/Datasets/balanced_eye_dataset/train/images
val: /home/ivan.garcia__utn.edu.ec/Datasets/balanced_eye_dataset/valid/images
```

Fuente: Elaboración propia

2.3.1.5. Entorno de entrenamiento

Para el entrenamiento del modelo RT-DETR, se utilizó la infraestructura de HPC CEDIA (High-Performance Computing de la Corporación Ecuatoriana para el Desarrollo de la Investigación y la Academia). Esta plataforma facilita el acceso a supercomputadoras y clusters de procesamiento optimizados para tareas computacionalmente intensivas como el entrenamiento de modelos de inteligencia artificial, simulaciones científicas, análisis de big data, procesamiento de imágenes y bioinformática (CEDIA, 2024).

El uso de HPC CEDIA permitió entrenar el modelo de manera eficiente, sin comprometer el rendimiento computacional. Sin embargo, la asignación de recursos depende de la demanda del sistema, y entornos con mayor capacidad pueden requerir tiempos de espera prolongados. Para evitar retrasos, se configuró un entorno con los siguientes recursos:

Configuración del entorno de entrenamiento:

- **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:** 3.8.12
- **PyTorch**: 2.4.1+cu121
- **Ultralytics:** 8.3.58

2.3.1.6. Configuración personalizada del entrenamiento

Para adaptar el modelo RT-DETR a la tarea de detección de somnolencia a través del estado de los ojos, se ajustaron diversos parámetros con el objetivo de optimizar el rendimiento del modelo en este contexto específico. Se configuró un esquema de entrenamiento enfocado en la detección de ojos abiertos y cerrados, asegurando un equilibrio entre eficiencia y precisión.

La configuración personalizada utilizada en este trabajo se define en la Tabla 6, donde se detallan los parámetros generales del entrenamiento:

Tabla 6

Hiperparámetros generales del entrenamiento.

Parámetro	**Valor**	**Descripción**
epochs	50	Número de épocas de entrenamiento.
imgsz	128	Tamaño de las imágenes de entrada.
patience	5	Número de épocas sin mejora antes de detener el entrenamiento.
batch	8	Tamaño del batch en cada iteración de entrenamiento.
project	“Somnolencia”	Nombre del proyecto dentro de la estructura de Ultralytics.
name	“RTDETR_balanceado2”	Nombre de la ejecución del modelo.
val	True	Habilita la validación durante el entrenamiento.
plots	True	Genera gráficos y métricas del entrenamiento.
verbose	True	Muestra información detallada del proceso.

Fuente: Elaboración propia

En la Tabla 7, se presentan los hiperparámetros utilizados en el entrenamiento para optimizar la convergencia del modelo.

Tabla 7

Hiperparámetros personalizados.

Parámetro	**Valor**	**Descripción**
optimizer	AdamW	Optimizador utilizado para la actualización de pesos.

cos_lr	True	Habilita la estrategia de aprendizaje con reducción cosenoidal.
dropout	0.5	Proporción de neuronas desactivadas en cada capa para evitar sobreajuste.
lr_0	0.0005	Tasa de aprendizaje inicial.
weight_decay	0.005	Penalización en los pesos para evitar sobreajuste.
momentum	0.95	Factor de momentum para optimización.
warmup_epochs	3.0	Número de épocas de calentamiento antes del aprendizaje principal.
warmup_momentum	0.8	Momentum inicial durante el calentamiento.
warmup_bias_lr	0.00	Tasa de aprendizaje inicial para los sesgos.

Fuente: Elaboración propia

Justificación de la configuración:

- **Reducción del tamaño de imagen (imgsz=128):**
  - RT-DETR trabaja con imágenes de múltiples resoluciones. En este caso, se usó un tamaño de 128x128 píxeles, suficiente para identificar los ojos en la imagen.
  - Al reducir la resolución, se optimiza la velocidad de entrenamiento sin comprometer la calidad de detección.
- **Ajuste de hiperparámetros para estabilidad y generalización:**
  - Se utilizó AdamW como optimizador en lugar del optimizador por defecto, lo que mejora la estabilidad del entrenamiento con peso de regularización (weight_decay=0.005).
  - Dropout (0.5) se configuró alto para evitar sobreajuste, dado que el dataset fue generado manualmente y no tiene una cantidad excesiva de datos.
  - Se implementó reducción cosenoidal de la tasa de aprendizaje (cos_lr=True) para mejorar la convergencia y evitar fluctuaciones en el entrenamiento.
- **Balance entre eficiencia y desempeño:**
  - Se redujo el batch size (batch=8) para optimizar el uso de memoria en la GPU.

- Momento de calentamiento (warmup_momentum=0.8) y tasa de aprendizaje de sesgo (warmup_bias_lr=0.00) fueron configurados para suavizar la transición en las primeras épocas.

En la Figura 29, se muestra la configuración implementada en el código.

Figura 29

Configuración de Hiperparámetros Personalizada.

```
# Configuración personalizada para deteccion de somnolencia
params = {
    **default_config,
    'epochs': 50,
    'imgsz': 128,
    'patience': 5,
    'batch': 8,
    'project': 'Somnolencia',
    'name': 'RTDETR_balanceado2',
    'val': True,
    'plots': True,
    'verbose': True,

    # Hyperparameters
    'optimizer': 'AdamW',
    'cos_lr': True,
    'dropout': 0.5,
    'lr0': 0.0005,
    'weight_decay': 0.005,
    'momentum': 0.95,
    'warmup_epochs': 3.0,
    'warmup_momentum': 0.8,
    'warmup_bias_lr': 0.00
}
```

Fuente: Elaboración propia

Para llevar a cabo el entrenamiento del modelo RT-DETR, se estableció una configuración óptima adaptada a la detección de somnolencia en conductores. En primer lugar, se utilizó el archivo de configuración data.yaml, el cual contiene la definición estructurada de las rutas de los conjuntos de datos de entrenamiento, validación y prueba. La configuración de este archivo se muestra en la Figura 28.

Además, se definieron las clases de interés, restringiendo la detección a dos categorías específicas:

- Clase 0: Ojos abiertos (open eye).

- Clase 1: Ojos cerrados (closed eye).

Asimismo, se empleó la variable **params, que almacena un diccionario con los hiperparámetros optimizados para ajustar el modelo a la tarea específica de detección de somnolencia. Esta configuración asegura que el modelo se entrene con los parámetros adecuados para maximizar su rendimiento y precisión.

En la Figura 30, se muestra el código correspondiente a la ejecución del entrenamiento con la configuración definida:

Figura 30

Ejecución de entrenamiento.

```
results = model.train(data = 'Datasets/balanced_eye_dataset/data.yaml', classes=[0,1],  **params)
```

Fuente: Elaboración propia

2.3.1.7. Estructura de archivos generada por Ultralytics

Una vez ejecutado el proceso de entrenamiento con RT-DETR en Ultralytics, la biblioteca crea automáticamente una carpeta de proyecto en la que se almacenan todos los archivos generados durante el entrenamiento. Esta carpeta contiene múltiples elementos clave para la evaluación del modelo, tales como:

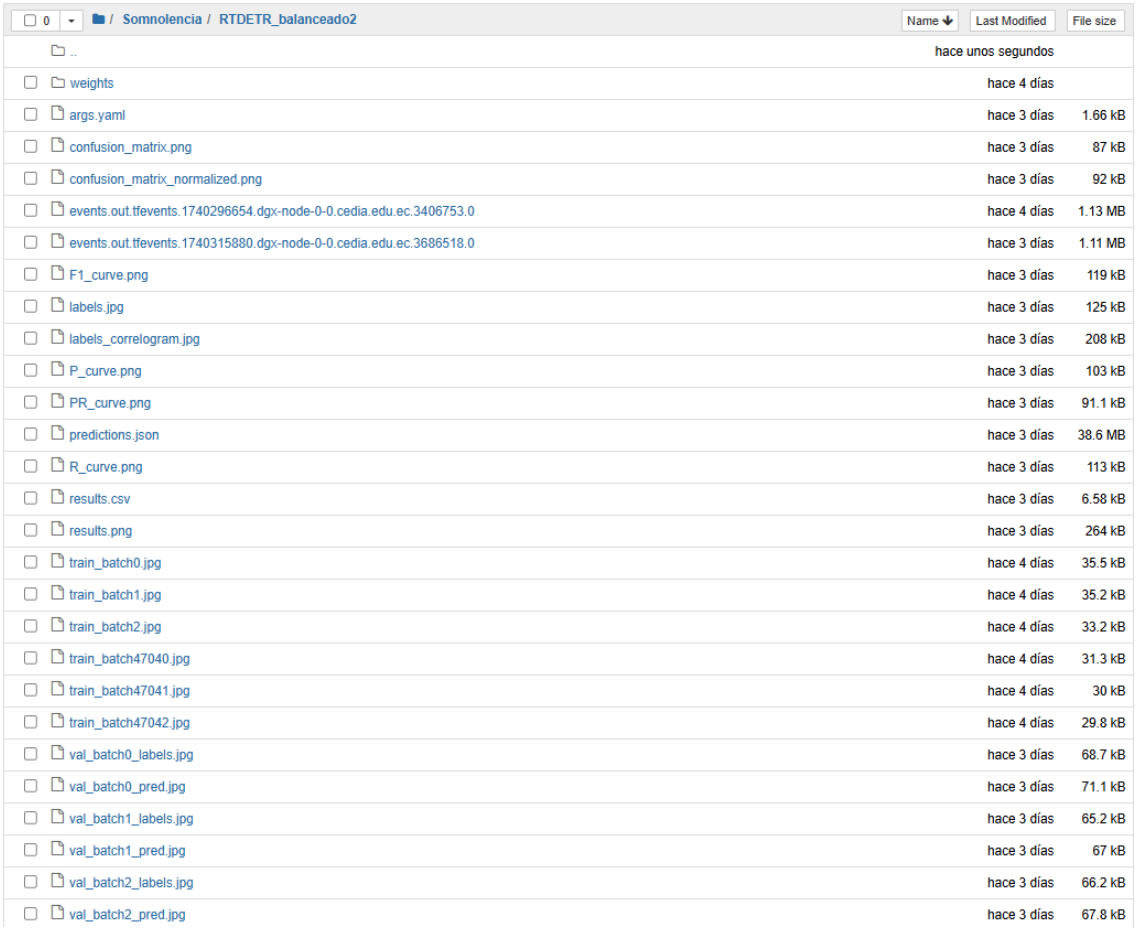
- **Pesos entrenados (weights):** Carpeta que almacena los modelos guardados, incluyendo el mejor modelo (best.pt) y el último (last.pt).
