Universidad Autónoma de Querétaro

Facultad de Ingeniería

Maestría en Ciencias en Inteligencia Artificial

Sistema de reconocimiento de expresiones faciales para detección de estados de somnolencia con imágenes nocturnas.

Tesis

Que como parte de los requisitos para obtener el grado de
Maestría en Ciencias en Inteligencia Artificial

Presenta:

Ing. Karen Andrea Ramírez Arriaga

Dirigido por:

Dr. Juan Manuel Ramos Arreguín

Dr. Juan Manuel Ramos Arreguín	
Presidente	Firma
Dr. Jesús Carlos Pedraza Ortega	
Secretario	Firma
Dr. Saúl Tovar Arriaga	
Vocal	Firma
Dr. Sebastián Salazar Colores	
Suplente	Firma
Dr. Efrén Gorrostieta Hurtado	
Suplente	Firma

Centro Universitario Querétaro, Qro., México.
Junio 2023

Dedicatorias

Dedico esta tesis con mucho amor a mi mamá, que me apoyó incondicionalmente en los dos años del curso de este posgrado.

Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) por la beca otorgada durante los dos años de maestría, a la Universidad Autónoma de Querétaro por la beca institucional y a mi director de tesis, Dr. Juan Manuel Ramos Arreguín, por ser el guía en esta tesis. Agradezco también al cuerpo académico de la maestría y a mis sinodales, por la retroalimentación hacia este trabajo y el apoyo en las asignaturas.

RESUMEN

La somnolencia está directamente vinculada con el cansancio, por consiguiente, es un problema para las personas que conducen vehículos motorizados (automóviles, camiones, entre otros), ya que es responsable de alrededor del 20-30% de los accidentes en autopistas. Por esta razón, detectar que el conductor se está quedando dormido, es una forma de evitar accidentes ocasionados por fatiga. Existen varios trabajos que tratan este problema, especialmente utilizando imágenes tomadas con luz de día, siendo muy pocos los que trabajan en la detección de la somnolencia en ambientes nocturnos usando sistemas de visión nocturna. Esta tesis presenta la implementación de un algoritmo en el que se detecta la somnolencia en imágenes tomadas de noche, utilizando una cámara infrarroja. Para ello se utiliza una arquitectura YOLO-v4, obteniendo un 88% de precisión y un 93% de exhaustividad. Se utiliza una base de datos pública de somnolencia para entrenar y probar el algoritmo. Además, se realiza la implementación de dicho algoritmo en un sistema embebido.

Palabras clave: somnolencia, infrarrojo cercano, YOLO-v4, imágenes nocturnas, Jetson-nano.

ABSTRACT

Drowsiness is directly linked to fatigue, therefore, it is a problem for people who drive motor vehicles (cars, trucks, among others), since it is responsible for around 20-30% of accidents on highways. For this reason, detecting that the driver is falling asleep is a way to avoid accidents caused by fatigue. There are several works that deal with this problem, but especially with images taken in daylight, with very few working on the detection of drowsiness in night environments using night vision systems. This work presents an algorithm in which drowsiness is detected in images taken at night, using an infrared camera. For this, a YOLO-v4 architecture is used, obtaining 88% accuracy and 93% Recall. A public Drowsiness database is used to train and test the algorithm. In addition, the implementation of this algorithm in an embedded system is carried out.

Keywords: drowsiness, near infrared, YOLO-v4, nighttime images, Jetson-nano.

ÍNDICE DE CONTENIDO

RESUMEN i
ABSTRACT ii
ÍNDICE DE CONTENIDO iii
ÍNDICE DE FIGURAS vi
ÍNDICE DE TABLAS viii
ÍNDICE DE CÓDIGOS ix
Capítulo 1. Introducción 10
1.1. Antecedentes 10
1.2. Estado del arte 12
1.3. Descripción Del Problema 14
1.4. Justificación 15
1.5. Hipótesis y Objetivos 16
1.5.1. Hipótesis 16
1.5.2. Objetivo General 16
1.5.3. Objetivos Específicos 16
Capítulo 2. Marco teórico 18
2.1. Inteligencia artificial 18
2.2. Aprendizaje automático 19
2.3. Aprendizaje supervisado 19
2.4. Aprendizaje no supervisado 19
2.5. Neuronas artificiales 20
2.6. Redes neuronales artificiales 21
2.7. Redes neuronales convolucionales 21
2.7.1. Convolución 21
2.7.2. Redes neuronales convolucionales (CNN) 22
2.8. Infrarrojo cercano 23
2.9. Detección de imágenes 24
2.10. YOLO 24
2.11. Detección de rostros 25
2.12. Métricas para detección de objetos 25

2.12.1. mAP 25
2.12.2. IoU 26
2.12.3. Matriz de confusión 27
2.12.4. Exactitud 28
2.12.5. Precisión 28
2.12.6. Exhaustividad 28
2.12.7. Valor F1 29
2.13. Lenguaje de programación Python 29
2.14. Google Colaboratory 29
Capítulo 3. Metodología 30
3.1. Investigación bibliográfica 30
3.2. Análisis de los requerimientos 30
3.3. Selección de la base de datos 31
3.4. Etiquetado de las imágenes 31
3.5. División de los datos 36
3.6. Google Colab y Google Drive 38
3.7. Entrenamiento 39
3.7.1. Configuración del ambiente GPU en Google Colab 40
3.7.2. Instalación del darknet 41
3.7.3. Configuración del archivo personalizado para el darknet y descarga de los pesos pre-entrenados 41
3.7.4. Configuración de los directorios locales y del archivo “.cfg” 42
3.7.5. Entrenamiento personalizado de YOLOv4 45
3.7.6. Cargar los pesos entrenados 45
3.7.7. Realizar inferencias / pruebas 46
3.8. Implementación en sistema embebido 47
3.8.1. Instalación del darknet 47
3.8.2. Configuración del archivo personalizado para el darknet 48
3.8.3. Copiar los archivos “.cfg”, “.data” y “.names” 49
3.8.4. Pruebas del detector en sistema embebido 49
Capítulo 4. Resultados 50
4.1. Métricas en el entrenamiento 50
4.2. Pruebas 51

4.3. Métricas 55
4.4. Implementación en sistema embebido 56
4.5. Métricas en sistema embebido 60
4.6. Comparación de los resultados con el estado del arte 61
4.7. Publicación 62
Capítulo 5. Conclusiones 63
Anexos 65
Referencias 67

ÍNDICE DE FIGURAS

Figura 1. Ramas de la inteligencia artificial (Ciencia Carbónica, 2019). 18
Figura 2. Partes de una neurona artificial (Ramos, 2018). 20
Figura 3. Ejemplo de una red neuronal (Norman, 2019)..... 21
Figura 4. Arquitectura de una CNN (Pacheco, 2017). 22
Figura 5. Espectro electromagnético, (González, 2017). 23
Figura 6. Imagen luz visible / Imagen infrarroja, (RGB-NIR Data, n.d.). 23
Figura 7. Arquitectura de un detector (Gutta, 2021). 24
Figura 8. Intersección sobre unión (LearnOpencv, 2022). 26
Figura 9. Diferentes valores de IoU (LearnOpencv, 2022). 27
Figura 10. Matriz de confusión (Imagen propia). 28
Figura 11. Diagrama general de la metodología (Imagen propia)..... 30
Figura 12. Ejemplo de las imágenes de la base de datos Drozy Database (Massoz et al., 2016). 31
Figura 13. Imágenes de la base de datos dentro de una misma carpeta. 32
Figura 14. Herramientas del software LabelImg. 32
Figura 15. Etiquetado de una imagen en la clase Somnolencia en el software LabelImg... 33
Figura 16. Archivos generados después de realizar el etiquetado. 34
Figura 17. Contenido del archivo classes.txt. 34
Figura 18. Contenido del archivo image3.txt con su respectiva imagen. 35
Figura 19. Imagen etiquetada en clase No Somnolencia. 35
Figura 20. Imagen etiquetada en clase Somnolencia. 35
Figura 21. Division de las imágenes en carpetas. 37
Figura 22. Carpeta que contiene los archivos para el proyecto. 38
Figura 23. Carpeta con los datos divididos. 38
Figura 24. Carpeta con imágenes y archivos para el entrenamiento del detector. 39
Figura 25. Diagrama de la personalización de YOLO-v4 (Imagen propia). 40
Figura 26. Entorno de ejecución GPU. 41
Figura 27. Archivos con los pesos después del entrenamiento. 46
Figura 28. Descarga del darknet desde la terminal de Ubuntu. 48
Figura 29. Parte del código para la configuración del archivo “make” en Ubuntu. 48

Figura 30. Creación del archivo ejecutable. 48
Figura 31. Archivo “objcustom.data”..... 49
Figura 32. Archivo “obj.names”. 49
Figura 33. Desempeño durante el entrenamiento (Imagen propia). 50
Figura 34. Detección de la clase somnolencia en un 98%. 51
Figura 35. Detección de la clase no somnolencia en un 80%. 52
Figura 36. Detección de la clase somnolencia en un 95%. 52
Figura 37. Detección de la clase no somnolencia en un 97%. 53
Figura 38. Detección de la clase no somnolencia en un 92%. 53
Figura 39. Detección de la clase no somnolencia en un 99%. 54
Figura 40. Detección de la clase somnolencia en un 99%. 54
Figura 41. Detección de la clase somnolencia en un 76%. 55
Figura 42. Matriz de confusión obtenida en las pruebas (Imagen propia). 56
Figura 43. Detección de la clase No somnolencia en 73% y Somnolencia en 36% detectadas en el sistema embebido. 57
Figura 44. Detección de la clase somnolencia en un 96% detectada en el sistema embebido. 58
Figura 45. Detección de la clase No somnolencia en un 94% detectada en el sistema embebido. 58
Figura 46. Detección de la clase No somnolencia en un 97% detectada en el sistema embebido. 59
Figura 47. Detección de la clase No somnolencia en 79% y Somnolencia en 29% detectadas en el sistema embebido. 59
Figura 48. Detección de la clase No somnolencia en 55% y Somnolencia en 54% detectadas en el sistema embebido. 60
Figura 49. Matriz de confusión obtenida en las pruebas en el sistema embebido (Imagen propia). 60

ÍNDICE DE TABLAS

Tabla 1. Concentrado con publicaciones del estado del arte. 14
Tabla 2. Funciones de activación comunmente usadas (López, 2017). 20
Tabla 3. Métricas durante el entrenamiento. 51
Tabla 4. Métricas obtenidas en las pruebas. 56
Tabla 5. Métricas obtenidas en las pruebas en el sistema embebido. 61
Tabla 6. Comparación de resultados. 61

ÍNDICE DE CÓDIGOS

Código 1. Parte del código en Python que se usa para dividir las imágenes de la base de datos. 37
Código 2. Acceso a Google Drive desde Google Colab. 40
Código 3. Descarga del darknet desde el repositorio de GitHub. 41
Código 4. Parte del código para la configuración del archivo “make”. 42
Código 5. Creación del archivo ejecutable. 42
Código 6. Descarga de los pesos pre-entrenados. 42
Código 7. Configuración de directorios locales en espacio de trabajo. 43
Código 8. Configuración del archivo “.cfg” para entrenamiento personalizado. 44
Código 9. Muestra el archivo “.cfg”. 44
Código 10. Parte del archivo con las especificaciones de la red para el detector. 45
Código 11. Instrucción para realizar el entrenamiento personalizado de YOLO. 45
Código 12. Función para mostrar las imágenes. 46
Código 13. Prueba de cada imagen en el detector de somnolencia. 47
Código 14. Descarga del darknet desde el repositorio de GitHub para Linux. 47
Código 15. Instrucción para probar una imagen en el detector 49

Capítulo 1. Introducción

1.1. Antecedentes

El problema de somnolencia es algo muy común en personas que realizan actividades como vigilancia, conducción de vehículos, entre otras (Instituto Mexicano del Transporte, 2017). Debido a esto, se han desarrollado diversos algoritmos basados en inteligencia artificial para detectar estados de somnolencia. A continuación, se muestra el resultado de la revisión del estado del arte correspondiente.

Según Zhang et al. (2017) nuestras caras contienen mucha información útil que podemos usar para detectar la fatiga, como lo es el estado de los ojos. El análisis del porcentaje de cierre del párpado sobre la pupila (*PERCLOS*), es una medida comúnmente usada en los sistemas de detección de fatiga. En 2016, se desarrolló un sistema de detección de somnolencia en tiempo real, basado en *PERCLOS,* utilizando procesamiento de imágenes en escala de grises. En este trabajo se realizó un modelo de fatiga personal para cada conductor y de esta manera poder detectar un estado de somnolencia. Una vez que el conductor muestra fatiga, el sistema emite una alerta para que deje de conducir (Yan et al., 2016). En otra investigación, se realizó un sistema de tiempo real en donde se usan videos infrarrojos para desarrollar un método de reconocimiento del estado del ojo basado en una Red Neuronal Convolucional (CNN), calculando *PERCLOS*, así como la frecuencia de parpadeo para detectar la somnolencia. Los resultados experimentales muestran que el método propuesto tiene una alta precisión de reconocimiento del estado de los ojos cuando se usan gafas y puede detectar la fatiga de manera efectiva (Zhang et al., 2017).

En la investigación elaborada por Jie et al. (2017), se realizó un prototipo para detectar la somnolencia del conductor a partir de secuencias de video para un sistema avanzado de asistencia al conductor. El método extrae los descriptores faciales efectivos para describir la somnolencia en función de la alineación de la cara y clasifica los estados faciales del conductor a través de un *Random Forest* (RF). La clasificación y alineación

basadas en la estructura de RF son muy eficientes para la detección de somnolencia. El sistema puede obtener hasta un 94% de exactitud.

En 2018 y 2019, en los estudios de reconocimientos de expresiones de rostros nocturnos utilizaron principalmente infrarrojos que son más resistentes a la baja iluminación. A pesar de lo anterior, es difícil el reconocimiento de expresiones faciales, como el estado de somnolencia ante poca iluminación. Por esto, Geng et al. (2018), planteó un nuevo método de detección de fatiga del conductor basado en características morfológicas de infrarrojos y aprendizaje profundo. Utilizando una fuente de luz infrarroja, se obtuvo la imagen facial. La detección de las expresiones faciales se realizó por medio de una CNN con características morfológicas en imagen infrarroja, así como el uso de un filtro que midió el desplazamiento de la cabeza, con el objetivo de reducir el impacto del cambio de postura. Los resultados experimentales mostraron que la exactitud del algoritmo de detección de fatiga puede alcanzar el 94,48%. Otra contribución, con el mismo enfoque, se presentó en un trabajo que consta de un sistema de reconocimiento de la somnolencia del conductor basado en la detección del bostezo mediante imágenes térmicas, realizando observaciones continuas. El método puede funcionar en condiciones diurnas y nocturnas. Los experimentos se realizaron en laboratorio, así como en condiciones reales de automóvil (Knapik & Cyganek, 2019).

Dentro de los sistemas de vigilancia de estado de somnolencia actuales se encuentran aquellos que implementan conceptos de *Machine Learning*, en concreto de redes neuronales, como lo mostraron distintas investigaciones recientes, entre los que destaca el trabajo de Villanueva et al. (2019) que propuso un robusto sistema que incluye la detección de patrones en los rasgos faciales (cierre de ojos, la posición de la cabeza y los bostezos) de un conductor. En dicho sistema se usaron imágenes capturadas de una cámara, así como una red neuronal profunda. El dispositivo tiene una alarma que alerta al conductor cuando el patrón exhibido por los rasgos faciales se detecta como somnoliento. El sistema tiene una exactitud general del 97%. En una investigación del año 2020 se realizó un sistema de prevención de la somnolencia para evitar accidentes de tránsito, en

el cual se aplicó el aprendizaje automático para predecir la somnolencia mediante la tecnología de reconocimiento facial y parpadeo (Jang & Ahn, 2020).

Los sistemas de detección de fatiga o somnolencia se basan en el reconocimiento de expresiones faciales, por lo que es importante revisar la literatura con respecto a reconocimiento de expresiones faciales. En una investigación reciente, se desarrolló un prototipo para la detección de expresiones faciales utilizando modelos CNN. Los resultados experimentales lograron una exactitud del 57% en una tarea de cinco diferentes clasificaciones de expresiones faciales (Sai Mani Teja et al., 2020). En 2021, se implementó un prototipo de CNN utilizado para construir y entrenar un modelo de aprendizaje profundo. El objetivo fue clasificar imágenes faciales en un clasificador de rostros con siete categorías utilizando OpenCV (Khopkar, 2021).

