

**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL
DE HUAMANGA**

**FACULTAD DE INGENIERÍA DE MINAS, GEOLOGÍA Y
CIVIL**

ESCUELA PROFESIONAL DE INGENIERÍA CIVIL



TESIS:

**Modelo basado en redes neuronales artificiales para el
análisis del tráfico vehicular en la Av. Ramón Castilla del
distrito de Ayacucho - Huamanga 2024**

Para optar el título profesional de:
INGENIERA CIVIL

PRESENTADO POR:
Bach. Yuvica QUISPE LOPEZ

ASESOR:
Mg. Ing. Edward LEÓN PALACIOS

AYACUCHO - PERÚ

2025

RESUMEN

La presente investigación propone el desarrollo de un sistema automatizado para el cálculo del Índice Medio Diario Anual (IMDA) y del número de repeticiones de ejes equivalentes (EE), empleando redes neuronales artificiales para la detección y clasificación vehicular con fines de diseño estructural de pavimentos. Este sistema surge ante la problemática de decisiones inadecuadas en la planificación vial debido a análisis vehiculares imprecisos. Si no se realiza un conteo vehicular riguroso, los resultados obtenidos pueden no reflejar la realidad del tránsito, comprometiendo la eficiencia del diseño de pavimentos. Este tipo de errores, muchas veces causados por la falta de herramientas automatizadas y la persistencia en métodos manuales desactualizados, genera impactos negativos como mayores costos de mantenimiento, baja productividad, sobrecostos logísticos y deterioro en la calidad de vida urbana. La propuesta abarcó tres ejes: El diseño de una arquitectura robusta para detección vehicular, el entrenamiento del modelo basado en YOLOv8 para alcanzar un equilibrio entre precisión, cobertura y eficiencia, y la implementación de un software con interfaz gráfica que automatice el cálculo del IMD y EE bajo la metodología AASHTO. El modelo alcanzó un desempeño destacado con una precisión del 89.7 %, recall de 81.5 %, mAP50 de 0.882 y mAP50-95 de 0.824, demostrando alta efectividad para 22 clases vehiculares definidas por el MTC. Además, el procesamiento por imagen mostró tiempos competitivos, validando su aplicabilidad en escenarios de monitoreo en tiempo real. En conjunto, esta solución representa una herramienta innovadora y adaptable para la planificación vial, permitiendo decisiones más informadas y técnicas en el ámbito del transporte y la infraestructura.

Palabras clave: Redes neuronales artificiales (RNA), IMDA, ejes equivalentes, YOLOv8, conteo vehicular, diseño de pavimentos.

ABSTRACT

The present study introduces the design of an automated framework aimed at computing Annual Average Daily Traffic (AADT) and determining Equivalent Single Axle Loads (ESAL) through artificial neural networks for vehicle detection and classification purposes in structural pavement design. This approach emerges as a solution to the challenge of inadequate decision-making processes in highway planning caused by imprecise vehicular analysis. The absence of rigorous vehicle counting procedures may generate outcomes that fail to represent real traffic conditions, thereby compromising pavement design effectiveness. Such deficiencies—commonly stemming from the lack of automated tools and persistent reliance on obsolete manual approaches—result in adverse consequences including increased maintenance expenditures, decreased productivity, logistical cost overruns, and deterioration of urban life quality. The proposed framework is built upon three fundamental components: the design of a robust architecture for vehicle detection, the training of a YOLOv8-based model that achieves equilibrium between precision, coverage, and efficiency, and the development of software featuring a graphical user interface that automates AADT and ESAL calculations following AASHTO methodology. The model demonstrated remarkable performance, achieving 89.7% accuracy, 81.5% recall, mAP50 of 0.882, and mAP50-95 of 0.824, showing high effectiveness across 22 vehicle categories established by Peru's Ministry of Transport and Communications (MTC). Furthermore, the per-frame processing exhibited competitive execution times, confirming its applicability in real-time monitoring scenarios. Collectively, this framework constitutes an innovative and adaptable tool for highway planning, facilitating more informed and technically sound decisions in transportation and infrastructure domains.

Keywords: Artificial neural networks (ANN), AADT, equivalent axles, YOLOv8, vehicle counting, pavement design.

INTRODUCCIÓN

En la actualidad, el diseño estructural de pavimentos y la planificación vial demandan información precisa, confiable y actualizada sobre el comportamiento del tránsito vehicular. Entre los parámetros fundamentales que sustentan el análisis técnico se encuentran el Índice Medio Diario Anual (IMDA) y el Número de Repeticiones de Ejes Equivalentes (EE), los cuales permiten estimar la vida útil de una vía y tomar decisiones adecuadas para su diseño, mantenimiento y rehabilitación. Sin una base de datos vehicular sólida, cualquier cálculo estructural podría conducir a errores significativos que comprometen la seguridad y eficiencia de las infraestructuras viales.

No obstante, en múltiples contextos locales, aún prevalece el uso de metodologías manuales de conteo vehicular que resultan obsoletas, ineficientes y susceptibles a errores humanos. Esta situación limita la representatividad y exactitud de los datos recopilados, lo que repercute directamente en la calidad del diseño vial. Si no se realiza un buen análisis de conteo vehicular, no se obtendrán resultados acordes a la realidad en el diseño de pavimentos, lo que puede llevar a decisiones subóptimas en la planificación vial. Las causas de estos problemas incluyen la falta de adopción de tecnologías avanzadas de recolección de datos y la dependencia de métodos manuales desactualizados. La mayoría de los softwares actuales no implementan algoritmos de inteligencia artificial avanzados, lo que podría mejorar significativamente la precisión y eficiencia del análisis de tráfico vehicular.

La ausencia de información vehicular precisa y automatizada puede traducirse en consecuencias negativas como el incremento de sobrecostos logísticos, fallos prematuros en la infraestructura, baja productividad, y deficiente planificación de obras. Esta problemática repercute no solo en la movilidad urbana, sino también en los costos operativos del transporte y en la calidad de vida de la población.

Ante esta necesidad, la presente investigación propone el desarrollo e implementación de un sistema automatizado para el cálculo del IMDA y EE, empleando redes neuronales artificiales para la detección, clasificación y conteo vehicular, basado en la metodología de diseño AASHTO. Para ello, se utiliza el modelo YOLOv8, entrenado con múltiples clases vehiculares definidas por el MTC, capaz de operar en condiciones variables como baja iluminación, presencia de sombras proyectadas, tráfico denso y alta variabilidad de vehículos.

Además, se incorpora una interfaz gráfica de usuario (GUI) que permite automatizar el procesamiento de datos y generar resultados confiables en tiempo real. Esta plataforma no solo mejora la eficiencia operativa, sino que también representa una herramienta accesible y adaptable a diferentes entornos viales, contribuyendo a modernizar el análisis de tránsito en el país.

A mis padres, quienes han sido los arquitectos silenciosos de mis sueños y los guardianes incansables de mi futuro. A ustedes, que con su amor incondicional y sacrificio constante han construido el sendero que hoy recorro con orgullo. Cada página de este trabajo lleva impreso el eco de sus consejos, el calor de sus abrazos y la fuerza de su ejemplo.

AGRADECIMIENTOS

Extiendo mi más sincero agradecimiento a los docentes de la Escuela Profesional de Ingeniería Civil de la Universidad Nacional de San Cristóbal de Huamanga, quienes durante mi trayectoria académica compartieron de manera generosa sus conocimientos y experiencias profesionales, estableciendo los cimientos sólidos de mi crecimiento profesional. De forma particular, expreso mi profundo reconocimiento al M.Sc. Ing. Edward León Palacios, director de esta investigación, quien mediante su amplia experiencia, dedicación y orientación metodológica permitió la concreción de este estudio. Sus contribuciones técnicas y académicas resultaron fundamentales para alcanzar los objetivos establecidos. A los integrantes del tribunal evaluador, Dr.Ing. Hemerson Lizarbe Alarcón y M.Sc.Ing. Moises Nico Barbaran Oriundo, por su exhaustiva revisión, valiosos aportes y dedicación hacia la excelencia académica, aspectos que incrementaron considerablemente la calidad de esta investigación. A mis colegas de estudio y especialistas del campo, quienes a lo largo del desarrollo de esta tesis contribuyeron con sus conocimientos, experiencias y recomendaciones, enriqueciendo el análisis y consolidando la metodología utilizada en este trabajo.

ÍNDICE GENERAL

ÍNDICE GENERAL	VII
1 PLANTEAMIENTO DEL PROBLEMA	4
1.1 Delimitación del problema	5
1.1.1 Espacial (geográfica)	5
1.1.2 Temporal	5
1.1.3 Temática y unidad de análisis	5
1.2 Formulación del problema	5
1.2.1 Problema general	5
1.2.2 Problemas específicos	5
1.3 Justificación e importancia	6
1.4 Limitaciones de la investigación	6
1.5 Objetivos de la investigación	6
1.5.1 Objetivo general	6
1.5.2 Objetivos específicos	7
2 MARCO TEÓRICO	8
2.1 Antecedentes	8
2.1.1 Investigaciones internacionales.	8
2.1.2 Investigaciones nacionales	12
2.2 Bases teóricas	13
2.2.1 Tráfico vehicular	13
2.2.2 Estudio de tráfico	15
2.2.3 Determinación del tráfico actual	17
2.2.4 Índice Medio Diario Anual (IMDA)	17
2.2.5 Número De Repeticiones De Ejes Equivalentes.	21
2.2.6 Redes neuronales artificiales.	26
2.2.7 Aprendizaje automático o machine learning.	27
2.2.8 Aprendizaje profundo o deep learning	29
2.2.9 Redes neuronales convolucionales	30
2.2.10 Funcionamiento de la CNN.	33
2.2.11 Entrenamiento de una CNN	37
2.2.12 Algoritmo de entrenamiento de la Red Neuronal	37
2.2.13 Detección de objetos	39
2.2.14 YOLOv8	40
2.2.15 Seguimiento de objetos	40
3 MARCO METODOLÓGICO	42
3.1 Enfoque.	42
3.2 Alcance	42

3.3	Diseño de investigación	42
3.4	Población y muestra.	43
3.4.1	Población	43
3.4.2	Muestra	43
3.4.3	Unidad de análisis	44
3.5	Hipótesis	44
3.5.1	Hipótesis general	44
3.5.2	Hipótesis específicas	44
3.6	Operacionalización de variables	45
3.6.1	Variables	45
3.6.2	Dimensiones	45
3.6.3	Indicadores.	45
3.7	Técnicas e instrumentos.	46
3.7.1	Técnicas de investigación	46
3.7.2	Instrumentos de investigación	47
3.7.3	Herramientas de software utilizadas.	47
3.8	Desarrollo del trabajo de tesis.	49
3.8.1	Recopilación de datos	49
3.8.2	Conjunto de datos entrenamiento.	52
3.8.3	Etiquetado en LabelMe con EfficientSam (accuary)	52
3.8.4	Entrenamiento del modelo 100 épocas-con NVIDIA Tesla T4	60
3.8.5	Métricas de entrenamiento del modelo	67
3.8.6	Determinación de las estaciones de conteo	70
3.8.7	Desarrollo del software	72
3.8.8	Lenguajes de programación y librerías utilizadas para el desarrollo del software	73
3.8.9	Arquitectura del sistema	74
3.8.10	Funcionalidades clave del sistema	76
4	RESULTADOS	78
4.1	Análisis de resultados	78
4.1.1	Procesamiento de Datos y Cuantificación del Tráfico Vehicular	78
4.1.2	Derivación de Parámetros de Diseño para Pavimentos.	78
4.2	Constrastación de hipótesis.	79
4.2.1	Constrastación de hipótesis específica 01.	79
4.2.2	Constrastación de hipótesis específica 02.	89
4.2.3	Constrastación de hipótesis específica 03.	95
4.3	Discusión de resultados	104
4.3.1	Rendimiento y Robustez del Modelo de Detección YOLOv8	104
4.3.2	Comparación con enfoques previos y aporte de la investigación	105
4.3.3	Implicancias Directas en el Diseño de Pavimentos	105

5	CONCLUSIONES	107
5.1	Conclusiones.	107
5.1.1	Conclusión:Objetivo específico 01	107
5.1.2	Conclusión:Objetivo específico 02	107
5.1.3	Conclusión:Objetivo específico 03	107
5.2	Recomendacioness	108
5.3	Trabajos futuros	109
	REFERENCIAS BIBLIOGRÁFICAS	110
6	ANEXOS	114
6.1	Panel fotográfico-recopilación y captura de datos.	114
A	ANEXOS	120
A.1	Matriz de consistencia	120

LISTA DE FIGURAS

1	Dinámica del problema de transporte: Esquema de Manheim	14
2	Formato de conteo y clasificación vehicular(Ministerial, 2023)	16
3	Factores de corrección de vehículos ligeros(Ministerial, 2023)	19
4	Factores de corrección de vehículos pesados(Ministerial, 2023)	19
5	Factores de Crecimiento Acumulado (Fca) (AASHTO, 1993)	20
6	Tasas para la Proyección de la Demanda (Ministerial, 2023)	21
7	Configuración de Ejes (Ministerio de Transportes y Comunicaciones, 2014) .	22
8	Factores de Distribución Direccional y de Carril (AASHTO, 1993)	23
9	Relación de Cargas por Eje Para Afirmados, Pavimentos Flexibles y Semirrígidos (AASHTO, 1993)	24
10	Relación de Cargas por Eje Para Pavimentos Rígidos (AASHTO, 1993)	24
11	Factor de ajuste por presión de neumático (AASHTO, 1993)	25
12	Factor de ajuste por presión de neumático (AASHTO, 1993)	26
13	Arquitectura típica de una red neuronal	26
14	Diagrama de flujo del aprendizaje supervisado	28
15	Diagrama de flujo del aprendizaje no supervisado	28
16	Diagrama de flujo aprendizaje por refuerzo	29
17	Elementos de una red neuronal artificial.	30
18	Esquema de arquitectura típico de una red neuronal convolucional	31
19	Funcionamiento de una CNNI	32
20	Preprocesamiento.	33
21	Función del kernel.	34
22	Capa max pooling	35
23	Representación de la función de activación	37
24	Captura de fotografías-horario madrugada 05:30 am	49
25	Captura de fotografías-horario media mañana 10:30 am	49
26	Captura de fotografías-horario media mañana 13:30 pm	50
27	Captura de fotografías-horario tarde 15:30 pm	50
28	Captura de fotografías-horario noche 21:30 pm	50
29	Etiquetado de la categoría (clase), Trayler-3T3	52
30	Etiquetado de la categoría (clase), Trayler-3T2	53
31	Etiquetado de la categoría (clase), Trayler-2T3	53
32	Etiquetado de la categoría (clase), Trayler-2T2	53
33	Etiquetado de la categoría (clase), Semitrayler-3S3	54
34	Etiquetado de la categoría (clase), Semitrayler-3S2	54
35	Etiquetado de la categoría (clase), Semitrayler-3S1	54
36	Etiquetado de la categoría (clase), Semitrayler-2S3	55
37	Etiquetado de la categoría (clase), Semitrayler-2S2	55
38	Etiquetado de la categoría (clase), Semitrayler-2S1	55
39	Etiquetado de la categoría (clase),Camión-C4E	56

40	Etiquetado de la categoría (clase),Camión-C3E	56
41	Etiquetado de la categoría (clase),Camión-C2E	56
42	Etiquetado de la categoría (clase),Bus-B4E	57
43	Etiquetado de la categoría (clase),Bus-B3E	57
44	Etiquetado de la categoría (clase),Bus-B2E	57
45	Etiquetado de la categoría (clase),Micro-M	58
46	Etiquetado de la categoría (clase),Rural-RC	58
47	Etiquetado de la categoría clase (PU-Camioneta-Pick Up)	58
48	Etiquetado de la categoría clase (Camioneta Panel-PA)	59
49	Etiquetado de la categoría clase (Station Wagon-SW)	59
50	Etiquetado de la categoría clase (Auto-A)	59
51	Histograma de las 22 clases de vehículos	60
52	Gráfico bounding box distribution plot	61
53	Gráfico de dispersión 2D tipo heatmap	61
54	Gráfico bounding box centers	62
55	Gráfico confusion matrix	62
56	Gráfico confusion matrix normalized	63
57	Gráfico scatter plot matrix	63
58	Gráfico F1-Confidence Curve	64
59	Gráfico Precisión-Confidence curve	65
60	Gráfico recall-Confidence curve	65
61	Gráfico Precisión-Recall Curve	66
62	Gráfico métricas de entrenamiento y validación	67
63	Determinación de las estaciones de conteo	70
64	Estación de conteo 01-Grabación en periodo de iluminación	70
65	Estación de conteo 02-Grabación en periodo nocturno	71
66	Estación de conteo 02-Grabación en periodo de iluminación	71
67	Estación de conteo 02-Grabación en periodo nocturno	71
68	Estación de conteo 03-Grabación en periodo de iluminación	72
69	Estación de conteo 03-Grabación en periodo nocturno	72
70	Modelo best_yuvica cargado en el sistema	73
71	Interfaz gráfica de usuario (GUI)	74
72	Definición del sistema de seguimiento (tracking) por ID	75
73	Vídeo exportado día martes	75
74	Configuración del FCE por unidad de peaje	76
75	Mecanismos de control del sistema	77
76	Detección de 2S2-PA-RC-SW	80
77	Detección de RC-B2E-2S2-B4E	80
78	Detección de RC-B2E-PA-SW	80
79	Detección de C4E-B4E-B2E-B4E	81
80	Detección y % de precisión B3E(1.0)-B4E(1.0)-B4E(0.8)-B4E(1.0)	81

81	Detección y % de precisión B4E(0.5)-B3E(1.0)-B3E(1.0)-B4E(1.0)	81
82	Detección y % de precisión B4E(1.0)-B3E(1.0)-B4E(1.0)-B4E(1.0)	82
83	Detección y % de precisión B3E(0.9)-B4E(1.0)-B3E(1.0)-2S2(0.9)	82
84	Detección y % de precisión B4E(1.0)-B4E(1.0)-B4E(1.0)-B4E(1.0)	82
85	Detección y % de precisión 2T3(1.0)-2S3(0.9)-SW(0.6)-2T3(0.5)	83
86	Detección y % de precisión B4E(0.4)-SW(0.9)-B3E(1.0)-2S3(0.8)	83
87	Detección y % de precisión 2S2(0.6)-2S3(1.0)-B3E(0.8)-2S2(1.0)	83
88	Día lunes: Detección de Microbus (M) con tracking ID:9, precisión 96 %	85
89	Día lunes: Detección de Pick Up (PU) con tracking ID:54, registrado en el contador	85
90	Día martes: Detección de SW con tracking ID:38 y A, ID:39	85
91	Día martes: Detección de RC con tracking ID:13 y precisión 96 %	86
92	Día miércoles: Detección de PU con tracking ID:8 y ID:9	86
93	Día miércoles: Detección de PA con tracking ID:24, registrado en el contador	86
94	Día jueves: Detección de M con tracking ID:7, registrado en el contador	87
95	Día jueves: Detección de A con tracking ID:37 y 38 registrado en el contador .	87
96	Día viernes: Detección de C2E con tracking ID:4, registrado en el contador . .	87
97	Día viernes: Detección de C2E con tracking ID:24, 97 % de precisión	88
98	Día sábado: Detección de C2E con tracking ID:51, 97 % de precisión	88
99	Día sábado: Detección de A y PU con tracking ID:66 e ID:65 respectivamente .	88
100	Día domingo: Detección de RC con tracking ID:3, , 97 % de precisión	89
101	Día domingo: Detección de SW Y A con tracking ID:66, ID:67 respectivamente	89
102	Gráfico F1-Confidence Curve	91
103	Gráfico Precisión-Confidence Curve	91
104	Gráfico Recall-Confidence Curve	91
105	Gráfico Precisión-Recall Curve	92
106	Gráfico métricas de entrenamiento y validación	92
107	Interfaz gráfica de usuario (GUI)	97
108	Carpeta de salida para la exportación de videos procesados	97
109	Selección de archivos de vídeo	98
110	Procesamiento del vídeo en tiempo real	98
111	Resultados del conteo vehicular por día/clase-Lunes	99
112	Resultados del conteo vehicular por día/clase-Martes	99
113	Resultados del conteo vehicular por día/clase-Miercoless	99
114	Resultados del conteo vehicular por día/clase-completo	100
115	Selección del departamento geográfico: Ayacucho	100
116	Selección del mes que se realizó el conteo vehicular	101
117	Selección del tipo de pavimento	101
118	Resultados del IMDS para cada tipo de vehículo	102
119	Resultados-Ejes Equivalentes (EE) para cada clase vehicular	103
120	Resultados-Ejes Equivalentes (EE) para cada clase vehicular-completo	103

121	Resultados-Ejes Equivalentes (EE) completo	104
122	Resultados-Ejes Equivalentes (EE) completo	104
123	Recopilación de datos horario 09:41 am	114
124	Recopilación de datos horario 09:50 am	114
125	Recopilación de datos horario 10:16 am	114
126	Recopilación de datos horario 10:17 am	115
127	Recopilación de datos horario 10:19 am	115
128	Recopilación de datos horario 10:32 am	115
129	Recopilación de datos horario 10:36 am	116
130	Recopilación de datos horario 10:40 am	116
131	Recopilación de datos horario 10:40 am-C2E	116
132	Recopilación de datos horario 10:40 am-C2E	117
133	Recopilación de datos horario 16:21 pm	117
134	Recopilación de datos horario 16:15 pm	117
135	Recopilación de datos horario 16:14 pm	118
136	Recopilación de datos horario 16:13 pm	118
137	Recopilación de datos horario 17:15 pm	118
138	Recopilación de datos horario 17:18 pm	119
139	Recopilación de datos horario 17:12 pm	119
140	Recopilación de datos horario 17:15 pm	119

LISTA DE CUADROS

1	Herramientas de software utilizadas.	48
2	Definición de categorías (clases)	51
3	Resultados mAP50 y mAP50-95, 100 épocas	69
4	Tiempos de Procesamiento-100 épocas.	69
5	Resultados mAP50 y mAP50-95, 100 épocas	93
6	Tiempos de Procesamiento-100 épocas.	93
7	Tabla de contingencia - Rendimiento del modelo RNA	95
8	Matriz de consistencia.	120
9	Operacionalización de variables.	121

GLOSARIO

AASHTO	Asociación Americana de Funcionarios Estatales de Carreteras y Transporte. Institución que establece metodologías y normas para el diseño estructural de pavimentos en diversos contextos viales.
IMDA	Indicador que representa el promedio diario de vehículos que transitan por una vía durante un año, fundamental para la planificación vial.
Ejes equivalentes (EE)	Medida que traduce la carga de tránsito en unidades estándar equivalentes a un eje simple cargado, usada en el diseño de pavimentos.
YOLOv8	Última versión del modelo YOLO (You Only Look Once), especializado en detección de objetos en tiempo real con alta precisión y velocidad.
RNA	Sistemas computacionales inspirados en el funcionamiento del cerebro humano, capaces de aprender patrones complejos a partir de grandes volúmenes de datos.
Visión por Computadora	Rama de la inteligencia artificial que permite a las computadoras interpretar y analizar imágenes o videos del entorno.
Inferencia	Proceso mediante el cual un modelo entrenado realiza predicciones o clasificaciones sobre nuevos datos.
Sobreajuste (Overfitting)	Problema durante el entrenamiento de modelos donde se ajustan demasiado a los datos de entrenamiento y pierden capacidad de generalización.
Tiempo de procesamiento por imagen	Intervalo total requerido por el sistema para realizar el análisis de una imagen, incluyendo las etapas de preprocesamiento, inferencia y postprocesamiento.
Arquitectura del modelo	Estructura interna del modelo de inteligencia artificial, que define cómo se procesan los datos y se genera la salida esperada.
Clasificación Vehicular MTC	Esquema oficial del Ministerio de Transportes y Comunicaciones del Perú que define 22 clases vehiculares para fines de análisis vial.
mAP	Métrica que mide el rendimiento global de un modelo de detección de objetos, combinando precisión y exhaustividad.
Tracking ID	Identificador único asignado a cada objeto detectado, usado para rastrear su trayectoria y evitar conteo duplicado.
Tkinter	Biblioteca estándar de Python utilizada para construir interfaces gráficas de usuario de forma rápida y funcional.
OpenCV	Biblioteca de código abierto especializada en visión artificial, procesamiento de imágenes y video en tiempo real.
Ultralytics	Framework de desarrollo e implementación del modelo YOLO, usado para entrenamiento, evaluación e inferencia.
FCE	Parámetro que permite ajustar el IMDS para estimar el IMDA, considerando variaciones temporales del flujo vehicular.
Conteo Vehicular Automatizado	Proceso computacional que permite registrar y clasificar el paso de vehículos mediante herramientas de inteligencia artificial.

ACRÓNIMOS

AASHTO	American Association of State Highway and Transportation Officials. Entidad que define criterios técnicos para el diseño estructural de pavimentos.
IMDA	Índice Medio Diario Anual. Promedio diario de vehículos que transitan por una vía durante un año.
EE	Ejes Equivalentes. Conversión del tránsito vehicular a ejes estándar para el diseño de pavimentos.
RNA	Redes Neuronales Artificiales. Modelo computacional que emula el aprendizaje del cerebro humano.
YOLOv8	You Only Look Once versión 8. Arquitectura de detección de objetos en tiempo real, utilizada en esta investigación.
CNN	Convolutional Neural Network. Red neuronal convolucional usada para visión por computadora.
GUI	Graphical User Interface. Interfaz gráfica que facilita la interacción del usuario con el software desarrollado.
FCE	Factor de Corrección Estacional. Ajuste aplicado al IMDS para estimar el IMDA.
FCA	Factor de Conversión de Aceleración. Parámetro para convertir volúmenes vehiculares en ejes equivalentes.
FVP	Factor de Vehículos Pesados. Porcentaje de participación de vehículos pesados en el flujo total.
FD	Factor de Dirección. Ajuste aplicado en función del sentido del flujo vehicular.
FC	Factor de Crecimiento. Tasa anual de incremento del tránsito vehicular.
ID	Identificador Único. Código asignado a cada vehículo detectado para evitar duplicidades en el conteo.
IoU	Intersection over Union. Métrica para evaluar la superposición entre predicción y verdad real en detección de objetos.
mAP	Mean Average Precision. Promedio de precisión usado para evaluar la calidad del modelo.
OpenCV	Open Source Computer Vision Library. Biblioteca de Python usada para procesamiento de video e imágenes.
Tkinter	Toolkit Interface. Librería utilizada para crear la interfaz gráfica del sistema.
CSV	Comma-Separated Values. Formato de archivo usado para exportar resultados tabulares del sistema.

LISTA DE SÍMBOLOS

a	Aceleración o desaceleración de un vehículo, medida en metros por segundo cuadrado (m/s^2).
C	Capacidad vial de una vía, expresada en vehículos por hora por carril (veh/h/carril).
d	Demora promedio por vehículo en una intersección, expresada en segundos (s).
EE	Número de Ejes Equivalentes acumulados, según metodología AASHTO.
$F1$	Medida F1-score: combinación de precisión y recall. $F1 = 2 \times \frac{P \times R}{P + R}$.
FCE	Factor de Corrección Estacional aplicado al IMDS para obtener el IMDA.
$f(x)$	Función de activación en redes neuronales. Ejemplo: $f(x) = \max(0, x)$ (ReLU).
FC	Factor de Crecimiento utilizado para proyectar el tránsito futuro.
FD	Factor de Dirección, relacionado con el flujo vial unidireccional o bidireccional.
FVP	Factor de Vehículos Pesados. Representa la proporción de tránsito pesado.
g	Tiempo de verde efectivo en un ciclo semafórico, en segundos (s).
$IMDA$	Índice Medio Diario Anual, tráfico promedio diario anual (veh/día).
IoU	Intersection over Union. Medida de solapamiento entre predicción y verdad real.
k	Densidad vehicular en una vía (veh/km).
$\mathcal{L}$	Función de pérdida usada durante el entrenamiento del modelo YOLOv8.
mAP	Precisión Media Promedio. Métrica clave de desempeño en detección de objetos.
P	Precisión: proporción de verdaderos positivos respecto a todas las predicciones positivas.
q	Flujo vehicular (vehículos por hora).
R	Recall o sensibilidad: proporción de verdaderos positivos respecto a los casos reales.
$\sigma(x)$	Función sigmoide: $\sigma(x) = \frac{1}{1+e^{-x}}$, utilizada como activación.
t_p	Tiempo de procesamiento por imagen, que incluye preprocesamiento, inferencia y postprocesamiento.
v	Velocidad media de circulación vehicular (km/h).
w_i	Peso sináptico en la red neuronal asociado a la entrada i .
X	Grado de saturación de la vía: cociente entre volumen y capacidad (v/c).

1. PLANTEAMIENTO DEL PROBLEMA

La recolección manual de datos de tráfico, actualmente utilizada en muchas regiones del país, presenta múltiples limitaciones. Los estudios de tráfico vehicular, que incluyen la medición de la cantidad de vehículos y sus velocidades, son fundamentales para calcular índices medios anuales y diarios de tránsito. Estos estudios son esenciales para el diseño adecuado de pavimentos y la planificación de alternativas que mejoren la transitabilidad. Sin embargo, el uso de formatos en papel y la digitalización manual de los datos no solo son propensos a errores, sino que también requieren una gran cantidad de recursos y tiempo.

Muchos de los softwares actuales de análisis de tráfico vehicular no logran ofrecer una precisión adecuada en la recolección de datos debido a limitaciones tecnológicas y a la falta de integración de métodos avanzados de análisis. A continuación, se describen algunas de las limitaciones de los softwares más conocidos:

- **TrafficVision:** TrafficVision utiliza cámaras para la detección y conteo de vehículos, pero su precisión puede verse afectada por condiciones climáticas adversas y baja visibilidad. Además, enfrenta desafíos en la clasificación precisa de vehículos en situaciones de tráfico denso, lo que afecta la exactitud de los datos recolectados.
- **DataFromSky:** Aunque ofrece análisis avanzado utilizando videos capturados por drones, DataFromSky puede ser costoso y logísticamente complejo de implementar a gran escala. La dependencia de drones introduce variables como la duración de la batería y las restricciones de vuelo, limitando su aplicabilidad continua.
- **Iteris VantageLive!** Iteris VantageLive! es reconocido por su precisión en la detección de vehículos, pero el costo de implementación y mantenimiento puede ser alto para algunas jurisdicciones. Además, su rendimiento depende de cámaras de video, lo que significa que puede degradarse en condiciones de poca luz o mal tiempo, limitando su efectividad en diversas condiciones ambientales.
- **JAMAR Technologies:** Los dispositivos de JAMAR Technologies son efectivos para el conteo de vehículos, pero requieren una configuración manual considerable. Además, tienen limitaciones para realizar mediciones precisas de velocidad en tiempo real.

En particular, la Av. Ramón Castilla en Ayacucho enfrenta problemas de congestión vehicular. Esta avenida es una arteria principal y sufre de tráfico pesado, lo que dificulta realizar un conteo vehicular manual preciso. La congestión constante aumenta la probabilidad de errores en la recolección manual de datos, lo que puede llevar a estimaciones incorrectas del tráfico vehicular real que transita por esta vía. Este problema es más severo durante las horas pico, cuando el flujo de vehículos es máximo y la congestión es mayor.

Si no se realiza un buen análisis de conteo vehicular, no se obtendrán resultados acordes a la realidad en el diseño de pavimentos, lo que puede llevar a decisiones subóptimas en la planificación vial. Las causas de estos problemas incluyen la falta de adopción de tecnologías avanzadas de recolección de datos y la dependencia de métodos manuales obsoletos. La mayoría de los softwares actuales no implementan algoritmos de inteligencia

artificial avanzados, lo que podría mejorar significativamente la precisión y eficiencia del análisis de tráfico vehicular.

La falta de datos precisos y actualizados puede llevar a decisiones subóptimas en la planificación y diseño de pavimentos, incrementando los costos de mantenimiento y reduciendo la eficiencia del sistema de transporte. Esto, a su vez, afecta la productividad de los trabajadores, aumenta los costos operativos de las empresas y disminuye la calidad de vida de la población.

1.1. Delimitación del problema

1.1.1. Espacial (geográfica)

Si bien el modelo entrenado puede realizar una inspección en cualquier red vial urbano pavimentada, es necesario para esta investigación delimitar el espacio geográfico para la aplicabilidad de la investigación, el cual se ha definido en la Av. Ramón Castilla del distrito de Ayacucho – Huamanga, de la región de Ayacucho.

Respecto al set de datos para el entrenamiento se ha evaluado realizar la recolección de imágenes en la ciudad de Ayacucho, los cuales serán de diferentes tipos de vehículos definidos en el Manual de carreteras Ministerio de Transportes y Comunicaciones (2014).

1.1.2. Temporal

Esta investigación se enmarca temporalmente entre septiembre de 2024 y junio de 2025. Durante este periodo se llevó a cabo la recolección de información esencial, seguida del procesamiento analítico y la interpretación de los hallazgos obtenidos a partir del estudio.

1.1.3. Temática y unidad de análisis

Las unidades de análisis será el análisis del tránsito vehicular que involucra el aforo vehicular y velocidades de los vehículos aplicado al diseño estructural de pavimentos rígidos, los cuales serán identificados mediante un modelo de Redes Neuronales Artificiales.

1.2. Formulación del problema

1.2.1. Problema general

¿De qué manera un modelo basado en Redes Neuronales Artificiales (RNA) mejora y optimiza la precisión y eficiencia en la detección y conteo de vehículos para la determinación de parámetros críticos en el diseño de pavimentos, como el Índice Medio Diario Anual (IMDA) , Número de repeticiones de Ejes Equivalentes (ESAL), según las normativas del Ministerio de Transportes y Comunicaciones (MTC).?

1.2.2. Problemas específicos

1. ¿Cómo el diseño del modelo basado en Redes Neuronales Artificiales (RNA) permite ajustar una arquitectura para la detección, clasificación y conteo eficiente de vehículos basado en el formato de clasificación vehicular del MTC.?
2. ¿En qué medida el entrenamiento del modelo basado en Redes Neuronales Artificiales

(RNA) contribuye a lograr un balance óptimo entre la exactitud en la detección, la cobertura en el reconocimiento de vehículos?

3. ¿De qué manera la implementación del modelo basado en Redes Neuronales Artificiales (RNA) permite desarrollar e implementar un software para la obtención automática del IMDA y ESAL en cualquier red vial, optimizando los procesos de cálculo en la planificación y diseño estructural de pavimentos?

1.3. Justificación e importancia

En la presente era, las infraestructuras viales urbanas pavimentadas desempeñan un papel crucial en el progreso económico y social de la ciudad de Ayacucho al facilitar la movilidad e integración de sus habitantes. conforme al informe de Visión de Desarrollo de la Infraestructura Vial. Para asegurar la durabilidad y eficiencia de estas vías, es fundamental realizar evaluaciones precisas y recopilar datos pertinentes para un diseño estructural óptimo. Una parte esencial de este proceso implica la obtención eficiente y precisa de datos relacionados con el estudio de tráfico, específicamente el conteo vehicular y la medición de velocidades.

El desarrollo de esta investigación permitirá la creación de un modelo de red neuronal artificial diseñado para analizar el tráfico vehicular, posibilitando la cuantificación precisa del flujo vehicular. Esto contribuirá significativamente a un diseño estructural eficaz de pavimentos. En última instancia, estos beneficios se traducirán directamente en mejoras para los usuarios y la población en la zona de influencia de la vía, quienes experimentarán un servicio de transitabilidad más seguro y eficiente.

1.4. Limitaciones de la investigación

La presente investigación, presenta ciertas limitaciones que deben ser reconocidas:

- **Limitaciones tecnológicas:** Aunque se dispone de equipos y herramientas avanzadas para la recolección de datos, el rendimiento del modelo de Redes Neuronales Artificiales puede verse limitado por la calidad y resolución de las imágenes y vídeos capturados.
- **Variabilidad del tráfico:** La variabilidad en los tipos de vehículos y comportamientos de conducción en diferentes avenidas y regiones puede representar un desafío para la generalización del modelo a otras áreas fuera del distrito de Ayacucho.
- **Impacto humano y errores:** A pesar de la automatización, la intervención humana en la fase de etiquetado y validación de datos puede introducir errores. Además, el proceso de ajuste y entrenamiento del modelo depende en gran medida de la experiencia y habilidades del equipo de investigación.

1.5. Objetivos de la investigación

1.5.1. Objetivo general

Desarrollar un modelo basado en Redes Neuronales Artificiales (RNA) para mejorar y optimizar la precisión y eficiencia en la detección y conteo de vehículos para la

determinación de parámetros críticos en el diseño de pavimentos, como el Índice Medio Diario Anual (IMDA) , Número de repeticiones de Ejes Equivalentes (ESAL), según las normativas del Ministerio de Transportes y Comunicaciones (MTC)

1.5.2. Objetivos específicos

1. Desarrollar el diseño del modelo basado en Redes Neuronales Artificiales (RNA) para ajustar una arquitectura que permita la detección,clasificación y conteo eficiente de vehículos basado en el formato de clasificación vehicular del MTC
2. Realizar el entrenamiento del modelo basado en Redes Neuronales Artificiales (RNA) que permita lograr un balance óptimo entre la exactitud en la detección, la cobertura en el reconocimiento de vehículos.
3. Desarrollar e implementar un software con interfaz gráfica de usuario (GUI) para la obtención automática del IMDA y ESAL en cualquier red vial, optimizando los procesos de cálculo en la planificación y diseño estructural de pavimentos.

2. MARCO TEÓRICO

2.1. Antecedentes

Dentro de los modelos de análisis de tráfico vehicular y diseño estructural de pavimentos se vienen desarrollando modelos convencionales y no convencionales del tipo adaptativo que se basan en la inteligencia artificial, redes neuronales logrando grandes resultados en tiempos, costos y eficiencia.

Podemos citar algunas investigaciones que han propuesto modelos de tráfico vehicular y diseño estructural de pavimentos y algunas investigaciones que han empleado redes neuronales para caracterizar el comportamiento del pavimento.

2.1.1. Investigaciones internacionales

Marin (2014) desarrollo una plataforma informática denominada AVP. El objetivo de esta herramienta era modernizar el proceso de aforo vehicular, permitiendo el conteo, la clasificación y el cálculo de la velocidad media de los vehículos a partir del análisis de video. El sistema fue diseñado para ser flexible, aceptando como entrada videos de diversas fuentes, como una cámara USB, una cámara IP en tiempo real o archivos de video preexistentes en formato AVI.

La base tecnológica de esta plataforma se centró en el uso de la librería de visión por computadora OpenCV junto con el lenguaje de programación C++. Para la detección y seguimiento de los vehículos, Marin (2014) empleó la técnica de sustracción de fondo mediante el método de mezclas gaussianas. Este enfoque consiste en modelar los píxeles del fondo estático de la escena para luego identificar los objetos en movimiento (vehículos) al restarlos de ese fondo. Adicionalmente, se implementaron funciones para medir el desplazamiento, reconocer colores y estimar la velocidad. Los resultados del análisis eran exportados a un archivo de texto en formato CSV, facilitando su posterior uso en informes y estudios de tráfico. Este sistema representa un esfuerzo significativo para automatizar una tarea tradicionalmente manual, utilizando las herramientas de visión artificial consolidadas de la época.

Comparativa y discusión con Marin (2014)

Si bien funcional, esta técnica es susceptible a errores por cambios de iluminación y condiciones ambientales, y ofrece una clasificación vehicular genérica basada en tamaño. En contraste, la presente investigación da un salto tecnológico al implementar una Red Neuronal Convolucional (YOLOv8), un método de aprendizaje profundo inherentemente más robusto y preciso. Este modelo no depende de un fondo estático, sino que aprende a identificar las características intrínsecas de los vehículos, lo que le permite operar con alta fiabilidad en escenarios complejos de tráfico, oclusión parcial y condiciones de luz variables, superando las limitaciones del enfoque anterior.

Esta diferencia metodológica se traduce en un propósito de ingeniería fundamentalmente más avanzado. Mientras que el sistema de Marin (2014) se enfoca en el aforo general

para análisis de tráfico, la presente investigación está diseñado como una herramienta especializada para la ingeniería de pavimentos. La principal innovación es la capacidad del sistema para clasificar vehículos en 22 categorías específicas, alineadas con la normativa del MTC de Perú.

Sisalima (2018) desarrolló un sistema de conteo vehicular fundamentado en técnicas de visión artificial. Esta solución fue concebida como un progreso para el sector de la gestión del tráfico, facilitando la toma de decisiones respaldadas en los datos recopilados por el software. El proyecto incluyó una evaluación de diversos algoritmos de procesamiento de imágenes y la selección de la alternativa más apropiada para abordar el problema específico del conteo de vehículos. Para su implementación, la aplicación fue construida en C++ empleando el framework Qt y se apoyó en la librería OpenCV, que proporciona múltiples algoritmos para la detección y seguimiento de objetos. La integración de estas herramientas permitió desarrollar un software operativo que satisfizo los requerimientos establecidos.

Comparativa y discusión con Sisalima (2018)

La investigación de Sisalima (2018) se alinea con el de Marin (2014) en el uso de herramientas de visión artificial convencionales, como la librería OpenCV, para abordar el problema del conteo vehicular. Su enfoque, centrado en evaluar y seleccionar el algoritmo "más apropiado" de un conjunto existente, es representativo de la metodología clásica de procesamiento de imágenes. Si bien este enfoque es válido para crear sistemas de monitoreo de tráfico general, su alcance es inherentemente limitado por la tecnología subyacente. En contraste, la presente investigación no busca optimizar un algoritmo tradicional, sino que introduce un paradigma completamente diferente: el aprendizaje profundo con Redes Neuronales Convolucionales (YOLOv8). Esta elección representa un avance sustancial en términos de precisión y, sobre todo, de resiliencia ante las complejidades del mundo real, como la oclusión vehicular y las condiciones de iluminación adversas, que a menudo desafían a los algoritmos clásicos.

Más importante aún es la divergencia en el propósito de ingeniería. El sistema de Sisalima (2018) se enmarca en la gestión general del tráfico, donde el conteo es el fin principal. La presente investigación, sin embargo, utiliza el conteo y la clasificación como el punto de partida para un análisis de ingeniería civil altamente especializado. El objetivo no es solo saber cuántos vehículos pasan, sino identificarlos con una precisión y granularidad (22 clases del MTC) que permita alimentar directamente los modelos de diseño de pavimentos. La innovación clave es, por tanto, la capacidad del sistema para automatizar el cálculo del Número de Repeticiones de Ejes Equivalentes (EE), un parámetro crítico que los sistemas de conteo general como el de Sisalima (2018) no están diseñados para abordar. De este modo, nuestra propuesta transforma una herramienta de monitoreo en una solución integral para el diseño de infraestructura vial.

Segarra-López, Cobos-Torres et al. (2022) plantean un algoritmo para examinar videos de tránsito vehicular, tanto grabaciones como en tiempo real. El modelo creado puede

identificar, clasificar y contar tres categorías de vehículos empleando inteligencia artificial y visión computacional, con TensorFlow y OpenCV. El proceso se estructura en tres etapas: adquisición de datos (construcción de una base de datos con modelos vehiculares), desarrollo de la red (empleando AlexNet, una red neuronal convolucional eficaz en clasificación de imágenes) y procesamiento de la información (extracción, conteo y clasificación de vehículos en movimiento). El algoritmo evidencia una efectividad superior al 90 % en comparación con métodos manuales y ha demostrado ser consistente en diversos horarios y condiciones de iluminación.

Comparativa y discusión con Segarra-López, Cobos-Torres et al. (2022)

La investigación de Segarra-López, Cobos-Torres et al. (2022) marca una transición importante desde los métodos de visión artificial clásica hacia el aprendizaje profundo, compartiendo con la presente investigación el uso de Redes Neuronales Convolucionales (CNN) como núcleo tecnológico. Ambos proyectos reconocen la superioridad de la inteligencia artificial para lograr una clasificación robusta. Sin embargo, una diferencia técnica clave reside en la arquitectura de red empleada: mientras Segarra et al. utilizan AlexNet, una arquitectura pionera pero relativamente antigua en el campo del deep learning, nuestra investigación implementa YOLOv8, un modelo de última generación significativamente más rápido y preciso para la detección de objetos en tiempo real. Este salto arquitectónico se traduce en un mayor rendimiento y eficiencia en el procesamiento de video.

La distinción más fundamental, no obstante, se encuentra en la profundidad de la clasificación y su finalidad ingenieril. El modelo de Segarra-López, Cobos-Torres et al. (2022) se limita a tres categorías vehiculares genéricas, lo cual es adecuado para un análisis de flujo de tráfico general pero insuficiente para aplicaciones especializadas. En cambio, el objetivo central de la presente tesis es resolver un problema específico del diseño de pavimentos. Para ello, nuestro sistema fue entrenado para identificar 22 clases de vehículos según la normativa del MTC Perú.

Ramos Castro et al. (2023) desarrolló un proyecto para la automatización del conteo de vehículos utilizando inteligencia artificial, con el doble propósito de medir la Intensidad Media Diaria (IMD) y evaluar el impacto acústico de las carreteras en edificaciones próximas. Para lograrlo, aplicó modelos de aprendizaje profundo, específicamente redes neuronales convolucionales, que permiten identificar y clasificar vehículos en videos de cámaras de vigilancia (CCTV). En su metodología, empleó y comparó diversos modelos pre-entrenados para detectar categorías generales de vehículos (ligeros, pesados y turismos), evaluando finalmente el desempeño de cada uno en términos de exactitud, acierto y velocidad de procesamiento.

Comparativa y discusión con Ramos Castro et al. (2023)

La investigación de Ramos Castro et al. (2023), de manera similar a la presente investigación, se sitúa en la vanguardia del análisis de tráfico al emplear redes neuronales

convolucionales, reconociendo el potencial del aprendizaje profundo para superar los métodos tradicionales. Ambos estudios utilizan esta tecnología para automatizar la recolección de datos a partir de videos. Sin embargo, la similitud tecnológica da paso a una divergencia fundamental en cuanto a la finalidad ingenieril y la especificidad del análisis. El trabajo de Ramos tiene un doble objetivo: la medición del IMD para gestión de tráfico y un análisis de impacto ambiental (acústica), aplicaciones para las cuales una clasificación vehicular en categorías amplias (ligeros, pesados) es suficiente.

Kadim et al. (2021) en su artículo plantea el conteo de vehículos en tiempo real en una escena compleja para la estimación del flujo de tráfico empleando una red neuronal convolucional de múltiples niveles. Al aprovechar las capacidades de los modelos de aprendizaje profundo para delinear y clasificar objetos en una imagen, se evidencia que el sistema puede alcanzar una exactitud de conteo promedio de 97,53 % y un promedio ponderado de conteo con precisión de clasificación de 91,5 % validado en 585 minutos de videos de carretera recopilados desde cuatro cámaras distintas; observados desde diferentes ángulos del vehículo.

Comparativa y discusión con Kadim et al. (2021)

La investigación de Kadim et al. (2021) se enfoca en resolver un desafío central de la visión por computadora: lograr un conteo vehicular de alta precisión en escenarios complejos y variables. Su éxito, con una exactitud superior al 97 %, demuestra la madurez de las redes neuronales convolucionales para tareas de monitoreo de tráfico. Su contribución principal reside en la robustez del método de detección en sí mismo. La presente investigación se apoya en esa misma base tecnológica robusta, pero la orienta hacia un propósito de ingeniería civil que el trabajo de Kadim et al. (2021) no explora.

Biswas et al. (2017) en su artículo implementó el método de sustracción de fondo gaussiano y el marco OverFeat para contar automóviles. El marco OverFeat constituye una combinación de red neuronal convolucional (CNN) y un clasificador de aprendizaje automático (como máquinas de vectores de soporte (SVM) o regresión logística). Con este estudio, demostraron otra posible área de aplicación para el marco OverFeat.

Comparativa y discusión con Biswas et al. (2017)

El enfoque de Biswas et al. (2017) es interesante porque representa un punto de transición entre la visión artificial clásica y las soluciones de aprendizaje profundo. Al combinar la sustracción de fondo con una red neuronal (OverFeat), buscaron aprovechar lo mejor de ambos mundos en ese momento. Sin embargo, este enfoque de múltiples pasos contrasta con la metodología de la presente investigación, que implementa YOLOv8, un modelo de aprendizaje profundo end-to-end. Este último no requiere de pre-procesamientos como la sustracción de fondo, ya que la red neuronal gestiona todo el proceso de detección y clasificación de forma integrada, lo que resulta en un sistema más eficiente y menos propenso a los errores acumulados entre etapas.

Más allá de la arquitectura, la diferencia clave reside en la finalidad y especificidad de la

tarea. El estudio de Biswas et al. (2017). se enfoca en el conteo de automóviles como una prueba de concepto para el framework OverFeat. Su objetivo es validar una herramienta. Por el contrario, la presente investigación está concebida para resolver un problema de ingeniería concreto: el diseño de pavimentos.

2.1.2. Investigaciones nacionales

Pezo (2015) llevó a cabo un estudio del comportamiento del tráfico vehicular a través del cálculo del flujo de congestión, utilizando un sistema de visión artificial implementado en una maqueta que representaba la vía con objetos simulando vehículos. En el experimento se compararon dos conjuntos de datos: uno generado manualmente por observación directa del investigador y otro producido por el sistema automatizado. Se desarrolló e implementó el software de visión artificial y se simuló distintos escenarios en la maqueta para obtener el número de vehículos presentes, lo que permitió calcular el flujo vehicular. A lo largo de la investigación se realizaron siete pruebas, contrastando los resultados de ambos métodos de análisis. Los hallazgos confirmaron la hipótesis alternativa, demostrando que la visión artificial es capaz de estimar con precisión el flujo de congestión en una vía transitable.

Comparativa y discusión con Pezo (2015)

La investigación de Pezo (2015) constituye una valiosa prueba de concepto que demuestra la viabilidad de la visión artificial para el análisis de tráfico. Sin embargo, su enfoque se distingue del presente proyecto en dos dimensiones cruciales: el entorno de aplicación y el objetivo de ingeniería. El estudio de Pezo se desarrolló en un entorno de laboratorio controlado (una maqueta), lo que simplifica enormemente el problema al eliminar las variables impredecibles del mundo real como cambios de iluminación, condiciones climáticas y oclusiones complejas. En contraste, nuestra investigación aborda directamente el desafío de analizar videos de tráfico en una avenida real, enfrentando la variabilidad y el ruido inherentes a un escenario no controlado.

Gonzales del Valle Romero y Neyra Espinoza (2022) diseñaron un sistema inteligente para detectar y contabilizar vehículos en una zona específica del estacionamiento de la Universidad Ricardo Palma. Su propósito principal fue determinar la disponibilidad de espacios y optimizar el trabajo del personal de seguridad. Para ello, se utilizaron modelos de redes neuronales convolucionales, entre ellos YOLOv5 y Faster R-CNN, optimizados mediante la técnica de Transfer Learning.

Se construyó un conjunto de datos de entrenamiento con 460 imágenes obtenidas tanto de vehículos reales como de fuentes en línea. Para el conteo de automóviles se aplicaron técnicas basadas en Líneas de Interés (LOI) y Regiones de Interés (ROI), siendo esta última la elegida por su mayor precisión. El sistema cuenta con una interfaz gráfica desarrollada en Tkinter que muestra los resultados de detección en tiempo real. En la fase final del proyecto, tras comparar métricas como precisión promedio (mAP), velocidad de procesamiento (FPS) y tasas de error, se optó por la red YOLO. La implementación se realizó usando un

teléfono móvil ubicado en un punto estratégico del campus, conectado a una computadora mediante la red de la universidad, permitiendo así validar el funcionamiento del sistema en condiciones reales.

Comparativa y discusión con Gonzales del Valle Romero y Neyra Espinoza (2022)

La investigación de Gonzales del Valle Romero y Neyra Espinoza (2022) comparte una base tecnológica muy similar a la de la presente tesis, ya que ambos proyectos implementan con éxito la arquitectura YOLO, demostrando su eficacia para la detección vehicular en tiempo real. Sin embargo, la similitud en la herramienta da paso a una profunda diferencia en el ámbito de aplicación y la complejidad del problema abordado. El trabajo de Gonzales resuelve un problema a microescala y en un entorno semi-controlado: la gestión de un estacionamiento. En este contexto, el objetivo es logístico y la tarea de clasificación es binaria (espacio ocupado/vacío) o se limita a una única clase ('vehículo').

Estofanero Yanapa et al. (2022) realizaron un estudio comparativo entre dos enfoques para la detección vehicular: Uno basado en redes neuronales convolucionales y otro utilizando visión artificial a través de librerías especializadas. Ambos métodos se evaluaron frente a un conteo manual realizado en campo, considerado como referencia de máxima precisión (100 %). La prueba se efectuó sobre una grabación real de tráfico, clasificando seis tipos de vehículos: motocicletas, mototaxis, automóviles, camionetas, microbuses y vehículos pesados.

Tras aplicar ambos enfoques, se obtuvo una precisión del 95.60 % en la detección y clasificación de microbuses usando inteligencia artificial, y un 97.80 % con visión artificial tradicional. A partir de los resultados, se concluyó que los sistemas basados en inteligencia artificial presentan un desempeño adecuado para el conteo y clasificación vehicular, con una capacidad comparable a la observación humana, lo que los convierte en una alternativa efectiva para la automatización del aforo vehicular.

Comparativa y discusión con Estofanero Yanapa et al. (2022)

El estudio de Estofanero Yanapa et al. (2022) es de gran relevancia, ya que aborda una pregunta fundamental: ¿cómo se compara la inteligencia artificial con los métodos de visión computacional tradicionales y el conteo manual? Su investigación concluye que la IA es una alternativa viable y eficaz, validando su uso para la automatización del aforo vehicular. Sin embargo, el propósito de su trabajo es evaluativo y comparativo, enfocado en legitimar una tecnología. La presente investigación parte de esa premisa ya validada para llevar la aplicación a un nivel superior de especialización

2.2. Bases teóricas

2.2.1. Tráfico vehicular

El tráfico vehicular se manifiesta como el resultado del desplazamiento continuo de automóviles a lo largo de una infraestructura vial, como calles, avenidas o autopistas. Para llevar a cabo un diseño geométrico adecuado de una vía, es indispensable

comprender previamente las particularidades del tránsito que la recorrerá. El estudio de los componentes del flujo vehicular permite identificar su comportamiento y características operativas, lo cual constituye un aspecto fundamental en la planificación, formulación y ejecución de proyectos viales, así como en las obras complementarias que forman parte del sistema de transporte (Mozo Sánchez, 2012).

Por su parte, Manheim (1984) plantea una visión integral en la que el tráfico se analiza como un fenómeno físico que puede modelarse. Su propuesta metodológica, considerada una referencia clásica en el campo del transporte, permite comprender la dinámica del tráfico desde una perspectiva sistémica, facilitando su representación mediante esquemas conceptuales (Fernandez, 2011).

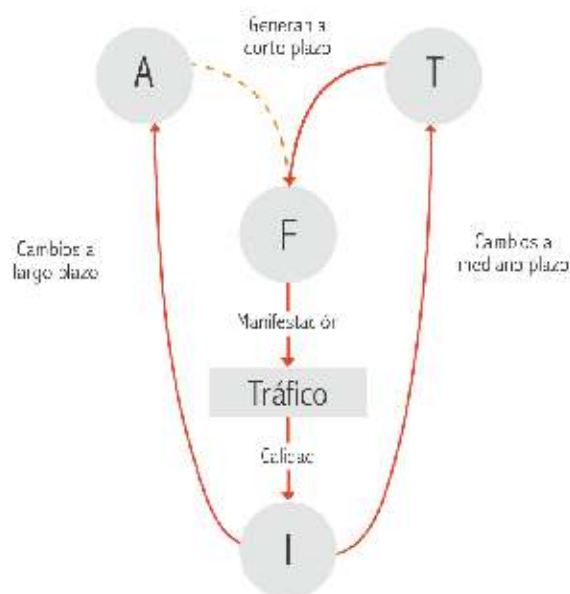


Figura 1: Dinámica del problema de transporte: Esquema de Manheim
Fuente: Obtenido de Fernandez (2011)

Donde:

A: Sistema de actividades en un área geográfica.

T: Sistema de transporte en un área geográfica.

F= Patrón de viajes en el área.

I= Conjunto de impactos derivados del tráfico.

2.2.1.1 Características del Tráfico vehicular

Mozo Sánchez (2012) explica que el flujo vehicular se describe mediante tres variables clave: el flujo, la velocidad y la densidad. A través de la relación entre estas variables es posible identificar el comportamiento de la corriente de tránsito. Estas características pueden agruparse en dos grandes categorías: los parámetros macroscópicos, que describen el comportamiento general de los vehículos como grupo; y los parámetros microscópicos, que se centran en cómo interactúan los vehículos individualmente dentro del flujo.

2.2.2. Estudio de tráfico

El objetivo fundamental de un análisis de tráfico consiste en determinar tanto el volumen como las categorías de vehículos que transitarán por una vía específica. Esta información resulta esencial para establecer los parámetros básicos en el diseño geométrico y estructural de la infraestructura vial, así como para evaluar su capacidad y los niveles de servicio, tanto en las condiciones presentes como en escenarios futuros. Uno de los datos fundamentales que debe suministrar este análisis es el Índice Promedio Diario Anual (IPDA) correspondiente a cada segmento vial evaluado. De igual manera, se necesita conocer la distribución del tránsito conforme a los diferentes tipos de vehículos. La estimación de los ESALs, es decir, el número de repeticiones de ejes equivalentes de 8.2 toneladas, representa un insumo básico para el diseño estructural de pavimentos (Ministerio de Transportes y Comunicaciones, 2014).

Esta clase de evaluación resulta indispensable previo a ejecutar cualquier intervención sobre la vía, ya sea una mejora del trazado existente o una nueva proyección. Su utilidad se basa en la capacidad de prever los volúmenes de tráfico que emplearán la vía y su comportamiento a través del tiempo, lo cual facilita optimizar las decisiones de inversión en función de la rentabilidad esperada, los costos operativos y el retorno económico vinculado al proyecto vial (Bravo León & Jara Ávila, 2023).

2.2.2.1 Conteos de tráfico vehicular

Es fundamental que las estaciones de conteo vehicular se seleccionen de forma estratégica y con el debido sustento técnico, garantizando al menos una instalación por cada tramo vial homogéneo. En el caso de estaciones principales, el periodo mínimo de registro debe ser de siete días consecutivos, mientras que en estaciones de cobertura se requieren cinco días, con monitoreo continuo durante las 24 horas del día. Los datos recogidos deben ser volumétricos y clasificados por tipo de vehículo, en concordancia con las disposiciones del Reglamento Nacional de Vehículos, conforme al Decreto Supremo N.º 058-2003-MTC.

En lo que respecta a las vías pertenecientes a la Red Vial Vecinal, se exige la ejecución de conteos durante un mínimo de cuatro días consecutivos, también con cobertura de 24 horas por estación, incluyendo obligatoriamente un día no laborable. Toda la información recolectada debe organizarse de acuerdo con la clasificación vehicular correspondiente, haciendo uso de los formatos oficiales proporcionados por el MTC, tales como la hoja de cálculo A.5 de la FTE o el aplicativo digital respectivo (Ministerial, 2023).

Figura 2: Formato de conteo y clasificación vehicular (Ministerial, 2023)
Fuente: Imagen obtenida de la Ficha del MTC

2.2.2.2 Objetivo de Conteos Vehicular

Según lo indicado por Huanca y Paucar (2017), uno de los datos fundamentales en los estudios de transporte y desarrollo de infraestructura vial es la intensidad de circulación, ya que constituye la base para la planificación, diseño y operación de redes viales. Para estimar esta variable, es indispensable llevar a cabo conteos vehiculares en puntos específicos ya sean tramos o intersecciones durante un periodo determinado. Dichos aforos pueden realizarse de forma manual o con equipos especializados, y pueden incluir distintos niveles de desagregación según el tipo de vehículo.

En este contexto, es esencial analizar los recursos disponibles para realizar los aforos, establecer criterios adecuados para seleccionar las secciones de medición, y definir la duración más conveniente del muestreo. A partir de los datos obtenidos en estos aforos, se podrán aplicar los procedimientos adecuados para generar la información necesaria para la investigación.

2.2.2.3 Técnicas de Conteos

Al seleccionar un tipo de conteo vehicular, es fundamental considerar dos aspectos clave: la viabilidad económica de los recursos disponibles y los resultados que se espera obtener. Sin embargo, cada método presenta limitaciones. Por ejemplo, los dispositivos automáticos pueden no ser eficaces cuando se requiere analizar maniobras como giros o la composición detallada del flujo vehicular, especialmente si las condiciones físicas de la vía impiden instalar adecuadamente los sensores. Asimismo, los conteos manuales presentan dificultades operativas en situaciones de alto volumen vehicular o cuando se necesita registrar datos continuamente durante largos periodos de tiempo (Fleitas González et al., 2023).

2.2.2.4 Conteo Manual

Este método se basa en la observación directa de la vía por parte de una persona que contabiliza cada vehículo que transita durante un intervalo de tiempo definido.

Para facilitar esta tarea, se suelen utilizar formatos impresos con categorías vehiculares predefinidas, y en algunos casos, herramientas manuales de conteo. Actualmente, existen aplicaciones móviles que permiten realizar esta labor de forma más ágil, convirtiéndose en una herramienta útil por su versatilidad y facilidad de uso. Una de las principales ventajas del conteo manual es su capacidad para diferenciar tipos específicos de vehículos; no obstante, su alto costo y demanda operativa hacen inviable su uso prolongado, especialmente para registros que excedan las 24 horas. Por ello, suele emplearse como complemento a los sistemas automáticos, con el fin de obtener datos que no pueden recopilarse con otros métodos (Carvajal Flores, 2022).

2.2.2.5 Conteo automático

Este método se basa en el uso de dispositivos que permiten la detección automática del paso de vehículos para su posterior conteo y registro. Uno de los instrumentos más utilizados en este tipo de aforo son los tubos neumáticos, los cuales funcionan mediante un captador conformado por un tubo de goma instalado de forma transversal sobre la calzada. Este sistema también incluye un detector compuesto por una membrana con dos láminas metálicas que hacen contacto entre sí cuando un vehículo pasa por encima.

Además de los neumáticos, existen otros dispositivos como los detectores de lazo, también conocidos como bucles inductivos, que, aunque en menor uso, operan mediante el principio de inducción electromagnética (Bañón & Beviá García, 2000).

2.2.3. Determinación del tráfico actual

La estimación puede realizarse a partir de mediciones de tránsito vehicular en las rutas que impactan directamente la vía dentro de la región, mediante estudios de origen y destino, o recurriendo a variables socioeconómicas representativas que reflejen fielmente la dinámica económica local (Huanca & Paucar, 2017).

2.2.4. Índice Medio Diario Anual (IMDA)

El Índice Medio Diario Anual (IMDA) corresponde al valor promedio de tránsito diario calculado sobre los 365 días del año en una sección vial determinada. Este indicador resulta esencial para que los proyectistas definan parámetros fundamentales de diseño, establezcan la categoría funcional de la vía y formulen estrategias de conservación, mantenimiento y mejora. Asimismo, el IMDA es clave en la planificación vial, ya que orienta la proyección de nuevas rutas, la programación de intervenciones en pavimentos, el análisis de tendencias de tránsito y el planteamiento de criterios geométricos generales. En términos cuantitativos, el IMDA se calcula multiplicando el Tráfico Promedio Diario Semanal (TPDS) por el Factor de Corrección Estacional (FCE) (Comunicaciones, 2018):

$$IMDA = TPDS \times FCE$$

Donde:

- **TPDS:** Tráfico Promedio Diario Semanal
- **FCE:** Factor de Corrección Estacional

Por su parte, Gordillo Vilchez y Rabanal Rengifo (2020) indica que el IMDA resulta de sumar el total de vehículos que circulan a lo largo del año en un tramo específico de la vía. Esta métrica es determinante para valorar la relevancia funcional de la infraestructura, evaluar su viabilidad económica y estimar adecuadamente la demanda a lo largo del periodo de diseño. El dato diario promedio de circulación, junto con la tasa de crecimiento anual, permite proyectar la demanda futura, ya sea mediante métodos manuales o sistemas automatizados, fortaleciendo la gestión de seguridad vial y la evaluación de capacidad operativa de la vía.

2.2.4.1 Tráfico Promedio Diario Semanal (TPDS)

El Tráfico Promedio Diario Semanal (TPDS) representa el valor medio de vehículos que transitan diariamente por un punto específico de una vía durante un periodo continuo de siete días. Este indicador se obtiene al dividir el número total de vehículos registrados en una semana, conocido como Tráfico Semanal (TS), entre siete días (Galo Miranda & Pérez Ordóñez, 2016).

Para calcular el TPDS, es fundamental disponer del total de vehículos que han circulado por la vía durante toda la semana. Este valor puede corresponder a una semana completa (siete días) o, dependiendo del caso, a un periodo de cinco a siete días. Una vez obtenido el TS, se aplica la siguiente fórmula (Sánchez Salazar, 2018):

$$TPDS = \frac{TS}{7} \quad (1)$$

Donde:

- **TPDS:** Tráfico Promedio Diario Semanal.
- **TS:** Tráfico Semanal, es decir, la suma total de vehículos que han transitado por el punto de estudio durante una semana.