- **Métricas y resultados (results.csv, results.png):** Datos del desempeño del modelo durante el entrenamiento.
- **Matriz de confusión (confusion_matrix.png, confusion_matrix_normalized.png):** Representaciones gráficas del rendimiento del modelo en la clasificación de las categorías open eye y closed eye.
- **Curvas de evaluación (P_curve.png, R_curve.png, F1_curve.png, PR_curve.png):** Gráficos que reflejan precisión, recall, F1-score y precisión-recall, respectivamente.
- **Predicciones (predictions.json):** Archivo que almacena los resultados de inferencia en el conjunto de validación.

- **Lotes de entrenamiento y validación (train_batchX.jpg, val_batchX_pred.jpg, val_batchX_labels.jpg):** Imágenes generadas para visualizar ejemplos de entrenamiento y validación, mostrando las etiquetas reales y las predicciones del modelo.

En la Figura 31, se muestra la estructura de la carpeta generada por Ultralytics tras la finalización del entrenamiento.

Figura 31

Carpeta con archivos de entrenamiento.



Fuente: Elaboración propia

2.3.1.8. Arquitectura adaptada de RT-DETR para detección de somnolencia

El modelo RT-DETR, originalmente preentrenado con 80 clases en el conjunto de datos COCO, fue adaptado para detectar exclusivamente dos clases: open eye y closed eye. Esta adaptación implicó la reconfiguración de la arquitectura y los parámetros de entrenamiento,

ajustando el modelo para enfocarse en la detección de estados de los ojos con el objetivo de evaluar signos de somnolencia.

En la Figura 32, se presenta la arquitectura resultante del modelo entrenado específicamente para esta tarea. Se destacan los siguientes aspectos clave:

- **Número de capas:** 681 capas en total.
- **Parámetros entrenables:** 32,810,186.
- **GFLOPs:** 108.0, indicando la cantidad de operaciones de coma flotante necesarias para la inferencia.
- **Cabeza de detección RTDETRDecoder:** Adaptada para la detección específica de los ojos, con las clases open eye y closed eye.

Figura 32

Arquitectura

```
 12                -1  1     66048  ultralytics.nn.modules.conv.Conv             [256, 256, 1, 1]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14                 7  1    262656  ultralytics.nn.modules.conv.Conv             [1024, 256, 1, 1, None, 1, 1, False]
 15           [-2, -1]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 16                -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 17                -1  1     66048  ultralytics.nn.modules.conv.Conv             [256, 256, 1, 1]
 18                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 19                 3  1    131584  ultralytics.nn.modules.conv.Conv             [512, 256, 1, 1, None, 1, 1, False]
 20           [-2, -1]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 22                -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 23           [-1, 17]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 24                -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 25                -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 26           [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 27                -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 28       [21, 24, 27]  1   7305962  ultralytics.nn.modules.head.RTDETRDecoder    [2, [256, 256, 256]]
rt-detr-l summary: 681 layers, 32,810,186 parameters, 32,810,186 gradients, 108.0 GFLOPs
```

Fuente: Elaboración propia

Esta configuración demuestra cómo el modelo ha sido optimizado para el problema específico de detección de somnolencia, aprovechando un modelo preentrenado en una tarea más general de detección de objetos y ajustándolo para la identificación precisa de los estados de los ojos.

2.4. Desarrollo del sistema embebido

2.4.1. Especificaciones de hardware

El sistema embebido desarrollado para la detección de somnolencia se basa en una Raspberry Pi 3 Modelo B+, donde se aloja y ejecuta todo el proceso. Este dispositivo cuenta con

una memoria RAM de 1GB y utiliza una tarjeta microSD de 32GB para la instalación del sistema operativo y almacenamiento de archivos. Además, dispone de un puerto GPIO de 40 pines, empleado para conectar el zumbador de alerta a la placa principal, y un puerto de cámara CSI para la conexión con la cámara NIR OV5647, encargada de capturar las imágenes necesarias para el modelo. El sistema incluye un zumbador de 3V, que emite una alarma cuando se detecta somnolencia, y se alimenta mediante una fuente de 5V/2.5A, que proporciona energía tanto a la placa como a sus periféricos.

2.4.2. Especificaciones de software

El sistema embebido se desarrolló en una Raspberry Pi 4, ejecutando Raspberry Pi OS 32-bit (versión 2021-12-02) instalado en la tarjeta microSD, junto con las principales bibliotecas y herramientas necesarias para su funcionamiento. El código fue desarrollado en Geany 1.33, utilizando Python 3.7.3 para la implementación de la lógica de detección de somnolencia, con OpenCV 4.1.0 para el procesamiento de imágenes y NumPy 1.21.6 para la manipulación de datos. La ejecución del modelo ResNeXt-50 se optimizó mediante TensorFlow Lite Runtime 2.11.0, diseñado para dispositivos embebidos, mientras que Imutils 0.5.4 facilitó el acceso a la cámara y RPi.GPIO 0.7.1 permitió controlar los pines de la placa principal.

2.4.3. Integración del modelo RT-DETR en la placa base

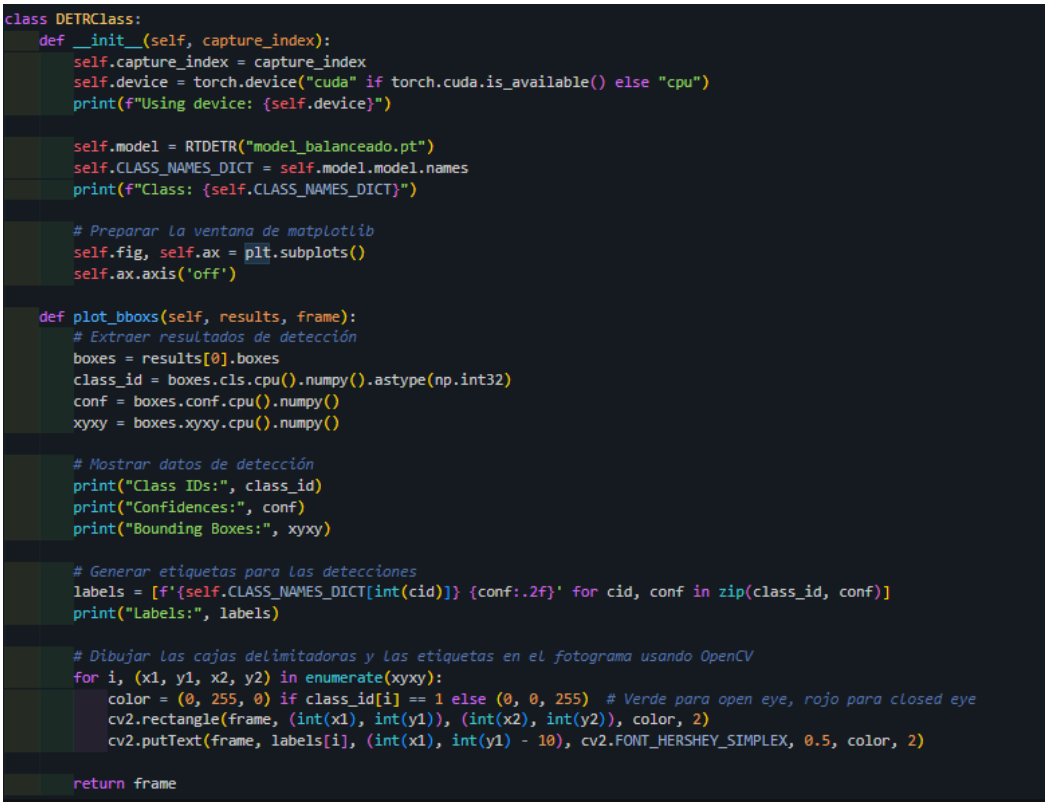
Para la integración del modelo en el sistema embebido, se configuró un entorno de desarrollo utilizando las herramientas y bibliotecas mencionadas en la sección **2.4.2 "Especificaciones de software"**, lo que permite administrar la instalación y eliminación de librerías sin comprometer otros elementos del sistema.