Como resumen a toda la investigación realizada, se tiene que las principales técnicas de detección de expresiones faciales que involucran inteligencia artificial para los sistemas de vigilancia de somnolencia son aquellas que utilizan redes neuronales profundas (CNN principalmente). Para la detección de somnolencia con condiciones de poca luz o por la noche, así como en ambientes que generen ruido en las imágenes, normalmente se realizan implementaciones que mejoren los algoritmos de reconocimiento de expresiones faciales con las técnicas arriba mencionadas.

1.2. Estado del arte

El estado del arte comprende las publicaciones relacionadas con el proyecto desarrollado en esta tesis. Algunos de estos trabajos se presentan a continuación.

En el año 2019 Ma. et al., presenta una arquitectura de una red neuronal convolucional que se implementa para detectar fatiga en conductores. Para el entrenamiento de la red, utilizan una base de datos propia que contiene 39,000 imágenes

que obtienen a partir de una serie de videos infrarrojos. Los resultados obtenidos muestran una exactitud del 94%.

También en 2020, una contribución mostró un sistema que consta de la extracción de tres características de fatiga: frecuencia cardíaca, nivel de apertura de ojos y de boca; este artículo propuso una nueva red neuronal recurrente de fusión multimodal (MFRNN), que integra las tres características para mejorar la precisión de la detección de fatiga del conductor. Tanto la simulación como los resultados experimentales mostraron que el método propuesto proporciona un mejor rendimiento que métodos similares (Du et al., 2021).

Un artículo de investigación del año 2021 es de Chen et al. (2021), fue un modelo de estimación de la somnolencia del conductor basado en la fusión de características para detectar la somnolencia del conductor de manera eficiente y precisa; se logró mediante el uso de una CNN, extrayendo así de manera efectiva la representación profunda de las características de fatiga relacionadas con los ojos y la boca. Los resultados experimentales mostraron que esta propuesta tiene una mejor estabilidad y robustez en comparación con otros métodos.

Un trabajo más reciente del año 2022 es de Minhas et al. (2022), en el cual se presenta un método de monitoreo de distracción del conductor utilizando modelos de CNN de última generación como InceptionV3, VGG16 y ResNet50. Se usaron una colección de imágenes de rostros tanto activos como con somnolencia y el resultado que se obtuvo muestra mejor precisión para el modelo ResNet50 sobre los demás modelos evaluados.

En la Tabla 1 se muestra un concentrado con las publicaciones que son relevantes al desarrollo de este proyecto de tesis.

Tabla 1. Concentrado con publicaciones del estado del arte.

							Resultados
Autor y año	Articulo	Descripción	Técnica	Base de datos	# imágenes	Tipo de imágenes	Exactitud
(Ma et al., 2019)	Convolutional Three-stream Network Fusion for Driver Fatigue Detection from Infrared Videos	Arquitectura de una red neuronal convolucional para la detección de fatiga del conductor a partir de videos infrarrojos	Red neuronal convolucional (CNN).	Base de datos propia	39,000	Infrarrojos	94.68%
(Du et al., 2020)	Vision-Based Fatigue Driving Recognition Method Integrating Heart Rate and Facial Features	Sistema que consta de la extracción de tres características de fatiga: frecuencia cardíaca, nivel de apertura de ojos y de boca	Red neuronal recurrente (RNN)	Base de datos propia	60,000	RGB-D	94.74%
(Chen et al., 2021)	Driver Drowsiness Estimation Based on Factorized Bilinear Feature Fusion and a Long-Short-Term Recurrent Convolutional Network	Modelo de estimación de somnolencia del conductor basado en la fusión de características mediante el uso de una CNN	Red neuronal convolucional (CNN)	NTHU-DDD dataset	450 video clips	Escala de grises	75.80%
(Minhas et al., 2022)	A smart analysis of driver fatigue and drowsiness detection using convolutional neural networks	Método de monitoreo del conductor en tiempo real utilizando la red neuronal convolucional (CNN)	Red neuronal convolucional (CNN)	Colección de imágenes de rostros humanos activos y dormidos.	-	-	93,69 %

1.3. Descripción Del Problema

El reconocimiento de expresiones faciales enfocado a detección de fatiga implica un gran desafío debido a condiciones de poca luz que pueda generar el ambiente, así como también la posición de la cara. Los sistemas de reconocimiento de estados de somnolencia en ambientes diurnos se basan en utilizar cámaras de luz visible para obtener las imágenes y realizar el reconocimiento de expresiones, bajo condiciones nocturnas o de poca iluminación, que suelen ser una problemática a resolver (Knapik & Cyganek, 2019).

La implementación de algoritmos para detección de somnolencia con imágenes ante ambientes como el mencionado anteriormente, significa un reto en cuanto a exactitud. Por ello, con la aplicación de técnicas de inteligencia artificial, como lo son las distintas variantes de redes neuronales, se puede tener un algoritmo robusto, tomando en cuenta

las condiciones de iluminación, así como también de la posición del rostro, para realizar una correcta detección ante una señal de fatiga, alertando al conductor para detenerse y evitar un accidente de tránsito (Ma et al., 2019).

1.4. Justificación

La Organización Mundial de la Salud estimó que las muertes debido a accidentes viales alcanzaron 1.35 millones en 2016 a nivel mundial (WHO, 2018). Específicamente en México, según datos del Instituto Nacional de Estadística y Geografía (Instituto Nacional de Estadística y Geografía (INEGI), 2016), la ocurrencia de accidentes viales fue de 378,232 en el año 2015. Datos de la policía federal indican que las causas de accidentes viales en carreteras federales se distribuyen en los siguientes factores: 87% por el conductor, 1.82% por el peatón o pasajero, 7.43% por el vehículo, 1.68% por el camino, 1.26% por irrupción de ganado y 0.44% por un agente natural (Dirección General de Servicios Técnicos, 2017). Según la Estadística de accidentes de tránsito realizada por el Instituto Mexicano del Transporte, el 4% de los accidentes ocurridos en 2020 en la república mexicana, tuvieron como circunstancia que el conductor se encontraba “Dormitando” (Instituto Mexicano del Transporte, 2021).

En diversos países se han realizado estudios que muestran la influencia de la fatiga como responsable de accidentes de tránsito. Por ejemplo, en Reino Unido un estudio realizado por Loughborough University (Chávez, 2010) señaló que la fatiga es la responsable del 20% de los accidentes ocurridos en autopistas, 35% en Alemania, 20% en Noruega y 17% en Estados Unidos. Otro estudio (Friswell & Williamson, 2008) realizado con la finalidad de encontrar las experiencias de fatiga en conductores de transporte por carretera encontró que el 38% de los participantes experimentaron fatiga al menos una vez a la semana y el 45% se había quedado dormido mientras conducían. Dichos resultados sugieren que la fatiga es un problema para los conductores de transporte. De acuerdo a una entrevista realizada a 41 agentes de tránsito de la ciudad de México (Torres et al., 2020) con la finalidad de estudiar los factores que influyen en un accidente vial, se encontró

que el 65% de los agentes viales consideran que la conducción con fatiga o sueño influye en accidentes viales.

La somnolencia o la fatiga en conductores es un factor que causa accidentes de tráfico, debido a que un conductor somnoliento disminuye su capacidad de atención y concentración durante el manejo; por lo que después de un largo periodo de tiempo de permanecer conduciendo, el conductor puede llegar a quedarse dormido. El pestañear y dormitar durante la conducción significa falta de sueño por parte del conductor, y usualmente los accidentes producidos en estas circunstancias tienen alta siniestralidad en términos de pasajeros muertos, heridos y pérdidas materiales (D. Martinez et al., 2018).

Por lo anterior, contar con un sistema de vigilancia que detecte estados de somnolencia emitiendo una alerta al conductor, puede reducir los accidentes viales que, en ocasiones, son accidentes fatales.

1.5. Hipótesis y Objetivos

1.5.1. Hipótesis

Con el uso de algoritmos de inteligencia artificial se puede obtener una exactitud similar a los trabajos del estado del arte en la detección de expresiones faciales de estados de somnolencia considerando imágenes nocturnas.

1.5.2. Objetivo General

Aplicar técnicas de inteligencia artificial en imágenes que contengan rostros, para detectar estados de somnolencia, considerando condiciones nocturnas.

1.5.3. Objetivos Específicos

a. Analizar e implementar al menos un algoritmo de inteligencia artificial que detecte rostros y extraiga puntos importantes para detectar somnolencia en imágenes nocturnas con variaciones de intensidad.

b. Realizar adecuaciones necesarias al algoritmo para poder incrementar la exactitud al momento de reconocer el estado de somnolencia.

c. Diseñar una interfaz de comunicación con el usuario para alertar al conductor del riesgo de quedarse dormido y se tome alguna precaución.

d. Probar el algoritmo utilizado y compararlo con al menos un algoritmo del estado del arte, para verificar la exactitud del reconocimiento del estado de somnolencia.

e. Implementar el algoritmo en un sistema embebido para realizar pruebas de funcionalidad y desempeño del algoritmo, considerando variaciones de iluminación.

Capítulo 2. Marco teórico

A continuación, se explica más a detalle los conceptos relacionados a la investigación y al desarrollo del prototipo aquí presentado.

2.1. Inteligencia artificial

La inteligencia es un término difícil de definir, pero podría resumirse como la capacidad de adaptación al entorno para resolver los problemas que se presentan. La inteligencia artificial (IA) dota a un sistema con un mecanismo que permite simular el comportamiento de un ser vivo e incluso adaptar este sistema a cambios y modificaciones en el entorno (Mathivet, 2018). Según Boden (2016) la IA tiene por objeto que las computadoras hagan la misma clase de cosas que puede hacer la mente. Desde un punto de vista más informático, IA es un conjunto de técnicas, algoritmos y herramientas que nos permiten resolver problemas para los que es necesario cierto grado de inteligencia (García, 2012). La Figura 1 muestra algunas de las ramas que tiene la inteligencia artificial.

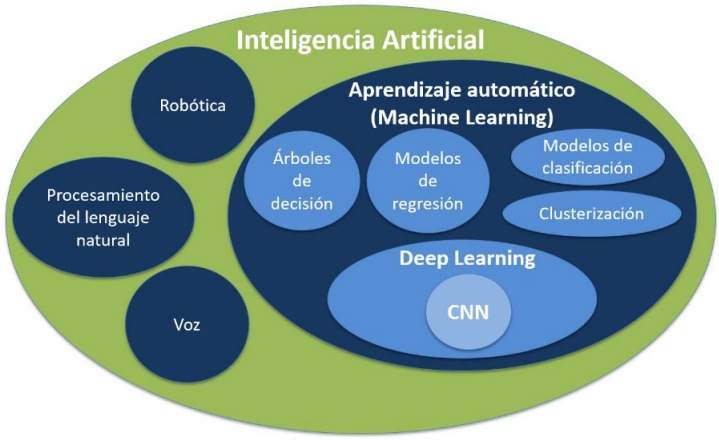


Figura 1. Ramas de la inteligencia artificial (Ciencia Carbónica, 2019).

Algunas técnicas de inteligencia artificial se explican a continuación.

2.2. Aprendizaje automático

Es una disciplina científica del ámbito de la IA que crea sistemas que aprenden automáticamente. Aprendizaje automático trata sobre el aprendizaje automatizado el cual se encuentra dentro de la derivación de las ciencias de la computación y propia de las ramas más sonantes de la IA (Milagros & Julian, 2020). El aprendizaje automático es la ciencia que enseña a las computadoras a hacer predicciones basadas en datos; en un nivel básico, el aprendizaje automático implica dar a una computadora un conjunto de datos y pedirle que haga una predicción. Al principio, estas predicciones serán incorrectas, sin embargo, con el transcurso de miles de predicciones, la computadora actualizará su algoritmo para hacer mejores predicciones (Norman, 2019). La importancia del aprendizaje reside en que sus resultados habitualmente se traducen en mejoras en la calidad de actuación del sistema (Moreno et al., 1994).

2.3. Aprendizaje supervisado

Se caracteriza por necesitar, al menos, un vector de entrada y otro de salida; el primero de ellos contendrá la información que habrá de ser aprendida, mientras que el segundo, contendrá la etiqueta de salida que habrá de considerarse correcta para dicho vector de entrada. Este tipo de aprendizaje se realiza enseñándole al sistema la relación entre sus vectores de entrada y salida, estableciendo la relación como referencia del comportamiento que se espera que tenga. Como ejemplo de este tipo de aprendizaje pueden mencionarse las redes neuronales (Alcántara, 2018).

2.4. Aprendizaje no supervisado

Los algoritmos de aprendizaje automático no supervisado no utilizan información previa, el programa tiene la capacidad de configurarse a medida que procesa las observaciones que va recibiendo. La salida será la respuesta ante la similitud entre la entrada actual y la información que se le ha otorgado a la entrada en ocasiones previas. Está formado de un conjunto de reglas que ayudan a la red a aprender asociaciones entre

los patrones que frecuentemente se les presentan, uno de los ejemplos de esta clase de aprendizaje es el *clustering* (Alcántara, 2018).

2.5. Neuronas artificiales

Las neuronas artificiales tienen mucha importancia en el desarrollo de la tecnología, especialmente en lo que tiene que ver con control, procesamiento de señales, reconocimiento de patrones, entre otros. La 0 muestra el esquema de una neurona en general. Se pueden apreciar las siguientes partes: entradas, pesos, función sumatoria, función de activación y salida (Ramos, 2018). Las funciones de activación comúnmente usadas se muestran en la Tabla 2.

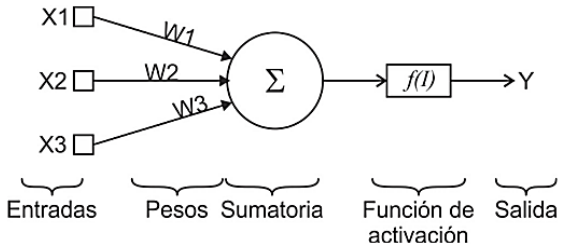
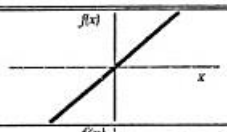
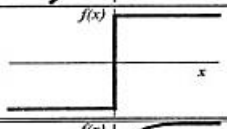
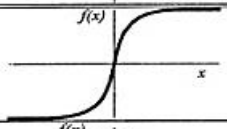
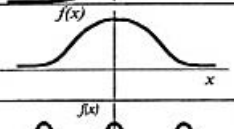
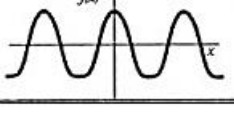


Figura 2. Partes de una neurona artificial (Ramos, 2018).

Tabla 2. Funciones de activación comunmente usadas (López, 2017).

	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0, +1]$ $[-1, +1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0,+1]$	
Sinusoidal	$y = A\,\mathrm{sen}(\omega x + \varphi)$	$[-1,+1]$	

2.6. Redes neuronales artificiales

El modelo abstracto de las redes neuronales artificiales surge de intentar imitar el comportamiento de las neuronas que se encuentran en el cerebro (Ameijeiras et al., 2020) Una red neuronal consta de muchos nodos que trabajan juntos para producir una respuesta. Al principio, una red neuronal cometerá muchos errores, pero con el transcurso de las pruebas, actualizará cada nodo para mejorar la búsqueda de la respuesta correcta (Norman, 2019). En la Figura 3 se aprecia una red neuronal multicapa compuesta por una capa de entrada, dos capas ocultas o capas intermedias y una capa de salida.

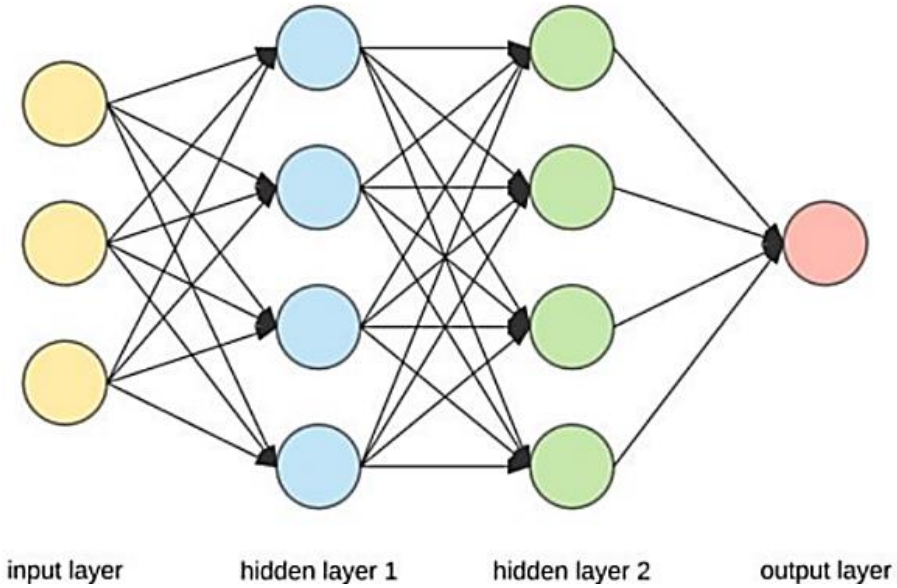


Figura 3. Ejemplo de una red neuronal (Norman, 2019).