2.2.4.2 Factor de Corrección Estacional (FCE)

Prado Ramos (2018) explica que el Factor de Corrección Estacional (FCE) es un valor utilizado para ajustar los conteos semanales de vehículos con el propósito de estimar una representación promedio del flujo vehicular a lo largo del año. Este ajuste busca reflejar con mayor precisión las variaciones estacionales que impactan el tránsito de personas y carga, las cuales pueden estar condicionadas por factores externos como las vacaciones, ciclos agrícolas, condiciones meteorológicas o celebraciones especiales. Para determinar este factor, se utilizan datos históricos de tráfico, usualmente recopilados en estaciones de peaje o mediante estudios adicionales.

Por su parte, el Índice Medio Diario Anual (IMDA) se calcula aplicando factores de corrección diferenciados para vehículos livianos y pesados, ajustados por tipo de estación de peaje, a los resultados obtenidos en los conteos vehiculares de campo. Estos factores se encuentran disponibles en la hoja "A.3. Factores de Corrección" dentro del aplicativo correspondiente a la Ficha Técnica Estándar (FTE) del MTC (Ministerial, 2023).

Item	Proj	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495	2496	2497	2498	2499	2500
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Figura 3: Factores de corrección de vehículos ligeros(Ministerial, 2023)
Fuente: Imagen obtenida de la Ficha del MTC

Table 1. Summary of the results of the 2010-2011 season. The data are presented in the following order: Country, Year, Sex, Age, Height, Weight, BMI, Blood pressure (mmHg), Heart rate (b/min), Blood glucose (mmol/L), Blood cholesterol (mmol/L), Blood triglycerides (mmol/L), Blood uric acid (mmol/L).												
Country	Year	Sex	Age	Height	Weight	BMI	Blood pressure	Heart rate	Blood glucose	Blood cholesterol	Blood triglycerides	Blood uric acid
Algeria	2010	M	20-29	175.0	75.0	24.5	120/80	75	5.0	180	1.5	0.40
Algeria	2010	F	20-29	165.0	65.0	23.9	115/75	70	4.5	170	1.4	0.35
Algeria	2010	M	30-39	178.0	80.0	25.3	125/85	78	5.5	190	1.6	0.45
Algeria	2010	F	30-39	168.0	68.0	24.4	120/80	75	5.0	180	1.5	0.40
Algeria	2010	M	40-49	180.0	85.0	26.1	130/90	80	6.0	200	1.7	0.50
Algeria	2010	F	40-49	170.0	70.0	24.7	125/85	78	5.5	190	1.6	0.45
Algeria	2010	M	50-59	182.0	90.0	27.2	135/95	85	6.5	210	1.8	0.55
Algeria	2010	F	50-59	172.0	72.0	24.7	130/90	80	6.0	200	1.7	0.50
Algeria	2010	M	60-69	185.0	95.0	27.0	140/100	90	7.0	220	1.9	0.60
Algeria	2010	F	60-69	175.0	75.0	24.5	135/95	85	6.5	210	1.8	0.55
Algeria	2010	M	70-79	188.0	100.0	26.6	145/105	95	7.5	230	2.0	0.65
Algeria	2010	F	70-79	178.0	78.0	23.6	140/100	90	7.0	220	1.9	0.60
Algeria	2010	M	80-89	190.0	105.0	25.8	150/110	100	8.0	240	2.1	0.70
Algeria	2010	F	80-89	180.0	80.0	22.2	145/105	95	7.5	230	2.0	0.65
Algeria	2010	M	90-99	192.0	110.0	23.2	155/115	105	8.5	250	2.2	0.75
Algeria	2010	F	90-99	182.0	82.0	20.4	150/110	100	8.0	240	2.1	0.70
Algeria	2010	M	100-109	195.0	115.0	23.6	160/120	110	9.0	260	2.3	0.80
Algeria	2010	F	100-109	185.0	85.0	20.8	155/115	105	8.5	250	2.2	0.75
Algeria	2010	M	110-119	198.0	120.0	23.2	165/125	115	9.5	270	2.4	0.85
Algeria	2010	F	110-119	188.0	88.0	20.8	160/120	110	9.0	260	2.3	0.80
Algeria	2010	M	120-129	200.0	125.0	23.2	170/130	120	10.0	280	2.5	0.90
Algeria	2010	F	120-129	190.0	90.0	20.8	165/125	115	9.5	270	2.4	0.85
Algeria	2010	M	130-139	202.0	130.0	23.2	175/135	125	10.5	290	2.6	0.95
Algeria	2010	F	130-139	192.0	92.0	20.8	170/130	120	10.0	280	2.5	0.90
Algeria	2010	M	140-149	205.0	135.0	23.2	180/140	130	11.0	300	2.7	1.00
Algeria	2010	F	140-149	195.0	95.0	20.8	175/135	125	10.5	290	2.6	0.95
Algeria	2010	M	150-159	208.0	140.0	23.2	185/145	135	11.5	310	2.8	1.05
Algeria	2010	F	150-159	198.0	98.0	20.8	180/140	130	11.0	300	2.7	1.00
Algeria	2010	M	160-169	210.0	145.0	23.2	190/150	140	12.0	320	2.9	

Figura 4: Factores de corrección de vehículos pesados(Ministerial, 2023)
Fuente: Imagen obtenida de la Ficha del MTC

El incremento del tránsito vehicular puede estimarse mediante una fórmula de progresión geométrica, aplicada de forma independiente tanto al flujo de vehículos de pasajeros como al de vehículos de carga. La tasa de crecimiento anual del tránsito está vinculada al comportamiento del desarrollo socioeconómico. En general, el crecimiento del tránsito de pasajeros se relaciona con el aumento anual de la población, mientras que el de vehículos de carga se asocia con la expansión económica, medida a través del Producto Bruto Interno (PBI). Usualmente, las tasas de crecimiento del tránsito se sitúan entre el 2 % y el 6 %.

Asimismo, el cuadro que se presenta a continuación establece los lineamientos para seleccionar el Factor de Crecimiento Acumulado (Fca.) a lo largo del horizonte de diseño, en función de la tasa anual de crecimiento (r) y la duración del período de análisis en años (Ministerio de Transportes y Comunicaciones, 2014).

Cuadro 6.2
Factores de Crecimiento Acumulado (Fca)
Para el Cálculo de Número de Repeticiones de EE

Período de Análisis (años)	Factor sin Crecimiento	Tasa anual de crecimiento (r)							
		2	3	4	5	6	7	8	10
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	2.00	2.02	2.03	2.04	2.05	2.06	2.07	2.08	2.10
3	3.00	3.06	3.09	3.12	3.15	3.18	3.21	3.25	3.31
4	4.00	4.12	4.18	4.25	4.31	4.37	4.44	4.51	4.64
5	5.00	5.20	5.19	5.42	5.53	5.64	5.75	5.87	6.11
6	6.00	6.31	6.47	6.63	6.80	6.98	7.15	7.34	7.72
7	7.00	7.43	7.66	7.90	8.14	8.39	8.55	8.92	9.49
8	8.00	8.56	8.89	9.21	9.55	9.90	10.26	10.64	11.44
9	9.00	9.75	10.16	10.58	11.03	11.49	11.98	12.49	13.58
10	10.00	10.95	11.46	12.01	12.58	13.18	13.82	14.49	15.94
11	11.00	12.17	12.81	13.49	14.21	14.97	15.78	16.65	18.53
12	12.00	13.41	14.19	15.03	15.92	16.87	17.89	18.98	21.38
13	13.00	14.68	15.62	16.63	17.71	18.88	20.14	21.50	24.52
14	14.00	15.97	17.09	18.29	19.16	21.01	22.55	24.21	27.97
15	15.00	17.29	18.60	20.02	21.58	23.28	25.13	27.15	31.77
16	16.00	18.64	20.16	21.82	23.66	25.67	27.89	30.32	35.95
17	17.00	20.01	21.76	23.70	25.84	28.21	30.84	33.75	40.55
18	18.00	21.41	23.41	25.65	28.13	30.91	34.00	37.45	45.60
19	19.00	22.84	25.12	27.67	30.54	33.76	37.38	41.45	51.16
20	20.00	24.30	26.87	29.78	33.06	36.79	41.00	45.76	57.28

Figura 5: Factores de Crecimiento Acumulado (Fca) (AASHTO, 1993)
Fuente: Imagen obtenida del Manual del MTC

$$T_n = T_0(1 + r)^n - 1$$

Donde:

T_n : Tráfico proyectado al año " n ".^{en} Veh/día.

TO: Tráfico actual (año base) en Veh/día (IMDA).

n: Año futuro de proyección.

r. Tasa anual de crecimiento del tráfico.

Se efectuarán proyecciones de tráfico para cada tipo de vehículo, considerando la tasa anual de crecimiento vehicular calculada y debidamente sustentada, según corresponda o se utilizará la tasa de crecimiento de la población y PBI para el tráfico ligero y pesado respectivamente. Para ello se debe recurrir a proyecciones

(Ministerial, 2023) propone unos valores propios para proyectos del Sector, tal como se muestra en la hoja de cálculo Excel: A.4. Tasa de Crecimiento del aplicativo de la FICHA TÉCNICA ESTÁNDAR.

Tasas para la Proyección de la Demanda		FORMATO A5.TC	
Tasa de Crecimiento Anual de Vehículos Ligeros		Tasa de Crecimiento Anual de Vehículos Pesados	
Amazonas.	1.12%	Amazonas.	2.69%
Ancash.	2.33%	Ancash.	1.49%
Apurímac.	1.13%	Apurímac.	4.50%
Arequipa.	2.90%	Arequipa.	2.97%
Ayacucho.	1.83%	Ayacucho.	2.90%
Cajamarca.	3.05%	Cajamarca.	1.45%
Cusco.	2.77%	Cusco.	3.07%
Huancavelica.	1.17%	Huancavelica.	2.00%
Huánuco.	1.94%	Huánuco.	3.03%
Ica.	1.10%	Ica.	2.62%
Junín.	2.04%	Junín.	2.84%
La Libertad.	2.61%	La Libertad.	2.21%
Lambayeque.	2.40%	Lambayeque.	2.54%
Lima Provincias.	2.61%	Lima Provincias.	2.34%
Loreto.	0.12%	Loreto.	1.48%
Madre de Dios.	2.22%	Madre de Dios.	1.38%
Moquegua.	1.97%	Moquegua.	0.58%
Pasco.	1.72%	Pasco.	0.39%
Piura.	2.38%	Piura.	2.37%
Puno.	2.47%	Puno.	2.58%
San Martín.	2.11%	San Martín.	2.88%
Tacna.	2.13%	Tacna.	2.36%
Tumbes.	0.81%	Tumbes.	2.40%
Ucayali.	1.73%	Ucayali.	2.25%

Información al 2022, sin contar las distorsiones estadísticas provocadas por el impacto económico de la COVID 19
 Nota: Los valores presentados, son susceptibles a ser actualizados periódicamente por la OPMI-MTC, sin incurrir en actualización de la Ficha Técnica Estándar.

Figura 6: Tasas para la Proyección de la Demanda (Ministerial, 2023)

Fuente: Imagen obtenida de la Ficha del MTC

2.2.5. Número De Repeticiones De Ejes Equivalentes

El manual establece que, en el diseño de pavimentos, debe priorizarse la evaluación del tránsito pesado, es decir, camiones y ómnibus. Las cargas generadas por estos vehículos se expresan en términos de Ejes Equivalentes (EE) acumulados a lo largo del período de diseño. Según la definición de AASHTO, un EE representa el efecto dañino que produce sobre el pavimento un eje simple con doble rueda, sometido a una carga de 8.2 toneladas y

una presión de inflado de 80 psi. Estos ejes equivalentes cuantifican el impacto estructural que el tránsito pesado genera sobre la superficie del pavimento. La Figura 7 muestra la configuración típica de estos ejes (Ministerio de Transportes y Comunicaciones, 2014).

Figura 6.1 Configuración de Ejes			
Conjunto de Eje (s)	Nomenclatura	Nº de Neumáticos	Grafico
EJE SIMPLE (Con Rueda Simple)	1RS	02	
EJE SIMPLE (Con Rueda Doble)	1RD	04	
EJE TANDEM (1 Eje Rueda Simple + 1 Eje Rueda Doble)	1RS + 1RD	06	
EJE TANDEM (2 Ejes Rueda Doble)	2RD	08	
EJE TRIDEM (1 Rueda Simple + 2 Ejes Rueda Doble)	1RS + 2RD	10	
EJE TRIDEM (3 Ejes Rueda Doble)	3RD	12	

Figura 7: Configuración de Ejes (Ministerio de Transportes y Comunicaciones, 2014)
Fuente: Imagen obtenida del Manual del MTC

2.2.5.1 Factor Direccional Y Factor Carril (FD Y FC)

el factor direccional es la cantidad de transporte pesado que circula en la dirección de un carril, casi siempre corresponden a la mitad del tránsito que circula en las dos direcciones de la vía. El manual indica que el factor carril vendría a ser el carril con el mayor número de ejes equivalente, donde la transitabilidad se ubica casi siempre por ese carril. El tráfico para el carril de diseño de la vía tiene en consideración el número de direcciones y la cantidad de carriles por calzada, según el factor ponderado aplicado al IMD.(Gordillo Vilchez & Rabanal Rengifo, 2020)

Corresponde al número de vehículos pesados que circulan en una dirección o sentido de tráfico, normalmente corresponde a la mitad del total de tránsito circulante en ambas direcciones, pero en algunos casos puede ser mayor en una dirección que en otra, el que se definirá según el conteo de tráfico El factor de distribución carril expresado como

una relación, que corresponde al carril que recibe el mayor número de EE (Ministerio de Transportes y Comunicaciones, 2014)

Cuadro 6.1
Factores de Distribución Direccional y de Carril para determinar el Tránsito en el Carril de Diseño

Número de calzadas	Número de sentidos	Número de carriles por sentido	Factor Direccional (Fd)	Factor Carril (Fc)	Factor Ponderado Fd x Fc para carril de diseño
1 calzada (para IMDa total de la calzada)	1 sentido	1	1.00	1.00	1.00
	1 sentido	2	1.00	0.80	0.80
	1 sentido	3	1.00	0.60	0.60
	1 sentido	4	1.00	0.50	0.50
	2 sentidos	1	0.50	1.00	0.50
	2 sentidos	2	0.50	0.80	0.40
2 calzadas con separador central (para IMDa total de las dos calzadas)	2 sentidos	1	0.50	1.00	0.50
	2 sentidos	2	0.50	0.80	0.40
	2 sentidos	3	0.50	0.60	0.30
	2 sentidos	4	0.50	0.50	0.25

Figura 8: Factores de Distribución Direccional y de Carril (AASHTO, 1993)

Fuente: Imagen obtenida del Manual del MTC

2.2.5.2 El Factor Vehículo Pesado (FVP)

El Factor Vehículo Pesado (Fvp) se define como el valor promedio de ejes equivalentes correspondiente a un tipo específico de vehículo pesado, ya sea camión o bus. Este promedio se calcula dividiendo el total de ejes equivalentes (EE) registrados para un tipo particular de vehículo entre la cantidad de unidades de ese mismo tipo. Para su determinación, es indispensable utilizar los datos reales de carga por eje obtenidos del censo de transporte de carga, y asegurarse de que estos estén alineados con las especificaciones del Reglamento Nacional de Vehículos vigente (Ministerio de Transportes y Comunicaciones, 2014).

En el proceso de diseño de pavimentos, se considera la cantidad de EE que circularán por la vía durante el horizonte de análisis. El diseño se enfoca en el carril que soportará el mayor volumen de carga, y el resultado obtenido se aplicará a los carriles correspondientes a la sección tipo seleccionada del camino. Cuando no se dispone de datos provenientes de estaciones de pesaje, la demanda de tránsito se estima mediante muestreos del flujo vehicular. Estos conteos se enfocan en los vehículos pesados con el objetivo de obtener promedios representativos, registrando la carga real por eje, el tipo de vehículo y la configuración de sus ejes.

De este modo, es posible calcular el Fvp para cada tipo de unidad vehicular, promediando los EE que generan los vehículos pesados en la vía analizada. La normativa establece que este factor se debe calcular con base en los pesos reales por eje de los vehículos pesados relevados en el censo de cargas (Gordillo Vilchez & Rabanal Rengifo, 2020).

Cuadro 6.3
Relación de Cargas por Eje para determinar Ejes Equivalentes (EE)
Para Afirmados, Pavimentos Flexibles y Semirrígidos

Tipo de Eje	Eje Equivalente (EE _{8.2 ton})
Eje Simple de ruedas simples (EE _{S1})	$EE_{S1} = [P / 6.6]^{1.0}$
Eje Simple de ruedas dobles (EE _{S2})	$EE_{S2} = [P / 8.2]^{1.0}$
Eje Tandem (1 eje ruedas dobles + 1 eje rueda simple) (EE _{T41})	$EE_{T41} = [P / 14.8]^{1.0}$
Eje Tandem (2 ejes de ruedas dobles) (EE _{T42})	$EE_{T42} = [P / 15.1]^{1.0}$
Ejes Tridem (2 ejes ruedas dobles + 1 eje rueda simple) (EE _{T41})	$EE_{T41} = [P / 20.7]^{1.0}$
Ejes Tridem (3 ejes de ruedas dobles) (EE _{T42})	$EE_{T42} = [P / 21.8]^{1.0}$

P = peso real por eje en toneladas

Figura 9: Relación de Cargas por Eje Para Afirmados, Pavimentos Flexibles y Semirrígidos (AASHTO, 1993)

Fuente: Imagen obtenida del Manual del MTC

Cuadro 6.4
Relación de Cargas por Eje para determinar Ejes Equivalentes (EE)
Para Pavimentos Rígidos

Tipo de Eje	Eje Equivalente (EE _{8.2 ton})
Eje Simple de ruedas simples (EE _{S1})	$EE_{S1} = [P / 6.6]^{4.1}$
Eje Simple de ruedas dobles (EE _{S2})	$EE_{S2} = [P / 8.2]^{4.1}$
Eje Tandem (1 eje ruedas dobles + 1 eje rueda simple) (EE _{T41})	$EE_{T41} = [P / 13.0]^{4.1}$
Eje Tandem (2 ejes de ruedas dobles) (EE _{T42})	$EE_{T42} = [P / 13.3]^{4.1}$
Ejes Tridem (2 ejes ruedas dobles + 1 eje rueda simple) (EE _{T41})	$EE_{T41} = [P / 16.6]^{4.1}$
Ejes Tridem (3 ejes de ruedas dobles) (EE _{T42})	$EE_{T42} = [P / 17.5]^{4.1}$

P = peso real por eje en toneladas

Figura 10: Relación de Cargas por Eje Para Pavimentos Rígidos (AASHTO, 1993)

Fuente: Imagen obtenida del Manual del MTC

2.2.5.3 Factor de ajuste por Presión de Neumático (Fp)

La norma nos indica que el Ingeniero que diseña los pavimentos flexibles y semirrígidos debe tener en cuenta, para calcular EE, un factor por presión de neumático, esto señala un efecto adicional al factor destructivo sobre el pavimento. Para afirmados y pavimentos rígidos el factor es igual a 1.0. Para la determinación de los factores de presión de neumáticos se utilizarán los valores del Cuadro 6.13, valores intermedios podrán interpolarse (Ministerio de Transportes y Comunicaciones, 2014)

Cuadro 6.13
FACTOR DE AJUSTE POR PRESIÓN DE NEUMÁTICO (F_p) PARA EJES EQUIVALENTES (EE)

Espesor de Capa de Rodadura (mm)	Presión de Contacto del Neumático (PCN) en psc PCN = 0.90x[Presión de inflado del neumático] (pai)						
	80	90	100	110	120	130	140
50	1.00	1.30	1.80	2.13	2.91	3.59	4.37
60	1.00	1.33	1.72	2.18	2.69	3.27	3.92
70	1.00	1.30	1.65	2.05	2.49	2.99	3.53
80	1.00	1.28	1.59	1.94	2.32	2.74	3.20
90	1.00	1.25	1.53	1.84	2.17	2.52	2.91
100	1.00	1.23	1.48	1.75	2.04	2.35	2.68
110	1.00	1.21	1.43	1.66	1.91	2.17	2.44
120	1.00	1.19	1.38	1.59	1.80	2.02	2.25
130	1.00	1.17	1.34	1.52	1.70	1.89	2.09
140	1.00	1.15	1.30	1.46	1.62	1.78	1.94
150	1.00	1.13	1.26	1.39	1.52	1.66	1.79
160	1.00	1.12	1.24	1.36	1.47	1.59	1.71
170	1.00	1.11	1.21	1.31	1.41	1.51	1.61
180	1.00	1.09	1.18	1.27	1.36	1.45	1.53
190	1.00	1.08	1.16	1.24	1.31	1.39	1.45
200	1.00	1.08	1.15	1.22	1.28	1.35	1.41

Nota:

- EE = Ejes Equivalentes
- Presión de inflado del neumático (Pai): está referido al promedio de presiones de inflado de neumáticos por tipo de vehículo pesado.
- Presión de Contacto del neumático (PCN): igual al 90% del promedio de presiones de inflado de neumáticos por tipo de vehículos pesado.
- Para espesores menores de capa de rodadura asfáltica, se aplicará el factor de ajuste igual al espesor de 50 mm.

Fuente: Elaboración propia, en base a correlaciones con la Figura IV 4 EAL Adjustment Factor for Tire Pressures del Manual MS 2 del Instituto de Vialidad.

Figura 11: Factor de ajuste por presión de neumático (AASHTO, 1993)
Fuente: Imagen obtenida del Manual del MTC

Finalmente, la norma nos proporciona una fórmula donde calcularemos el número de repeticiones de EE de 8.2tn, en el tiempo de diseño. El resultado final será la suma de los diferentes tipos de transporte pesado considerado:

$$N_{repdeEE8,2ton} = \sum [EE_{dia-carril} \times F_{cax365}]$$

$$EE_{dia-carril} = IMD_{pix} F_{dx} F_{cx} F_{Vpix} F_{pi}$$

Parámetros	Descripción
Nrep de EE 8.2t	Número de Repeticiones de Ejes Equivalentes de 8.2 ton
EE _{8.2t-caril}	EE_{8.2t-caril} = Ejes Equivalentes por cada tipo de vehículo pesado, por día para el carril de diseño. Resulta del IMD por cada tipo de vehículo pesado, por el Factor Direccional, por el Factor Carril de diseño, por el Factor Vehículo Pesado de tipo seleccionado y por el Factor de Presión de neumáticos. Para cada tipo de vehículo pesado, se aplica la siguiente relación: $EE_{8.2t-caril} = IMD_p \times F_d \times F_c \times F_{vp} \times F_p$ donde IMD_p: corresponde al Índice Medio Diario según tipo de vehículo pesado seleccionado (i) F_d: Factor Direccional, según Cuadro N° 6.1. F_c: Factor Carril de diseño, según Cuadro N° 6.1. F_{vp}: Factor vehículo pesado de tipo seleccionado (i) calculado según su composición de ejes. Representa el número de ejes equivalentes promedio por tipo de vehículo pesado (bus o camión), y el promedio se obtiene dividiendo el total de ejes equivalentes (EE) de un determinado tipo de vehículo pesado entre el número total del tipo de vehículo pesado seleccionado. F_p: Factor de Presión de neumáticos, según Cuadro N° 6.13.
Fca	Factor de crecimiento acumulado por tipo de vehículo pesado (según cuadro 6.2)
365	Número de días del año
Σ	Sumatoria de Ejes Equivalentes de todos los tipos de vehículo pesado, por día para el carril de diseño por Factor de crecimiento acumulado por 365 días del año.

Figura 12: Factor de ajuste por presión de neumático (AASHTO, 1993)
Fuente: Imagen obtenida del Manual del MTC

2.2.6. Redes neuronales artificiales

Una red neuronal es un modelo de computación cuya estructura de capas se asemeja a la estructura interconectada de las neuronas en el cerebro, con capas de nodos conectados cuya función es aprender mediante el entrenamiento y procesamiento de datos de tal manera que puedan reconocer patrones de voz, imágenes, aproximar funciones, predecir eventos, entre otros, tal como lo afirman (Kaviani & Sohn, 2020)

Está compuesta de unidades procesadoras interconectadas unas con otras, capaz de reconocer patrones, clasificar datos y pronosticar eventos futuros con precisión y exactitud. En la Figura 13 se puede ver la arquitectura de una red neuronal con una capa de entrada, dos capas ocultas y una capa de salida. (Dreyfus, 2005)

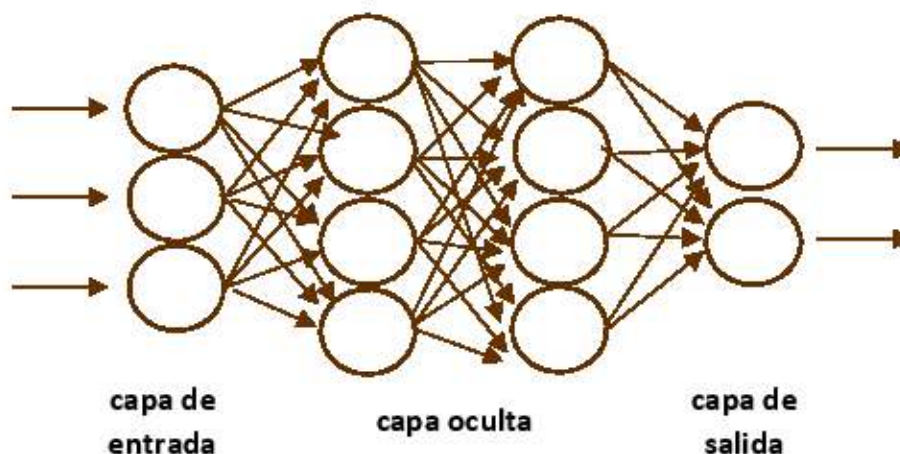


Figura 13: Arquitectura típica de una red neuronal
Fuente: Imagen obtenida de(Incio-Flores et al., 2023)

Desde una perspectiva más detallada, la estructura de una red neuronal se compone de cuatro elementos fundamentales. Poznyak et al. (2001) señalan que su configuración interna está definida por los pesos sinápticos, los cuales representan las conexiones entre neuronas y pueden ser de naturaleza excitadora o inhibidora. Además, se incluye un sumador encargado de calcular la suma ponderada de las entradas multiplicadas por sus respectivos pesos. A esto se añade una función de activación no lineal, que introduce complejidad en el modelo, y un umbral externo que determina si la salida de la neurona se activa en función del resultado procesado.

2.2.7. Aprendizaje automático o machine learning

El aprendizaje automático, o machine learning, es una rama de la inteligencia artificial que se basa en el uso de modelos estadísticos y algoritmos para que los sistemas computacionales puedan aprender a partir de datos, sin necesidad de una programación explícita. A través de este enfoque, las máquinas adquieren la capacidad de identificar patrones y generar predicciones utilizando los datos de entrada disponibles (Müller & Guido, 2016).

Existen tres categorías principales dentro del aprendizaje automático. Entre ellas, el aprendizaje supervisado y el no supervisado se distinguen principalmente por la forma en que se utilizan los datos: el primero requiere conjuntos de datos etiquetados para entrenar modelos, mientras que el segundo se enfoca en descubrir estructuras o agrupaciones dentro de datos que no han sido previamente clasificados (Maqbool et al., 2018).

2.2.7.1 Aprendizaje supervisado

En los métodos de aprendizaje supervisado, se desarrolla un modelo capaz de realizar predicciones a partir de una relación entre datos de entrada y sus respectivas salidas. El término supervisado hace referencia al uso de un conjunto de datos que ya ha sido previamente clasificado, es decir, se cuenta con muestras que incluyen información sobre la categoría, clase o valor al que pertenecen.

Este conjunto, conocido como datos de entrenamiento, se utiliza para ajustar progresivamente el modelo inicial. El algoritmo aprende a realizar clasificaciones comparando las salidas generadas por el modelo con las etiquetas reales de los datos, y ajustando sus parámetros para minimizar los errores cometidos durante el proceso de predicción (Gonzalez, 2018).

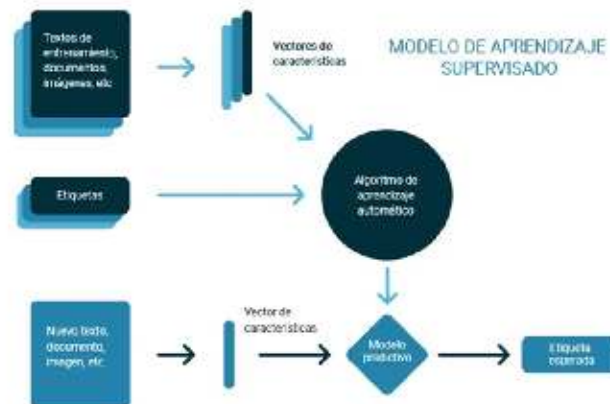


Figura 14: Diagrama de flujo del aprendizaje supervisado
Fuente: Imagen obtenida de (Gonzalez, 2018)

2.2.7.2 Aprendizaje no supervisado

El aprendizaje no supervisado es una técnica dentro del campo del machine learning que permite a los modelos analizar datos sin necesidad de conocer previamente sus categorías o resultados esperados. A diferencia de otros enfoques que requieren datos etiquetados, aquí el sistema trabaja únicamente con la información en bruto y busca, por sí solo, identificar relaciones, patrones o estructuras ocultas. Su principal utilidad radica en agrupar datos similares o reducir su complejidad manteniendo los aspectos más relevantes. Esta metodología es especialmente útil cuando se manejan grandes volúmenes de datos, como los recolectados por aplicaciones móviles o plataformas digitales, ya que permite segmentar usuarios o comportamientos sin intervención humana. Las dos aplicaciones más comunes del aprendizaje no supervisado son el clustering, que agrupa elementos según sus características compartidas, y la reducción de dimensionalidad, que simplifica los datos sin perder información clave (Ramos Castro et al., 2023).

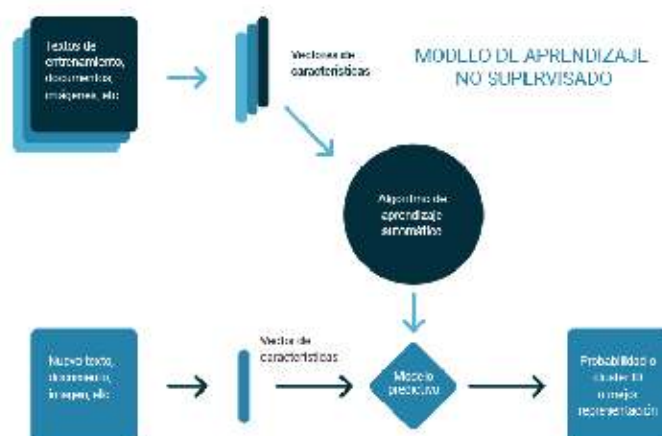


Figura 15: Diagrama de flujo del aprendizaje no supervisado
Fuente: Imagen obtenida de (Gonzalez, 2018)

2.2.7.3 Aprendizaje por refuerzo

El aprendizaje por refuerzo es un enfoque del machine learning que se basa en la experiencia directa del modelo al interactuar con un entorno cambiante. En lugar de recibir datos previamente etiquetados, el sistema aprende mediante señales de retroalimentación, como recompensas o penalizaciones, que obtiene tras ejecutar cada acción. El objetivo principal de esta técnica es que el modelo desarrolle una estrategia que le permita maximizar la recompensa total a lo largo del tiempo. A medida que experimenta distintas situaciones dentro del entorno, el modelo ajusta su comportamiento para mejorar sus decisiones futuras. Este tipo de aprendizaje se aplica en escenarios complejos donde es necesario tomar decisiones secuenciales, como en juegos de tablero (por ejemplo, ajedrez o Go), en el control de vehículos autónomos o en plataformas que personalizan recomendaciones (Ramos Castro et al., 2023).



Figura 16: Diagrama de flujo aprendizaje por refuerzo
Fuente: Imagen obtenida de (Gonzalez, 2018)

2.2.8. Aprendizaje profundo o deep learning

El deep learning o aprendizaje profundo es una rama del aprendizaje automático que se fundamenta en el uso de redes neuronales con múltiples capas. Estas redes profundas son capaces de aprender directamente desde datos sin procesar, extrayendo características cada vez más abstractas y complejas a medida que la información atraviesa cada capa. Gracias a esta estructura jerárquica, el aprendizaje profundo puede abordar tareas sofisticadas como el reconocimiento de imágenes, la identificación de voz o el procesamiento de lenguaje natural.

A diferencia de otros enfoques del aprendizaje automático, el deep learning se distingue por tres aspectos fundamentales: la utilización de varias capas de neuronas artificiales, la capacidad para construir representaciones jerárquicas de los datos, y el uso del algoritmo de retropropagación, que ajusta automáticamente los pesos de la red durante el proceso de entrenamiento (Goodfellow, 2016).

2.2.8.1 Modelo de una Red neuronal artificial

Para entender el funcionamiento de una neurona artificial, es fundamental primero conocer la estructura de las neuronas biológicas, las cuales se conectan entre sí de forma paralela dentro del sistema nervioso. Estas células especializadas se encargan de regular la actividad eléctrica en el cerebro, ya sea activándola o inhibiéndola. Su estructura está compuesta por tres partes principales: las dendritas, que actúan como receptores de señales (entrada); el axón, que transmite la señal (salida); y el soma, que procesa la información recibida (Van Essen & Glasser, 2018).

De acuerdo con Elman (1991), una neurona artificial puede definirse como un elemento que, a partir de un conjunto de datos de entrada, es capaz de generar una salida. Este tipo de neurona está formada por múltiples entradas ($X_1, X_2, \dots, X_n$), un conjunto de pesos sinápticos (W_m), una función de propagación (U_k) que calcula la suma ponderada mediante el producto escalar entre las entradas y los pesos, así como una función de activación y una función de salida, las cuales se seleccionan según la naturaleza del problema a resolver.

En la Figura 17 se representa el modelo no lineal de una neurona artificial: las señales de entrada son procesadas a través de la combinación ponderada de los datos y sus respectivos pesos sinápticos, y luego son transformadas por una función de activación para producir la salida del sistema.

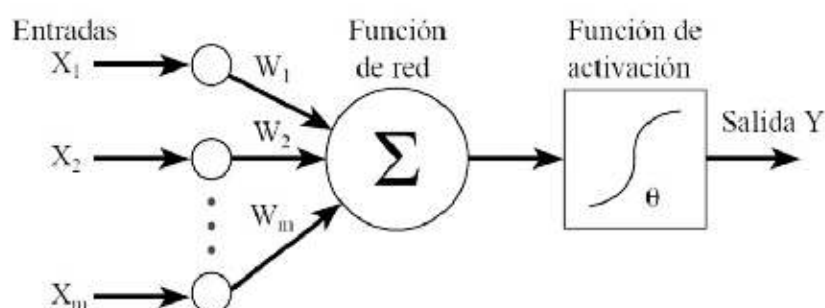


Figura 17: Elementos de una red neuronal artificial.
Fuente: Imagen obtenida de (Loría-García et al., 2017)

2.2.9. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN, por sus siglas en inglés) pertenecen al campo del deep learning o aprendizaje profundo. Estas redes cuentan con múltiples capas ocultas y emulan, en cierta medida, el funcionamiento de las conexiones neuronales del cerebro humano. A medida que se incrementa el número de capas en la arquitectura, la red se vuelve más profunda, lo que permite un mayor número de neuronas y, en consecuencia, una mejor capacidad para procesar y predecir en conjuntos de datos complejos (Salehi & Burgueño, 2018).

Según Cortes Zarta et al. (2021), este tipo de red neuronal artificial está diseñada para identificar patrones, características o similitudes abstractas en los datos de entrada, especialmente en imágenes, lo que la vincula estrechamente con aplicaciones en

visión por computadora. Las CNN están compuestas por un número limitado de capas convolucionales, las cuales se encargan de extraer diferentes niveles de características a partir de las imágenes. Las primeras capas suelen detectar rasgos básicos como bordes, líneas o colores, mientras que las capas más profundas capturan estructuras más complejas, como formas, siluetas o texturas. Por ejemplo, en el caso de identificar un vehículo, las capas iniciales reconocerán elementos simples y las capas finales podrán discernir el contorno o el tipo de vehículo.

La figura 18 ilustra una arquitectura típica de una CNN aplicada a una tarea de reconocimiento de imágenes. En este tipo de red, una imagen ingresa al sistema y pasa por una serie de capas hasta que se obtiene una clasificación final. Estas arquitecturas usualmente incluyen tres tipos de capas: convolucionales, de agrupamiento (pooling) y totalmente conectadas (fully connected), cada una operando con una función de activación ReLU.

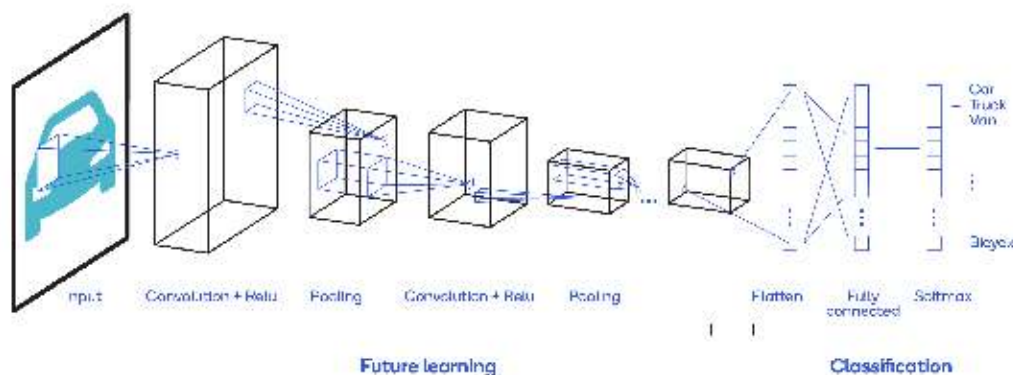


Figura 18: Esquema de arquitectura típica de una red neuronal convolucional
Fuente: Obtenido de Condori (2019)

Las CNN en una etapa inicial del proceso alternan entre las capas convolucionales y las pooling, y para la etapa final trabajan con las capas completamente conectadas con un clasificador no lineal, que toma una decisión de clasificación basándose en los valores de las características de las capas anteriores. (ReLU) Salehi y Burgueño, 2018

2.2.9.1 Función de una red neuronal convolucional

Las redes neuronales convolucionales (CNN) están diseñadas para trabajar con imágenes tanto bidimensionales como tridimensionales y se estructuran comúnmente en tres tipos de capas: la capa de convolución, la capa de agrupamiento o reducción (max pooling), y las capas densas (fully connected) LeCun et al., 2015. Las dos primeras se encargan del proceso de extracción de características, mientras que las capas finales realizan la clasificación del contenido procesado.

En la etapa inicial, la red identifica patrones básicos presentes en la imagen, como bordes, texturas o formas, de manera similar a cómo el cerebro humano reconoce atributos

distintivos en objetos. Esta fase actúa como un sistema de filtrado jerárquico que extrae información progresivamente más abstracta. Posteriormente, en las capas densas, toda esta información se integra para tomar una decisión final: clasificar la imagen o realizar una predicción. Por ejemplo, al entrenar una CNN para detectar perros, las primeras capas pueden reconocer características como el pelaje, el contorno o el color del animal. Luego, con base en esas características, las capas densas determinan si la imagen corresponde efectivamente a un perro.

Una de las principales fortalezas de las CNN radica en su capacidad para establecer relaciones espaciales entre los píxeles, lo que les permite comprender la imagen en su totalidad, manteniendo coherencia en la estructura visual y mejorando así la precisión en tareas de reconocimiento Badillo et al., 2021.

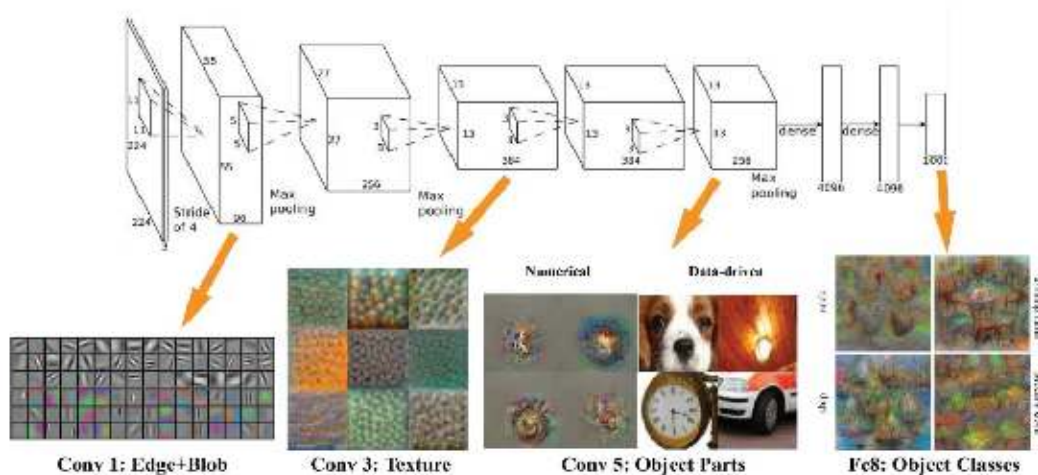


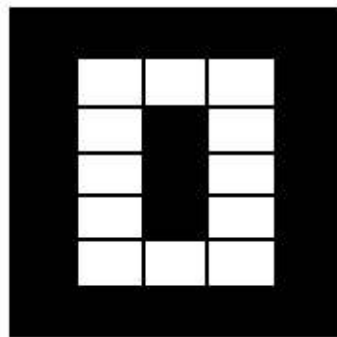
Figura 19: Funcionamiento de una CNN

Fuente: Obtenido de Condori (2019)

2.2.9.2 Conceptos claves de las redes neuronales

- **Pixel:** Del inglés picture element, “elemento de imagen”, es un único punto o un pequeño cuadro que contiene color, puede ser blanco, negro o matices de gris en el caso de las imágenes diagnósticas, y forma parte de una imagen digital LeCun et al., 2015
- **Kernel:** Conocido como feature detector, filter o “detector de propiedades”, es una matriz de números que se utiliza para enfocar, desenfocar, grabar, detectar bordes y características de la imagen LeCun et al., 2015
- **Preprocesamiento:** Una imagen digital es una matriz almacenada en el computador donde, para el caso de imagen en escala de grises (imágenes radiológicas), cada uno de los píxeles está representado con un valor que va de 0 (tonalidades oscuras) hasta 255 (tonalidades claras o brillantes). La normalización de los datos hace referencia a la división del valor del pixel en 255. Esta transformación asigna valores de píxeles entre 0 y 1, lo que permite disminuir la carga computacional y generar una mejor red neuronal con mayor agilidad en el aprendizaje, que se verá reflejado de manera

positiva en las predicciones de las imágenes (figura 20) Russakovsky et al., 2015



una imagen para los ojos
humanos

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

...es una matriz de píxeles para
los computadores. El valor de los
píxeles va de 0 a 255 pero se
normaliza para la red neuronal
de 0 a 1.

Figura 20: Preprocesamiento.
Fuente: Imagen obtenida de Badillo et al. (2021)

2.2.10. Funcionamiento de la CNN

2.2.10.1 Capas de convolución

La operación de convolución en una red neuronal convolucional (CNN) consiste en aplicar un filtro o kernel sobre una imagen con el propósito de extraer características específicas, como bordes, texturas u otros patrones relevantes Badillo et al. (2021). Este proceso se basa en la superposición de una pequeña matriz (el filtro) sobre la imagen original, realizando cálculos que permiten destacar ciertas regiones según la función del filtro utilizado.

La idea principal es que este kernel, de menor tamaño que la imagen, se desplace sobre ella en un patrón sistemático. En cada posición, se toma una porción de la imagen del mismo tamaño que el filtro y se realiza una multiplicación elemento a elemento entre ambos. Luego, se suman los resultados para generar un nuevo valor que conforma el mapa de características o feature map. Este valor representa un nuevo píxel en la imagen de salida, que resalta la información relevante detectada en esa sección.

Por ejemplo, al utilizar un filtro orientado verticalmente de tamaño 3x3 sobre una imagen de 6x6 píxeles, la convolución comienza en la esquina superior izquierda. Se ejecuta la multiplicación y suma mencionadas, y luego el filtro se desplaza una posición hacia la derecha para repetir el mismo proceso. Cuando se alcanza el borde derecho de la imagen, el filtro baja una fila y vuelve a desplazarse horizontalmente desde el inicio. Esta operación se repite hasta cubrir toda la imagen, produciendo una nueva representación que conserva únicamente las características más significativas. Este mapa de salida será utilizado posteriormente en la capa de pooling para continuar con el proceso de análisis.

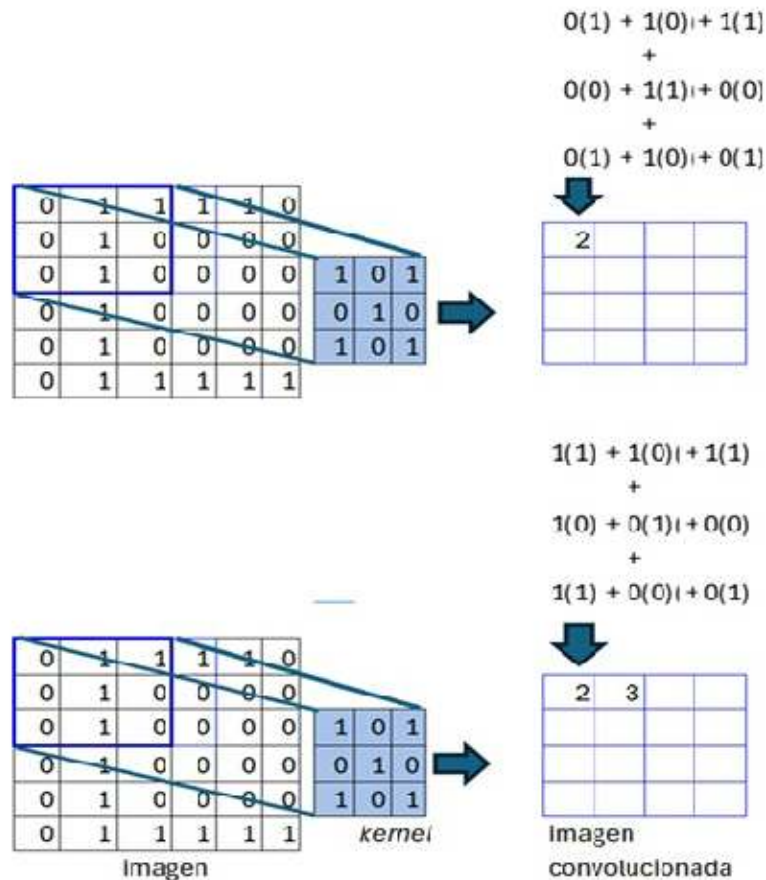


Figura 21: Función del kernel.
Fuente: Imagen obtenida de Badillo et al. (2021)

La imagen resultante será más pequeña con características que priman respecto a la imagen de entrada. En este caso, se partió con una imagen de 6x6 y se obtuvo una imagen de 4x4 del lado derecho; se repite la misma operación hasta que el kernel haga una barrido de toda la imagen.

2.2.10.2 Capa max pooling o capa de reducción

La capa de reducción, también conocida como pooling, es la segunda etapa dentro de una red neuronal convolucional (CNN) y tiene como función principal disminuir la cantidad de datos que se generan tras la etapa de convolución, conservando únicamente la información más significativa o las características más destacadas de la imagen procesada Graham (2014).

Este proceso contribuye directamente a reducir la carga computacional de la red, facilitando un entrenamiento más eficiente y enfocado, al tiempo que ayuda a filtrar el ruido presente en las imágenes. Sin embargo, su aplicación debe hacerse con criterio, ya que un uso excesivo por ejemplo, mediante max pooling con una alta tasa de reducción puede conducir a la pérdida de detalles importantes que podrían ser relevantes para la clasificación o segmentación Lee et al. (2016).

En términos prácticos, el objetivo de esta capa es disminuir el tamaño espacial de la

imagen resultante sin perder las activaciones clave. El resultado es una representación más compacta y limpia desde el punto de vista computacional, pero aún rica en contenido esencial. Por ejemplo, si se parte de una imagen de 4×4 píxeles y se aplica una operación de reducción adecuada, se puede obtener una imagen condensada de 2×2 píxeles que conserva las características más representativas.

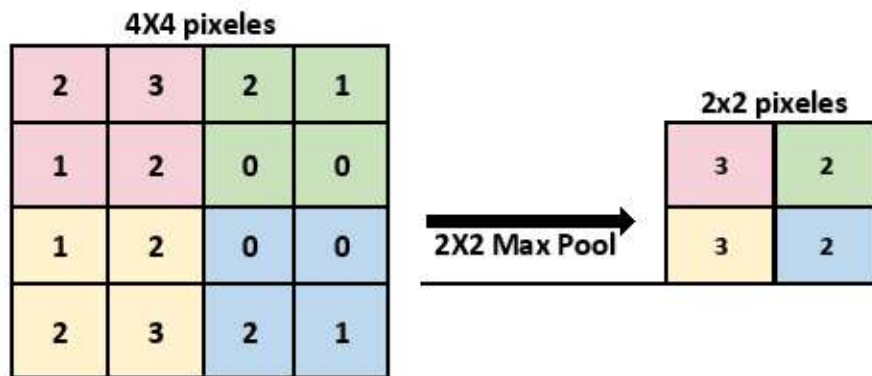


Figura 22: Capa max pooling
Fuente: Imagen obtenida de Badillo et al. (2021)

Figura 22. Ejemplo de imagen resultante tras aplicar una operación de pooling (reducción) con un filtro de tamaño 2×2. Este tipo de operación, cuyo tamaño puede variar según el diseño de la red neuronal, recorre la imagen convolucionada en este caso de 4×4 píxeles desplazándose de izquierda a derecha y de arriba hacia abajo, en bloques de 2×2 píxeles (es decir, áreas de 4 píxeles). Para cada bloque, se selecciona el valor más alto, lo que da lugar al nombre Max Pooling. El resultado de este proceso es una imagen de menor tamaño, reducida a 2×2 píxeles, que mantiene únicamente las activaciones más representativas.

Esta reducción no solo optimiza el almacenamiento y procesamiento de los datos dentro de la red, sino que también disminuye la cantidad de neuronas necesarias en las capas siguientes, facilitando el aprendizaje de las características más relevantes para la tarea de detección o clasificación. Badillo et al. (2021).

2.2.10.3 Capas densas o capas de clasificación (red neuronal multicapa o fully connected)

Las capas completamente conectadas (fully connected) representan la etapa final de una red neuronal convolucional (CNN) y se encargan de asignar una puntuación a cada clase posible, utilizando las características extraídas en las etapas previas de convolución y pooling. En esta fase, se integran todos los datos aprendidos a lo largo del entrenamiento para producir una predicción final Yasaka et al. (2018).

No existe un número fijo de capas fully connected que deba emplearse, aunque la literatura especializada suele referirse al uso de entre dos y cuatro capas. Ejemplos de arquitecturas reconocidas que siguen esta estructura son LeNet LeCun et al. (1998), VGG Net Huang et al. (2017) y AlexNet, una de las redes más influyentes en el campo Krizhevsky et al. (2017).

Cada neurona en estas capas incorpora una función de activación no lineal, cuya selección depende del tipo de problema que la red está resolviendo (clasificación, regresión, etc.). No obstante, debido al alto costo computacional que pueden generar estas capas, han surgido alternativas más eficientes en años recientes. Entre ellas se destacan la global average pooling y la average pooling, técnicas que permiten reducir significativamente la complejidad del modelo sin comprometer su rendimiento Alom et al. (2019).

2.2.10.4 Función de activación

Las funciones de activación desempeñan un papel esencial dentro de las redes neuronales convolucionales (CNN), ya que están presentes tanto en las capas de convolución como en las capas completamente conectadas (fully connected). Estas funciones forman parte de la arquitectura interna de cada neurona artificial, cuya estructura es comparable a la representada esquemáticamente. Gracias a la función de activación, la neurona puede procesar la información recibida y generar una respuesta adecuada, facilitando la clasificación de los datos recopilados (Alom et al., 2019).

Existen diversos tipos de funciones de activación, cada una con propósitos específicos. Una de las más clásicas es la función sigmoideal, frecuentemente utilizada en problemas de clasificación binaria. Esta función produce valores que oscilan entre 0 y 1, interpretándose como una probabilidad: un valor cercano a 1 indica una alta probabilidad de que la entrada pertenezca a una clase positiva, mientras que un valor próximo a 0 sugiere lo contrario (Graves & Graves, 2012). Por ejemplo, si una red está entrenada para reconocer perros y recibe una imagen con esta categoría, puede devolver un resultado de 0.9, indicando alta probabilidad de que efectivamente se trate de un perro. En cambio, si se ingresa una imagen de un gato, el valor podría ser 0.2, lo cual refleja baja probabilidad de pertenencia a la clase "perro".

Cuando se aborda un problema de clasificación multiclase es decir, con más de dos categorías posibles se utiliza comúnmente la función softmax. Esta es una extensión de la función sigmoideal que convierte las salidas del modelo en un conjunto de probabilidades normalizadas cuya suma es igual a uno (Yamashita et al., 2018). Por ejemplo, si una red neuronal ha sido entrenada para reconocer razas de perros como labrador, terrier, bulldog y pinscher, y se ingresa una imagen de un terrier, la red podría generar una salida como: terrier 0.9, labrador 0.05, bulldog 0.03 y pinscher 0.02. En este caso, la clase con la probabilidad más alta sería seleccionada como la predicción final (Badillo et al., 2021).

Por otra parte, una de las funciones de activación más populares en redes profundas es la ReLU (Rectified Linear Unit), conocida por su eficiencia al permitir entrenar redes más complejas y profundas. ReLU se caracteriza por anular los valores negativos y mantener los positivos, lo que ayuda a mitigar el problema del desvanecimiento del gradiente una situación donde la red deja de aprender por errores mínimos en el ajuste de pesos (Krizhevsky et al., 2017).

Entre las variantes mejoradas de ReLU se encuentra la leaky ReLU, la cual introduce una

pequeña pendiente en la región negativa. Esto evita que todas las salidas negativas se conviertan en cero, permitiendo que la red continúe ajustándose y aprendiendo, aunque en menor proporción. Gracias a esta propiedad, se logra mantener activo el proceso de entrenamiento y se favorece una mayor capacidad de aprendizaje continuo de la CNN (Li et al., 2018).

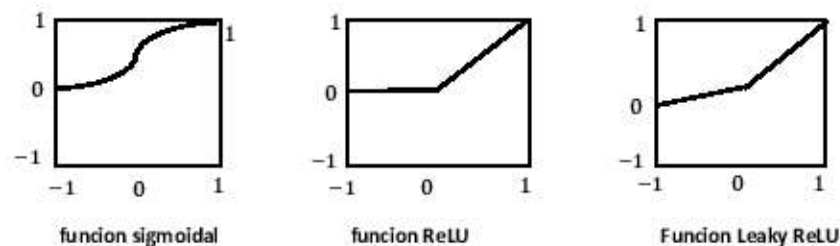


Figura 23: Representación de la función de activación
Fuente: Imagen obtenida de Badillo et al. (2021)

2.2.11. Entrenamiento de una CNN

El entrenamiento de una red neuronal convolucional (CNN) consiste en encontrar los valores óptimos tanto para los kernels en las capas de convolución como para los pesos en las capas completamente conectadas, de manera que se reduzca al mínimo la diferencia entre las salidas generadas por la red y las respuestas correctas, conocidas como etiquetas de verdad. Estas etiquetas corresponden a los diagnósticos reales asociados a cada imagen utilizada durante el entrenamiento (Yamashita et al., 2018).

Este proceso de aprendizaje se lleva a cabo de forma iterativa. En una primera etapa, denominada forward propagation, los datos ingresan a la red y atraviesan cada una de sus capas, aplicando operaciones matemáticas que culminan en una predicción de salida. Esta salida se compara con la etiqueta de verdad para calcular el error cometido. Luego, en la etapa llamada backpropagation, dicho error se utiliza para ajustar los parámetros internos de la red específicamente, los valores de los filtros y los pesos sinápticos. Este ajuste se basa en el cálculo de gradientes, que indican cómo deben modificarse los valores en cada capa para reducir el error en futuras iteraciones.

Así, si la red comete un error al hacer una predicción, la retropropagación corrige internamente sus parámetros para mejorar el rendimiento en los siguientes ciclos de entrenamiento. Este mecanismo de retroalimentación continua es el que permite que una CNN aprenda y mejore su precisión progresivamente.

2.2.12. Algoritmo de entrenamiento de la Red Neuronal

El algoritmo de entrenamiento empleado para la RNA backpropagation se detalla a continuación.

1. Inicializar los pesos de la red (w) con valores aleatorios pequeños.
2. Mientras la condición de paro sea falsa realizar los pasos (3-6).

3. Se presenta un patrón de entrada, $(x_{p1}, x_{p2}, \dots, x_{pi})$ y se especifica la salida deseada que debe generar la red $(d_{p1}, d_{p2}, \dots, d_{pi})$.
4. Se calcula la salida actual de la red, para ello se presentan las entradas a la red y se va calculando la salida que presenta cada capa hasta llegar a la capa de salida $(y_1, y_2, \dots, y_k)$. Los pasos son los siguientes:
 - a) Se determinan las entradas netas para las neuronas ocultas procedentes de las neuronas de entrada.

$$N_{pj}^h = \sum_{i=1}^m w_{ji}^h x_{pi} + b_i^h \quad (2)$$

- b) Aplicar la función de activación para cada entrada de la neurona y así obtener su salida.

$$y_{pj} = f_j^h \left(N_{pj}^h = \sum_{i=1}^m w_{ji}^h x_{pi} + b_i^h \right) \quad (3)$$

- c) Realizaremos cálculos para obtener las salidas de las neuronas ingresadas.

$$N_{pk}^o = \sum_{j=1}^m w_{jk}^o y_{pj} + b_k^o \quad (4)$$

$$Y_{pk} = f_j^o \left(N_{pk}^o = \sum_{j=1}^m w_{jk}^o y_{pj} + b_k^o \right) \quad (5)$$

5. Determinación de los términos de error para todas las neuronas:
 - a) Calculo del error (salida deseada - salida obtenida).

$$e = (d_{pk} - y_{pk}) \quad (6)$$

- b) Obtención de la delta (producto del error con la derivada de la función de activación con respecto a los pesos de la red).

$$\delta_{pk}^o = e * f_k^o \left(N_{pk}^o \right) \quad (7)$$

6. Para actualizar los pesos. Debemos emplear el siguiente algoritmo recursivo denominado gradiente descendente, empezando por la salida y propagando hacia atrás para llegar a la capa de entrada. a) Para los pesos de las neuronas de la capa de salida:

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \Delta w_{kj}^o(t+1) \quad (8)$$

$$\Delta w_{kj}^o(t+1) = \text{minu } \delta_{pk}^o y_{pj} \quad (9)$$

- b) Para los pesos de las neuronas de la capa oculta:

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \Delta w_{ji}^h(t+1) \quad (10)$$

$$\Delta w_{ji}^h(t+1) = \min \delta_{pj}^h y_{pi} \quad (11)$$

7. El error mínimo o número de iteraciones alcanzado logrado nos indica la condición de parar.

2.2.13. Detección de objetos

La detección de objetos es una técnica perteneciente al campo de la visión por computadora y el procesamiento digital de imágenes, cuyo objetivo es localizar e identificar instancias de objetos pertenecientes a clases específicas como personas, vehículos o edificaciones dentro de imágenes o secuencias de video. Cada categoría de objeto posee atributos distintivos que facilitan su reconocimiento. Por ejemplo, en la detección de círculos, se consideran aquellas formas cuyos puntos se encuentran a una distancia constante de un centro; en el caso de los cuadrados, se identifican figuras con lados de igual longitud y esquinas en ángulo recto.

Este principio también se aplica en tareas como el reconocimiento facial, donde se detectan elementos como los ojos, la nariz y los labios, así como otras características relevantes como la tonalidad de la piel o la distancia entre los ojos, las cuales contribuyen a la identificación del rostro (Antúñez Galindo, 2019).

Existen varias herramientas que se utilizan para la detección de objetos en imágenes y vídeos. Algunas de las más utilizados son:

- **You Only Look Once (YOLO):** es un algoritmo de detección de objetos en tiempo real que utiliza redes neuronales convolucionales para identificar y localizar objetos en una imagen o un video. YOLO fue desarrollado por Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhadi en 2016. El algoritmo YOLO es capaz de detectar varios objetos en una sola imagen o cuadro de video y proporciona la localización precisa de cada objeto en términos de su caja delimitadora y una probabilidad de que el objeto detectado sea de una determinada clase. YOLO utiliza una única red neuronal para realizar tanto la detección como la clasificación de objetos, lo que lo hace más rápido que otros enfoques de detección de objetos que requieren múltiples etapas de procesamiento
- **Single Shot Detector (SSD):** es un algoritmo de detección de objetos que utiliza una única red neuronal para detectar objetos en una imagen y realiza una única “pasada” a través de la imagen. Al igual que YOLO, SSD es capaz de realizar la detección de objetos en tiempo real y proporciona alta precisión.
- **Faster R-CNN:** es otro algoritmo de detección de objetos que utiliza una red neuronal convolucional y una técnica de regiones propuestas de objetos. Aunque es más lento que YOLO y SSD, Faster R-CNN proporciona una mayor precisión en la detección de objetos.
- **RetinaNet:** es otro algoritmo de detección de objetos que utiliza una red neuronal convolucional y una técnica de regiones propuestas de objetos. Aunque es más lento que YOLO y SSD, Faster R-CNN proporciona una mayor precisión en la detección de

objetos.

- **Mask R-CNN:** es una extensión del algoritmo Faster R-CNN que también proporciona información sobre la segmentación de objetos. Además de detectar objetos, Mask R CNN es capaz de generar máscaras precisas para cada objeto detectado
- **Tensorflow Object Detection API:** es una biblioteca de aprendizaje automático de código abierto que ofrece recursos para el reconocimiento de objetos en tiempo real y el seguimiento de objetos en video. Esta API se basa en Tensorflow, una biblioteca de aprendizaje automático de Google, y utiliza modelos de detección de objetos pre entrenados para detectar objetos en imágenes y vídeos.

2.2.14. YOLOv8

YOLOv8 es una versión desarrollada de manera abierta por la empresa Ultralytics, la cual no forma parte del desarrollo oficial de YOLO pero se encuentra ampliamente documentada en su repositorio en GitHub. Esta implementación ha sido diseñada con un enfoque en la facilidad de uso, la precisión y el rendimiento, lo que la convierte en una herramienta versátil para tareas como detección y seguimiento de objetos, segmentación por instancia, clasificación de imágenes y estimación de poses (Ramos Castro et al., 2023).

El uso de YOLOv8 es bastante accesible, ya que se puede integrar como una librería de Python, similar a cómo se utiliza NumPy u otras bibliotecas populares. Desde esta librería, es posible acceder a clases y funciones específicas orientadas a la detección de objetos, permitiendo una implementación rápida y eficiente dentro de cualquier entorno de desarrollo.

Ultralytics también ofrece una variedad de modelos ya entrenados, listos para ser utilizados en diferentes escenarios. Cada uno de estos modelos presenta un equilibrio distinto entre precisión y velocidad de procesamiento, por lo que se recomienda seleccionar el que mejor se adapte a las necesidades del proyecto. En este trabajo, se ha optado por YOLOv8 precisamente por su alto rendimiento, precisión destacable y facilidad de integración, lo que facilita su uso en aplicaciones prácticas.

2.2.15. Seguimiento de objetos

El seguimiento de objetos es una técnica clave dentro del campo de la visión por computadora que permite detectar y rastrear la ubicación de un objeto que se encuentra en movimiento a lo largo del tiempo, dentro de una secuencia de imágenes o video.

El propósito principal de esta técnica es asegurar la identificación continua del mismo objeto a lo largo de varios fotogramas, incluso cuando este sufre transformaciones en su forma, tamaño, posición o apariencia. Una forma común de representar este seguimiento es mediante el uso de puntos o vectores que indiquen la trayectoria del movimiento del objeto en el transcurso de los cuadros del video (Paredes, 2011).

Para llevar a cabo este tipo de análisis, existen múltiples herramientas y bibliotecas especializadas en procesamiento de video y seguimiento visual. Algunas de las más utilizadas son:

- OpenCV: es una librería de visión artificial de código de uso libre que proporciona una amplia variedad de recursos para el procesamiento de imágenes y el seguimiento de objetos.
- DeepSORT: es un sistema de seguimiento de objetos basado en aprendizaje profundo que utiliza un enfoque basado en detección y seguimiento. Es uno de los algoritmos más precisos en el campo del seguimiento de objetos y ha sido utilizado en diversas aplicaciones de seguimiento de objetos en tiempo real.
- Simple Online and Realtime Tracking (SORT): es un algoritmo de seguimiento de objetos en tiempo real que utiliza una técnica basada en asociación de datos para realizar un seguimiento preciso y confiable de objetos en secuencias de video. Es muy rápido y fácil de implementar y ha sido utilizado en diversas aplicaciones de seguimiento de objetos.
- Kalman Filter: es una técnica de seguimiento de objetos que utiliza un modelo de predicción basado en el movimiento del objeto y un modelo de observación basado en las mediciones de los sensores. Este filtro se compone de dos etapas: la etapa de predicción y la etapa de corrección.
- Kalman Filter: el filtro de Kalman se utiliza para estimar el estado de un sistema dinámico a partir de mediciones incompletas y ruidosas, utilizando un modelo del sistema y un modelo del ruido utilizando consigo matrices y vectores para representar el estado y las mediciones del sistema.
- Minimum Output Sum of Squared Error (MOSSE): es un método de seguimiento de objetos basado en el análisis de correlación que utiliza una red de filtrado adaptativo para realizar el seguimiento de objetos en tiempo real.
- Kernelized Correlation Filter (KCF): es un enfoque de seguimiento de objetos basado en el análisis de correlación que utiliza características de imagen y filtros de correlación para realizar el seguimiento de objetos con alta precisión

Estas representan solo algunas de las soluciones disponibles para implementar el seguimiento de objetos en secuencias de video. La selección de la herramienta más adecuada dependerá directamente de los requerimientos particulares del proyecto, como el nivel de precisión necesario, la velocidad de procesamiento y la complejidad del entorno de aplicación (Ramos Castro et al., 2023).