Se desarrolló un script en Python que implementa la lógica completa del detector de somnolencia en el sistema embebido. Este script ejecuta diversas funciones con el objetivo de obtener predicciones de objetos, en este caso, "open eye" o "closed eye", para la detección de somnolencia durante la conducción.

- **Importación de librerías y carga del modelo:** Se incorporaron las bibliotecas necesarias, entre ellas NumPy, OpenCV, TensorFlow Lite Runtime, Imutils y RPi.GPIO. Además, se implementó un método para la carga del modelo RT-DETR, como se ilustra en la Figura 33.

Figura 33

Método para cargar el modelo RT-DERT .



```python
class DETRClass:
    def __init__(self, capture_index):
        self.capture_index = capture_index
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        print(f"Using device: {self.device}")

        self.model = RTDETR("model_balanceado.pt")
        self.CLASS_NAMES_DICT = self.model.model.names
        print(f"Class: {self.CLASS_NAMES_DICT}")

        # Preparar la ventana de matplotlib
        self.fig, self.ax = plt.subplots()
        self.ax.axis('off')

    def plot_bboxs(self, results, frame):
        # Extraer resultados de detección
        boxes = results[0].boxes
        class_id = boxes.cls.cpu().numpy().astype(np.int32)
        conf = boxes.conf.cpu().numpy()
        xyxy = boxes.xyxy.cpu().numpy()

        # Mostrar datos de detección
        print("Class IDs:", class_id)
        print("Confidences:", conf)
        print("Bounding Boxes:", xyxy)

        # Generar etiquetas para las detecciones
        labels = [f'{self.CLASS_NAMES_DICT[int(cid)]} {conf:.2f}' for cid, conf in zip(class_id, conf)]
        print("Labels:", labels)

        # Dibujar las cajas delimitadoras y las etiquetas en el fotograma usando OpenCV
        for i, (x1, y1, x2, y2) in enumerate(xyxy):
            color = (0, 255, 0) if class_id[i] == 1 else (0, 0, 255)  # Verde para open eye, rojo para closed eye
            cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), color, 2)
            cv2.putText(frame, labels[i], (int(x1), int(y1) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

        return frame
```

Fuente: Elaboración propia

2.4.4. Aplicación para detección de somnolencia utilizando detección de objetos y PERCLOS

La detección de somnolencia es un aspecto fundamental en aplicaciones de seguridad vial y vigilancia. Para abordar este problema, se emplea un enfoque basado en detección de objetos, donde cada ojo se detecta como un objeto independiente y se clasifica como "open eye" o "closed eye". A partir de estas detecciones, se calcula el porcentaje de tiempo en que los ojos permanecen cerrados dentro de una ventana temporal para determinar la somnolencia.

- **Enfoque de Detección de Objetos:** En este proyecto, se utiliza un modelo de detección de objetos entrenado para identificar los estados de los ojos en secuencias de video. En cada cuadro del video, el modelo clasifica los ojos en las siguientes clases:
    - Clase 0: open eye
    - Clase 1: closed eye

Cada detección se representa con un rectángulo y una etiqueta indicando la clase y la confianza del modelo. La detección de los ojos permite analizar su estado sin necesidad de un detector de rostros, lo que mejora la robustez en diferentes condiciones.

- **Implementación de PERCLOS:** Para determinar el nivel de somnolencia, se implementa PERCLOS, un método basado en el análisis de cuadros dentro de una ventana de tiempo. La idea principal es calcular el porcentaje de cuadros donde los ojos están cerrados dentro de un periodo corto.

  En este caso, se ha definido una ventana de 2 segundos, lo que equivale a fps * 2 cuadros. La cantidad de cuadros cerrados en esta ventana se compara con un umbral predefinido:

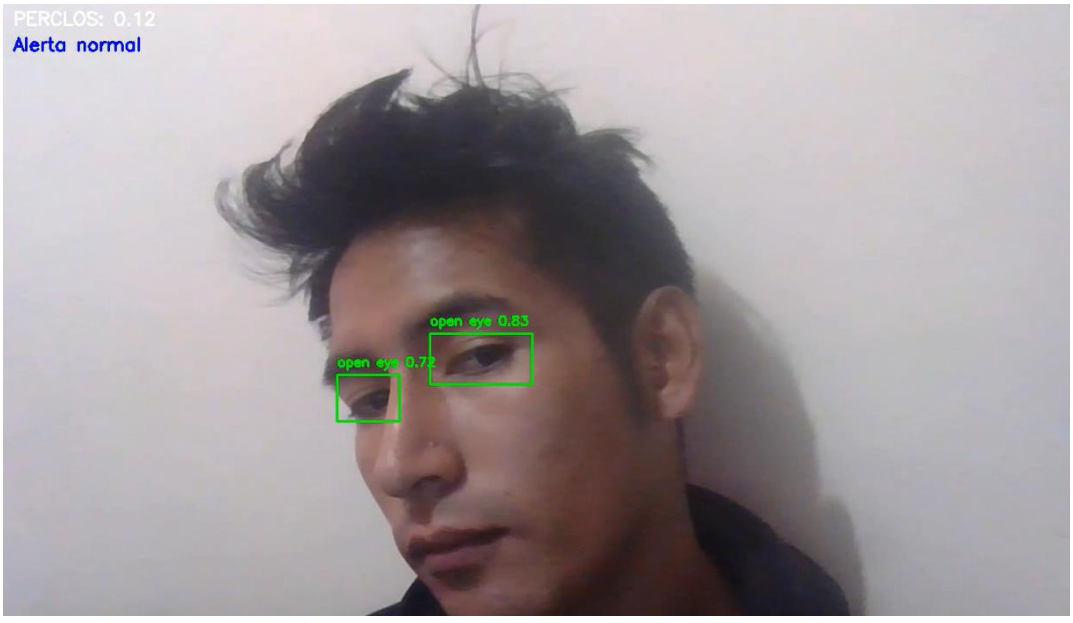
  - PERCLOS <= 15%: Estado normal (azul).
  - 15% < PERCLOS <= 30%: Signos de fatiga leve (amarillo).
  - PERCLOS > 30%: Somnolencia detectada (rojo).

  Si el porcentaje supera el 30%, se activa una alarma para alertar al conductor.

  En la Figura 34 se aprecia la clase y la confianza de esa clase, y tenemos un estado de alerta normal.

Figura 34

Alerta normal.



Fuente: Elaboración propia

La Figura 35 un estado de somnolencia con sus respectivas clases y confianza.

Figura 35

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

Fuente: Elaboración propia

Capítulo 3

Resultados

La evaluación de los modelos se va a realizar por medio de las métricas de clasificación. Además, se realizarán pruebas al modelo integrado en el sistema embebido. Estas pruebas serán tanto simuladas como reales, con el objetivo de analizar los resultados en diferentes entornos.

3.1. Evaluación de las métricas de Inteligencia Artificial

En el presente estudio, se evaluaron las métricas de desempeño de los modelos de Inteligencia Artificial entrenados, considerando indicadores clave en la tarea de clasificación, tales como pérdida (Loss), exactitud (Accuracy), precisión (Precision), sensibilidad (Recall) y F1 Score.

El entrenamiento de los modelos se llevó a cabo en el entorno HPC-CEDIA, obteniendo una pérdida combinada de 0.88% para el modelo entrenado con el dataset balanceado y 0.85% para el modelo entrenado con el dataset desbalanceado. En términos de exactitud, el modelo balanceado alcanzó un 95.21%, mientras que el modelo desbalanceado logró un 92.62%.

En cuanto a los tiempos de entrenamiento, el modelo basado en el dataset balanceado completó las 50 épocas en un tiempo total de 4 horas, 28 minutos y 21 segundos. En contraste, el modelo entrenado con el dataset desbalanceado finalizó el entrenamiento en 4 horas, 41 minutos y 22 segundos, tras completar únicamente 33 épocas. La detención temprana en la época 33 se debió a la ausencia de mejoras significativas en las métricas de validación, lo que indicaba que el modelo había alcanzado su punto de convergencia y no continuaría aprendiendo de manera efectiva.