2.7. Redes neuronales convolucionales

2.7.1. Convolución

Una convolución (López, 2017) es un operador matemático que trabaja con dos argumentos (funciones), f y g, y las transforma en una tercera. La convolución en tiempo continuo se define como la integral del producto de dos funciones después de desplazar una de ellas una distancia t. Sean las funciones f y g. Su convolución está denotada como:

$$f * g = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau) d\tau \qquad (1)$$

El intervalo de integración depende del dominio sobre el cual están definidas las funciones. Para el caso discreto se tiene que la integral se intercambia por la sumatoria, dando como resultado:

$$f * g = \sum_{-\infty}^{\infty} f(k) g(n-k) \qquad (2)$$

Donde una de las funciones está desplazada una distancia n.

2.7.2. Redes neuronales convolucionales (CNN)

Presentadas por LeCun et al. (1989) a principios de la década de los noventa, las redes convolucionales son un ejemplo de una arquitectura de red neuronal especializada. Las CNN presentan una arquitectura multicapa, donde cada capa está constituida por un número determinado de convoluciones con funciones de activación no lineal. El tipo de conexiones de las neuronas en la red convolucional está inspirado en la organización de la corteza visual de los animales, en la cual la respuesta de cada neurona puede representarse matemáticamente como una operación de convolución (Pacheco, 2017). La Figura 4, muestra la arquitectura de una CNN.

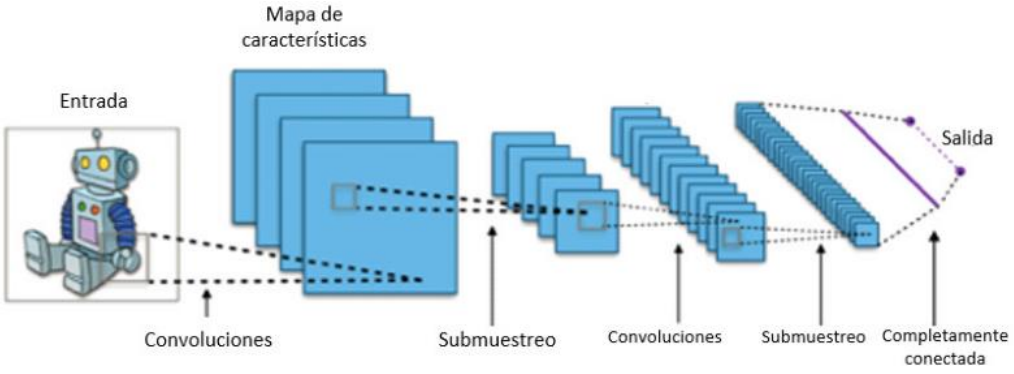


Figura 4. Arquitectura de una CNN (Pacheco, 2017).

2.8. Infrarrojo cercano

Dentro del espectro electromagnético, la radiación infrarroja se encuentra comprendida entre el espectro de luz visible y las microondas (González, 2017). El infrarrojo cercano (NIR) es un subconjunto de la banda infrarroja del espectro electromagnético, que cubre las longitudes de onda que van desde los 0.78 a 1.1 micrones.

Esta longitud de onda esta fuera del rango de lo que los humanos pueden ver y, a veces, ofrece detalles más claros que los que se pueden lograr con imágenes de luz visible (Tokkie & Hinner, 2006).

En la Figura 5 podemos observar el espectro electromagnético así como en dónde se sitúa la banda del infrarrojo; en la Figura 6 vemos la diferencia entre una imagen de luz visible y una imagen de infrarrojo cercano.

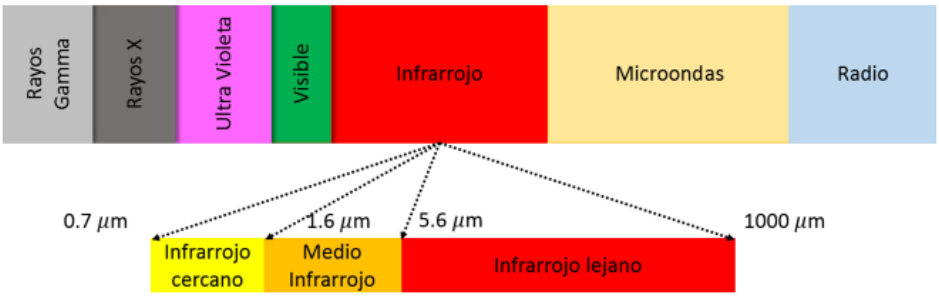


Figura 5. Espectro electromagnético, (González, 2017).

Figura 6. Imagen luz visible / Imagen infrarroja, (RGB-NIR Data, n.d.).

2.9. Detección de imágenes

La detección de imágenes mediante algoritmos de Machine Learning implica una CNN que detecta una cantidad limitada (o específica) de objetos, no pudiendo detectar objetos que antes no hubiese visto (Aprende Machine Leaning, 2020). Un algoritmo de Machine Learning de detección, para considerarse como tal, deberá:

- Detectar múltiples objetos.
- Dar la posición X e Y del objeto en la imagen

2.10. YOLO

Uno de los mejores algoritmos para procesar imágenes en tiempo real se llama YOLO (You Only Look Once). Este algoritmo procesa la imagen con una CNN. La red neuronal YOLO puede predecir diferentes objetos en una imagen.

YOLO-v4 es un algoritmo de detección de objetos de una etapa de alta precisión realizando el procesamiento en tiempo real; este algoritmo integra las características de sus antecesores, YOLO-v1, YOLO-v2 y YOLO-v3. YOLO-v4 logra mejorar la velocidad y precisión de la detección. Los componentes del detector yolov4 se pueden observar en la Figura 7.

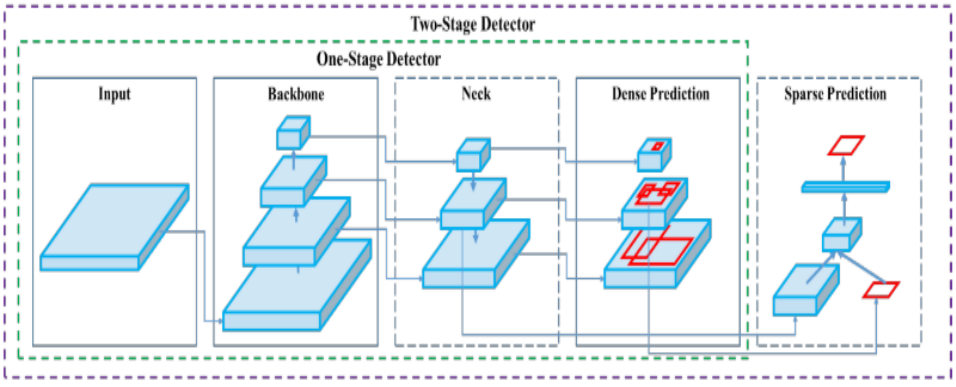


Figura 7. Arquitectura de un detector (Gutta, 2021).

En general, existen dos tipos de algoritmos de detección de objetos: de una etapa o de dos etapas. YOLOv4 es un detector de una etapa, es decir, lo cual nos da detecciones más rápidas. Los principales componentes de YOLOv4 es el *backbone* que es la parte encargada de la extracción de características esenciales. La siguiente parte es el *neck***,** cuya tarea es recopilar mapas de características formados en la etapa anterior para así prepararse para la detección. Por último, tenemos el componente *head,* que para el caso de un detector de una etapa se trata de realizar una predicción densa. Esta predicción densa es la predicción final que se compone de un vector que contiene las coordenadas del cuadro delimitador (centro, alto, ancho), el porcentaje de confiabilidad de la predicción y la etiqueta.

2.11. Detección de rostros

El término “detección de rostro” hace referencia a los mecanismos computacionales mediante los cuales es posible determinar si existe o no un rostro en una imagen digital dada; cuando se trata de localizar o detectar un rostro el grado de dificultad se eleva dado que los rostros pueden encontrarse en un amplio rango de poses, con una infinidad de gestos y detalles visuales que modifican su apariencia; todo aquello que visualmente ayuda a nuestro cerebro a distinguir entre una persona y otra, resulta en un grado extra de complejidad en el caso de la localización facial mediante el procesamiento de imágenes y visión por computadora (Montiel, 2019).

2.12. Métricas para detección de objetos

2.12.1. mAP

La precisión media (mAP) es una métrica utilizada para evaluar modelos de detección de objetos.

La fórmula de mAP se basa en los siguientes conceptos (LearnOpenCv, n.d.):

- Intersección sobre unión (IoU)
- Matriz de confusión

- Exhaustividad
- Precisión

2.12.2. IoU

Intersección sobre unión (IoU) es un número que cuantifica el grado de superposición entre dos cajas. En el caso de la detección y segmentación de objetos, IoU evalúa la superposición de la región verdadera y la predicción. Esta es una métrica que nos ayuda a medir la exactitud de una predicción. En la Figura 8 se observa la manera en la que se calcula.

Los valores de IoU varían de 0 a 1. Donde 0 significa que no hay superposición y 1 significa superposición perfecta. Con la ayuda del umbral de IoU, podemos decidir si la predicción es Verdadero Positivo (TP), Falso Positivo (FP) o Falso Negativo (FN) (LearnOpencv, 2022).

La Figura 9 nos muestra diferentes valores de IoU con sus correspondientes predicciones.

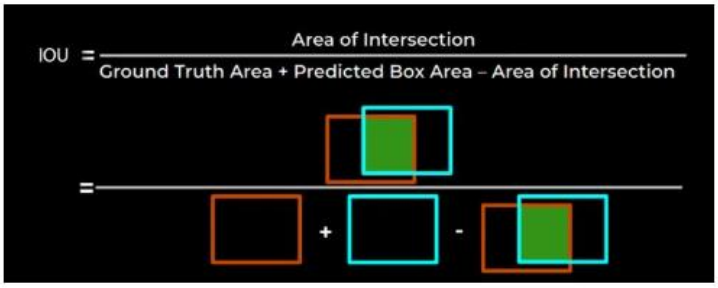


Figura 8. Intersección sobre unión (LearnOpencv, 2022).

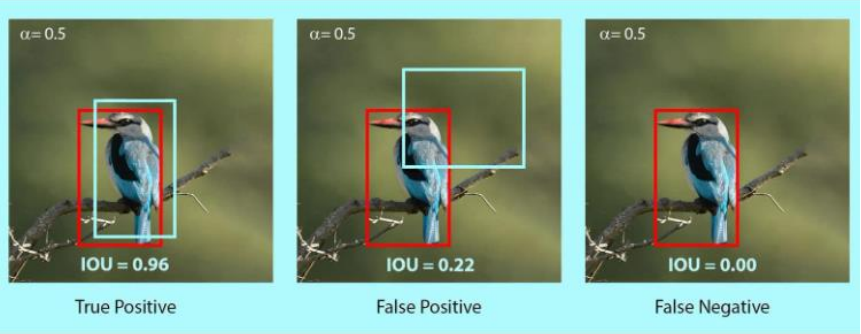


Figura 9. Diferentes valores de IoU (LearnOpencv, 2022).

2.12.3. Matriz de confusión

Una matriz de confusión es una herramienta que permite visualizar el desempeño de nuestro algoritmo. Para poder obtener la matriz de confusión, se necesita saber que son los verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. A continuación, se muestra una definición para cada uno de estos conceptos.

- Verdaderos Positivos (VP): corresponden a aquellos eventos que son verdaderos y se predijeron como verdaderos.
- Verdaderos Negativos (VN): son aquellos eventos falsos que se predicen como falsos.
- Falsos Positivos (FP): son valores negativos que el modelo detecta como positivos
- Falsos Negativos (FN): ocurren cuando tenemos valores positivos, pero que se predicen como negativos.

En la Figura 10 se muestra cómo es una matriz de confusión.

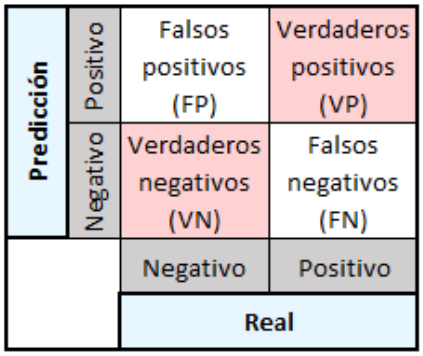


Figura 10. Matriz de confusión (Imagen propia).

2.12.4. Exactitud

La métrica ***exactitud*** representa el porcentaje total de valores correctamente clasificados, tanto positivos como negativos.

$$exactitud = \frac{VP+VN}{VP+VN+FP+FN} \qquad (3)$$

Es recomendable utilizar esta métrica en problemas en los que los datos están balanceados, es decir, que haya misma cantidad de valores de cada etiqueta (Diaz, n.d.).

2.12.5. Precisión

La métrica de ***precisión*** es utilizada para poder saber qué porcentaje de valores que se han clasificado como positivos son realmente positivos (Diaz, n.d.). Es decir, con esta métrica podemos medir la calidad del modelo.

$$precisión = \frac{VP}{VP+FP} \qquad (4)$$

2.12.6. Exhaustividad

La métrica de ***exhaustividad*** nos va a informar sobre la cantidad de valores positivos son correctamente clasificados por el modelo (J. Martinez, 2020).

$$exhaustividad = \frac{VP}{VP+FN} \qquad (5)$$

2.12.7. Valor F1

Esta es una métrica muy utilizada en problemas en los que el conjunto de datos a analizar está desbalanceado. Esta métrica combina la precisión y el recall, para obtener un valor mucho más objetivo (Diaz, n.d.).

$$valor\ F1 = 2 \cdot \frac{precisión \cdot exhaustividad}{precisión + exhaustividad} \quad (6)$$

2.13. Lenguaje de programación Python

Python es un lenguaje de programación muy usado en inteligencia artificial, ya que nos ofrece la flexibilidad necesaria y es un lenguaje con una curva de aprendizaje muy rápido. Python tiene la ventaja de ser un lenguaje interpretado y no necesita una compilación previa, lo que facilita el ciclo de programación y pruebas. Es un lenguaje de propósito general que puede utilizarse también para crear complejos programas orientados a objetos y con interfaces gráficas de usuario (García, 2012).

2.14. Google Colaboratory

Google Colaboratory, o "Colab" para abreviar, es un producto de Google Research similar a Jupyter Notebook. Permite a cualquier usuario escribir y ejecutar código arbitrario de Python en el navegador. Es especialmente adecuado para tareas de aprendizaje automático, análisis de datos y educación. Desde un punto de vista más técnico, Colab es un servicio de cuaderno alojado de Jupyter que ofrece acceso sin coste adicional a recursos informáticos, como GPU’s (Google, n.d.).

Para usar Colab, no necesita instalar y ejecutar ni actualizar el hardware de la computadora para cumplir con los requisitos intensivos de carga de trabajo de CPU/GPU de Python. Además, Colab brinda acceso gratuito a la infraestructura informática como almacenamiento, memoria, capacidad de procesamiento, unidades de procesamiento de gráficos (GPU) y unidades de procesamiento de tensor (TPU). (Geekflare, 2022)

Capítulo 3. Metodología

La metodología que se propone para el desarrollo del presente proyecto se muestra en la Figura 6, donde se puede apreciar las diferentes etapas a seguir. A continuación, se explicará cada una de estas etapas.

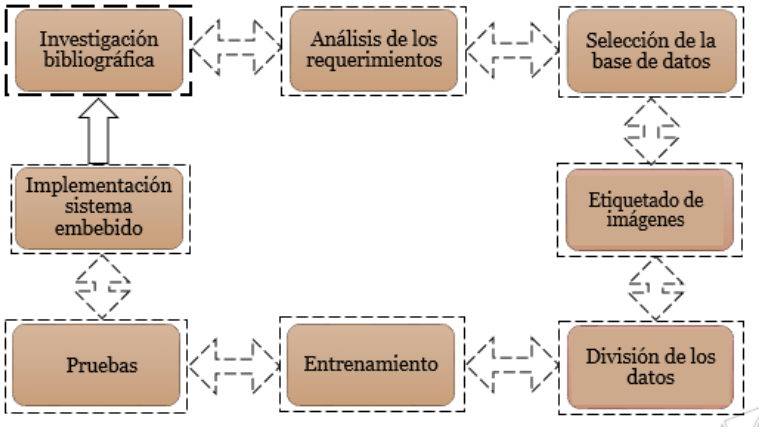


Figura 11. Diagrama general de la metodología (Imagen propia).