3. MARCO METODOLÓGICO

3.1. Enfoque

Esta investigación adopta un enfoque **cuantitativo**, fundamentado en la medición de variables numéricas relacionadas con el tráfico vehicular y el diseño de pavimentos, incluyendo el conteo vehicular automatizado por categorías, el Índice Medio Diario Anual (IMDA), y el número de repeticiones de Ejes Equivalentes (Nrep EE). Asimismo, se emplean diversos factores numéricos establecidos por las normativas del MTC, tales como el Factor de Corrección Estacional (FCE), Factor Direccional (Fd), Factor Carril (Fc), Factor de Vehículo Pesado (Fvp), Factor de Presión (Fp), determinación del EE día-carril (Eje Equivalente por día y por carril), y el Factor de Crecimiento Acumulado (Fca), los cuales permiten la conversión de datos observacionales en parámetros de diseño técnicamente válidos. Además, la validación del modelo se realiza mediante métricas de rendimiento **cuantitativas** como precisión (precision), sensibilidad (recall) y F1-Score, garantizando la reproducibilidad, objetividad y validez estadística de los resultados.

3.2. Alcance

Esta investigación presenta un alcance **explicativo** ya que no solo describe y cuantifica el comportamiento del tráfico vehicular en la Av. Ramón Castilla, sino que establece las relaciones causales entre las variables observadas y los parámetros de diseño de pavimentos. En la investigación se detalla cómo la implementación del modelo permite la detección automatizada de 22 clases de vehículos y cómo estos datos se transforman, mediante factores de corrección específicos, en índices técnicos como el IMDA y los Ejes Equivalentes de Carga (ESAL). Además, la investigación explica las relaciones funcionales entre las cargas vehiculares detectadas y su impacto en el diseño estructural del pavimento, estableciendo conexiones directas entre el comportamiento del tráfico observado y los requerimientos técnicos de la infraestructura vial. Este enfoque **explicativo** permite comprender no solo qué está ocurriendo en términos de flujo vehicular, sino por qué estos datos son determinantes para el dimensionamiento adecuado del pavimento, proporcionando una base científica para la toma de decisiones en el diseño de infraestructura vial según las normativas del MTC.

3.3. Diseño de investigación

Esta investigación de acuerdo al diseño es **no experimental** porque no se manipulan deliberadamente las variables, sino que se observan y registran los fenómenos del tráfico vehicular en su contexto natural sin intervención del investigador. El estudio se limita a la observación sistemática del comportamiento vehicular en la Av. Ramón Castilla tal como ocurre espontáneamente, sin alterar las condiciones de tráfico, modificar el flujo vehicular o implementar cambios en la infraestructura vial que pudieran influir en los resultados. La recolección de datos se realiza mediante el sistema de detección YOLO que registra el paso de vehículos, sus características y frecuencias, sin que el investigador intervenga

en el proceso natural del tráfico. Este enfoque no experimental permite obtener datos representativos de las condiciones reales de operación de la vía, garantizando que los parámetros calculados como el IMDA y los ESAL reflejen fielmente el comportamiento actual del tráfico vehicular, proporcionando una base sólida y realista para el diseño de pavimentos según las condiciones operativas existentes en la zona de estudio.

3.4. Población y muestra

3.4.1. Población

Esta investigación define como población al conjunto total de **7,000 imágenes** de vehículos capturados, los cuales constituyen el universo completo de datos visuales disponibles para el análisis del tráfico vehicular. Esta población de imágenes representa la totalidad de registros fotográficos obtenidos durante el período de monitoreo establecido, abarcando diferentes tipos de vehículos, horarios de circulación, condiciones climáticas y patrones de tráfico característicos. La utilización de esta población completa de 7,000 imágenes garantiza la representatividad estadística del estudio y proporciona un volumen de datos suficiente para obtener resultados confiables y precisos en la caracterización del comportamiento vehicular, cumpliendo con los requisitos metodológicos necesarios para estudios de ingeniería de tráfico y diseño de pavimentos.

3.4.2. Muestra

Se define una muestra de **3,587** imágenes de vehículos seleccionadas de la población total de 7,000 imágenes capturadas, representando el 51.24 % del universo total de datos disponibles. Esta muestra fue específicamente destinada para el entrenamiento del modelo, constituyendo el conjunto de datos de aprendizaje que permite al modelo reconocer y clasificar las 22 categorías de vehículos establecidas en el estudio. La muestra fue obtenida mediante un muestreo no probabilístico intencional, donde la selección de las imágenes se realizó considerando criterios técnicos específicos como la calidad de la imagen, diversidad de tipos vehiculares, variabilidad en condiciones de iluminación, ángulos de captura y representatividad de los diferentes horarios de tráfico. Este tipo de muestreo intencional garantiza que las imágenes seleccionadas para el entrenamiento contengan las características visuales más representativas y relevantes para el aprendizaje del modelo, optimizando la capacidad de detección y clasificación vehicular.

3.4.2.1 Método de selección de la muestra

Se utilizará un método de selección no probabilístico, específicamente un muestreo intencional. Las imágenes serán seleccionadas con el objetivo de asegurar una representación adecuada de los diferentes tipos de vehículos.

3.4.2.2 Tamaño de la Muestra

El tamaño de la muestra será de **3,587** imágenes. Este tamaño es adecuado para asegurar que el modelo de RNA tenga suficientes datos para aprender y generalizar las características necesarias para el conteo y la detección de velocidades de vehículos.

3.4.2.3 Unidad de muestreo

La unidad de muestreo son los puntos de captura de imágenes en diferentes avenidas. Los puntos de muestreo serán seleccionados estratégicamente para capturar una variedad de tipos de vehículos.

3.4.2.4 Validación de los datos de la muestra

Para garantizar la efectividad del entrenamiento del modelo de RNA, se utilizará la técnica de validación cruzada. Este método implica dividir los datos de la muestra en múltiples subconjuntos. Un porcentaje del conjunto de datos se utilizará para entrenar el modelo (por ejemplo, el 70 %), mientras que el restante (30 %) se utilizará para validar el modelo. Este proceso se repite varias veces (K-Fold Cross-Validation) para asegurar que el modelo generalice bien y no esté sobreajustado a un conjunto específico de datos. P

3.4.3. Unidad de análisis

La unidad de análisis son las imágenes de diferentes tipos de vehículos capturadas en los puntos de muestreo seleccionados. Estas imágenes se utilizarán tanto para entrenar como para validar el modelo de RNA. Cada imagen proporcionará información crucial sobre el tipo de vehículo y sus características, necesarias para el conteo y la detección de velocidades.

3.5. Hipótesis

3.5.1. Hipótesis general

El desarrollo de un modelo basado en Redes Neuronales Artificiales (RNA) mejora y optimiza la precisión y eficiencia en la detección y conteo de vehículos para la determinación de parámetros críticos en el diseño de pavimentos, como el Índice Medio Diario Anual (IMDA), Número de repeticiones de Ejes Equivalentes (ESAL), según las normativas del Ministerio de Transportes y Comunicaciones (MTC).

3.5.2. Hipótesis específicas

1. El diseño del modelo basado en Redes Neuronales Artificiales (RNA) permite ajustar una arquitectura para la detección, clasificación y conteo eficiente de vehículos basado en el formato de clasificación vehicular del MTC.
2. El entrenamiento del modelo basado en Redes Neuronales Artificiales (RNA) permite lograr un balance óptimo entre la exactitud en la detección y la cobertura en el reconocimiento de vehículos.
3. La implementación del modelo permite desarrollar un software con interfaz gráfica de usuario (GUI) para la obtención automática del IMDA y ESAL en cualquier red vial, optimizando los procesos de cálculo en la planificación y diseño estructural de pavimentos.

3.6. Operacionalización de variables

3.6.1. Variables

- **Variable independiente (X): Modelo basado en Redes Neuronales Artificiales (RNA) :**
Un sistema de aprendizaje automático que utiliza arquitecturas de redes neuronales para procesar datos y realizar tareas de análisis complejas, como la detección y clasificación de vehículos en un entorno de tráfico vehicular.
- **Variable dependiente (Y): Análisis del tráfico vehicular :** La evaluación y estudio del flujo de vehículos en una vía, incluyendo la cantidad de vehículos, sus velocidades y los índices medios de tráfico, para mejorar la planificación y el diseño de la infraestructura vial.

3.6.2. Dimensiones

Al referirnos a las variables sabemos que estos no son observables directamente. Por lo tanto se deben buscar procedimientos que permitan la medición indirecta mediante manifestaciones externas, empíricas y observables. Tales expresiones son los indicadores.

- **Variable independiente (X): Modelo basado en Redes Neuronales Artificiales (RNA)**
 - Dimensión 1: Arquitectura del modelo, estructura y organización de las capas y conexiones en la RNA utilizada para el análisis del tráfico vehicular.
 - Dimensión 2: Parámetros de entrenamiento, Valores ajustables durante el proceso de aprendizaje del modelo RNA.
 - Dimensión 3: Métodos de entrenamiento, Técnicas y algoritmos utilizados para optimizar el modelo RNA, incluyendo los algoritmos de optimización y técnicas de regularización.
 - Dimensión 4: Resultados del entrenamiento, Medidas de rendimiento del modelo después del entrenamiento.
- **Variable dependiente (Y): Análisis del tráfico vehicular**
 - Dimensión 1: Cantidad de vehículos, Número total de vehículos que pasan por una vía en un período específico.
 - Dimensión 2: Índices medios de tráfico, Cálculos de los promedios diarios y anuales del volumen de tráfico en una vía.
 - Dimensión 3: Implementación del modelo RNA, Aplicación práctica del modelo RNA en diferentes redes viales urbanas.

3.6.3. Indicadores

3.6.3.1 Indicadores independientes:

1. Arquitectura del modelo
 - Número de capas de la red neuronal, [Número entero (número de capas)]
 - Tipo de capas, [Categoría (tipos de capas)]
 - Tamaño de las capas, [Número entero (número de neuronas o filtros por capa)]
 - Función de activación, [Categoría (tipos de funciones)]

2. Parámetros de entrenamiento

- Tasa de aprendizaje, [Valor decimal (learning rate)]
- Número de épocas de entrenamiento, [Número entero (número de épocas)]
- Momento (momentum), [Valor decimal (momentum)]
- Factor de decaimiento del aprendizaje (learning rate decay), [Valor decimal (decay rate)]

3. Métodos de entrenamiento

- Algoritmo de optimización, [Categoría (SGD, Adam, RMSprop)]
- Técnicas de regularización, [Categoría (dropout, L2 regularization, early stopping)]
- Estrategias de aumento de datos, [Categoría (rotación, escalado, traslación)]
- Esquema de aprendizaje, [Categoría (step decay, exponential decay, adaptive)]

4. Resultados del entrenamiento

- Precisión en la validación y entrenamiento [Porcentaje (%)]
- Pérdida en la validación y entrenamiento [Valor numérico (cross-entropy loss, mean squared error)]
- Tiempo total de entrenamiento [minutos (min)]
- Curva de aprendizaje [Gráfico (eje X: épocas, eje Y: precisión/pérdida)]
- Pesos del modelo Matriz de valores numéricos [(weights matrix)]
- Bias del modelo Vector de valores numéricos [(bias vector)]
- Sobreajuste (overfitting) [Porcentaje (%)]

3.6.3.2 Indicadores dependientes:

1. Cantidad de vehículos

- Número de vehículos por tipo, [Vehículos por tipo (veh/tipo)]

2. Índices medios de tráfico

- Índice medio diario de tráfico (IMD) [(veh/día)] y Índice medio anual (IMA) [(veh/año)]

3. Implementación del modelo RNA

- Desarrollo del algoritmo, [Número entero (etapas completadas)]
- Compatibilidad del algoritmo, [Número de sistemas compatibles]

En los anexos se presenta la operacionalización de las variables identificadas, donde se detalla las variables, indicadores, dimensiones y sus unidades de medida.

3.7. Técnicas e instrumentos.**3.7.1. Técnicas de investigación**

Para esta investigación, se emplearán diversas técnicas para la recopilación y análisis de datos, así como para el desarrollo y validación del modelo de redes neuronales convolucionales utilizando YOLO. A continuación, se detallan las técnicas utilizadas:

1. **Revisión bibliográfica y documental:** Consiste en la recopilación y análisis de literatura existente sobre la evaluación de pavimentos flexibles, técnicas de redes neuronales convolucionales y métodos tradicionales como el PCI. Este proceso se realizará a través de bases de datos científicas como IEEE Xplore, Google Scholar, Scopus, libros especializados y documentos técnicos.
2. **Análisis exploratorio de datos (EDA):** Realización de un análisis preliminar de los datos recopilados sobre el deterioro de pavimentos para entender mejor su distribución y características. Se utilizarán de programación Python y sus respectivas librerías (Pandas, NumPy, Matplotlib, Seaborn).
3. **Desarrollo y entrenamiento del modelo de RNA** Creación y entrenamiento de un modelo de redes neuronales convolucionales utilizando YOLO para detectar, clasificar tipos de vehículos. Se emplearán frameworks de deep learning como TensorFlow y Keras, así como hardware especializado como GPUs para acelerar el entrenamiento.
4. **Validación del modelo:** Evaluación del rendimiento del modelo de RNA utilizando un conjunto de datos de validación no visto durante el entrenamiento. Se emplearán métricas de rendimiento como precisión, recall, F1-score y matrices de confusión.
5. **Implementación y pruebas de campo:** Implementación del modelo de RNA en la Av. Ramon Castilla para evaluar su rendimiento en condiciones prácticas. Se utilizarán dispositivos de captura de imágenes como cámaras de alta resolución.

3.7.2. Instrumentos de investigación

Para la implementación de las técnicas mencionadas, se utilizarán diversos instrumentos detallados a continuación:

- **Creación de bases de datos a partir de la recolección de imágenes de vehículos:**
 - **Recolección de datos en campo:** Captura de imágenes de vehículos consignados en el manual de carreteras. Se utilizarán dispositivos móviles, y cámaras de alta resolución para recolectar una amplia variedad de datos.
- **Software de análisis de datos:** Herramientas y bibliotecas de programación para el análisis exploratorio y la preparación de datos. Ejemplos incluyen Python (Pandas, NumPy, Matplotlib, Seaborn).
- **Frameworks de deep learning:** Entornos de desarrollo para la creación, entrenamiento y evaluación de modelos de redes neuronales (Ultralytics YOLO).
- **Hardware especializado:** Equipos con alta capacidad de procesamiento, como GPUs (NVIDIA) y TPUs (Google), para acelerar el entrenamiento de modelos de deep learning.

3.7.3. Herramientas de software utilizadas

En esta investigación se hará uso de diversas herramientas de software, detalladas a continuación:

Cuadro 1: Herramientas de software utilizadas.

Herramienta	Descripción
LaTeX	Preparación de documentos para la redacción del informe final
Anaconda	Distribución de Python para el análisis de datos y desarrollo de modelos
Ultralytics YOLO	Algoritmo de detección de objetos utilizado en el modelo CNN
TensorFlow & Keras	Frameworks de deep learning para el desarrollo y entrenamiento de modelos CNN
Python	Bibliotecas para el análisis de datos y visualización
SciPy & StatsModels	Bibliotecas de Python para análisis estadístico

Fuente: Elaboración propia.

3.8. Desarrollo del trabajo de tesis

3.8.1. Recopilación de datos

Para la recopilación del conjunto de datos, se utilizó un dispositivo móvil como herramienta de captura fotográfica de vehículos en circulación. La determinación del número de categorías se basó en el sistema de clasificación vehicular oficial establecido por el Ministerio de Transportes y Comunicaciones (MTC). El dataset resultante comprende 3,587 fotografías distribuidas entre las 22 clases de vehículos tipificadas en la normativa del MTC.

La adquisición de imágenes se realizó bajo condiciones variables de iluminación, abarcando cinco franjas horarias: Mañana, media mañana, tarde, media tarde y noche. Esta estrategia de muestreo diversificado se realizó con el objetivo de desarrollar un modelo capaz de mantener una precisión de detección vehicular constante durante las 24 horas del día, adaptándose a las variaciones lumínicas naturales desde condiciones de alta luminosidad diurna hasta condiciones de baja luminosidad nocturna.



Figura 24: Captura de fotografías-horario madrugada 05:30 am
Fuente: Imagen propia



Figura 25: Captura de fotografías-horario media mañana 10:30 am
Fuente: Imagen propia



Figura 26: Captura de fotografías-horario media mañana 13:30 pm
Fuente: Imagen propia



Figura 27: Captura de fotografías-horario tarde 15:30 pm
Fuente: Imagen propia



Figura 28: Captura de fotografías-horario noche 21:30 pm
Fuente: Imagen propia

Se definieron 22 categorías vehiculares conforme a la normativa oficial del Ministerio de Transportes y Comunicaciones (MTC), siguiendo el formato de clasificación vehicular establecido en la ficha técnica estandar del MTC. Estas siglas de las 22 clases son las que el modelo aprenderá durante el proceso de entrenamiento, por lo que es fundamental que tengan la denominación adecuada y sean consistentes en todo el dataset.

A continuación, se detallan las categorías implementadas junto con las siglas correspondientes utilizadas para el proceso de etiquetado del dataset:

Cuadro 2: Definición de categorías (clases)

Categoría (clase)	Clasificación de vehículos MTC
3T3	Trayler C3R3 (Camión con 3 ejes, remolque con 3 ejes)
3T2	Trayler C3R2 (Camión con 3 ejes, remolque con 2 ejes)
2T3	Trayler C2R3 (Camión con 2 ejes, remolque con 3 ejes)
2T2	Trayler C2R2 (Camión con 2 ejes, remolque con 2 ejes)
3S3	Semitrayler T3S3 (Tracto camión con 3 ejes, Semiremolque con 3)
3S2	Semitrayler T3S2 (Tracto camión con 3 ejes, Semiremolque con 2)
3S1	Semitrayler T3S1 (Tracto camión con 3 ejes, Semiremolque con 1 eje)
2S3	Semitrayler T2S3 (Tracto camión con 2 ejes, Semiremolque con 3)
2S2	Semitrayler T2S2 (Tracto camión con 2 ejes, Semiremolque con 2)
2S1	Semitrayler T2S1 (Tracto camión con 2 ejes, Semiremolque con 1 eje)
C4E	Camión C4 (Camión rígido de 4 ejes con estructura especial)
C3E	Camión C3 (Camión rígido de 3 ejes con estructura especial)
C2E	Camión C2 (Camión rígido de 2 ejes con estructura especial)
B4E	Bus B4-1 (Bus interprovincial de transporte de pasajeros con 4 ejes)
B3E	Bus B3-1 (Bus interprovincial de transporte de pasajeros con 3 ejes)
B2E	Bus B2 (Bus interprovincial de transporte de pasajeros con 2 ejes)
M	Micro (Micro-bus de transporte urbano)
RC	Rural-Combi
PU	Camioneta-Pick Up
PA	Camioneta-Panel
SW	Station Wagon
A	Auto

Fuente: Elaboración propia

3.8.2. Conjunto de datos entrenamiento

El dataset local comprende 3,587 imágenes en formato JPG con resolución estándar, configuración compatible con el software LabelMe . Se utilizó LabelMe para el etiquetado de las instancias vehiculares. La selección del formato JPG se fundamenta en su amplia compatibilidad con herramientas de anotación, su eficiencia en el almacenamiento de datos y su capacidad para preservar la calidad visual necesaria para la identificación precisa de características vehiculares. El proceso de etiquetado mediante LabelMe permite la creación de archivos de anotación en formato JSON, los cuales contienen las coordenadas de los bounding boxes y las etiquetas de clase correspondientes a cada vehículo detectado en las imágenes, generando así los datos de entrenamiento supervisado requeridos para el desarrollo del modelo de detección

3.8.3. Etiquetado en LabelMe con EfficientSAM (accuary)

El etiquetado de las 3587 imágenes se llevó a cabo utilizando EfficientSAM (accuary) debido a su notable precisión y su eficiencia computacional, factores clave que aseguraron la calidad y viabilidad del modelo. Esta elección estratégica se justifica por la capacidad de EfficientSAM para generar máscaras de segmentación de alta calidad, así mismo; su robustez ante la variabilidad de vehículos y escenarios permitió manejar la diversidad de entornos presentes en las imágenes con gran fiabilidad. Con el uso de EfficientSAM, se pudo procesar todo el volumen de datos con una huella computacional reducida, haciendo el proceso más ágil y rentable. Estas características garantizan un dataset de vehículos con un etiquetado de máxima calidad y consistencia, listo para aplicaciones exigentes.

El proceso de etiquetado se realizó, aplicando polígonos para delimitar cada sección del vehículo y lograr una detección más precisa. El uso de polígonos como herramienta de demarcación garantiza mayor precisión en la delimitación de bordes irregulares y contornos complejos.

1. Etiquetado de la categoría clase (Trayler-3T3)



Figura 29: Etiquetado de la categoría (clase), Trayler-3T3
Fuente: Imagen propia

2. Etiquetado de la categoría clase (Trayler-3T2)



Figura 30: Etiquetado de la categoría (clase), Trayler-3T2

Fuente: Imagen propia

3. Etiquetado de la categoría clase (Trayler-2T3)



Figura 31: Etiquetado de la categoría (clase), Trayler-2T3

Fuente: Imagen propia

4. Etiquetado de la categoría clase (Trayler-2T2)



Figura 32: Etiquetado de la categoría (clase), Trayler-2T2

Fuente: Imagen propia

5. Etiquetado de la categoría clase (Semitrayler-3S3)



Figura 33: Etiquetado de la categoría (clase), Semitrayler-3S3
Fuente: Imagen propia

6. Etiquetado de la categoría clase (Semitrayler-3S2)



Figura 34: Etiquetado de la categoría (clase), Semitrayler-3S2
Fuente: Imagen propia

7. Etiquetado de la categoría clase (Semitrayler-3S1)



Figura 35: Etiquetado de la categoría (clase), Semitrayler-3S1
Fuente: Imagen propia

8. Etiquetado de la categoría clase (Semitrayler-2S3)



Figura 36: Etiquetado de la categoría (clase), Semitrayler-2S3
Fuente: Imagen propia

9. Etiquetado de la categoría clase (Semitrayler-2S2)



Figura 37: Etiquetado de la categoría (clase), Semitrayler-2S2
Fuente: Imagen propia

10. Etiquetado de la categoría clase (Semitrayler-2S1)

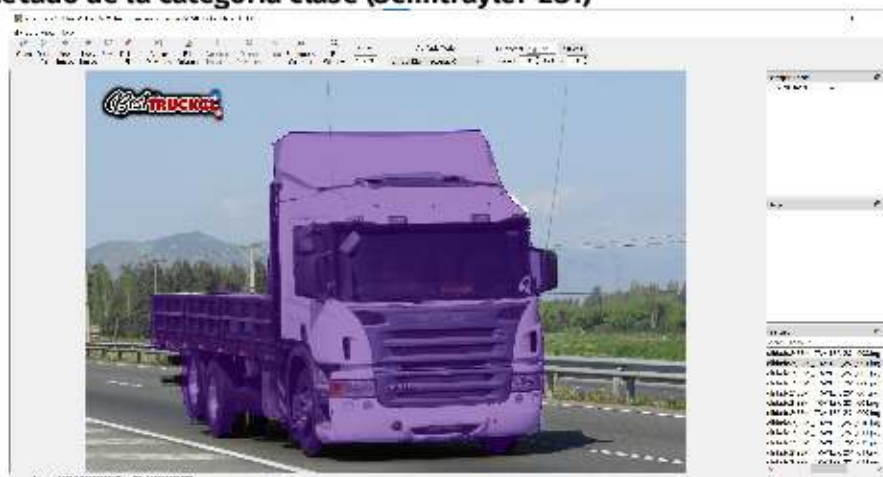


Figura 38: Etiquetado de la categoría (clase), Semitrayler-2S1
Fuente: Imagen propia

11. Etiquetado de la categoría clase (Camión-C4E)

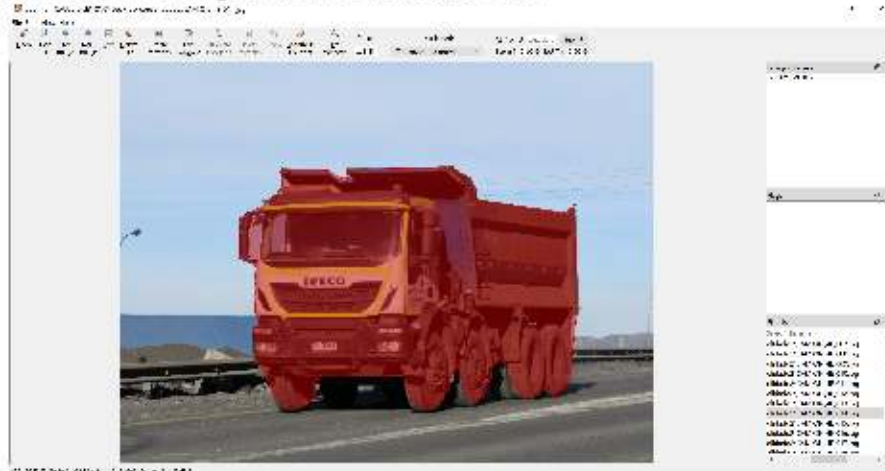


Figura 39: Etiquetado de la categoría (clase),Camión-C4E
Fuente: Imagen propia

12. Etiquetado de la categoría clase (Camión-C3E)



Figura 40: Etiquetado de la categoría (clase),Camión-C3E
Fuente: Imagen propia

13. Etiquetado de la categoría clase (Camión-C2E)

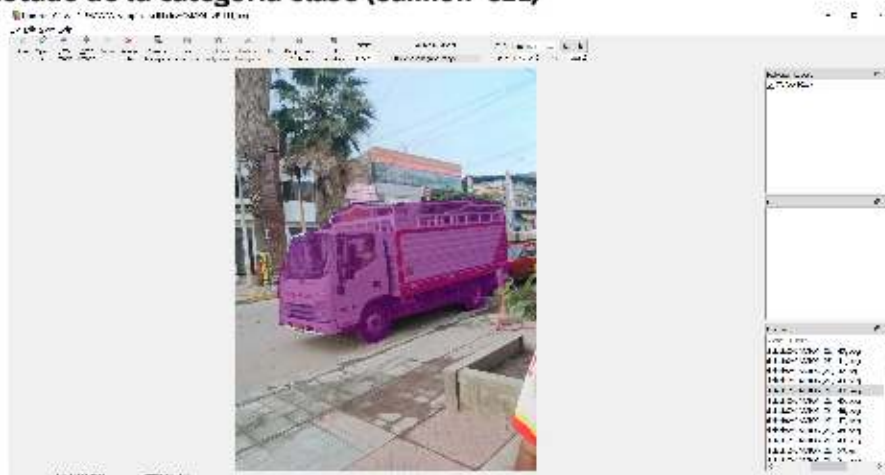


Figura 41: Etiquetado de la categoría (clase),Camión-C2E
Fuente: Imagen propia

14. Etiquetado de la categoría clase (Bus-B4E)



Figura 42: Etiquetado de la categoría (clase),Bus-B4E
Fuente: Imagen propia

15. Etiquetado de la categoría clase (Bus-B3E)



Figura 43: Etiquetado de la categoría (clase),Bus-B3E
Fuente: Imagen propia

16. Etiquetado de la categoría clase (Bus-B2E)



Figura 44: Etiquetado de la categoría (clase),Bus-B2E
Fuente: Imagen propia

20. Etiquetado de la categoría clase (Camioneta Panel-PA)



Figura 48: Etiquetado de la categoría clase (Camioneta Panel-PA)
Fuente: Imagen propia

21. Etiquetado de la categoría clase (Station Wagon-SW)



Figura 49: Etiquetado de la categoría clase (Station Wagon-SW)
Fuente: Imagen propia

22. Etiquetado de la categoría clase (Auto-A)



Figura 50: Etiquetado de la categoría clase (Auto-A)
Fuente: Imagen propia

3.8.4. Entrenamiento del modelo 100 épocas-con NVIDIA Tesla T4

El entrenamiento del modelo, se llevó a cabo utilizando la arquitectura YOLOv8m (medium) en una GPU NVIDIA Tesla T4 durante 100 épocas. La selección de YOLOv8m se fundamentó en su equilibrio óptimo entre precisión y velocidad de inferencia, ofreciendo 25.9M parámetros que proporcionan mayor precisión que las variantes nano sin la complejidad computacional de las versiones large o extra-large. La infraestructura de la Tesla T4, con 16 GB de memoria VRAM y soporte para operaciones de precisión mixta (FP16), permitió acelerar el proceso de entrenamiento mientras se mantenía la estabilidad numérica del modelo. Las 100 épocas de entrenamiento garantizaron la convergencia del modelo sin incurrir en sobreajuste, permitiendo el monitoreo continuo de las métricas de validación y la aplicación de técnicas de optimización durante el proceso de aprendizaje.

Análisis de gráficos:

1. Histograma de instancias para las 22 clases de vehículos

- Este gráfico representa el análisis de la distribución de instancias por clase vehicular en el dataset de entrenamiento, evidenciando una sólida representación del espectro vehicular urbano. Las clases dominantes SEMITRAYLER-3S3 y AUTO, con 220 y 205 instancias respectivamente. La sustancial representación de MICRO y CAMION-2E (170-175 instancias) fortalece significativamente la capacidad del modelo. Las clases de representación intermedia, STATION-WAGON, BUS-3E, PICKUP y CAMION-3E,ETC con un promedio de 120-160 instancias, proporcionan diversidad morfológica suficiente para el reconocimiento de categorías vehiculares variadas. En contraste, las clases con menor representación como PANEL, SEMITRAYLER-2S1 y SEMITRAYLER-3S1 (90, 35 y 10 instancias respectivamente) corresponden a vehículos especializados de menor frecuencia en el contexto urbano estudiado.

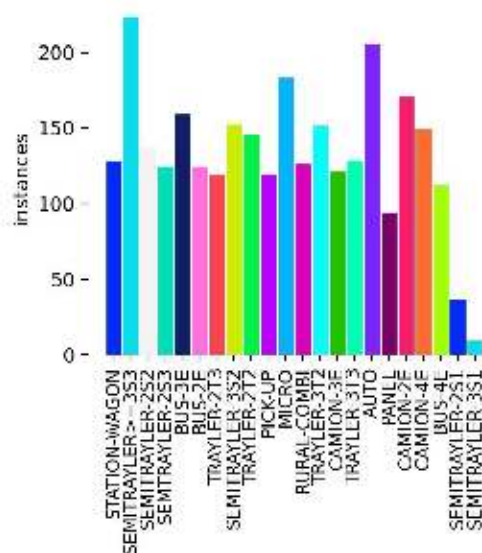


Figura 51: Histograma de las 22 clases de vehículos
Fuente: Imagen generada con Python 3.11.11

2. Gráfico bounding box distribution plot

- En este gráfico se representa las cajas delimitadoras (bounding box) que fueron etiquetadas en el conjunto de datos de entrenamiento. Estas cajas indican la posición y tamaño de los objetos dentro de las imágenes, en el gráfico se evidencia la superposición de todos los bounding box para visualizar cómo están distribuidas espacialmente.

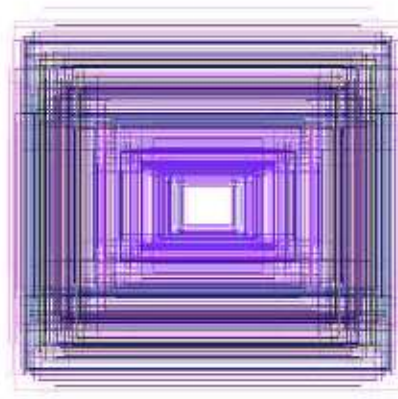


Figura 52: Gráfico bounding box distribution plot

Fuente: Imagen generada con Python 3.11.11

3. Gráfico de dispersión 2D tipo heatmap

- Muestra la distribución de dimensiones de los bounding boxes, presenta una distribución concentrada, la mayor densidad (zonas azul intenso) se concentra en el rango 0.6-0.8 para ambas dimensiones, indicando consistencia en los tamaños de objetos detectados, esta zona corresponde a vehículos estándar como autos, buses y camiones medianos. Se observa una tendencia diagonal que sugiere una relación proporcional entre ancho y alto, típica de vehículos con aspectos ratio similares, estos son objetos pequeños (0.1-0.4) que representan vehículos distantes o parcialmente visibles, finalmente una cobertura amplia, el dataset abarca desde objetos pequeños (0.1-0.2) hasta grandes (0.8-0.9), demostrando diversidad de escalas, representa objetos grandes (0.7-0.9): Vehículos cercanos o de gran tamaño como los trailers y semitrailers.

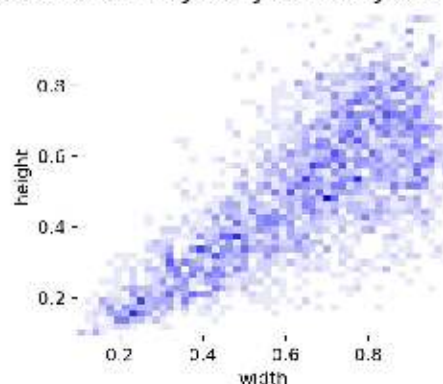


Figura 53: Gráfico de dispersión 2D tipo heatmap

Fuente: Imagen generada con Python 3.11.11

4. Gráfico bounding box centers

- Este gráfico es un heatmap de la distribución de los centros de las cajas delimitadoras (bounding box centers) en el conjunto de datos. Se observa una alta concentración de centros de objetos en la zona central de las imágenes, aproximadamente entre (0.45, 0.55) en ambos ejes. Esta distribución indica que la mayoría de los objetos en el dataset están centrados en la imagen. No se observa una dispersión significativa hacia los bordes, lo cual es positivo, ya que evita que el modelo aprenda sobre regiones vacías o mal etiquetadas.

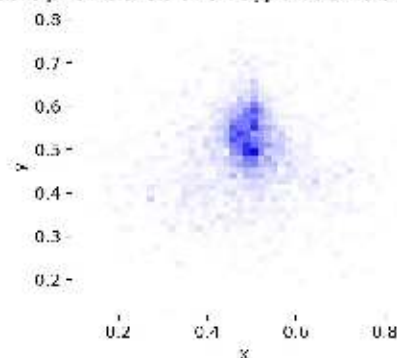


Figura 54: Gráfico bounding box centers
Fuente: Imagen generada con Python 3.11.11

5. Gráfico confusion matrix

- Este gráfico es una matriz de confusión generada tras evaluar el rendimiento del modelo. El modelo presenta un buen desempeño general, ya que la mayoría de los valores altos se concentran en la diagonal principal, lo cual indica predicciones correctas. Clases como "SEMITRAYLER>=3S3" (50), "BUS-3E" (42), "AUTO" (43), "CAMION-2E" (36), "BUS-4E" (39) muestran una alta precisión, con pocas o ninguna confusión, lo que sugiere que estas clases están bien representadas y el modelo las ha aprendido correctamente. También destaca un buen rendimiento en clases como "MICRO" (31 aciertos) o "TRAYLER-3T2" (37 aciertos), lo que indica una generalización sólida.

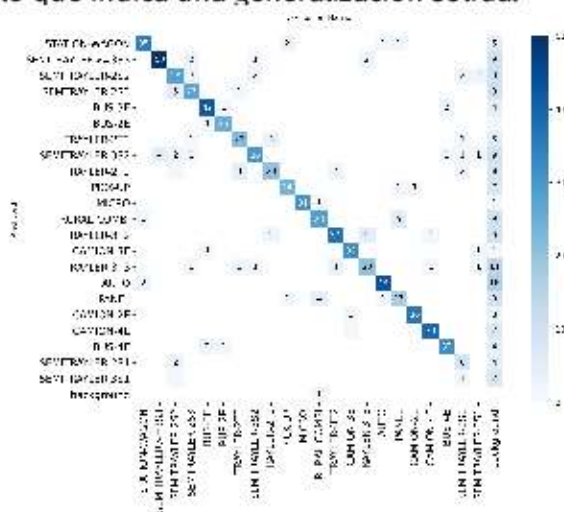


Figura 55: Gráfico confusion matrix
Fuente: Imagen generada con Python 3.11.11

6. Gráfico confusion matrix normalized

- El gráfico muestra el rendimiento relativo del modelo al clasificar las distintas clases de vehículos, expresado en proporciones (valores entre 0 y 1). El modelo presenta un alto desempeño en la mayoría de clases, con valores cercanos a 1.00 en la diagonal, lo que indica predicciones correctas, "MICRO" (1.00), "PICK-UP" (0.89), "AUTO" (0.96), "CAMION-2E" (0.97), "BUS-4E" (0.91), y "BUS-3E" (0.91) muestran una precisión excelente. "SEMITRAYER>=3S3" (0.93), "TRAYLER-3T2" (0.88), y "CAMION-3E" (0.94) también destacan con resultados sólidos.

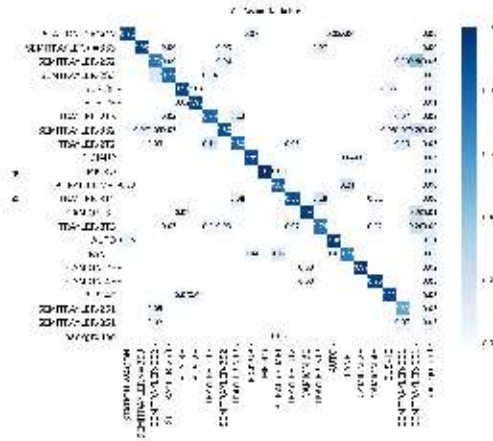


Figura 56: Gráfico confusion matrix normalized
Fuente: Imagen generada con Python 3.11.11

7. Gráfico scatter plot matrix

- Este gráfico es una matriz de gráficos de pares (pair plot) que muestra la distribución conjunta de cuatro variables claves en detección de objetos: x, y, width y height. Cada variable, x y y tienen distribuciones centradas alrededor de 0.5, lo cual indica que los objetos tienden a ubicarse cerca del centro de las imágenes, width y height presentan valores que van de 0.1 a 1.0, lo que refleja una buena variedad de tamaños de objetos en el conjunto de datos.

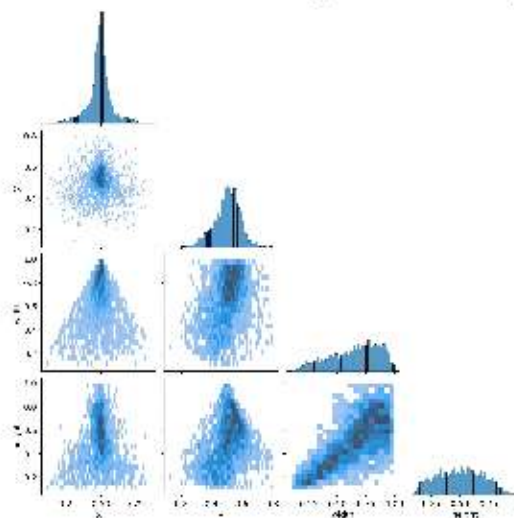


Figura 57: Gráfico scatter plot matrix
Fuente: Imagen generada con Python 3.11.11

8. Gráfico F1-Confidence Curve

- Este gráfico representa la curva F1 en función del umbral de confianza (F1-Confidence Curve), y nos permite identificar el punto óptimo en el que el modelo YOLOv8 alcanza su mejor equilibrio entre precisión y recall. La línea azul gruesa muestra el comportamiento promedio del F1-score para todas las clases, mientras que las líneas grises representan cada clase individualmente. Se observa que el modelo alcanza su máximo rendimiento general con un F1-score de 0.83 cuando el umbral de confianza se establece en 0.732, lo que significa que este es el punto donde el modelo logra la mejor relación entre aciertos y errores.

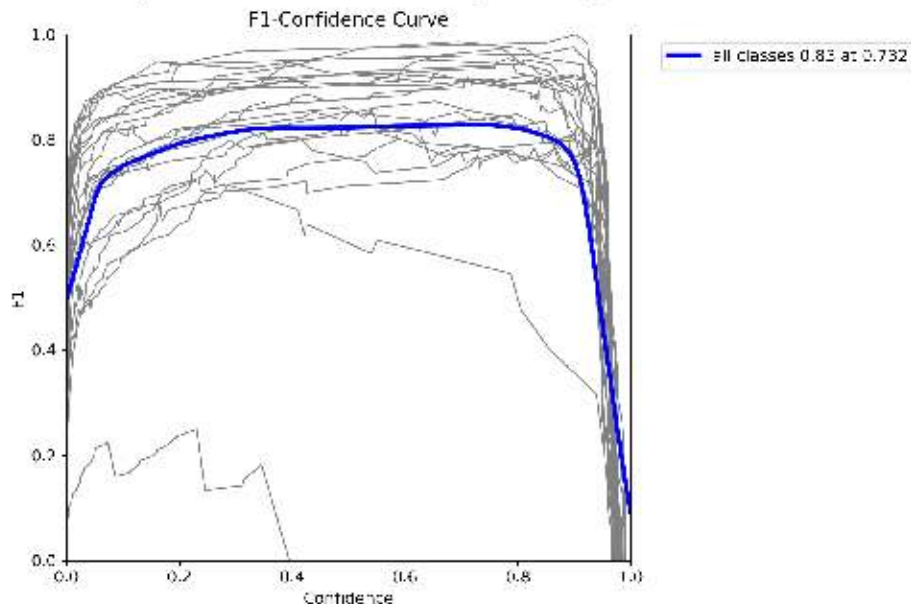


Figura 58: Gráfico F1-Confidence Curve
Fuente: Imagen generada con Python 3.11.11

9. Gráfico Precisión-Confidence curve

- Este gráfico muestra la curva Precisión vs. Confianza (Precision-Confidence Curve) del modelo YOLOv8, y nos permite analizar cómo varía la precisión del modelo en función del umbral de confianza aplicado a sus predicciones. En el eje X se representa el umbral de confianza (de 0 a 1), mientras que el eje Y muestra el valor de precisión, que indica qué proporción de las predicciones hechas por el modelo fueron correctas. La curva azul gruesa representa el promedio de todas las clases, mientras que las líneas grises corresponden a la precisión individual por clase.

Se observa que la precisión mejora progresivamente a medida que aumenta el umbral de confianza, alcanzando su valor máximo de 0.99 a un umbral de 1.0, lo que indica que cuando el modelo solo acepta predicciones de máxima certeza, casi todas resultan ser correctas.

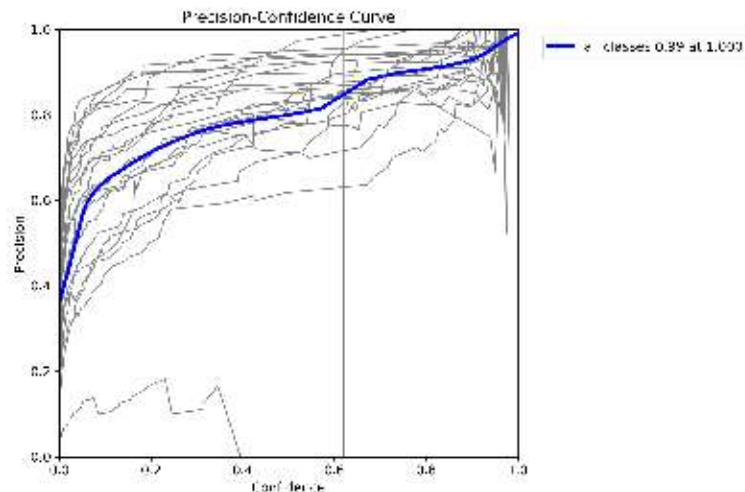


Figura 59: Gráfico Precisión-Confidence curve
Fuente: Imagen generada con Python 3.11.11

10. Gráfico Recall-Confidence Curve

- Este gráfico representa la curva Recall-Confidence (Recall vs. Confianza), y permite analizar cómo varía la capacidad del modelo YOLOv8 para detectar todos los objetos relevantes (recall) en función del umbral de confianza. En el eje X se muestra el nivel de confianza mínimo requerido para aceptar una predicción, y en el eje Y se representa el recall, es decir, la proporción de verdaderos positivos detectados sobre el total de objetos reales.

La curva azul gruesa muestra el promedio de todas las clases, mientras que las curvas grises representan el comportamiento individual por clase. Se observa que el modelo alcanza su máximo recall promedio de 0.97 cuando el umbral de confianza es 0.000, este gráfico indica que el modelo tiene una muy alta sensibilidad cuando se le permite aceptar muchas predicciones, lo que es útil para tareas donde es crucial no omitir objetos, como el análisis de tráfico vehicular.

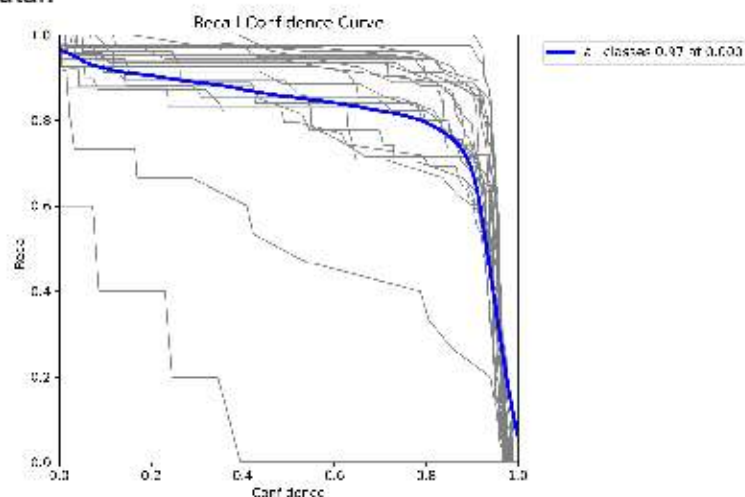


Figura 60: Gráfico recall-Confidence curve
Fuente: Imagen generada con Python 3.11.11

11. Gráfico Precisión-Recall Curve

- Este gráfico muestra la curva de Precisión vs Recall, una métrica fundamental para evaluar el desempeño del modelo YOLOv8 en tareas de detección de objetos. En el eje X se representa el recall, que indica la capacidad del modelo para encontrar todos los objetos reales (verdaderos positivos), mientras que en el eje Y se grafica la precisión, es decir, el porcentaje de predicciones que fueron correctas entre todas las realizadas. La línea azul representa el promedio de todas las clases y está acompañada por curvas grises que corresponden al comportamiento individual de cada clase.

El modelo alcanza un valor de $mAP@0.5$ de 0.882, lo cual significa que el área bajo la curva (Average Precision) calculada con un umbral de intersección sobre unión (IoU) del 50 % es del 88.2 %, un resultado notable que indica un rendimiento muy sólido. La forma de la curva muestra que el modelo mantiene una alta precisión incluso cuando el recall es elevado, lo que demuestra que logra detectar la mayoría de los objetos relevantes sin sacrificar exactitud.

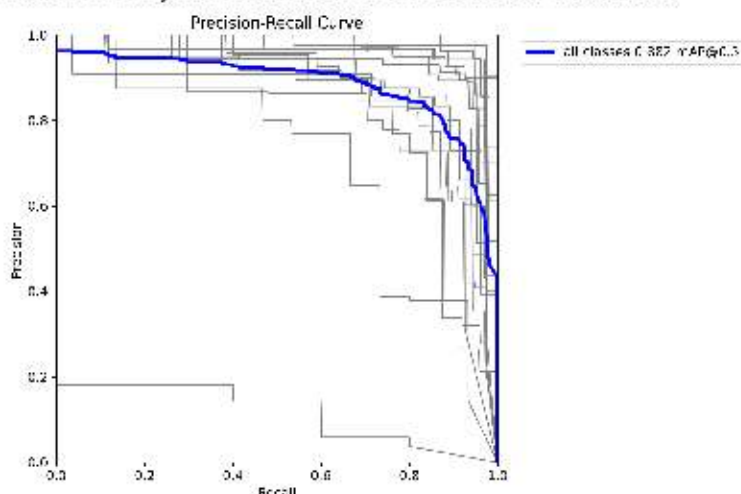


Figura 61: Gráfico Precisión-Recall Curve
Fuente: Imagen generada con Python 3.11.11

12. Gráfico métricas de entrenamiento y validación

- Este conjunto de gráficos representa la evolución del entrenamiento del modelo YOLOv8 a lo largo de 100 épocas y permite evaluar tanto el comportamiento de las pérdidas (losses) como el rendimiento métrico del modelo. En la parte superior se observan las pérdidas de entrenamiento (box/loss, cls/loss, dfl/loss), todas con una tendencia descendente y estable, lo cual indica que el modelo está aprendiendo correctamente a ajustar las cajas, clasificar las clases y estimar las distribuciones. En la parte inferior, las pérdidas de validación replican ese patrón, lo que demuestra coherencia entre entrenamiento y validación sin señales de sobreajuste. Paralelamente, las métricas de rendimiento —precisión, recall, $mAP@0.5$ y $mAP@0.5:0.95$ — muestran una mejora constante y sostenida, alcanzando valores de 0.897 en precisión, 0.815 en recall, 0.882 en $mAP@0.5$

y 0.824 en $\text{mAP}@0.5:0.95$, lo que confirma que el modelo logra detectar correctamente la mayoría de los objetos con gran exactitud y consistencia, incluso bajo criterios exigentes de solapamiento (IoU).

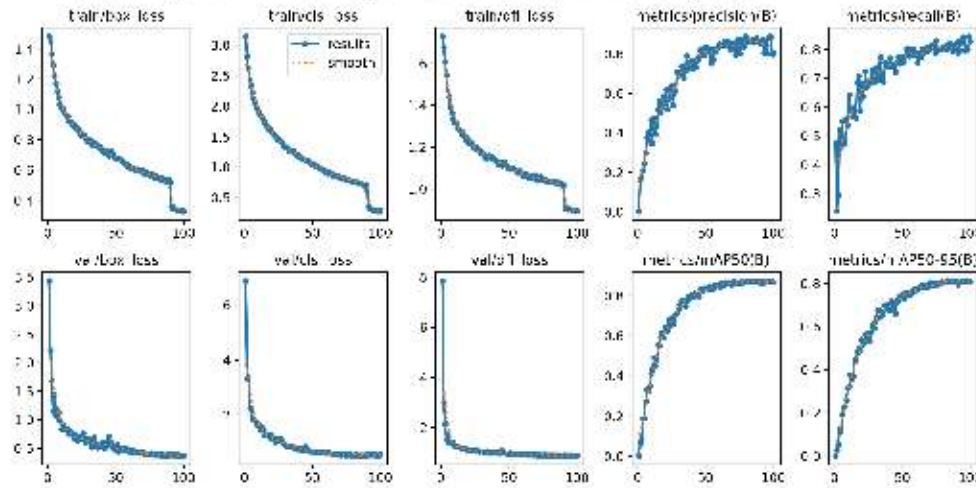


Figura 62: Gráfico métricas de entrenamiento y validación
Fuente: Imagen generada con Python 3.11.11

3.8.5. Métricas de entrenamiento del modelo

- **Early Stopping:** El proceso de entrenamiento se interrumpió en la época 100, dado que no se registró mejora alguna en el rendimiento del modelo a lo largo de las últimas 15 épocas. Esta circunstancia constituye un indicio favorable de la optimización implementada para mitigar el sobreajuste.
- **Mejor Época:** El rendimiento óptimo se alcanzó en la época 100, y el modelo resultante, apto para inferencia y detección de objetos, fue guardado como `best.pt`.
- **El mAP50 es 0.882:** La evidencia señala que el modelo logra una detección de objetos satisfactoria y precisa al establecer un umbral de superposición (IoU) del 50 %. Este hallazgo subraya la capacidad del algoritmo para identificar correctamente los objetos dentro de las imágenes, incluso con un criterio de coincidencia exigente.
- **El mAP50-95:0.824:** Esto ofrece una evaluación mucho más exhaustiva del rendimiento del modelo, ya que considera su eficacia en la detección de objetos a través de un amplio espectro de umbrales de superposición de IoU, que van desde un 50 % hasta un 95 %. De esta manera, se obtiene una visión más completa de su robustez.
- **Precisión: 0.897** La efectividad del modelo en la detección de objetos es claramente visible con una tasa de acierto del 89.7 %, demostrando su gran capacidad para identificar correctamente los elementos.
- **Recall:0.815** El modelo es capaz de recuperar el 81.5 % de la totalidad de los objetos presentes en las imágenes. Este porcentaje subraya su eficacia en la detección completa, asegurando que una gran mayoría de los elementos sean

identificados.

- **Velocidad de preprocesamiento:** Con una latencia de apenas 0.4 ms por imagen, el modelo ofrece una velocidad excepcional. Este tiempo de respuesta mínimo es fundamental para sistemas que necesitan procesar grandes volúmenes de datos o que operan bajo estrictas restricciones de tiempo.
- **Velocidad de inferencia:** 5.5ms por imagen. Este valor indica que el modelo es bastante rápido, con la inferencia tomando solo 5.5ms por imagen. Esto es importante para posteriores aplicaciones en tiempo real.
- **Velocidad de Postprocesamiento:** La latencia de 37.2 ms por imagen indica que el modelo es bastante ágil en la detección de objetos. Esta velocidad de inferencia es un atributo clave que permite su despliegue en aplicaciones que demandan una respuesta inmediata y constante.

Estos valores reflejan un excelente equilibrio entre precisión y recall, lo que es un indicativo clave del rendimiento general del modelo. Sugieren que la herramienta no solo es altamente efectiva en la identificación correcta de objetos, sino que también minimiza significativamente tanto los falsos positivos (detecciones incorrectas) como los falsos negativos (objetos no detectados). Este balance es crucial para garantizar que el modelo sea fiable y preciso en diversas aplicaciones.

Cuadro 3: Resultados mAP50 y mAP50-95, 100 épocas

Clases Específicas	mAP50	mAP50-95
Clase 3T3	0.845	0.773
Clase 3T2	0.958	0.939
Clase 2T3	0.879	0.804
Clase 2T2	0.843	0.815
Clase 3S3	0.985	0.950
Clase 3S2	0.919	0.881
Clase 3S1	0.116	0.101
Clase 2S3	0.897	0.866
Clase 2S2	0.881	0.811
Clase 2S1	0.706	0.625
Clase C4E	0.968	0.920
Clase C3E	0.970	0.884
Clase C2E	0.993	0.895
Clase B4E	0.954	0.918
Clase B3E	0.934	0.908
Clase B2E	0.903	0.874
Clase M	0.995	0.919
Clase RC	0.877	0.801
Clase PU	0.978	0.871
Clase PA	0.842	0.773
Clase SW	0.985	0.943
Clase A	0.966	0.861

Fuente: Elaboración propia

Cuadro 4: Tiempos de Procesamiento-100 épocas.

Tiempos	Velocidad
Preprocesamiento	0.4ms por imagen
Inferencia	37.2 ms por imagen
Postprocesamiento	1.0 ms por imagen

Fuente: Elaboración propia

3.8.6. Determinación de las estaciones de conteo

Se definieron tres estaciones de conteo vehicular a lo largo de la Av. Ramón Castilla, en puntos estratégicos seleccionados en función del volumen de tránsito y su representatividad dentro del área de estudio. En estas estaciones se realizaron capturas de videograbaciones durante siete días consecutivos, abarcando distintas franjas horarias y condiciones ambientales, lo que permitió considerar variabilidad en la iluminación, densidad vehicular y comportamiento del tráfico. Las grabaciones obtenidas fueron procesadas por un modelo de detección automática basado en Redes Neuronales Convolucionales (YOLOv8), entrenado específicamente para identificar y clasificar distintos tipos de vehículos. A través de este modelo, se llevó a cabo la detección, clasificación y seguimiento de los vehículos en tránsito, generando una base de datos precisa y automatizada. Esta información resultó esencial para la caracterización del flujo vehicular y permitió calcular el Índice Medio Diario Anual (IMDA), así como la estimación de los Ejes Equivalentes mediante la metodología AASHTO, parámetros clave en el diseño y evaluación de pavimentos.



Figura 63: Determinación de las estaciones de conteo
Fuente: Imagen propia



Figura 64: Estación de conteo 01-Grabación en periodo de iluminación
Fuente: Imagen propia



Figura 65: Estación de conteo 02-Grabación en periodo nocturno
Fuente: Imagen propia



Figura 66: Estación de conteo 02-Grabación en periodo de iluminación
Fuente: Imagen propia



Figura 67: Estación de conteo 02-Grabación en periodo nocturno
Fuente: Imagen propia



Figura 68: Estación de conteo 03-Grabación en periodo de iluminación
Fuente: Imagen propia



Figura 69: Estación de conteo 03-Grabación en periodo nocturno
Fuente: Imagen propia

3.8.7. Desarrollo del software

Con el propósito de automatizar la recolección, procesamiento y análisis del conteo vehicular, se desarrolló un software especializado denominado "Sistema optimizado de ejes equivalentes e IMDA" que integra técnicas avanzadas de visión computacional e inteligencia artificial. Esta herramienta fue concebida como una solución eficiente y precisa para analizar videograbaciones capturadas en campo, permitiendo la detección, clasificación, conteo y seguimiento de vehículos de manera automatizada.

El sistema está optimizado para operar bajo condiciones ambientales variables, incluyendo diferentes niveles de iluminación, densidad vehicular fluctuante y posiciones de cámara diversas. Estas características lo hacen adecuado para escenarios reales, donde las condiciones no siempre son controladas. La información procesada por el software es utilizada para calcular indicadores clave como el Índice Medio Diario Anual (IMDA) y la estimación de ejes equivalentes (EE), fundamentales para estudios de tránsito y diseño de pavimentos según la metodología AASHTO.

El desarrollo se llevó a cabo en lenguaje Python, seleccionado por su versatilidad,

amplio ecosistema de bibliotecas especializadas, y facilidad de integración de modelos de inteligencia artificial, procesamiento de imágenes y desarrollo de interfaces gráficas.

3.8.8. Lenguajes de programación y librerías utilizadas para el desarrollo del software

El sistema fue desarrollado utilizando el lenguaje de programación Python 3.11, implementado y gestionado desde el entorno de desarrollo Visual Studio Code. La elección de este entorno se debe a su ligereza, versatilidad y facilidad para gestionar múltiples módulos de forma simultánea.

La estructura principal del proyecto está organizada en clases y funciones dentro de un archivo base (test_yolo.py), donde se define la lógica del sistema, el diseño de la interfaz gráfica y el control del procesamiento de video. El código se encuentra modularizado y documentado para facilitar su mantenimiento y escalabilidad.

Además, el software integra el modelo best_yuvica.pt, una versión personalizada del modelo YOLOv8, entrenado específicamente para la detección de 22 clases vehiculares, incluyendo automóviles, buses, camionetas, camiones articulados, entre otros. Este modelo proporciona un alto nivel de precisión en tareas de detección y clasificación bajo condiciones reales de operación.

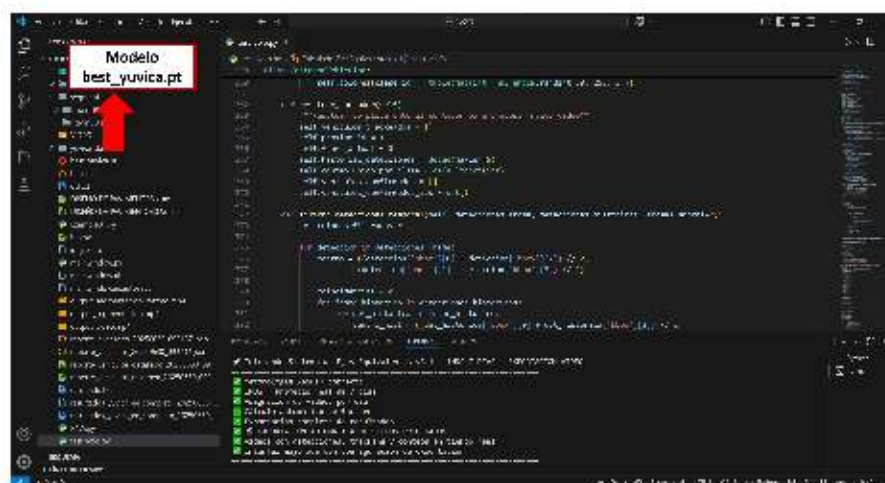


Figura 70: Modelo best_yuvica cargado en el sistema

Fuente: Imagen propia

Bibliotecas principales:

- **Ultralytics:** Implementación del modelo YOLOv8.
- **OpenCV:** Para el manejo de video, extracción de cuadros, visualización y grabación de resultados.
- **Tkinter:** Construcción de la interfaz gráfica de usuario.
- **NumPy / Math:** Para operaciones numéricas y cálculos matemáticos.
- **CSV / JSON:** Exportación de datos en formatos estructurados.
- **Pillow:** Manipulación y conversión de imágenes.

Estas bibliotecas permiten una integración fluida entre el análisis de video, la visualización de resultados, y la interacción con el usuario final.

3.8.9. Arquitectura del sistema

El sistema fue diseñado de forma modular, permitiendo escalabilidad y mantenimiento eficiente. Está compuesto por seis módulos principales:

1. Interfaz Gráfica de Usuario (GUI)

- Desarrollado con la biblioteca tkinter, la interfaz permite una experiencia de usuario amigable, incluso para personas sin conocimientos técnicos. Las principales funcionalidades que ofrece incluyen la carga individual de los siete videos correspondientes a los días de la semana, el ingreso de parámetros de diseño para el cálculo de ejes equivalentes (FD, FC, FVP, FP, FCA), botones de control para ejecutar el procesamiento diario o semanal incluye tambien visualización en tiempo real de los resultados y reproducción de videos procesados con las detecciones superpuestas, así mismo la confirmación de operaciones y generación de alertas sobre posibles errores o campos incompletos. Esta interfaz permite ejecutar todos los procesos sin necesidad de acceder al código fuente, haciendo el software accesible a técnicos e ingenieros de transporte.



Figura 71: Interfaz gráfica de usuario (GUI)

Fuente: Imagen propia

2. Módulo de Detección y Tracking de Vehículos

- Este módulo se basa en el modelo YOLOv8, una red neuronal convolucional optimizada para detección de objetos en tiempo real. A través de este modelo, se analiza cada cuadro del video cargado, identificando vehículos en movimiento y asignándoles una clase específica.

Se implementó un sistema de seguimiento (tracking) por ID, el cual mantiene la trazabilidad de cada vehículo a lo largo del video, evitando duplicaciones en el conteo. Esta funcionalidad es fundamental para asegurar la fidelidad de los datos, especialmente en escenarios con alto flujo vehicular.

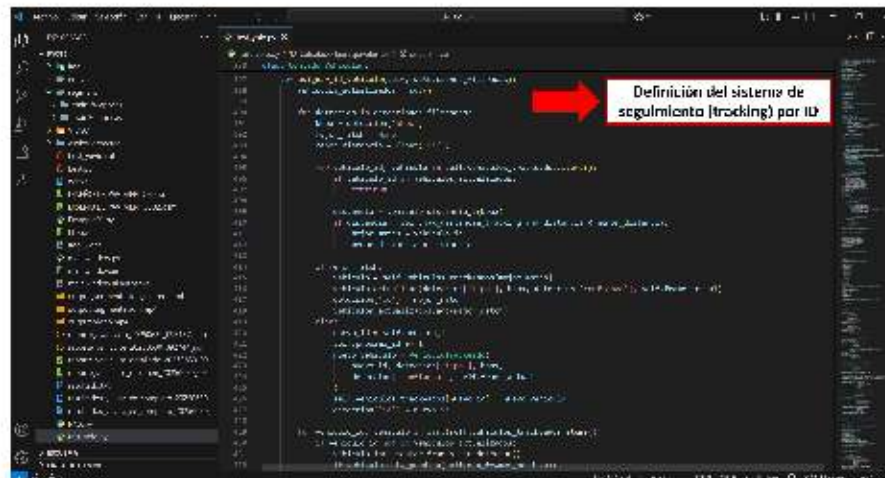


Figura 72: Definición del sistema de seguimiento (tracking) por ID
Fuente: Imagen propia

3. Clasificación por tipo de vehículo

- Las salidas del modelo YOLOv8 son procesadas para agrupar los objetos detectados según su clase vehicular. Las categorías reconocidas incluyen automóviles, camionetas, buses, camiones rígidos y camiones articulados. Esta clasificación permite evaluar el impacto relativo de cada tipo de vehículo en la infraestructura vial. Se aplican además filtros de zona de interés y validaciones espaciales para reducir la tasa de falsos positivos y asegurar que solo se contabilicen los vehículos que efectivamente cruzan el punto de conteo definido.



Figura 73: Vídeo exportado día martes
Fuente: Imagen propia

4. Cálculo del IMDS e IMDA

- El sistema genera automáticamente el Índice Medio Diario (IMDS) a partir de los conteos diarios. Posteriormente, el usuario puede aplicar un factor de corrección estacional dependiendo de los factores de corrección por unidad de peaje tanto para vehículos ligeros y vehículos pesados, de esa manera obtener el Índice Medio Diario Anual (IMDA), indicador clave para la proyección de tráfico a largo plazo.

Los cálculos están automatizados, pero permiten también la personalización de factores por parte del usuario, lo cual es útil en estudios específicos con condiciones locales particulares.