A pesar de que el dataset desbalanceado contiene un 70% más de datos que el balanceado, los tiempos de entrenamiento son comparables debido a la diferencia en el número de épocas ejecutadas. Esto sugiere que el modelo entrenado con datos balanceados requirió un mayor número de iteraciones para optimizar sus parámetros, mientras que el modelo con datos desbalanceados alcanzó su límite de aprendizaje en una cantidad menor de épocas.

Los resultados obtenidos en las métricas de evaluación se presentan en la Tabla 8.

Tabla 8

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

Dataset	Combine Loss	Clasificación					Tiempo de entrenamiento
		Loss	Exactitud (%)	Precisión (%)	Recall (%)	F1 Score (%)	
Balanceado	0.88	0.43	95.21	95.47	95.75	95.61	4h 28m 21s
Desbalanceado	0.85	0.41	92.62	94.60	94.41	94.50	4h 41m 22s

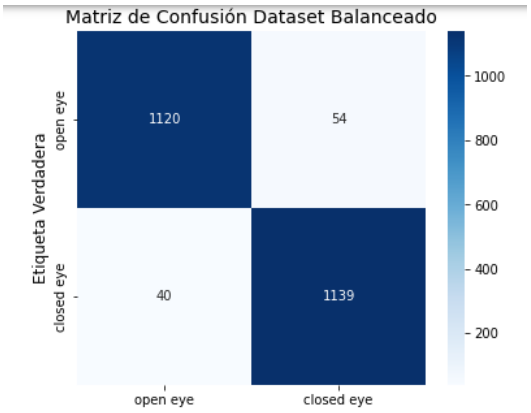
Fuente: Elaboración propia

Para cada modelo entrenado, se generó su respectiva matriz de confusión, junto con gráficos que reflejan la evolución de sus métricas durante el entrenamiento. En la Figura 36, se presenta la matriz de confusión correspondiente al modelo entrenado con el dataset balanceado, donde las etiquetas reales ("open eye" para ojos abiertos y "closed eye" para ojos cerrados) se encuentran representadas en el eje vertical, mientras que las predicciones del modelo aparecen en el eje horizontal.

La matriz de confusión obtenida refleja el desempeño del modelo en la clasificación de imágenes de ojos abiertos ("open eye") y ojos cerrados ("closed eye"). Se observa que el modelo identificó correctamente 1,120 imágenes de ojos abiertos y 1,139 imágenes de ojos cerrados, lo que indica un buen desempeño en ambas categorías. Sin embargo, se presentan 40 falsos positivos, en los cuales el modelo clasificó erróneamente imágenes de ojos cerrados como abiertos, y 54 falsos negativos, donde imágenes de ojos abiertos fueron detectadas como cerrados.

Figura 36

Matriz de confusión del modelo con el Dataset balanceado.

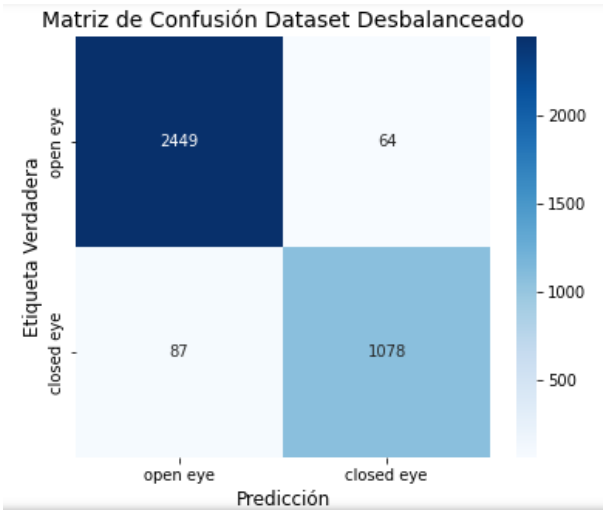


Fuente: Elaboración propia

En la Figura 37, se presenta la matriz de confusión obtenida para el modelo entrenado con el dataset desbalanceado, la cual refleja su desempeño en la clasificación de imágenes de ojos abiertos ("open eye") y ojos cerrados ("closed eye"). Se observa que el modelo identificó correctamente 2,449 imágenes de ojos abiertos y 1,078 imágenes de ojos cerrados, lo que indica que la detección de ojos abiertos es más precisa debido a la mayor cantidad de datos en esa categoría. Sin embargo, se presentan 64 falsos negativos, en los cuales el modelo clasificó erróneamente imágenes de ojos abiertos como cerrados, y 87 falsos positivos, donde imágenes de ojos cerrados fueron detectadas como abiertas. Este comportamiento sugiere que el desbalance en el dataset puede haber afectado la capacidad del modelo para clasificar con la misma precisión ambas clases, ya que la clase minoritaria ("closed eye") presenta una mayor cantidad de errores.

Figura 37

Matriz de confusión del modelo con el Dataset desbalanceado.



Fuente: Elaboración propia

La Figura 38 muestra la evolución de las métricas durante el entrenamiento del modelo con el dataset balanceado. En la primera fila se presentan las pérdidas durante el entrenamiento, donde se observa una disminución progresiva de la pérdida GIoU (train/giou_loss), la pérdida de clasificación (train/cls_loss) y la pérdida L1 (train/l1_loss), lo que indica que el modelo está aprendiendo a minimizar los errores en la detección de objetos a lo largo de las épocas. En la

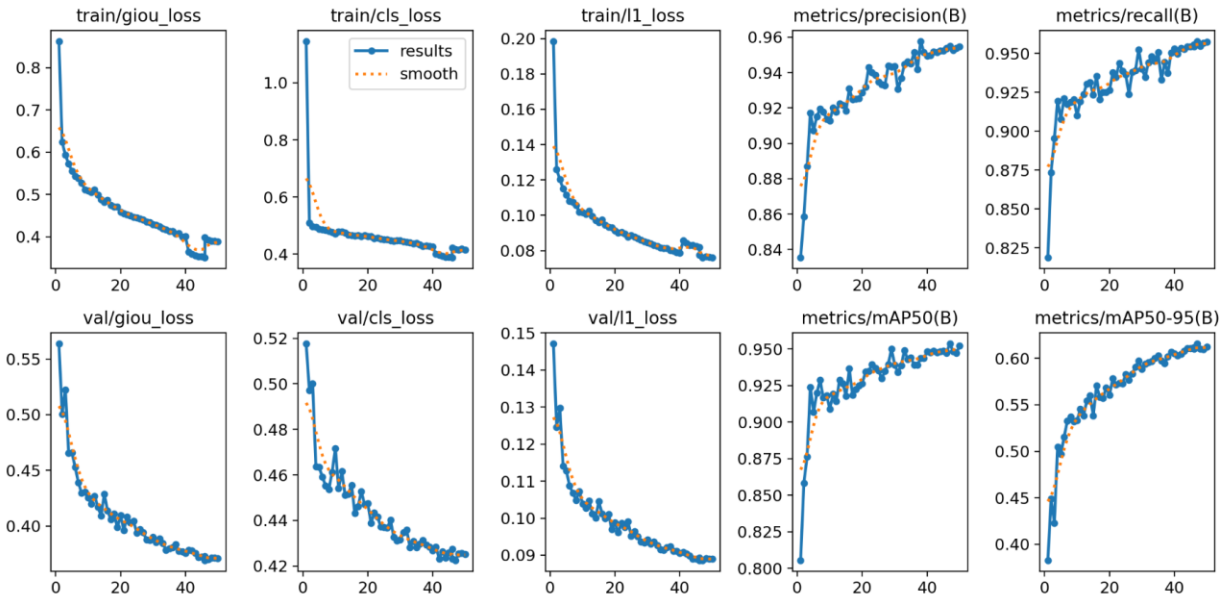
segunda fila, se muestran las mismas métricas, pero evaluadas en el conjunto de validación, donde también se aprecia una tendencia decreciente, lo que sugiere que el modelo está generalizando correctamente.

Asimismo, se presentan las métricas de precisión y recall, las cuales muestran un comportamiento ascendente, lo que indica que el modelo mejora su capacidad de detección a medida que avanza el entrenamiento. En particular, la precisión (metrics/precision(B)), que mide la cantidad de predicciones correctas sobre el total de predicciones realizadas, y el recall (metrics/recall(B)), que refleja la capacidad del modelo para detectar correctamente todas las instancias de la clase de interés, alcanzan valores superiores al 95%. Finalmente, las métricas mAP@50 (metrics/mAP50(B)) y mAP@50-95 (metrics/mAP50-95(B)), que evalúan la calidad de la detección utilizando diferentes umbrales de intersección sobre unión (IoU), evidencian una mejora constante, consolidando la capacidad del modelo para identificar correctamente las clases presentes en las imágenes de validación.

En la figura, los valores de cada métrica se representan con dos curvas: "results" (línea azul con puntos) y "smooth" (línea punteada naranja). La primera muestra los valores exactos obtenidos en cada época del entrenamiento, reflejando fluctuaciones naturales debido a la variabilidad inherente en el proceso de optimización. La segunda curva es una versión suavizada de estos valores, permitiendo identificar la tendencia general sin que el ruido afecte la interpretación. Esto facilita el análisis de la convergencia de las métricas y ayuda a evaluar si el modelo ha estabilizado su aprendizaje.