3.1. Investigación bibliográfica

Primeramente, se realiza una investigación bibliográfica, buscando literatura relacionada con detección de somnolencia en conductores, principalmente en ambientes nocturnos o de poca iluminación utilizando imágenes de tipo infrarrojo cercano. Algunos de los trabajos revisados en esta investigación se pueden ver en la Tabla 1 del Capítulo 1.

3.2. Análisis de los requerimientos

Seguido de la investigación, se realiza el análisis de los requerimientos que tiene que cumplir el proyecto, que se enlistan a continuación:

- Realizar la detección de somnolencia con imágenes infrarrojas.
- Implementación en un sistema embebido.

3.3. Selección de la base de datos

Lo siguiente es la selección de la base de datos, una dificultad que se tiene es que no hay suficientes bases de datos públicas, sin embargo, se propone una base de datos con 710 imágenes de tipo infrarrojo cercano. Esta base de datos es la Drozy database (Massoz et al., 2016), que contiene imágenes como las que se muestran en la Figura 12, en las que tenemos rostros con signos de somnolencia y sin somnolencia.

Figura 12. Ejemplo de las imágenes de la base de datos Drozy Database (Massoz et al., 2016).

3.4. Etiquetado de las imágenes

El etiquetado de la base de datos se realiza en el software LabelImg. El etiquetado consiste en delimitar en la imagen un cuadrado el cual contenga un rostro con o sin somnolencia.

El primer paso para poder realizar el etiquetado es tener todas las imágenes de la base de datos en una carpeta, como lo podemos ver en la Figura 13. Para este proyecto, se tienen dos clases, *Somnolencia* y *No Somnolencia*. Una vez que se tienen las imágenes en la carpeta, se le tiene que indicar al software el tipo de detector para el cual se van a realizar las etiquetas, para este proyecto YOLO, pues dependiendo del detector, serán los archivos que nos dé como resultado el software después del etiquetado; en la Figura 14 se puede ver la opción donde se puede cambiar el tipo de detector que se usará. Posteriormente, se procede a etiquetar cada una de las 710 imágenes en alguna de las dos clases, realizando un cuadro que delimite el rostro con o sin somnolencia con la herramienta *Create RectBox* que se muestra en la Figura 14. En la Figura 15 podemos ver del lado izquierdo las herramientas que tiene el software, en el centro la imagen en la que se realizó el cuadro delimitador que contiene la clase que queremos etiquetar, en la parte

superior derecha tenemos la clase a la que pertenece la imagen y en la parte inferior derecha tenemos la lista de todas las imágenes que contiene la carpeta, en este caso son todas las imágenes de la base de datos.

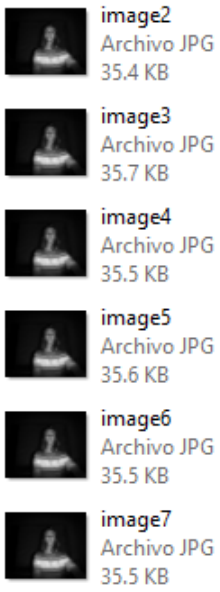


Figura 13. Imágenes de la base de datos dentro de una misma carpeta.

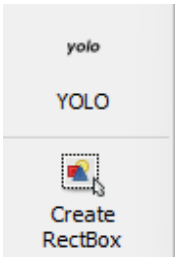


Figura 14. Herramientas del software LabelImg.

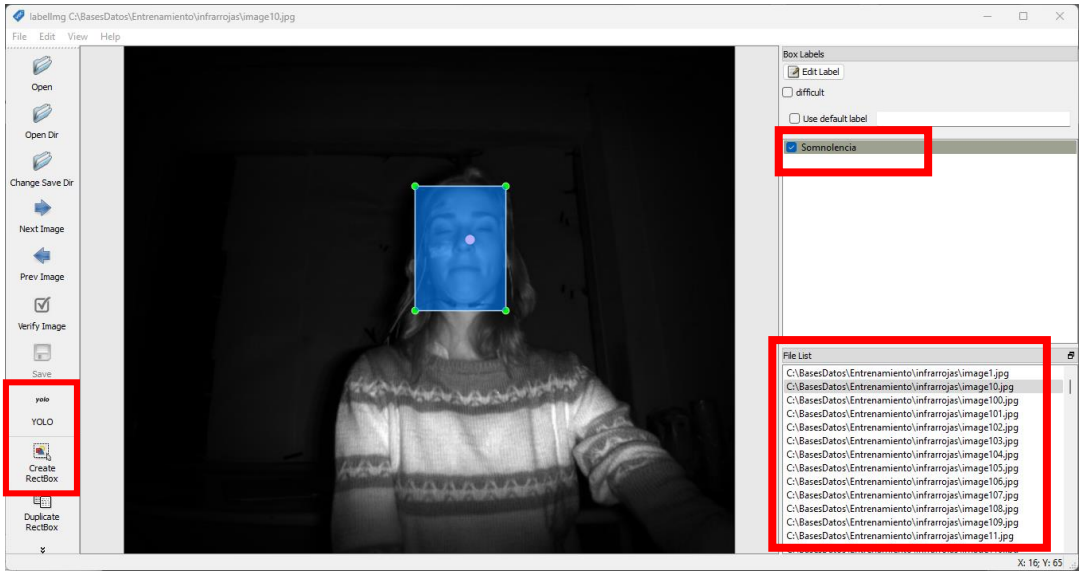


Figura 15. Etiquetado de una imagen en la clase Somnolencia en el software LabelImg.

El software LabelImg nos entrega los archivos necesarios que más adelante van a ser utilizados para poder entrenar nuestro detector. Al final del etiquetado los archivos que vamos a tener son los que se muestran en la Figura 16; el archivo clases.txt contiene el nombre de las clases que se encuentran en las imágenes, como podemos ver en la Figura 17. También se genera un archivo de texto por cada imagen (image1.txt, image2.txt, …) que contiene la clase a la que pertenece dicha imagen y las coordenadas del cuadro delimitador donde se encuentra el rostro con somnolencia o no somnolencia. Un ejemplo de esto se observa en la Figura 18, donde tenemos el contenido del archivo image3.txt con su respectiva imagen.

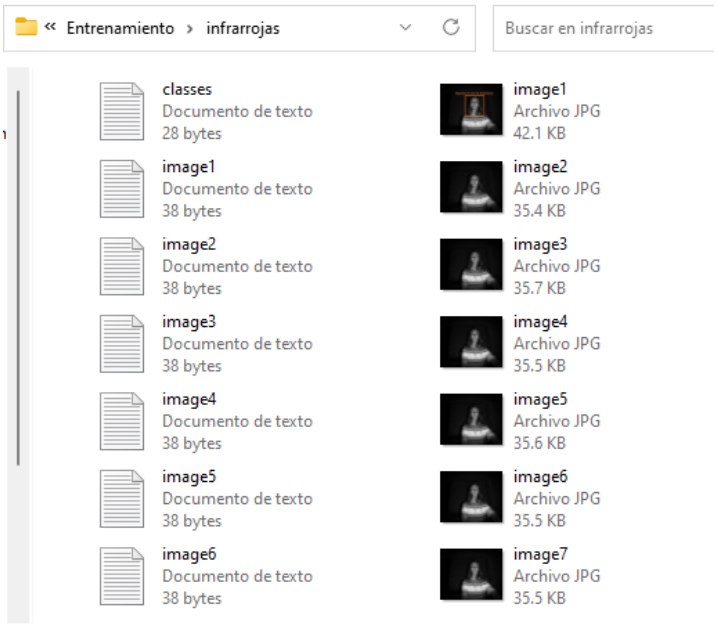


Figura 16. Archivos generados después de realizar el etiquetado.

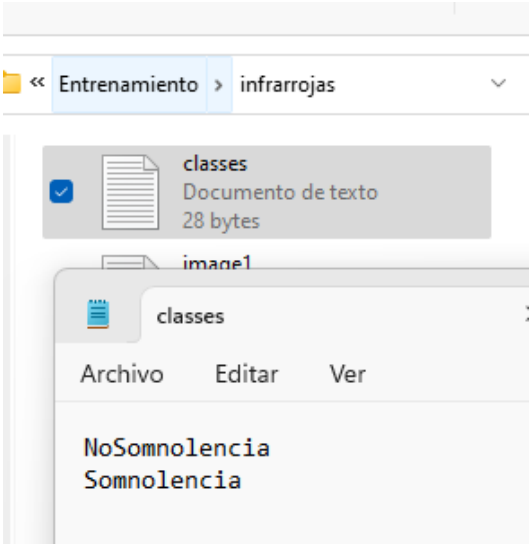


Figura 17. Contenido del archivo classes.txt.

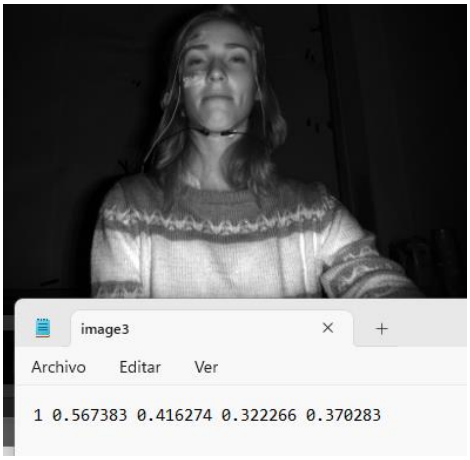


Figura 18. Contenido del archivo image3.txt con su respectiva imagen.

La Figura 19 y Figura 20 muestran un ejemplo del etiquetado que se realiza en cada una de las imágenes que contiene la base de datos.

Figura 19. Imagen etiquetada en clase No Somnolencia.

Figura 20. Imagen etiquetada en clase Somnolencia.

3.5. División de los datos

Ahora, se realiza una división de las imágenes de la base de datos en tres conjuntos. Se toma el 80% de los datos para el entrenamiento, 15% para validación y el 5% restante para realizar las pruebas al algoritmo. Esto se realiza mediante un código en lenguaje Python, el cual nos crea las tres carpetas diferentes con las imágenes correspondientes; una parte de este código se muestra en el Código 1.

```
#Importamos la libreria para el manejo de archivos
import os
import random
from shutil import copyfile
import shutil
def img_train_test_split(img_source_dir, train_size, validation_size):
    """
    Parametros
    ----------
    img_source_dir : string
    Directorio de las imagenes

    train_size : float
    Porcentaje de la muestra de entrenamiento, ejemplo 0.80 (80%)

    validation_size : float
    Porcentaje de la muestra de validación, ejemplo 0.15 (15%)

    El restante 5% quedaria para el numero de imagenes de prueba,
imagenes que nunca ha visto el modelo
    """
    # Configurar la estructura de carpetas vacías si no existe
    if not os.path.exists('dataset'):
        os.makedirs('dataset')
    else :
        shutil.rmtree('dataset')

    subdir_fullpath = img_source_dir

    if len(os.listdir(subdir_fullpath)) == 0:
        print(subdir_fullpath + ' is empty')

    train_subdir = 'dataset/train'
    validation_subdir = 'dataset/valid'
    test_subdir = 'dataset/test'
    # Cree subdirectorios en carpetas de entrenamiento, validación y
test
    if not os.path.exists(train_subdir):
```

```
        os.makedirs(train_subdir)

    if not os.path.exists(validation_subdir):
        os.makedirs(validation_subdir)

    if not os.path.exists(test_subdir):
        os.makedirs(test_subdir)
    train_counter = 0
    validation_counter = 0
    test_counter = 0
    # Contar el número de imagenes totales
    count_images=0
    for filename in os.listdir(subdir_fullpath):
        if filename.endswith(".jpg"):
            count_images+= 1
    print(count_images)

    total_images=count_images
    count_images=0

    #Ordenar de manera aleatoria las imagenes
    list_files=os.listdir(subdir_fullpath)
    random.shuffle(list_files)


```

Código 1. Parte del código en Python que se usa para dividir las imágenes de la base de datos.

Al ejecutar el código en Python del Código 1, se crea una carpeta llamada *dataset,* que contiene tres carpetas, *test, train y valid* que contienen las imágenes en los porcentajes anteriormente descritos.

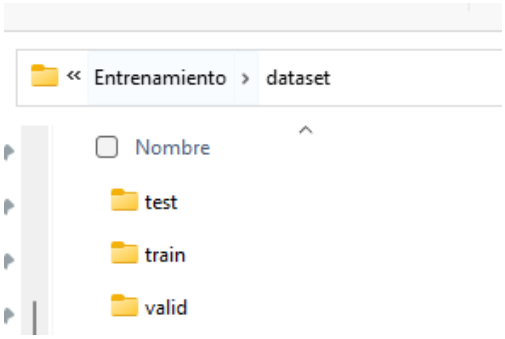


Figura 21. Division de las imágenes en carpetas.

3.6. Google Colab y Google Drive

Ahora, es momento de preparar el ambiente en google drive para posteriormente crear el código para el entrenamiento en google colab. Primero, se crea la carpeta *data* y dentro de ésta se suben las carpetas creadas en la sección 3.5. En la Figura 22, Figura 24 y Figura 25 se aprecia cómo se vería la carpeta *data* y las carpetas para entrenamiento, validación y prueba, con sus imágenes y archivos correspondientes creados en el proceso de etiquetado de la sección 3.4.

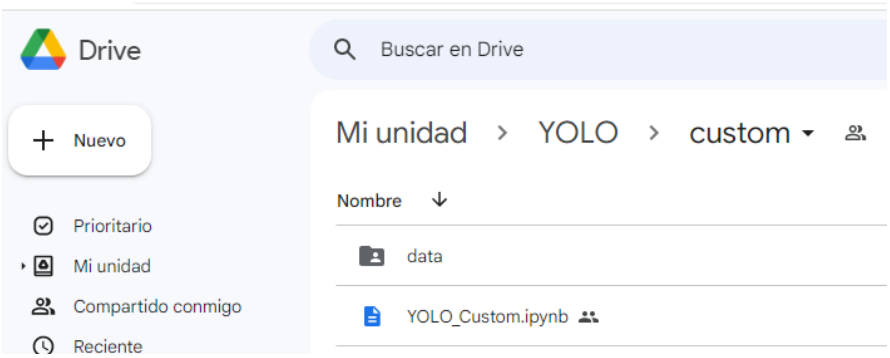


Figura 22. Carpeta que contiene los archivos para el proyecto.

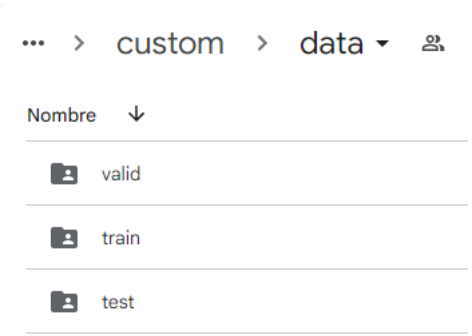


Figura 23. Carpeta con los datos divididos.

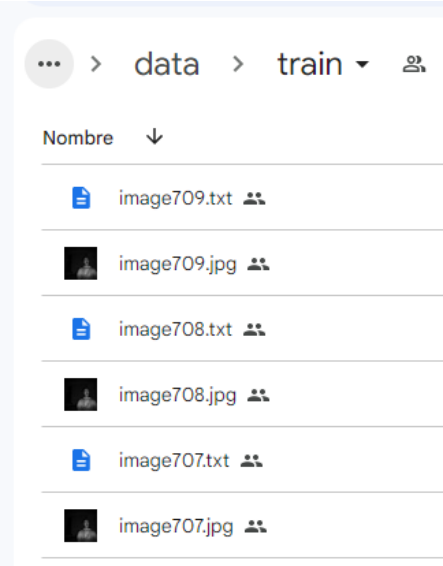


Figura 24. Carpeta con imágenes y archivos para el entrenamiento del detector.

Después, se crea el archivo ".ipynb", que es el archivo que se utiliza para realizar el entrenamiento del detector, el cual se explica más a detalle en la siguiente sección.

3.7. Entrenamiento

Para realizar el entrenamiento del algoritmo, se sigue el diagrama que se presenta en la Figura 25, que son los pasos para la configuración del ambiente para la implementación de YOLO-v4, usando Google Colab.