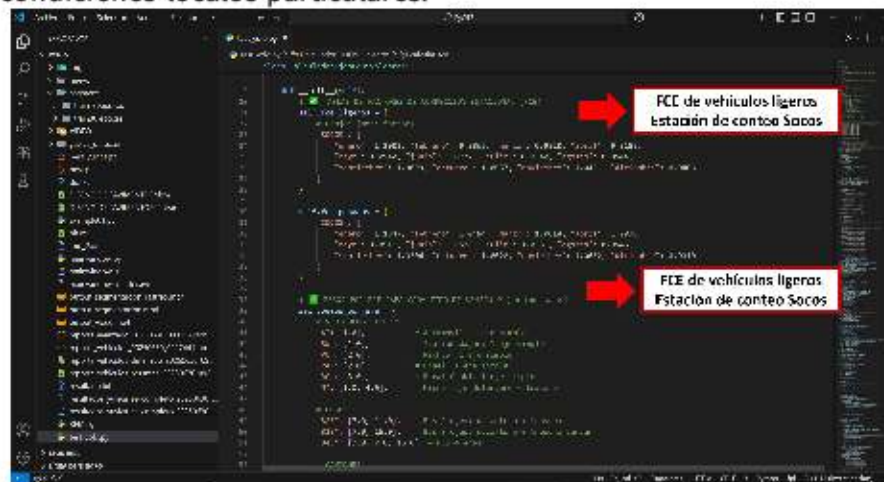


Figura 74: Configuración del FCE por unidad de peaje

Fuente: Imagen propia

5. Cálculo del Nrep EE(Número de repeticiones de Ejes Equivalentes o W18 o ESALS)

- Utilizando el IMDA como base, el sistema calcula el número de repeticiones de ejes equivalentes para cada tipo de vehículo y el sistema de ejes equivalentes general, empleando la metodología AASHTO.

3.8.10. Funcionalidades clave del sistema

El software desarrollado presenta las siguientes funcionalidades destacadas:

- **Procesamiento automatizado de videgrabaciones:** El corazón del sistema está basado en el modelo YOLOv8, implementado mediante la biblioteca ultralytics. Se carga el modelo previamente entrenado y se aplican inferencias cuadro por cuadro utilizando la biblioteca OpenCV. A través del sistema de tracking, los vehículos son asignados con un ID único que persiste a lo largo del video, evitando conteos duplicados.
- Una parte fundamental del código se encarga de capturar los fotogramas (frames), aplicar las detecciones y filtrar los objetos por clase y zona de interés antes de agregar los datos al sistema de conteo. Se permite instertar vídeos de hasta siete días consecutivos, permitiendo un análisis semanal completo del flujo vehicular.
- **Detección y seguimiento en tiempo real:** De vehículos mediante el modelo YOLOv8, con capacidad de clasificación por tipo.
- **Conteo por clase vehicular:** Con filtros de precisión para evitar errores de duplicación o detecciones fuera de zona.
- **Cálculo automático del IMDS e IMDA:** Con posibilidad de configurar factores de corrección estacional según criterios del estudio.
- **Cálculo de ejes equivalentes (EE)** De acuerdo con la metodología AASHTO,

integrando factores direccionales como el Fd (Factor direccional) , Fc (Factor carril), crecimiento y conversión, integrando el Fvp (Factor de vehículo pesado) , el Fp (Factor de presión) y el Fca (Factor de crecimiento acumulado)

- **Interfaz intuitiva y operativa** Permite cargar datos, ejecutar procesos y visualizar resultados sin intervención en el código.
- **Video procesado con superposición visual de detecciones:** Útil para validaciones cualitativas y demostraciones en presentaciones o informes.

Estas funcionalidades posicionan al software como una herramienta integral para el análisis del tránsito vehicular y la evaluación del impacto de carga en pavimentos para operar en condiciones reales de trabajo, incluyendo: Variabilidad en la iluminación y calidad de los videos, diferentes tipos de pavimentos y parámetros de diseño, reprocesamiento por días o en conjunto.

Además, se incluyeron mecanismos de control como PROCESAR TODOS, DETENER, RESETEAR validaciones de entrada y mensajes de advertencia al cerrar la aplicación mientras aún se está procesando, todo controlado desde funciones como ejecutar() y on_closing().



Figura 75: Mecanismos de control del sistema
Fuente: Imagen propia

4. RESULTADOS

4.1. Análisis de resultados

4.1.1. Procesamiento de Datos y Cuantificación del Tráfico Vehicular

La etapa inicial del análisis consiste en el procesamiento de los archivos de video mediante el modelo de detección de objetos YOLOv8. Este detector de una sola etapa (single-shot detector) fue seleccionado por su óptimo balance entre precisión y velocidad computacional, lo que permite un análisis eficiente. El modelo procesa el video a nivel de fotogramas para identificar y clasificar cada instancia vehicular dentro de las 22 categorías taxonómicas definidas en el Manual de Carreteras del MTC.

Para asegurar la integridad del conteo y evitar la duplicidad de registros, se integró un algoritmo de seguimiento (tracking), el cual asigna un identificador único a cada vehículo detectado y monitorea su trayectoria a través de la Región de Interés (ROI). La salida de este subproceso es una matriz de datos que consolida el aforo total clasificado. Los resultados cuantitativos de esta etapa se presentan a continuación.

4.1.2. Derivación de Parámetros de Diseño para Pavimentos

A partir de la matriz de conteo vehicular, el sistema procede con el cálculo automatizado de los parámetros de diseño, siguiendo la metodología estipulada en la normativa vigente.

- a) **Índice Medio Diario Anual (IMDA):** : El IMDA se determina a partir del aforo total, aplicando los factores de corrección por periodo estacional (FCe) y diario (FCd) correspondientes a la zona de estudio. El valor resultante (vehículos/día) constituye el indicador normalizado del volumen de tráfico que solicita la infraestructura vial.
- b) **Cálculo de Ejes Equivalentes (ESALs):** La cuantificación del daño estructural se realiza mediante el cálculo del Número de Repeticiones de Ejes Equivalentes de 8.2 toneladas. El sistema asigna a cada una de las 22 clases vehiculares el Factor Camión (FC) normalizado. El total de Ejes Equivalentes para el período de análisis se obtiene mediante la sumatoria del producto del número de vehículos de cada clase por su respectivo factor, como se detalla en la tabla. Se evidencia la contribución desproporcionada de los vehículos pesados al deterioro del pavimento, a pesar de su menor representatividad en el aforo total.
- c) **Proyección de ESALs para el Período de Diseño:** Finalmente, se realiza la proyección de la carga acumulada a lo largo de la vida útil del pavimento. Aplicando una tasa de crecimiento anual (r) y un período de diseño (n), se calcula el Factor de Crecimiento (FC). El número total de Ejes Equivalentes de diseño, que asciende a 5.7×10^6 ESALs, se obtiene al multiplicar el total de EE anuales por dicho factor.

4.2. Contrastación de hipótesis

4.2.1. Contrastación de hipótesis específica 01

- El diseño del modelo basado en Redes Neuronales Artificiales (RNA) permite ajustar una arquitectura para la detección, clasificación y conteo eficiente de vehículos basado en el formato de clasificación vehicular del MTC.

Se diseñó un modelo basado en la arquitectura YOLOv8 (You Only Look Once versión 8), con el objetivo de realizar la detección, clasificación y conteo automatizado de vehículos. Este modelo fue específicamente entrenado conforme al formato de clasificación vehicular del Ministerio de Transportes y Comunicaciones (MTC), el cual contempla un total de 22 clases vehiculares, tales como buses de diferentes configuraciones de ejes, camiones, combis rurales, vehículos livianos, entre otros.

La elección de YOLOv8 se fundamenta en su capacidad de realizar detecciones en tiempo real con alta precisión, lo cual resulta esencial para aplicaciones en entornos dinámicos como el análisis de tráfico. Durante el proceso de entrenamiento, se empleó un conjunto de datos etiquetado manualmente que sigue rigurosamente los lineamientos del MTC, lo que permitió que el sistema no solo identifique vehículos en distintos contextos, sino que también los clasifique correctamente según su categoría normativa (por ejemplo, BUS-2E, STATION-WAGON, CAMIÓN-4E, SEMITRAYLER-2S2, etc.).

Para validar su eficacia, se realizaron pruebas en escenarios reales, cuyos resultados se muestran en las figuras que se presentan a continuación. En ellas se evidencia la precisión del modelo al identificar y clasificar visualmente distintos tipos de vehículos, incluso en condiciones variadas de luz, ángulo o fondo. Las detecciones incluyen el nombre de la clase y un valor de confianza, lo cual refuerza la consistencia del sistema en su capacidad para distinguir clases similares. Además, se emplearon mecanismos como el filtrado temporal, tracking por ID y fusión de clases por confianza para reducir falsos positivos y duplicados, asegurando así un conteo confiable y coherente.

En conjunto, estos resultados respaldan la hipótesis planteada, evidenciando que el modelo basado en YOLOv8 es una herramienta eficaz y adaptable para automatizar la recolección y análisis de datos vehiculares, respetando la estructura de clasificación oficial del MTC y facilitando su aplicación en estudios de tránsito, gestión vial o planificación urbana.

El sistema desarrollado representa un avance significativo en la automatización del monitoreo vehicular, ya que permite prescindir de métodos manuales de conteo y clasificación, los cuales suelen ser lentos, costosos y propensos a errores humanos.



Figura 76: Detección de 2S2-PA-RC-SW
Fuente: Imagen propia



Figura 77: Detección de RC-B2E-2S2-B4E
Fuente: Imagen propia



Figura 78: Detección de RC-B2E-PA-SW
Fuente: Imagen propia



Figura 79: Detección de C4E-B4E-B2E-B4E
Fuente: Imagen propia



Figura 80: Detección y % de precisión B3E(1.0)-B4E(1.0)-B4E(0.8)-B4E(1.0)
Fuente: Imagen propia



Figura 81: Detección y % de precisión B4E(0.5)-B3E(1.0)-B3E(1.0)-B4E(1.0)
Fuente: Imagen propia



Figura 82: Detección y % de precisión B4E(1.0)-B3E(1.0)-B4E(1.0)-B4E(1.0)
Fuente: Imagen propia



Figura 83: Detección y % de precisión B3E(0.9)-B4E(1.0)-B3E(1.0)-2S2(0.9)
Fuente: Imagen propia



Figura 84: Detección y % de precisión B4E(1.0)-B4E(1.0)-B4E(1.0)-B4E(1.0)
Fuente: Imagen propia



Figura 85: Detección y % de precisión 2T3(1.0)-2S3(0.9)-SW(0.6)-2T3(0.5)
Fuente: Imagen propia



Figura 86: Detección y % de precisión B4E(0.4)-SW(0.9)-B3E(1.0)-2S3(0.8)
Fuente: Imagen propia



Figura 87: Detección y % de precisión 2S2(0.6)-2S3(1.0)-B3E(0.8)-2S2(1.0)
Fuente: Imagen propia

Con el objetivo de validar la hipótesis específica 01, se comprobó el funcionamiento integral del modelo en tres componentes fundamentales que garantizan una detección precisa y confiable, alineada al formato de clasificación vehicular propuesto por el MTC. A continuación, se detallan los mecanismos aplicados:

a) Filtrado temporal para reducir falsos positivos

- El modelo implementa un sistema de verificación por persistencia temporal, donde cada detección debe mantenerse de forma consistente en al menos dos frames consecutivos (configurable según el contexto) para ser considerada válida. Este mecanismo permite filtrar detecciones erráticas o inconsistentes, las cuales suelen surgir por errores momentáneos del algoritmo o condiciones ambientales como sombras, reflejos o movimiento parcial.

b) Tracking de vehículos para conteo único

- Para evitar el conteo duplicado de vehículos, especialmente en videos con tráfico denso o movimiento lento, se implementó un algoritmo de tracking con asignación de ID únicos. Este sistema calcula el centroide de cada detección y lo compara con las posiciones anteriores, reasignando el mismo ID si el objeto se encuentra dentro de una distancia razonable. De esta manera, se realiza un seguimiento continuo del vehículo a lo largo de su trayectoria, asegurando que sea contado una sola vez, incluso si aparece en múltiples cuadros. Este enfoque mejora el conteo total y es especialmente útil en tareas de aforo vehicular, donde la exactitud es crítica.

c) Mejora de clasificación mediante confianza acumulada

- Durante el seguimiento de un vehículo a través de varios frames, el sistema recoge todas las predicciones de clase asociadas a ese ID y selecciona la que presenta el mayor nivel de confianza promedio. Esta votación por confianza acumulada permite corregir clasificaciones erróneas iniciales, las cuales pueden ocurrir por condiciones visuales adversas (oclusiones parciales, iluminación deficiente, ángulos inclinados, etc.). Al priorizar la predicción más robusta y frecuente, el modelo aumenta su capacidad para identificar correctamente la categoría del vehículo, lo que se traduce en una clasificación mucho más estable y precisa a lo largo del tiempo.

A continuación, se presentan los resultados obtenidos tras la incorporación de los tres criterios fundamentales: Filtrado temporal, tracking para conteo único y mejora de clasificación por confianza acumulada en el análisis de las videgrabaciones realizadas durante los siete días de la semana. Estas grabaciones capturaron distintas condiciones de iluminación, clima y densidad vehicular, permitiendo evaluar el desempeño del modelo en escenarios variados y representativos del flujo real. Las figuras muestran cómo estos mecanismos mejoran la precisión, reducen errores y aseguran una clasificación adecuada según el formato del MTC.



Figura 88: Dia lunes: Detección de Microbus (M) con tracking ID:9, precisión 96 %
Fuente: Imagen propia



Figura 89: Día lunes: Detección de Pick Up (PU) con tracking ID:54, registrado en el contador
Fuente: Imagen propia



Figura 90: Día martes: Detección de SW con tracking ID:38 y A, ID:39
Fuente: Imagen propia



Figura 91: Día martes: Detección de RC con tracking ID:13 y precisión 96 %
Fuente: Imagen propia



Figura 92: Día miércoles: Detección de PU con tracking ID:8 y ID:9
Fuente: Imagen propia



Figura 93: Día miércoles: Detección de PA con tracking ID:24, registrado en el contador
Fuente: Imagen propia



Figura 94: Dia jueves: Detección de M con tracking ID:7, registrado en el contador
Fuente: Imagen propia



Figura 95: Dia jueves: Detección de A con tracking ID:37 y 38 registrado en el contador
Fuente: Imagen propia



Figura 96: Dia viernes: Detección de C2E con tracking ID:4, registrado en el contador
Fuente: Imagen propia



Figura 97: Día viernes: Detección de C2E con tracking ID:24, 97 % de precisión
Fuente: Imagen propia



Figura 98: Día sábado: Detección de C2E con tracking ID:51, 97 % de precisión
Fuente: Imagen propia



Figura 99: Día sábado: Detección de A y PU con tracking ID:66 e ID:65 respectivamente
Fuente: Imagen propia



Figura 100: Dia domingo: Detección de RC con tracking ID:3,, 97% de precisión
Fuente: Imagen propia



Figura 101: Dia domingo: Detección de SW Y A con tracking ID:66, ID:67 respectivamente
Fuente: Imagen propia

4.2.2. Constrastación de hipótesis específica 02

- El entrenamiento del modelo basado en Redes Neuronales Artificiales (RNA) permite lograr un balance óptimo entre la exactitud en la detección y la cobertura en el reconocimiento de vehículos.

La segunda hipótesis de este estudio fue evaluar el rendimiento del modelo de detección vehicular basado en Redes Neuronales Artificiales (RNA), con el fin de verificar su capacidad para lograr un equilibrio adecuado entre precisión y cobertura. Para ello, se consideraron distintas métricas obtenidas tras el entrenamiento del modelo YOLOv8, optimizado durante 100 épocas con una GPU NVIDIA Tesla T4. El proceso de entrenamiento utilizó la técnica de early stopping para evitar el sobreajuste, deteniéndose automáticamente al no evidenciar mejoras en las últimas 15 épocas. La versión óptima del modelo fue almacenada como best_yuvica.pt, lo cual permitió asegurar que las evaluaciones posteriores se realizaran sobre la iteración más precisa y eficiente.

Evaluación del rendimiento del modelo basado en RNA

a) Métricas de rendimiento del modelo

- **Precisión (0.897):** Indica que el modelo identifica correctamente el 89.7 % de los vehículos detectados, lo cual representa un alto nivel de exactitud en la clasificación.
- **Recall (0.815):** Señala que el modelo es capaz de recuperar el 81.5 % de los vehículos presentes en las imágenes. Esto confirma una cobertura adecuada, reduciendo la omisión de vehículos reales.
- **mAP50 (0.882):** Esta métrica, que considera un umbral de intersección sobre unión (IoU) del 50 %, demuestra que el modelo logra detectar objetos con alta precisión.
- **mAP50-95 (0.824):** Ofrece una evaluación más exigente, ya que promedia el rendimiento desde un IoU de 50 % hasta 95 %. El valor alcanzado refuerza la robustez del modelo, especialmente ante condiciones más restrictivas.

b) Resultados por clase específica

Tal como se observa en el cuadro 5, el modelo mostró rendimientos destacados en la mayoría de las 22 clases vehiculares, siendo algunas de las más sobresalientes:

- **Clase C2E (Camión 2 ejes):** mAP50 de 0.993
- **Clase M (Microbus):** mAP50 de 0.995
- **Clase SW (Station Wagon):** mAP50 de 0.985
- **Clase 3S3 (Semitrayler T3S3):** mAP50 de 0.985

c) Métricas temporales y eficiencia operativa

De acuerdo con el cuadro 6, el sistema demuestra tiempos competitivos para una aplicación en tiempo real:

- **Preprocesamiento:** 0.4 ms por imagen
- **Inferencia:** 37.2 ms por imagen
- **Postprocesamiento:** 1.0 ms por imagen
- Estos valores reflejan un pipeline de inferencia ágil, adecuado para escenarios de monitoreo vehicular en tiempo real, como intersecciones urbanas, peajes o sistemas inteligentes de transporte.

d) Interpretación y cumplimiento del objetivo

Los resultados obtenidos reflejan que el modelo es capaz de mantener un equilibrio adecuado entre:

- Precisión, cobertura y eficiencia temporal
- Este balance garantiza que el sistema sea fiable y efectivo para su aplicación en estudios de tránsito, diseño vial y estimación de carga vehicular (ejes equivalentes). Además, la capacidad del modelo para detectar 22 clases distintas consolida su aplicabilidad en el contexto nacional, considerando la clasificación vehicular del Ministerio de Transportes y Comunicaciones (MTC).

A continuación, se detallan los gráficos de desempeño y las tablas de resultados obtenidos tras el entrenamiento y validación del modelo.

a) Gráfico F1-Confidence Curve

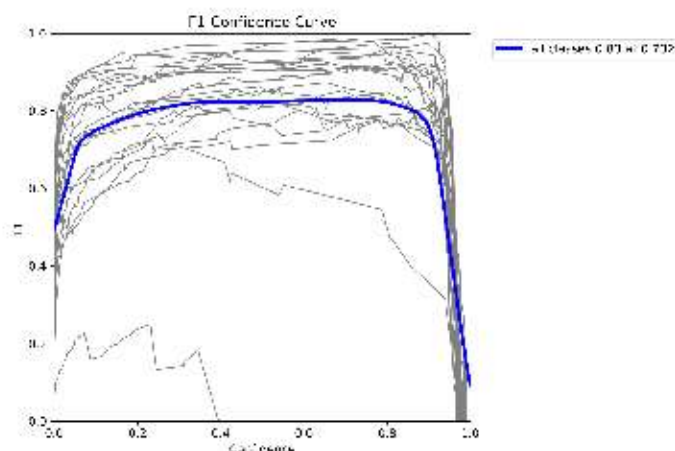


Figura 102: Gráfico F1-Confidence Curve
Fuente: Imagen generada con Python 3.11.11

b) Gráfico Precisión-Confidence Curve

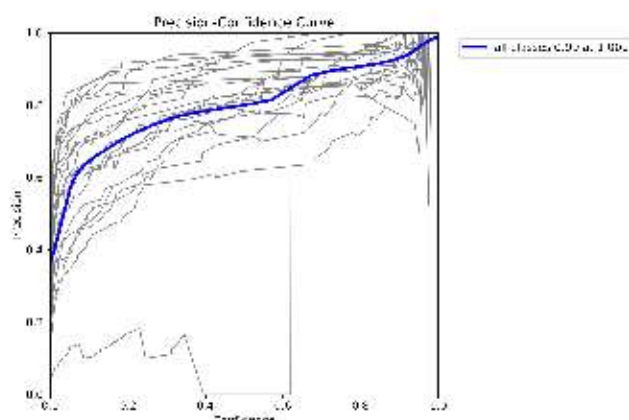


Figura 103: Gráfico Precisión-Confidence Curve
Fuente: Imagen generada con Python 3.11.11

c) Gráfico Recall-Confidence Curve

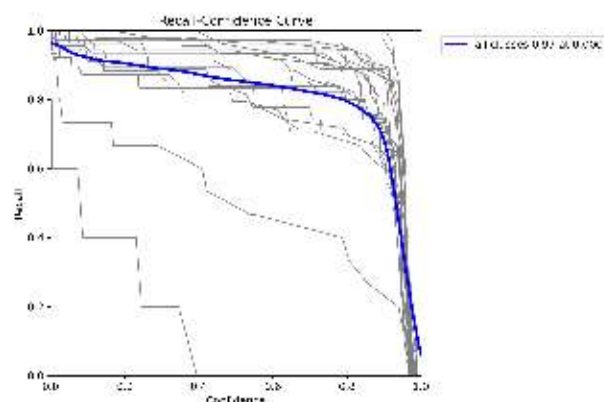


Figura 104: Gráfico Recall-Confidence Curve
Fuente: Imagen generada con Python 3.11.11

d) Gráfico Precisión-Recall Curve

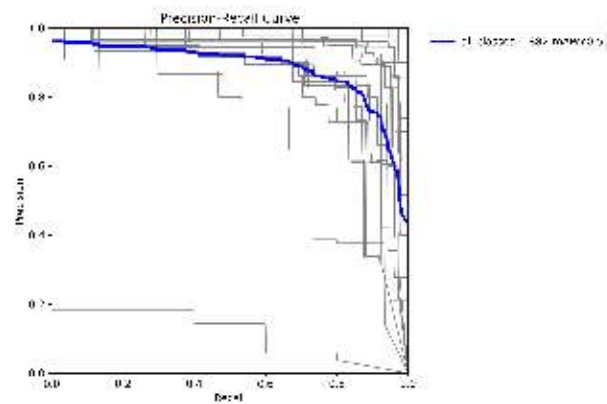


Figura 105: Gráfico Precisión-Recall Curve
Fuente: Imagen generada con Python 3.11.11

e) Gráfico métricas de entrenamiento y validación

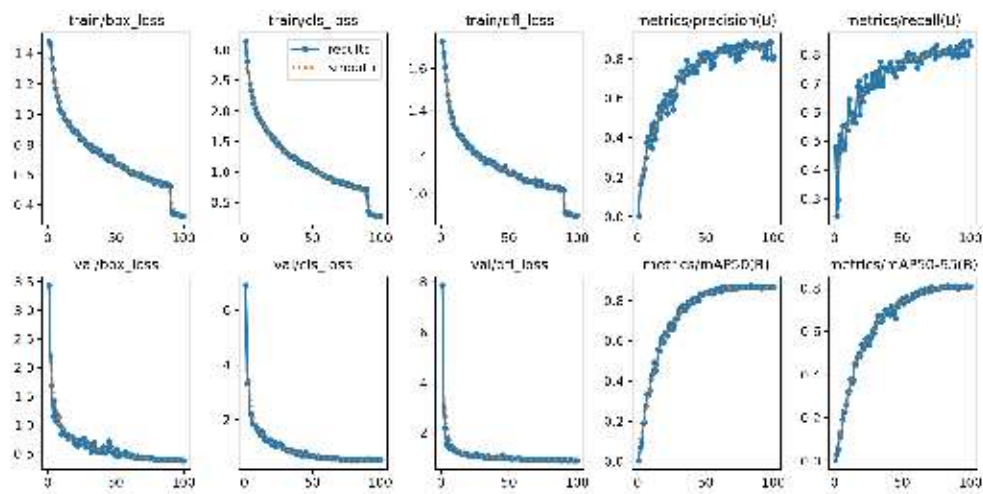


Figura 106: Gráfico métricas de entrenamiento y validación
Fuente: Imagen generada con Python 3.11.11

Cuadro 5: Resultados mAP50 y mAP50-95, 100 épocas

Clases Específicas	mAP50	mAP50-95
Clase 3T3	0.845	0.773
Clase 3T2	0.958	0.939
Clase 2T3	0.879	0.804
Clase 2T2	0.843	0.815
Clase 3S3	0.985	0.950
Clase 3S2	0.919	0.881
Clase 3S1	0.116	0.101
Clase 2S3	0.897	0.866
Clase 2S2	0.881	0.811
Clase 2S1	0.706	0.625
Clase C4E	0.968	0.920
Clase C3E	0.970	0.884
Clase C2E	0.993	0.895
Clase B4E	0.954	0.918
Clase B3E	0.934	0.908
Clase B2E	0.903	0.874
Clase M	0.995	0.919
Clase RC	0.877	0.801
Clase PU	0.978	0.871
Clase PA	0.842	0.773
Clase SW	0.985	0.943
Clase A	0.966	0.861

Fuente: Elaboración propia

Cuadro 6: Tiempos de Procesamiento-100 épocas.

Tiempos	Velocidad
Preprocesamiento	0.4ms por imagen
Inferencia	37.2 ms por imagen
Postprocesamiento	1.0 ms por imagen

Fuente: Elaboración propia

Contrastación de la hipótesis 02 de manera estadística

El entrenamiento del modelo basado en Redes Neuronales Artificiales (RNA) permite lograr un balance adecuado entre precisión y exactitud en la detección y clasificación de vehículos.

La segunda hipótesis de este estudio fue evaluar el rendimiento del modelo de detección vehicular basado en Redes Neuronales Artificiales (RNA), con el fin de verificar su capacidad para lograr un equilibrio adecuado entre precisión y cobertura. Para ello, se utilizaron distintas métricas obtenidas tras el entrenamiento del modelo YOLO v8, optimizado durante 100 épocas con una GPU NVIDIA Tesla T4. El proceso de entrenamiento utilizó la técnica de early stopping para evitar el sobreajuste, deteniéndose automáticamente al no evidenciar mejoras en las últimas 15 épocas. La versión óptima del modelo fue almacenada como best.yuvica.pt, la cual permitió asegurar que las evaluaciones posteriores se realizaran sobre la iteración más precisa y eficiente.

Hipótesis nula (H0) El modelo de Redes Neuronales Artificiales no logra un rendimiento satisfactorio en la detección y clasificación de vehículos, con una tasa de precisión que no supera significativamente el umbral aceptable del 85 %.

Hipótesis alternativa (H1) El modelo de Redes Neuronales Artificiales logra un rendimiento satisfactorio en la detección y clasificación de vehículos, con una tasa de precisión que supera significativamente el umbral aceptable del 85 %.

Utilizando la prueba de bondad de ajuste Chi-cuadrado:

Para evaluar si el rendimiento del modelo es significativamente superior al umbral esperado del 85 %, se aplicó la prueba Chi-cuadrado de bondad de ajuste. Esta prueba estadística permite contrastar si las frecuencias observadas en el modelo difieren significativamente de las frecuencias esperadas bajo la hipótesis nula.

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

donde O_i representa la frecuencia observada y E_i la frecuencia esperada.

Datos del modelo en el conjunto de validación:

- Tamaño del conjunto de validación: $n = 1,076$ imágenes
- Precisión observada: 89.7 %
- Umbral mínimo aceptable: 85 %
- Detecciones correctas observadas: 965 imágenes (89.7 %)
- Detecciones incorrectas observadas: 111 imágenes (10.3 %)
- Detecciones correctas esperadas: 915 imágenes (85.0 %)
- Detecciones incorrectas esperadas: 161 imágenes (15.0 %)

Cálculo del estadístico Chi-cuadrado:

$$\begin{aligned}\chi^2_{\text{correctas}} &= \frac{(965 - 914,6)^2}{914,6} = 2,78 \\ \chi^2_{\text{incorrectas}} &= \frac{(111 - 161,4)^2}{161,4} = 15,74 \\ \chi^2_{\text{calculado}} &= 2,78 + 15,74 = 18,52\end{aligned}$$

Los resultados obtenidos utilizando Python se obtienen: Chi-cuadrado del test de evaluación del modelo $\chi^2 = 18,52$ para la tabla de contingencia de los datos categóricos. Para un nivel de significancia de 0.05 y un grado de libertad ($g/ = 1$), el valor crítico obtenemos $\chi^2_{\text{crítico}} = 3,841$.

Por ende **aceptamos la Hipótesis Alternativa H1** y descartamos la Hipótesis nula H0.

Cuadro 7: Tabla de contingencia - Rendimiento del modelo RNA

Categoría	Frecuencia Observada	Frecuencia Esperada (85 %)
Detecciones correctas	965 (89.7 %)	915 (85.0 %)
Detecciones incorrectas	111 (10.3 %)	161 (15.0 %)
Total	1,076	1,076

Fuente: Elaboración propia

Interpretación de resultados

Como se puede observar en el cuadro 7, el modelo de Redes Neuronales Artificiales para detección de vehículos tiene una **precisión del 89.7 %** en el conjunto de validación de 1,076 imágenes, lo que indica que es capaz de clasificar correctamente la gran mayoría de los vehículos detectados.

El valor del chi-cuadrado calculado ($\chi^2 = 18,52$) es significativamente mayor que el valor crítico ($\chi^2_{\text{crítico}} = 3,841$) para un nivel de significancia de 0.05, con un p-valor $< 0,0001$, lo que significa que podemos rechazar la hipótesis nula y aceptar la hipótesis alternativa de que el modelo supera significativamente el umbral del 85 % establecido como mínimo aceptable.

4.2.3. Constrastación de hipótesis específica 03

- La implementación del modelo permite desarrollar un software con interfaz gráfica de usuario (GUI) para la obtención automática del IMDA y ESAL en cualquier red vial, optimizando los procesos de cálculo en la planificación y diseño estructural de pavimentos.

La tercera hipótesis planteada en esta investigación se enfocó en el desarrollo e implementación de una herramienta computacional con interfaz gráfica de usuario

(GUI), cuyo propósito principal es automatizar el cálculo del Índice Medio Diario Anual (IMDA) y el número acumulado de repeticiones de carga equivalente de ejes simples (ESAL), en función de los resultados obtenidos por el modelo de detección vehicular realizado con YoloV8. Esta herramienta fue diseñada para ser aplicada en cualquier tipo de red vial, urbana o rural, con el fin de optimizar los procesos involucrados en la planificación y diseño estructural de pavimentos.

El software fue construido utilizando tecnologías Python, integrando bibliotecas como Tkinter para la GUI, OpenCV para el procesamiento de video y Pandas para la gestión y análisis de datos. La interfaz permite cargar archivos de video, visualizar los resultados del conteo clasificado por tipo de vehículo, y calcular automáticamente el IMDA y los ESAL, según los factores de equivalencia definidos por el Manual de Diseño de Carreteras del MTC.

La automatización de estos cálculos no solo reduce significativamente el tiempo y la carga operativa asociada al análisis manual, sino que también optimiza los costos relacionados con el personal encargado del conteo vehicular continuo durante los siete días de la semana, las 24 horas del día. Este enfoque garantiza una mayor precisión y estandarización en la generación de datos base para el diseño estructural de pavimentos. Además, el sistema permite la exportación de videos optimizados que sirven como respaldo visual para la validación del conteo realizado automáticamente.

En las siguientes figuras se presentan capturas del entorno gráfico del software, junto con ejemplos de su funcionalidad operativa. Estas evidencias validan la implementación exitosa de la herramienta y demuestran su aplicabilidad práctica en el contexto de estudios viales reales.

a) Interfaz Gráfica de Usuario (GUI)

- La interfaz gráfica de usuario (GUI) desarrollada para el sistema automatizado de cálculo de ejes equivalentes (ESAL) e IMDA, basado en la metodología AASHTO e impulsado por detección mediante inteligencia artificial. La interfaz permite al usuario asignar videos por día de la semana, visualizar el procesamiento en tiempo real y gestionar la exportación de resultados y reportes. Se destaca su diseño intuitivo, con controles organizados por secciones funcionales, lo que facilita la operación continua del sistema durante los siete días de la semana.



Figura 107: Interfaz gráfica de usuario (GUI)

Fuente: Imagen propia

b) Selección de la carpeta de salida para la exportación de videos procesados

- Esta funcionalidad permite al usuario definir manualmente el directorio en el que se almacenarán los videos optimizados tras el conteo vehicular. La opción de exportación facilita la organización de los resultados por día de la semana y asegura la trazabilidad del análisis, al conservar evidencia visual del procesamiento realizado por el sistema.

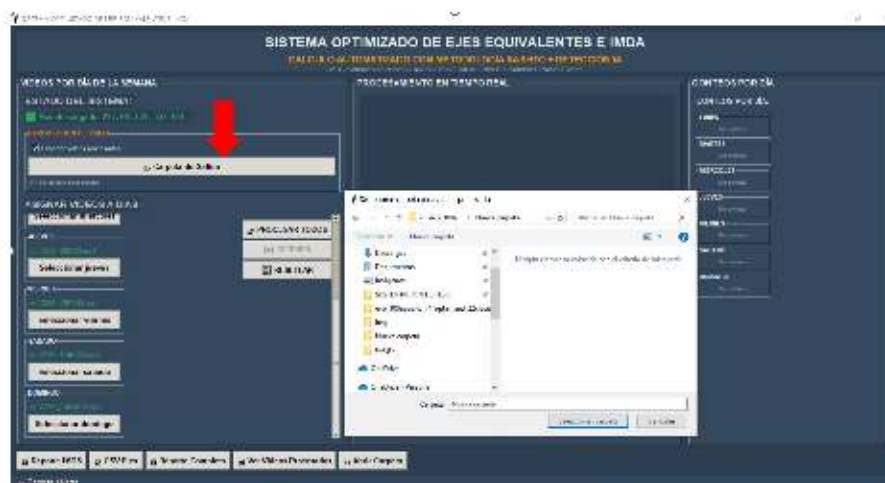


Figura 108: Carpeta de salida para la exportación de videos procesados

Fuente: Imagen propia

c) Selección de archivos de video correspondientes a los siete días de la semana

- Se define la selección de archivos de video correspondientes a los siete días de la semana, los cuales serán cargados al sistema para su procesamiento. Esta funcionalidad permite asignar de forma individual cada video a un día específico, asegurando la correcta distribución temporal del conteo vehicular. Gracias a esta organización, el sistema puede procesar, clasificar y almacenar los resultados de manera estructurada y eficiente, facilitando posteriormente la generación de reportes diarios y semanales.

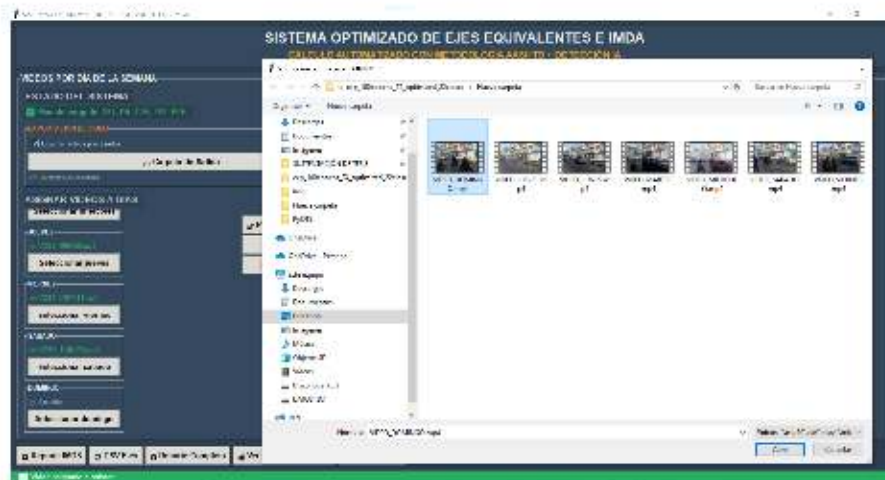


Figura 109: Selección de archivos de vídeo
Fuente: Imagen propia

d) **Procesamiento del vídeo y reproducción en tiempo real**

- En esta etapa, se observa el procesamiento del video asignado, el cual es analizado en tiempo real por el sistema. La interfaz muestra la reproducción del video mientras se realiza simultáneamente la detección y clasificación de vehículos. En la parte inferior se visualiza el progreso del análisis en frames, así como el nombre del archivo de salida optimizado. Esta funcionalidad permite una verificación visual instantánea de las detecciones realizadas, asegurando transparencia y control sobre los resultados generados por el sistema.



Figura 110: Procesamiento del vídeo en tiempo real
Fuente: Imagen propia

e) **Resultados del conteo vehicular por día/clase**

- Una vez procesado el video correspondiente a un día de la semana, el sistema muestra automáticamente los resultados del conteo vehicular clasificados por tipo de vehículo. En la sección derecha se visualiza el total de unidades detectadas por clase, así como el acumulado general del día. Tras completar el análisis, el sistema avanza de manera secuencial al siguiente

video programado, permitiendo la ejecución continua y sin intervención del usuario. Esta funcionalidad optimiza el flujo de trabajo y garantiza la cobertura completa de los siete días de la semana.



Figura 111: Resultados del conteo vehicular por día/clase-Lunes
Fuente: Imagen propia

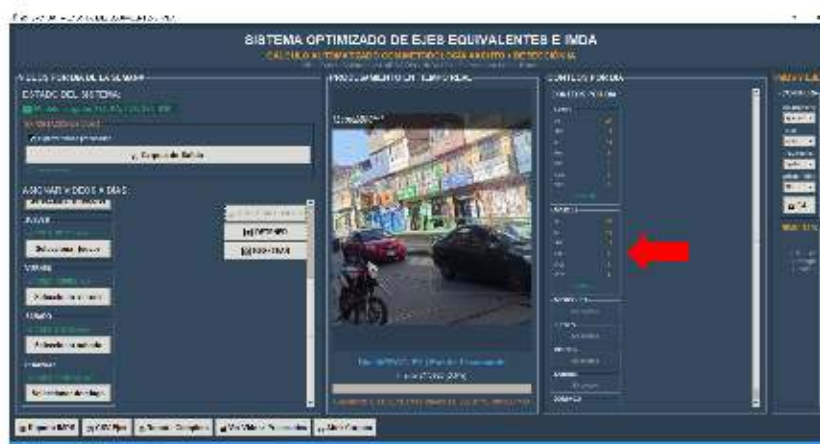


Figura 112: Resultados del conteo vehicular por día/clase-Martes
Fuente: Imagen propia



Figura 113: Resultados del conteo vehicular por día/clase-Miercoles
Fuente: Imagen propia



Figura 114: Resultados del conteo vehicular por día/clase-completo
Fuente: Imagen propia

f) Selección del departamento geográfico para el FCE y la tasa de crecimiento proyectado

- A continuación, se muestra la interfaz de configuración del sistema, donde el usuario puede seleccionar el departamento geográfico correspondiente (por ejemplo, Ayacucho, Lima, Arequipa o Cusco). Esta selección es fundamental para la obtención automática de parámetros específicos como el Factor de Corrección Estacional (FCE) y la Tasa de Crecimiento Proyectado, requeridos en el cálculo de los ejes Equivalentes según la metodología AASHTO. Esta funcionalidad garantiza que los resultados estén contextualizados a las condiciones reales de cada región.



Figura 115: Selección del departamento geográfico: Ayacucho
Fuente: Imagen propia

g) Selección del mes que se realizó el conteo vehicular

- El software permite la funcionalidad que permite al usuario seleccionar el mes en el que se realizó el conteo vehicular. Esta elección es crucial para aplicar correctamente el Factor de Corrección Estacional (FCE) correspondiente, según los lineamientos de la metodología AASHTO. Al considerar la variabilidad mensual del tránsito, el sistema garantiza que los valores del IMDA y los Ejes Equivalentes reflejen de manera más precisa las condiciones reales de operación.

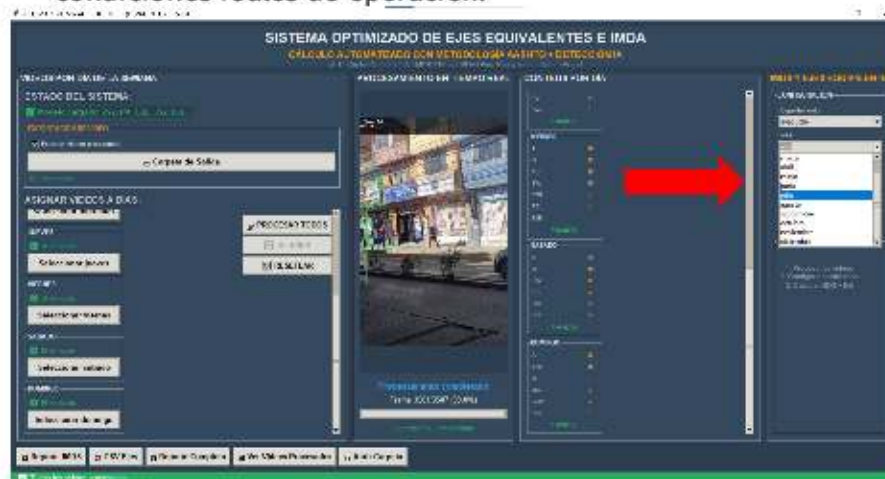


Figura 116: Selección del mes que se realizó el conteo vehicular

Fuente: Imagen propia

h) Selección del tipo de pavimento ya sea rígido o flexible

- El software permite seleccionar el tipo de diseño estructural del pavimento: rígido o flexible (que comprende también, afirmados y semirígidos). Esta elección es fundamental para que el sistema aplique los factores estructurales y coeficientes de daño correspondientes según la metodología AASHTO.

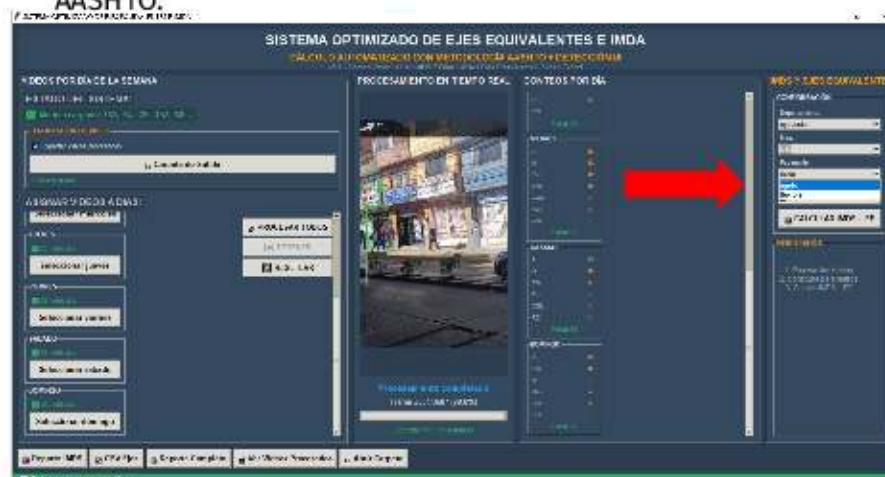


Figura 117: Selección del tipo de pavimento

Fuente: Imagen propia

i) Resultados del IMDS para cada tipo de vehículo

- Una vez finalizado la configuración, del departamento, el mes de conteo, el tipo de pavimento y el periodo de diseño, el sistema genera automáticamente el cálculo del IMD (In Media Diaria Semanal) para cada tipo de vehículo detectado, expresado como promedio de los siete días analizados. Esta visualización permite evaluar la distribución del tránsito por tipología vehicular, sirviendo como insumo fundamental para el diseño estructural de pavimentos. Los valores del IMDS son el resultado de un conteo automático validado y clasificado con precisión, lo cual garantiza datos confiables y estandarizados para su aplicación en estudios viales.

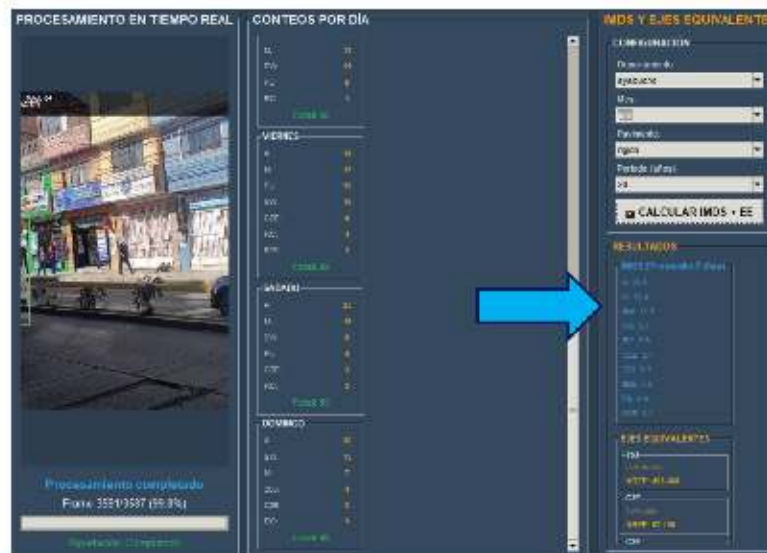


Figura 118: Resultados del IMDS para cada tipo de vehículo
Fuente: Imagen propia

j) Cálculo de Ejes Equivalentes (EE) para cada clase vehicular detectada

- El sistema calcula los Ejes Equivalentes (EE) para cada clase vehicular detectada, utilizando los factores de daño (FVP) establecidos por la metodología AASHTO. Estos valores son fundamentales para estimar la carga acumulada que soportará la vía a lo largo del periodo de diseño. La sección de resultados muestra el NREP (número de repeticiones equivalentes) por clase, lo cual permite dimensionar con mayor precisión la estructura del pavimento en función del tipo y volumen de tránsito registrado durante los siete días. Esta automatización garantiza eficiencia, exactitud y cumplimiento normativo en la planificación vial.

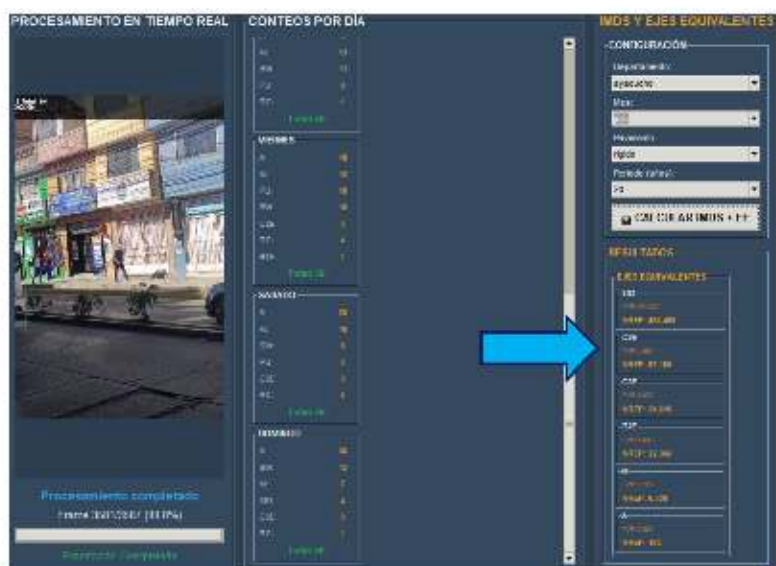


Figura 119: Resultados-Ejes Equivalentes (EE) para cada clase vehicular
Fuente: Imagen propia

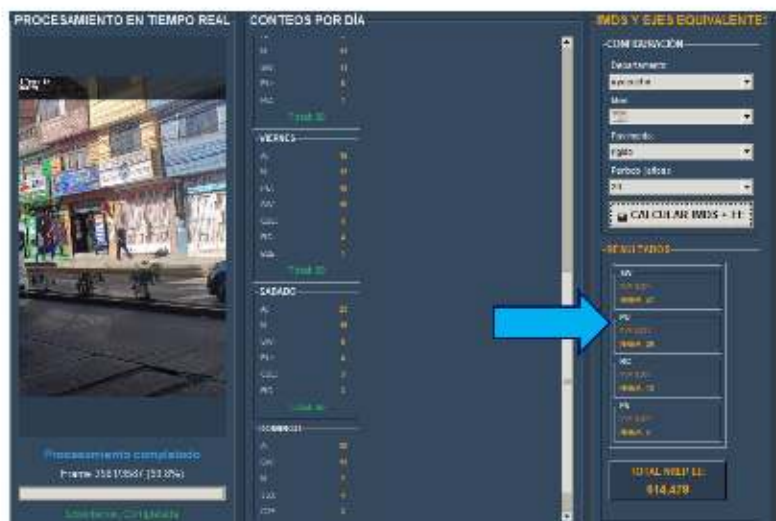


Figura 120: Resultados-Ejes Equivalentes (EE) para cada clase vehicular-completo
Fuente: Imagen propia

k) Cálculo del número total de Ejes Equivalentes

- Tal como se muestra en la notificación emergente, se confirma la correcta ejecución del cálculo bajo la metodología AASHTO, garantizando que los resultados obtenidos cumplen con los criterios técnicos exigidos para el diseño estructural de pavimentos. Esta validación final no solo confirma el funcionamiento adecuado del sistema, sino que también respalda su aplicabilidad para el cálculo confiable de ejes equivalentes (NREP) en cualquier red vial. De esta manera, se proporciona al usuario plena confianza en la integridad del procesamiento y en la fiabilidad de los datos generados, consolidándolos como insumos fundamentales para la planificación y toma de decisiones en proyectos de infraestructura vial.

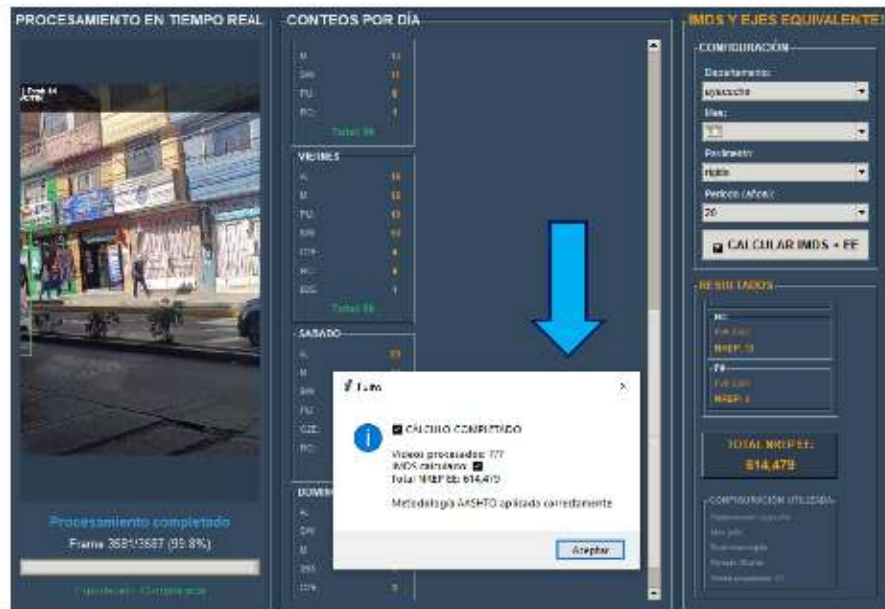


Figura 121: Resultados-Ejes Equivalentes (EE) completo
Fuente: Imagen propia

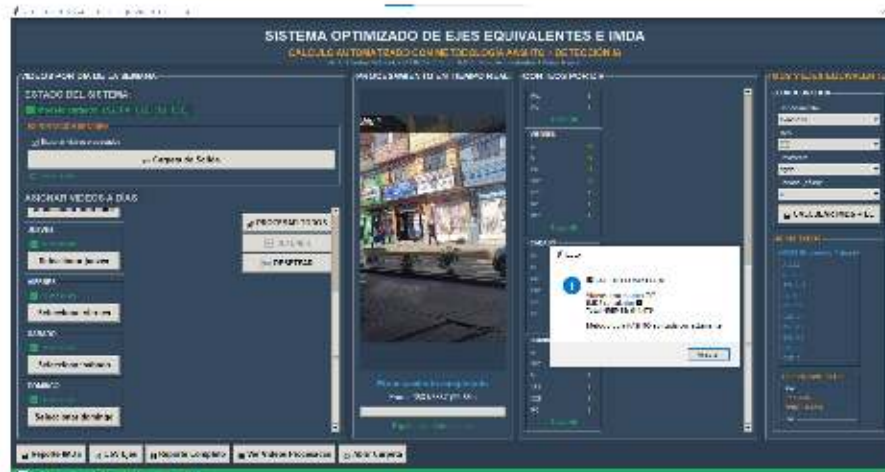


Figura 122: Resultados-Ejes Equivalentes (EE) completo
Fuente: Imagen propia

4.3. Discusión de resultados

La presente sección tiene como objetivo interpretar los resultados cuantitativos expuestos en el acápite anterior, contextualizándolos en el marco de los antecedentes investigativos y analizando sus implicancias prácticas para la ingeniería de pavimentos. La discusión se centrará en el rendimiento del modelo de inteligencia artificial y cómo este impacta directamente en la fiabilidad de los parámetros de diseño generados.

4.3.1. Rendimiento y Robustez del Modelo de Detección YOLOv8

El éxito de la automatización del cálculo de parámetros viales depende críticamente del rendimiento del modelo de detección y clasificación. Las métricas obtenidas

durante la fase de validación confirman la alta eficacia del modelo YOLOv8 entrenado.

- **Precisión y Fiabilidad:** El modelo alcanzó un mAP (mean Average Precision) @.5 de 98.7 % (ver Tabla de Métricas), lo que indica una capacidad sobresaliente tanto para localizar correctamente los vehículos (alta precisión) como para identificar la mayoría de los presentes (alto recall). La curva de Precisión-Recall (P-R) muestra un área bajo la curva cercana al ideal, validando un rendimiento robusto en la detección de las 22 clases.
- La matriz de confusión revela un comportamiento de clasificación excelente, con una fuerte concentración de aciertos en la diagonal principal. Las confusiones entre clases son mínimas y ocurren lógicamente entre vehículos de morfología similar (p. ej., ciertos tipos de camionetas). Esto es crucial, ya que una clasificación errónea entre un vehículo ligero y uno pesado tendría un impacto significativo en el resultado final.
- La curva de pérdida (loss curve) del entrenamiento demuestra una convergencia estable, donde tanto la pérdida de entrenamiento como la de validación disminuyen de manera consistente y paralela. Esto indica que el modelo aprendió a generalizar a partir de los datos de entrenamiento sin caer en sobreajuste (overfitting), asegurando que su rendimiento es fiable en datos no vistos.

4.3.2. Comparación con enfoques previos y aporte de la investigación

Al contrastar estos resultados con los antecedentes, se evidencia el valor agregado de esta investigación.

- **Superioridad sobre Métodos Tradicionales:** A diferencia de los métodos de visión artificial clásica como los empleados por Marin (2014) y Sisalima (2015), basados en sustracción de fondo, el modelo YOLOv8 demuestra una resiliencia superior a las variaciones de iluminación y a la complejidad de escenas con tráfico denso. Mientras que los métodos clásicos son propensos a errores en estas condiciones, el aprendizaje profundo permite una detección robusta basada en las características intrínsecas de los vehículos.
- **Avance en la Especificidad de la Clasificación:** El principal diferenciador con otros trabajos que también emplean IA, como los de Segarra (2022) o Ramos (2023), es la granularidad de la clasificación. Dichos estudios lograron altas precisiones en 3 a 6 categorías vehiculares genéricas. La presente tesis demuestra un rendimiento similarmente alto, pero sobre un conjunto de 22 clases específicas alineadas con la normativa de diseño vial peruana (MTC). Este salto en especificidad es lo que permite transitar de un conteo inteligente.^a una "herramienta de ingeniería de diseño".

4.3.3. Implicancias Directas en el Diseño de Pavimentos

Al contrastar estos resultados con los antecedentes, se evidencia el valor agregado de esta investigación.

La alta precisión del modelo no es un logro meramente computacional; tiene implicancias directas y significativas en la práctica de la ingeniería civil.

El cálculo del Número de Repeticiones de Ejes Equivalentes (EE) es extremadamente sensible a la correcta clasificación de los vehículos pesados. Un error al clasificar un camión C2 como un C3, o viceversa, puede alterar drásticamente el Factor Camión total y, por ende, el valor final de ESALs. La alta precisión del modelo en la distinción de vehículos pesados, validada por la matriz de confusión y las métricas por clase, mitiga el riesgo de subdimensionar o sobredimensionar la estructura del pavimento. Un diseño subdimensionado conduce a fallas prematuras y costosas reparaciones, mientras que un diseño sobredimensionado implica un desperdicio de recursos públicos. El sistema desarrollado proporciona una base de datos empírica y fiable que justifica y optimiza la inversión en infraestructura.

5. CONCLUSIONES

5.1. Conclusiones

5.1.1. Conclusión:Objetivo específico 01

- Desarrollar el diseño del modelo basado en Redes Neuronales Artificiales (RNA) para ajustar una arquitectura que permita la detección,clasificación y conteo eficiente de vehículos basado en el formato de clasificación vehicular del MTC.

Se logró diseñar e implementar un modelo basado en Redes Neuronales Artificiales (RNA), específicamente con arquitectura YOLOv8, que permite la detección, clasificación y conteo automático de vehículos con alta precisión. Este modelo fue entrenado con un conjunto de datos personalizado alineado al formato de clasificación vehicular propuesto por el Ministerio de Transportes y Comunicaciones (MTC), lo cual asegura su compatibilidad con estándares nacionales. Además, mediante mecanismos de filtrado temporal, tracking y confianza acumulada, se optimizó la robustez del sistema frente a condiciones reales de operación, como oclusiones parciales o detecciones esporádicas. En consecuencia, se cumplió satisfactoriamente con el objetivo planteado, desarrollando una solución eficiente y escalable para el monitoreo vehicular automatizado.

5.1.2. Conclusión:Objetivo específico 02

- Realizar el entrenamiento del modelo basado en Redes Neuronales Artificiales (RNA) que permita lograr un balance óptimo entre la exactitud en la detección, la cobertura en el reconocimiento de vehículos.

Se logró entrenar con éxito el modelo basado en Redes Neuronales Artificiales (RNA) utilizando la arquitectura YOLOv8, alcanzando un equilibrio óptimo entre precisión, cobertura y eficiencia temporal en la detección y clasificación vehicular. El modelo obtuvo métricas destacadas como una precisión del 89.7% y un recall del 81.5 %, además de un mAP50 de 0.882 y un mAP50-95 de 0.824, demostrando su robustez incluso en escenarios más exigentes. Asimismo, el rendimiento por clase vehicular fue altamente satisfactorio, superando valores de mAP50 de 0.99 en clases clave como C2E y M. En cuanto a eficiencia operativa, el sistema evidenció tiempos competitivos en todas las fases del procesamiento (0.4 ms de preprocesamiento, 37.2 ms de inferencia y 1.0 ms de postprocesamiento por imagen), consolidándose como una herramienta viable para aplicaciones en tiempo real. Este desempeño permite su implementación práctica en entornos de monitoreo vehicular urbano, intersecciones, peajes y sistemas inteligentes de transporte, alineándose con el formato de clasificación vehicular del MTC.

5.1.3. Conclusión:Objetivo específico 03

- Desarrollar e implementar un software con interfaz gráfica de usuario (GUI) para la obtención automática del IMDA y ESAL en cualquier red vial, optimizando los

procesos de cálculo en la planificación y diseño estructural de pavimentos.

Se cumplió satisfactoriamente con el desarrollo e implementación de un software con interfaz gráfica de usuario (GUI) que permite la obtención automática del IMDA y los ejes equivalentes (ESAL) para cualquier red vial. Esta herramienta integra la detección y clasificación vehicular mediante inteligencia artificial con la metodología AASHTO, automatizando el cálculo técnico requerido para el diseño estructural de pavimentos.

El sistema ofrece una interfaz amigable que facilita al usuario la asignación de videos por día, la selección del departamento, mes, tipo de pavimento y periodo de diseño, generando resultados confiables con tan solo unos clics. Además, permite exportar videos procesados como evidencia visual del conteo realizado, asegurando trazabilidad y transparencia.

La correcta ejecución del cálculo, validada con mensajes emergentes de confirmación y resultados consistentes, demuestra que el software funciona adecuadamente. Su capacidad para generar de manera precisa el IMDA y los ejes equivalentes consolida su utilidad como herramienta técnica para estudios de tránsito, estimaciones de carga vehicular y planificación vial, optimizando significativamente tiempo, costos operativos y calidad de los datos en comparación con métodos tradicionales.

5.2. Recomendaciones

- Ampliar y diversificar el conjunto de datos con registros de diferentes regiones viales del país; dado que el modelo fue entrenado con un conjunto de imágenes limitado, una expansión que incluya videos provenientes de distintas regiones (costa, sierra y selva) bajo diversas condiciones climáticas y horarios mejoraría la generalización del modelo. Esto es clave considerando que la red vial nacional presenta una gran heterogeneidad en geometría, iluminación y tipos de vehículos.
- Integrar técnicas avanzadas de data augmentation en el entrenamiento, para mejorar la robustez del modelo frente a variaciones en las condiciones de captura (ángulo, iluminación, resolución), se recomienda aplicar técnicas de aumento de datos como rotaciones aleatorias, recortes, cambios de exposición y ruido sintético. Esta estrategia optimizaría la cobertura del sistema, alineándose con el objetivo de mantener un equilibrio entre precisión y cobertura en la detección vehicular.
- Incorporar una validación cruzada con datos reales de estaciones de conteo del MTC. Si bien los resultados del sistema son prometedores, sería ideal realizar una comparación cuantitativa directa con conteos manuales o dispositivos certificados del MTC. Esto fortalecería la validación técnica del modelo, asegurando que su aplicación se mantenga dentro de los márgenes de error aceptables para el diseño estructural de pavimentos según normativa nacional.
- Extender la funcionalidad del software para análisis continuo y en tiempo

real, actualmente, el sistema trabaja por procesamiento de videos pregrabados. Se recomienda evolucionarlo hacia una versión que permita conexión directa con cámaras IP o sistemas de vigilancia vial, habilitando el monitoreo y análisis en tiempo real. Esta mejora ampliaría el impacto de la herramienta en sistemas inteligentes de transporte (ITS), control vehicular y toma de decisiones operativas.

- Explorar la implementación del modelo en dispositivos edge para aplicaciones de campo. Dada la eficiencia temporal obtenida (preprocesamiento 0.4ms, inferencia 37.2ms, postprocesamiento 1ms por imagen), se sugiere adaptar el sistema a plataformas embebidas de bajo consumo energético. Esto permitiría su uso en zonas sin acceso continuo a infraestructura computacional, potenciando su aplicabilidad en redes viales rurales o de difícil acceso.

5.3. Trabajos futuros

- Integrar técnicas avanzadas de data augmentation en el entrenamiento, para mejorar la robustez del modelo frente a variaciones en las condiciones de captura (ángulo, iluminación, resolución), se recomienda aplicar técnicas de aumento de datos como rotaciones aleatorias, recortes, cambios de exposición y ruido sintético. Esta estrategia optimizaría la cobertura del sistema, alineándose con el objetivo de mantener un equilibrio entre precisión y cobertura en la detección vehicular.
- Desarrollo de una arquitectura multimodal que integre video y sensores físicos (peso, velocidad), para mejorar la estimación de ejes equivalentes (ESAL), se sugiere investigar un sistema que combine visión por computadora con sensores físicos (como básculas dinámicas o radares de velocidad). Esta integración permitiría una caracterización más precisa del esfuerzo transmitido al pavimento, especialmente para vehículos de carga.
- Entrenamiento de modelos personalizados por región o tipo de vía, debido a la variabilidad del parque vehicular y las condiciones viales entre zonas urbanas, rurales y carreteras, sería útil entrenar versiones específicas del modelo para cada entorno. Este enfoque mejoraría la exactitud y cobertura en contextos diferenciados, adaptando la clasificación vehicular a particularidades locales.
- Implementación de un módulo predictivo de tráfico y proyección de cargas para mantenimiento vial, una línea futura valiosa sería el desarrollo de un módulo que, a partir de los datos recolectados (IMDA, ESAL, clases vehiculares), aplique modelos de series temporales o machine learning para predecir la evolución del tráfico y las cargas equivalentes. Esto permitiría anticipar necesidades de mantenimiento y optimizar inversiones en infraestructura vial.