Figura 38

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

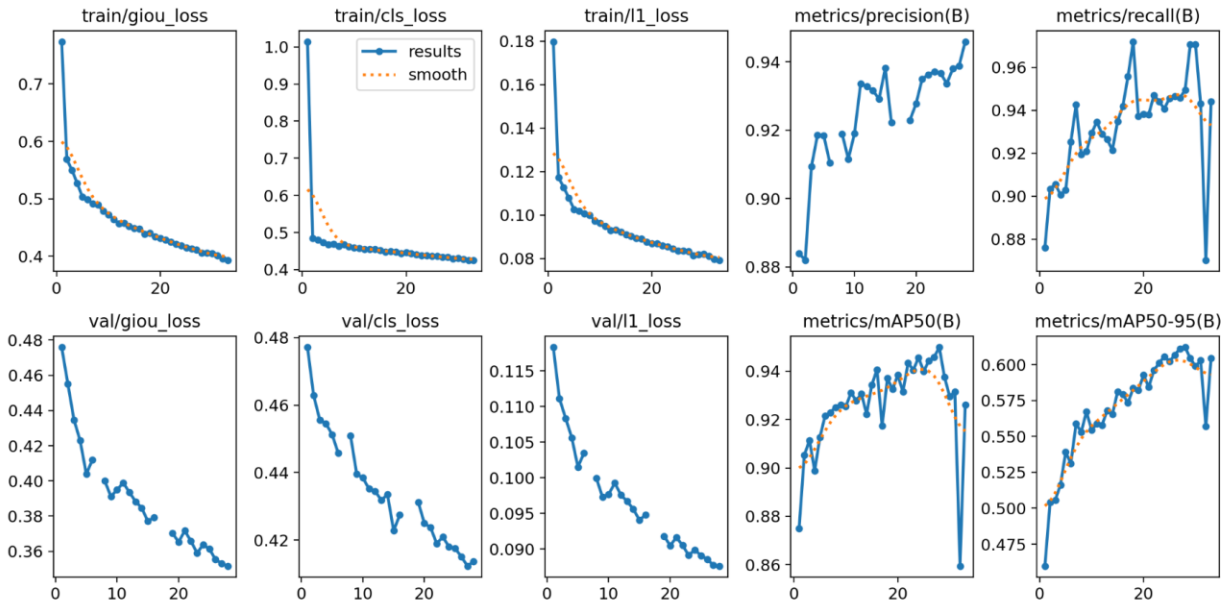


Fuente: Elaboración propia

La Figura 39 muestra la evolución de las métricas durante el entrenamiento con el dataset desbalanceado, donde la cantidad de imágenes con ojos abiertos es un 70% mayor que la de ojos cerrados. Si bien se observa una reducción progresiva de las pérdidas en entrenamiento y validación, lo que indica que el modelo está aprendiendo, las métricas de precisión, recall y mAP presentan fluctuaciones significativas. Esta inestabilidad sugiere que el modelo tiene dificultades para generalizar correctamente, posiblemente debido al desbalance de clases. La precisión es alta, lo que indica que las predicciones correctas son frecuentes, pero la variabilidad en el recall y mAP sugiere que el modelo podría estar sesgado hacia la clase mayoritaria (ojos abiertos), comprometiendo su capacidad de detectar ojos cerrados.

Figura 39

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



Fuente: Elaboración propia

Las gráficas presentadas son fundamentales para analizar el proceso de aprendizaje del modelo, permitiendo evaluar si está generalizando correctamente o si está cayendo en sobreajuste. A través de la evolución de las pérdidas y métricas como precisión y recall, es posible identificar patrones de mejora o inestabilidad, lo que facilita la toma de decisiones sobre ajustes en los hiperparámetros, como la tasa de aprendizaje, número de épocas o estrategias de balanceo de datos.

3.1.1. Evaluación con diferentes conjuntos de pruebas

Se evaluaron los dos modelos seleccionados utilizando cinco conjuntos de datos distintos: rostro sin accesorios (bareface), rostro con lentes (glasses), rostro con gafas de sol (sunglasses), y condiciones de iluminación diurna y nocturna. La comparación entre el modelo entrenado con datos balanceados y el entrenado con datos desbalanceados se realizará con base en métricas de clasificación, incluyendo Loss, exactitud, precisión, recall y F1 Score.

3.1.1.1. Métricas con accesorios faciales

Para analizar el desempeño del modelo en función de los accesorios faciales presentes en el rostro, se evaluó su rendimiento en tres conjuntos de datos distintos: imágenes de rostros sin

accesorios (bareface), imágenes de rostros con lentes (glasses), e imágenes de rostros con gafas de sol (sunglasses). Cada conjunto incluye muestras tomadas tanto en condiciones diurnas como nocturnas.

En la Tabla 9, se muestra la comparación de métricas entre estos tres conjuntos de datos obtenidas con el modelo entrenado en el Dataset balanceado.

En términos de Combine Loss, el modelo presenta un valor constante de 0.8849 en los tres casos, lo que sugiere que la pérdida general no varía significativamente entre los diferentes escenarios. Sin embargo, al analizar la Loss de clasificación, se observa un valor de 0.4251 en todos los conjuntos, lo que indica una estabilidad en la función de pérdida utilizada para la clasificación.

En cuanto a la exactitud, el modelo logra un desempeño consistente, con valores de 93.03% para rostros sin accesorios, 92.72% para rostros con lentes y 93.42% para rostros con gafas de sol. Esto indica que el modelo mantiene una precisión alta en todos los casos, aunque con una ligera disminución cuando los sujetos llevan lentes.

Las métricas de precisión (precision), recall y F1 Score muestran variaciones mínimas entre los diferentes conjuntos, con promedios de 94.41%, 94.34% y 94.38%, respectivamente. El rendimiento más bajo en recall (94.08%) se presenta en el conjunto de imágenes con gafas de sol, lo que podría indicar una ligera dificultad del modelo para detectar correctamente ciertos casos en este escenario.

Tabla 9

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

Modelo con Dataset Balanceado	**Combine Loss**	**Clasificación**				
		Loss	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1Score (%)**
Bareface	0.8849	0.4251	93.0297	94.3375	94.3333	94.3354
Glasses	0.8849	0.4251	92.7173	94.5817	94.6250	94.6034
Sunglasses	0.8849	0.4251	93.4208	94.3247	94.0809	94.2026
Promedio	0.8849	0.4251	93.0559	94.4146	94.3464	94.3805

Fuente: Elaboración propia

Se evaluó el modelo entrenado con el Dataset desbalanceado utilizando los mismos tres conjuntos de datos: imágenes de rostros sin accesorios (bareface), imágenes con lentes (glasses), e imágenes con gafas de sol (sunglasses).

En la Tabla 10, se presenta la comparación de métricas obtenidas para estos conjuntos de datos. En términos de Combine Loss y Loss de clasificación, los valores permanecen constantes en todos los conjuntos (0.8849 y 0.4251, respectivamente), lo que indica que la pérdida del modelo no se ve afectada por la presencia de accesorios faciales.

En cuanto a la exactitud, se observan diferencias significativas entre los distintos conjuntos:

- Bareface: 94.99%
- Glasses: 96.98%
- Sunglasses: 92.32%

El modelo presenta un mejor desempeño en la clasificación de rostros con lentes (96.98%), lo que sugiere que ha aprendido a manejar bien este tipo de imágenes. Sin embargo, la exactitud disminuye al clasificar rostros con gafas de sol (92.32%), lo que indica una mayor dificultad en este escenario.

Las métricas de precisión, recall y F1 Score muestran una variación similar:

- En bareface, los valores son consistentes, con una precisión de 95.02%, recall de 95.06% y F1 Score de 95.03%, lo que confirma un buen desempeño en rostros sin accesorios.
- En glasses, la precisión sigue alta (95.07%), pero el recall disminuye a 94.50%, lo que sugiere que el modelo tiende a ser ligeramente más conservador al detectar rostros con lentes.
- En sunglasses, la precisión baja a 93.28%, y el recall es el más bajo de los tres conjuntos (91.78%), lo que afecta la puntuación F1 (92.52%), reflejando la dificultad del modelo para clasificar correctamente rostros con gafas de sol.

En promedio, el modelo con el Dataset desbalanceado obtiene una exactitud de 94.76%, con precisión, recall y F1 Score de 94.46%, 93.78% y 94.11%, respectivamente.

Tabla 10

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

Modelo con Dataset Desbalanceado	Combine Loss	Clasificación				
		Loss	Exactitud (%)	Precisión (%)	Recall (%)	F1Score (%)
Bareface	0.8849	0.4251	94.9901	95.0169	95.0564	95.0317
Glasses	0.8849	0.4251	96.9788	95.0707	94.5000	94.7845
Sunglasses	0.8849	0.4251	92.3222	93.2849	91.7753	92.5240
Promedio	0.8849	0.4251	94.7637	94.4575	93.7772	94.1134

Fuente: Elaboración propia

3.1.1.2. Métricas en condiciones de luz diurna y nocturna

Se analizó el desempeño del modelo entrenado con el Dataset balanceado en dos condiciones de iluminación: diurna y nocturna. Para ello, se evaluó su rendimiento con imágenes capturadas en cada escenario.