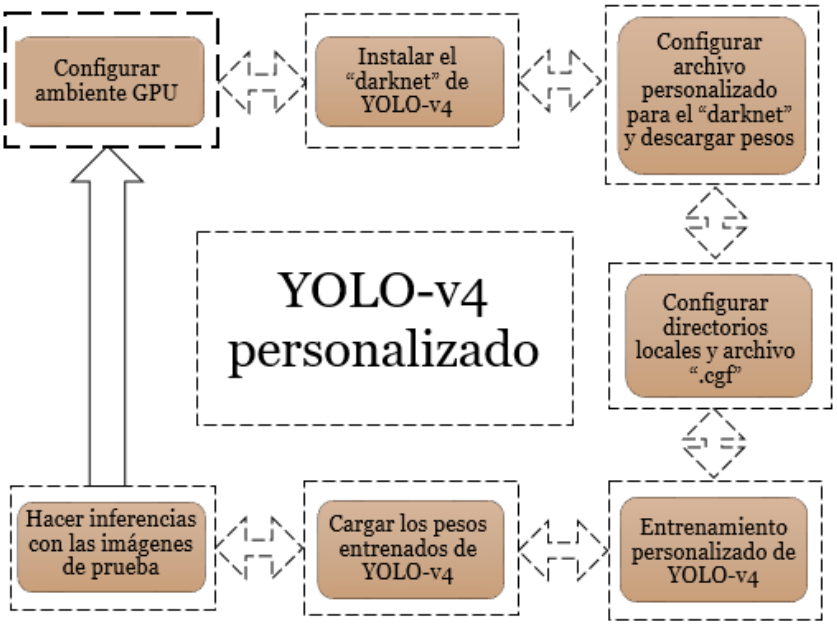


Figura 25. Diagrama de la personalización de YOLO-v4 (Imagen propia).

3.7.1. Configuración del ambiente GPU en Google Colab

Lo primero es crear el archivo con extensión ".ipynb" y permitir el acceso a nuestro Google Drive. Así como también cambiar el entorno de ejecución a GPU, esto para poder disponer de mayor capacidad de procesamiento. En el Código 2 y en la Figura 26 se muestra lo anterior.

```
1  # Cargamos nuestro drive en el notebook de trabajo
2  from google.colab import drive
3  drive.mount('/content/drive')
```

Código 2. Acceso a Google Drive desde Google Colab.

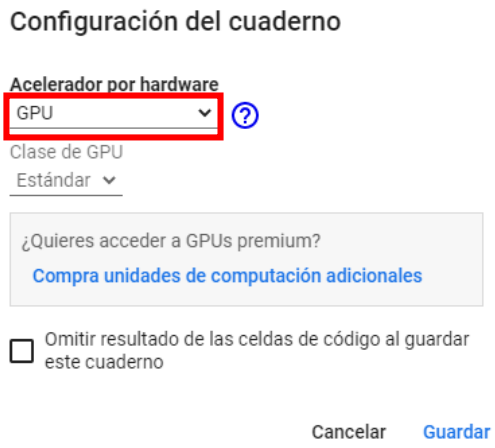


Figura 26. Entorno de ejecución GPU.

3.7.2. Instalación del darknet

Después, para poder usar YOLOv4, se clona el repositorio de GitHub "darknet" usando el código mostrado en Código 3. También, se configura el espacio de trabajo dentro de Google Colab.

```
# Descargamos darknet
%cd /content/
%rm -rf darknet
!git clone https://github.com/roboflow-ai/darknet.git
%cd /content/darknet/
%rm Makefile
```

Código 3. Descarga del darknet desde el repositorio de GitHub.

3.7.3. Configuración del archivo personalizado para el darknet y descarga de los pesos pre-entrenados

Ahora es momento de configurar el archivo "make" para hacer un uso más eficiente de YOLO, asegurando que OpenCV tenga compatibilidad con CUDA y GPU. En el Código 4 se muestra parte de esta configuración y en el Código 5 las instrucciones para crear el archivo ejecutable.

```
#Configuración del archivo Makefile
%%writefile Makefile
GPU=1
```

```
CUDNN=1
CUDNN_HALF=0
OPENCV=1
AVX=0
OPENMP=0
LIBSO=1
ZED_CAMERA=0
ZED_CAMERA_v2_8=0
USE_CPP=0
DEBUG=0
```

Código 4. Parte del código para la configuración del archivo “make”.

```
#Creamos el ejecutable
%cd /content/darknet
!make
```

Código 5. Creación del archivo ejecutable.

En seguida, con el Código 6, se descargan los pesos, esto se realiza para asegurarnos de que nuestro detector de objetos sea preciso, utilizando pesos previamente entrenados en lugar de una inicialización de pesos aleatoria. También ayuda a que el modelo converja rápidamente, reduciendo así el tiempo de entrenamiento.

```
#Descargamos los pesos
%cd /content/darknet
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_
yolo_v3_optimal/yolov4.conv.137
```

Código 6. Descarga de los pesos pre-entrenados.

3.7.4. Configuración de los directorios locales y del archivo “.cfg”

En seguida, usando el Código 7, se configuran los directorios locales que contienen las imágenes para el entrenamiento, así como los archivos con las etiquetas correspondientes a estas imágenes.

```
#Configurar directorios de archivos de entrenamiento para conjuntos de
datos personalizados
%cd /content/darknet/
%cp train/_darknet.labels data/obj.names
%mkdir data/obj
#copy image and labels
%cp train/*.jpg data/obj/
```

```
%cp valid/*.jpg data/obj/

%cp train/*.txt data/obj/
%cp valid/*.txt data/obj/

with open('data/obj.data', 'w') as out:
  out.write('classes = 1\n')
  out.write('train = data/train.txt\n')
  out.write('valid = data/valid.txt\n')
  out.write('names = data/obj.names\n')
  out.write('backup = backup/')

#write train file (just the image list)
import os

with open('data/train.txt', 'w') as out:
  for img in [f for f in os.listdir('train') if f.endswith('jpg')]:
    out.write('data/obj/' + img + '\n')

#write the valid file (just the image list)
import os

with open('data/valid.txt', 'w') as out:
  for img in [f for f in os.listdir('valid') if f.endswith('jpg')]:
    out.write('data/obj/' + img + '\n')
```

Código 7. Configuración de directorios locales en espacio de trabajo.

Una vez hecho esto, se configura el archivo “.cfg” con las especificaciones para este proyecto, es decir los hiperparámetros como lo son la tasa de aprendizaje, el batch size, y también el número de capas que tendrá la red y el número de clases que se tienen. En el Código 8 se observa una parte del código que se usa para la configuración y en el Código 9 algunas de las especificaciones que tiene la red.

```
#we build config dynamically based on number of classes
#we build iteratively from base config files. This is the same file shape as cfg/yolo-obj.cfg
def file_len(fname):
  with open(fname) as f:
    for i, l in enumerate(f):
      pass
  return i + 1

num_classes = file_len('train/_darknet.labels')
print("Número de clases: " + str(num_classes))

```

```
#Instructions from the darknet repo
#change line max_batches to (classes*2000 but not less than number of
training images, and not less than 6000), f.e. max_batches=6000 if you
train for 3 classes
#change line steps to 80% and 90% of max_batches, f.e. steps=4800,5400
if os.path.exists('./cfg/custom-yolov4-detector.cfg'):
os.remove('./cfg/custom-yolov4-detector.cfg')


with open('./cfg/custom-yolov4-detector.cfg', 'a') as f:
  f.write('[net]' + '\n')
  f.write('batch=64' + '\n')
  #####smaller subdivisions help the GPU run faster. 12 is optimal, but
you might need to change to 24,36,64####
  f.write('subdivisions=24' + '\n')
  f.write('width=416' + '\n')
  f.write('height=416' + '\n')
  f.write('channels=3' + '\n')
  f.write('momentum=0.949' + '\n')
  f.write('decay=0.0005' + '\n')
  f.write('angle=0' + '\n')
  f.write('saturation = 1.5' + '\n')
  f.write('exposure = 1.5' + '\n')
  f.write('hue = .1' + '\n')
  f.write('\n')
  f.write('learning_rate=0.001' + '\n')
  f.write('burn_in=1000' + '\n')

```

Código 8. Configuración del archivo ".cfg" para entrenamiento personalizado.

```
#aquí está el archivo que se acaba de escribir.
%cat cfg/custom-yolov4-detector.cfg
```

Código 9. Muestra el archivo ".cfg".

```
[net]
batch=64
subdivisions=24
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue = .1

learning_rate=0.001
```

```
burn_in=1000
max_batches=4000
policy=steps
steps=3200.0,3600.0
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=608
height=608
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
```

Código 10. Parte del archivo con las especificaciones de la red para el detector.

3.7.5. Entrenamiento personalizado de YOLOv4

Una vez que se tiene completa la personalización del modelo de detección de somnolencia, es momento de comenzar con el entrenamiento. En el Código 11 se puede ver la línea que se ejecuta para iniciar el entrenamiento.

```
!. /darknet detector train data/obj.data cfg/custom-yolov4-detector.cfg
yolov4.conv.137 -dont_show -map
```

Código 11. Instrucción para realizar el entrenamiento personalizado de YOLO.

3.7.6. Cargar los pesos entrenados

El entrenamiento se realiza con 569 imágenes y el tiempo aproximado que dura este entrenamiento son 3 horas. Al final del entrenamiento, se van a tener tres archivos, uno con los pesos más óptimos, uno con los pesos después de las 1000 iteraciones y un archivo más con los pesos de la última iteración. Estos archivos, se guardan en Google Drive, para poder utilizarlos posteriormente, o en el caso de este proyecto, cuando se implemente en

el sistema embebido no se tenga que volver a entrenar el modelo. En la Figura 27 se observan los archivos en la carpeta de Drive.

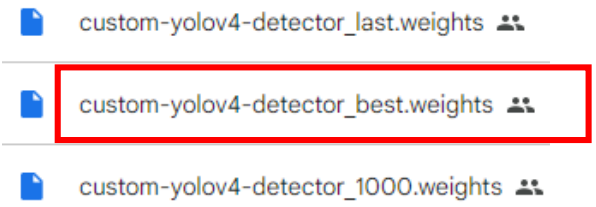


Figura 27. Archivos con los pesos después del entrenamiento.

3.7.7. Realizar inferencias / pruebas

Para realizar las inferencias con las imágenes que se tienen en la carpeta de Test, primero se crea la función que nos va a permitir desplegar las imágenes. En el Código 12 se muestra dicha función.

```
#Definimos la función que nos desplegará las imagenes
def imShow(path):
  import cv2
  import numpy as np
  import matplotlib.pyplot as plt
  %matplotlib inline
  #image = cv2.imdecode(np.fromfile(path, dtype=np.uint8),
cv2.IMREAD_UNCHANGED)

  image = cv2.imread(path)
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation =
cv2.INTER_CUBIC)

  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  #plt.rcParams['figure.figsize'] = [10, 5]
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()
```

Código 12. Función para mostrar las imágenes.

Ahora es momento de probar cada imagen y ver como realiza la detección nuestro detector de somnolencia ya entrenado. Para esto se realiza el Código 13, en el cual se tiene un ciclo “for" que ingresa cada imagen al detector.

```
# Utilizamos las imagenes de la carpeta /test para probar nuestro modelo

contador = 1
for i in range(34):
  test_images = [f for f in os.listdir('test') if f.endswith('.jpg')]
  image=test_images[i]
  img_path = "test/" + image;
  #test out our detector!
  #print(backup/custom-yolov4-detector_best.weights {img_path})
#hiperparámetro
  #print(predictions)
  !./darknet detect cfg/custom-yolov4-detector.cfg backup/custom-yolov4-
detector_best.weights {img_path}  -thresh 0.3 -dont-show
  imShow('/content/darknet/predictions.jpg')
  print("********************")
  print("*****",image,"*****")
  print("********************")
#Para guardar las imágenes en una carpeta
#!cp predictions.jpg  "/content/drive/My
Drive/YOLO/custom/data/image710_92Somn_detectado.jpg"
```

Código 13. Prueba de cada imagen en el detector de somnolencia.

3.8. Implementación en sistema embebido

3.8.1. Instalación del darknet

Lo primero es realizar la instalación del darket, para esto, se clona el repositorio por medio del comando del Código 14 desde la terminal de Ubuntu (sistema operativo de la tarjeta Jetson), como se muestra en la Figura 28.

```
git clone https://github.com/AlexeyAB/darknet.git
```

Código 14. Descarga del darknet desde el repositorio de GitHub para Linux.

Figura 28. Descarga del darknet desde la terminal de Ubuntu.

3.8.2. Configuración del archivo personalizado para el darknet

Una vez clonado el repositorio, se realizan algunos cambios al archivo “make”. En la Figura 29 se observan algunos de estos cambios. También se corre la instrucción “make” para crear el archivo ejecutable, lo cual podemos verlo en la Figura 30

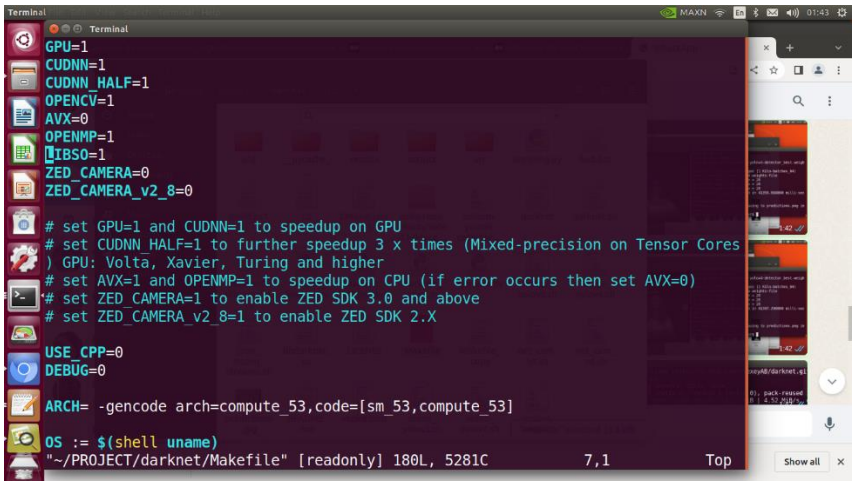


Figura 29. Parte del código para la configuración del archivo “make” en Ubuntu.

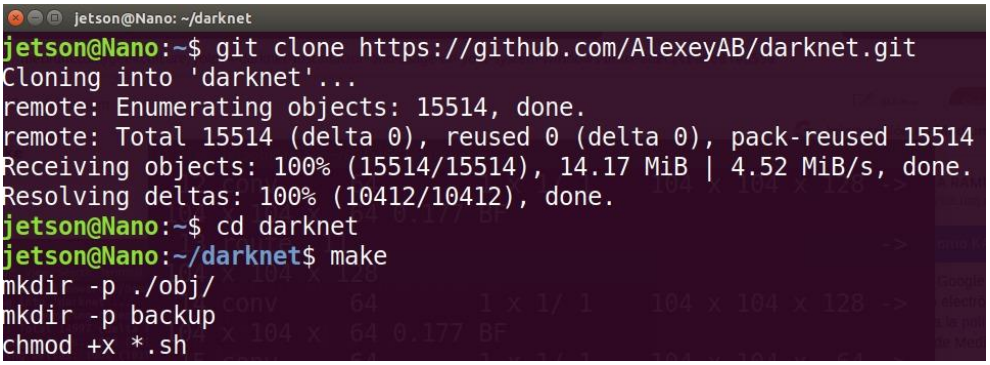


Figura 30. Creación del archivo ejecutable.

3.8.3. Copiar los archivos ".cfg", ".data" y ".names"

El siguiente paso es copiar algunos archivos al directorio donde estamos trabajando con el detector en la tarjeta Jetson; se copia el Código 10 que es el archivo ".cfg", este archivo contiene detalles sobre el detector; también, copiamos el archivo "objcustom.data" y "obj.names" que contienen el número de clases y el nombre de las clases, respectivamente. En la Figura 31 y Figura 32 observamos dichos archivos.

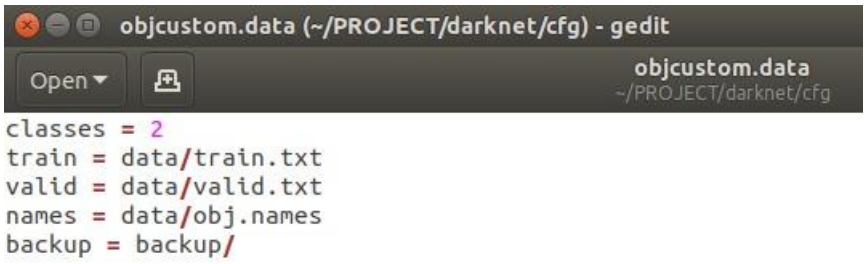


Figura 31. Archivo "objcustom.data".

Figura 32. Archivo "obj.names".