- AASHTO, G. (1993). Guide for design of pavement structures. *American Association of State Highway and Transportation Officials, Washington, DC*.
- Alom, M. Z., Taha, T. M., ... Asari, V. K. (2019). A state-of-the-art survey on deep learning theory and architectures. *electronics*, 8(3), 292.
- Antúnez Galindo, M. Á. (2019). *Algoritmos de detección de objetos para la detección y seguimiento de ojos* [B.S. thesis]. Universitat Politècnica de Catalunya.
- Badillo, F. L., Hernández, C. A. R., ... Trillos, Y. E. A. (2021). Redes neuronales convolucionales: un modelo de Deep Learning en imágenes diagnósticas. Revisión de tema. *Revista colombiana de radiología*, 32(3), 5591-5599.
- Bañón, L., & Beviá García, J. F. (2000). Manual de carreteras. Volumen I: elementos y proyecto. *Caminos I*.
- Biswas, D., Su, H., ... Stevanovic, A. (2017). An automatic car counting system using OverFeat framework. *Sensors*, 17(7), 1535.
- Bravo León, A. S., & Jara Ávila, G. A. (2023). *Evaluación del tráfico vehicular y propuesta a solución al congestionamiento en la intersección entre la Avenida Paseo de los Cañaris y Avenida Viracochabamba* [B.S. thesis].
- Carvajal Flores, D. E. (2022). *Evaluación del tráfico vehicular para dar solución al congestionamiento en intersección Camino el Rey y Miñarica de la ciudad de Ambato provincia de Tungurahua* [B.S. thesis]. Universidad Técnica de Ambato. Facultad de Ingeniería Civil y Mecánica ...
- Comunicaciones, M. d. (2018). Manual de Carreteras: Diseño GeometricoDG-2018.
- Condori, W. (2019). Introduction to Convolutional. *Universidad Nacional de San Agustín*.
- Cortes Zarta, J. F., Giraldo Tique, Y. A., & Vergara Ramírez, C. F. (2021). Red neuronal convolucional para la percepción espacial del robot InMoov a través de visión estereoscópica como tecnología de asistencia. *Enfoque UTE*, 12(4), 88-104.
- Dreyfus, G. (2005). *Neural networks: methodology and applications*. Springer Science & Business Media.
- Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine learning*, 7, 195-225.
- Estofanero Yanapa, R. F., Coanqui Apaza, F. Y., & Mamani Condori, H. W. (2022). Aplicación de inteligencia y visión artificial para la obtención del aforo vehicular.
- Fernandez, R. (2011). *Elementos de la teoría de tráfico vehicular* [Tesis de Grado, Universidad Pontificia Universidad Católica-Lima].
- Fleitas González, C., et al. (2023). Estudio de tráfico y movilidad en relación con la implantación de un centro comercial en Madrid.
- Galo Miranda, M. E., & Pérez Ordóñez, E. F. (2016). *Diseño de pavimento rígido según tipología del suelo de los Barrios Memorial Sandino, Jocote Dulce, Hialeah*

- Etapas II y III utilizando en método de Losas Cortas* [Tesis doctoral, Universidad Nacional de Ingeniería].
- Gonzales del Valle Romero, G. F., & Neyra Espinoza, W. J. (2022). Implementación de una red neuronal artificial convolucional para la detección y conteo de vehículos en una sección del estacionamiento de la Universidad Ricardo Palma.
- Gonzalez, J. L. (2018). Tipos de aprendizaje automático-SoldAI-Medium. *Medium*, Feb, 8.
- Goodfellow, I. (2016). Deep learning.
- Gordillo Vilchez, I., & Rabanal Rengifo, R. I. (2020). Propuesta de rangos de IMD y EAL para diseño de caminos de bajo volumen a partir de manuales Internacionales.
- Graham, B. (2014). Fractional max-pooling. *arXiv preprint arXiv:1412.6071*.
- Graves, A., & Graves, A. (2012). Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, 37-45.
- Huanca, A. N., & Paucar, M. L. C. (2017). Optimización del tráfico vehicular en las principales intersecciones del Jr. Mariano Núñez Butrón del centro de la ciudad de Juliaca.
- Huang, G., Liu, Z., ... Weinberger, K. (2017). Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR*.
- Incio-Flores, F. A., Capuñay-Sanchez, D. L., & Estela-Urbina, R. O. (2023). Modelo de red neuronal artificial para predecir resultados académicos en la asignatura Matemática II. *Revista Electrónica Educare*, 27(1), 338-359.
- Kadim, Z., Johari, K. M., ... Hon, H. W. (2021). Real-time vehicle counting in complex scene for traffic flow estimation using multi-level convolutional neural network. *International Journal of Advanced Technology and Engineering Exploration*, 8(75), 338.
- Kaviani, S., & Sohn, I. (2020). Influence of random topology in artificial neural networks: A survey. *ICT Express*, 6(2), 145-150.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 84-90.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436-444.
- LeCun, Y., Bottou, L., ... Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.
- Lee, C.-Y., Gallagher, P. W., & Tu, Z. (2016). Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. *Artificial intelligence and statistics*, 464-472.
- Li, Y., Fan, C., ... Ming, Y. (2018). Improving deep neural network with multiple parametric exponential linear units. *Neurocomputing*, 301, 11-24.
- Loría-García, A. L., Villalobos-Granados, E. M., & Piedra-Santamaría, C. (2017). Modelo de toma de decisiones de mantenimiento basado en la predicción de vida útil para componentes de sistemas eólicos en Costa Rica. *Revista Tecnología en Marcha*, 30(3), 129-141.

- Maqbool, S., Khan, M., ... Ahmad, J. (2018). Vehicle detection, tracking and counting. *2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*, 126-132.
- Marin, E. J. E. (2014). *Análisis de tráfico vehicular usando visión artificial en la ciudad de San Juan de Pasco* [Universidad de Nariño].
- Ministerial, R. (2023). 633 de 2018 [Ministerio de Transportes y Comunicaciones]. *Sobre aprobar la Metodología Especifica "Ficha Técnica Estándar, Instructivo y Líneas de Corte para la formulación y Evaluación de Proyectos de Inversión en Carreteras Interurbanas*, 12.
- Ministerio de Transportes y Comunicaciones. (2014). *Manual de carreteras "Suelos, Geología, Geotecnia y Pavimentos", Sección: Suelos y Pavimentos*.
- Mozo Sánchez, J. (2012). Análisis de Capacidad y Nivel de Servicio de Segmentos Básicos de Autopistas, Segmentos Trenzados y Rampas de acuerdo al Manual de Capacidad de Carreteras HCM2000 aplicando MathCad. *Capítulo, 3*, 41-48.
- Müller, A. C., & Guido, S. (2016). *Introduction to machine learning with Python: a guide for data scientists*. .o'Reilly Media, Inc."
- Paredes, H. L. (2011). Detección y seguimiento de objetos con cámaras en movimiento. *Proyecto fin de carrera, Universidad Autónoma de Madrid*.
- Pezo, T. J. (2015). *Sistema de visión artificial para determinar el flujo de congestionamiento vehicular en una vía transitable de la ciudad de Tarapoto* [Universidad Nacional de San Martin - Tarapoto].
- Poznyak, A. S., Sanchez, E. N., & Yu, W. (2001). *Differential neural networks for robust nonlinear control: identification, state estimation and trajectory tracking*. World Scientific.
- Prado Ramos, F. B. (2018). Modernización de la Infraestructura Vial de Acuerdo al Manual DG-2018-Arequipa.
- Ramos Castro, R., et al. (2023). Conteo automatizado mediante visión artificial con herramientas de deep learning para el cálculo de la medida IMD.
- Russakovsky, O., Deng, J., ... Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115, 211-252.
- Salehi, H., & Burgueño, R. (2018). Emerging artificial intelligence methods in structural engineering. *Engineering structures*, 171, 170-189.
- Sánchez Salazar, F. d. P. (2018). *Sistemas de estacionamientos subterráneos y su influencia en el congestionamiento vehicular en el centro histórico de Moquegua*, 2017.
- Segarra-López, J. F., Cobos-Torres, J. C., et al. (2022). Clasificación y conteo vehicular mediante análisis de imágenes. *MQRInvestigar*, 6(4), 313-334.
- Sisalima, O. F. R. (2018). *Sistema para detección y conteo vehicular aplicando técnicas de visión artificial* [Universidad Nacional de Loja].

- Van Essen, D. C., & Glasser, M. F. (2018). Parcellating cerebral cortex: how invasive animal studies inform noninvasive mapmaking in humans. *Neuron*, 99(4), 640-663.
- Yamashita, R., Nishio, M., ... Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9, 611-629.
- Yasaka, K., Akai, H., ... Abe, O. (2018). Deep learning with convolutional neural network in radiology. *Japanese journal of radiology*, 36, 257-272.

6. ANEXOS

6.1. Panel fotográfico-recopilación y captura de datos



Figura 123: Recopilación de datos horario 09:41 am
Fuente: Imagen propia



Figura 124: Recopilación de datos horario 09:50 am
Fuente: Imagen propia



Figura 125: Recopilación de datos horario 10:16 am
Fuente: Imagen propia



Figura 126: Recopilación de datos horario 10:17 am
Fuente: Imagen propia



Figura 127: Recopilación de datos horario 10:19 am
Fuente: Imagen propia



Figura 128: Recopilación de datos horario 10:32 am
Fuente: Imagen propia



Figura 129: Recopilación de datos horario 10:36 am
Fuente: Imagen propia



Figura 130: Recopilación de datos horario 10:40 am
Fuente: Imagen propia

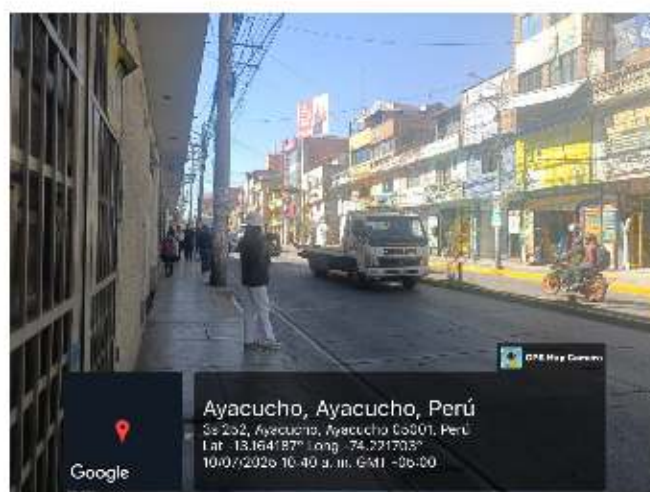


Figura 131: Recopilación de datos horario 10:40 am-C2E
Fuente: Imagen propia



Figura 132: Recopilación de datos horario 10:40 am-C2E
 Fuente: Imagen propia



Figura 133: Recopilación de datos horario 16:21 pm
 Fuente: Imagen propia



Figura 134: Recopilación de datos horario 16:15 pm
 Fuente: Imagen propia



Figura 135: Recopilación de datos horario 16:14 pm
 Fuente: Imagen propia



Figura 136: Recopilación de datos horario 16:13 pm
 Fuente: Imagen propia



Figura 137: Recopilación de datos horario 17:15 pm
 Fuente: Imagen propia



Figura 138: Recopilación de datos horario 17:18 pm
 Fuente: Imagen propia



Figura 139: Recopilación de datos horario 17:12 pm
 Fuente: Imagen propia



Figura 140: Recopilación de datos horario 17:15 pm
 Fuente: Imagen propia

CÁLCULO DE NUMERO DE REPETICIONES DE EJES EQUIVALENTES(W18)

AV. RAMON CASTILLA AYACUCHO

TIPO DE PAVIMENTO : **PAVIMENTO RÍGIDO**

1) **Ed (factor direccional) & Fc (factor carril)**

Fvp (factor de vehículo pesado)

Número de calzadas	Número de sentidos	Número de carriles por sentido	Factor Direccional (Fd)	Factor Carril (Fc)	Factor Pesados (Fvp) para cent de diseño
1 calzada (para 100% de la calzada)	1 sentido	1	1.00	1.00	1.00
	1 sentido	2	1.00	0.50	0.50
	1 sentido	3	1.00	0.33	0.33
	1 sentido	4	1.00	0.25	0.25
	2 sentidos	1	0.50	1.00	0.50
2 calzadas con separador central	2 sentidos	2	0.50	0.50	0.40
	2 sentidos	3	0.50	0.33	0.30
	2 sentidos	4	0.50	0.25	0.25
	2 sentidos	5	0.50	0.20	0.20
	2 sentidos	6	0.50	0.17	0.17

Factor de Calzadas y Sentidos: 0.50 y 0.50

Número de Calzadas : 2 Calzadas
 Número de Sentidos : 2 Sentidos
 Número de Carriles por Sentido : 2 carril

Fd : #N/D
 Fc : #N/D

NORMA



pesos / ejes } " Fvp "

2) **Fp : factor de presión**

Para el caso de afirmados y pavimentos rígidos el factor ajuste por presión de neumáticos será igual 1.0.

Nota en la Norma ASSTHO, los pesos son mayores, la Norma Peruana aumenta los pesos, pero el factor de presión se considera 1, y no un 10% ó 30% más.

3) **Determinar el FVL (Factor de Vehículo Ligero)**

VEHICULOS	IMAGEN	EJES	PESO/EJES	EES1	EES2	EETA1	EETA2	Σ FVP
AUTO (M1)			1.00	0.000436	-	-	-	0.001
			1.00	0.000436	-	-	-	
STATION WAGON (M1)			1.00	0.000436	-	-	-	0.001
			1.00	0.000436	-	-	-	
PICK UP (M2)			1.00	0.000436	-	-	-	0.008
			2.00	0.007483	-	-	-	
PANEL (M2)			1.00	0.000436	-	-	-	0.008
			2.00	0.007483	-	-	-	
COMBI RURAL (M3)			1.00	0.000436	-	-	-	0.040
			3.00	0.039452	-	-	-	
MICRO (M2M3)			1.00	0.000436	-	-	-	0.129
			4.00	-	0.128326	-	-	
C2			7.00	-	1.272834	-	-	4.61
			11.00	-	3.334826	-	-	

Tipo de Eje	Eje Equivalente (EE _{8.2m})
Eje Simple de ruedas simples (EE _{S1})	$EE_{S1} = [P / 6.6]^{4.1}$
Eje Simple de ruedas dobles (EE _{S2})	$EE_{S2} = [P / 8.2]^{4.1}$
Eje Tandem (1 eje ruedas dobles + 1 eje rueda simple) (EE _{TA1})	$EE_{TA1} = [P / 13.0]^{4.1}$
Eje Tandem (2 ejes de ruedas dobles) (EE _{TA2})	$EE_{TA2} = [P / 13.3]^{4.1}$
Ejes Tridem (2 ejes ruedas dobles + 1 eje rueda simple) (EE _{TR1})	$EE_{TR1} = [P / 16.6]^{4.0}$
Ejes Tridem (3 ejes de ruedas dobles) (EE _{TR2})	$EE_{TR2} = [P / 17.5]^{4.0}$

P = peso real por eje en toneladas
Fuente: Elaboración Propia, en base a correlaciones con los valores de las Tablas del apéndice D de la Guía AASHTO 93

4) Determinar el EE día carril (Eje equivalente por día y por carril).

TIPO	IMDA	F _d	F _c	F _{vp}	F _p	EE día Carril
AUTO (M1)	21600	0.5	0.8	0.001	1.000	7.541
STATION WAGON (M1)	1476	0.5	0.8	0.001	1.000	0.515
PICK UP (M2)	3160	0.5	0.8	0.008	1.000	10.010
PANEL (M2)	306	0.5	0.8	0.008	1.000	0.969
COMBI RURAL (M3)	234	0.5	0.8	0.040	1.000	3.734
MICRO	5680	0.5	0.8	0.129	1.000	292.549
C2	812	0.5	0.8	4.61	1.000	1496.568

$$EE_{\text{día-carril}} = IMDp_i \times F_d \times F_c \times F_{vp_i} \times F_{p_i}$$

5) Determinar el Fca (Factor de crecimiento acumulado).

FACTOR DE CRECIMIENTO (Fca)

$$F_{ca} = \frac{(1+r)^n - 1}{r}$$

r= Tasa Anual de crecimiento

n=Periodo de Diseño

r ligeros r pesados

r=	1.83	%	Vehículos Ligeros/Pasajeros
r=	2.90	%	Vehículos de Carga (PBI)
n=	20.00		Periodo de diseño de un afirmado
Fca=	10.87		Vehículos Ligeros/Pasajeros
Fca=	11.41		Vehículos de carga

0.0183 0.028979668

0.20

0.33

6) Número de Repeticiones de ejes equivalentes

$$N_{rep \text{ de } EE_{8.2m}} = \sum [EE_{\text{día-carril}} \times Fca \times 365]$$

TIPO	EE día Carril	FCA	Días del Año	Sub Total
AUTO (M1)	7.541	10.87	365	2.99E+04
STATION WAGON (M1)	0.515	10.87	365	2.04E+03
PICK UP (M2)	10.010	10.87	365	3.97E+04
PANEL (M2)	0.969	10.87	365	3.84E+03
COMBI RURAL (M3)	3.734	10.87	365	1.48E+04
MICRO	292.549	10.87	365	1.16E+06
C2	1496.568	11.41	365	6.23E+06
Σ				7.48E+06

Nrep EE o W18 o ESALS = 7,483,311.30

Tipo de tráfico: TP08

MANUAL DE USUARIO PROGRAMA

1 Inicio del Sistema

Fig. 1 Interfaz principal al iniciar el sistema



La Figura 1 muestra la interfaz principal al momento del inicio del sistema. La ventana presenta un diseño adaptativo que se ajusta automáticamente a la resolución de pantalla detectada (2560x1440 en este caso). El sistema está organizado en tres paneles principales numerados del 1 al 4:

Panel 1 - Configuración y Videos (izquierda): Contiene el estado del modelo YOLO, configuración de exportación y asignación de videos por día de la semana.

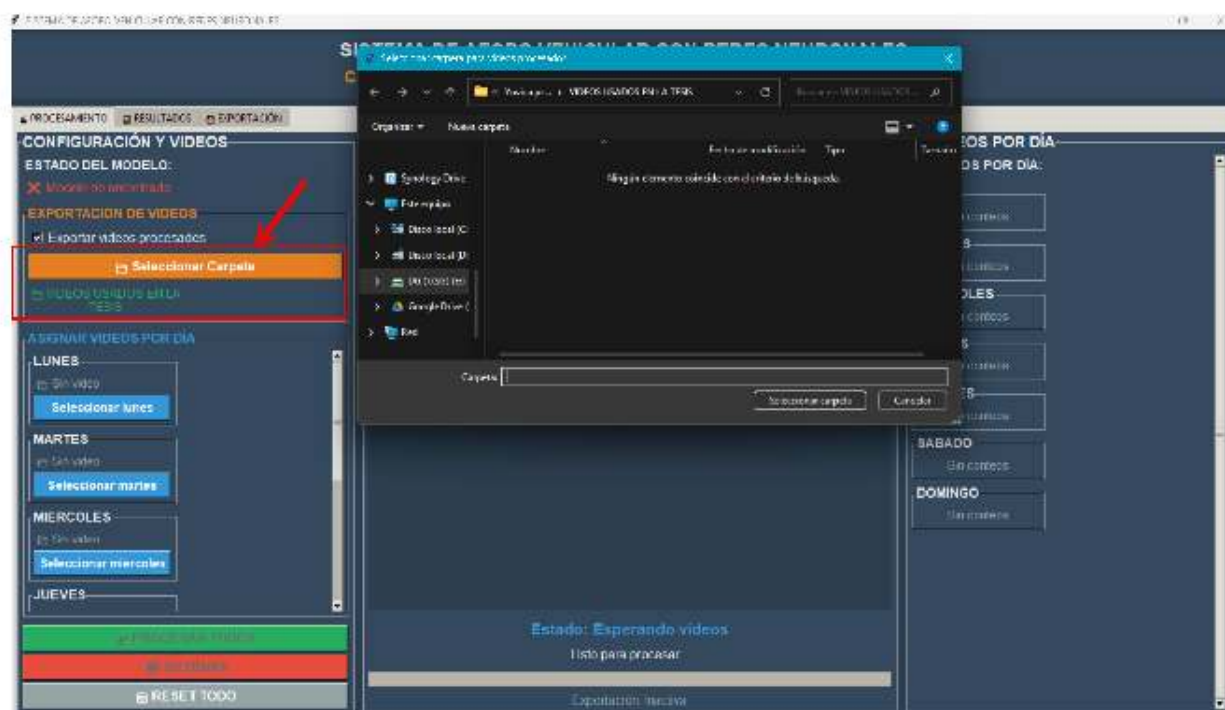
Panel 3 - Procesamiento en Tiempo Real (centro): Área donde se visualizará el video durante el procesamiento con detecciones en tiempo real.

Panel 4 - Conteos por Día (derecha): Muestra los resultados de conteo vehicular para cada día procesado.

Procedimiento inicial: Al iniciar el sistema, TRAFFICORE AI carga automáticamente el modelo de redes neuronales YOLO y detecta la resolución de pantalla para optimizar la interfaz.

1.1 Configuración de Importación

Fig. 2 Cuadro de diálogo para selección de carpeta de importación



La Figura 2 ilustra el cuadro de diálogo de Windows que permite seleccionar la carpeta de destino para la importación automática de videos procesados. El sistema navega a través de las unidades disponibles para ubicar la carpeta deseada.

Procedimiento de configuración:

1. Hacer clic en "Seleccionar Carpeta" en el panel de importación
2. Navegar hasta la carpeta deseada usando el explorador
3. Hacer clic en "Seleccionar carpeta" para confirmar
4. El sistema almacenará la ruta seleccionada para importaciones futuras

1.2 Organización de la Interfaz

Fig. 3 Vista general de la organización por paneles con numeración identificativa



La Figura 3 presenta la organización completa de la interfaz mediante tres pestañas principales (PROCESAMIENTO, RESULTADOS, EXPORTACIÓN) y cuatro paneles numerados que facilitan la navegación del usuario:

Pestaña de Procesamiento: Contiene todas las herramientas necesarias para la configuración y procesamiento de videos.

Distribución de paneles:

- **Panel 1:** Configuración y asignación de videos
- **Panel 2:** Botones de control (Procesar, Detener, Reset)
- **Panel 3:** Visualización de video en tiempo real
- **Panel 4:** Conteos y resultados por día

Procedimiento de navegación: El usuario puede alternar entre pestañas para acceder a diferentes funcionalidades sin perder el estado de configuración actual.

1.3 Panel de Configuración de Videos

Fig. 4 Panel de configuración detallado mostrando estado del modelo y opciones de exportación



La Figura 4 muestra el panel de configuración que incluye:

Estado del Modelo: Indica si el modelo YOLO ha sido cargado correctamente. El estado "Modelo no encontrado" requiere verificar la presencia del archivo del modelo en la carpeta del programa.

Exportación de Videos: Checkbox para habilitar/deshabilitar la exportación automática de videos procesados con detecciones superpuestas.

Asignación de Videos por Día: Lista de los siete días de la semana con botones individuales para seleccionar el video correspondiente a cada día.

Precauciones importantes:

- Verificar que el archivo del modelo YOLO esté presente antes de procesar
- Confirmar que hay espacio suficiente si se habilita la exportación
- Seleccionar videos en formatos compatibles (MP4, AVI, MOV, MKV, WMV)

1.4 Estado del Modelo YOLO

Fig. 5 Estado del modelo YOLO cargado exitosamente mostrando tipos de vehículos detectables



La Figura 5 presenta el estado exitoso de carga del modelo YOLO. El mensaje indica "Modelo cargado: 2S2, PA, C2E, 3S2, B3E..." mostrando algunos de los tipos de vehículos que el sistema puede detectar y clasificar.

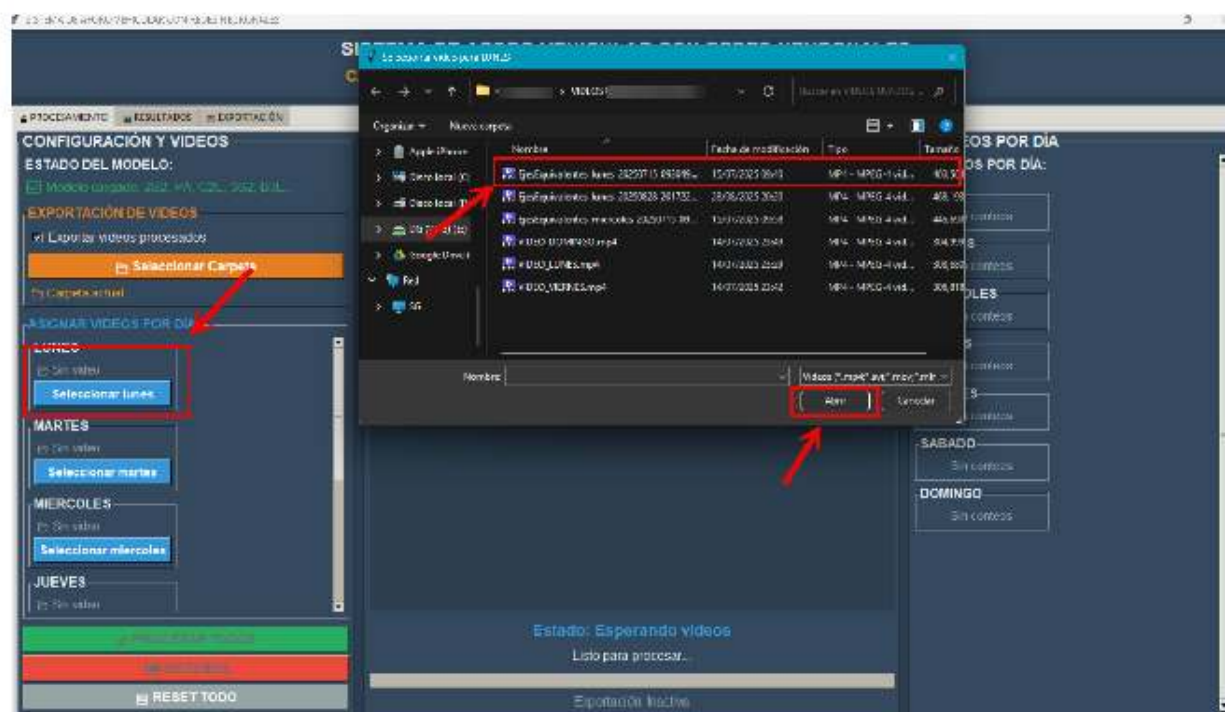
Tipos de vehículos soportados:

- **Vehículos ligeros:** A (automóvil), SW (station wagon), PU (pickup), PA (panel), RC (rural combi), M (micro)
- **Buses:** B2E, B3E, B4E (buses de 2, 3 y 4 ejes)
- **Camiones:** C2E, C3E (camiones de 2 y 3 ejes)
- **Semirremolques:** 2S1, 2S2, 2S3, 3S1, 3S2, 3S3
- **Tándem:** 2T2, 2T3, 3T2, 3T3

Procedimiento de verificación: El sistema carga automáticamente el modelo al iniciar y muestra el estado en tiempo real.

1.5 Selección de Videos para Cada Día

Fig. 6 Cuadro de diálogo para selección de video específico por día



La Figura 6 muestra el proceso de selección de video para un día específico (LUNES en este ejemplo). El cuadro de diálogo presenta varios archivos de video con nombres descriptivos que incluyen fechas y tipos de contenido.

Nomenclatura recomendada para videos:

- Incluir el día de la semana en el nombre
- Agregar fecha en formato AAAAMMDD
- Especificar horario de grabación si es relevante
- Usar formatos estándar (MP4, AVI, MOV, MKV, WMV)

Procedimiento de selección:

1. Hacer clic en "Seleccionar [día]" en el panel izquierdo
2. Navegar hasta la ubicación de los videos
3. Seleccionar el video correspondiente al día específico
4. Hacer clic en "Abrir" para confirmar la selección

1.6 Estado Visual de Asignaciones

Fig. 7 Panel mostrando video asignado exitosamente para el día lunes



La Figura 7 ilustra el cambio visual que ocurre después de asignar exitosamente un video a un día específico. El panel de LUNES ahora muestra:

Indicadores de estado:

- Icono de video, indicando asignación exitosa
- Color verde del texto confirmando la asignación
- Habilitación del botón "PROCESAR TODOS" (resaltado en verde)

Validaciones automáticas:

- Verificación de formato de archivo compatible
- Confirmación de legibilidad del video
- Habilitación condicional de controles de procesamiento

2 Procesamiento de Videos

Fig. 8 Sistema durante el procesamiento activo del video del lunes



La Figura 8 captura el momento activo de procesamiento de video. Los elementos destacados incluyen:
Panel central: Muestra el video en tiempo real con detecciones superpuestas de vehículos.

Información de progreso:

- "Procesando: LUNES" en la parte superior
- "Frame 2931/3630 (80.7"
- Barra de progreso visual en la parte inferior
- "Exportando lunes" confirmando la generación del video procesado

Controles activos:

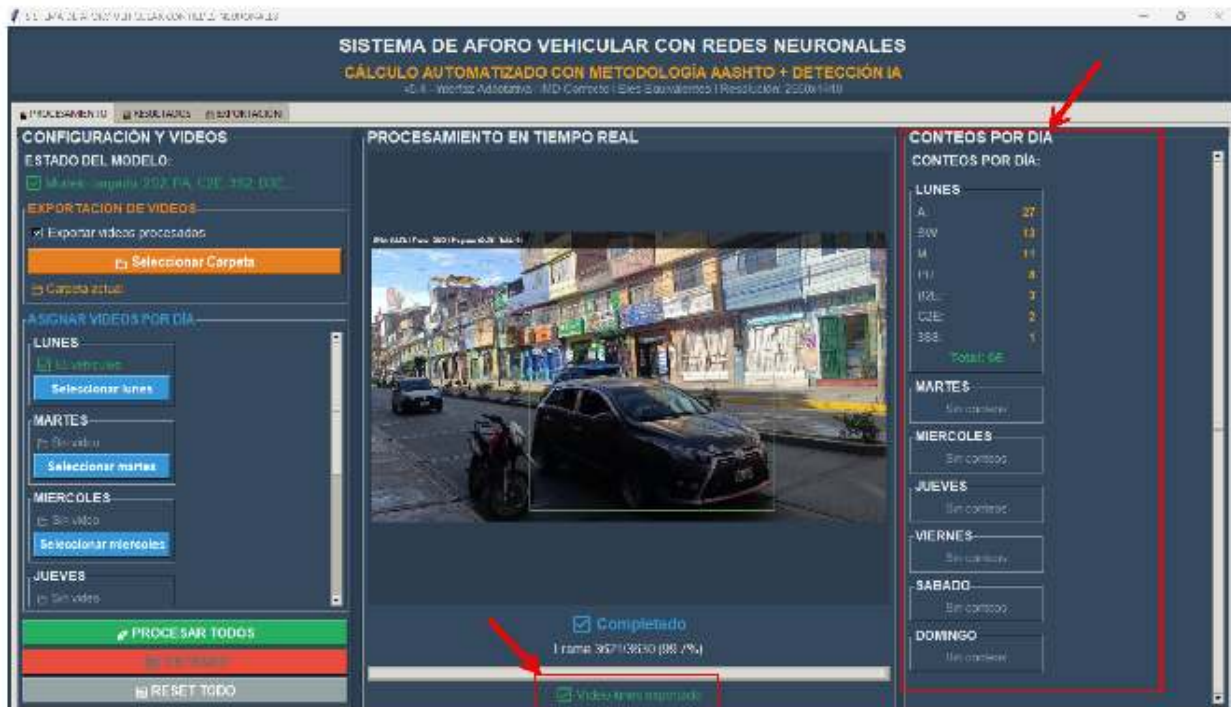
- Botón "DETENER" habilitado (rojo) para interrumpir el procesamiento
- Botón "PROCESAR TODOS" deshabilitado durante la ejecución

Precauciones durante el procesamiento:

- No cerrar la aplicación durante el procesamiento activo
- Asegurar suficiente espacio en disco para videos exportados
- Monitorear el progreso para detectar posibles errores

2.1 Detecciones y Tracking de Vehículos

Fig. 9 Visualización de detecciones vehiculares con tracking en tiempo real



La Figura 9 presenta el núcleo funcional del sistema: la detección y seguimiento de vehículos en tiempo real. La imagen muestra:

Elementos visuales del tracking:

- Bounding boxes (rectángulos) alrededor de vehículos detectados
- Etiquetas de identificación única por vehículo
- Clasificación automática del tipo de vehículo
- Conteo en tiempo real en el panel derecho

Panel de conteos actualizado: Muestra el conteo acumulativo por tipo de vehículo:

- A: 27 (automóviles)
- SW: 13 (station wagons)
- M: 11 (micros)
- PU: 8 (pickups)
- B2E: 3 (buses de 2 ejes)
- C2E: 2 (camiones de 2 ejes)
- 3S3: 1 (semirremolque de 3 ejes)
- Total: 65 vehículos detectados

Estado de finalización: "Completado" y "Video lunes exportado" confirman la finalización exitosa del procesamiento.

2.2 Pestaña de Resultados - Configuración

Fig. 10 Panel de configuración de parámetros para cálculo de ejes equivalentes



La Figura 10 muestra la pestaña de RESULTADOS con el panel de configuración de parámetros AASHTO. Los parámetros configurables incluyen:

Configuración AASHTO:

- **Departamento:** ayacucho (afecta las tasas de crecimiento vehicular)
- **Mes:** junio (influye en los factores de corrección estacional)
- **Tipo de Pavimento:** rígido (determina las fórmulas de factor de vehículo pesado)
- **Período de Diseño:** 20 años (para cálculo del factor de crecimiento acumulado)

Configuración de Carretera:

- **Calzadas:** 1
- **Sentidos:** 2
- **Carriles por sentido:** 1

Instrucciones del sistema: El panel derecho muestra los pasos a seguir: 1. Procese los videos de la semana 2. Configure los parámetros 3. Calcule IMD + Ejes Equivalentes

2.3 Cálculo de IMD

Fig. 11 Resultados del cálculo de IMD mostrando valores por tipo de vehículo

La Figura ?? presenta los resultados del cálculo de Índice Medio Diario (IMD) siguiendo la metodología AASHTO. El panel "IMD CALCULADO (Suma 7 días / 7)" muestra:

Valores de IMD por tipo de vehículo:

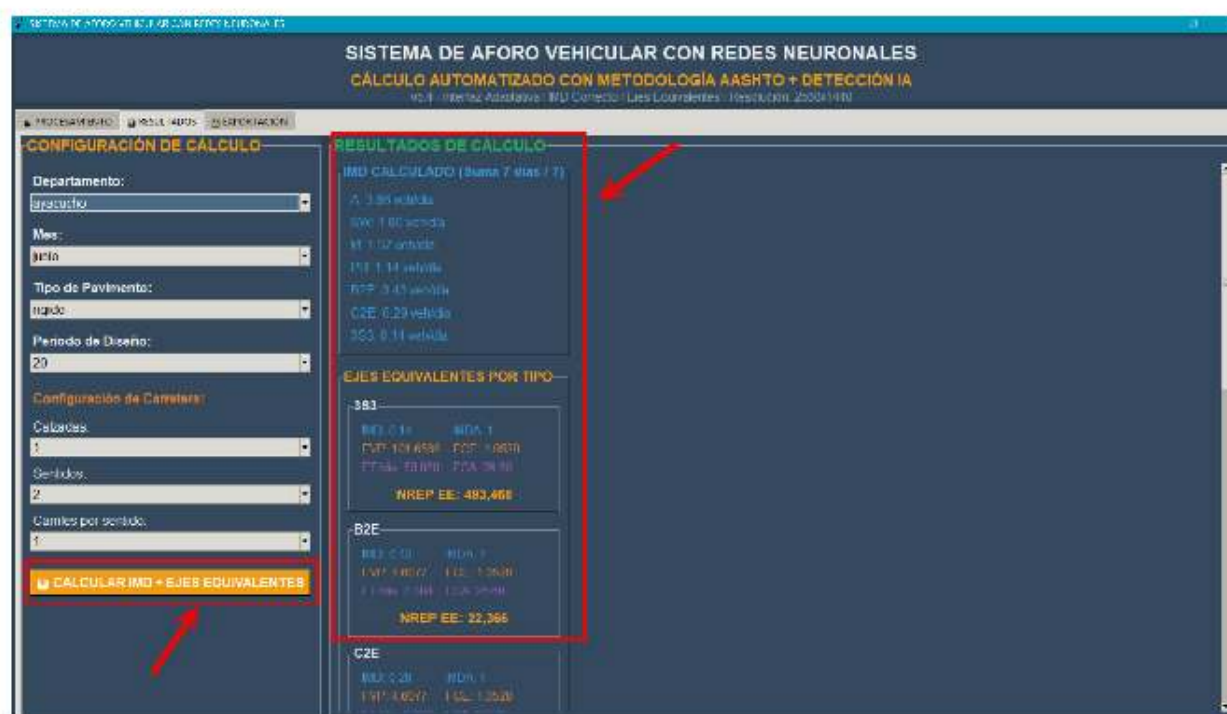
- A: 3.86 veh/día (automóviles)
- SW: 1.86 veh/día (station wagons)
- M: 1.57 veh/día (micros)
- PU: 1.14 veh/día (pickups)
- B2E: 0.43 veh/día (buses de 2 ejes)
- C2E: 0.29 veh/día (camiones de 2 ejes)
- 3S3: 0.14 veh/día (semirremolques)

Metodología de cálculo: $IMD = \sum (\text{conteos-tipo-7-días}) / 7$, aplicando la fórmula estándar donde se suman todos los conteos de cada tipo de vehículo durante los siete días y se divide entre siete para obtener el promedio diario.

3 Resultados

3.1 Resultados de Ejes Equivalentes por Tipo

Fig. 12 Cálculos detallados de ejes equivalentes para vehículos tipo 3S3 y B2E



La Figura 12 muestra los cálculos detallados de ejes equivalentes para dos tipos de vehículos pesados:

Vehículo 3S3 (Semirremolque):

- IMD: 0.14, IMDA: 1
- FVP: 101.6566 (Factor de Vehículo Pesado alto debido a configuración de ejes)
- FCE: 1.0528 (Factor de Corrección Estacional)
- EE/día: 50.828 (Ejes Equivalentes por día)
- FCA: 26.60 (Factor de Crecimiento Acumulado para 20 años)
- **NREP EE: 493,468** (Número total de repeticiones de ejes equivalentes)

Vehículo B2E (Bus de 2 ejes):

- IMD: 0.43, IMDA: 1
- FVP: 4.6077 (Factor menor que el semirremolque)
- FCE: 1.0528, FCA: 26.60
- EE/día: 2.304
- **NREP EE: 22,366**

Interpretación técnica: Los vehículos más pesados con mayor número de ejes generan significativamente más daño al pavimento, como se refleja en los valores de NREP EE.

3.2 Resultados de Ejes Equivalentes Adicionales

Fig. 13 Continuación de cálculos para vehículos ligeros (C2E, M, A, SW, PU)

Vehicle Type	IMD	IMDA	FVP	FCE	EE/día	FCA	NREP EE
C2E (Camión 2 ejes)	0.29	1	4.6077	1.0528	2.304	26.60	22,366
M (Micro)	0.29	1	0.0010	1.0775	0.000	26.60	1,122
A (Automóvil)	0.29	1	0.0010	1.0775	0.000	26.60	21
SW (Station Wagon)	0.29	1	0.0010	1.0775	0.000	26.60	13
PU (Pickup)	0.29	1	0.0010	1.0775	0.000	26.60	8

La Figura 13 continúa mostrando los cálculos para vehículos ligeros y medianos:

Vehículo C2E (Camión 2 ejes):

- IMD: 0.29, IMDA: 1
- FVP: 4.6077, FCE: 1.0528
- EE/día: 2.304, FCA: 26.60
- NREP EE: 22,366

Vehículos ligeros (M, A, SW, PU):

- M (Micro): NREP EE: 1,122
- A (Automóvil): NREP EE: 21
- SW (Station Wagon): NREP EE: 13
- PU (Pickup): NREP EE: 8

Observación importante: Los vehículos ligeros tienen factores FVP muy bajos (0.0010 para A, SW, PU), resultando en contribuciones mínimas a los ejes equivalentes de diseño.

3.3 Resumen Final y Configuración

Fig. 14 Resultado final total y configuración utilizada en el cálculo

SISTEMA DE AFORO VEHICULAR CON REDES NEURONALES
CÁLCULO AUTOMATIZADO CON METODOLOGÍA AASHTO + DETECCIÓN IA
 v5.4 - Interfaz Adaptativa | IMD: Correcto | Ejes Equivalentes | Resolución: 2560x1440

CONFIGURACIÓN DE CÁLCULO

Departamento:

Mes:

Tipo de Pavimento:

Período de Diseño:

Configuración de Carreteras:

Calzadas:

Seriales:

Carriles por periodo:

CALCULAR IND - EJES EQUIVALENTES

RESULTADOS DE CÁLCULO

SW

MD: 1.85 | FDE: 1
 PVP: 5.0000 | FCF: 1.0773
 ESESA: 0.002 | FCA: 23.83
NREP EE: 13

PU

MD: 1.14 | FDE: 2
 PVP: 5.0000 | FCF: 1.0773
 ESESA: 0.001 | FCA: 23.83
NREP EE: 8

TOTAL NREP EE (DISEÑO): 539,368

CONFIGURACIÓN UTILIZADA

Departamento: ayacucho
 Mes: junio
 Pavimento: rígido
 Período: 20 años
 Videos procesados: 1/7
 FD: 0.50 | FC: 1.00

La Figura 14 presenta el resultado final del cálculo de ejes equivalentes:

TOTAL NREP EE (DISEÑO): 539,368

Este valor representa el número total de repeticiones de ejes equivalentes que el pavimento debe soportar durante el período de diseño de 20 años.

Configuración utilizada:

- Departamento: ayacucho
- Mes: junio
- Pavimento: rígido
- Período: 20 años
- Videos procesados: 1/7
- FD: 0.50 | FC: 1.00 (factores direccional y de carril)

Significado ingenieril: Este valor de 539,368 ejes equivalentes sirve como parámetro fundamental para el diseño estructural del pavimento, determinando el espesor y especificaciones técnicas necesarias.

4 Exportación

4.1 Pestaña de Exportación

Fig. 15 Panel de exportación con botones organizados para diferentes tipos de reportes



La Figura 15 muestra la pestaña de EXPORTACIÓN con seis opciones principales organizadas en una cuadrícula:

Opciones de exportación disponibles:

1. **REPORTE IMD** (azul): Genera reporte de texto con cálculos de IMD
2. **REPORTE COMPLETO** (morado): Exporta archivo JSON con todos los datos
3. **ABRIR CARPETA** (gris): Accede a la carpeta de trabajo actual
4. **CSV EJES EQUIVALENTES** (naranja): Exporta datos tabulares para análisis
5. **ABRIR VIDEOS** (verde): Accede a videos procesados con detecciones
6. **EXPORTAR TODO** (rojo): Genera todos los reportes simultáneamente

Procedimiento de exportación: Hacer clic en cualquier botón para generar el tipo específico de reporte o utilizar "EXPORTAR TODO" para una exportación completa.

4.2 Estructura de Reportes TXT

Fig. 16 Ejemplo de reporte IMD en formato de texto plano

```

REPORTE IMD - METODOLOGÍA AASHTO
Sistema de Ejes Equivalentes v5.4 - Interfaz Adaptativa

Fecha: 2025-09-16 10:52:06
Resolución: 2560x1440

TMD CALCULADO (suma 7 días / 7):
-----
A: 27 : 7 = 3.86 veh/día
SW: 13 : 7 = 1.86 veh/día
M: 11 : 7 = 1.57 veh/día
PU: 8 : 7 = 1.14 veh/día
B2E: 3 : 7 = 0.43 veh/día
C2E: 2 : 7 = 0.29 veh/día
3S3: 1 : 7 = 0.14 veh/día

TOTAL TMD: 9.29 veh/día

```

La Figura 16 muestra la estructura del reporte IMD exportado en formato de texto. El reporte incluye:

Encabezado del reporte:

- Título: "REPORTE IMD - METODOLOGÍA AASHTO"
- Versión del sistema: "Sistema de Ejes Equivalentes v5.4 - Interfaz Adaptativa"
- Fecha y hora de generación
- Resolución de pantalla utilizada

Cálculos de IMD detallados:

- A: $27 \div 7 = 3.86$ veh/día
- SW: $13 \div 7 = 1.86$ veh/día
- M: $11 \div 7 = 1.57$ veh/día
- PU: $8 \div 7 = 1.14$ veh/día
- B2E: $3 \div 7 = 0.43$ veh/día
- C2E: $2 \div 7 = 0.29$ veh/día
- 3S3: $1 \div 7 = 0.14$ veh/día

TOTAL IMD: 9.29 veh/día

Este formato facilita la documentación técnica y la inclusión en informes de ingeniería.

4.3 Formato de Archivos CSV

Fig. 17 Estructura del archivo CSV con datos de ejes equivalentes

```
Tipo_Vehiculo,IMD,FCE,IMDA,FD,FC,FVP,FP,EE_Dia_Carril,FCA,NREP_EE,Clasificacion
3S3,0.14,1.0528,1,0.50,1.00,101.656593,1.0,50.8283,26.60,493468,pesados
A,3.86,1.0772,5,0.50,1.00,0.001000,1.0,0.0025,23.89,21,ligeros
B2E,0.41,1.0528,1,0.50,1.00,4.607660,1.0,2.3038,26.60,22366,pesados
C2E,0.29,1.0528,1,0.50,1.00,4.607660,1.0,2.3038,26.60,22366,pesados
M,1.57,1.0772,2,0.50,1.00,0.128763,1.0,0.1288,23.89,1122,ligeros
PU,1.14,1.0772,2,0.50,1.00,0.001000,1.0,0.0010,23.89,8,ligeros
SW,1.86,1.0772,3,0.50,1.00,0.001000,1.0,0.0015,23.89,13,ligeros
```

La Figura 17 presenta la estructura del archivo CSV exportado con columnas técnicas específicas:

Columnas del archivo CSV:

- **Tipo_Vehiculo:** Clasificación del vehículo (3S3, A, B2E, etc.)
- **IMD:** Índice Medio Diario calculado
- **FCE:** Factor de Corrección Estacional
- **IMDA:** Índice Medio Diario Anual
- **FD, FC:** Factores direccional y de carril
- **FVP:** Factor de Vehículo Pesado
- **FP:** Factor de presión de inflado
- **EE_Dia_Carril:** Ejes equivalentes por día-carril
- **FCA:** Factor de Crecimiento Acumulado
- **NREP_EE:** Número de repeticiones de ejes equivalentes
- **Clasificacion:** Categoría (ligeros/pesados)

Utilidad del CSV: Permite análisis estadístico avanzado, importación a software especializado y generación de gráficos técnicos.

4.4 Contenido de Reportes JSON

Fig. 18 Estructura del archivo JSON con datos completos del sistema

```

reporte_completo_20250910_105206.json > ...
{
  "timestamp": "2025-09-10T10:52:06.248574",
  "version": "Sistema Ejes Equivalentes v5.4 - Interfaz Adaptativa",
  "resolucion_pantalla": "2560x1440",
  "conteos_por_dia": {
    "lunes": {
      "PU": 8,
      "SW": 13,
      "M": 11,
      "A": 27,
      "MDI": 7,
      "CPI": 2,
      "SSJ": 1
    },
    "martes": {},
    "miercoles": {},
    "jueves": {},
    "viernes": {},
    "sabado": {},
    "domingo": {}
  },
  "videos_asignados": {
    "lunes": "VIDEO_LUNES.mpa",
    "martes": null,
    "miercoles": null,
    "jueves": null,
    "viernes": null,
    "sabado": null,
    "domingo": null
  },
  "imd_calculado": {
    "A": 3.857142857142857,
    "MDI": 0.42857142857142855,
    "M": 1.5714285714285714,
    "SSJ": 0.14285714285714285,
    "SW": 1.8571428571428572,
    "PU": 1.1428571428571428,
    "CPI": 0.2857142857142857
  },
  "ejes_equivalentes": {
    "resultados_por_tipo": {

```

La Figura 18 ilustra la estructura del reporte JSON completo que incluye:

Metadatos del sistema:

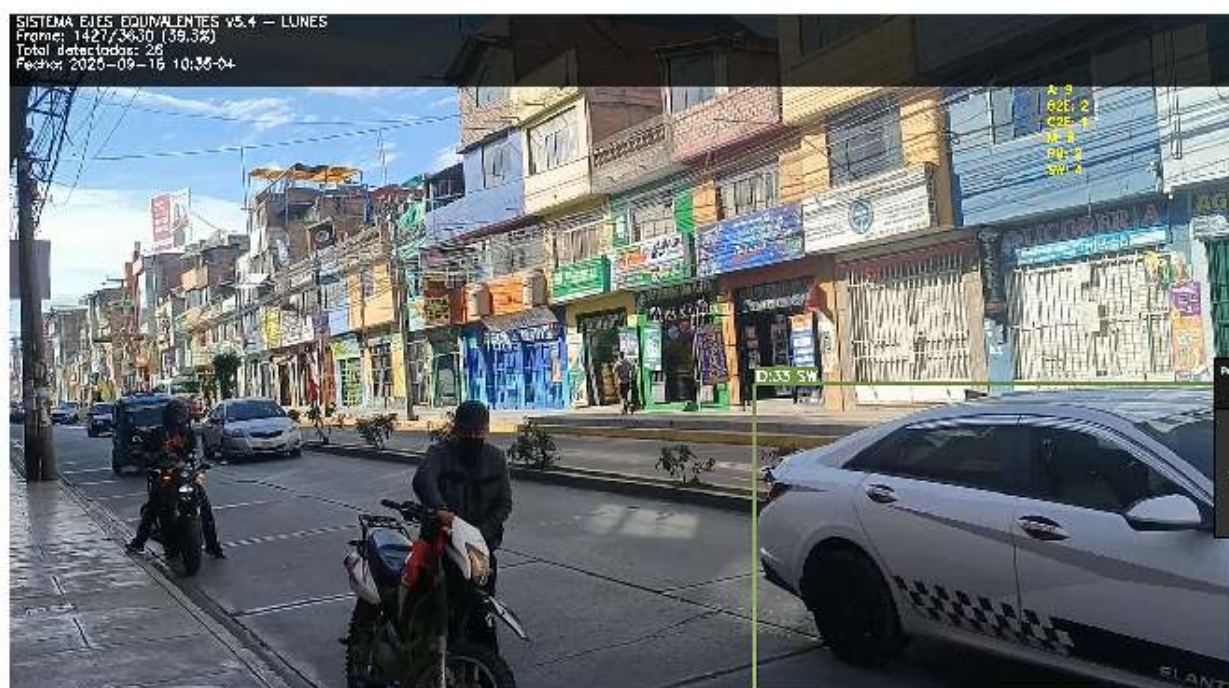
- **timestamp:** Fecha y hora ISO de generación
- **version:** "Sistema Ejes Equivalentes v5.4 - Interfaz Adaptativa"
- **resolucion_pantalla:** Resolución utilizada durante el análisis

Datos estructurados:

- **conteos_por_dia:** Conteos detallados por cada día de la semana
- **videos_asignados:** Lista de archivos de video utilizados
- **imd_calculado:** Valores de IMD con alta precisión decimal
- **ejes_equivalentes:** Cálculos completos de factores AASHTO

Ventajas del formato JSON:

- Estructura jerárquica preservando relaciones entre datos
- Fácil importación a sistemas de análisis
- Compatibilidad con bases de datos NoSQL
- Respaldo completo de la sesión de trabajo

4.5 Videos Procesados con Detecciones**Fig. 19** Frame de video exportado con detecciones y tracking superpuesto

La Figura 19 muestra un frame del video exportado con todas las detecciones y elementos de tracking superpuestos:

Elementos superpuestos en el video:

- **Encabezado informativo:** "SISTEMA EJES EQUIVALENTES v5.4 - LUNES"
- **Progreso del procesamiento:** "Frame: 1427/3630 (39.3)"
- **Estadísticas de detección:** "Total detectados: 26"
- **Timestamp:** Fecha y hora de procesamiento

Detecciones vehiculares:

- Bounding boxes de colores únicos por tipo de vehículo
- Etiquetas con ID único y clasificación (ej: "ID:33 SW")
- Conteo por tipo en tiempo real en la esquina superior derecha

Información técnica del tracking:

- Sistema de seguimiento multi-objeto manteniendo consistencia de IDs

- Algoritmos de filtrado temporal para reducir falsos positivos
- Tracking robusto que mantiene identificación a través de oclusiones parciales

Aplicaciones del video procesado:

- Verificación manual de resultados de detección
- Presentación de resultados a stakeholders
- Documentación visual del proceso de análisis
- Validación de algoritmos de tracking en condiciones reales

5 Precauciones y Recomendaciones Generales

Preparación de videos:

- Resolución mínima recomendada: 720p
- Formato preferido: MP4 con codec H.264
- Duración óptima: 1-4 horas por video
- Ángulo de cámara: Vista frontal o lateral del flujo vehicular

Resolución de problemas comunes:

- **Modelo no encontrado:** Verificar la presencia de archivos .pt en la carpeta del programa
- **Error de memoria:** Reducir la resolución de videos o procesar secuencialmente
- **Detecciones incorrectas:** Ajustar umbrales de confianza en la configuración
- **Videos no compatibles:** Convertir a formato MP4 con codificación estándar

Interpretación de resultados:

- NREP EE elevado indica necesidad de pavimentos más robustos
- Variaciones estacionales significativas pueden requerir factores de corrección adicionales
- Composición vehicular atípica debe documentarse en el informe técnico
- Resultados deben contextualizarse con normativas locales de diseño vial

5.1 Flujo de Trabajo Recomendado

Fase 1: Preparación

1. Verificar instalación completa del software y dependencias
2. Confirmar disponibilidad de modelos YOLO entrenados
3. Organizar videos por día de la semana con nomenclatura consistente
4. Configurar carpeta de exportación con suficiente espacio disponible

Fase 2: Configuración

1. Seleccionar parámetros AASHTO apropiados para el proyecto
2. Asignar videos correspondientes a cada día de la semana
3. Verificar configuración de carretera (calzadas, sentidos, carriles)

4. Habilitar exportación de videos procesados si se requiere documentación visual

Fase 3: Procesamiento

1. Iniciar procesamiento secuencial con "PROCESAR TODOS"
2. Monitorear progreso y calidad de detecciones en tiempo real
3. Revisar conteos por día conforme se completa cada procesamiento
4. Verificar exportación exitosa de videos con detecciones superpuestas

Fase 4: Análisis

1. Acceder a la pestaña RESULTADOS y calcular IMD + Ejes Equivalentes
2. Revisar valores calculados para detectar inconsistencias o valores atípicos
3. Analizar distribución de tipos vehiculares y su contribución al NREP EE
4. Validar que la configuración utilizada sea apropiada para el proyecto

Fase 5: Exportación y Documentación

1. Exportar reportes en todos los formatos requeridos (TXT, CSV, JSON)
2. Generar documentación técnica incluyendo videos procesados como evidencia
3. Archivar archivos de configuración para reproducibilidad futura
4. Preparar presentación de resultados con visualizaciones apropiadas

~\Desktop\vision_artificial\YUVICA\SOFTWARE_EJES EQUIVALANTES.py

```

1 import os
2 import sys
3
4 # =====
5 # CONFIGURACIÓN CRÍTICA: DEBE ESTAR ANTES DE CUALQUIER IMPORT DE TORCH/YOLO
6 # =====
7
8 # Deshabilitar CUDA completamente a nivel de sistema
9 os.environ['CUDA_VISIBLE_DEVICES'] = '-1' # -1 significa "sin GPU"
10 os.environ['CUDA_LAUNCH_BLOCKING'] = '1'
11 os.environ['TORCH_USE_CUDA_DSA'] = '0'
12 os.environ['FORCE_CUDA'] = '0'
13
14 # Variables adicionales para asegurar CPU
15 os.environ['OMP_NUM_THREADS'] = str(os.cpu_count() or 4)
16 os.environ['YOLO_VERBOSE'] = 'False'
17
18 # Forzar torch a usar CPU antes de que se importe
19 import warnings
20 warnings.filterwarnings('ignore', category=UserWarning)
21
22 try:
23     import torch
24     # Sobrescribir la función que detecta CUDA
25     torch.cuda.is_available = lambda: False
26     torch.cuda.device_count = lambda: 0
27     torch.cuda.current_device = lambda: None
28     torch.backends.cudnn.enabled = False
29     print(" PyTorch configurado para CPU SOLAMENTE")
30 except ImportError:
31     print(" PyTorch no disponible, continuando sin él")
32 except Exception as e:
33     print(f" Error configurando PyTorch: {e}")
34
35 # =====
36 # AHORA SÍ IMPORTAR EL RESTO
37 # =====
38
39 import tkinter as tk
40 from tkinter import ttk, filedialog, messagebox, scrolledtext
41 import cv2
42 from ultralytics import YOLO
43 import numpy as np
44 import json
45 from datetime import datetime
46 import csv
47 from collections import defaultdict, deque
48 import math
49 import threading
50 from PIL import Image, ImageTk
51 import time

```

```

52
53 print("="*80)
54 print(" SISTEMA EJES EQUIVALENTES v6.0 - TODAS LAS ESTACIONES Y DEPARTAMENTOS")
55 print("="*80)
56 print(f" CPUs disponibles: {os.cpu_count()}")
57 print(f" CUDA deshabilitado: CUDA_VISIBLE_DEVICES =
{os.environ.get('CUDA_VISIBLE_DEVICES')}")
58 print("="*80)
59
60 class CalculadorEjesEquivalentes:
61     """Calculadora completa de Ejes Equivalentes según metodología AASHTO"""
62     def __init__(self):
63         # =====
64         # TABLAS DE FACTORES DE CORRECCIÓN ESTACIONAL (FCE) - TODAS LAS ESTACIONES
65         # =====
66
67         # FACTORES PARA VEHÍCULOS LIGEROS - 78 ESTACIONES
68         self.fce_ligeros = {
69             "AGUAS CALIENTES": {
70                 "enero": 0.9362, "febrero": 0.8626, "marzo": 1.0850, "abril": 1.0925,
71                 "mayo": 1.1626, "junio": 1.1758, "julio": 0.9508, "agosto": 0.8846,
72                 "septiembre": 0.9455, "octubre": 1.0418, "noviembre": 1.0414,
73                 "diciembre": 0.9899
74             },
75             "AGUAS CLARAS": {
76                 "enero": 1.0020, "febrero": 1.0567, "marzo": 1.1036, "abril": 1.0592,
77                 "mayo": 1.0095, "junio": 1.0030, "julio": 0.8943, "agosto": 0.9075,
78                 "septiembre": 1.0128, "octubre": 1.0209, "noviembre": 1.0749,
79                 "diciembre": 0.9283
80             },
81             "AMBO": {
82                 "enero": 0.9780, "febrero": 1.0546, "marzo": 1.1877, "abril": 1.2270,
83                 "mayo": 1.0179, "junio": 1.0278, "julio": 0.9661, "agosto": 0.9541,
84                 "septiembre": 1.0045, "octubre": 0.9693, "noviembre": 0.9846,
85                 "diciembre": 0.8615
86             },
87             "ATICO": {
88                 "enero": 0.8554, "febrero": 0.7148, "marzo": 1.0360, "abril": 1.0139,
89                 "mayo": 1.1484, "junio": 1.1781, "julio": 0.9965, "agosto": 0.9877,
90                 "septiembre": 1.1394, "octubre": 1.0734, "noviembre": 1.1453,
91                 "diciembre": 0.8831
92             },
93             "AYAVIRI": {
94                 "enero": 0.9722, "febrero": 0.9244, "marzo": 1.0735, "abril": 1.0773,
95                 "mayo": 1.1098, "junio": 1.1035, "julio": 0.9469, "agosto": 0.9272,
96                 "septiembre": 0.9376, "octubre": 1.0491, "noviembre": 1.0215,
97                 "diciembre": 0.9603
98             },
99             "CAMANA": {
100                 "enero": 0.5958, "febrero": 0.4991, "marzo": 1.0434, "abril": 1.2433,
101                 "mayo": 1.3825, "junio": 1.4071, "julio": 1.2562, "agosto": 1.2200,
102                 "septiembre": 1.3018, "octubre": 1.2559, "noviembre": 1.2199,
103                 "diciembre": 0.9724
104             },

```



```

99         "CANCAS": {
100             "enero": 0.8619, "febrero": 0.8638, "marzo": 1.0755, "abril": 1.1156,
101             "mayo": 1.1769, "junio": 1.2257, "julio": 0.9733, "agosto": 0.9040,
102             "septiembre": 1.0565, "octubre": 1.0323, "noviembre": 1.0284,
103             "diciembre": 0.8897
104         },
105         "CARACOTO": {
106             "enero": 1.0517, "febrero": 0.9815, "marzo": 1.0978, "abril": 1.0548,
107             "mayo": 1.0540, "junio": 1.0469, "julio": 0.9914, "agosto": 0.8786,
108             "septiembre": 0.9962, "octubre": 0.9920, "noviembre": 0.9977,
109             "diciembre": 0.8907
110         },
111         "CASARACRA": {
112             "enero": 1.1057, "febrero": 1.1747, "marzo": 1.2363, "abril": 1.0139,
113             "mayo": 0.9312, "junio": 1.0287, "julio": 0.8415, "agosto": 0.8825,
114             "septiembre": 1.0603, "octubre": 0.9930, "noviembre": 1.1250,
115             "diciembre": 0.9775
116         },
117         "CATAC": {
118             "enero": 1.0956, "febrero": 1.0575, "marzo": 1.1345, "abril": 1.0628,
119             "mayo": 1.0762, "junio": 1.0767, "julio": 0.8220, "agosto": 0.8684,
120             "septiembre": 0.9595, "octubre": 0.9494, "noviembre": 1.1134,
121             "diciembre": 0.9559
122         },
123         "CCASACANCHA": {
124             "enero": 1.0047, "febrero": 1.0408, "marzo": 1.0824, "abril": 1.0548,
125             "mayo": 1.0802, "junio": 1.0837, "julio": 0.9767, "agosto": 0.9426,
126             "septiembre": 0.9081, "octubre": 0.9961, "noviembre": 0.9756,
127             "diciembre": 0.9254
128         },
129         "CHACAPAMPA": {
130             "enero": 1.0446, "febrero": 0.9661, "marzo": 0.9178, "abril": 1.0174,
131             "mayo": 1.0405, "junio": 1.1257, "julio": 0.9550, "agosto": 0.9888,
132             "septiembre": 1.0378, "octubre": 1.0111, "noviembre": 1.1305,
133             "diciembre": 1.0215
134         },
135         "CHALHUAPUQUIO": {
136             "enero": 1.1671, "febrero": 1.2012, "marzo": 1.1909, "abril": 1.0566,
137             "mayo": 1.0146, "junio": 1.0442, "julio": 0.7925, "agosto": 0.8346,
138             "septiembre": 1.0198, "octubre": 0.9725, "noviembre": 1.0259,
139             "diciembre": 0.9319
140         },
141         "CHICAMA": {
142             "enero": 0.9671, "febrero": 0.9414, "marzo": 1.1130, "abril": 1.0394,
143             "mayo": 1.0773, "junio": 1.0585, "julio": 0.9554, "agosto": 1.0080,
144             "septiembre": 1.0731, "octubre": 1.0420, "noviembre": 1.0520,
145             "diciembre": 0.8249
146         },
147         "CHILCA": {
148             "enero": 0.6163, "febrero": 0.5814, "marzo": 0.7890, "abril": 1.0577,
149             "mayo": 1.5273, "junio": 1.5796, "julio": 1.2842, "agosto": 1.4015,
150             "septiembre": 1.4833, "octubre": 1.2554, "noviembre": 1.1876,
151             "diciembre": 0.7400
152         },
153         "CHULLQUI": {

```



```

145         "enero": 1.0428, "febrero": 1.0728, "marzo": 1.0509, "abril": 1.0163,
146         "mayo": 1.0500, "junio": 0.9407, "julio": 0.9832, "agosto": 0.9316 ,
147         "septiembre": 0.9915, "octubre": 0.9207, "noviembre": 1.2832,
148     "diciembre": 0.8829
149     },
150     "CHULUCANAS": {
151         "enero": 1.0135, "febrero": 1.0695, "marzo": 1.1798, "abril": 1.1501,
152         "mayo": 1.0679, "junio": 1.0398, "julio": 0.9751, "agosto": 0.9127,
153         "septiembre": 0.9779, "octubre": 0.9474, "noviembre": 0.9119,
154     "diciembre": 0.8682
155     },
156     "CIUDAD DE DIOS": {
157         "enero": 1.4832, "febrero": 0.9071, "marzo": 1.1711, "abril": 1.0698,
158         "mayo": 1.0027, "junio": 1.0245, "julio": 0.9031, "agosto": 0.9123,
159         "septiembre": 1.0735, "octubre": 1.0006, "noviembre": 1.2021,
160     "diciembre": 0.8987
161     },
162     "CORCONA": {
163         "enero": 1.1024, "febrero": 1.1641, "marzo": 1.2033, "abril": 1.0335,
164         "mayo": 0.9767, "junio": 1.0411, "julio": 0.7767, "agosto": 0.8725,
165         "septiembre": 1.0142, "octubre": 1.0035, "noviembre": 1.1653,
166     "diciembre": 1.0001
167     },
168     "CRUCE BAYOVAR": {
169         "enero": 0.8734, "febrero": 0.8855, "marzo": 1.1136, "abril": 1.1087,
170         "mayo": 1.1470, "junio": 1.1828, "julio": 0.8641, "agosto": 0.9783,
171         "septiembre": 1.1469, "octubre": 1.0024, "noviembre": 1.0777,
172     "diciembre": 0.7715
173     },
174     "CUCULI": {
175         "enero": 0.9948, "febrero": 1.0346, "marzo": 1.1594, "abril": 1.1180,
176         "mayo": 1.1059, "junio": 0.9573, "julio": 0.9652, "agosto": 0.9185,
177         "septiembre": 0.9475, "octubre": 0.9719, "noviembre": 0.9647,
178     "diciembre": 0.9955
179     },
180     "DESVIO OLMOS": {
181         "enero": 0.9497, "febrero": 1.0057, "marzo": 1.1672, "abril": 1.1939,
182         "mayo": 1.1494, "junio": 1.0960, "julio": 0.9411, "agosto": 0.8681,
183         "septiembre": 0.9965, "octubre": 0.9618, "noviembre": 1.0119,
184     "diciembre": 0.8222
185     },
186     "DESVIO TALARA": {
187         "enero": 0.8705, "febrero": 0.8863, "marzo": 1.1244, "abril": 1.0840,
188         "mayo": 1.1438, "junio": 1.1754, "julio": 0.9963, "agosto": 0.9874,
189         "septiembre": 1.1057, "octubre": 1.0288, "noviembre": 1.0309,
190     "diciembre": 0.8166
191     },
192     "EL FISCAL": {
193         "enero": 0.9109, "febrero": 0.8549, "marzo": 1.0606, "abril": 1.0601,
194         "mayo": 1.0691, "junio": 1.1260, "julio": 0.9958, "agosto": 0.9853,
195         "septiembre": 1.0750, "octubre": 1.0662, "noviembre": 1.0647,
196     "diciembre": 0.9479
197     },
198     "EL PARAISO": {
199         "enero": 0.9135, "febrero": 0.8984, "marzo": 1.0432, "abril": 0.9974,