En la Tabla 11, se presentan los resultados obtenidos. La pérdida combinada (Combine Loss) es similar en ambas condiciones, manteniéndose en 0.8849, al igual que la pérdida de clasificación (Loss: 0.4251), lo que indica que el modelo no experimenta una variación significativa en la optimización de la función de pérdida en función de la iluminación.

Sin embargo, en términos de exactitud, el modelo muestra una diferencia notable entre ambos conjuntos de datos:

- Día: 89.49%
- Noche: 91.78%

El modelo tiene un mejor desempeño en condiciones nocturnas, con un incremento de 2.29 % en la exactitud con respecto a las imágenes capturadas durante el día. Esto sugiere que el modelo es más robusto en escenarios con menor iluminación.

Las métricas de precisión, recall y F1 Score refuerzan esta observación:

- Para imágenes diurnas, la precisión es 91.36%, el recall 91.23% y el F1 Score 91.29%.
- Para imágenes nocturnas, se alcanzan valores más altos: 92.57% de precisión, 92.50% de recall y 92.53% de F1 Score.

El promedio general del modelo en ambos escenarios es una exactitud de 90.64%, con precisión, recall y F1 Score de 91.96%, 91.86% y 91.91%, respectivamente.

Tabla 11

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

Modelo con Dataset Balanceado	Combine Loss	Clasificación				
		Loss	Exactitud (%)	Precisión (%)	Recall (%)	F1Score (%)
Día	0.8849	0.4251	89.4921	91.3589	91.2271	91.293
Noche	0.8849	0.4251	91.7840	92.5682	92.5	92.5342
Promedio	0.8849	0.4251	90.6381	91.96	91.8636	91.9136

Fuente: Elaboración propia

Se evaluó el rendimiento del modelo entrenado con el Dataset desbalanceado en condiciones de iluminación diurna y nocturna, cuyos resultados se presentan en la Tabla 12. La pérdida combinada (Combine Loss) y la pérdida de clasificación (Loss) se mantuvieron constantes en 0.8529 y 0.4137, respectivamente, lo que indica una optimización estable independientemente de la iluminación. Sin embargo, al analizar las métricas de clasificación, se observa un mejor desempeño en condiciones nocturnas, donde el modelo alcanzó una exactitud del 92.77%, mientras que en condiciones diurnas obtuvo 90.77%. Asimismo, la precisión fue ligeramente superior en imágenes diurnas con 91.88%, en comparación con 92.73% en nocturnas, mientras que el recall y el F1 Score fueron más altos en la noche, con valores de 93.14% y 92.93%, respectivamente, frente a 91.79% y 91.84% en el día.

Tabla 12

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

Modelo con Dataset Desbalanceado	Combine Loss	Clasificación				
		Loss	Exactitud (%)	Precisión (%)	Recall (%)	F1Score (%)
Día	0.8529	0.4137	90.7769	91.8884	91.7996	91.844
Noche	0.8529	0.4137	92.7708	92.7299	93.1467	92.9378
Promedio	0.8529	0.4137	91.7739	92.3092	92.4732	92.3909

Fuente: Elaboración propia

3.1.2. Selección del mejor modelo

Para determinar cuál modelo es más adecuado, se compararon las métricas obtenidas en tres escenarios: evaluación general, accesorios faciales y condiciones de luz. En la Tabla 13, se presentan los resultados del modelo entrenado con el Dataset balanceado y el Dataset desbalanceado.

En la evaluación general, el modelo balanceado demuestra un desempeño superior, alcanzando una exactitud del 95.21%, frente al 92.62% del modelo desbalanceado. Además, presenta mayores valores de precisión, recall y F1 Score, lo que indica que tiene un mejor rendimiento global.

En la clasificación de rostros con accesorios faciales, el modelo desbalanceado obtiene una ligera ventaja en exactitud (94.76%), sin embargo, el modelo balanceado mantiene valores más altos en precisión, recall y F1 Score, lo que sugiere una mayor estabilidad en la clasificación.

En cuanto a las condiciones de iluminación, el modelo desbalanceado muestra una mejor exactitud (91.77% frente a 90.64%), además de superar al balanceado en precisión, recall y F1 Score, lo que indica que es más robusto para la detección en entornos con diferentes niveles de luz.

Sin embargo, al analizar el promedio de todas las métricas, el modelo balanceado presenta una mejor precisión (93.95% vs. 93.79%), recall (93.99% vs. 93.55%) y F1 Score (93.97% vs. 93.67%), mientras que el desbalanceado tiene una ligera ventaja en exactitud (93.15% vs. 92.97%).

Tabla 13

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	95.21	95.47	95.75	95.61	92.62	94.60	94.41	94.50
Accesorios faciales	93.06	94.42	94.35	94.38	94.76	94.46	93.78	94.11
Condiciones de luz	90.64	91.96	91.86	91.91	91.77	92.31	92.47	92.39
Promedio	**92.97**	**93.95**	**93.99**	**93.97**	**93.15**	**93.79**	**93.55**	**93.67**

Fuente: Elaboración propia

Con base en estos resultados, y dado que la prioridad es obtener una clasificación robusta en diferentes escenarios, se elige el modelo entrenado con el dataset balanceado.

3.1.3. Comparación de métricas con la literatura

El modelo seleccionado fue comparado con artículos relacionados con clasificación, aunque su arquitectura está basada en detección de objetos. A pesar de esta diferencia, los principales parámetros de evaluación siguen siendo comparables, ya que incluyen exactitud, precisión, recall y F1 Score, métricas comúnmente utilizadas en tareas de clasificación.

La diferencia clave radica en las funciones de pérdida. En modelos de clasificación, la pérdida se centra en la asignación correcta de clases, mientras que, en modelos de detección de objetos, como el nuestro, la pérdida se desglosa en tres componentes principales:

- Pérdida de ubicación (relacionada con la precisión en la predicción de las cajas delimitadoras).
- Pérdida de clasificación (asociada a la correcta identificación de la clase del objeto).
- Pérdida de asignación de objetos (que mide la correspondencia entre predicciones y objetos reales).

Sin embargo, la comparación con modelos de clasificación sigue siendo válida porque, en la práctica, nuestro modelo funciona como un clasificador, ya que detecta el objeto, coloca una caja delimitadora y asigna una confianza a la detección. Esto nos permite evaluar su desempeño de manera equivalente a un modelo de clasificación en términos de la precisión en la detección y asignación de clases.

En la Tabla 14, se comparan los resultados de clasificación de nuestro modelo (RT-DETR-L) con otros modelos de referencia. Nuestro modelo logra una exactitud del 95.21%, con precisión, recall y F1 Score superiores al 95%, posicionándolo entre los mejores modelos evaluados.

La comparación más relevante es con ResNext-50 (Inlago, 2025), ya que fue entrenado con los mismos datasets. Nuestro modelo supera a ResNext-50 en todas las métricas clave:

- Exactitud: 95.21% (RT-DETR-L) vs. 93.06% (ResNext-50)
- Precisión: 95.47% vs. 92.93%

- Recall: 95.75% vs. 93.02%
- F1 Score: 95.61% vs. 92.97%

Estos resultados demuestran que RT-DETR-L logra un rendimiento superior en términos de clasificación, lo que sugiere que la arquitectura de detección basada en RT-DETR es más efectiva para este problema específico.

A nivel global, el único modelo que supera a RT-DETR-L en exactitud es la CNN de Ahmed et al. (2023) con 97.00%, aunque sin detalles sobre la pérdida y con un margen reducido. Sin embargo, otros modelos populares como ResNet-50, InceptionV3 y VGG16 tienen una precisión significativamente menor, reforzando la efectividad de nuestro enfoque.

Tabla 14

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

Referencia	**Modelo**	**Loss (%)**	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1 Score (%)**
Nuestro modelo	RT-DETR-L		95.21	95.47	95.75	95.61
(Inlago, 2025)	ResNext-50	22.11	93.06	92.93	93.02	92.97
(Enriquez, 2025)	MobileNetV3Small	0..5	98.07			
(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.,	AlexNet	-	76.48	-	-	-
2021)	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	-	85.00	-	-	-

	GoogleNet Inception					
(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

En la Tabla 15, se presentan los resultados obtenidos en imágenes de rostros sin accesorios faciales. Nuestro modelo RT-DETR-L alcanza una exactitud del 93.03%, con una precisión del 94.34%, un recall del 94.33% y un F1 Score del 94.34%.