3.8.4. Pruebas del detector en sistema embebido

Ahora es momento de realizar pruebas del detector en el sistema embebido, para esto, ejecutamos el Código 15 en la terminal de Ubuntu.

```
1  ./darknet detector test cfg/objcustom.data cfg/custom-yolov4-detector.cfg
   custom-yolov4-detector_best.weights data/Img95.jpg -gpus 0
```

Código 15. Instrucción para probar una imagen en el detector.

En el Capítulo 4 se muestran las métricas obtenidas en el entrenamiento así como las inferencias con las imágenes de prueba utilizando los pesos óptimos, tanto en Google Colab como en el sistema embebido.

Capítulo 4. Resultados

4.1. Métricas en el entrenamiento

La Tabla 3 muestra algunas de las métricas obtenidas durante el entrenamiento, dependiendo del número de épocas. El entrenamiento se detiene a las 1500 épocas, debido a los resultados favorables para la métrica mAP.

La Figura 33 muestra el desempeño que tiene el modelo durante el entrenamiento. La línea roja representa la métrica mAP y los puntos azules la función de pérdida.

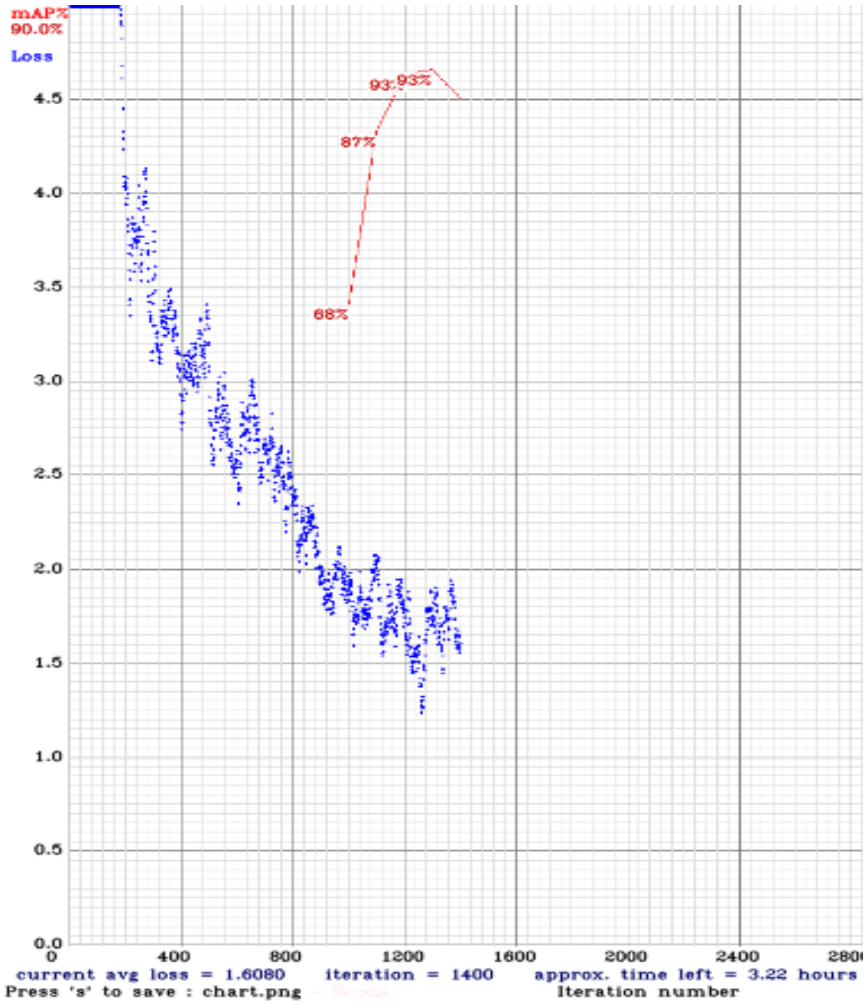


Figura 33. Desempeño durante el entrenamiento (Imagen propia).

Tabla 3. Métricas durante el entrenamiento.

Época	Precisión	Exhaustividad	Valor F1	mAP@0.5
1000	61%	76%	68%	68.24%
1100	85%	94%	89%	86.52%
1200	85%	94%	90%	92.54%
1300	80%	92%	85%	93.11%
1400	83%	91%	87%	90%
1500	80%	91%	85%	93.37%

4.2. Pruebas

Una vez realizado el entrenamiento se prueba el desempeño del algoritmo con las imágenes que se dejaron en la carpeta para pruebas. Se prueba el detector previamente entrenado para detectar somnolencia con 35 imágenes que el detector no ha visto antes. Algunas de estas imágenes se muestran a continuación.

En las siguientes figuras, se aprecian algunas imágenes que se sometieron a prueba y la clase a la que pertenecen, ya sea somnolencia o no somnolencia y además se muestra el porcentaje de fiabilidad de la clase a la que pertenecen. El tiempo aproximado que el detector tarda en procesar cada imagen es 44 milisegundos.

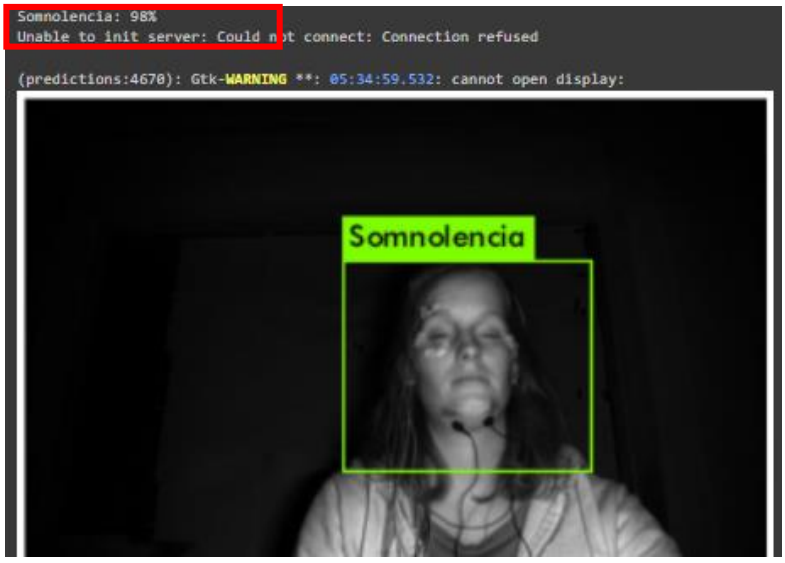


Figura 34. Detección de la clase somnolencia en un 98%.

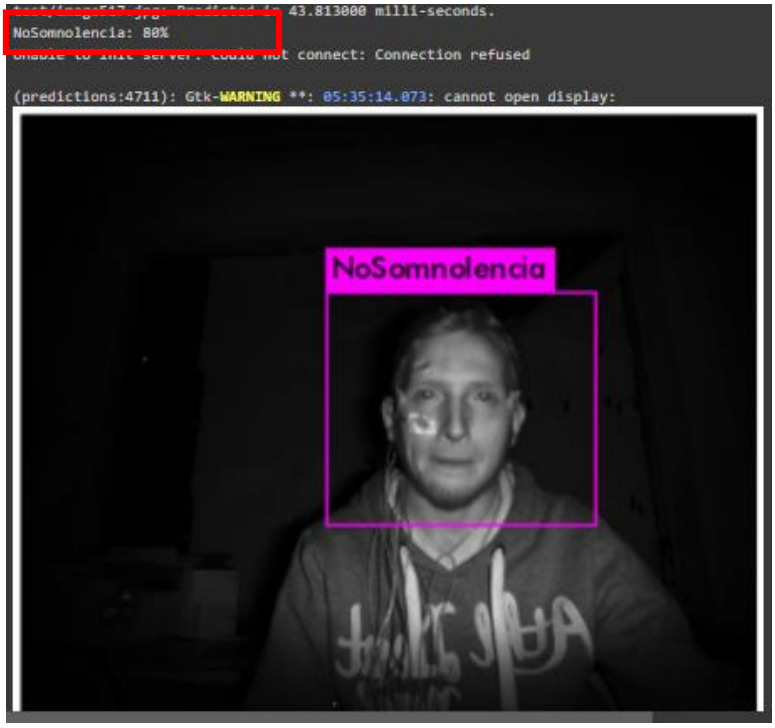


Figura 35. Detección de la clase no somnolencia en un 80%.

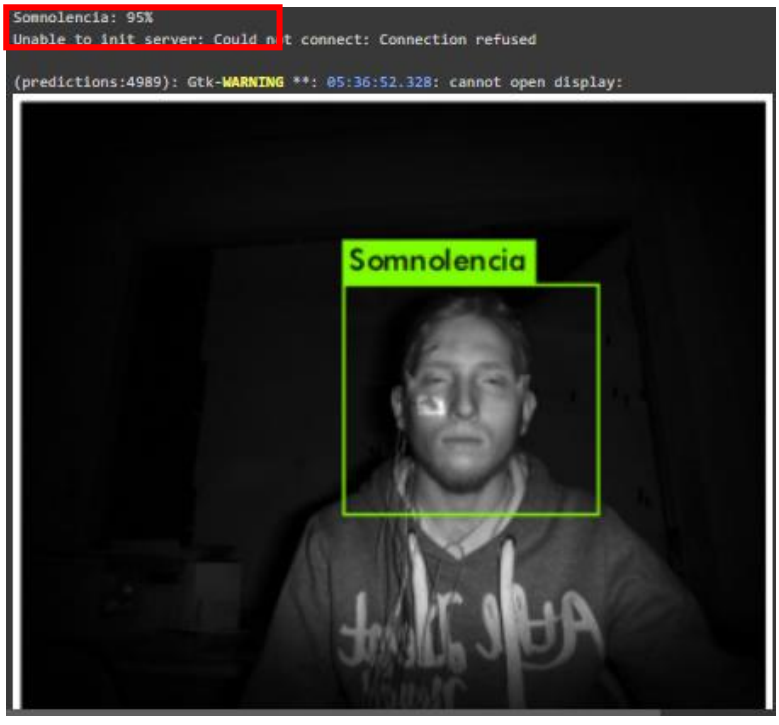


Figura 36. Detección de la clase somnolencia en un 95%.

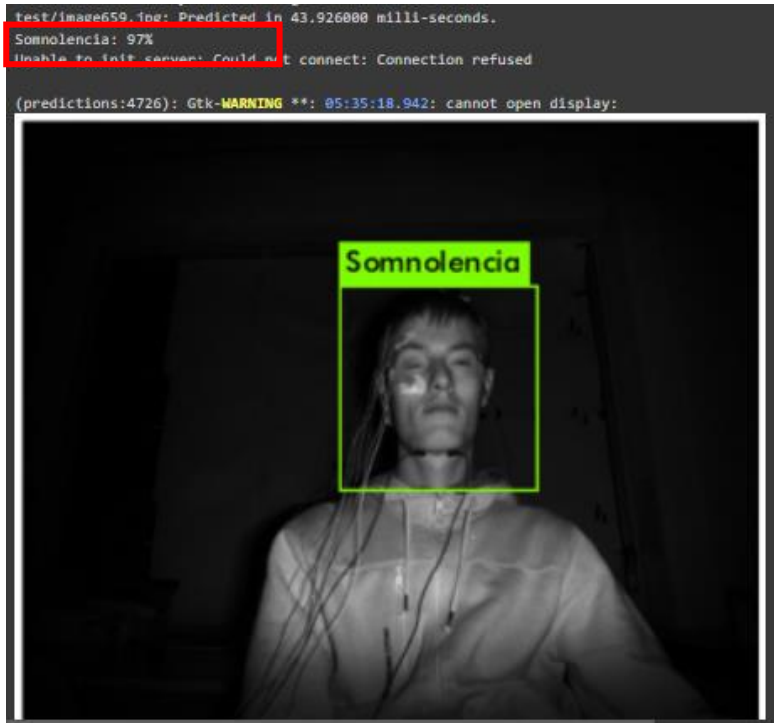


Figura 37. Detección de la clase no somnolencia en un 97%.

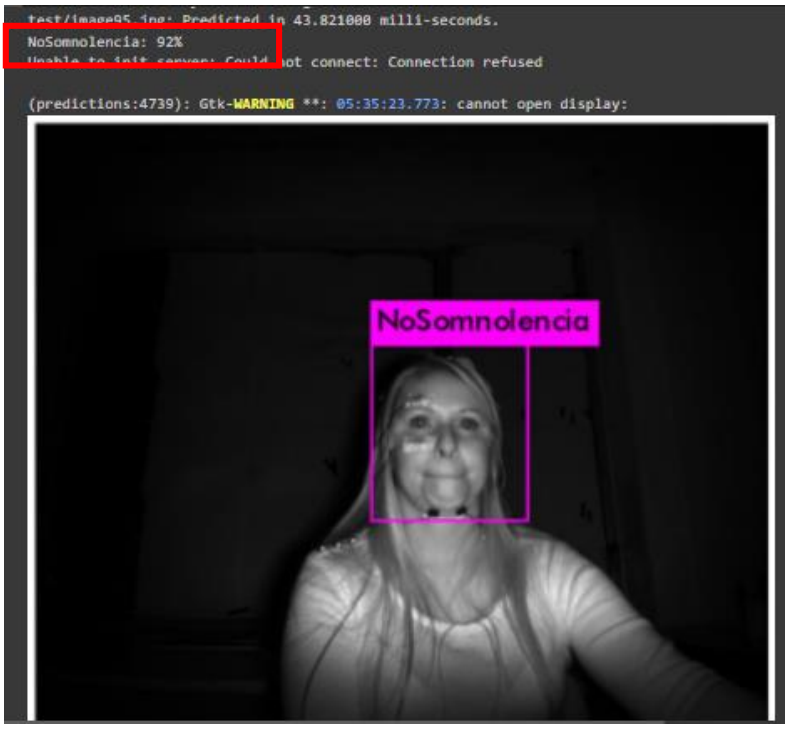


Figura 38. Detección de la clase no somnolencia en un 92%.

Figura 39. Detección de la clase no somnolencia en un 99%.

Figura 40. Detección de la clase somnolencia en un 99%.

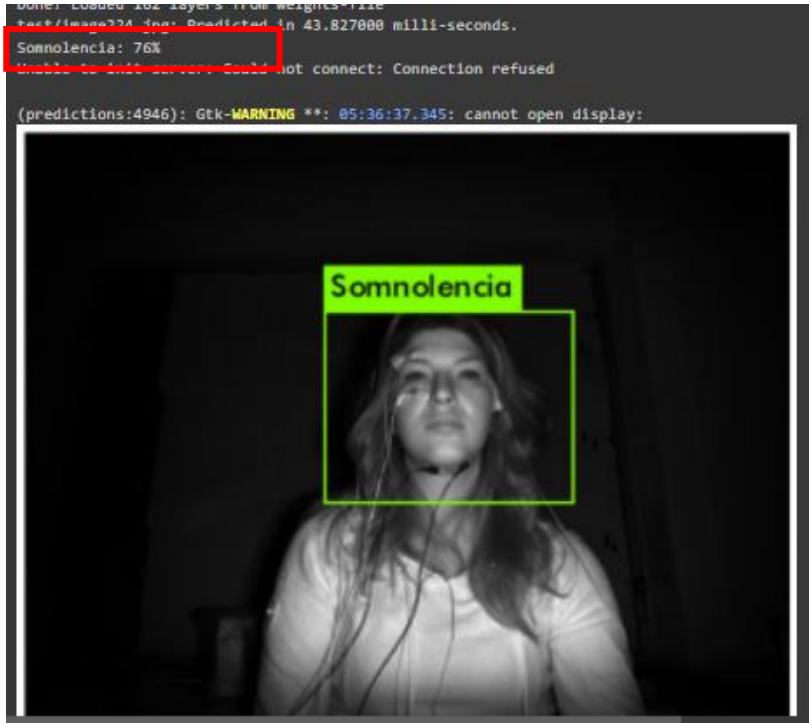


Figura 41. Detección de la clase somnolencia en un 76%.

4.3. Métricas

En la Figura 42 se presenta la matriz de confusión, en la que se concentran los valores obtenidos al probar el algoritmo utilizando las 35 imágenes que se tienen para pruebas. Esta matriz de confusión nos da una información sobre los valores reales de las clases y los valores predichos por el detector de somnolencia.

Aplicando las fórmulas que se presentan en la sección 2.12.4 se obtiene la Tabla 4, en la que podemos ver algunas métricas como la exactitud, precisión, exhaustividad y el valor F1 que nos indica el desempeño del algoritmo previamente entrenado.