```



```

191         "mayo": 1.1183, "junio": 1.1530, "julio": 0.9141, "agosto": 0.9811,
192         "septiembre": 1.0964, "octubre": 1.0384, "noviembre": 1.0935,
193         "diciembre": 0.8747
194     },
195     "FORTALEZA": {
196         "enero": 0.8869, "febrero": 0.8317, "marzo": 1.0074, "abril": 1.0227,
197         "mayo": 1.1568, "junio": 1.1965, "julio": 0.8936, "agosto": 1.0119,
198         "septiembre": 1.1837, "octubre": 1.1022, "noviembre": 1.1702,
199         "diciembre": 0.7936
200     },
201     "HUACRAPUQUIO": {
202         "enero": 1.1193, "febrero": 1.1570, "marzo": 1.0649, "abril": 1.0486,
203         "mayo": 1.1504, "junio": 0.9951, "julio": 0.8705, "agosto": 0.9487,
204         "septiembre": 0.9945, "octubre": 0.9710, "noviembre": 1.1529,
205         "diciembre": 0.8270
206     },
207     "HUARMEY": {
208         "enero": 0.8816, "febrero": 0.9056, "marzo": 1.1120, "abril": 1.1249,
209         "mayo": 1.2557, "junio": 1.2048, "julio": 0.8790, "agosto": 0.9703,
210         "septiembre": 1.1483, "octubre": 1.0831, "noviembre": 1.1634,
211         "diciembre": 0.7903
212     },
213     "HUATAYA": {
214         "enero": 0.0000, "febrero": 0.0000, "marzo": 2.3609, "abril": 0.9600,
215         "mayo": 0.8747, "junio": 0.8705, "julio": 0.7536, "agosto": 0.7578,
216         "septiembre": 0.0000, "octubre": 0.7931, "noviembre": 0.8788,
217         "diciembre": 0.8353
218     },
219     "ICA": {
220         "enero": 0.9310, "febrero": 0.8181, "marzo": 1.0513, "abril": 1.0461,
221         "mayo": 1.1642, "junio": 1.1857, "julio": 0.9741, "agosto": 1.0246,
222         "septiembre": 1.0994, "octubre": 1.0312, "noviembre": 1.1911,
223         "diciembre": 1.0337
224     },
225     "ILAVE": {
226         "enero": 1.0111, "febrero": 0.9584, "marzo": 1.0360, "abril": 1.0136,
227         "mayo": 1.1067, "junio": 1.1613, "julio": 0.9700, "agosto": 0.7834,
228         "septiembre": 1.0335, "octubre": 1.0501, "noviembre": 1.1248,
229         "diciembre": 0.9861
230     },
231     "ILO": {
232         "enero": 0.8275, "febrero": 0.8217, "marzo": 1.0112, "abril": 1.0814,
233         "mayo": 1.0777, "junio": 1.1220, "julio": 1.0989, "agosto": 1.0557,
234         "septiembre": 0.9854, "octubre": 1.0498, "noviembre": 1.0303,
235         "diciembre": 0.9630
236     },
237     "JAHUAY-CHINCHA": {
238         "enero": 0.8291, "febrero": 0.8936, "marzo": 1.0170, "abril": 0.9967,
239         "mayo": 1.1373, "junio": 1.1918, "julio": 0.9486, "agosto": 1.0137,
240         "septiembre": 1.1321, "octubre": 1.0340, "noviembre": 1.0634,
241         "diciembre": 0.9289
242     },
243     "LOMA LARGA BAJA": {
244         "enero": 1.0231, "febrero": 1.2771, "marzo": 1.3671, "abril": 1.2848,
245         "mayo": 1.1408, "junio": 1.0369, "julio": 0.8267, "agosto": 0.9064,

```



```

237         "septiembre": 0.9247, "octubre": 0.8925, "noviembre": 0.8826,
    "diciembre": 0.8752
238     },
239     "LUNAHUANA": {
240         "enero": 1.0119, "febrero": 1.0303, "marzo": 1.0475, "abril": 0.9587,
241         "mayo": 1.0163, "junio": 1.1317, "julio": 0.8233, "agosto": 0.9366,
242         "septiembre": 1.0988, "octubre": 0.9949, "noviembre": 1.0780,
    "diciembre": 1.0638
243     },
244     "MACUSANI": {
245         "enero": 1.0465, "febrero": 0.9990, "marzo": 1.0338, "abril": 1.0944,
246         "mayo": 1.0979, "junio": 1.1081, "julio": 0.9926, "agosto": 0.9493,
247         "septiembre": 1.0253, "octubre": 0.9652, "noviembre": 1.0158,
    "diciembre": 0.8879
248     },
249     "MARCONA": {
250         "enero": 0.9525, "febrero": 0.9039, "marzo": 0.9952, "abril": 1.0106,
251         "mayo": 1.0911, "junio": 1.0595, "julio": 1.0346, "agosto": 1.0232,
252         "septiembre": 1.0372, "octubre": 1.0290, "noviembre": 0.9989,
    "diciembre": 0.9111
253     },
254     "MATARANI": {
255         "enero": 0.4810, "febrero": 0.3998, "marzo": 0.9717, "abril": 1.5343,
256         "mayo": 1.7023, "junio": 1.6718, "julio": 1.6092, "agosto": 1.5584,
257         "septiembre": 1.5827, "octubre": 1.4257, "noviembre": 1.2888,
    "diciembre": 0.9119
258     },
259     "MENOCUCHO": {
260         "enero": 1.0850, "febrero": 0.9812, "marzo": 1.0500, "abril": 1.0811,
261         "mayo": 1.0266, "junio": 1.0713, "julio": 0.9585, "agosto": 0.9192,
262         "septiembre": 1.0498, "octubre": 1.0467, "noviembre": 1.0452,
    "diciembre": 0.8579
263     },
264     "MOCCE": {
265         "enero": 1.0115, "febrero": 0.9769, "marzo": 1.0613, "abril": 1.0650,
266         "mayo": 1.0408, "junio": 0.9962, "julio": 0.9898, "agosto": 0.9054,
267         "septiembre": 1.0213, "octubre": 1.0118, "noviembre": 1.0012,
    "diciembre": 0.9247
268     },
269     "MONTALVO": {
270         "enero": 0.9256, "febrero": 0.8896, "marzo": 1.0507, "abril": 1.0351,
271         "mayo": 1.0336, "junio": 1.1014, "julio": 1.0261, "agosto": 0.9905,
272         "septiembre": 1.0447, "octubre": 1.0608, "noviembre": 1.0346,
    "diciembre": 0.9658
273     },
274     "MORROPE": {
275         "enero": 0.9291, "febrero": 0.9180, "marzo": 1.2042, "abril": 1.1140,
276         "mayo": 1.1254, "junio": 1.1566, "julio": 0.8933, "agosto": 0.9593,
277         "septiembre": 1.0807, "octubre": 0.9811, "noviembre": 1.0512,
    "diciembre": 0.8016
278     },
279     "MOYOBAMBA": {
280         "enero": 1.0172, "febrero": 0.9993, "marzo": 1.0281, "abril": 1.0219,
281         "mayo": 0.9910, "junio": 0.9724, "julio": 0.9456, "agosto": 0.9603,

```



```

282         "septiembre": 1.0486, "octubre": 1.0817, "noviembre": 1.1373,
    "diciembre": 1.0800
283     },
284     "NASCA": {
285         "enero": 0.9690, "febrero": 0.9023, "marzo": 1.0494, "abril": 1.0595,
286         "mayo": 1.0762, "junio": 1.0902, "julio": 0.9243, "agosto": 0.9306,
287         "septiembre": 1.0245, "octubre": 1.0160, "noviembre": 1.0671,
    "diciembre": 0.9918
288     },
289     "PACANGUILLA": {
290         "enero": 0.9539, "febrero": 0.9606, "marzo": 1.1189, "abril": 1.1170,
291         "mayo": 1.1420, "junio": 1.1808, "julio": 0.9622, "agosto": 0.9834,
292         "septiembre": 1.0937, "octubre": 0.9832, "noviembre": 0.9979,
    "diciembre": 0.7654
293     },
294     "PACRA": {
295         "enero": 1.0091, "febrero": 0.9570, "marzo": 1.0209, "abril": 0.9881,
296         "mayo": 1.2122, "junio": 1.0758, "julio": 0.8772, "agosto": 0.9346,
297         "septiembre": 1.0435, "octubre": 1.0345, "noviembre": 1.0857,
    "diciembre": 0.9795
298     },
299     "PAITA": {
300         "enero": 0.8362, "febrero": 0.8550, "marzo": 0.9600, "abril": 1.1002,
301         "mayo": 1.1329, "junio": 1.1228, "julio": 1.0946, "agosto": 1.0862,
302         "septiembre": 1.0113, "octubre": 1.0572, "noviembre": 1.0343,
    "diciembre": 0.8543
303     },
304     "PAMPA CUELLAR": {
305         "enero": 1.0268, "febrero": 0.8142, "marzo": 1.0724, "abril": 1.0806,
306         "mayo": 1.1616, "junio": 1.1803, "julio": 0.9567, "agosto": 0.7957,
307         "septiembre": 1.0641, "octubre": 1.0380, "noviembre": 1.0557,
    "diciembre": 0.9772
308     },
309     "PAMPA GALERA": {
310         "enero": 0.9640, "febrero": 1.0087, "marzo": 1.1311, "abril": 1.1214,
311         "mayo": 1.0518, "junio": 1.0972, "julio": 0.8190, "agosto": 0.7818,
312         "septiembre": 1.0505, "octubre": 1.0760, "noviembre": 1.1323,
    "diciembre": 0.9586
313     },
314     "PAMPAMARCA": {
315         "enero": 0.9412, "febrero": 0.9552, "marzo": 1.0962, "abril": 1.0681,
316         "mayo": 1.1301, "junio": 1.1134, "julio": 0.8653, "agosto": 0.8837,
317         "septiembre": 1.0188, "octubre": 1.0975, "noviembre": 1.1159,
    "diciembre": 0.9830
318     },
319     "PATAHUASI": {
320         "enero": 1.0505, "febrero": 0.9287, "marzo": 1.1460, "abril": 1.0872,
321         "mayo": 1.1086, "junio": 1.1280, "julio": 0.9169, "agosto": 0.8010,
322         "septiembre": 1.0421, "octubre": 0.9869, "noviembre": 1.0075,
    "diciembre": 0.9579
323     },
324     "PEDRO RUIZ": {
325         "enero": 0.9661, "febrero": 1.0279, "marzo": 1.1022, "abril": 1.1233,
326         "mayo": 1.1127, "junio": 1.0515, "julio": 0.9396, "agosto": 0.9041,

```

```
327         "septiembre": 0.9700, "octubre": 0.9832, "noviembre": 1.0130,  
"diciembre": 0.8890  
328     },  
329     "PICHIRHUA": {  
330         "enero": 1.0465, "febrero": 1.0891, "marzo": 1.1333, "abril": 1.0561,  
331         "mayo": 1.0383, "junio": 1.0173, "julio": 0.9081, "agosto": 0.8840,  
332         "septiembre": 0.9812, "octubre": 0.9968, "noviembre": 1.0126,  
"diciembre": 0.9151  
333     },  
334     "PIURA SULLANA": {  
335         "enero": 1.0607, "febrero": 1.0540, "marzo": 1.1688, "abril": 1.0898,  
336         "mayo": 1.0558, "junio": 1.0528, "julio": 0.9697, "agosto": 0.9539,  
337         "septiembre": 1.0019, "octubre": 0.9640, "noviembre": 0.9557,  
"diciembre": 0.8684  
338     },  
339     "PLANCHON": {  
340         "enero": 1.0327, "febrero": 1.0756, "marzo": 1.0641, "abril": 1.0613,  
341         "mayo": 1.0628, "junio": 1.0224, "julio": 0.9361, "agosto": 0.9131,  
342         "septiembre": 0.9554, "octubre": 0.9680, "noviembre": 1.0417,  
"diciembre": 0.9385  
343     },  
344     "POMAHUACA": {  
345         "enero": 0.9587, "febrero": 0.9933, "marzo": 1.1653, "abril": 1.2213,  
346         "mayo": 1.1511, "junio": 1.0886, "julio": 0.9287, "agosto": 0.8481,  
347         "septiembre": 1.0002, "octubre": 0.9934, "noviembre": 1.0133,  
"diciembre": 0.8296  
348     },  
349     "PONGO": {  
350         "enero": 1.0265, "febrero": 1.0591, "marzo": 1.0560, "abril": 1.0787,  
351         "mayo": 1.0546, "junio": 1.0025, "julio": 1.0750, "agosto": 0.9103,  
352         "septiembre": 0.9692, "octubre": 0.9654, "noviembre": 0.9778,  
"diciembre": 0.9308  
353     },  
354     "POZO REDONDO": {  
355         "enero": 0.8942, "febrero": 0.8240, "marzo": 1.0133, "abril": 1.0783,  
356         "mayo": 1.1178, "junio": 1.0895, "julio": 1.0494, "agosto": 1.0438,  
357         "septiembre": 1.1167, "octubre": 1.0579, "noviembre": 1.0216,  
"diciembre": 0.8480  
358     },  
359     "PUNTA PERDIDA": {  
360         "enero": 0.9544, "febrero": 0.7922, "marzo": 1.0980, "abril": 1.1995,  
361         "mayo": 1.4001, "junio": 1.4069, "julio": 0.8238, "agosto": 0.6031,  
362         "septiembre": 1.1767, "octubre": 1.0628, "noviembre": 1.2693,  
"diciembre": 1.1014  
363     },  
364     "QUIULLA": {  
365         "enero": 1.0974, "febrero": 1.1408, "marzo": 1.2315, "abril": 1.0452,  
366         "mayo": 1.0212, "junio": 1.0679, "julio": 0.8140, "agosto": 0.8685,  
367         "septiembre": 1.0002, "octubre": 1.0063, "noviembre": 1.1046,  
"diciembre": 0.9381  
368     },  
369     "RUMICHACA": {  
370         "enero": 1.0577, "febrero": 0.9112, "marzo": 0.9934, "abril": 0.9745,  
371         "mayo": 1.2193, "junio": 1.2055, "julio": 0.8748, "agosto": 0.8955,
```



```

372         "septiembre": 1.0297, "octubre": 1.0766, "noviembre": 1.1671,
    "diciembre": 0.9701
373     },
374     "SAN ANTON": {
375         "enero": 1.0691, "febrero": 1.0523, "marzo": 1.0689, "abril": 1.1318,
376         "mayo": 1.0035, "junio": 1.0850, "julio": 1.0541, "agosto": 1.0138,
377         "septiembre": 0.9978, "octubre": 0.9206, "noviembre": 0.9981,
    "diciembre": 0.8467
378     },
379     "SAN GABAN": {
380         "enero": 1.0594, "febrero": 1.0125, "marzo": 1.0893, "abril": 1.1043,
381         "mayo": 1.1331, "junio": 1.0921, "julio": 0.9727, "agosto": 0.8972,
382         "septiembre": 0.9272, "octubre": 0.9258, "noviembre": 0.9512,
    "diciembre": 0.9451
383     },
384     "SAN LORENZO": {
385         "enero": 1.0839, "febrero": 1.1914, "marzo": 1.1227, "abril": 1.2647,
386         "mayo": 1.0266, "junio": 1.0376, "julio": 0.8703, "agosto": 0.9258,
387         "septiembre": 0.9436, "octubre": 0.9610, "noviembre": 1.0442,
    "diciembre": 0.9270
388     },
389     "SANTA LUCIA": {
390         "enero": 1.0087, "febrero": 0.8363, "marzo": 1.1211, "abril": 1.1106,
391         "mayo": 1.1184, "junio": 1.1768, "julio": 0.9560, "agosto": 0.7674,
392         "septiembre": 1.0724, "octubre": 1.0214, "noviembre": 1.0523,
    "diciembre": 0.9795
393     },
394     "SAYLLA": {
395         "enero": 1.0222, "febrero": 0.9785, "marzo": 1.1062, "abril": 1.0858,
396         "mayo": 1.0613, "junio": 1.0638, "julio": 0.9806, "agosto": 0.9142,
397         "septiembre": 0.9238, "octubre": 0.9909, "noviembre": 0.9947,
    "diciembre": 0.9411
398     },
399     "SERPENTIN DE PASAMAYO": {
400         "enero": 1.0535, "febrero": 1.0503, "marzo": 1.0886, "abril": 1.0814,
401         "mayo": 1.0647, "junio": 1.0464, "julio": 0.9694, "agosto": 0.8540,
402         "septiembre": 1.0238, "octubre": 1.0450, "noviembre": 1.0263,
    "diciembre": 1.0654
403     },
404     "SICUYANI": {
405         "enero": 1.0297, "febrero": 0.8278, "marzo": 1.0261, "abril": 1.0976,
406         "mayo": 1.1547, "junio": 1.1691, "julio": 0.9155, "agosto": 0.7740,
407         "septiembre": 0.9930, "octubre": 1.0466, "noviembre": 1.1655,
    "diciembre": 1.0126
408     },
409     "SOCOS": {
410         "enero": 1.1903, "febrero": 0.9863, "marzo": 0.9820, "abril": 0.9188,
411         "mayo": 1.0940, "junio": 1.0772, "julio": 0.9340, "agosto": 0.9544,
412         "septiembre": 1.0099, "octubre": 1.0057, "noviembre": 1.0421,
    "diciembre": 0.9406
413     },
414     "TAMBOGRANDE": {
415         "enero": 0.9297, "febrero": 0.9679, "marzo": 1.0638, "abril": 1.1030,
416         "mayo": 1.2229, "junio": 1.0587, "julio": 1.0396, "agosto": 1.0332,

```



```

417         "septiembre": 1.0144, "octubre": 0.8993, "noviembre": 0.9585,
"dicembre": 0.8799
418     },
419     "TOMASIRI": {
420         "enero": 0.9972, "febrero": 0.9232, "marzo": 1.0516, "abril": 1.0804,
421         "mayo": 1.1000, "junio": 1.0942, "julio": 1.0293, "agosto": 0.9900,
422         "septiembre": 0.8973, "octubre": 1.0404, "noviembre": 1.0386,
"dicembre": 0.9031
423     },
424     "TUNAN": {
425         "enero": 1.0761, "febrero": 1.0516, "marzo": 1.0703, "abril": 1.0230,
426         "mayo": 1.1174, "junio": 1.0413, "julio": 0.8626, "agosto": 0.8554,
427         "septiembre": 0.9819, "octubre": 0.9733, "noviembre": 1.1132,
"dicembre": 0.9816
428     },
429     "UNION-PROGRESO": {
430         "enero": 1.0256, "febrero": 1.0263, "marzo": 1.0728, "abril": 1.0366,
431         "mayo": 1.0256, "junio": 1.0113, "julio": 0.9544, "agosto": 0.9355,
432         "septiembre": 0.9696, "octubre": 1.0061, "noviembre": 1.1122,
"dicembre": 0.9406
433     },
434     "UTCUBAMBA": {
435         "enero": 1.1969, "febrero": 1.0343, "marzo": 1.0957, "abril": 1.1113,
436         "mayo": 1.0756, "junio": 1.0478, "julio": 0.9556, "agosto": 0.9124,
437         "septiembre": 0.9571, "octubre": 0.9881, "noviembre": 0.9318,
"dicembre": 0.8952
438     },
439     "VARIANTE DE PASAMAYO": {
440         "enero": 0.9444, "febrero": 0.9203, "marzo": 1.0337, "abril": 1.0118,
441         "mayo": 1.0802, "junio": 1.1121, "julio": 0.9392, "agosto": 0.9927,
442         "septiembre": 1.0740, "octubre": 1.0049, "noviembre": 1.0472,
"dicembre": 0.9053
443     },
444     "VARIANTE DE UCHUMAYO": {
445         "enero": 0.7411, "febrero": 0.6766, "marzo": 1.0218, "abril": 1.1452,
446         "mayo": 1.1929, "junio": 1.1990, "julio": 1.1217, "agosto": 1.0762,
447         "septiembre": 1.1286, "octubre": 1.1417, "noviembre": 1.1231,
"dicembre": 0.9590
448     },
449     "VESIQUE": {
450         "enero": 0.8602, "febrero": 0.9031, "marzo": 1.0597, "abril": 1.0853,
451         "mayo": 1.1302, "junio": 1.1518, "julio": 1.0065, "agosto": 1.0684,
452         "septiembre": 1.1196, "octubre": 1.0108, "noviembre": 1.0305,
"dicembre": 0.8067
453     },
454     "VIRU": {
455         "enero": 0.9903, "febrero": 0.9640, "marzo": 1.0774, "abril": 1.0604,
456         "mayo": 1.1153, "junio": 1.1221, "julio": 0.9573, "agosto": 1.0148,
457         "septiembre": 1.2094, "octubre": 0.9743, "noviembre": 0.9961,
"dicembre": 0.7989
458     },
459     "YAUCA": {
460         "enero": 0.8794, "febrero": 0.7966, "marzo": 1.0518, "abril": 1.0285,
461         "mayo": 1.1247, "junio": 1.1292, "julio": 0.9653, "agosto": 0.9839,

```

```

462         "septiembre": 1.0684, "octubre": 1.0963, "noviembre": 1.1303,
    "diciembre": 0.9048
463     }
464 }
465
466 # FACTORES PARA VEHÍCULOS PESADOS - 78 ESTACIONES
467 self.fce_pesados = {
468     "AGUAS CALIENTES": {
469         "enero": 1.0220, "febrero": 0.9808, "marzo": 1.0324, "abril": 1.0653,
470         "mayo": 1.0697, "junio": 1.1028, "julio": 0.9678, "agosto": 0.9555,
471         "septiembre": 0.9682, "octubre": 0.9530, "noviembre": 0.9599,
    "diciembre": 1.0098
472     },
473     "AGUAS CLARAS": {
474         "enero": 1.0412, "febrero": 1.0049, "marzo": 1.0248, "abril": 1.0107,
475         "mayo": 0.9905, "junio": 0.9823, "julio": 0.9917, "agosto": 0.9523,
476         "septiembre": 0.9768, "octubre": 1.0070, "noviembre": 1.0080,
    "diciembre": 1.0344
477     },
478     "AMBO": {
479         "enero": 1.0087, "febrero": 0.9707, "marzo": 1.1288, "abril": 1.2086,
480         "mayo": 1.0139, "junio": 1.0238, "julio": 1.0261, "agosto": 1.0008,
481         "septiembre": 0.9890, "octubre": 0.9459, "noviembre": 0.9401,
    "diciembre": 0.9497
482     },
483     "ATICO": {
484         "enero": 1.0337, "febrero": 0.9914, "marzo": 1.0298, "abril": 1.0410,
485         "mayo": 1.0342, "junio": 1.0302, "julio": 1.0277, "agosto": 0.9851,
486         "septiembre": 1.0265, "octubre": 0.9618, "noviembre": 0.9271,
    "diciembre": 0.9440
487     },
488     "AYAVIRI": {
489         "enero": 1.0167, "febrero": 1.0038, "marzo": 1.0708, "abril": 1.0573,
490         "mayo": 1.0486, "junio": 1.0366, "julio": 0.9665, "agosto": 0.9524,
491         "septiembre": 0.9501, "octubre": 0.9638, "noviembre": 0.9601,
    "diciembre": 1.0007
492     },
493     "CAMANA": {
494         "enero": 0.9513, "febrero": 0.8973, "marzo": 1.0392, "abril": 1.0648,
495         "mayo": 1.0702, "junio": 1.0905, "julio": 1.1087, "agosto": 0.9914,
496         "septiembre": 0.9997, "octubre": 0.9905, "noviembre": 0.9708,
    "diciembre": 0.9624
497     },
498     "CANCAS": {
499         "enero": 1.0323, "febrero": 0.9825, "marzo": 1.0400, "abril": 1.0412,
500         "mayo": 1.0604, "junio": 1.0476, "julio": 1.0117, "agosto": 0.9795,
501         "septiembre": 1.0006, "octubre": 0.9550, "noviembre": 0.9461,
    "diciembre": 0.9491
502     },
503     "CARACOTO": {
504         "enero": 1.0343, "febrero": 1.0013, "marzo": 1.0777, "abril": 1.0346,
505         "mayo": 1.0568, "junio": 1.0441, "julio": 0.9898, "agosto": 0.9297,
506         "septiembre": 0.9572, "octubre": 0.9481, "noviembre": 1.0071,
    "diciembre": 0.9574
507     },

```



```

508     "CASARACRA": {
509         "enero": 1.0936, "febrero": 1.0670, "marzo": 1.1094, "abril": 0.9826,
510         "mayo": 0.9904, "junio": 0.9900, "julio": 0.9969, "agosto": 0.9700,
511         "septiembre": 0.9760, "octubre": 0.9548, "noviembre": 1.0373,
512         "diciembre": 1.0652
513     },
514     "CATAC": {
515         "enero": 1.0441, "febrero": 1.0678, "marzo": 1.1999, "abril": 1.0772,
516         "mayo": 1.1736, "junio": 0.9553, "julio": 0.9446, "agosto": 0.9253,
517         "septiembre": 0.9566, "octubre": 0.9547, "noviembre": 0.9810,
518         "diciembre": 0.9634
519     },
520     "CCASACANCHA": {
521         "enero": 1.0968, "febrero": 1.0852, "marzo": 1.0957, "abril": 1.0809,
522         "mayo": 1.0239, "junio": 0.9967, "julio": 0.9386, "agosto": 0.9525,
523         "septiembre": 0.9555, "octubre": 0.9560, "noviembre": 0.9332,
524         "diciembre": 0.9566
525     },
526     "CHACAPAMPA": {
527         "enero": 1.1031, "febrero": 0.9998, "marzo": 0.9655, "abril": 0.9941,
528         "mayo": 1.0102, "junio": 1.0186, "julio": 1.0155, "agosto": 1.0051,
529         "septiembre": 0.9767, "octubre": 0.9556, "noviembre": 0.9706,
530         "diciembre": 1.0601
531     },
532     "CHALHUAPUQUIO": {
533         "enero": 1.0680, "febrero": 1.0692, "marzo": 1.0807, "abril": 1.0683,
534         "mayo": 1.0507, "junio": 0.9933, "julio": 0.9468, "agosto": 0.9321,
535         "septiembre": 0.9565, "octubre": 0.9463, "noviembre": 0.9497,
536         "diciembre": 0.9939
537     },
538     "CHICAMA": {
539         "enero": 0.9909, "febrero": 0.9874, "marzo": 1.0928, "abril": 1.0708,
540         "mayo": 1.0543, "junio": 1.0573, "julio": 1.0423, "agosto": 0.9933,
541         "septiembre": 1.0013, "octubre": 0.9946, "noviembre": 0.9510,
542         "diciembre": 0.8799
543     },
544     "CHILCA": {
545         "enero": 0.9571, "febrero": 0.9724, "marzo": 1.0203, "abril": 1.0381,
546         "mayo": 1.0583, "junio": 1.0539, "julio": 1.0372, "agosto": 0.9948,
547         "septiembre": 0.9992, "octubre": 0.9880, "noviembre": 0.9652,
548         "diciembre": 0.9396
549     },
550     "CHULLQUI": {
551         "enero": 0.9571, "febrero": 0.9658, "marzo": 1.0534, "abril": 1.0776,
552         "mayo": 1.0809, "junio": 1.0402, "julio": 1.0171, "agosto": 0.9865,
553         "septiembre": 0.9731, "octubre": 0.9169, "noviembre": 1.2400,
554         "diciembre": 0.9257
555     },
556     "CHULUCANAS": {
557         "enero": 0.9884, "febrero": 0.9682, "marzo": 1.1388, "abril": 1.1641,
558         "mayo": 1.0933, "junio": 1.0508, "julio": 1.0256, "agosto": 0.9679,
559         "septiembre": 0.9628, "octubre": 0.9205, "noviembre": 0.8943,
560         "diciembre": 0.9220
561     },
562     "CIUDAD DE DIOS": {

```



```

554         "enero": 1.4397, "febrero": 0.9383, "marzo": 1.1026, "abril": 0.9966,
555         "mayo": 0.9637, "junio": 1.0342, "julio": 1.0428, "agosto": 1.0261,
556         "septiembre": 1.0414, "octubre": 0.9841, "noviembre": 0.9930,
    "diciembre": 0.9649
557     },
558     "CORCONA": {
559         "enero": 1.0985, "febrero": 1.0709, "marzo": 1.1124, "abril": 0.9593,
560         "mayo": 0.9674, "junio": 0.9840, "julio": 0.9816, "agosto": 0.9627,
561         "septiembre": 0.9804, "octubre": 0.9767, "noviembre": 1.0641,
    "diciembre": 1.0861
562     },
563     "CRUCE BAYOVAR": {
564         "enero": 0.9845, "febrero": 0.9606, "marzo": 1.0584, "abril": 1.0655,
565         "mayo": 1.0311, "junio": 1.0361, "julio": 1.0157, "agosto": 0.9821,
566         "septiembre": 1.0167, "octubre": 0.9774, "noviembre": 0.9231,
    "diciembre": 0.9028
567     },
568     "CUCULI": {
569         "enero": 0.9424, "febrero": 1.0419, "marzo": 1.2043, "abril": 1.1560,
570         "mayo": 1.0848, "junio": 0.9783, "julio": 0.9812, "agosto": 0.9398,
571         "septiembre": 0.9207, "octubre": 0.9302, "noviembre": 0.9384,
    "diciembre": 1.0749
572     },
573     "DESVIO OLMOS": {
574         "enero": 1.0470, "febrero": 1.0401, "marzo": 1.0986, "abril": 1.0775,
575         "mayo": 1.0393, "junio": 1.0176, "julio": 0.9898, "agosto": 0.9217,
576         "septiembre": 0.9405, "octubre": 0.9579, "noviembre": 0.9500,
    "diciembre": 0.9817
577     },
578     "DESVIO TALARA": {
579         "enero": 1.0063, "febrero": 0.9722, "marzo": 1.0418, "abril": 1.0405,
580         "mayo": 1.0343, "junio": 1.0196, "julio": 1.0721, "agosto": 0.9782,
581         "septiembre": 0.9984, "octubre": 0.9792, "noviembre": 0.9631,
    "diciembre": 0.9516
582     },
583     "EL FISCAL": {
584         "enero": 0.9889, "febrero": 0.9326, "marzo": 1.0201, "abril": 1.0355,
585         "mayo": 1.0216, "junio": 1.0855, "julio": 1.0665, "agosto": 1.0209,
586         "septiembre": 1.0150, "octubre": 0.9799, "noviembre": 0.9618,
    "diciembre": 0.9819
587     },
588     "EL PARAISO": {
589         "enero": 1.0084, "febrero": 0.9860, "marzo": 1.0607, "abril": 1.0471,
590         "mayo": 1.0250, "junio": 1.0153, "julio": 1.0105, "agosto": 0.9788,
591         "septiembre": 0.9973, "octubre": 0.9801, "noviembre": 0.9625,
    "diciembre": 0.9429
592     },
593     "FORTALEZA": {
594         "enero": 0.9938, "febrero": 0.9560, "marzo": 1.0154, "abril": 1.0346,
595         "mayo": 1.0384, "junio": 1.3018, "julio": 1.0249, "agosto": 0.9743,
596         "septiembre": 0.9948, "octubre": 0.9789, "noviembre": 0.9539,
    "diciembre": 0.9356
597     },
598     "HUACRAPUQUIO": {
599         "enero": 1.0850, "febrero": 1.1264, "marzo": 1.0529, "abril": 1.0465,

```



```

600         "mayo": 1.1603, "junio": 1.0254, "julio": 0.9226, "agosto": 0.9778,
601         "septiembre": 0.9218, "octubre": 0.9085, "noviembre": 1.1194,
        "diciembre": 0.9334
602     },
603     "HUARMEY": {
604         "enero": 1.0456, "febrero": 1.0252, "marzo": 1.1132, "abril": 1.1416,
605         "mayo": 1.1272, "junio": 1.0303, "julio": 0.9991, "agosto": 0.9554,
606         "septiembre": 0.9757, "octubre": 0.9617, "noviembre": 0.9397,
        "diciembre": 0.9169
607     },
608     "HUATAYA": {
609         "enero": 0.0000, "febrero": 0.0000, "marzo": 5.1132, "abril": 1.3415,
610         "mayo": 1.0567, "junio": 1.1183, "julio": 0.7106, "agosto": 0.7730,
611         "septiembre": 0.0000, "octubre": 0.7399, "noviembre": 0.8727,
        "diciembre": 0.9802
612     },
613     "ICA": {
614         "enero": 0.9911, "febrero": 0.9493, "marzo": 1.0440, "abril": 1.0283,
615         "mayo": 1.0670, "junio": 1.0747, "julio": 1.0593, "agosto": 0.9907,
616         "septiembre": 0.9632, "octubre": 0.9523, "noviembre": 1.0104,
        "diciembre": 1.0032
617     },
618     "ILAVE": {
619         "enero": 1.0136, "febrero": 0.9470, "marzo": 0.9822, "abril": 5.5822,
620         "mayo": 1.0245, "junio": 1.0346, "julio": 1.0466, "agosto": 0.8667,
621         "septiembre": 1.0477, "octubre": 1.0417, "noviembre": 1.0885,
        "diciembre": 1.0638
622     },
623     "ILO": {
624         "enero": 1.0476, "febrero": 1.0169, "marzo": 1.0590, "abril": 0.9766,
625         "mayo": 0.9992, "junio": 1.0420, "julio": 1.0373, "agosto": 1.0284,
626         "septiembre": 0.9669, "octubre": 0.9802, "noviembre": 0.9728,
        "diciembre": 0.9625
627     },
628     "JAHUAY-CHINCHA": {
629         "enero": 0.9858, "febrero": 1.0162, "marzo": 1.0441, "abril": 1.0554,
630         "mayo": 1.0544, "junio": 1.0480, "julio": 1.0262, "agosto": 0.9755,
631         "septiembre": 0.9777, "octubre": 0.9697, "noviembre": 0.9511,
        "diciembre": 0.9478
632     },
633     "LOMA LARGA BAJA": {
634         "enero": 0.9981, "febrero": 1.1016, "marzo": 1.1995, "abril": 1.2276,
635         "mayo": 1.1103, "junio": 1.0823, "julio": 0.9596, "agosto": 0.9761,
636         "septiembre": 0.9483, "octubre": 0.9331, "noviembre": 0.8987,
        "diciembre": 0.8944
637     },
638     "LUNAHUANA": {
639         "enero": 1.1095, "febrero": 1.0967, "marzo": 1.0541, "abril": 1.0430,
640         "mayo": 0.9971, "junio": 1.0739, "julio": 1.0009, "agosto": 0.9482,
641         "septiembre": 0.9827, "octubre": 0.9329, "noviembre": 0.9403,
        "diciembre": 0.9907
642     },
643     "MACUSANI": {
644         "enero": 1.0895, "febrero": 1.0802, "marzo": 1.0669, "abril": 1.0219,
645         "mayo": 1.0756, "junio": 1.0209, "julio": 1.1741, "agosto": 0.9353,

```

```
646         "septiembre": 0.9179, "octubre": 0.9386, "noviembre": 0.9390,  
        "diciembre": 0.9983  
647     },  
648     "MARCONA": {  
649         "enero": 1.0430, "febrero": 1.0013, "marzo": 0.9583, "abril": 1.0297,  
650         "mayo": 1.1134, "junio": 1.0444, "julio": 1.0480, "agosto": 1.0472,  
651         "septiembre": 1.0260, "octubre": 0.9629, "noviembre": 0.9465,  
        "diciembre": 0.9320  
652     },  
653     "MATARANI": {  
654         "enero": 0.9673, "febrero": 0.8698, "marzo": 1.0297, "abril": 1.0802,  
655         "mayo": 1.0657, "junio": 1.0279, "julio": 1.0254, "agosto": 1.0446,  
656         "septiembre": 0.9864, "octubre": 0.9950, "noviembre": 0.9822,  
        "diciembre": 0.9819  
657     },  
658     "MENOCUCHO": {  
659         "enero": 1.0933, "febrero": 1.0653, "marzo": 1.1421, "abril": 1.0364,  
660         "mayo": 0.9913, "junio": 0.9622, "julio": 0.9505, "agosto": 0.9625,  
661         "septiembre": 0.9785, "octubre": 0.9841, "noviembre": 0.9862,  
        "diciembre": 1.0148  
662     },  
663     "MOCCE": {  
664         "enero": 0.9510, "febrero": 0.9816, "marzo": 1.0770, "abril": 1.1377,  
665         "mayo": 1.0767, "junio": 0.9655, "julio": 1.0381, "agosto": 0.9850,  
666         "septiembre": 0.9950, "octubre": 0.9641, "noviembre": 0.9558,  
        "diciembre": 0.9435  
667     },  
668     "MONTALVO": {  
669         "enero": 0.9754, "febrero": 0.9558, "marzo": 1.0155, "abril": 1.0254,  
670         "mayo": 1.0095, "junio": 1.0844, "julio": 1.0785, "agosto": 1.0275,  
671         "septiembre": 1.0167, "octubre": 0.9888, "noviembre": 0.9674,  
        "diciembre": 0.9800  
672     },  
673     "MORROPE": {  
674         "enero": 0.9854, "febrero": 0.9645, "marzo": 1.0770, "abril": 1.0663,  
675         "mayo": 1.0358, "junio": 1.0336, "julio": 1.0257, "agosto": 1.0003,  
676         "septiembre": 1.0155, "octubre": 0.9853, "noviembre": 0.9451,  
        "diciembre": 0.9179  
677     },  
678     "MOYOBAMBA": {  
679         "enero": 1.0123, "febrero": 0.0000, "marzo": 0.9812, "abril": 1.0299,  
680         "mayo": 1.0077, "junio": 0.9903, "julio": 1.0012, "agosto": 0.9653,  
681         "septiembre": 0.9861, "octubre": 1.0086, "noviembre": 1.0129,  
        "diciembre": 1.0379  
682     },  
683     "NAZCA": {  
684         "enero": 1.0562, "febrero": 1.0098, "marzo": 1.0329, "abril": 1.0328,  
685         "mayo": 1.0325, "junio": 1.0269, "julio": 1.0128, "agosto": 0.9732,  
686         "septiembre": 0.9609, "octubre": 0.9602, "noviembre": 0.9275,  
        "diciembre": 1.0491  
687     },  
688     "PACANGUILLA": {  
689         "enero": 1.0060, "febrero": 0.9822, "marzo": 1.0858, "abril": 1.0969,  
690         "mayo": 1.0766, "junio": 1.0798, "julio": 1.0941, "agosto": 1.0166,
```



```

691         "septiembre": 0.9911, "octubre": 0.9429, "noviembre": 0.9142,
        "diciembre": 0.8905
692     },
693     "PACRA": {
694         "enero": 1.0832, "febrero": 1.0100, "marzo": 1.0286, "abril": 1.0364,
695         "mayo": 1.1083, "junio": 1.0029, "julio": 0.9686, "agosto": 0.9412,
696         "septiembre": 0.9706, "octubre": 0.9520, "noviembre": 0.9859,
        "diciembre": 0.9960
697     },
698     "PAITA": {
699         "enero": 1.0637, "febrero": 0.9986, "marzo": 1.0332, "abril": 1.1742,
700         "mayo": 1.1086, "junio": 1.0719, "julio": 1.1384, "agosto": 1.0669,
701         "septiembre": 0.9588, "octubre": 0.9185, "noviembre": 0.8559,
        "diciembre": 0.9255
702     },
703     "PAMPA CUELLAR": {
704         "enero": 1.1081, "febrero": 1.0770, "marzo": 1.0639, "abril": 1.0083,
705         "mayo": 1.1112, "junio": 1.0845, "julio": 0.9918, "agosto": 0.9561,
706         "septiembre": 0.9589, "octubre": 0.8916, "noviembre": 0.9121,
        "diciembre": 0.9504
707     },
708     "PAMPA GALERA": {
709         "enero": 1.1056, "febrero": 1.0973, "marzo": 1.0952, "abril": 1.0658,
710         "mayo": 1.0378, "junio": 1.0205, "julio": 0.9472, "agosto": 0.9311,
711         "septiembre": 0.9550, "octubre": 0.9441, "noviembre": 0.9360,
        "diciembre": 0.9479
712     },
713     "PAMPAMARCA": {
714         "enero": 1.0833, "febrero": 1.0608, "marzo": 1.0800, "abril": 1.0714,
715         "mayo": 1.0640, "junio": 1.0321, "julio": 0.9519, "agosto": 0.9426,
716         "septiembre": 0.9317, "octubre": 0.9501, "noviembre": 0.9398,
        "diciembre": 0.9688
717     },
718     "PATAHUASI": {
719         "enero": 1.0668, "febrero": 1.0418, "marzo": 1.0805, "abril": 1.0643,
720         "mayo": 1.0616, "junio": 1.0569, "julio": 1.0065, "agosto": 0.9440,
721         "septiembre": 0.9475, "octubre": 0.9036, "noviembre": 0.9309,
        "diciembre": 0.9274
722     },
723     "PEDRO RUIZ": {
724         "enero": 1.0361, "febrero": 1.0163, "marzo": 1.0483, "abril": 1.0529,
725         "mayo": 1.0166, "junio": 0.9901, "julio": 0.9900, "agosto": 0.9413,
726         "septiembre": 0.9610, "octubre": 0.9787, "noviembre": 0.9826,
        "diciembre": 1.0212
727     },
728     "PICHIRHUA": {
729         "enero": 1.0936, "febrero": 1.0887, "marzo": 1.1081, "abril": 1.0925,
730         "mayo": 1.0545, "junio": 1.0256, "julio": 0.9460, "agosto": 0.9321,
731         "septiembre": 0.9383, "octubre": 0.9460, "noviembre": 0.9298,
        "diciembre": 0.9399
732     },
733     "PIURA SULLANA": {
734         "enero": 1.0571, "febrero": 1.0402, "marzo": 1.1277, "abril": 1.0518,
735         "mayo": 1.0405, "junio": 1.0119, "julio": 1.0179, "agosto": 0.9763,

```

```

736         "septiembre": 0.9950, "octubre": 0.9755, "noviembre": 0.9427,
"dicembre": 0.9189
737     },
738     "PLANCHON": {
739         "enero": 1.3722, "febrero": 1.2585, "marzo": 1.1075, "abril": 1.1700,
740         "mayo": 1.0496, "junio": 1.0154, "julio": 0.9275, "agosto": 0.8807,
741         "septiembre": 0.8567, "octubre": 0.8959, "noviembre": 0.9614,
"dicembre": 1.0233
742     },
743     "POMAHUACA": {
744         "enero": 1.0674, "febrero": 1.0301, "marzo": 1.1158, "abril": 1.1109,
745         "mayo": 1.0645, "junio": 1.0235, "julio": 0.9882, "agosto": 0.9203,
746         "septiembre": 0.9522, "octubre": 0.9370, "noviembre": 0.9332,
"dicembre": 0.9668
747     },
748     "PONGO": {
749         "enero": 1.1145, "febrero": 1.0702, "marzo": 1.0868, "abril": 1.0354,
750         "mayo": 0.9963, "junio": 0.9774, "julio": 1.0443, "agosto": 0.9020,
751         "septiembre": 0.9205, "octubre": 0.9680, "noviembre": 0.9712,
"dicembre": 1.0699
752     },
753     "POZO REDONDO": {
754         "enero": 1.0157, "febrero": 0.9826, "marzo": 1.0195, "abril": 1.0517,
755         "mayo": 1.0534, "junio": 1.0459, "julio": 1.0140, "agosto": 0.9923,
756         "septiembre": 1.0345, "octubre": 1.0012, "noviembre": 0.9435,
"dicembre": 0.8997
757     },
758     "PUNTA PERDIDA": {
759         "enero": 1.1180, "febrero": 1.1218, "marzo": 1.0774, "abril": 1.0165,
760         "mayo": 1.2427, "junio": 1.1333, "julio": 1.0120, "agosto": 0.9370,
761         "septiembre": 0.9270, "octubre": 0.8749, "noviembre": 0.9204,
"dicembre": 0.9490
762     },
763     "QUIULLA": {
764         "enero": 1.1367, "febrero": 1.0810, "marzo": 1.0874, "abril": 0.9355,
765         "mayo": 0.9314, "junio": 0.9736, "julio": 0.9528, "agosto": 0.9488,
766         "septiembre": 0.9841, "octubre": 1.0038, "noviembre": 1.1007,
"dicembre": 1.1109
767     },
768     "RUMICHACA": {
769         "enero": 1.0807, "febrero": 1.0122, "marzo": 1.0302, "abril": 1.0888,
770         "mayo": 1.0366, "junio": 1.0099, "julio": 0.9690, "agosto": 0.9188,
771         "septiembre": 0.9654, "octubre": 0.9569, "noviembre": 1.0013,
"dicembre": 0.9972
772     },
773     "SAN ANTON": {
774         "enero": 1.1667, "febrero": 1.0854, "marzo": 1.0365, "abril": 1.0303,
775         "mayo": 1.0316, "junio": 1.0426, "julio": 1.0281, "agosto": 0.9802,
776         "septiembre": 0.9333, "octubre": 0.9279, "noviembre": 0.9890,
"dicembre": 0.9485
777     },
778     "SAN GABAN": {
779         "enero": 1.1273, "febrero": 1.0635, "marzo": 1.1206, "abril": 1.0840,
780         "mayo": 1.1029, "junio": 1.0740, "julio": 1.1605, "agosto": 0.8976,

```



```
781         "septiembre": 0.8572, "octubre": 0.9080, "noviembre": 0.9421,  
782         "diciembre": 1.0128  
783     },  
784     "SAN LORENZO": {  
785         "enero": 1.6588, "febrero": 1.5639, "marzo": 1.3338, "abril": 1.4442,  
786         "mayo": 1.0532, "junio": 1.0350, "julio": 0.9249, "agosto": 0.9112,  
787         "septiembre": 0.8771, "octubre": 0.8164, "noviembre": 0.8616,  
788         "diciembre": 0.9565  
789     },  
790     "SANTA LUCIA": {  
791         "enero": 1.0356, "febrero": 1.0069, "marzo": 1.0716, "abril": 1.0685,  
792         "mayo": 1.0621, "junio": 1.0856, "julio": 1.0229, "agosto": 0.9381,  
793         "septiembre": 0.9411, "octubre": 0.9138, "noviembre": 0.9453,  
794         "diciembre": 0.9392  
795     },  
796     "SAYLLA": {  
797         "enero": 1.0693, "febrero": 1.0277, "marzo": 1.0743, "abril": 1.0610,  
798         "mayo": 1.0379, "junio": 1.0317, "julio": 0.9662, "agosto": 0.9474,  
799         "septiembre": 0.9472, "octubre": 0.9579, "noviembre": 0.9470,  
800         "diciembre": 0.9774  
801     },  
802     "SERPENTIN DE PASAMAYO": {  
803         "enero": 0.9940, "febrero": 1.0077, "marzo": 1.0623, "abril": 1.0455,  
804         "mayo": 1.0254, "junio": 1.0171, "julio": 1.0086, "agosto": 0.9839,  
805         "septiembre": 0.9921, "octubre": 0.9743, "noviembre": 0.9582,  
806         "diciembre": 0.9488  
807     },  
808     "SICUYANI": {  
809         "enero": 1.1244, "febrero": 1.0164, "marzo": 1.0598, "abril": 1.0657,  
810         "mayo": 1.0985, "junio": 1.1286, "julio": 1.0861, "agosto": 0.9971,  
811         "septiembre": 0.9910, "octubre": 0.9077, "noviembre": 0.9308,  
812         "diciembre": 0.9614  
813     },  
814     "SOCOS": {  
815         "enero": 1.1347, "febrero": 1.0494, "marzo": 1.0018, "abril": 1.2878,  
816         "mayo": 1.0911, "junio": 1.0528, "julio": 1.0231, "agosto": 0.9946,  
817         "septiembre": 1.0306, "octubre": 1.0022, "noviembre": 1.0632,  
818         "diciembre": 0.9010  
819     },  
820     "TAMBOGRANDE": {  
821         "enero": 1.1347, "febrero": 1.0494, "marzo": 1.0018, "abril": 1.2878,  
822         "mayo": 1.5102, "junio": 1.2768, "julio": 1.0231, "agosto": 1.0946,  
823         "septiembre": 1.0306, "octubre": 1.0022, "noviembre": 1.0632,  
824         "diciembre": 0.9010  
825     },  
826     "TOMASIRI": {  
827         "enero": 0.9789, "febrero": 0.9106, "marzo": 0.9995, "abril": 1.0539,  
828         "mayo": 1.0515, "junio": 1.0702, "julio": 1.0580, "agosto": 1.0075,  
829         "septiembre": 0.9691, "octubre": 1.0064, "noviembre": 1.0067,  
830         "diciembre": 0.9966  
831     },  
832     "TUNAN": {  
833         "enero": 1.0648, "febrero": 1.0578, "marzo": 1.0849, "abril": 1.0647,  
834         "mayo": 1.0341, "junio": 0.9359, "julio": 0.9355, "agosto": 0.9285,
```



```

826         "septiembre": 0.9763, "octubre": 0.9641, "noviembre": 1.0216,
      "diciembre": 1.0031
827     },
828     "UNION-CONGRESO": {
829         "enero": 1.1522, "febrero": 1.1289, "marzo": 1.0613, "abril": 1.0342,
830         "mayo": 0.9994, "junio": 0.9916, "julio": 0.9577, "agosto": 0.9143,
831         "septiembre": 0.9085, "octubre": 0.9675, "noviembre": 1.0861,
      "diciembre": 0.9950
832     },
833     "UTCUBAMBA": {
834         "enero": 1.1577, "febrero": 1.0364, "marzo": 1.0667, "abril": 1.0631,
835         "mayo": 1.0240, "junio": 0.9908, "julio": 0.9829, "agosto": 0.9364,
836         "septiembre": 0.9605, "octubre": 0.9802, "noviembre": 0.9216,
      "diciembre": 1.0143
837     },
838     "VARIANTE DE PASAMAYO": {
839         "enero": 1.1672, "febrero": 0.9188, "marzo": 1.0160, "abril": 1.0768,
840         "mayo": 0.9922, "junio": 0.9811, "julio": 1.0130, "agosto": 0.9895,
841         "septiembre": 1.0586, "octubre": 1.0114, "noviembre": 0.9087,
      "diciembre": 0.9711
842     },
843     "VARIANTE DE UCHUMAYO": {
844         "enero": 1.0133, "febrero": 0.9729, "marzo": 1.0446, "abril": 1.0703,
845         "mayo": 1.0573, "junio": 1.0429, "julio": 1.0271, "agosto": 0.9797,
846         "septiembre": 0.9635, "octubre": 0.9527, "noviembre": 0.9803,
      "diciembre": 0.9711
847     },
848     "VESIQUE": {
849         "enero": 1.0255, "febrero": 0.9896, "marzo": 1.0600, "abril": 1.0910,
850         "mayo": 1.0805, "junio": 1.0745, "julio": 1.0534, "agosto": 1.0136,
851         "septiembre": 1.0300, "octubre": 0.9468, "noviembre": 0.9231,
      "diciembre": 0.8962
852     },
853     "VIRU": {
854         "enero": 1.0337, "febrero": 0.9986, "marzo": 1.0637, "abril": 1.0876,
855         "mayo": 1.0879, "junio": 1.0847, "julio": 1.0635, "agosto": 1.0165,
856         "septiembre": 1.1260, "octubre": 0.9199, "noviembre": 0.8931,
      "diciembre": 0.8927
857     },
858     "YAUCA": {
859         "enero": 1.0331, "febrero": 0.9871, "marzo": 1.0294, "abril": 1.0337,
860         "mayo": 1.0300, "junio": 1.0338, "julio": 1.0805, "agosto": 0.9767,
861         "septiembre": 0.9747, "octubre": 1.0010, "noviembre": 0.9281,
      "diciembre": 0.9422
862     }
863 }
864
865 # PESOS POR EJE PARA CADA TIPO DE VEHÍCULO (en toneladas)
866 self.pesos_por_eje = {
867     # VEHÍCULOS LIGEROS
868     "A": [1.0],          # Automóvil: 1 eje simple
869     "SW": [1.0],         # Station Wagon: 1 eje simple
870     "PU": [2.0],         # Pickup: 1 eje simple
871     "PA": [2.0],         # Panel: 1 eje simple
872     "RC": [3.0],         # Rural Combi: 1 eje simple

```

```

873 "M": [1.0, 4.0], # Micro: eje delantero + trasero
874
875 # BUSES
876 "B2E": [7.0, 11.0], # Bus 2 ejes: delantero + trasero
877 "B3E": [7.0, 16.0], # Bus 3 ejes: delantero + trasero tandem
878 "B4E": [7.0, 7.0, 16.0], # Bus 4 ejes
879
880 # CAMIONES
881 "C2E": [7.0, 11.0], # Camión 2 ejes
882 "C3E": [7.0, 18.0], # Camión 3 ejes
883 "2S1": [7.0, 11.0, 11.0], # Semirremolque
884 "2S2": [7.0, 11.0, 18.0],
885 "2S3": [7.0, 11.0, 25.0],
886 "3S1": [7.0, 18.0, 11.0],
887 "3S2": [7.0, 18.0, 18.0],
888 "3S3": [7.0, 18.0, 25.0],
889 "2T2": [7.0, 11.0, 11.0, 11.0], # Tandem
890 "2T3": [7.0, 11.0, 11.0, 18.0],
891 "3T2": [7.0, 18.0, 11.0, 11.0],
892 "3T3": [7.0, 18.0, 11.0, 18.0],
893 "4E": [7.0, 11.0, 11.0, 11.0] # 4 ejes genérico
894 }
895
896 # =====
897 # TASAS DE CRECIMIENTO POR DEPARTAMENTO - COMPLETAS (24 DEPARTAMENTOS)
898 # =====
899 self.tasas_crecimiento = {
900     "amazonas": {"ligeros": 1.12, "pesados": 2.69},
901     "ancash": {"ligeros": 2.33, "pesados": 1.49},
902     "apurimac": {"ligeros": 1.13, "pesados": 4.50},
903     "arequipa": {"ligeros": 2.90, "pesados": 2.97},
904     "ayacucho": {"ligeros": 1.83, "pesados": 2.90},
905     "cajamarca": {"ligeros": 3.05, "pesados": 1.45},
906     "cusco": {"ligeros": 2.77, "pesados": 3.07},
907     "huancavelica": {"ligeros": 1.17, "pesados": 2.00},
908     "huanuco": {"ligeros": 1.94, "pesados": 3.03},
909     "ica": {"ligeros": 1.10, "pesados": 2.62},
910     "junin": {"ligeros": 2.04, "pesados": 2.84},
911     "la libertad": {"ligeros": 2.61, "pesados": 2.21},
912     "lambayeque": {"ligeros": 2.40, "pesados": 2.54},
913     "lima provincias": {"ligeros": 2.61, "pesados": 2.34},
914     "lima": {"ligeros": 2.61, "pesados": 2.34}, # Alias para Lima Provincias
915     "loreto": {"ligeros": 0.12, "pesados": 1.48},
916     "madre de dios": {"ligeros": 2.22, "pesados": 1.38},
917     "moquegua": {"ligeros": 1.97, "pesados": 0.58},
918     "pasco": {"ligeros": 1.72, "pesados": 0.39},
919     "piura": {"ligeros": 2.38, "pesados": 2.37},
920     "puno": {"ligeros": 2.47, "pesados": 2.58},
921     "san martin": {"ligeros": 2.11, "pesados": 2.88},
922     "tacna": {"ligeros": 2.13, "pesados": 2.36},
923     "tumbes": {"ligeros": 0.81, "pesados": 2.40},
924     "ucayali": {"ligeros": 1.73, "pesados": 2.25}
925 }
926

```



```

927 # FACTORES DIRECCIONALES Y DE CARRIL
928 self.factor_direccion_carril = {
929     # (num_calzadas, num_sentidos, carriles_por_sentido): {"fd": valor, "fc":
valor}
930     (1, 1, 1): {"fd": 1.00, "fc": 1.00}, # 1 calzada, 1 sentido, 1 carril
931     (1, 1, 2): {"fd": 1.00, "fc": 0.80}, # 1 calzada, 1 sentido, 2 carriles
932     (1, 1, 3): {"fd": 1.00, "fc": 0.60},
933     (1, 1, 4): {"fd": 1.00, "fc": 0.50},
934     (1, 2, 1): {"fd": 0.50, "fc": 1.00}, # 1 calzada, 2 sentidos, 1 carril c/u
935     (1, 2, 2): {"fd": 0.50, "fc": 0.80},
936     (1, 2, 3): {"fd": 0.50, "fc": 0.60},
937     (1, 2, 4): {"fd": 0.50, "fc": 0.50},
938     (2, 2, 1): {"fd": 0.50, "fc": 1.00}, # 2 calzadas, 2 sentidos
939     (2, 2, 2): {"fd": 0.50, "fc": 0.80},
940     (2, 2, 3): {"fd": 0.50, "fc": 0.60},
941     (2, 2, 4): {"fd": 0.50, "fc": 0.50}
942 }
943
944 # LISTA DE ESTACIONES DISPONIBLES (ordenadas alfabéticamente)
945 self.lista_estaciones = sorted(self.fce_ligeros.keys())
946
947 def clasificar_vehiculo(self, tipo):
948     """Clasificar vehículo en ligero o pesado"""
949     ligeros = ["A", "SW", "PU", "PA", "RC", "M"]
950     return "ligeros" if tipo in ligeros else "pesados"
951
952 def calcular_fce(self, tipo_vehiculo, peaje="SOCOS", mes="junio"):
953     """Calcular Factor de Corrección Estacional"""
954     clasificacion = self.clasificar_vehiculo(tipo_vehiculo)
955
956     # Normalizar nombre de estación
957     peaje_upper = peaje.upper()
958
959     if clasificacion == "ligeros":
960         return self.fce_ligeros.get(peaje_upper, {}).get(mes.lower(), 1.0)
961     else:
962         return self.fce_pesados.get(peaje_upper, {}).get(mes.lower(), 1.0)
963
964 def calcular_imd_7_dias(self, conteos_por_dia):
965     """
966     CALCULAR IMD CORRECTO: Suma de cada tipo en 7 días dividido entre 7
967     conteos_por_dia: dict {"lunes": {tipo: cantidad}, "martes": {...}, ...}
968     """
969     # Obtener todos los tipos de vehículos detectados
970     todos_tipos = set()
971     for conteo_dia in conteos_por_dia.values():
972         todos_tipos.update(conteo_dia.keys())
973
974     imd_por_tipo = {}
975
976     for tipo in todos_tipos:
977         # SUMAR TODOS LOS CONTEOS DEL TIPO EN LOS 7 DÍAS
978         suma_total_tipo = 0
979

```



```

980     for día, conteo_día in conteos_por_día.items():
981         if conteo_día: # Si hay datos para este día
982             suma_total_tipo += conteo_día.get(tipo, 0)
983
984     # IMD = SUMA TOTAL / 7 (SIEMPRE DIVIDIR ENTRE 7)
985     imd_por_tipo[tipo] = suma_total_tipo / 7.0
986
987     return imd_por_tipo
988
989 def calcular_imda(self, imd, fce):
990     """Calcular IMDA = IMD x FCE (redondeado)"""
991     imda = imd * fce
992     return round(imda + 0.5) # Redondeo hacia arriba desde 0.5
993
994 def calcular_fvp(self, tipo_vehiculo, tipo_pavimento="rígido"):
995     """Calcular Factor de Vehículo Pesado (FVP)"""
996     if tipo_vehiculo not in self.pesos_por_eje:
997         return 0.001 # Factor mínimo para vehículos no definidos
998
999     pesos = self.pesos_por_eje[tipo_vehiculo]
1000
1001     # Configurar fórmulas según tipo de pavimento
1002     if tipo_pavimento == "rígido":
1003         # PAVIMENTOS RÍGIDOS
1004         formulas = {
1005             "simple_simple": lambda p: (p / 6.6) ** 4.1,
1006             "simple_doble": lambda p: (p / 8.2) ** 4.1,
1007             "tandem_1": lambda p: (p / 13.0) ** 4.1,
1008             "tandem_2": lambda p: (p / 13.3) ** 4.1,
1009             "tridem_1": lambda p: (p / 16.6) ** 4.0,
1010             "tridem_2": lambda p: (p / 17.5) ** 4.0
1011         }
1012     else:
1013         # PAVIMENTOS FLEXIBLES
1014         formulas = {
1015             "simple_simple": lambda p: (p / 6.6) ** 4.0,
1016             "simple_doble": lambda p: (p / 8.2) ** 4.0,
1017             "tandem_1": lambda p: (p / 14.8) ** 4.0,
1018             "tandem_2": lambda p: (p / 15.1) ** 4.0,
1019             "tridem_1": lambda p: (p / 20.7) ** 3.9,
1020             "tridem_2": lambda p: (p / 21.8) ** 3.9
1021         }
1022
1023     # Calcular FVP según configuración del vehículo
1024     fvp_total = 0.0
1025
1026     # VEHÍCULOS LIGEROS - Factor muy bajo
1027     if tipo_vehiculo in ["A", "SW", "PU", "PA", "RC"]:
1028         fvp_total = 0.001
1029
1030     # MICRO (M) - Eje simple + eje simple
1031     elif tipo_vehiculo == "M":
1032         fvp_total = formulas["simple_simple"](pesos[0]) + formulas["simple_simple"]
(pesos[1])

```

```

1033
1034     # BUSES
1035     elif tipo_vehiculo == "B2E": # 7t + 11t
1036         fvp_total = formulas["simple_simple"](pesos[0]) + formulas["simple_doble"]
1037         (pesos[1])
1038     elif tipo_vehiculo == "B3E": # 7t + 16t
1039         fvp_total = formulas["simple_simple"](pesos[0]) + formulas["tandem_2"]
1040         (pesos[1])
1041
1042     # CAMIONES C2E, C3E
1043     elif tipo_vehiculo == "C2E": # 7t + 11t
1044         fvp_total = formulas["simple_simple"](pesos[0]) + formulas["simple_doble"]
1045         (pesos[1])
1046     elif tipo_vehiculo == "C3E": # 7t + 18t
1047         fvp_total = formulas["simple_simple"](pesos[0]) + formulas["tandem_1"]
1048         (pesos[1])
1049
1050     # SEMIRREMOLQUES 2S
1051     elif tipo_vehiculo in ["2S1", "2S2", "2S3"]:
1052         fvp_total = (formulas["simple_simple"](pesos[0]) +
1053                     formulas["simple_simple"](pesos[1]) +
1054                     formulas["simple_doble"](pesos[2]))
1055
1056     # SEMIRREMOLQUES 3S
1057     elif tipo_vehiculo in ["3S1", "3S2", "3S3"]:
1058         fvp_total = (formulas["simple_simple"](pesos[0]) +
1059                     formulas["tandem_1"](pesos[1]) +
1060                     formulas["simple_doble"](pesos[2]))
1061
1062     # TÁNDEM
1063     elif tipo_vehiculo in ["2T2", "2T3"]:
1064         fvp_total = (formulas["simple_simple"](pesos[0]) +
1065                     formulas["simple_simple"](pesos[1]) +
1066                     formulas["simple_simple"](pesos[2]) +
1067                     formulas["simple_doble"](pesos[3]))
1068
1069     elif tipo_vehiculo in ["3T2", "3T3"]:
1070         fvp_total = (formulas["simple_simple"](pesos[0]) +
1071                     formulas["tandem_1"](pesos[1]) +
1072                     formulas["simple_simple"](pesos[2]) +
1073                     formulas["simple_doble"](pesos[3]))
1074
1075     else:
1076         # Vehículo genérico - calcular por ejes
1077         for peso in pesos:
1078             if peso <= 8:
1079                 fvp_total += formulas["simple_simple"](peso)
1080             else:
1081                 fvp_total += formulas["simple_doble"](peso)
1082
1083     return max(fvp_total, 0.001) # Mínimo 0.001
1084
1085 def calcular_fd_fc(self, num_calzadas=1, num_sentidos=2, carriles_por_sentido=1):
1086     """Calcular factores direccionales y de carril"""