En comparación con el modelo ResNext-50 (Inlago, 2025), que obtiene una exactitud del 93.50%, nuestro modelo presenta un desempeño similar, aunque con una leve diferencia en precisión y recall. Otros modelos, como ResNet-18 - VGG16 - LSTM (92.73%) y MC-KCF (92.10%), muestran un rendimiento inferior. Sin embargo, el modelo PFLD – ViT - LSTM (Li & Li, 2024) se destaca con la mayor exactitud (95.26%), lo que indica que podría ofrecer un mejor desempeño en esta tarea específica.

Tabla 15

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

Referencia	**Modelo**	**Loss (%)**	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1 Score (%)**
Nuestro modelo	RT-DETR-L		93.03	94.34	94.33	94.34
(Inlago, 2025)	ResNext-50	20.34	93.50	94.84	92.00	93.40
(Enriquez, 2025)	MobileNetV3Small		97			
(Jamshidi et al., 2021)	ResNet-18 - VGG16 - LSTM	-	92.73	-	-	-
(Deng & Wu, 2019)	MC-KCF	-	92.10	-	-	-
(Li & Li, 2024)	PFLD – ViT - LSTM	-	95.26	-	-	-

Fuente: Elaboración propia

En la Tabla 16, se presentan los resultados obtenidos en imágenes de rostros con lentes. Nuestro modelo RT-DETR-L logra una exactitud del 92.72%, con una precisión del 94.58%, un recall del 94.63% y un F1 Score del 94.60%, mostrando un desempeño superior en comparación con otros modelos evaluados.

Frente al modelo ResNext-50 (Inlago, 2025), que obtiene una exactitud del 92.25%, nuestro modelo presenta un rendimiento ligeramente mejor, especialmente en precisión y recall. Otros modelos como MC-KCF (91.55%) y PFLD – ViT - LSTM (91.61%) muestran un desempeño inferior, mientras que métodos como 3DDGAN y ResNet-18 - VGG16 - LSTM tienen una exactitud considerablemente menor. Estos resultados sugieren que nuestro modelo mantiene una clasificación sólida incluso en presencia de accesorios faciales como lentes.

Tabla 16

Comparación de resultados obtenidos en imágenes de rostros con lentes.

Referencia	**Modelo**	**Loss (%)**	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1 Score (%)**
Nuestro modelo	RT-DETR-L		92.72	94.58	94.63	94.60
(Inlago, 2025)	ResNext-50	23.24	92.25	92.60	91.83	92.21
(Enriquez, 2025)	MobileNetV3Small		94			
(Bearly & Chitra, 2024)	3DDGAN	-	80.95	-	-	-
	3DDGAN - LA	-	85.50	-	-	-
(Jamshidi et al., 2021)	ResNet-18 - VGG16 - LSTM	-	84.19	-	-	-
(Deng & Wu, 2019)	MC-KCF	-	91.55	-	-	-
(Li & Li, 2024)	PFLD – ViT - LSTM	-	91.61	-	-	-

Fuente: Elaboración propia

En la Tabla 17, se comparan los resultados obtenidos en imágenes de rostros con gafas. Nuestro modelo RT-DETR-L alcanza una exactitud del 93.42%, con precisión del 94.33%, recall del 94.08% y un F1 Score del 94.20%, lo que lo posiciona por encima de la mayoría de los modelos evaluados.

En comparación con ResNext-50 (Inlago, 2025), que presenta una exactitud del 88.68%, nuestro modelo muestra una mejora significativa en esta métrica, aunque ResNext-50 obtiene una precisión ligeramente superior. Otros modelos, como PFLD – ViT - LSTM (91.10%) y MC-KCF, presentan un rendimiento inferior, mientras que modelos como 3DDGAN y 3D-CNN basado en GoogleNet Inception tienen una exactitud aún menor.

Estos resultados refuerzan la robustez de RT-DETR-L, destacándose como una de las mejores opciones para la detección de rostros con gafas.

Tabla 17

Comparación de resultados obtenidos en imágenes de rostros con gafas.

Referencia	**Modelo**	**Loss (%)**	**Exactitud (%)**	**Precisión (%)**	**Recall (%)**	**F1 Score (%)**
Nuestro modelo	RT-DETR-L		93.42	94.33	94.08	94.20
(Inlago, 2025)	ResNext-50	28.41	88.68	94.67	88.83	91.65
(Enriquez, 2025)	(Enriquez, 2025)		92			
(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

3.2. Evaluaciones con condiciones reales

Para verificar el desempeño del modelo en un entorno real, se implementó el sistema embebido en una camioneta Ford modelo 81 y un Hyundai Tucson, incorporando una cámara NIR con infrarrojo. Se llevaron a cabo pruebas con diez participantes, cada uno evaluado en sesiones de un minuto bajo condiciones similares a las utilizadas en la recopilación del conjunto de datos original. De cada sesión, se extrajeron 100 fotogramas, con una distribución equilibrada de 50 imágenes correspondientes a estados de somnolencia y 50 a estados de alerta. Este enfoque permitió obtener un conjunto de datos balanceado, facilitando la construcción de la matriz de confusión y el cálculo de las métricas de evaluación. La Tabla 28 muestra la matriz de confusión obtenida para el primer sujeto de prueba.

Tabla 18

Matriz de confusión sujeto de prueba 1

Sujeto de prueba 1		**Valores Predichos**	
		Awake	Drowsiness
Valores Reales	Awake	45	5
	Drowsiness	4	46

Fuente: Elaboración propia

Con base en las matrices de confusión obtenidas, se calcularon las métricas de evaluación, tales como accuracy, precision, recall y F1-score, para cada uno de los 10 sujetos de prueba. Los resultados se presentan en la Tabla 19.

Tabla 19

Métricas obtenidas de los 10 sujetos

Persona	**Accuracy**	**Recall**	**Precisión**	**F1 Score**
1	95	96	93	95
2	94	95	93	94
3	94	97	94	95

4	94	95	93	94
5	94	95	94	94
6	95	95	93	94
7	97	97	96	93
8	96	98	95	96
9	95	95	96	95
10	96	96	96	94
	94.9	**95.8**	**94.3**	**94.4**

Fuente: Elaboración propia

3.4. Análisis y reporte de resultados

3.4.1. Sistema embebido

El modelo RT-DETR, integrado en el sistema embebido Raspberry Pi 3 Modelo B+ para la detección de somnolencia mediante detección de objetos, presentó el siguiente rendimiento durante la ejecución del script. Mientras el proceso estaba en marcha, el uso promedio de la CPU alcanzó el 96%, y la velocidad de procesamiento del modelo se mantuvo en aproximadamente 5 fotogramas por segundo (FPS). Esta tasa de FPS relativamente baja se debe a la alta complejidad del modelo, que cuenta con más de 32 millones de parámetros, lo que demanda un procesamiento intensivo.

A pesar del bajo rendimiento en términos de velocidad, el modelo demuestra una alta precisión en la detección de los ojos, lo que es clave para la identificación de somnolencia. Además, es importante señalar que la tasa de fotogramas está limitada por las especificaciones del hardware utilizado. Con un dispositivo de mayor capacidad, sería posible lograr un incremento significativo en la velocidad de procesamiento.

Al evaluar el desempeño del modelo en la detección de somnolencia en 10 sujetos de prueba, se obtuvieron los siguientes resultados: exactitud del 94.9%, precisión del 95.8%, recall del 93.3% y F1-score del 94.4%. Estos valores reflejan un rendimiento estable, indicando un buen equilibrio en la detección de ojos abiertos y cerrados, lo que permite identificar con precisión los signos de somnolencia.

Limitaciones

Una de las principales limitaciones al inicio del desarrollo fue la dificultad para obtener un dataset adecuado y representativo para la detección de somnolencia. El conjunto de datos original presentaba un desequilibrio significativo, con una mayor cantidad de imágenes de ojos abiertos en comparación con ojos cerrados. Además, muchas de estas imágenes correspondían a personas en contextos no relacionados con la conducción, como figuras públicas o participantes de reality shows, lo que reducía la aplicabilidad del modelo a un entorno real de detección de somnolencia en conductores.

Para abordar este problema, se tomó la cantidad total de imágenes disponibles con ojos abiertos y se realizó una selección cuidadosa de aquellas que mejor representaban escenarios de conducción. Con esta estrategia, se logró construir un dataset balanceado, asegurando una proporción equitativa entre ojos abiertos y cerrados, lo que permitió iniciar las pruebas con una base de datos más adecuada para la tarea específica del modelo.

Otra limitación importante fue la necesidad de re-etiquetar completamente el dataset para adaptarlo al modelo RT-DETR, ya que inicialmente los datos estaban estructurados para modelos de regresión mediante landmarks y no para detección de objetos.

El dataset original contenía anotaciones basadas en coordenadas de landmarks para representar la posición de los ojos, lo cual no era compatible con el formato requerido por RT-DETR, que opera bajo un enfoque de detección de objetos con bounding boxes. Debido a esto, fue necesario volver a etiquetar todas las imágenes, definiendo las cajas delimitadoras en lugar de los puntos específicos de los landmarks.