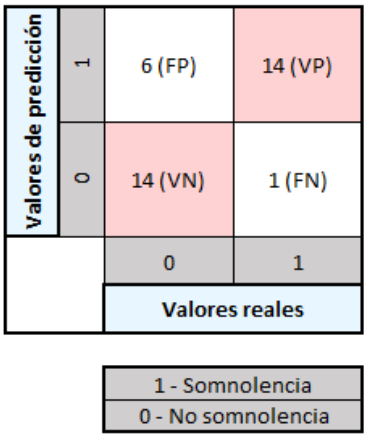


Figura 42. Matriz de confusión obtenida en las pruebas (Imagen propia).

Tabla 4. Métricas obtenidas en las pruebas.

Exactitud	Precisión	Exhaustividad	Valor F1
80%	70%	93%	80%

4.4. Implementación en sistema embebido

Finalmente, se realiza la implementación del algoritmo YOLO-v4 personalizado en un sistema embebido. Se elige la tarjeta NVIDIA Jetson Nano debido a su buen desempeño en aplicaciones de inteligencia artificial. Algunas de las características de este kit de desarrollo son las siguientes:

- **GPU** maxwell de 128 núcleos
- **CPU** ARM de cuatro núcleos a 1,43 GHz
- **Memoria** 4GB LPDDR4 de 64 bits 25,6 GB/s
- Sistema operativo Ubuntu

Para la implementación del algoritmo en la tarjeta, únicamente se configura el entorno, sin realizar la parte del entrenamiento, pues ese paso ya se realizó en la sección

3.7. Los archivos con los pesos guardados en la sección 3.7.6 son los que se utilizan para realizar la detección de somnolencia en este sistema embebido. Algunos de los resultados que se obtuvieron se muestran en las siguientes figuras

El tiempo que se tarda el detector en procesar cada imagen es de aproximadamente 40 segundos. En el caso en el que el detector dé como resultado las dos clases en la misma imagen, se toma la clase que tenga un porcentaje de fiabilidad más alto; como por ejemplo, en la Figura 43 la clase que mayor porcentaje de fiabilidad tiene es la clase No-Somnolencia, por lo que la imagen pertenece a esta clase. Lo mismo pasa con la Figura 47, que pertenece a la clase No-Somnolencia y la Figura 48 a la clase Somnolencia.

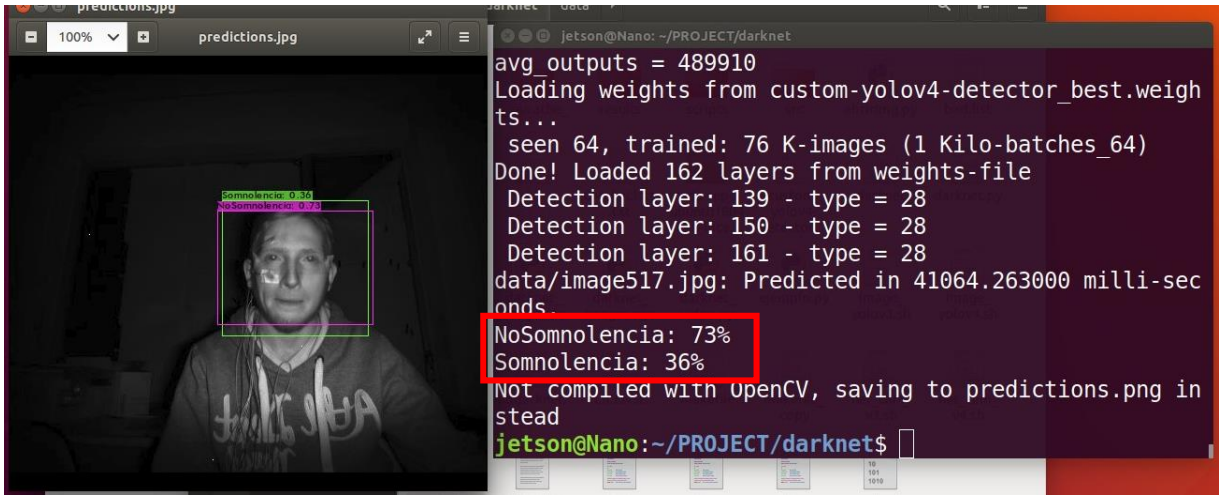


Figura 43. Detección de la clase No somnolencia en 73% y Somnolencia en 36% detectadas en el sistema embebido.

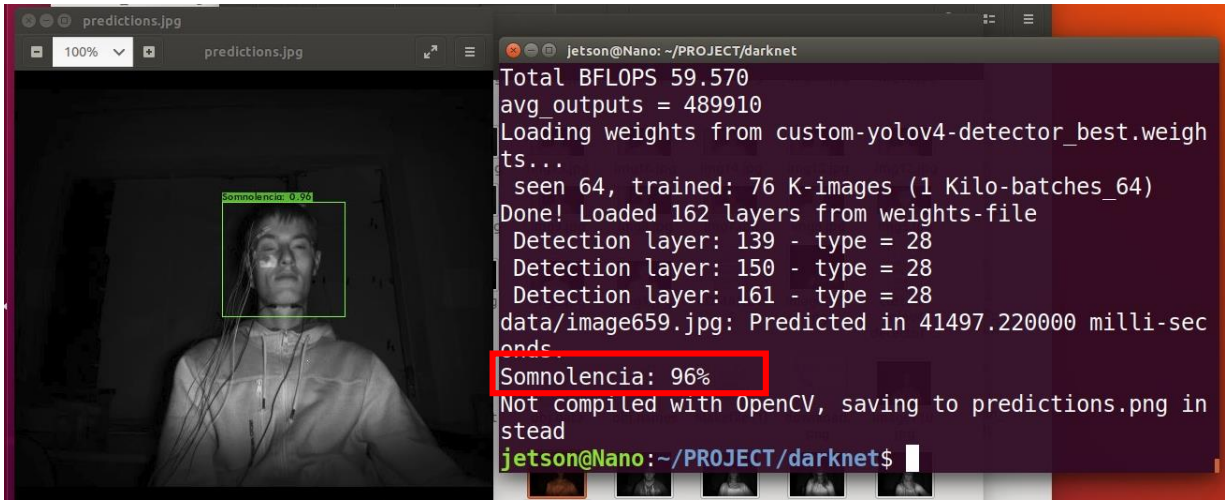


Figura 44. Detección de la clase somnolencia en un 96% detectada en el sistema embebido.

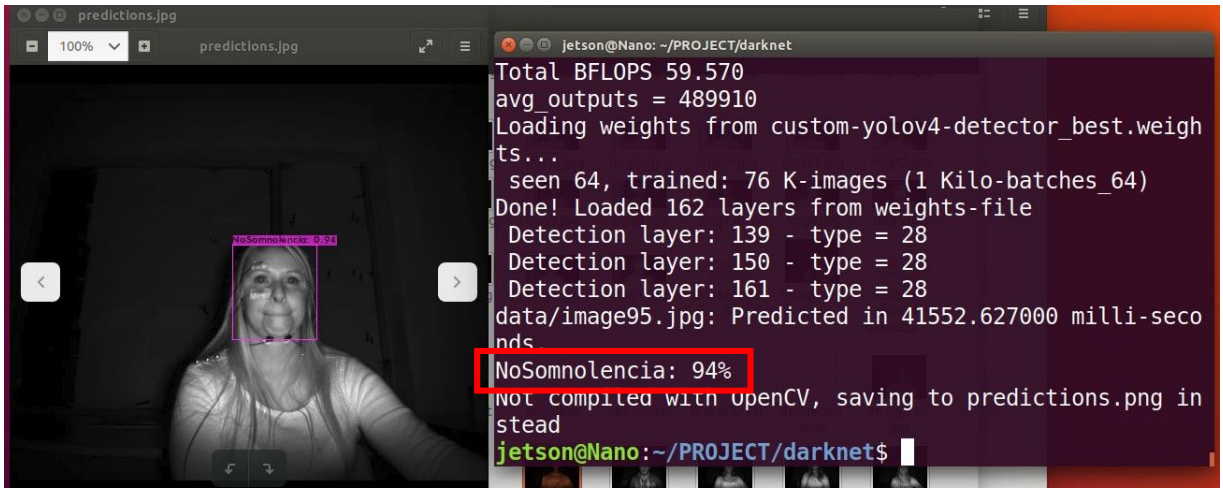


Figura 45. Detección de la clase No somnolencia en un 94% detectada en el sistema embebido.

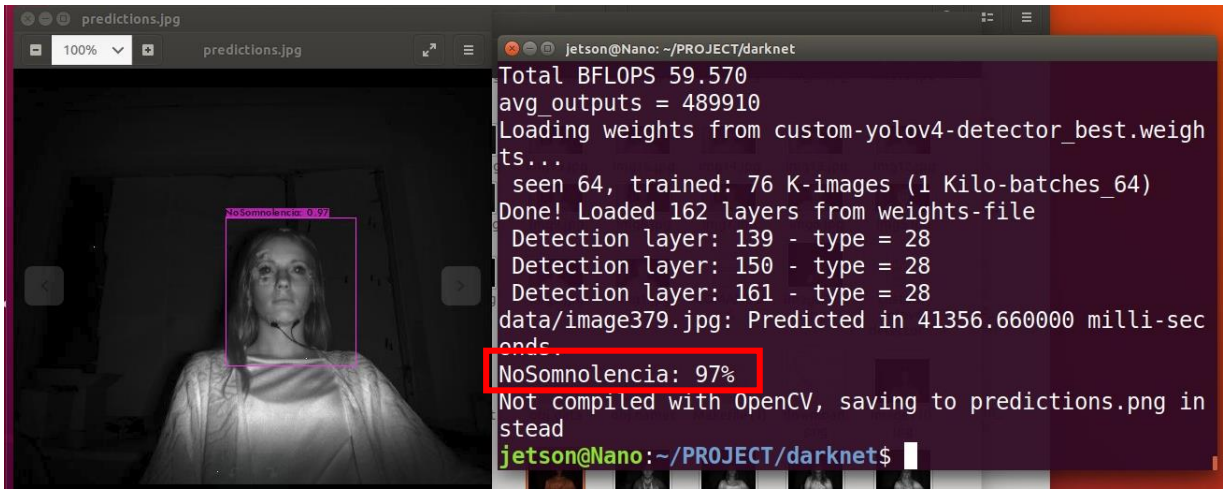


Figura 46. Detección de la clase No somnolencia en un 97% detectada en el sistema embebido.

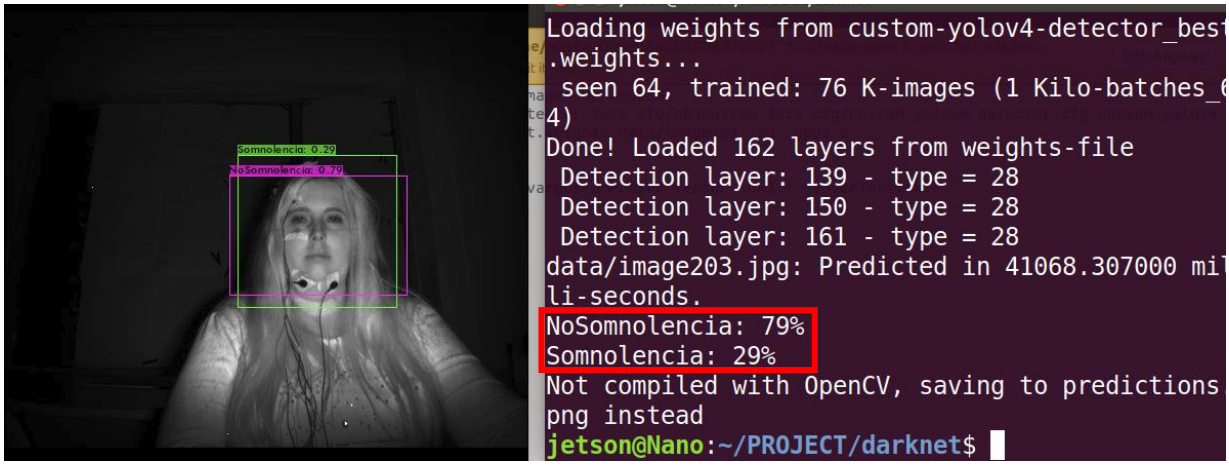


Figura 47. Detección de la clase No somnolencia en 79% y Somnolencia en 29% detectadas en el sistema embebido.

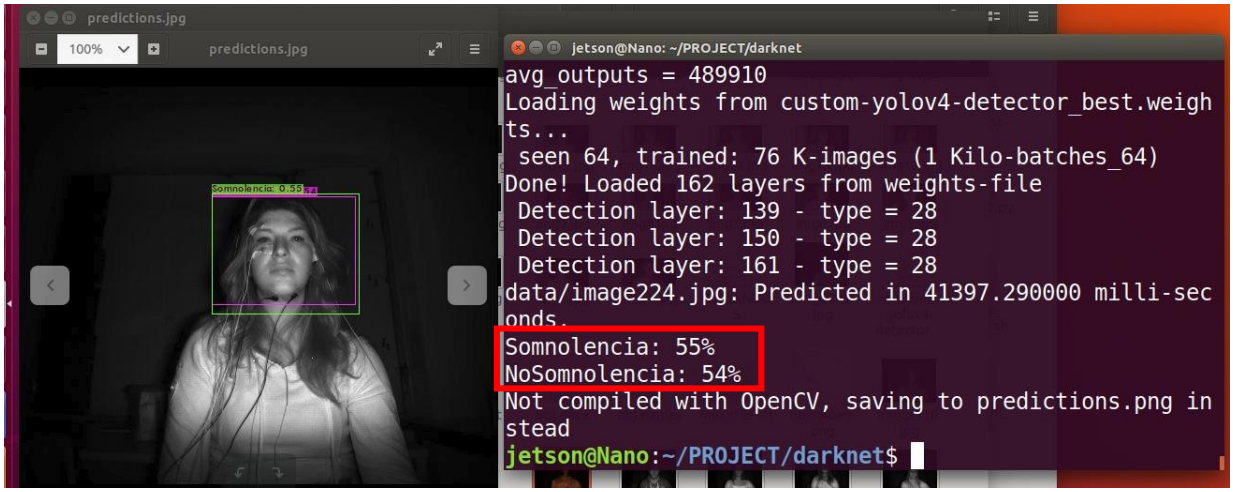


Figura 48. Detección de la clase No somnolencia en 55% y Somnolencia en 54% detectadas en el sistema embebido.

4.5. Métricas en sistema embebido

Por último, se realiza una matriz de confusión como la mostrada en la sección 4.3, pero ahora con los datos obtenidos probando cada una de las imágenes de la carpeta de pruebas en el sistema embebido. Esta matriz de confusión se puede ver en la Figura 49 y en la Tabla 5 las métricas del algoritmo implementado en el sistema embebido.

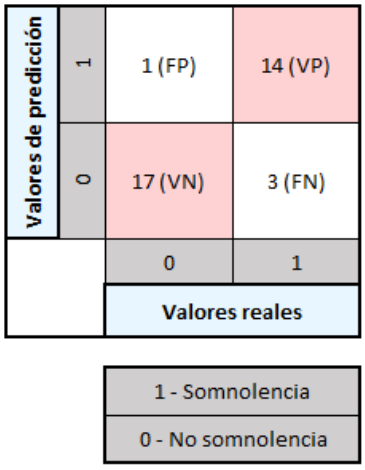


Figura 49. Matriz de confusión obtenida en las pruebas en el sistema embebido (Imagen propia).

Tabla 5. Métricas obtenidas en las pruebas en el sistema embebido.

Exactitud	Precisión	Exhaustividad	Valor F1
88%	94%	82%	87%

4.6. Comparación de los resultados con el estado del arte

En la Tabla 6 se muestra una comparación de la métrica exactitud y el tamaño de la base de datos de esta tesis, en relación con el tamaño de la base de datos utilizada para realizar el entrenamiento de los algoritmos de algunos de los trabajos del estado del arte mencionados en el Capítulo 1. Se puede ver como a mayor cantidad de imágenes utilizadas el valor de la exactitud también es mayor.

Tabla 6. Comparación de resultados.

			Resultados
Autor y año	Articulo	# imágenes	Exactitud
(Ma et al., 2019)	Convolutional Three-stream Network Fusion for Driver Fatigue Detection from Infrared Videos	39,000	94.68%
(Du et al., 2020)	Vision-Based Fatigue Driving Recognition Method Integrating Heart Rate and Facial Features	60,000	94.74%
(Chen et al., 2021)	Driver Drowsiness Estimation Based on Factorized Bilinear Feature Fusion and a Long-Short-Term Recurrent Convolutional Network	450 video clips	75.80%
Este trabajo	Sistema de reconocimiento de expresiones faciales para detección de estados de somnolencia con imágenes nocturnas	710	88 %

4.7. Publicación

Como parte de los resultados de este trabajo se tiene la publicación del artículo titulado “Detección de somnolencia en conductores en imágenes nocturnas”, el cual fue presentado en el 16° coloquio de posgrado de la facultad de ingeniería. Las memorias del coloquio serán publicadas en la página https://coloquioposgradoingenieria.home.blog/. En los anexos se encuentra el artículo, así como también la constancia de la ponencia realizada en el coloquio.