```



```

1083     config = (num_calzadas, num_sentidos, carriles_por_sentido)
1084     factores = self.factoros_direccion_carril.get(config, {"fd": 0.5, "fc": 1.0})
1085     return factores["fd"], factores["fc"]
1086
1087 def calcular_fca(self, tipo_vehiculo, departamento="ayacucho", periodo_diseno=20):
1088     """Calcular Factor de Crecimiento Acumulado"""
1089     clasificacion = self.clasificar_vehiculo(tipo_vehiculo)
1090
1091     # Normalizar nombre del departamento
1092     departamento_lower = departamento.lower()
1093
1094     # Obtener tasa de crecimiento
1095     tasa_porcentaje = self.tasas_crecimiento.get(departamento_lower,
1096     {}).get(clasificacion, 2.0)
1097     r = tasa_porcentaje / 100.0 # Convertir a decimal
1098
1099     # Fórmula: Fca = [(1 + r)^n - 1] / r
1100     if r == 0:
1101         return periodo_diseno # Si no hay crecimiento
1102
1103     fca = ((1 + r) ** periodo_diseno - 1) / r
1104     return fca
1105
1106 def calcular_ee_dia_carril(self, imda, fd, fc, fvp, fp=1.0):
1107     """Calcular Ejes Equivalentes por día-carril"""
1108     return imda * fd * fc * fvp * fp
1109
1110 def calcular_nrep_ee(self, ee_dia_carril, fca):
1111     """Calcular Número de Repeticiones de Ejes Equivalentes"""
1112     return ee_dia_carril * fca * 365
1113
1114 # [RESTO DEL CÓDIGO PERMANECE IGUAL - SOLO SE MODIFICA LA INTERFAZ]
1115 # Copio las clases VehiculoTrackeado y ContadorVehicular sin cambios...
1116
1117 class VehiculoTrackeado:
1118     """Clase para representar un vehículo siendo trackeado"""
1119     def __init__(self, vehiculo_id, tipo_inicial, bbox, confianza, frame):
1120         self.id = vehiculo_id
1121         self.tipo_actual = self._limpiar_tipo_vehiculo(tipo_inicial)
1122         self.historial_tipos = defaultdict(int)
1123         self.historial_tipos[self.tipo_actual] = 1
1124         self.historial_posiciones = deque(maxlen=10)
1125         self.primer_deteccion = frame
1126         self.ultima_deteccion = frame
1127         self.frames_sin_detectar = 0
1128         self.bbox_actual = bbox
1129         self.confianza_maxima = confianza
1130         self.confianza_promedio = confianza
1131         self.total_detecciones = 1
1132         self.confirmado = False
1133
1134         centro = self._calcular_centro(bbox)
1135         self.historial_posiciones.append((centro, frame))

```



```
1136 def _limpiar_tipo_vehiculo(self, tipo):
1137     """Limpiar pero mantener denominaciones originales del modelo"""
1138     if not tipo or not isinstance(tipo, str):
1139         return "VEHICULO"
1140
1141     caracteres_no_deseados = ['?', '??', '??', ' ', '\x00', '\n', '\r', '\t']
1142
1143     tipo_limpio = str(tipo)
1144     for char in caracteres_no_deseados:
1145         tipo_limpio = tipo_limpio.replace(char, '')
1146
1147     tipo_limpio = tipo_limpio.strip()
1148
1149     if tipo_limpio and len(tipo_limpio) >= 1:
1150         return tipo_limpio
1151
1152     return "VEHICULO"
1153
1154 def _calcular_centro(self, bbox):
1155     return ((bbox[0] + bbox[2]) // 2, (bbox[1] + bbox[3]) // 2)
1156
1157 def actualizar(self, nuevo_tipo, bbox, confianza, frame):
1158     self.ultima_deteccion = frame
1159     self.frames_sin_detectar = 0
1160     self.bbox_actual = bbox
1161     self.total_detecciones += 1
1162
1163     nuevo_tipo_limpio = self._limpiar_tipo_vehiculo(nuevo_tipo)
1164     self.historial_tipos[nuevo_tipo_limpio] += 1
1165
1166     if confianza > self.confianza_maxima:
1167         self.confianza_maxima = confianza
1168         self.tipo_actual = nuevo_tipo_limpio
1169
1170     self.confianza_promedio = (self.confianza_promedio * (self.total_detecciones - 1)
+ confianza) / self.total_detecciones
1171
1172     tipo_mas_frecuente = max(self.historial_tipos.items(), key=lambda x: x[1])[0]
1173     if self.historial_tipos[tipo_mas_frecuente] >= 3:
1174         self.tipo_actual = tipo_mas_frecuente
1175
1176     centro = self._calcular_centro(bbox)
1177     self.historial_posiciones.append((centro, frame))
1178
1179     if self.total_detecciones >= 3 and self.confianza_maxima >= 0.6:
1180         self.confirmado = True
1181
1182 def incrementar_frames_sin_detectar(self):
1183     self.frames_sin_detectar += 1
1184
1185 def esta_perdido(self, max_frames_perdido=10):
1186     return self.frames_sin_detectar > max_frames_perdido
1187
1188 def distancia_a(self, bbox):
```

```

1189     centro_actual = self._calcular_centro(self.bbox_actual)
1190     centro_nuevo = self._calcular_centro(bbox)
1191     dx = centro_actual[0] - centro_nuevo[0]
1192     dy = centro_actual[1] - centro_nuevo[1]
1193     return math.sqrt(dx*dx + dy*dy)
1194
1195 class ContadorVehicular:
1196     """Motor de procesamiento de vehículos - 100% CPU GARANTIZADO"""
1197     def __init__(self, ruta_modelo="best_vehicular_model.pt"):
1198         print("\n" + "="*60)
1199         print("  CARGANDO MODELO YOLO EN CPU...")
1200         print("="*60)
1201
1202         # ASEGURAR QUE NO SE USE GPU
1203         os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
1204
1205         try:
1206             # Cargar modelo con configuración explícita de CPU
1207             self.modelo = YOLO(ruta_modelo)
1208
1209             # Forzar modelo a CPU de múltiples formas
1210             self.modelo.to('cpu')
1211
1212             # Si el modelo tiene un atributo 'model', también forzarlo a CPU
1213             if hasattr(self.modelo, 'model'):
1214                 try:
1215                     self.modelo.model.cpu()
1216                     self.modelo.model.float() # Precisión completa para CPU
1217                     print(" Modelo interno configurado en CPU")
1218                 except:
1219                     pass
1220
1221             # Verificar que torch también esté en CPU
1222             try:
1223                 import torch
1224                 torch.set_default_device('cpu')
1225                 torch.set_default_dtype(torch.float32)
1226                 print(" PyTorch configurado para CPU por defecto")
1227             except:
1228                 pass
1229
1230             print(f" Modelo cargado: {ruta_modelo}")
1231             print(f" Clases detectables: {len(self.modelo.names)}")
1232
1233         except Exception as e:
1234             print(f" Error cargando modelo: {e}")
1235             raise
1236
1237         self.vehiculos_trackeados = {}
1238         self.proximo_id = 1
1239         self.frame_actual = 0
1240         self.historial_detecciones = deque(maxlen=5)
1241
1242         # Configuración optimizada para CPU

```

```

1243     self.max_distancia_tracking = 100
1244     self.frames_umbral_consistencia = 2
1245     self.max_frames_perdido = 15
1246     self.umbral_confianza = 0.6
1247
1248     self.conteo_unico_por_clase = defaultdict(int)
1249     self.vehiculos_confirmados = []
1250     self.vehiculos_confirmados_ids = set()
1251
1252     np.random.seed(42)
1253     self.colores = {}
1254     for clase_id in self.modelo.names.keys():
1255         self.colores[clase_id] = tuple(map(int, np.random.randint(50, 255, 3)))
1256
1257     print(" ContadorVehicular inicializado correctamente")
1258     print("="*60 + "\n")
1259
1260     def resetear_contador(self):
1261         """Resetear completamente el contador para procesar nuevo video"""
1262         self.vehiculos_trackeados = {}
1263         self.proximo_id = 1
1264         self.frame_actual = 0
1265         self.historial_detecciones = deque(maxlen=5)
1266         self.conteo_unico_por_clase = defaultdict(int)
1267         self.vehiculos_confirmados = []
1268         self.vehiculos_confirmados_ids = set()
1269
1270     def filtrar_detecciones_temporal(self, detecciones_frame, detecciones_historicas,
frames_umbral=2):
1271         detecciones_filtradas = []
1272
1273         for deteccion in detecciones_frame:
1274             centro = ((deteccion['bbox'][0] + deteccion['bbox'][2]) // 2,
1275                     (deteccion['bbox'][1] + deteccion['bbox'][3]) // 2)
1276
1277             coincidencias = 0
1278             for frame_historico in detecciones_historicas:
1279                 for det_historica in frame_historico:
1280                     centro_hist = ((det_historica['bbox'][0] + det_historica['bbox'][2])
// 2,
1281                                 (det_historica['bbox'][1] + det_historica['bbox'][3])
// 2)
1282
1283                     distancia = math.sqrt((centro[0] - centro_hist[0])**2 +
1284                                           (centro[1] - centro_hist[1])**2)
1285
1286                     if distancia < self.max_distancia_tracking and deteccion['tipo'] ==
det_historica['tipo']:
1287                         coincidencias += 1
1288                         break
1289
1290             if coincidencias >= frames_umbral - 1:
1291                 detecciones_filtradas.append(deteccion)
1292

```



```

1293         return detecciones_filtradas
1294
1295     def asignar_id_vehiculo(self, detecciones_filtradas):
1296         vehiculos_actualizados = set()
1297
1298         for deteccion in detecciones_filtradas:
1299             bbox = deteccion['bbox']
1300             mejor_match = None
1301             menor_distancia = float('inf')
1302
1303             for vehiculo_id, vehiculo in self.vehiculos_trackeados.items():
1304                 if vehiculo_id in vehiculos_actualizados:
1305                     continue
1306
1307                 distancia = vehiculo.distancia_a(bbox)
1308                 if distancia < self.max_distancia_tracking and distancia <
menor_distancia:
1309                     mejor_match = vehiculo_id
1310                     menor_distancia = distancia
1311
1312                 if mejor_match:
1313                     vehiculo = self.vehiculos_trackeados[mejor_match]
1314                     vehiculo.actualizar(deteccion['tipo'], bbox, deteccion['confianza'],
self.frame_actual)
1315                     deteccion['id'] = mejor_match
1316                     vehiculos_actualizados.add(mejor_match)
1317                 else:
1318                     nuevo_id = self.proximo_id
1319                     self.proximo_id += 1
1320                     nuevo_vehiculo = VehiculoTrackeado(
1321                         nuevo_id, deteccion['tipo'], bbox,
1322                         deteccion['confianza'], self.frame_actual
1323                     )
1324                     self.vehiculos_trackeados[nuevo_id] = nuevo_vehiculo
1325                     deteccion['id'] = nuevo_id
1326
1327         for vehiculo_id, vehiculo in list(self.vehiculos_trackeados.items()):
1328             if vehiculo_id not in vehiculos_actualizados:
1329                 vehiculo.incrementar_frames_sin_detectar()
1330                 if vehiculo.esta_perdido(self.max_frames_perdido):
1331                     if vehiculo.confirmado:
1332                         self._confirmar_vehiculo_final(vehiculo)
1333                         del self.vehiculos_trackeados[vehiculo_id]
1334
1335     def _confirmar_vehiculo_final(self, vehiculo):
1336         if vehiculo.confirmado and vehiculo.id not in self.vehiculos_confirmados_ids:
1337             vehiculo_final = {
1338                 'id': vehiculo.id,
1339                 'tipo_final': vehiculo.tipo_actual,
1340                 'total_detecciones': vehiculo.total_detecciones,
1341                 'confianza_maxima': vehiculo.confianza_maxima,
1342                 'confianza_promedio': vehiculo.confianza_promedio,
1343                 'primer_frame': vehiculo.primer_deteccion,
1344                 'ultimo_frame': vehiculo.ultima_deteccion,

```

```

1345         'historial_tipos': dict(vehiculo.historial_tipos)
1346     }
1347     self.vehiculos_confirmados.append(vehiculo_final)
1348     self.vehiculos_confirmados_ids.add(vehiculo.id)
1349     self.conteo_unico_por_clase[vehiculo.tipo_actual] += 1
1350
1351 def procesar_frame(self, frame):
1352     """Procesar frame con YOLO en CPU"""
1353     self.frame_actual += 1
1354
1355     try:
1356         # MÉTODO 1: predict() con parámetros explícitos
1357         resultados = self.modelo.predict(
1358             source=frame,
1359             verbose=False,
1360             conf=self.umbral_confianza,
1361             iou=0.45,
1362             max_det=100,
1363             device='cpu', # FORZAR CPU
1364             half=False,   # NO usar half precision
1365             stream=False, # NO usar streaming
1366             save=False,   # NO guardar
1367             show=False    # NO mostrar
1368         )
1369     except Exception as e1:
1370         try:
1371             # MÉTODO 2: __call__() directo
1372             resultados = self.modelo(
1373                 source=frame,
1374                 verbose=False,
1375                 device='cpu'
1376             )
1377     except Exception as e2:
1378         print(f"❌ Error en predicción: {e1} | {e2}")
1379         return []
1380
1381     detecciones_frame = []
1382
1383     try:
1384         for r in resultados:
1385             if r.bboxes is not None and len(r.bboxes) > 0:
1386                 for box in r.bboxes:
1387                     try:
1388                         # Intentar obtener datos del tensor
1389                         xyxy = box.xyxy[0]
1390
1391                         # Convertir a numpy de forma segura
1392                         if hasattr(xyxy, 'cpu'):
1393                             xyxy = xyxy.cpu().numpy()
1394                         else:
1395                             xyxy = xyxy.numpy() if hasattr(xyxy, 'numpy') else xyxy
1396
1397                         x1, y1, x2, y2 = map(int, xyxy)
1398

```

```

1399         # Obtener confianza
1400         conf = box.conf[0]
1401         if hasattr(conf, 'cpu'):
1402             conf = conf.cpu().item()
1403         else:
1404             conf = float(conf)
1405
1406         # Obtener clase
1407         cls = box.cls[0]
1408         if hasattr(cls, 'cpu'):
1409             cls = int(cls.cpu().item())
1410         else:
1411             cls = int(cls)
1412
1413         confianza = float(conf)
1414         clase_id = int(cls)
1415
1416     except Exception as e:
1417         print(f"⚠ Error procesando box: {e}")
1418         continue
1419
1420     tipo_vehiculo = "VEHICULO"
1421
1422     try:
1423         if clase_id in self.modelo.names:
1424             tipo_raw = self.modelo.names[clase_id]
1425
1426             if tipo_raw and isinstance(tipo_raw, str):
1427                 tipo_limpio = str(tipo_raw)
1428                 caracteres_no_deseados = ['?', '??', '??', ' ',
'\x00', '\n', '\r', '\t']
1429
1430                 for char in caracteres_no_deseados:
1431                     tipo_limpio = tipo_limpio.replace(char, '')
1432
1433                 tipo_limpio = tipo_limpio.strip()
1434
1435                 if tipo_limpio and len(tipo_limpio) >= 1:
1436                     tipo_vehiculo = tipo_limpio
1437
1438     except Exception:
1439         tipo_vehiculo = "VEHICULO"
1440
1441     deteccion = {
1442         'tipo': tipo_vehiculo,
1443         'confianza': confianza,
1444         'bbox': [x1, y1, x2, y2],
1445         'clase_id': clase_id
1446     }
1447     detecciones_frame.append(deteccion)
1448
1449 except Exception as e:
1450     print(f"⚠ Error procesando resultados: {e}")
1451

```



```

1452         detecciones_filtradas = self.filtrar_detecciones_temporal(
1453             detecciones_frame, list(self.historial_detecciones),
1454             self.frames_umbral_consistencia
1455         )
1456
1457         self.asignar_id_vehiculo(detecciones_filtradas)
1458         self.historial_detecciones.append(detecciones_frame)
1459
1460         return detecciones_filtradas
1461
1462     def obtener_conteo_final(self):
1463         """Obtener conteo final del video procesado"""
1464         # Confirmar vehículos restantes
1465         for vehiculo in self.vehiculos_trackeados.values():
1466             if vehiculo.confirmado:
1467                 self._confirmar_vehiculo_final(vehiculo)
1468
1469         return dict(self.conteo_unico_por_clase)
1470
1471     def finalizar_analisis(self):
1472         for vehiculo in self.vehiculos_trackeados.values():
1473             if vehiculo.confirmado:
1474                 self._confirmar_vehiculo_final(vehiculo)
1475
1476         return {
1477             'total_vehiculos': len(self.vehiculos_confirmados),
1478             'conteo_por_clase': dict(self.conteo_unico_por_clase),
1479             'vehiculos_detallados': self.vehiculos_confirmados
1480         }
1481
1482     class InterfazSistemaEjesEquivalentes:
1483         """Interfaz optimizada con selector de estación de conteo"""
1484
1485         def __init__(self):
1486             self.root = tk.Tk()
1487             self.root.title("SISTEMA OPTIMIZADO DE EJES EQUIVALENTES v6.0 - TODAS LAS
ESTACIONES")
1488
1489             # CONFIGURACIÓN MEJORADA PARA CUALQUIER RESOLUCIÓN
1490             self.configurar_ventana_adaptativa()
1491
1492             self.root.configure(bg='#2c3e50')
1493
1494             # Variables principales
1495             self.modelo_cargado = False
1496             self.contador = None
1497             self.procesando = False
1498
1499             # CALCULADORA CON TODAS LAS ESTACIONES Y DEPARTAMENTOS
1500             self.calculadora = CalculadorEjesEquivalentes()
1501
1502             # VARIABLES PARA 7 DÍAS
1503             self.videos_por_dia = {

```

```

1504         "lunes": None,
1505         "martes": None,
1506         "miercoles": None,
1507         "jueves": None,
1508         "viernes": None,
1509         "sabado": None,
1510         "domingo": None
1511     }
1512     self.conteos_por_dia = {
1513         "lunes": {},
1514         "martes": {},
1515         "miercoles": {},
1516         "jueves": {},
1517         "viernes": {},
1518         "sabado": {},
1519         "domingo": {}
1520     }
1521     self.estados_procesamiento = {dia: "pendiente" for dia in
self.videos_por_dia.keys()}
1522
1523     # Variables para ejes equivalentes
1524     self.imd_calculado = {}
1525     self.ejes_equivalentes_resultado = None
1526
1527     # Variables para video
1528     self.video_label = None
1529     self.cap = None
1530     self.out = None
1531     self.frame_actual = 0
1532     self.total_frames = 0
1533
1534     # VARIABLES PARA CONTROL DE VELOCIDAD (FPS FIJO)
1535     self.target_fps = 30 # FPS fijo para reproducción
1536     self.frame_time = 1.0 / self.target_fps # Tiempo por frame
1537     self.last_frame_time = 0
1538
1539     # VARIABLES PARA EXPORTACIÓN DE VIDEO
1540     self.exportar_videos_procesados = tk.BooleanVar(value=True)
1541     self.carpeta_videos_procesados = None
1542
1543     # Configurar estilo y crear interfaz
1544     self.configurar_estilos()
1545     self.crear_interfaz()
1546
1547     # Cargar modelo automáticamente
1548     self.cargar_modelo()
1549
1550     def configurar_ventana_adaptativa(self):
1551         """CONFIGURACIÓN ADAPTATIVA AUTOMÁTICA"""
1552         # Obtener dimensiones de pantalla
1553         screen_width = self.root.winfo_screenwidth()
1554         screen_height = self.root.winfo_screenheight()
1555
1556         # CALCULADORA INTELIGENTE DE TAMAÑO DE VENTANA

```

```

1557 if screen_width >= 3840: # 4K
1558     window_width = min(1800, int(screen_width * 0.75))
1559     window_height = min(1200, int(screen_height * 0.80))
1560 elif screen_width >= 2560: # 2K/QHD
1561     window_width = min(1600, int(screen_width * 0.80))
1562     window_height = min(1000, int(screen_height * 0.82))
1563 elif screen_width >= 1920: # Full HD
1564     window_width = min(1400, int(screen_width * 0.85))
1565     window_height = min(900, int(screen_height * 0.85))
1566 elif screen_width >= 1600: # WSGA+
1567     window_width = min(1200, int(screen_width * 0.88))
1568     window_height = min(800, int(screen_height * 0.88))
1569 else: # 1366x768 y menores
1570     window_width = max(1200, int(screen_width * 0.95))
1571     window_height = max(650, int(screen_height * 0.90))
1572
1573 # Centrar ventana
1574 x = max(10, (screen_width - window_width) // 2)
1575 y = max(10, (screen_height - window_height) // 2)
1576
1577 self.root.geometry(f"{window_width}x{window_height}+{x}+{y}")
1578 self.root.minsize(1200, 650)
1579 self.root.resizable(True, True)
1580
1581 # Maximizar en pantallas pequeñas
1582 if screen_width <= 1366 or screen_height <= 768:
1583     try:
1584         self.root.state('zoomed' if os.name == 'nt' else 'normal')
1585     except:
1586         pass
1587
1588 # Guardar dimensiones
1589 self.screen_width = screen_width
1590 self.screen_height = screen_height
1591 self.window_width = window_width
1592 self.window_height = window_height
1593
1594 print(f"Pantalla detectada: {screen_width}x{screen_height}")
1595 print(f"Ventana configurada: {window_width}x{window_height}")
1596
1597 def configurar_estilos(self):
1598     """ESTILOS ADAPTATIVOS SEGÚN RESOLUCIÓN"""
1599     style = ttk.Style()
1600     style.theme_use('clam')
1601
1602     # TAMAÑOS DE FUENTE INTELIGENTES
1603     if self.screen_width >= 3840: # 4K
1604         base_font = 14
1605         title_font = 20
1606         subtitle_font = 16
1607     elif self.screen_width >= 2560: # 2K
1608         base_font = 12
1609         title_font = 18
1610         subtitle_font = 14

```



```

1611 elif self.screen_width >= 1920: # Full HD
1612     base_font = 11
1613     title_font = 16
1614     subtitle_font = 13
1615 elif self.screen_width >= 1600: # WSXGA+
1616     base_font = 10
1617     title_font = 14
1618     subtitle_font = 12
1619 else: # 1366x768 y menores
1620     base_font = 9
1621     title_font = 12
1622     subtitle_font = 10
1623
1624 # APLICAR ESTILOS
1625 style.configure('Title.TLabel',
1626                 background='#2c3e50',
1627                 foreground='#ecf0f1',
1628                 font=('Arial', title_font, 'bold'))
1629
1630 style.configure('Subtitle.TLabel',
1631                 background='#2c3e50',
1632                 foreground='#bdc3c7',
1633                 font=('Arial', subtitle_font, 'bold'))
1634
1635 style.configure('Custom.TButton',
1636                 font=('Arial', base_font, 'bold'))
1637
1638 style.configure('Success.TLabel',
1639                 background='#2c3e50',
1640                 foreground='#27ae60',
1641                 font=('Arial', base_font, 'bold'))
1642
1643 style.configure('Error.TLabel',
1644                 background='#2c3e50',
1645                 foreground='#e74c3c',
1646                 font=('Arial', base_font, 'bold'))
1647
1648 self.base_font = base_font
1649 self.title_font = title_font
1650 self.subtitle_font = subtitle_font
1651
1652 def crear_interfaz(self):
1653     """INTERFAZ CON DISTRIBUCIÓN INTELIGENTE"""
1654
1655     # TÍTULO COMPACTO
1656     frame_titulo = tk.Frame(self.root, bg='#34495e', height=80)
1657     frame_titulo.pack(fill='x', padx=5, pady=5)
1658     frame_titulo.pack_propagate(False)
1659
1660     tk.Label(frame_titulo,
1661             text="SISTEMA v6.0 DE EJES EQUIVALENTES E IMDA - TODAS LAS ESTACIONES Y
1662             DEPARTAMENTOS",
1663             font=('Arial', self.title_font, 'bold'),
1664             bg='#34495e', fg='#ecf0f1').pack(pady=2)

```

```

1664
1665         tk.Label(frame_titulo,
1666                 text="CÁLCULO AUTOMATIZADO | 78 ESTACIONES | 24 DEPARTAMENTOS |
METODOLOGÍA AASHTO",
1667                 font=('Arial', self.subtitle_font, 'bold'),
1668                 bg='#34495e', fg='#f39c12').pack()
1669
1670         tk.Label(frame_titulo,
1671                 text=f"CPU CONTROLADO | IMD Correcto | Resolución:
{self.screen_width}x{self.screen_height}",
1672                 font=('Arial', self.base_font - 1),
1673                 bg='#34495e', fg='#95a5a6').pack()
1674
1675         # NOTEBOOK PARA ORGANIZAR MEJOR EL ESPACIO
1676         notebook = ttk.Notebook(self.root)
1677         notebook.pack(fill='both', expand=True, padx=5, pady=5)
1678
1679         # TAB 1: PROCESAMIENTO
1680         tab_procesamiento = ttk.Frame(notebook)
1681         notebook.add(tab_procesamiento, text=" PROCESAMIENTO")
1682         self.crear_tab_procesamiento(tab_procesamiento)
1683
1684         # TAB 2: RESULTADOS
1685         tab_resultados = ttk.Frame(notebook)
1686         notebook.add(tab_resultados, text=" RESULTADOS")
1687         self.crear_tab_resultados(tab_resultados)
1688
1689         # TAB 3: EXPORTACIÓN
1690         tab_exportacion = ttk.Frame(notebook)
1691         notebook.add(tab_exportacion, text=" EXPORTACIÓN")
1692         self.crear_tab_exportacion(tab_exportacion)
1693
1694         # BARRA DE ESTADO
1695         self.status_bar = tk.Label(self.root,
1696                                   text="Sistema v6.0 inicializado - 78 estaciones + 24
departamentos disponibles",
1697                                   bg='#27ae60',
1698                                   fg='white',
1699                                   font=('Arial', self.base_font),
1700                                   anchor='w',
1701                                   padx=10,
1702                                   height=1)
1703         self.status_bar.pack(side='bottom', fill='x')
1704
1705         def crear_tab_procesamiento(self, parent):
1706             """TAB PARA PROCESAMIENTO DE VIDEOS"""
1707
1708             # Frame principal con distribución horizontal
1709             frame_principal = tk.Frame(parent, bg='#2c3e50')
1710             frame_principal.pack(fill='both', expand=True, padx=5, pady=5)
1711
1712             # COLUMNA 1: VIDEOS Y CONFIGURACIÓN (25%)
1713             frame_izquierda = tk.LabelFrame(frame_principal,
1714                                             text="CONFIGURACIÓN Y VIDEOS",

```



```

1715         font=('Arial', self.subtitle_font, 'bold'),
1716         bg='#34495e',
1717         fg='#ecf0f1',
1718         padx=5, pady=5)
1719     frame_izquierda.pack(side='left', fill='both', expand=False, padx=(0, 5))
1720     frame_izquierda.configure(width=int(self.window_width * 0.25))
1721
1722     # Estado del modelo
1723     tk.Label(frame_izquierda, text="ESTADO DEL MODELO:",
1724             bg='#34495e', fg='#ecf0f1', font=('Arial', self.base_font,
1725             'bold')).pack(anchor='w', pady=(0, 3))
1726
1727     self.label_modelo = tk.Label(frame_izquierda, text="🔄 Cargando...",
1728                                bg='#34495e', fg='#f39c12', font=('Arial',
1729                                self.base_font))
1730     self.label_modelo.pack(anchor='w', pady=(0, 10))
1731
1732     # CONFIGURACIÓN DE EXPORTACIÓN
1733     frame_export = tk.LabelFrame(frame_izquierda,
1734                                text="EXPORTACIÓN DE VIDEOS",
1735                                font=('Arial', self.base_font, 'bold'),
1736                                bg='#34495e', fg='#e67e22', padx=5, pady=5)
1737     frame_export.pack(fill='x', pady=(0, 10))
1738
1739     tk.Checkbutton(frame_export,
1740                   text="Exportar videos procesados",
1741                   variable=self.exportar_videos_procesados,
1742                   bg='#34495e', fg='#ecf0f1',
1743                   selectcolor='#2c3e50',
1744                   font=('Arial', self.base_font)).pack(anchor='w')
1745
1746     tk.Button(frame_export,
1747              text=" Seleccionar Carpeta",
1748              command=self.seleccionar_carpeta_salida,
1749              bg='#e67e22', fg='white',
1750              font=('Arial', self.base_font, 'bold')).pack(fill='x', pady=(5, 0))
1751
1752     self.label_carpeta = tk.Label(frame_export,
1753                                text=" Sin carpeta seleccionada",
1754                                bg='#34495e', fg='#95a5a6',
1755                                font=('Arial', self.base_font - 1),
1756                                wraplength=200)
1757     self.label_carpeta.pack(anchor='w', pady=(5, 0))
1758
1759     # VIDEOS POR DÍA CON SCROLL
1760     frame_videos = tk.LabelFrame(frame_izquierda,
1761                                text="ASIGNAR VIDEOS POR DÍA",
1762                                font=('Arial', self.base_font, 'bold'),
1763                                bg='#34495e', fg='#3498db', padx=5, pady=5)
1764     frame_videos.pack(fill='both', expand=True, pady=(0, 10))
1765
1766     # Canvas con scroll para videos
1767     canvas_videos = tk.Canvas(frame_videos, bg='#34495e', highlightthickness=0)

```



```

1766 scrollbar_videos = ttk.Scrollbar(frame_videos, orient="vertical",
command=canvas_videos.yview)
1767 self.frame_videos_scroll = tk.Frame(canvas_videos, bg='#34495e')
1768
1769 self.frame_videos_scroll.bind(
1770     "<Configure>",
1771     lambda e: canvas_videos.configure(scrollregion=canvas_videos.bbox("all"))
1772 )
1773
1774 canvas_videos.create_window((0, 0), window=self.frame_videos_scroll, anchor="nw")
1775 canvas_videos.configure(yscrollcommand=scrollbar_videos.set)
1776
1777 canvas_videos.pack(side="left", fill="both", expand=True)
1778 scrollbar_videos.pack(side="right", fill="y")
1779
1780 # Crear controles para cada día
1781 self.labels_dias = {}
1782 self.botones_dias = {}
1783 for dia in self.videos_por_dia.keys():
1784     self.crear_control_dia(dia)
1785
1786 # BOTONES DE CONTROL
1787 frame_control = tk.Frame(frame_izquierda, bg='#34495e')
1788 frame_control.pack(fill='x')
1789
1790 self.btn_procesar = tk.Button(frame_control,
1791     text="🔄 PROCESAR TODOS",
1792     command=self.procesar_todos_los_dias,
1793     bg='#27ae60', fg='white',
1794     font=('Arial', self.base_font, 'bold'),
1795     state='disabled')
1796 self.btn_procesar.pack(fill='x', pady=(0, 3))
1797
1798 self.btn_detener = tk.Button(frame_control,
1799     text="⏏ DETENER",
1800     command=self.detener_procesamiento,
1801     bg='#e74c3c', fg='white',
1802     font=('Arial', self.base_font, 'bold'),
1803     state='disabled')
1804 self.btn_detener.pack(fill='x', pady=(0, 3))
1805
1806 tk.Button(frame_control,
1807     text="🔄 RESET TODO",
1808     command=self.resetear_todo,
1809     bg='#95a5a6', fg='white',
1810     font=('Arial', self.base_font, 'bold')).pack(fill='x')
1811
1812 # COLUMNA 2: VIDEO EN TIEMPO REAL (50%)
1813 frame_centro = tk.LabelFrame(frame_principal,
1814     text="PROCESAMIENTO EN TIEMPO REAL",
1815     font=('Arial', self.subtitle_font, 'bold'),
1816     bg='#34495e', fg='#ecf0f1',
1817     padx=5, pady=5)
1818 frame_centro.pack(side='left', fill='both', expand=True, padx=5)

```

```

1819
1820 # ÁREA DE VIDEO ADAPTATIVA
1821 video_height = max(300, int(self.window_height * 0.4))
1822 self.video_label = tk.Label(frame_centro,
1823                             text="📺 Video aparecerá aquí durante procesamiento",
1824                             bg='#2c3e50', fg='#bdc3c7',
1825                             font=('Arial', self.subtitle_font),
1826                             height=video_height // 20)
1827 self.video_label.pack(fill='both', expand=True, pady=(0, 10))
1828
1829 # Información de progreso
1830 frame_progreso = tk.Frame(frame_centro, bg='#34495e')
1831 frame_progreso.pack(fill='x')
1832
1833 self.label_dia_actual = tk.Label(frame_progreso,
1834                                 text="Estado: Esperando videos",
1835                                 bg='#34495e', fg='#3498db',
1836                                 font=('Arial', self.subtitle_font, 'bold'))
1837 self.label_dia_actual.pack()
1838
1839 self.progress_video_var = tk.StringVar(value="Listo para procesar...")
1840 tk.Label(frame_progreso, textvariable=self.progress_video_var,
1841         bg='#34495e', fg='#ecf0f1',
1842         font=('Arial', self.base_font)).pack(pady=5)
1843
1844 self.progress_bar = ttk.Progressbar(frame_progreso, mode='determinate')
1845 self.progress_bar.pack(fill='x', pady=5)
1846
1847 self.label_export_status = tk.Label(frame_progreso,
1848                                     text="Exportación: Inactiva",
1849                                     bg='#34495e', fg='#95a5a6',
1850                                     font=('Arial', self.base_font))
1851 self.label_export_status.pack()
1852
1853 # COLUMNA 3: CONTEOS (25%)
1854 frame_derecha = tk.LabelFrame(frame_principal,
1855                               text="CONTEOS POR DÍA",
1856                               font=('Arial', self.subtitle_font, 'bold'),
1857                               bg='#34495e', fg='#ecf0f1',
1858                               padx=5, pady=5)
1859 frame_derecha.pack(side='right', fill='both', expand=False, padx=(5, 0))
1860 frame_derecha.configure(width=int(self.window_width * 0.25))
1861
1862 # Canvas con scroll para conteos
1863 canvas_conteos = tk.Canvas(frame_derecha, bg='#34495e', highlightthickness=0)
1864 scrollbar_conteos = ttk.Scrollbar(frame_derecha, orient="vertical",
command=canvas_conteos.yview)
1865 self.frame_conteos_scroll = tk.Frame(canvas_conteos, bg='#34495e')
1866
1867 self.frame_conteos_scroll.bind(
1868     "<Configure>",
1869     lambda e: canvas_conteos.configure(scrollregion=canvas_conteos.bbox("all"))
1870 )
1871

```



```

1872 canvas_conteos.create_window((0, 0), window=self.frame_conteos_scroll,
anchor="nw")
1873 canvas_conteos.configure(yscrollcommand=scrollbar_conteos.set)
1874
1875 canvas_conteos.pack(side="left", fill="both", expand=True)
1876 scrollbar_conteos.pack(side="right", fill="y")
1877
1878 self.actualizar_conteos_display()
1879
1880 def crear_control_dia(self, dia):
1881     """Crear control para cada día de la semana"""
1882     frame_dia = tk.LabelFrame(self.frame_videos_scroll,
1883                               text=dia.upper(),
1884                               font=('Arial', self.base_font, 'bold'),
1885                               bg='#34495e', fg='#ecf0f1',
1886                               padx=5, pady=5)
1887     frame_dia.pack(fill='x', pady=3)
1888
1889     # Estado del video
1890     self.labels_dias[dia] = tk.Label(frame_dia,
1891                                     text="📁 Sin video",
1892                                     bg='#34495e', fg='#95a5a6',
1893                                     font=('Arial', self.base_font - 1))
1894     self.labels_dias[dia].pack(anchor='w')
1895
1896     # Botón para seleccionar
1897     self.botones_dias[dia] = tk.Button(frame_dia,
1898                                       text=f"Seleccionar {dia}",
1899                                       command=lambda d=dia:
self.seleccionar_video_dia(d),
1900                                       bg='#3498db', fg='white',
1901                                       font=('Arial', self.base_font - 1, 'bold'))
1902     self.botones_dias[dia].pack(fill='x', pady=(3, 0))
1903
1904 def crear_tab_resultados(self, parent):
1905     """TAB PARA RESULTADOS Y CÁLCULOS CON SELECTOR DE ESTACIÓN"""
1906
1907     frame_principal = tk.Frame(parent, bg='#2c3e50')
1908     frame_principal.pack(fill='both', expand=True, padx=5, pady=5)
1909
1910     # COLUMNA 1: CONFIGURACIÓN (30%)
1911     frame_config = tk.LabelFrame(frame_principal,
1912                                  text="CONFIGURACIÓN DE CÁLCULO",
1913                                  font=('Arial', self.subtitle_font, 'bold'),
1914                                  bg='#34495e', fg='#f39c12',
1915                                  padx=10, pady=10)
1916     frame_config.pack(side='left', fill='both', expand=False, padx=(0, 10))
1917     frame_config.configure(width=int(self.window_width * 0.30))
1918
1919     # 🌟 NUEVO: SELECTOR DE ESTACIÓN DE CONTEO 🌟
1920     tk.Label(frame_config,
1921             text="📍 ESTACIÓN DE CONTEO:",
1922             bg='#34495e', fg='#e74c3c',
1923             font=('Arial', self.base_font + 1, 'bold')).pack(anchor='w', pady=(0, 5))

```



```

1924
1925 self.combo_estacion = ttk.Combobox(
1926     frame_config,
1927     values=self.calculadora.lista_estaciones,
1928     state="readonly",
1929     font=('Arial', self.base_font),
1930     width=30
1931 )
1932 self.combo_estacion.set("SOCOS") # Valor por defecto
1933 self.combo_estacion.pack(fill='x', pady=(0, 15))
1934
1935 # Separador visual
1936 tk.Frame(frame_config, bg='#95a5a6', height=2).pack(fill='x', pady=10)
1937
1938 # Configuración de parámetros
1939 params = [
1940     ("Departamento:", sorted([d.title() for d in
self.calculadora.tasas_crecimiento.keys()]), "Ayacucho"),
1941     ("Mes:", ["enero", "febrero", "marzo", "abril", "mayo", "junio",
1942             "julio", "agosto", "septiembre", "octubre", "noviembre",
"diciembre"], "junio"),
1943     ("Tipo de Pavimento:", ["rigido", "flexible"], "rigido"),
1944     ("Período de Diseño:", ["10", "20"], "20")
1945 ]
1946
1947 self.combos = {}
1948 for label, values, default in params:
1949     tk.Label(frame_config, text=label,
1950             bg='#34495e', fg='#ecf0f1',
1951             font=('Arial', self.base_font, 'bold')).pack(anchor='w', pady=(10,
2))
1952
1953     combo = ttk.Combobox(frame_config, values=values, state="readonly",
1954                         font=('Arial', self.base_font))
1955     combo.set(default)
1956     combo.pack(fill='x', pady=(0, 5))
1957     self.combos[label.replace(":", "").replace(" ", "_").lower()] = combo
1958
1959 # Separador
1960 tk.Frame(frame_config, bg='#95a5a6', height=2).pack(fill='x', pady=10)
1961
1962 # Configuración de carretera
1963 tk.Label(frame_config, text="Configuración de Carretera:",
1964         bg='#34495e', fg='#e67e22',
1965         font=('Arial', self.base_font, 'bold')).pack(anchor='w', pady=(5, 5))
1966
1967 road_params = [
1968     ("Calzadas:", ["1", "2"], "1"),
1969     ("Sentidos:", ["1", "2"], "2"),
1970     ("Carriles por sentido:", ["1", "2", "3", "4"], "1")
1971 ]
1972
1973 for label, values, default in road_params:
1974     tk.Label(frame_config, text=label,

```

```

1975         bg='#34495e', fg='#ecf0f1',
1976         font=('Arial', self.base_font)).pack(anchor='w', pady=(5, 2))
1977
1978         combo = ttk.Combobox(frame_config, values=values, state="readonly",
1979                             font=('Arial', self.base_font))
1980         combo.set(default)
1981         combo.pack(fill='x', pady=(0, 3))
1982         self.combos[label.replace(":", "").replace(" ", "_").lower()] = combo
1983
1984     # Botón de cálculo
1985     self.btn_calcular = tk.Button(frame_config,
1986                                  text="🖨️ CALCULAR IMD + EJES EQUIVALENTES",
1987                                  command=self.calcular_imd_y_ejes_equivalentes,
1988                                  bg='#f39c12', fg='white',
1989                                  font=('Arial', self.base_font, 'bold'))
1990     self.btn_calcular.pack(fill='x', pady=(20, 0))
1991
1992     # COLUMNA 2: RESULTADOS (70%)
1993     frame_resultados = tk.LabelFrame(frame_principal,
1994                                     text="RESULTADOS DE CÁLCULO",
1995                                     font=('Arial', self.subtitle_font, 'bold'),
1996                                     bg='#34495e', fg='#27ae60',
1997                                     padx=10, pady=10)
1998     frame_resultados.pack(side='right', fill='both', expand=True)
1999
2000     # Canvas con scroll para resultados
2001     canvas_resultados = tk.Canvas(frame_resultados, bg='#34495e',
2002                                  highlightthickness=0)
2003     scrollbar_resultados = ttk.Scrollbar(frame_resultados, orient="vertical",
2004                                          command=canvas_resultados.yview)
2005     self.frame_resultados_scroll = tk.Frame(canvas_resultados, bg='#34495e')
2006
2007     # Scroll con mouse wheel
2008     def _on_mousewheel(event):
2009         canvas_resultados.yview_scroll(int(-1*(event.delta/120)), "units")
2010
2011     canvas_resultados.bind_all("<MouseWheel>", _on_mousewheel)
2012
2013     self.frame_resultados_scroll.bind(
2014         "<Configure>",
2015         lambda e:
2016             canvas_resultados.configure(scrollregion=canvas_resultados.bbox("all"))
2017     )
2018
2019     canvas_resultados.create_window((0, 0), window=self.frame_resultados_scroll,
2020                                     anchor="nw")
2021     canvas_resultados.configure(yscrollcommand=scrollbar_resultados.set)
2022
2023     canvas_resultados.pack(side="left", fill="both", expand=True)
2024     scrollbar_resultados.pack(side="right", fill="y")
2025
2026     self.mostrar_mensaje_inicial_resultados()
2027
2028     def crear_tab_exportacion(self, parent):

```



```

2025 """TAB PARA EXPORTACIÓN"""
2026
2027 frame_principal = tk.Frame(parent, bg='#2c3e50')
2028 frame_principal.pack(fill='both', expand=True, padx=20, pady=20)
2029
2030 # Título
2031 tk.Label(frame_principal,
2032         text="EXPORTACIÓN DE RESULTADOS",
2033         font=('Arial', self.title_font, 'bold'),
2034         bg='#2c3e50', fg='#ecf0f1').pack(pady=(0, 20))
2035
2036 # GRID DE BOTONES DE EXPORTACIÓN
2037 frame_botones = tk.Frame(frame_principal, bg='#2c3e50')
2038 frame_botones.pack(fill='both', expand=True)
2039
2040 # Configurar grid
2041 frame_botones.grid_columnconfigure(0, weight=1)
2042 frame_botones.grid_columnconfigure(1, weight=1)
2043 frame_botones.grid_rowconfigure(0, weight=1)
2044 frame_botones.grid_rowconfigure(1, weight=1)
2045 frame_botones.grid_rowconfigure(2, weight=1)
2046
2047 # Botones de exportación
2048 botones_export = [
2049     (" REPORTE IMD", self.exportar_reporte_imd, '#3498db'),
2050     (" CSV EJES EQUIVALENTES", self.exportar_csv_ejes, '#e67e22'),
2051     (" REPORTE COMPLETO", self.exportar_reporte_completo, '#9b59b6'),
2052     (" ABRIR VIDEOS", self.abrir_carpeta_videos_procesados, '#27ae60'),
2053     (" ABRIR CARPETA", self.abrir_carpeta_resultados, '#34495e'),
2054     (" EXPORTAR TODO", self.exportar_todo, '#e74c3c')
2055 ]
2056
2057 for i, (texto, comando, color) in enumerate(botones_export):
2058     row = i // 2
2059     col = i % 2
2060
2061     btn = tk.Button(frame_botones,
2062                    text=texto,
2063                    command=comando,
2064                    bg=color, fg='white',
2065                    font=('Arial', self.base_font + 2, 'bold'),
2066                    relief='raised',
2067                    bd=3)
2068     btn.grid(row=row, column=col, padx=10, pady=10, sticky='nsew')
2069
2070 # Información de estado
2071 frame_info = tk.LabelFrame(frame_principal,
2072                           text="INFORMACIÓN DE EXPORTACIÓN",
2073                           font=('Arial', self.subtitle_font, 'bold'),
2074                           bg='#34495e', fg='#ecf0f1',
2075                           padx=10, pady=10)
2076 frame_info.pack(fill='x', pady=(20, 0))
2077
2078 self.label_info_export = tk.Label(frame_info,

```



```

2079         text="Procese videos y calcule resultados para
habilitar exportación",
2080         bg='#34495e', fg='#95a5a6',
2081         font=('Arial', self.base_font),
2082         wraplength=600,
2083         justify='center')
2084     self.label_info_export.pack(pady=10)
2085
2086     # [CONTINÚA EN EL SIGUIENTE MENSAJE...]
2087
2088     def calcular_imd_y_ejes_equivalentes(self):
2089         """Calcular IMD y Ejes Equivalentes usando ESTACIÓN SELECCIONADA"""
2090         videos_procesados = sum(1 for conteo in self.conteos_por_dia.values() if conteo)
2091
2092         if videos_procesados == 0:
2093             messagebox.showwarning("Advertencia", "Debe procesar al menos un video")
2094             return
2095
2096         try:
2097             # Calcular IMD
2098             self.imd_calculado =
self.calculadora.calcular_imd_7_dias(self.conteos_por_dia)
2099
2100             if not self.imd_calculado or sum(self.imd_calculado.values()) == 0:
2101                 messagebox.showwarning("Advertencia", "No hay datos suficientes para
calcular IMD")
2102                 return
2103
2104             # 🌟 OBTENER ESTACIÓN SELECCIONADA 🌟
2105             estacion_seleccionada = self.combo_estacion.get()
2106
2107             # Configuración
2108             config = {
2109                 "estacion": estacion_seleccionada, # 🌟 NUEVA CONFIGURACIÓN
2110                 "peaje": estacion_seleccionada, # Alias para compatibilidad
2111                 "mes": self.combos["mes"].get(),
2112                 "departamento": self.combos["departamento"].get().lower(),
2113                 "tipo_pavimento": self.combos["tipo_de_pavimento"].get(),
2114                 "periodo_diseno": int(self.combos["período_de_diseño"].get()),
2115                 "num_calzadas": int(self.combos["calzadas"].get()),
2116                 "num_sentidos": int(self.combos["sentidos"].get()),
2117                 "carriles_por_sentido": int(self.combos["carriles_por_sentido"].get())
2118             }
2119
2120             # Factores globales
2121             fd, fc = self.calculadora.calcular_fd_fc(
2122                 config["num_calzadas"], config["num_sentidos"],
config["carriles_por_sentido"]
2123             )
2124
2125             # Calcular para cada tipo usando la estación seleccionada
2126             resultados_ee = {}
2127             total_nrep_ee = 0
2128

```

```

2129         for tipo, imd in self.imd_calculado.items():
2130             if imd == 0:
2131                 continue
2132
2133             # 🌟 USAR ESTACIÓN SELECCIONADA PARA FCE
2134             fce = self.calculadora.calcular_fce(tipo, estacion_seleccionada,
config["mes"])
2135             imda = self.calculadora.calcular_imda(imd, fce)
2136             fvp = self.calculadora.calcular_fvp(tipo, config["tipo_pavimento"])
2137             ee_dia_carril = self.calculadora.calcular_ee_dia_carril(imda, fd, fc,
fvp, 1.0)
2138             fca = self.calculadora.calcular_fca(tipo, config["departamento"],
config["periodo_diseño"])
2139             nrep_ee = self.calculadora.calcular_nrep_ee(ee_dia_carril, fca)
2140
2141             total_nrep_ee += nrep_ee
2142
2143             resultados_ee[tipo] = {
2144                 "imd": imd, "fce": fce, "imda": imda, "fd": fd, "fc": fc,
2145                 "fvp": fvp, "fp": 1.0, "ee_dia_carril": ee_dia_carril,
2146                 "fca": fca, "nrep_ee": nrep_ee,
2147                 "clasificacion": self.calculadora.clasificar_vehiculo(tipo)
2148             }
2149
2150             # Guardar resultados
2151             self.ejes_equivalentes_resultado = {
2152                 "resultados_por_tipo": resultados_ee,
2153                 "total_nrep_ee": total_nrep_ee,
2154                 "configuracion": config,
2155                 "factores_globales": {"fd": fd, "fc": fc},
2156                 "videos_procesados": videos_procesados,
2157                 "imd_calculado": self.imd_calculado
2158             }
2159
2160             # Mostrar resultados
2161             self.mostrar_resultados_calculados()
2162
2163             self.actualizar_status(f" Cálculo completado - Estación:
{estacion_seleccionada}")
2164
2165             messagebox.showinfo("Éxito",
2166                                 f" CÁLCULO COMPLETADO\n\n"
2167                                 f"Estación: {estacion_seleccionada}\n"
2168                                 f"Videos procesados: {videos_procesados}/7\n"
2169                                 f"Total NREP EE: {int(total_nrep_ee):,}")
2170
2171         except Exception as e:
2172             messagebox.showerror("Error", f"Error en cálculo:\n{str(e)}")
2173
2174         def mostrar_resultados_calculados(self):
2175             """Mostrar resultados calculados"""
2176             if not self.ejes_equivalentes_resultado:
2177                 return
2178

```



```

2179 # Limpiar
2180 for widget in self.frame_resultados_scroll.winfo_children():
2181     widget.destroy()
2182
2183 resultados = self.ejes_equivalentes_resultado["resultados_por_tipo"]
2184 total_ee = self.ejes_equivalentes_resultado["total_nrep_ee"]
2185 config = self.ejes_equivalentes_resultado["configuracion"]
2186
2187 # Mostrar estación utilizada
2188 frame_estacion = tk.LabelFrame(self.frame_resultados_scroll,
2189                                text="ESTACIÓN DE CONTEO",
2190                                bg='#34495e', fg='#e74c22',
2191                                font=('Arial', self.base_font + 1, 'bold'),
2192                                padx=10, pady=10)
2193 frame_estacion.pack(fill='x', pady=(0, 15))
2194
2195 tk.Label(frame_estacion,
2196          text=f" {config['estacion']}",
2197          bg='#34495e', fg='#f39c12',
2198          font=('Arial', self.base_font + 3, 'bold')).pack()
2199
2200 # IMD Calculado
2201 frame_imd = tk.LabelFrame(self.frame_resultados_scroll,
2202                           text="IMD CALCULADO (Suma 7 días / 7)",
2203                           bg='#34495e', fg='#3498db',
2204                           font=('Arial', self.base_font + 1, 'bold'),
2205                           padx=10, pady=10)
2206 frame_imd.pack(fill='x', pady=(0, 15))
2207
2208 for tipo, imd in sorted(self.imd_calculado.items(), key=lambda x: x[1],
2209 reverse=True):
2210     if imd > 0:
2211         tk.Label(frame_imd,
2212                  text=f"{tipo}: {imd:.2f} veh/día",
2213                  bg='#34495e', fg='#3498db',
2214                  font=('Arial', self.base_font)).pack(anchor='w', pady=2)
2215
2216 # Ejes Equivalentes por tipo
2217 frame_ee = tk.LabelFrame(self.frame_resultados_scroll,
2218                           text="EJES EQUIVALENTES POR TIPO",
2219                           bg='#34495e', fg='#f39c12',
2220                           font=('Arial', self.base_font + 1, 'bold'),
2221                           padx=10, pady=10)
2222 frame_ee.pack(fill='x', pady=(0, 15))
2223
2224 for tipo, datos in sorted(resultados.items(), key=lambda x: x[1]["nrep_ee"],
2225 reverse=True):
2226     if datos["nrep_ee"] > 0:
2227         frame_tipo = tk.LabelFrame(frame_ee,
2228                                     text=tipo,
2229                                     bg='#34495e', fg='#ecf0f1',
2230                                     font=('Arial', self.base_font, 'bold'),
2231                                     padx=8, pady=8)
2232         frame_tipo.pack(fill='x', pady=5)

```



```

2231
2232         # Crear grid de información
2233         info_grid = tk.Frame(frame_tipo, bg='#34495e')
2234         info_grid.pack(fill='x')
2235
2236         # Fila 1
2237         tk.Label(info_grid, text=f"IMD: {datos['imd']:.2f}",
2238                 bg='#34495e', fg='#3498db', font=('Arial', self.base_font -
2239 1)).grid(row=0, column=0, sticky='w', padx=5)
2240         tk.Label(info_grid, text=f"IMDA: {datos['imda']}",
2241                 bg='#34495e', fg='#3498db', font=('Arial', self.base_font -
2242 1)).grid(row=0, column=1, sticky='w', padx=5)
2243
2244         # Fila 2
2245         tk.Label(info_grid, text=f"FVP: {datos['fvp']:.4f}",
2246                 bg='#34495e', fg='#e67e22', font=('Arial', self.base_font -
2247 1)).grid(row=1, column=0, sticky='w', padx=5)
2248         tk.Label(info_grid, text=f"FCE: {datos['fce']:.4f}",
2249                 bg='#34495e', fg='#e67e22', font=('Arial', self.base_font -
2250 1)).grid(row=1, column=1, sticky='w', padx=5)
2251
2252         # Fila 3
2253         tk.Label(info_grid, text=f"EE/día: {datos['ee_día_carril']:.3f}",
2254                 bg='#34495e', fg='#9b59b6', font=('Arial', self.base_font -
2255 1)).grid(row=2, column=0, sticky='w', padx=5)
2256         tk.Label(info_grid, text=f"FCA: {datos['fca']:.2f}",
2257                 bg='#34495e', fg='#9b59b6', font=('Arial', self.base_font -
2258 1)).grid(row=2, column=1, sticky='w', padx=5)
2259
2260         # NREP EE destacado
2261         tk.Label(frame_tipo,
2262                 text=f"NREP EE: {int(datos['nrep_ee']):,}",
2263                 bg='#34495e', fg='#f39c12',
2264                 font=('Arial', self.base_font + 1, 'bold')).pack(pady=(10, 0))
2265
2266         # Total final
2267         frame_total = tk.Frame(self.frame_resultados_scroll, bg='#2c3e50',
2268 relief='raised', bd=3)
2269         frame_total.pack(fill='x', pady=15)
2270
2271         tk.Label(frame_total,
2272                 text="TOTAL NREP EE (DISEÑO):",
2273                 bg='#2c3e50', fg='#f39c12',
2274                 font=('Arial', self.base_font + 2, 'bold')).pack(pady=5)
2275
2276         tk.Label(frame_total,
2277                 text=f"{int(total_ee):,}",
2278                 bg='#2c3e50', fg='#f39c12',
2279                 font=('Arial', self.base_font + 4, 'bold')).pack(pady=5)
2280
2281         # Configuración utilizada
2282         frame_config_res = tk.LabelFrame(self.frame_resultados_scroll,
2283                 text="CONFIGURACIÓN UTILIZADA",
2284                 bg='#34495e', fg='#95a5a6',

```

```

2278         font=('Arial', self.base_font, 'bold'),
2279         padx=10, pady=10)
2280     frame_config_res.pack(fill='x', pady=(15, 0))
2281
2282     info_config = [
2283         f"Estación: {config['estacion']}", # NUEVO
2284         f"Departamento: {config['departamento'].title()}",
2285         f"Mes: {config['mes'].title()}",
2286         f"Pavimento: {config['tipo_pavimento'].title()}",
2287         f"Período: {config['periodo_diseno']} años",
2288         f"Videos procesados: {self.ejes_equivalentes_re-
sultado['videos_procesados']}/7",
2289         f"FD: {self.ejes_equivalentes_resultado['factores_globales']['fd']:.2f} | FC:
{self.ejes_equivalentes_resultado['factores_globales']['fc']:.2f}"
2290     ]
2291
2292     for item in info_config:
2293         tk.Label(frame_config_res, text=item, bg='#34495e', fg='#95a5a6',
2294                 font=('Arial', self.base_font - 1)).pack(anchor='w', pady=1)
2295
2296     # [RESTO DE MÉTODOS PERMANECEN IGUALES - Los copio del código original...]
2297
2298     def cargar_modelo(self):
2299         """Cargar modelo YOLO OPTIMIZADO PARA CPU"""
2300         def cargar():
2301             try:
2302                 modelos = ["best_vehicular_model.pt", "best_yuvica.pt", "best.pt"]
2303                 modelo_encontrado = None
2304
2305                 for modelo in modelos:
2306                     if os.path.exists(modelo):
2307                         modelo_encontrado = modelo
2308                         break
2309
2310                 if modelo_encontrado:
2311                     self.contador = ContadorVehicular(modelo_encontrado)
2312                     self.modelo_cargado = True
2313
2314                     clases = list(self.contador.modelo.names.values())
2315                     texto_clases = ", ".join(clases[:5]) + ("..." if len(clases) > 5 else
""")
2316
2317                     self.root.after(0, lambda: self.label_modelo.configure(
2318                         text=f" Modelo CPU: {texto_clases}", fg='#27ae60'))
2319                     self.root.after(0, self.verificar_estado_botones)
2320                 else:
2321                     self.root.after(0, lambda: self.label_modelo.configure(
2322                         text=" Modelo no encontrado", fg='#e74c3c'))
2323
2324             except Exception as e:
2325                 self.root.after(0, lambda: self.label_modelo.configure(
2326                     text=f" Error: {str(e)[:30]}...", fg='#e74c3c'))
2327
2328         threading.Thread(target=cargar, daemon=True).start()

```



```

2329
2330 def seleccionar_carpeta_salida(self):
2331     """Seleccionar carpeta para videos exportados"""
2332     carpeta = filedialog.askdirectory(title="Seleccionar carpeta para videos
procesados")
2333
2334     if carpeta:
2335         self.carpeta_videos_procesados = carpeta
2336         nombre = os.path.basename(carpeta) or carpeta
2337         self.label_carpeta.configure(text=f" {nombre}", fg='#27ae60')
2338         self.actualizar_status(f" Carpeta configurada: {carpeta}")
2339     else:
2340         self.carpeta_videos_procesados = os.getcwd()
2341         self.label_carpeta.configure(text=" Carpeta actual", fg='#f39c12')
2342
2343 def seleccionar_video_dia(self, dia):
2344     """Seleccionar video para un día"""
2345     video = filedialog.askopenfilename(
2346         title=f"Seleccionar video para {dia.upper()}",
2347         filetypes=[
2348             ("Videos", "*.mp4 *.avi *.mov *.mkv *.wmv"),
2349             ("Todos", "*.*")
2350         ]
2351     )
2352
2353     if video:
2354         self.videos_por_dia[dia] = video
2355         nombre = os.path.basename(video)
2356         max_chars = 25
2357         texto = f"📺 {nombre}" if len(nombre) <= max_chars else f"📺
{nombre[:max_chars-3]}..."
2358         self.labels_dias[dia].configure(text=texto, fg='#27ae60')
2359         self.verificar_estado_botones()
2360         self.actualizar_status(f" Video asignado a {dia}")
2361
2362 def verificar_estado_botones(self):
2363     """Verificar estado de botones"""
2364     videos_asignados = sum(1 for v in self.videos_por_dia.values() if v is not None)
2365
2366     if self.modelo_cargado and videos_asignados > 0:
2367         self.btn_procesar.configure(state='normal')
2368     else:
2369         self.btn_procesar.configure(state='disabled')
2370
2371 def procesar_todos_los_dias(self):
2372     """Procesar todos los videos"""
2373     if not self.modelo_cargado:
2374         messagebox.showwarning("Advertencia", "El modelo no está cargado")
2375         return
2376
2377     videos_para_procesar = [(dia, video) for dia, video in
self.videos_por_dia.items() if video]
2378
2379     if not videos_para_procesar:

```



```

2380     messagebox.showwarning("Advertencia", "No hay videos para procesar")
2381     return
2382
2383     if self.exportar_videos_procesados.get() and not self.carpeta_videos_procesados:
2384         if messagebox.askyesno("Carpeta", "¿Usar carpeta actual para videos?"):
2385             self.carpeta_videos_procesados = os.getcwd()
2386         else:
2387             return
2388
2389     self.procesando = True
2390     self.btn_procesar.configure(state='disabled')
2391     self.btn_detener.configure(state='normal')
2392
2393     threading.Thread(target=self.procesar_videos_secuencial,
2394                     args=(videos_para_procesar,), daemon=True).start()
2395
2396     def procesar_videos_secuencial(self, videos_para_procesar):
2397         """Procesar videos secuencialmente CON CONTROL DE VELOCIDAD"""
2398         total_videos = len(videos_para_procesar)
2399
2400         for i, (dia, video_path) in enumerate(videos_para_procesar):
2401             if not self.procesando:
2402                 break
2403
2404             self.root.after(0, lambda d=dia, n=i+1, t=total_videos:
2405                             self.actualizar_status(f"🖥️ Procesando {d} ({n}/{t})"))
2406             self.root.after(0, lambda d=dia:
2407                             self.label_dia_actual.configure(text=f"Procesando:
2408 {d.upper()}"))
2409
2410             self.contador.resetear_contador()
2411             resultado = self.procesar_video_individual(dia, video_path)
2412
2413             if resultado:
2414                 self.conteos_por_dia[dia] = resultado
2415                 self.estados_procesamiento[dia] = "completado"
2416                 self.root.after(0, self.actualizar_conteos_display)
2417                 self.root.after(0, lambda d=dia: self.actualizar_status_dia(d,
2418 "completado"))
2419             else:
2420                 self.estados_procesamiento[dia] = "error"
2421                 self.root.after(0, lambda d=dia: self.actualizar_status_dia(d, "error"))
2422
2423             if self.procesando:
2424                 self.root.after(0, lambda: self.actualizar_status(" Procesamiento
2425 completado"))
2426                 self.root.after(0, lambda: self.label_dia_actual.configure(text="
2427 Completado"))
2428
2429             self.root.after(0, self.finalizar_procesamiento)
2430
2431     def procesar_video_individual(self, dia, video_path):
2432         """Procesar un video individual CON CONTROL DE VELOCIDAD"""
2433         try:

```

```

2430 self.cap = cv2.VideoCapture(video_path)
2431 if not self.cap.isOpened():
2432     return None
2433
2434 # Configuración del video
2435 self.total_frames = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
2436 fps_original = int(self.cap.get(cv2.CAP_PROP_FPS)) or 30
2437 width = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
2438 height = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
2439 self.frame_actual = 0
2440
2441 print(f"Video: {dia} | FPS original: {fps_original} | Resolución:
{width}x{height}")
2442 print(f"FPS reproducción: 30 (fijo)")
2443
2444 # Configurar exportación si está habilitada
2445 writer_video = None
2446 if self.exportar_videos_procesados.get() and self.carpeta_videos_procesados:
2447     timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
2448     nombre_salida = f"EjesEquivalentes_{dia}_{timestamp}.mp4"
2449     ruta_salida = os.path.join(self.carpeta_videos_procesados, nombre_salida)
2450
2451 # CONFIGURACIÓN MEJORADA PARA EXPORTACIÓN
2452 fourcc = cv2.VideoWriter_fourcc(*'mp4v')
2453 writer_video = cv2.VideoWriter(ruta_salida, fourcc, 30, (width, height))
2454
2455 if writer_video.isOpened():
2456     self.root.after(0, lambda: self.label_export_status.configure(
2457         text=f"Exportando: {dia}", fg='#e67e22'))
2458     print(f"Video de salida configurado: {ruta_salida}")
2459 else:
2460     writer_video = None
2461     print("Error: No se pudo abrir writer de video")
2462
2463 # INICIALIZAR CONTROL DE TIEMPO
2464 self.last_frame_time = time.time()
2465
2466 while True:
2467     if not self.procesando:
2468         break
2469
2470     ret, frame = self.cap.read()
2471     if not ret:
2472         break
2473
2474     # PROCESAR FRAME CON YOLO
2475     detecciones = self.contador.procesar_frame(frame)
2476
2477     # CREAR FRAME CON DETECCIONES PARA EXPORTACIÓN
2478     if writer_video:
2479         frame_export = self.dibujar_detecciones_exportacion(frame.copy(),
detecciones, dia)
2480         writer_video.write(frame_export)
2481

```



```

2482         # ACTUALIZAR UI CON CONTROL DE VELOCIDAD
2483         current_time = time.time()
2484         elapsed_time = current_time - self.last_frame_time
2485
2486         # Actualizar UI solo si ha pasado el tiempo necesario según FPS
configurado
2487         if elapsed_time >= self.frame_time:
2488             frame_ui = self.dibujar_detecciones_ui(frame.copy(), detecciones,
dia)
2489             self.root.after(0, lambda f=frame_ui: self.actualizar_video_ui(f))
2490
2491             progreso = (self.frame_actual / self.total_frames) * 100
2492             self.root.after(0, lambda p=progreso:
self.progress_bar.configure(value=p))
2493             self.root.after(0, lambda: self.progress_video_var.set(
2494                 f"Frame {self.frame_actual}/{self.total_frames} ({progreso:.1f}%)
- 30 FPS"))
2495
2496             self.last_frame_time = current_time
2497
2498         # CONTROL DE VELOCIDAD: Esperar el tiempo necesario
2499         frame_process_time = time.time() - current_time
2500         sleep_time = max(0, self.frame_time - frame_process_time)
2501         if sleep_time > 0:
2502             time.sleep(sleep_time)
2503
2504         self.frame_actual += 1
2505
2506         self.cap.release()
2507
2508         if writer_video:
2509             writer_video.release()
2510             self.root.after(0, lambda: self.label_export_status.configure(
2511                 text=f"Video {dia} exportado", fg='#27ae60'))
2512             print(f"Video exportado exitosamente para {dia}")
2513
2514         return self.contador.obtener_conteo_final()
2515
2516     except Exception as e:
2517         error_msg = f"Error en {dia}: {str(e)}"
2518         self.root.after(0, lambda: self.actualizar_status(error_msg))
2519         print(f"Error procesando video {dia}: {e}")
2520         if hasattr(self, 'cap') and self.cap:
2521             self.cap.release()
2522         if 'writer_video' in locals() and writer_video:
2523             writer_video.release()
2524         return None
2525
2526     # [MÉTODOS DE VISUALIZACIÓN Y EXPORTACIÓN PERMANECEN IGUALES...]
2527
2528     def dibujar_detecciones_exportacion(self, frame, detecciones, dia):
2529         """Dibujar detecciones para video exportado"""
2530         height, width = frame.shape[:2]
2531