Este proceso implicó un esfuerzo adicional significativo, ya que hubo que convertir la información previa y asegurarse de que las nuevas etiquetas fueran precisas para garantizar un entrenamiento efectivo del modelo en detección de objetos.

Discusiones

Durante el proceso de desarrollo del modelo, se exploraron diversas arquitecturas antes de llegar a la elección final del RT-DETR. Inicialmente, la red estaba enfocada en regresión de landmarks, utilizando una arquitectura basada en Transformers. Se probaron diferentes enfoques, comenzando con Vision Transformers (ViT), así como una versión con un dataset pequeño. Sin embargo, estos modelos no mostraron mejoras significativas debido a la falta de datos suficientes para entrenar adecuadamente la red. La regresión de landmarks en un dataset pequeño no convergió correctamente, lo que llevó a la conclusión de que una red orientada exclusivamente a regresión no era viable con la cantidad de datos disponible.

A pesar de los intentos por adaptar la red a una combinación de regresión y clasificación, el rendimiento aún no fue satisfactorio, especialmente cuando se probó con imágenes en escala de grises. Aunque la idea era mejorar el modelo, esta adaptación no funcionó como se esperaba, lo que llevó a la decisión de probar una arquitectura orientada a detección de objetos, dado que los modelos de Transformers, como el Deformable DETR, están específicamente diseñados para este tipo de tarea.

La arquitectura Deformable DETR, que se centra en prestar atención a zonas específicas como los ojos, mostró una mejora significativa en la tarea de detección de somnolencia en comparación con la regresión de landmarks. Sin embargo, se optó por RT-DETR, una versión optimizada tanto de DETR como de Deformable DETR, ya que aprovecha las fortalezas de ambas arquitecturas. RT-DETR mejora la eficiencia de la detección de objetos mediante el uso de técnicas como Hungarian matching y Non-Maximum Suppression (NMS), lo que lo hace especialmente adecuado para tareas en tiempo real y de alta precisión. Si hubiéramos intentado adaptar esta red para regresión, habríamos perdido las características clave que hacen que RT-DETR sea tan eficiente y optimizado para la detección de objetos.

Es importante destacar que, debido a la alta demanda computacional de estos modelos, el rendimiento del sistema depende en gran medida de la infraestructura utilizada. A pesar de que el CEDIA proporciona una plataforma con recursos de alto rendimiento, la red aún requiere procesadores poderosos o GPUs para un entrenamiento eficiente. Durante las pruebas, se observó que la red RT-DETR funcionó mejor en plataformas con mayor capacidad de cómputo, ya que el tiempo de entrenamiento aumentaba considerablemente en equipos con menos recursos.

Aunque se presentaron varios desafíos relacionados con la adaptación de arquitecturas y la disponibilidad de datos, el modelo RT-DETR demostró ser eficaz en la detección de somnolencia bajo las condiciones específicas de este trabajo, obteniendo buenos resultados tanto en precisión como en tiempo de entrenamiento.

Al analizar los resultados obtenidos, el modelo balanceado presentó métricas excepcionales. La exactitud alcanzó un 95.21% en el conjunto balanceado, destacándose con un accuracy promedio del 94.38%. Las métricas específicas para cada escenario fueron las siguientes: bareface con un 93.03%, glasses con un 92.72%, y sunglasses con un 93.42%. Además, el rendimiento en condiciones diurnas y nocturnas fue también notable, alcanzando un 89.49% y un 90.64%, respectivamente. Estos resultados muestran una buena capacidad del modelo para adaptarse a distintos escenarios de conducción, lo que valida su efectividad en la detección de somnolencia en diversos contextos. En particular, las métricas para el dataset balanceado fueron superiores en comparación con el dataset desbalanceado, lo que refuerza la importancia de contar con un conjunto de datos representativo y equilibrado para mejorar el desempeño del modelo.

Conclusiones

- La creación de un dataset balanceado fue esencial para mejorar el rendimiento del modelo. Al equilibrar la cantidad de imágenes con ojos abiertos y cerrados, y asegurar que las imágenes fueran representativas de un entorno de conducción, se logró mejorar considerablemente la precisión del modelo, obteniendo resultados significativamente mejores que cuando se utilizó un dataset desbalanceado.
- Intentar adaptar RT-DETR para regresión no sería factible, ya que se perderían más ventajas de las que se ganarían. La arquitectura de RT-DETR, optimizada para detección de objetos, depende de técnicas como Hungarian Matching y Non-Maximum Suppression (NMS), fundamentales para su eficiencia. Modificarla para regresión comprometería estas características clave, afectando negativamente su rendimiento y eficiencia en la tarea para la que fue originalmente diseñada.
- Utilizar plataformas como CEDIA para el entrenamiento de modelos es esencial, ya que redes como RT-DETR demandan altos recursos computacionales. Estas redes, debido a su complejidad y tamaño, requieren una infraestructura robusta con capacidad de procesamiento y almacenamiento avanzado. CEDIA proporciona el entorno adecuado para realizar entrenamientos de manera eficiente, maximizando el uso de la potencia de cómputo necesaria para modelos complejos.
- La baja tasa de inferencia en el sistema embebido se debe principalmente a las limitaciones de hardware, como la capacidad de procesamiento y la memoria disponible. A pesar de que el modelo fue optimizado para sistemas embebidos, la complejidad de la tarea y el tamaño del modelo afectan la velocidad de ejecución, lo que reduce la eficiencia en aplicaciones en tiempo real.

Recomendaciones

- En base a la limitación de datos pequeños y desbalanceados, es recomendable ampliar el dataset con más ejemplos, especialmente de situaciones de conducción realistas, para mejorar la capacidad del modelo en generalizar y detectar somnolencia en condiciones más variadas.
- Como se observó, al intentar modificar RT-DETR para tareas de regresión, se pierde la eficiencia que hace a este modelo potente. Por lo tanto, se recomienda mantener RT-DETR para su tarea original de detección de objetos y no intentar adaptarlo para regresión de landmarks, ya que este enfoque no sería efectivo y comprometería el rend Como el modelo requiere un gran poder de cómputo, el uso de plataformas como CEDIA es altamente recomendable para entrenar modelos complejos. Continuar utilizando estas plataformas para tareas con grandes requerimientos de recursos ayudará a reducir los tiempos de entrenamiento y mejorar el rendimiento general del modelo.
- La mejora de la tasa de inferencia podría lograrse mediante el uso de hardware más potente, como dispositivos con mayor capacidad de procesamiento y memoria. La optimización del modelo, mediante la reducción de su tamaño o la utilización de versiones más ligeras, también podría mejorar el rendimiento sin comprometer significativamente la precisión.

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* [Universidad Técnica del Norte]. http://repositorio.utn.edu.ec/handle/123456789/10171

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

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

Bernat, M. D., Galarza, L., Bisbal, E., Cebrián, G., Pagés, G., Morán, M. Á., Ferrandiz, M. D., & Melgarejo, A. (2021). Factores que afectan a la calidad del sueño en las unidades de cuidados intensivos. *Medicina Intensiva*, *45*(8), 470–476. https://doi.org/https://doi.org/10.1016/j.medin.2020.03.016

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-End Object Detection with Transformers. *ArXiv*. http://arxiv.org/abs/2005.12872

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

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

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 UNIVERSO. (2022, February 13). *La apnea obstructiva del sueño, una enfermedad que podría afectar al 18,5 % de choferes ecuatorianos, según estudio*. EL UNIVERSO. https://www.eluniverso.com/noticias/ecuador/la-apnea-obstructiva-del-sueno-una-enfermedad-que-podria-afectar-al-185-de-choferes-ecuatorianos-segun-estudio-nota/

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

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

Gottlieb, D. J., & Punjabi, N. M. (2020). Diagnosis and Management of Obstructive Sleep Apnea: A Review. *JAMA*, *323*(14), 1380–1400. https://doi.org/10.1001/JAMA.2020.3514

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

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

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

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

Rodríguez González, M. T., Gallego Gómez, J. I., Vera Catalán, T., López López, M. L., Marín Sánchez, M. C., & Simonelli Muñoz, A. J. (2018). Excessive daytime sleepiness and sleep hygiene of working adults in Spain. *Anales Del Sistema Sanitario de Navarra*, *41*(3), 329–338. https://doi.org/10.23938/ASSN.0378

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gómez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. *ArXiv*. https://doi.org/https://doi.org/10.48550/arXiv.1706.03762

Zhao, Y., Lv, W., Xu, S., Wei, J., Wang, G., Dang, Q., Liu, Y., & Chen, J. (2024). DETRs Beat YOLOs on Real-time Object Detection. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 16965–16974. https://doi.org/10.1109/CVPR52733.2024.01605

Zhu, X., Su, W., Lu, L., Li, B., Wang, X., & Dai, J. (2020). Deformable DETR: Deformable Transformers for End-to-End Object Detection. *ArXiv*. http://arxiv.org/abs/2010.04159

Anexos

Anexo 1: Entrenamiento del modelo RT-DETR

https://github.com/hpmezam/tesis_somnolencia.git