Capítulo 5. Conclusiones

En este trabajo de tesis se presenta un detector de somnolencia utilizando imágenes nocturnas. Para lo anterior se utiliza un detector de objetos personalizado YOLO-v4, el cual es entrenado con 710 imágenes que contienen dos clases, Somnolencia y No-Somnolencia. Se realiza un etiquetado de cada imagen, para posteriormente entrenar el algoritmo en la herramienta Google Colab, esto para utilizar la mayor cantidad de recursos disponibles y de esta manera lograr optimizar los tiempos de procesamiento durante el entrenamiento.

Además, el algoritmo se implementa en un sistema embebido (Tarjeta NVIDIA Jetson Nano). En dicha tarjeta, se realiza únicamente la preparación para la implementación del detector y las pruebas con las imágenes, sin realizar la etapa del entrenamiento previamente realizada en Google Colab.

El detector alcanza una exactitud del 88% en las pruebas en el sistema embebido. Incluso el detector presenta resultados favorables probando con algunas imágenes propias que fueron tomadas con una cámara infrarroja, las cuales no fueron utilizadas para entrenar. Estos resultados no son mostrados en esta tesis debido a la confidencialidad de las imágenes.

Una de las dificultades que se tuvo fue tener acceso a bases de datos con mayor cantidad de imágenes, dado que se encuentran pocas bases públicas con imágenes infrarrojas con somnolencia; por lo anterior, se optó por utilizar la Drozy Database (Massoz et al., 2016) que es una base de datos pública con 710 imágenes.

Comparando los resultados con trabajos similares ya publicados, podemos comprender la importancia de la cantidad de datos con los que se entrena el detector, ya que en los trabajos del estado del arte los entrenamientos se realizan utilizando 39,000

imágenes, como en el trabajo de Ma et al. (2019), o en la investigación de Du el al. (2021) que utiliza 60,000 imágenes. La cantidad de datos utilizados es el motivo de que se presenten exactitudes superiores a los resultados obtenidos en este proyecto, como se presentó en la Tabla 6.

Como trabajo futuro se tiene la implementación del detector en un entorno real, es decir, dentro de un automóvil. Además, se pretende crear una base de datos con imágenes propias tomadas con una cámara infrarroja, esto debido a que existen pocas bases de datos públicas de imágenes con somnolencia tomadas con infrarrojo cercano en ambientes nocturnos.

Anexos

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA
DIPFI POSGRADO INGENIERÍA

COLOQUIO 16 POSGRADO FI

Se otorga la presente

CONSTANCIA a:

KAREN ANDREA RAMÍREZ ARRIAGA

Por su participación en el
XVI Coloquio de Posgrado de la Facultad de Ingeniería
de la Universidad Autónoma de Querétaro
con su Ponencia:

Detección de somnolencia
en conductores en imágenes
nocturnas

Noviembre de 2022
Facultad de Ingeniería

Dr. Manuel Toledano Ayala
DIRECTOR
FACULTAD DE INGENIERÍA

Dr. Juan Carlos Jáuregui Correa
Jefe de la División de Investigación y Posgrado
FACULTAD DE INGENIERÍA

Escaneado con CamScanner

XVI Coloquio de Posgrado FI

Detección de somnolencia en conductores en imágenes nocturnas

Driver drowsiness detection in nighttime images

Ramírez Arriaga Karen Andrea✉, Ramos Arreguín Juan Manuel✉, Pedraza Ortega Jesús Carlos, Tovar Arriaga Saúl, Salazar Colores Sebastián, Gorrostieta Hurtado Efrén
Facultad de ingeniería
Universidad Autónoma de Querétaro
Querétaro, México.
kramirez08@alumnos.uaq.mx, jsistdig@yahoo.com.mx

Resumen- La somnolencia está directamente vinculada con el cansancio por consiguiente es un problema para las personas que conducen vehículos motorizados (automóviles, camiones, entre otros), ya que es responsable de alrededor del 20-30% de los accidentes en autopistas. Por esta razón, detectar que el conductor se está quedando dormido, es una forma de evitar accidentes ocasionados por fatiga. Existen varios trabajos que tratan este problema, especialmente utilizando imágenes tomadas con luz de día, siendo muy pocos los que trabajan en la detección de la somnolencia en ambientes nocturnos usando sistemas de visión nocturna. Este trabajo presenta un algoritmo en el que se detecta la somnolencia en imágenes tomadas de noche, utilizando una cámara infrarroja. Para ello se utiliza una arquitectura YOLO-v4, obteniendo un 80% de precisión y un 93% de exhaustividad. Se utiliza una base de datos pública de somnolencia para entrenar y probar el algoritmo, incluidas imágenes propias.

Palabras clave: IAR, somnolencia, infrarrojo cercano, YOLO-v4, imágenes nocturnas.

Abstract- Drowsiness is directly linked to fatigue, therefore it is a problem for people who drive motor vehicles (cars, trucks, among others), since it is responsible for around 20-30% of accidents on highways. For this reason, detecting that the driver is falling asleep is a way to avoid accidents caused by fatigue. There are several works that deal with this problem, but especially with images taken in daylight, with very few working on the detection of drowsiness in night environments using night vision systems. This work presents an algorithm in which drowsiness is detected in images taken at night, using an infrared camera. For this, a YOLO-v4 architecture is used, obtaining 80% accuracy and 93% Recall. A public Drowsiness database is used to train and test the algorithm, including own nighttime images.

Keywords: IAR, drowsiness, near infrared, YOLO-v4, nighttime images.

1. Introducción

El problema de somnolencia es algo muy común en personas que realizan actividades como vigilancia, conducción de vehículos, entre otras [2]. Debido a esto, se han desarrollado diversos algoritmos basados en inteligencia artificial para detectar estados de somnolencia.

Según Zhang et al. [3] nuestras caras contienen mucha información útil que podemos usar para detectar la fatiga, como lo es el estado de los ojos, los bostezos y la posición del rostro. En 2016, se desarrolla

Referencias

Alcántara, C. (2018). *Sistema embebido de bajo costo para la asistencia al conductor mediante procesamiento de expresiones faciales*.

Ameijeiras, D., González, H., & Hernández, Y. (2020). Revisión de algoritmos de detección y seguimiento de objetos con redes profundas para videovigilancia inteligente. *Revista Cubana de Ciencias Informáticas*, *14*(3), 165–196.

Aprende Machine Leaning. (2020). *Modelos de Detección de Objetos | Aprende Machine Learning*. 1–40. https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/

Boden, M. (2016). *Inteligencia artificial*. https://books.google.com.mx/books?hl=es&lr=&id=LCnYDwAAQBAJ&oi=fnd&pg=PT3&dq=inteligencia+artificial+boden&ots=drVlwXfPoa&sig=Z3EDt75GoiWnxbWxIDBO7-bnYIw&redir_esc=y#v=onepage&q=inteligencia artificial boden&f=false

Chávez, G. (2010). *La Fatiga Al Conducir*. https://www.antp.org.mx/revista/86/articulo4.pdf

Chen, S., Wang, Z., & Chen, W. (2021). Driver drowsiness estimation based on factorized bilinear feature fusion and a long-short-term recurrent convolutional network. *Information*, *12*(1), 1–15. https://doi.org/10.3390/info12010003

Ciencia Carbónica. (2019). *Diagrama IA*. https://cienciacarbonica.es/la-inteligencia-artificial-sustituiran-a-los-medicos/diagrama-ia/

Diaz, R. (n.d.). *Métricas de clasificación*. The Machine Learners. https://www.themachinelearners.com/metricas-de-clasificacion/

Dirección General de Servicios Técnicos. (2017). *ESTADISTICA ACCIDENTES DE*

TRÁNSITO (2017).

Du, G., Li, T., Li, C., Liu, P. X., & Li, D. (2021). Vision-based fatigue driving recognition method integrating heart rate and facial features. *IEEE Transactions on Intelligent Transportation Systems*, *22*(5), 3089–3100. https://doi.org/10.1109/TITS.2020.2979527

Friswell, R., & Williamson, A. (2008). Exploratory study of fatigue in light and short haul transport drivers in NSW, Australia. *Accident Analysis and Prevention*, *40*(1), 410–417. https://doi.org/10.1016/j.aap.2007.07.009

García, A. (2012). *Inteligencia artificial. Fundamentos, práctica y aplicaciones*. https://books.google.com.mx/books?hl=es&lr=&id=WDuqquRP70UC&oi=fnd&pg=PP9&dq=Inteligencia+Artificial.+Fundamentos,+práctica+y+aplicaciones&ots=iUT1h0inHy&sig=S-Wg62VC17e4jbhl3CZNLS21WR4&redir_esc=y#v=onepage&q=Inteligencia Artificial. Fundamentos%2C práct

Geekflare. (2022). *Google Colab: todo lo que necesitas saber*. https://geekflare.com/es/google-colab/

Geng, L., Liang, X., Xiao, Z., & Li, Y. (2018). Real-time driver fatigue detection based on morphology infrared features and deep learning. *Infrared and Laser Engineering*, *47*(2), 1–9. https://doi.org/10.3788/IRLA201847.0203009

González, M. (2017). *Infrarrojos*. Mgmdenia’s Blog. https://mgmdenia.wordpress.com/2017/12/13/infrarrojos/

Google. (n.d.). *Colaboratory*. https://research.google.com/colaboratory/intl/es/faq.html#:~:text=Colaboratory%2C o %22Colab%22 para,análisis de datos y educación.

Gutta, S. (2021). *Object Detection Algorithm — YOLO v5 Architecture*. https://medium.com/analytics-vidhya/object-detection-algorithm-yolo-v5-architecture-

89e0a35472ef

Instituto Mexicano del Transporte. (2017). *Herramientas para la seguridad en la movilidad, modelos predictivos de somnolencia en conductores*. Publicación Bimestral de Divulgación Externa.

Instituto Mexicano del Transporte. (2021). *Estadística de accidentes de tránsito, año 2020* (Issue 84). https://www.sct.gob.mx/fileadmin/DireccionesGrales/DGST/Estadistica_de_accidentes/Año-2020/dt84.pdf

Instituto Nacional de Estadística y Geografía (INEGI). (2016). *Síntesis Metodológica de la Estadística de Accidentes de Tránsito Terrestre en Zonas Urbanas y Suburbanas*. 23. http://internet.contenidos.inegi.org.mx/contenidos/Productos/prod_serv/contenidos/espanol/bvinegi/productos/nueva_estruc/702825087999.pdf%0Aatencion.usuarios@inegi.org.mx%5CnSíntesis

Jang, S. W., & Ahn, B. (2020). Implementation of detection system for drowsy driving prevention using image recognition and IoT. *Sustainability*, *12*(7).

Jie, L., Hui, Z., & Zejian, Y. (2017). Joint Shape and Local Appearance Features for Real-Time Driver Drowsiness Detection. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10118 LNCS*, 178–194. https://doi.org/10.1007/978-3-319-54526-4

Khopkar, A. (2021). Facial Expression Recognition Using CNN with Keras. *Bioscience Biotechnology Research Communications*, *14*(5), 47–50. https://doi.org/10.21786/bbrc/14.5/10

Knapik, M., & Cyganek, B. (2019). Driver's fatigue recognition based on yawn detection in thermal images. *Neurocomputing*, *338*, 274–292. https://doi.org/10.1016/j.neucom.2019.02.014

LearnOpencv. (2022). *Intersection over Union (IoU) in Object Detection and Segmentation*. https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/

LearnOpenCv. (n.d.). *Mean Average Precision (mAP) in Object Detection*. https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/

López, M. (2017). *Identificación de sistemas no lineales con redes neuronales convolucionales.* 103.

Ma, X., Chau, L. P., Yap, K. H., & Ping, G. (2019). Convolutional three-stream network fusion for driver fatigue detection from infrared videos. *Proceedings - IEEE International Symposium on Circuits and Systems*, *2019-May*, 1–5. https://doi.org/10.1109/ISCAS.2019.8702447

Martinez, D., Sánchez, D., & Demmler, M. (2018). ACCIDENTES VIALES EN MÉXICO Y SU RELACIÓN CON EL HORARIO DE VERANO. *Gaceta Técnica*, *19*(2), 69–81. https://doi.org/10.13140/RG.2.2.13733.81127

Martinez, J. (2020). *Precision, Recall, F1, Accuracy en clasificación*. https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/

Massoz, Q., Langohr, T., François, C., & Verly, J. G. (2016). The ULg multimodality drowsiness database (called DROZY) and examples of use. *In 2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 1–7.

Mathivet, V. (2018). *Inteligencia Artificial para desarrolladores*. https://books.google.com.mx/books?hl=es&lr=&id=Fd06Ql4QRWkC&oi=fnd&pg=PA15&dq=Inteligencia+artificial+para+desarrolladores:+conceptos+e+implementación+en+C&ots=r-_fiu8goA&sig=86KZCHxnGgoX7lG2r-_gyFX3yxM&redir_esc=y#v=onepage&q=Inteligencia artificial para

Milagros, Z., & Julian, H. (2020). Implementación de un sistema de gestión de seguridad electrónica con Machine Learning dirigido a Prosegur Perú para gestión de seguridad en viviendas de Lima Metropolitana. *Universidad Tecnológica Del Perú*, 1–47. https://repositorio.utp.edu.pe/bitstream/handle/20.500.12867/2842/Zuly Huaman_Trabajo de Investigacion_Bachiller_2020.pdf?sequence=1&isAllowed=y

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

Moreno, A., Armengol, E., Bejar, J., Belanche, L., Cortez, U., Gavalda, R., Lopez, B., Sanchez, M., Martin, M., & Gimeno, J. (1994). *Aprendizaje automático*. https://upcommons.upc.edu/handle/2099.3/36157

Norman, A. (2019). *Aprendizaje automático en acción*. https://books.google.com.mx/books?hl=es&lr=&id=iTIREAAAQBAJ&oi=fnd&pg=PT15&dq=Aprendizaje+automático+en+acción&ots=hVW5vj8aF3&sig=nrevYBe_n4SC19R0w8ANEWJRIn8&redir_esc=y#v=onepage&q=Aprendizaje automático en acción&f=false

Ramos, J.-M. (2018). *OPTATIVA DE ESPECIALIDAD I: Redes Neuronales y Sistemas Difusos*.

RGB-NIR Data. (n.d.). *RGB-NIR Scene Dataset*. https://ivrlwww.epfl.ch/supplementary_material/cvpr11/index.html

Sai Mani Teja, B., Anita, C. S., Rajalakshmi, D., & Berlin, M. A. (2020). A CNN based facial expression recognizer. *Materials Today: Proceedings*, *37*(Part 2), 2578–2581. https://doi.org/10.1016/j.matpr.2020.08.501

Tokkie, G., & Hinner, K. (2006). Espectroscopia de Infrarrojo Cercano (NIRS) - La técnica de análisis rápidos del futuro. *Community of International Busines Related to Animal*

Production. https://www.engormix.com/balanceados/articulos/espectroscopia-infrarrojo-cercano-nirs-t26241.htm

Torres, F. F., López, M., Orozco, H. R., & Lazcano, S. (2020). Clasificación del género de peatones y conductores basada en su comportamiento en la vía pública. *Research in Computing Science*, *149*(8), 957–970.

Villanueva, A., Benemerito, R. L. L., Cabug-Os, M. J. M., Chua, R. B., Rebeca, C. K. D. C., & Miranda, M. (2019). Somnolence detection system utilizing deep neural network. *2019 International Conference on Information and Communications Technology, ICOIACT 2019*, 602–607. https://doi.org/10.1109/ICOIACT46704.2019.8938460

WHO. (2018). Global Status Report on Road. *World Health Organization*, 20. https://www.who.int/publications/i/item/9789241565684

Yan, J. J., Kuo, H. H., Lin, Y. F., & Liao, T. L. (2016). Real-time driver drowsiness detection system based on PERCLOS and grayscale image processing. *Proceedings - 2016 IEEE International Symposium on Computer, Consumer and Control, IS3C 2016*, 243–246. https://doi.org/10.1109/IS3C.2016.72

Zhang, F., Su, J., Geng, L., & Xiao, Z. (2017). Driver fatigue detection based on eye state recognition. *International Conference on Machine Vision and Information Technology*, 105–110. https://doi.org/10.1109/CMVIT.2017.25