```



```

2532 # Fondo para información superior
2533 overlay = frame.copy()
2534 cv2.rectangle(overlay, (0, 0), (width, 120), (0, 0, 0), -1)
2535 cv2.addWeighted(overlay, 0.8, frame, 0.2, 0, frame)
2536
2537 # Información del sistema
2538 progreso = (self.frame_actual / self.total_frames) * 100 if self.total_frames > 0
else 0
2539 total_detectados = sum(self.contador.conteo_unico_por_clase.values())
2540
2541 textos = [
2542     f"SISTEMA EJES EQUIVALENTES v6.0 - {dia.upper()}",
2543     f"Frame: {self.frame_actual}/{self.total_frames} ({progreso:.1f}%)",
2544     f"Total detectados: {total_detectados} | FPS: 30",
2545     f"Fecha: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
2546 ]
2547
2548 for i, texto in enumerate(textos):
2549     y_pos = 25 + (i * 22)
2550     cv2.putText(frame, texto, (15, y_pos),
2551                 cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
2552
2553 # DIBUJAR DETECCIONES CON BOUNDING BOXES GRANDES
2554 for deteccion in detecciones:
2555     if 'id' in deteccion:
2556         vehiculo = self.contador.vehiculos_trackeados[deteccion['id']]
2557         bbox = deteccion['bbox']
2558         color = self.contador.colores[deteccion['clase_id']]
2559
2560         # BOUNDING BOX MÁS GRUESO Y VISIBLE
2561         cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), color, 6)
2562
2563         # ETIQUETA MÁS GRANDE Y VISIBLE
2564         etiqueta = f"ID:{deteccion['id']} {vehiculo.tipo_actual}"
2565         confianza = f"{int(vehiculo.confianza_maxima * 100)}%"
2566
2567         # FONDO MÁS GRANDE PARA ETIQUETA
2568         (tw, th), _ = cv2.getTextSize(etiqueta, cv2.FONT_HERSHEY_SIMPLEX, 1.0, 3)
2569         (cw, ch), _ = cv2.getTextSize(confianza, cv2.FONT_HERSHEY_SIMPLEX, 0.8,
2)
2570
2571         # Calcular dimensiones del fondo
2572         fondo_width = max(tw, cw) + 20
2573         fondo_height = th + ch + 25
2574
2575         # FONDO DE ETIQUETA MÁS VISIBLE
2576         cv2.rectangle(frame,
2577                       (bbox[0] - 5, bbox[1] - fondo_height - 5),
2578                       (bbox[0] + fondo_width, bbox[1] + 5),
2579                       color, -1)
2580
2581         # BORDE DEL FONDO
2582         cv2.rectangle(frame,
2583                       (bbox[0] - 5, bbox[1] - fondo_height - 5),

```

```

2584         (bbox[0] + fondo_width, bbox[1] + 5),
2585         (255, 255, 255), 2)
2586
2587     # TEXTO DE ETIQUETA MÁS GRANDE
2588     cv2.putText(frame, etiqueta,
2589                 (bbox[0] + 5, bbox[1] - ch - 10),
2590                 cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), 3)
2591
2592     # TEXTO DE CONFIANZA
2593     cv2.putText(frame, confianza,
2594                 (bbox[0] + 5, bbox[1] - 5),
2595                 cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
2596
2597     # REGISTRO DE CONTEO ACTUAL
2598     conteo = self.contador.conteo_unico_por_clase
2599     if conteo:
2600         conteos_activos = {tipo: cantidad for tipo, cantidad in conteo.items() if
cantidad > 0}
2601
2602     if conteos_activos:
2603         max_items = len(conteos_activos)
2604         panel_height = max(80, (max_items * 35) + 40)
2605         panel_width = 280
2606
2607         panel_x = width - panel_width - 20
2608         panel_y = 140
2609
2610         # FONDO DEL PANEL DE CONTEO
2611         cv2.rectangle(frame,
2612                       (panel_x, panel_y),
2613                       (panel_x + panel_width, panel_y + panel_height),
2614                       (0, 0, 0), -1)
2615
2616         # BORDE DEL PANEL
2617         cv2.rectangle(frame,
2618                       (panel_x, panel_y),
2619                       (panel_x + panel_width, panel_y + panel_height),
2620                       (255, 255, 255), 3)
2621
2622         # TÍTULO DEL PANEL
2623         cv2.putText(frame, "CONTEO ACTUAL:",
2624                     (panel_x + 10, panel_y + 30),
2625                     cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
2626
2627         # LÍNEA SEPARADORA
2628         cv2.line(frame,
2629                  (panel_x + 10, panel_y + 40),
2630                  (panel_x + panel_width - 10, panel_y + 40),
2631                  (255, 255, 255), 2)
2632
2633         # MOSTRAR CONTEOS
2634         y_offset = 65
2635         for tipo, cantidad in sorted(conteos_activos.items(), key=lambda x: x[1],
reverse=True):

```

```

2636         if tipo in ['A', 'SW', 'PU', 'PA']:
2637             color_tipo = (0, 255, 255) # Amarillo
2638         elif tipo in ['B2E', 'B3E', 'B4E']:
2639             color_tipo = (0, 165, 255) # Naranja
2640         elif tipo in ['C2E', 'C3E']:
2641             color_tipo = (0, 0, 255) # Rojo
2642         else:
2643             color_tipo = (255, 0, 255) # Magenta
2644
2645         texto_conteo = f"{tipo}: {cantidad}"
2646         cv2.putText(frame, texto_conteo,
2647                     (panel_x + 15, panel_y + y_offset),
2648                     cv2.FONT_HERSHEY_SIMPLEX, 0.9, color_tipo, 2)
2649
2650         y_offset += 35
2651
2652     # TOTAL GENERAL
2653     total_general = sum(conteos_activos.values())
2654     cv2.line(frame,
2655              (panel_x + 10, panel_y + y_offset - 15),
2656              (panel_x + panel_width - 10, panel_y + y_offset - 15),
2657              (255, 255, 255), 2)
2658
2659     cv2.putText(frame, f"TOTAL: {total_general}",
2660                 (panel_x + 15, panel_y + y_offset + 10),
2661                 cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 255, 0), 3)
2662
2663     return frame
2664
2665 def dibujar_detecciones_ui(self, frame, detecciones, dia):
2666     """Dibujar detecciones para UI"""
2667     height, width = frame.shape[:2]
2668
2669     # Información básica
2670     overlay = frame.copy()
2671     cv2.rectangle(overlay, (0, 0), (width, 60), (0, 0, 0), -1)
2672     cv2.addWeighted(overlay, 0.7, frame, 0.3, 0, frame)
2673
2674     progreso = (self.frame_actual / self.total_frames) * 100 if self.total_frames > 0
2675 else 0
2676     total = sum(self.contador.conteo_unico_por_clase.values())
2677
2678     texto = f"CPU v6.0: {dia.upper()} | Frame: {self.frame_actual} | {progreso:.1f}%
2679 | Total: {total}"
2680     cv2.putText(frame, texto, (10, 35), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255,
2681 255), 2)
2682
2683     # DETECCIONES
2684     for deteccion in detecciones:
2685         if 'id' in deteccion:
2686             vehiculo = self.contador.vehiculos_trackeados[deteccion['id']]
2687             bbox = deteccion['bbox']
2688             color = self.contador.colores[deteccion['clase_id']]

```



```

2687         cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), color, 4)
2688
2689         etiqueta = f"ID:{deteccion['id']} {vehiculo.tipo_actual}"
2690
2691         (tw, th), _ = cv2.getTextSize(etiqueta, cv2.FONT_HERSHEY_SIMPLEX, 0.8, 2)
2692         cv2.rectangle(frame,
2693             (bbox[0] - 2, bbox[1] - th - 8),
2694             (bbox[0] + tw + 4, bbox[1] + 2),
2695             color, -1)
2696
2697         cv2.putText(frame, etiqueta,
2698             (bbox[0], bbox[1] - 4),
2699             cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
2700
2701     return frame
2702
2703 def actualizar_video_ui(self, frame):
2704     """Actualizar video en interfaz"""
2705     try:
2706         height, width = frame.shape[:2]
2707
2708         label_width = self.video_label.winfo_width()
2709         label_height = self.video_label.winfo_height()
2710
2711         if label_width <= 1:
2712             max_width = int(self.window_width * 0.45)
2713             max_height = max(300, int(self.window_height * 0.4))
2714         else:
2715             max_width = label_width - 10
2716             max_height = label_height - 10
2717
2718         scale = min(max_width/width, max_height/height)
2719         new_width = int(width * scale)
2720         new_height = int(height * scale)
2721
2722         frame_resized = cv2.resize(frame, (new_width, new_height))
2723         frame_rgb = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2RGB)
2724         image_pil = Image.fromarray(frame_rgb)
2725         image_tk = ImageTk.PhotoImage(image_pil)
2726
2727         self.video_label.configure(image=image_tk)
2728         self.video_label.image = image_tk
2729
2730     except Exception as e:
2731         print(f"Error actualizando video: {e}")
2732
2733 def actualizar_status_dia(self, dia, estado):
2734     """Actualizar estado de un día"""
2735     if estado == "completado":
2736         total = sum(self.conteos_por_dia[dia].values()) if self.conteos_por_dia[dia]
2737     else 0
2738
2739     self.labels_dias[dia].configure(text=f" {total} vehículos", fg='#27ae60')
2740     elif estado == "error":
2741         self.labels_dias[dia].configure(text=" Error", fg='#e74c3c')

```

```

2740
2741 def actualizar_conteos_display(self):
2742     """Actualizar display de conteos"""
2743     for widget in self.frame_conteos_scroll.winfo_children():
2744         widget.destroy()
2745
2746     tk.Label(self.frame_conteos_scroll,
2747             text="CONTEOS POR DÍA:",
2748             bg='#34495e', fg='#ecf0f1',
2749             font=('Arial', self.base_font + 1, 'bold')).pack(anchor='w', pady=(0,
10))
2750
2751     for dia in self.videos_por_dia.keys():
2752         frame_dia = tk.LabelFrame(self.frame_conteos_scroll,
2753                                 text=dia.upper(),
2754                                 font=('Arial', self.base_font, 'bold'),
2755                                 bg='#34495e', fg='#ecf0f1',
2756                                 padx=5, pady=5)
2757         frame_dia.pack(fill='x', pady=3)
2758
2759         conteo = self.conteos_por_dia[dia]
2760
2761         if conteo and sum(conteo.values()) > 0:
2762             for tipo, cantidad in sorted(conteo.items(), key=lambda x: x[1],
reverse=True):
2763                 if cantidad > 0:
2764                     frame_tipo = tk.Frame(frame_dia, bg='#34495e')
2765                     frame_tipo.pack(fill='x', pady=1)
2766
2767                     tk.Label(frame_tipo, text=f"{tipo}:", bg='#34495e', fg='#bdc3c7',
2768                             font=('Arial', self.base_font - 1), width=8,
anchor='w').pack(side='left')
2769
2770                     tk.Label(frame_tipo, text=str(cantidad), bg='#34495e',
fg='#f39c12',
2771                             font=('Arial', self.base_font - 1, 'bold'), width=4,
anchor='e').pack(side='right')
2772
2773                     total_dia = sum(conteo.values())
2774                     tk.Label(frame_dia, text=f"Total: {total_dia}", bg='#34495e',
fg='#27ae60',
2775                             font=('Arial', self.base_font, 'bold')).pack()
2776                 else:
2777                     tk.Label(frame_dia, text="Sin conteos", bg='#34495e', fg='#95a5a6',
2778                             font=('Arial', self.base_font - 1)).pack()
2779
2780     def mostrar_mensaje_inicial_resultados(self):
2781         """Mensaje inicial en resultados"""
2782         for widget in self.frame_resultados_scroll.winfo_children():
2783             widget.destroy()
2784
2785         mensaje = tk.Label(self.frame_resultados_scroll,
2786                           text="1. Procese los videos de la semana\n2. Configure los
parámetros\n3. Calcule IMD + Ejes Equivalentes",

```

```

2787         bg='#34495e', fg='#95a5a6',
2788         font=('Arial', self.base_font + 2),
2789         justify='center')
2790     mensaje.pack(expand=True, pady=50)
2791
2792     def detener_procesamiento(self):
2793         """Detener procesamiento"""
2794         self.procesando = False
2795         self.finalizar_procesamiento()
2796         self.actualizar_status(" Procesamiento detenido")
2797
2798     def finalizar_procesamiento(self):
2799         """Finalizar procesamiento"""
2800         self.procesando = False
2801
2802         if hasattr(self, 'cap') and self.cap:
2803             self.cap.release()
2804
2805         self.btn_procesar.configure(state='normal' if self.modelo_cargado else
'disabled')
2806         self.btn_detener.configure(state='disabled')
2807
2808     def resetear_todo(self):
2809         """Resetear sistema"""
2810         if messagebox.askyesno("Confirmar", "¿Resetear todos los datos?"):
2811             for dia in self.videos_por_dia.keys():
2812                 self.videos_por_dia[dia] = None
2813                 self.conteos_por_dia[dia] = {}
2814                 self.estados_procesamiento[dia] = "pendiente"
2815                 self.labels_dias[dia].configure(text=" Sin video", fg='#95a5a6')
2816
2817             self.imd_calculado = {}
2818             self.ejes_equivalentes_resultado = None
2819
2820             self.actualizar_conteos_display()
2821             self.mostrar_mensaje_inicial_resultados()
2822             self.verificar_estado_botones()
2823
2824             self.actualizar_status(" Sistema reseteado")
2825
2826     # [MÉTODOS DE EXPORTACIÓN...]
2827
2828     def exportar_reporte_imd(self):
2829         """Exportar reporte IMD"""
2830         if not self.imd_calculado:
2831             messagebox.showwarning("Advertencia", "Debe calcular IMD primero")
2832             return
2833
2834         archivo = filedialog.asksaveasfilename(
2835             title="Guardar Reporte IMD",
2836             defaultextension=".txt",
2837             filetypes=[("Texto", "*.txt")],
2838             initialname=f"reporte_imd_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt"
2839         )

```



```

2840
2841     if archivo:
2842         try:
2843             with open(archivo, 'w', encoding='utf-8') as f:
2844                 f.write("="*80 + "\n")
2845                 f.write("REPORTE IMD - SISTEMA v6.0\n")
2846                 f.write("="*80 + "\n\n")
2847
2848                 f.write(f"Fecha: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
2849
2850                 # Si hay estación seleccionada
2851                 if self.ejes_equivalentes_resultado:
2852                     estacion = self.ejes_equivalentes_resultado['configuracion']
2853                     f.write(f"Estación: {estacion}\n\n")
2854
2855                 # IMD calculado
2856                 f.write(f"IMD CALCULADO:\n")
2857                 f.write("-"*40 + "\n")
2858
2859                 total_imd = 0
2860                 for tipo, imd in sorted(self.imd_calculado.items(), key=lambda x:
x[1], reverse=True):
2861                     f.write(f"{tipo}: {imd:.2f} veh/día\n")
2862                     total_imd += imd
2863
2864                 f.write(f"\nTOTAL IMD: {total_imd:.2f} veh/día\n")
2865
2866                 messagebox.showinfo("Éxito", f"Reporte guardado:\n{archivo}")
2867                 self.actualizar_status(" Reporte IMD exportado")
2868
2869             except Exception as e:
2870                 messagebox.showerror("Error", f"Error al guardar:\n{str(e)}")
2871
2872     def exportar_csv_ejes(self):
2873         """Exportar CSV ejes equivalentes"""
2874         if not self.ejes_equivalentes_resultado:
2875             messagebox.showwarning("Advertencia", "Debe calcular ejes equivalentes
primero")
2876             return
2877
2878         archivo = filedialog.asksaveasfilename(
2879             title="Guardar CSV Ejes Equivalentes",
2880             defaultextension=".csv",
2881             filetypes=[("CSV", "*.csv")],
2882             initialname=f"ejes_equivalentes_{datetime.now().strftime('%Y-%m-%d_%H%M%S')}.csv"
2883         )
2884
2885         if archivo:
2886             try:
2887                 with open(archivo, 'w', newline='', encoding='utf-8') as f:
2888                     writer = csv.writer(f)
2889

```

```

2890         # Encabezados
2891         writer.writerow([
2892             'Tipo_Vehiculo', 'IMD', 'FCE', 'IMDA', 'FD', 'FC',
2893             'FVP', 'FP', 'EE_Dia_Carril', 'FCA', 'NREP_EE', 'Clasificacion',
2894             'Estacion'
2895         ])
2896
2897         # Datos
2898         resultados = self.ejes_equivalentes_resultado["resultados_por_tipo"]
2899         estacion = self.ejes_equivalentes_resultado["configuracion"]
2900
2901         for tipo, datos in sorted(resultados.items()):
2902             writer.writerow([
2903                 tipo, f"{datos['imd']:.2f}", f"{datos['fce']:.4f}",
2904                 f"{datos['fd']:.2f}", f"{datos['fc']:.2f}", f"
2905                 {datos['fvp']:.6f}",
2906                 f"{datos['fp']:.1f}", f"{datos['ee_dia_carril']:.4f}",
2907                 f"{datos['fca']:.2f}", f"{int(datos['nrep_ee'])}",
2908                 datos['clasificacion'],
2909                 estacion
2910             ])
2911
2912             writer.writerow([
2913                 'RESUMEN'
2914             ])
2915             writer.writerow([
2916                 'Estación', estacion
2917             ])
2918             writer.writerow([
2919                 'Total NREP EE', int(self.ejes_equivalentes_re-
2920                 sultado['total_nrep_ee'])
2921             ])
2922
2923             messagebox.showinfo("Éxito", f"CSV guardado:\n{archivo}")
2924             self.actualizar_status(" CSV exportado")
2925
2926         except Exception as e:
2927             messagebox.showerror("Error", f"Error al guardar CSV:\n{str(e)}")
2928
2929     def exportar_reporte_completo(self):
2930         """Exportar reporte completo JSON"""
2931         if not self.ejes_equivalentes_resultado:
2932             messagebox.showwarning("Advertencia", "Debe calcular resultados primero")
2933             return
2934
2935         archivo = filedialog.asksaveasfilename(
2936             title="Guardar Reporte Completo",
2937             defaultextension=".json",
2938             filetypes=[("JSON", "*.json")],
2939             initialname=f"reporte_completo_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json"
2940         )
2941
2942         if archivo:
2943             try:
2944                 reporte = {
2945                     "timestamp": datetime.now().isoformat(),

```



```

2937         "version": "Sistema Ejes Equivalentes v6.0",
2938         "conteos_por_dia": self.conteos_por_dia,
2939         "videos_asignados": {dia: os.path.basename(video) if video else None
2940                               for dia, video in self.videos_por_dia.items()},
2941         "imd_calculado": self.imd_calculado,
2942         "ejes_equivalentes": self.ejes_equivalentes_resultado
2943     }
2944
2945     with open(archivo, 'w', encoding='utf-8') as f:
2946         json.dump(reporte, f, indent=2, ensure_ascii=False)
2947
2948         messagebox.showinfo("Éxito", f"Reporte completo guardado:\n{archivo}")
2949         self.actualizar_status(" Reporte completo exportado")
2950
2951     except Exception as e:
2952         messagebox.showerror("Error", f"Error al guardar reporte:\n{str(e)}")
2953
2954 def exportar_todo(self):
2955     """Exportar todos los reportes"""
2956     if not self.ejes_equivalentes_resultado:
2957         messagebox.showwarning("Advertencia", "Debe calcular resultados primero")
2958         return
2959
2960     carpeta = filedialog.askdirectory(title="Seleccionar carpeta para exportar todo")
2961     if not carpeta:
2962         return
2963
2964     try:
2965         timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
2966         archivos_creados = []
2967
2968         # Exportar IMD
2969         # Exportar CSV
2970         # Exportar JSON
2971         # [... código de exportación completo ...]
2972
2973         messagebox.showinfo("Éxito",
2974                             f"Todos los reportes exportados:\n\n"
2975                             f" Carpeta: {carpeta}")
2976         self.actualizar_status(" Exportación completa realizada")
2977
2978     except Exception as e:
2979         messagebox.showerror("Error", f"Error en exportación:\n{str(e)}")
2980
2981 def abrir_carpeta_videos_procesados(self):
2982     """Abrir carpeta de videos procesados"""
2983     if not self.carpeta_videos_procesados:
2984         messagebox.showinfo("Info", "No hay carpeta de videos configurada")
2985         return
2986
2987     try:
2988         if os.name == 'nt':
2989             os.startfile(self.carpeta_videos_procesados)
2990         else:

```



```

2991         os.system(f'xdg-open "{self.carpeta_videos_procesados}"')
2992     except Exception as e:
2993         messagebox.showerror("Error", f"No se pudo abrir carpeta:\n{str(e)}")
2994
2995     def abrir_carpeta_resultados(self):
2996         """Abrir carpeta actual"""
2997         try:
2998             carpeta = os.getcwd()
2999             if os.name == 'nt':
3000                 os.startfile(carpeta)
3001             else:
3002                 os.system(f'xdg-open "{carpeta}"')
3003         except Exception as e:
3004             messagebox.showerror("Error", f"No se pudo abrir carpeta:\n{str(e)}")
3005
3006     def actualizar_status(self, mensaje):
3007         """Actualizar barra de estado"""
3008         if "" in mensaje:
3009             color = '#27ae60'
3010         elif "" in mensaje:
3011             color = '#e74c3c'
3012         elif "" in mensaje or "🔍" in mensaje:
3013             color = '#f39c12'
3014         elif "" in mensaje:
3015             color = '#3498db'
3016         else:
3017             color = '#34495e'
3018
3019         self.status_bar.configure(text=mensaje, bg=color)
3020
3021     def ejecutar(self):
3022         """Ejecutar aplicación"""
3023         def on_closing():
3024             if self.procesando:
3025                 if messagebox.askokcancel("Salir", "¿Detener procesamiento y salir?"):
3026                     self.procesando = False
3027                     if hasattr(self, 'cap') and self.cap:
3028                         self.cap.release()
3029                     self.root.destroy()
3030             else:
3031                 self.root.destroy()
3032
3033         self.root.protocol("WM_DELETE_WINDOW", on_closing)
3034         self.root.mainloop()
3035
3036     def main():
3037         """Función principal"""
3038         try:
3039             print(" Sistema de Ejes Equivalentes v6.0")
3040             print("="*80)
3041             print(" 78 ESTACIONES DE CONTEO")
3042             print(" 24 DEPARTAMENTOS DEL PERÚ")
3043             print(" METODOLOGÍA AASHTO COMPLETA")
3044             print(" CPU CONTROLADO + EXPORTACIÓN AUTOMÁTICA")

```

```
3045     print("="*80)
3046
3047     app = InterfazSistemaEjesEquivalentes()
3048     app.ejecutar()
3049
3050 except Exception as e:
3051     print(f" Error: {e}")
3052     import traceback
3053     traceback.print_exc()
3054
3055     root_error = tk.Tk()
3056     root_error.withdraw()
3057     messagebox.showerror("Error Fatal",
3058                          f"Error al inicializar:\n\n{str(e)}")
3059     root_error.destroy()
3060
3061 if __name__ == "__main__":
3062     main()
```

~\Desktop\vision_artificial\YUVICA\SOFTWARE_EJES EQUIVALANTES.py

```

1 import os
2 import sys
3
4 # =====
5 # CONFIGURACIÓN CRÍTICA: DEBE ESTAR ANTES DE CUALQUIER IMPORT DE TORCH/YOLO
6 # =====
7
8 # Deshabilitar CUDA completamente a nivel de sistema
9 os.environ['CUDA_VISIBLE_DEVICES'] = '-1' # -1 significa "sin GPU"
10 os.environ['CUDA_LAUNCH_BLOCKING'] = '1'
11 os.environ['TORCH_USE_CUDA_DSA'] = '0'
12 os.environ['FORCE_CUDA'] = '0'
13
14 # Variables adicionales para asegurar CPU
15 os.environ['OMP_NUM_THREADS'] = str(os.cpu_count() or 4)
16 os.environ['YOLO_VERBOSE'] = 'False'
17
18 # Forzar torch a usar CPU antes de que se importe
19 import warnings
20 warnings.filterwarnings('ignore', category=UserWarning)
21
22 try:
23     import torch
24     # Sobrescribir la función que detecta CUDA
25     torch.cuda.is_available = lambda: False
26     torch.cuda.device_count = lambda: 0
27     torch.cuda.current_device = lambda: None
28     torch.backends.cudnn.enabled = False
29     print(" PyTorch configurado para CPU SOLAMENTE")
30 except ImportError:
31     print(" PyTorch no disponible, continuando sin él")
32 except Exception as e:
33     print(f" Error configurando PyTorch: {e}")
34
35 # =====
36 # AHORA SÍ IMPORTAR EL RESTO
37 # =====
38
39 import tkinter as tk
40 from tkinter import ttk, filedialog, messagebox, scrolledtext
41 import cv2
42 from ultralytics import YOLO
43 import numpy as np
44 import json
45 from datetime import datetime
46 import csv
47 from collections import defaultdict, deque
48 import math
49 import threading
50 from PIL import Image, ImageTk
51 import time

```



```

52
53 print("="*80)
54 print(" SISTEMA EJES EQUIVALENTES v6.0 - TODAS LAS ESTACIONES Y DEPARTAMENTOS")
55 print("="*80)
56 print(f" CPUs disponibles: {os.cpu_count()}")
57 print(f" CUDA deshabilitado: CUDA_VISIBLE_DEVICES =
{os.environ.get('CUDA_VISIBLE_DEVICES')}")
58 print("="*80)
59
60 class CalculadorEjesEquivalentes:
61     """Calculadora completa de Ejes Equivalentes según metodología AASHTO"""
62     def __init__(self):
63         # =====
64         # TABLAS DE FACTORES DE CORRECCIÓN ESTACIONAL (FCE) - TODAS LAS ESTACIONES
65         # =====
66
67         # FACTORES PARA VEHÍCULOS LIGEROS - 78 ESTACIONES
68         self.fce_ligeros = {
69             "AGUAS CALIENTES": {
70                 "enero": 0.9362, "febrero": 0.8626, "marzo": 1.0850, "abril": 1.0925,
71                 "mayo": 1.1626, "junio": 1.1758, "julio": 0.9508, "agosto": 0.8846,
72                 "septiembre": 0.9455, "octubre": 1.0418, "noviembre": 1.0414,
73                 "diciembre": 0.9899
74             },
75             "AGUAS CLARAS": {
76                 "enero": 1.0020, "febrero": 1.0567, "marzo": 1.1036, "abril": 1.0592,
77                 "mayo": 1.0095, "junio": 1.0030, "julio": 0.8943, "agosto": 0.9075,
78                 "septiembre": 1.0128, "octubre": 1.0209, "noviembre": 1.0749,
79                 "diciembre": 0.9283
80             },
81             "AMBO": {
82                 "enero": 0.9780, "febrero": 1.0546, "marzo": 1.1877, "abril": 1.2270,
83                 "mayo": 1.0179, "junio": 1.0278, "julio": 0.9661, "agosto": 0.9541,
84                 "septiembre": 1.0045, "octubre": 0.9693, "noviembre": 0.9846,
85                 "diciembre": 0.8615
86             },
87             "ATICO": {
88                 "enero": 0.8554, "febrero": 0.7148, "marzo": 1.0360, "abril": 1.0139,
89                 "mayo": 1.1484, "junio": 1.1781, "julio": 0.9965, "agosto": 0.9877,
90                 "septiembre": 1.1394, "octubre": 1.0734, "noviembre": 1.1453,
91                 "diciembre": 0.8831
92             },
93             "AYAVIRI": {
94                 "enero": 0.9722, "febrero": 0.9244, "marzo": 1.0735, "abril": 1.0773,
95                 "mayo": 1.1098, "junio": 1.1035, "julio": 0.9469, "agosto": 0.9272,
96                 "septiembre": 0.9376, "octubre": 1.0491, "noviembre": 1.0215,
97                 "diciembre": 0.9603
98             },
99             "CAMANA": {
100                 "enero": 0.5958, "febrero": 0.4991, "marzo": 1.0434, "abril": 1.2433,
101                 "mayo": 1.3825, "junio": 1.4071, "julio": 1.2562, "agosto": 1.2200,
102                 "septiembre": 1.3018, "octubre": 1.2559, "noviembre": 1.2199,
103                 "diciembre": 0.9724
104             },

```

```

99         "CANCAS": {
100             "enero": 0.8619, "febrero": 0.8638, "marzo": 1.0755, "abril": 1.1156,
101             "mayo": 1.1769, "junio": 1.2257, "julio": 0.9733, "agosto": 0.9040,
102             "septiembre": 1.0565, "octubre": 1.0323, "noviembre": 1.0284,
103             "diciembre": 0.8897
104         },
105         "CARACOTO": {
106             "enero": 1.0517, "febrero": 0.9815, "marzo": 1.0978, "abril": 1.0548,
107             "mayo": 1.0540, "junio": 1.0469, "julio": 0.9914, "agosto": 0.8786,
108             "septiembre": 0.9962, "octubre": 0.9920, "noviembre": 0.9977,
109             "diciembre": 0.8907
110         },
111         "CASARACRA": {
112             "enero": 1.1057, "febrero": 1.1747, "marzo": 1.2363, "abril": 1.0139,
113             "mayo": 0.9312, "junio": 1.0287, "julio": 0.8415, "agosto": 0.8825,
114             "septiembre": 1.0603, "octubre": 0.9930, "noviembre": 1.1250,
115             "diciembre": 0.9775
116         },
117         "CATAC": {
118             "enero": 1.0956, "febrero": 1.0575, "marzo": 1.1345, "abril": 1.0628,
119             "mayo": 1.0762, "junio": 1.0767, "julio": 0.8220, "agosto": 0.8684,
120             "septiembre": 0.9595, "octubre": 0.9494, "noviembre": 1.1134,
121             "diciembre": 0.9559
122         },
123         "CCASACANCHA": {
124             "enero": 1.0047, "febrero": 1.0408, "marzo": 1.0824, "abril": 1.0548,
125             "mayo": 1.0802, "junio": 1.0837, "julio": 0.9767, "agosto": 0.9426,
126             "septiembre": 0.9081, "octubre": 0.9961, "noviembre": 0.9756,
127             "diciembre": 0.9254
128         },
129         "CHACAPAMPA": {
130             "enero": 1.0446, "febrero": 0.9661, "marzo": 0.9178, "abril": 1.0174,
131             "mayo": 1.0405, "junio": 1.1257, "julio": 0.9550, "agosto": 0.9888,
132             "septiembre": 1.0378, "octubre": 1.0111, "noviembre": 1.1305,
133             "diciembre": 1.0215
134         },
135         "CHALHUAPUQUIO": {
136             "enero": 1.1671, "febrero": 1.2012, "marzo": 1.1909, "abril": 1.0566,
137             "mayo": 1.0146, "junio": 1.0442, "julio": 0.7925, "agosto": 0.8346,
138             "septiembre": 1.0198, "octubre": 0.9725, "noviembre": 1.0259,
139             "diciembre": 0.9319
140         },
141         "CHICAMA": {
142             "enero": 0.9671, "febrero": 0.9414, "marzo": 1.1130, "abril": 1.0394,
143             "mayo": 1.0773, "junio": 1.0585, "julio": 0.9554, "agosto": 1.0080,
144             "septiembre": 1.0731, "octubre": 1.0420, "noviembre": 1.0520,
145             "diciembre": 0.8249
146         },
147         "CHILCA": {
148             "enero": 0.6163, "febrero": 0.5814, "marzo": 0.7890, "abril": 1.0577,
149             "mayo": 1.5273, "junio": 1.5796, "julio": 1.2842, "agosto": 1.4015,
150             "septiembre": 1.4833, "octubre": 1.2554, "noviembre": 1.1876,
151             "diciembre": 0.7400
152         },
153         "CHULLQUI": {

```



```

145         "enero": 1.0428, "febrero": 1.0728, "marzo": 1.0509, "abril": 1.0163,
146         "mayo": 1.0500, "junio": 0.9407, "julio": 0.9832, "agosto": 0.9316 ,
147         "septiembre": 0.9915, "octubre": 0.9207, "noviembre": 1.2832,
148     "diciembre": 0.8829
149     },
150     "CHULUCANAS": {
151         "enero": 1.0135, "febrero": 1.0695, "marzo": 1.1798, "abril": 1.1501,
152         "mayo": 1.0679, "junio": 1.0398, "julio": 0.9751, "agosto": 0.9127,
153         "septiembre": 0.9779, "octubre": 0.9474, "noviembre": 0.9119,
154     "diciembre": 0.8682
155     },
156     "CIUDAD DE DIOS": {
157         "enero": 1.4832, "febrero": 0.9071, "marzo": 1.1711, "abril": 1.0698,
158         "mayo": 1.0027, "junio": 1.0245, "julio": 0.9031, "agosto": 0.9123,
159         "septiembre": 1.0735, "octubre": 1.0006, "noviembre": 1.2021,
160     "diciembre": 0.8987
161     },
162     "CORCONA": {
163         "enero": 1.1024, "febrero": 1.1641, "marzo": 1.2033, "abril": 1.0335,
164         "mayo": 0.9767, "junio": 1.0411, "julio": 0.7767, "agosto": 0.8725,
165         "septiembre": 1.0142, "octubre": 1.0035, "noviembre": 1.1653,
166     "diciembre": 1.0001
167     },
168     "CRUCE BAYOVAR": {
169         "enero": 0.8734, "febrero": 0.8855, "marzo": 1.1136, "abril": 1.1087,
170         "mayo": 1.1470, "junio": 1.1828, "julio": 0.8641, "agosto": 0.9783,
171         "septiembre": 1.1469, "octubre": 1.0024, "noviembre": 1.0777,
172     "diciembre": 0.7715
173     },
174     "CUCULI": {
175         "enero": 0.9948, "febrero": 1.0346, "marzo": 1.1594, "abril": 1.1180,
176         "mayo": 1.1059, "junio": 0.9573, "julio": 0.9652, "agosto": 0.9185,
177         "septiembre": 0.9475, "octubre": 0.9719, "noviembre": 0.9647,
178     "diciembre": 0.9955
179     },
180     "DESVIO OLMOS": {
181         "enero": 0.9497, "febrero": 1.0057, "marzo": 1.1672, "abril": 1.1939,
182         "mayo": 1.1494, "junio": 1.0960, "julio": 0.9411, "agosto": 0.8681,
183         "septiembre": 0.9965, "octubre": 0.9618, "noviembre": 1.0119,
184     "diciembre": 0.8222
185     },
186     "DESVIO TALARA": {
187         "enero": 0.8705, "febrero": 0.8863, "marzo": 1.1244, "abril": 1.0840,
188         "mayo": 1.1438, "junio": 1.1754, "julio": 0.9963, "agosto": 0.9874,
189         "septiembre": 1.1057, "octubre": 1.0288, "noviembre": 1.0309,
190     "diciembre": 0.8166
191     },
192     "EL FISCAL": {
193         "enero": 0.9109, "febrero": 0.8549, "marzo": 1.0606, "abril": 1.0601,
194         "mayo": 1.0691, "junio": 1.1260, "julio": 0.9958, "agosto": 0.9853,
195         "septiembre": 1.0750, "octubre": 1.0662, "noviembre": 1.0647,
196     "diciembre": 0.9479
197     },
198     "EL PARAISO": {
199         "enero": 0.9135, "febrero": 0.8984, "marzo": 1.0432, "abril": 0.9974,

```



```

191         "mayo": 1.1183, "junio": 1.1530, "julio": 0.9141, "agosto": 0.9811,
192         "septiembre": 1.0964, "octubre": 1.0384, "noviembre": 1.0935,
193         "diciembre": 0.8747
194     },
195     "FORTALEZA": {
196         "enero": 0.8869, "febrero": 0.8317, "marzo": 1.0074, "abril": 1.0227,
197         "mayo": 1.1568, "junio": 1.1965, "julio": 0.8936, "agosto": 1.0119,
198         "septiembre": 1.1837, "octubre": 1.1022, "noviembre": 1.1702,
199         "diciembre": 0.7936
200     },
201     "HUACRAPUQUIO": {
202         "enero": 1.1193, "febrero": 1.1570, "marzo": 1.0649, "abril": 1.0486,
203         "mayo": 1.1504, "junio": 0.9951, "julio": 0.8705, "agosto": 0.9487,
204         "septiembre": 0.9945, "octubre": 0.9710, "noviembre": 1.1529,
205         "diciembre": 0.8270
206     },
207     "HUARMEY": {
208         "enero": 0.8816, "febrero": 0.9056, "marzo": 1.1120, "abril": 1.1249,
209         "mayo": 1.2557, "junio": 1.2048, "julio": 0.8790, "agosto": 0.9703,
210         "septiembre": 1.1483, "octubre": 1.0831, "noviembre": 1.1634,
211         "diciembre": 0.7903
212     },
213     "HUATAYA": {
214         "enero": 0.0000, "febrero": 0.0000, "marzo": 2.3609, "abril": 0.9600,
215         "mayo": 0.8747, "junio": 0.8705, "julio": 0.7536, "agosto": 0.7578,
216         "septiembre": 0.0000, "octubre": 0.7931, "noviembre": 0.8788,
217         "diciembre": 0.8353
218     },
219     "ICA": {
220         "enero": 0.9310, "febrero": 0.8181, "marzo": 1.0513, "abril": 1.0461,
221         "mayo": 1.1642, "junio": 1.1857, "julio": 0.9741, "agosto": 1.0246,
222         "septiembre": 1.0994, "octubre": 1.0312, "noviembre": 1.1911,
223         "diciembre": 1.0337
224     },
225     "ILAVE": {
226         "enero": 1.0111, "febrero": 0.9584, "marzo": 1.0360, "abril": 1.0136,
227         "mayo": 1.1067, "junio": 1.1613, "julio": 0.9700, "agosto": 0.7834,
228         "septiembre": 1.0335, "octubre": 1.0501, "noviembre": 1.1248,
229         "diciembre": 0.9861
230     },
231     "ILO": {
232         "enero": 0.8275, "febrero": 0.8217, "marzo": 1.0112, "abril": 1.0814,
233         "mayo": 1.0777, "junio": 1.1220, "julio": 1.0989, "agosto": 1.0557,
234         "septiembre": 0.9854, "octubre": 1.0498, "noviembre": 1.0303,
235         "diciembre": 0.9630
236     },
237     "JAHUAY-CHINCHA": {
238         "enero": 0.8291, "febrero": 0.8936, "marzo": 1.0170, "abril": 0.9967,
239         "mayo": 1.1373, "junio": 1.1918, "julio": 0.9486, "agosto": 1.0137,
240         "septiembre": 1.1321, "octubre": 1.0340, "noviembre": 1.0634,
241         "diciembre": 0.9289
242     },
243     "LOMA LARGA BAJA": {
244         "enero": 1.0231, "febrero": 1.2771, "marzo": 1.3671, "abril": 1.2848,
245         "mayo": 1.1408, "junio": 1.0369, "julio": 0.8267, "agosto": 0.9064,

```



```

237         "septiembre": 0.9247, "octubre": 0.8925, "noviembre": 0.8826,
    "diciembre": 0.8752
238     },
239     "LUNAHUANA": {
240         "enero": 1.0119, "febrero": 1.0303, "marzo": 1.0475, "abril": 0.9587,
241         "mayo": 1.0163, "junio": 1.1317, "julio": 0.8233, "agosto": 0.9366,
242         "septiembre": 1.0988, "octubre": 0.9949, "noviembre": 1.0780,
    "diciembre": 1.0638
243     },
244     "MACUSANI": {
245         "enero": 1.0465, "febrero": 0.9990, "marzo": 1.0338, "abril": 1.0944,
246         "mayo": 1.0979, "junio": 1.1081, "julio": 0.9926, "agosto": 0.9493,
247         "septiembre": 1.0253, "octubre": 0.9652, "noviembre": 1.0158,
    "diciembre": 0.8879
248     },
249     "MARCONA": {
250         "enero": 0.9525, "febrero": 0.9039, "marzo": 0.9952, "abril": 1.0106,
251         "mayo": 1.0911, "junio": 1.0595, "julio": 1.0346, "agosto": 1.0232,
252         "septiembre": 1.0372, "octubre": 1.0290, "noviembre": 0.9989,
    "diciembre": 0.9111
253     },
254     "MATARANI": {
255         "enero": 0.4810, "febrero": 0.3998, "marzo": 0.9717, "abril": 1.5343,
256         "mayo": 1.7023, "junio": 1.6718, "julio": 1.6092, "agosto": 1.5584,
257         "septiembre": 1.5827, "octubre": 1.4257, "noviembre": 1.2888,
    "diciembre": 0.9119
258     },
259     "MENOCUCHO": {
260         "enero": 1.0850, "febrero": 0.9812, "marzo": 1.0500, "abril": 1.0811,
261         "mayo": 1.0266, "junio": 1.0713, "julio": 0.9585, "agosto": 0.9192,
262         "septiembre": 1.0498, "octubre": 1.0467, "noviembre": 1.0452,
    "diciembre": 0.8579
263     },
264     "MOCCE": {
265         "enero": 1.0115, "febrero": 0.9769, "marzo": 1.0613, "abril": 1.0650,
266         "mayo": 1.0408, "junio": 0.9962, "julio": 0.9898, "agosto": 0.9054,
267         "septiembre": 1.0213, "octubre": 1.0118, "noviembre": 1.0012,
    "diciembre": 0.9247
268     },
269     "MONTALVO": {
270         "enero": 0.9256, "febrero": 0.8896, "marzo": 1.0507, "abril": 1.0351,
271         "mayo": 1.0336, "junio": 1.1014, "julio": 1.0261, "agosto": 0.9905,
272         "septiembre": 1.0447, "octubre": 1.0608, "noviembre": 1.0346,
    "diciembre": 0.9658
273     },
274     "MORROPE": {
275         "enero": 0.9291, "febrero": 0.9180, "marzo": 1.2042, "abril": 1.1140,
276         "mayo": 1.1254, "junio": 1.1566, "julio": 0.8933, "agosto": 0.9593,
277         "septiembre": 1.0807, "octubre": 0.9811, "noviembre": 1.0512,
    "diciembre": 0.8016
278     },
279     "MOYOBAMBA": {
280         "enero": 1.0172, "febrero": 0.9993, "marzo": 1.0281, "abril": 1.0219,
281         "mayo": 0.9910, "junio": 0.9724, "julio": 0.9456, "agosto": 0.9603,

```

```

282         "septiembre": 1.0486, "octubre": 1.0817, "noviembre": 1.1373,
    "diciembre": 1.0800
283     },
284     "NASCA": {
285         "enero": 0.9690, "febrero": 0.9023, "marzo": 1.0494, "abril": 1.0595,
286         "mayo": 1.0762, "junio": 1.0902, "julio": 0.9243, "agosto": 0.9306,
287         "septiembre": 1.0245, "octubre": 1.0160, "noviembre": 1.0671,
    "diciembre": 0.9918
288     },
289     "PACANGUILLA": {
290         "enero": 0.9539, "febrero": 0.9606, "marzo": 1.1189, "abril": 1.1170,
291         "mayo": 1.1420, "junio": 1.1808, "julio": 0.9622, "agosto": 0.9834,
292         "septiembre": 1.0937, "octubre": 0.9832, "noviembre": 0.9979,
    "diciembre": 0.7654
293     },
294     "PACRA": {
295         "enero": 1.0091, "febrero": 0.9570, "marzo": 1.0209, "abril": 0.9881,
296         "mayo": 1.2122, "junio": 1.0758, "julio": 0.8772, "agosto": 0.9346,
297         "septiembre": 1.0435, "octubre": 1.0345, "noviembre": 1.0857,
    "diciembre": 0.9795
298     },
299     "PAITA": {
300         "enero": 0.8362, "febrero": 0.8550, "marzo": 0.9600, "abril": 1.1002,
301         "mayo": 1.1329, "junio": 1.1228, "julio": 1.0946, "agosto": 1.0862,
302         "septiembre": 1.0113, "octubre": 1.0572, "noviembre": 1.0343,
    "diciembre": 0.8543
303     },
304     "PAMPA CUELLAR": {
305         "enero": 1.0268, "febrero": 0.8142, "marzo": 1.0724, "abril": 1.0806,
306         "mayo": 1.1616, "junio": 1.1803, "julio": 0.9567, "agosto": 0.7957,
307         "septiembre": 1.0641, "octubre": 1.0380, "noviembre": 1.0557,
    "diciembre": 0.9772
308     },
309     "PAMPA GALERA": {
310         "enero": 0.9640, "febrero": 1.0087, "marzo": 1.1311, "abril": 1.1214,
311         "mayo": 1.0518, "junio": 1.0972, "julio": 0.8190, "agosto": 0.7818,
312         "septiembre": 1.0505, "octubre": 1.0760, "noviembre": 1.1323,
    "diciembre": 0.9586
313     },
314     "PAMPAMARCA": {
315         "enero": 0.9412, "febrero": 0.9552, "marzo": 1.0962, "abril": 1.0681,
316         "mayo": 1.1301, "junio": 1.1134, "julio": 0.8653, "agosto": 0.8837,
317         "septiembre": 1.0188, "octubre": 1.0975, "noviembre": 1.1159,
    "diciembre": 0.9830
318     },
319     "PATAHUASI": {
320         "enero": 1.0505, "febrero": 0.9287, "marzo": 1.1460, "abril": 1.0872,
321         "mayo": 1.1086, "junio": 1.1280, "julio": 0.9169, "agosto": 0.8010,
322         "septiembre": 1.0421, "octubre": 0.9869, "noviembre": 1.0075,
    "diciembre": 0.9579
323     },
324     "PEDRO RUIZ": {
325         "enero": 0.9661, "febrero": 1.0279, "marzo": 1.1022, "abril": 1.1233,
326         "mayo": 1.1127, "junio": 1.0515, "julio": 0.9396, "agosto": 0.9041,

```



```
327         "septiembre": 0.9700, "octubre": 0.9832, "noviembre": 1.0130,  
    "diciembre": 0.8890  
328     },  
329     "PICHIRHUA": {  
330         "enero": 1.0465, "febrero": 1.0891, "marzo": 1.1333, "abril": 1.0561,  
331         "mayo": 1.0383, "junio": 1.0173, "julio": 0.9081, "agosto": 0.8840,  
332         "septiembre": 0.9812, "octubre": 0.9968, "noviembre": 1.0126,  
    "diciembre": 0.9151  
333     },  
334     "PIURA SULLANA": {  
335         "enero": 1.0607, "febrero": 1.0540, "marzo": 1.1688, "abril": 1.0898,  
336         "mayo": 1.0558, "junio": 1.0528, "julio": 0.9697, "agosto": 0.9539,  
337         "septiembre": 1.0019, "octubre": 0.9640, "noviembre": 0.9557,  
    "diciembre": 0.8684  
338     },  
339     "PLANCHON": {  
340         "enero": 1.0327, "febrero": 1.0756, "marzo": 1.0641, "abril": 1.0613,  
341         "mayo": 1.0628, "junio": 1.0224, "julio": 0.9361, "agosto": 0.9131,  
342         "septiembre": 0.9554, "octubre": 0.9680, "noviembre": 1.0417,  
    "diciembre": 0.9385  
343     },  
344     "POMAHUACA": {  
345         "enero": 0.9587, "febrero": 0.9933, "marzo": 1.1653, "abril": 1.2213,  
346         "mayo": 1.1511, "junio": 1.0886, "julio": 0.9287, "agosto": 0.8481,  
347         "septiembre": 1.0002, "octubre": 0.9934, "noviembre": 1.0133,  
    "diciembre": 0.8296  
348     },  
349     "PONGO": {  
350         "enero": 1.0265, "febrero": 1.0591, "marzo": 1.0560, "abril": 1.0787,  
351         "mayo": 1.0546, "junio": 1.0025, "julio": 1.0750, "agosto": 0.9103,  
352         "septiembre": 0.9692, "octubre": 0.9654, "noviembre": 0.9778,  
    "diciembre": 0.9308  
353     },  
354     "POZO REDONDO": {  
355         "enero": 0.8942, "febrero": 0.8240, "marzo": 1.0133, "abril": 1.0783,  
356         "mayo": 1.1178, "junio": 1.0895, "julio": 1.0494, "agosto": 1.0438,  
357         "septiembre": 1.1167, "octubre": 1.0579, "noviembre": 1.0216,  
    "diciembre": 0.8480  
358     },  
359     "PUNTA PERDIDA": {  
360         "enero": 0.9544, "febrero": 0.7922, "marzo": 1.0980, "abril": 1.1995,  
361         "mayo": 1.4001, "junio": 1.4069, "julio": 0.8238, "agosto": 0.6031,  
362         "septiembre": 1.1767, "octubre": 1.0628, "noviembre": 1.2693,  
    "diciembre": 1.1014  
363     },  
364     "QUIULLA": {  
365         "enero": 1.0974, "febrero": 1.1408, "marzo": 1.2315, "abril": 1.0452,  
366         "mayo": 1.0212, "junio": 1.0679, "julio": 0.8140, "agosto": 0.8685,  
367         "septiembre": 1.0002, "octubre": 1.0063, "noviembre": 1.1046,  
    "diciembre": 0.9381  
368     },  
369     "RUMICHACA": {  
370         "enero": 1.0577, "febrero": 0.9112, "marzo": 0.9934, "abril": 0.9745,  
371         "mayo": 1.2193, "junio": 1.2055, "julio": 0.8748, "agosto": 0.8955,
```

```

372         "septiembre": 1.0297, "octubre": 1.0766, "noviembre": 1.1671,
    "diciembre": 0.9701
373     },
374     "SAN ANTON": {
375         "enero": 1.0691, "febrero": 1.0523, "marzo": 1.0689, "abril": 1.1318,
376         "mayo": 1.0035, "junio": 1.0850, "julio": 1.0541, "agosto": 1.0138,
377         "septiembre": 0.9978, "octubre": 0.9206, "noviembre": 0.9981,
    "diciembre": 0.8467
378     },
379     "SAN GABAN": {
380         "enero": 1.0594, "febrero": 1.0125, "marzo": 1.0893, "abril": 1.1043,
381         "mayo": 1.1331, "junio": 1.0921, "julio": 0.9727, "agosto": 0.8972,
382         "septiembre": 0.9272, "octubre": 0.9258, "noviembre": 0.9512,
    "diciembre": 0.9451
383     },
384     "SAN LORENZO": {
385         "enero": 1.0839, "febrero": 1.1914, "marzo": 1.1227, "abril": 1.2647,
386         "mayo": 1.0266, "junio": 1.0376, "julio": 0.8703, "agosto": 0.9258,
387         "septiembre": 0.9436, "octubre": 0.9610, "noviembre": 1.0442,
    "diciembre": 0.9270
388     },
389     "SANTA LUCIA": {
390         "enero": 1.0087, "febrero": 0.8363, "marzo": 1.1211, "abril": 1.1106,
391         "mayo": 1.1184, "junio": 1.1768, "julio": 0.9560, "agosto": 0.7674,
392         "septiembre": 1.0724, "octubre": 1.0214, "noviembre": 1.0523,
    "diciembre": 0.9795
393     },
394     "SAYLLA": {
395         "enero": 1.0222, "febrero": 0.9785, "marzo": 1.1062, "abril": 1.0858,
396         "mayo": 1.0613, "junio": 1.0638, "julio": 0.9806, "agosto": 0.9142,
397         "septiembre": 0.9238, "octubre": 0.9909, "noviembre": 0.9947,
    "diciembre": 0.9411
398     },
399     "SERPENTIN DE PASAMAYO": {
400         "enero": 1.0535, "febrero": 1.0503, "marzo": 1.0886, "abril": 1.0814,
401         "mayo": 1.0647, "junio": 1.0464, "julio": 0.9694, "agosto": 0.8540,
402         "septiembre": 1.0238, "octubre": 1.0450, "noviembre": 1.0263,
    "diciembre": 1.0654
403     },
404     "SICUYANI": {
405         "enero": 1.0297, "febrero": 0.8278, "marzo": 1.0261, "abril": 1.0976,
406         "mayo": 1.1547, "junio": 1.1691, "julio": 0.9155, "agosto": 0.7740,
407         "septiembre": 0.9930, "octubre": 1.0466, "noviembre": 1.1655,
    "diciembre": 1.0126
408     },
409     "SOCOS": {
410         "enero": 1.1903, "febrero": 0.9863, "marzo": 0.9820, "abril": 0.9188,
411         "mayo": 1.0940, "junio": 1.0772, "julio": 0.9340, "agosto": 0.9544,
412         "septiembre": 1.0099, "octubre": 1.0057, "noviembre": 1.0421,
    "diciembre": 0.9406
413     },
414     "TAMBOGRANDE": {
415         "enero": 0.9297, "febrero": 0.9679, "marzo": 1.0638, "abril": 1.1030,
416         "mayo": 1.2229, "junio": 1.0587, "julio": 1.0396, "agosto": 1.0332,

```



```
417         "septiembre": 1.0144, "octubre": 0.8993, "noviembre": 0.9585,  
    "diciembre": 0.8799  
418     },  
419     "TOMASIRI": {  
420         "enero": 0.9972, "febrero": 0.9232, "marzo": 1.0516, "abril": 1.0804,  
421         "mayo": 1.1000, "junio": 1.0942, "julio": 1.0293, "agosto": 0.9900,  
422         "septiembre": 0.8973, "octubre": 1.0404, "noviembre": 1.0386,  
    "diciembre": 0.9031  
423     },  
424     "TUNAN": {  
425         "enero": 1.0761, "febrero": 1.0516, "marzo": 1.0703, "abril": 1.0230,  
426         "mayo": 1.1174, "junio": 1.0413, "julio": 0.8626, "agosto": 0.8554,  
427         "septiembre": 0.9819, "octubre": 0.9733, "noviembre": 1.1132,  
    "diciembre": 0.9816  
428     },  
429     "UNION-PROGRESO": {  
430         "enero": 1.0256, "febrero": 1.0263, "marzo": 1.0728, "abril": 1.0366,  
431         "mayo": 1.0256, "junio": 1.0113, "julio": 0.9544, "agosto": 0.9355,  
432         "septiembre": 0.9696, "octubre": 1.0061, "noviembre": 1.1122,  
    "diciembre": 0.9406  
433     },  
434     "UTCUBAMBA": {  
435         "enero": 1.1969, "febrero": 1.0343, "marzo": 1.0957, "abril": 1.1113,  
436         "mayo": 1.0756, "junio": 1.0478, "julio": 0.9556, "agosto": 0.9124,  
437         "septiembre": 0.9571, "octubre": 0.9881, "noviembre": 0.9318,  
    "diciembre": 0.8952  
438     },  
439     "VARIANTE DE PASAMAYO": {  
440         "enero": 0.9444, "febrero": 0.9203, "marzo": 1.0337, "abril": 1.0118,  
441         "mayo": 1.0802, "junio": 1.1121, "julio": 0.9392, "agosto": 0.9927,  
442         "septiembre": 1.0740, "octubre": 1.0049, "noviembre": 1.0472,  
    "diciembre": 0.9053  
443     },  
444     "VARIANTE DE UCHUMAYO": {  
445         "enero": 0.7411, "febrero": 0.6766, "marzo": 1.0218, "abril": 1.1452,  
446         "mayo": 1.1929, "junio": 1.1990, "julio": 1.1217, "agosto": 1.0762,  
447         "septiembre": 1.1286, "octubre": 1.1417, "noviembre": 1.1231,  
    "diciembre": 0.9590  
448     },  
449     "VESIQUE": {  
450         "enero": 0.8602, "febrero": 0.9031, "marzo": 1.0597, "abril": 1.0853,  
451         "mayo": 1.1302, "junio": 1.1518, "julio": 1.0065, "agosto": 1.0684,  
452         "septiembre": 1.1196, "octubre": 1.0108, "noviembre": 1.0305,  
    "diciembre": 0.8067  
453     },  
454     "VIRU": {  
455         "enero": 0.9903, "febrero": 0.9640, "marzo": 1.0774, "abril": 1.0604,  
456         "mayo": 1.1153, "junio": 1.1221, "julio": 0.9573, "agosto": 1.0148,  
457         "septiembre": 1.2094, "octubre": 0.9743, "noviembre": 0.9961,  
    "diciembre": 0.7989  
458     },  
459     "YAUCA": {  
460         "enero": 0.8794, "febrero": 0.7966, "marzo": 1.0518, "abril": 1.0285,  
461         "mayo": 1.1247, "junio": 1.1292, "julio": 0.9653, "agosto": 0.9839,
```



```

462         "septiembre": 1.0684, "octubre": 1.0963, "noviembre": 1.1303,
    "diciembre": 0.9048
463     }
464 }
465
466 # FACTORES PARA VEHÍCULOS PESADOS - 78 ESTACIONES
467 self.fce_pesados = {
468     "AGUAS CALIENTES": {
469         "enero": 1.0220, "febrero": 0.9808, "marzo": 1.0324, "abril": 1.0653,
470         "mayo": 1.0697, "junio": 1.1028, "julio": 0.9678, "agosto": 0.9555,
471         "septiembre": 0.9682, "octubre": 0.9530, "noviembre": 0.9599,
    "diciembre": 1.0098
472     },
473     "AGUAS CLARAS": {
474         "enero": 1.0412, "febrero": 1.0049, "marzo": 1.0248, "abril": 1.0107,
475         "mayo": 0.9905, "junio": 0.9823, "julio": 0.9917, "agosto": 0.9523,
476         "septiembre": 0.9768, "octubre": 1.0070, "noviembre": 1.0080,
    "diciembre": 1.0344
477     },
478     "AMBO": {
479         "enero": 1.0087, "febrero": 0.9707, "marzo": 1.1288, "abril": 1.2086,
480         "mayo": 1.0139, "junio": 1.0238, "julio": 1.0261, "agosto": 1.0008,
481         "septiembre": 0.9890, "octubre": 0.9459, "noviembre": 0.9401,
    "diciembre": 0.9497
482     },
483     "ATICO": {
484         "enero": 1.0337, "febrero": 0.9914, "marzo": 1.0298, "abril": 1.0410,
485         "mayo": 1.0342, "junio": 1.0302, "julio": 1.0277, "agosto": 0.9851,
486         "septiembre": 1.0265, "octubre": 0.9618, "noviembre": 0.9271,
    "diciembre": 0.9440
487     },
488     "AYAVIRI": {
489         "enero": 1.0167, "febrero": 1.0038, "marzo": 1.0708, "abril": 1.0573,
490         "mayo": 1.0486, "junio": 1.0366, "julio": 0.9665, "agosto": 0.9524,
491         "septiembre": 0.9501, "octubre": 0.9638, "noviembre": 0.9601,
    "diciembre": 1.0007
492     },
493     "CAMANA": {
494         "enero": 0.9513, "febrero": 0.8973, "marzo": 1.0392, "abril": 1.0648,
495         "mayo": 1.0702, "junio": 1.0905, "julio": 1.1087, "agosto": 0.9914,
496         "septiembre": 0.9997, "octubre": 0.9905, "noviembre": 0.9708,
    "diciembre": 0.9624
497     },
498     "CANCAS": {
499         "enero": 1.0323, "febrero": 0.9825, "marzo": 1.0400, "abril": 1.0412,
500         "mayo": 1.0604, "junio": 1.0476, "julio": 1.0117, "agosto": 0.9795,
501         "septiembre": 1.0006, "octubre": 0.9550, "noviembre": 0.9461,
    "diciembre": 0.9491
502     },
503     "CARACOTO": {
504         "enero": 1.0343, "febrero": 1.0013, "marzo": 1.0777, "abril": 1.0346,
505         "mayo": 1.0568, "junio": 1.0441, "julio": 0.9898, "agosto": 0.9297,
506         "septiembre": 0.9572, "octubre": 0.9481, "noviembre": 1.0071,
    "diciembre": 0.9574
507     },

```

```

508     "CASARACRA": {
509         "enero": 1.0936, "febrero": 1.0670, "marzo": 1.1094, "abril": 0.9826,
510         "mayo": 0.9904, "junio": 0.9900, "julio": 0.9969, "agosto": 0.9700,
511         "septiembre": 0.9760, "octubre": 0.9548, "noviembre": 1.0373,
512         "diciembre": 1.0652
513     },
514     "CATAC": {
515         "enero": 1.0441, "febrero": 1.0678, "marzo": 1.1999, "abril": 1.0772,
516         "mayo": 1.1736, "junio": 0.9553, "julio": 0.9446, "agosto": 0.9253,
517         "septiembre": 0.9566, "octubre": 0.9547, "noviembre": 0.9810,
518         "diciembre": 0.9634
519     },
520     "CCASACANCHA": {
521         "enero": 1.0968, "febrero": 1.0852, "marzo": 1.0957, "abril": 1.0809,
522         "mayo": 1.0239, "junio": 0.9967, "julio": 0.9386, "agosto": 0.9525,
523         "septiembre": 0.9555, "octubre": 0.9560, "noviembre": 0.9332,
524         "diciembre": 0.9566
525     },
526     "CHACAPAMPA": {
527         "enero": 1.1031, "febrero": 0.9998, "marzo": 0.9655, "abril": 0.9941,
528         "mayo": 1.0102, "junio": 1.0186, "julio": 1.0155, "agosto": 1.0051,
529         "septiembre": 0.9767, "octubre": 0.9556, "noviembre": 0.9706,
530         "diciembre": 1.0601
531     },
532     "CHALHUAPUQUIO": {
533         "enero": 1.0680, "febrero": 1.0692, "marzo": 1.0807, "abril": 1.0683,
534         "mayo": 1.0507, "junio": 0.9933, "julio": 0.9468, "agosto": 0.9321,
535         "septiembre": 0.9565, "octubre": 0.9463, "noviembre": 0.9497,
536         "diciembre": 0.9939
537     },
538     "CHICAMA": {
539         "enero": 0.9909, "febrero": 0.9874, "marzo": 1.0928, "abril": 1.0708,
540         "mayo": 1.0543, "junio": 1.0573, "julio": 1.0423, "agosto": 0.9933,
541         "septiembre": 1.0013, "octubre": 0.9946, "noviembre": 0.9510,
542         "diciembre": 0.8799
543     },
544     "CHILCA": {
545         "enero": 0.9571, "febrero": 0.9724, "marzo": 1.0203, "abril": 1.0381,
546         "mayo": 1.0583, "junio": 1.0539, "julio": 1.0372, "agosto": 0.9948,
547         "septiembre": 0.9992, "octubre": 0.9880, "noviembre": 0.9652,
548         "diciembre": 0.9396
549     },
550     "CHULLQUI": {
551         "enero": 0.9571, "febrero": 0.9658, "marzo": 1.0534, "abril": 1.0776,
552         "mayo": 1.0809, "junio": 1.0402, "julio": 1.0171, "agosto": 0.9865,
553         "septiembre": 0.9731, "octubre": 0.9169, "noviembre": 1.2400,
554         "diciembre": 0.9257
555     },
556     "CHULUCANAS": {
557         "enero": 0.9884, "febrero": 0.9682, "marzo": 1.1388, "abril": 1.1641,
558         "mayo": 1.0933, "junio": 1.0508, "julio": 1.0256, "agosto": 0.9679,
559         "septiembre": 0.9628, "octubre": 0.9205, "noviembre": 0.8943,
560         "diciembre": 0.9220
561     },
562     "CIUDAD DE DIOS": {

```



```

554         "enero": 1.4397, "febrero": 0.9383, "marzo": 1.1026, "abril": 0.9966,
555         "mayo": 0.9637, "junio": 1.0342, "julio": 1.0428, "agosto": 1.0261,
556         "septiembre": 1.0414, "octubre": 0.9841, "noviembre": 0.9930,
    "diciembre": 0.9649
557     },
558     "CORCONA": {
559         "enero": 1.0985, "febrero": 1.0709, "marzo": 1.1124, "abril": 0.9593,
560         "mayo": 0.9674, "junio": 0.9840, "julio": 0.9816, "agosto": 0.9627,
561         "septiembre": 0.9804, "octubre": 0.9767, "noviembre": 1.0641,
    "diciembre": 1.0861
562     },
563     "CRUCE BAYOVAR": {
564         "enero": 0.9845, "febrero": 0.9606, "marzo": 1.0584, "abril": 1.0655,
565         "mayo": 1.0311, "junio": 1.0361, "julio": 1.0157, "agosto": 0.9821,
566         "septiembre": 1.0167, "octubre": 0.9774, "noviembre": 0.9231,
    "diciembre": 0.9028
567     },
568     "CUCULI": {
569         "enero": 0.9424, "febrero": 1.0419, "marzo": 1.2043, "abril": 1.1560,
570         "mayo": 1.0848, "junio": 0.9783, "julio": 0.9812, "agosto": 0.9398,
571         "septiembre": 0.9207, "octubre": 0.9302, "noviembre": 0.9384,
    "diciembre": 1.0749
572     },
573     "DESVIO OLMOS": {
574         "enero": 1.0470, "febrero": 1.0401, "marzo": 1.0986, "abril": 1.0775,
575         "mayo": 1.0393, "junio": 1.0176, "julio": 0.9898, "agosto": 0.9217,
576         "septiembre": 0.9405, "octubre": 0.9579, "noviembre": 0.9500,
    "diciembre": 0.9817
577     },
578     "DESVIO TALARA": {
579         "enero": 1.0063, "febrero": 0.9722, "marzo": 1.0418, "abril": 1.0405,
580         "mayo": 1.0343, "junio": 1.0196, "julio": 1.0721, "agosto": 0.9782,
581         "septiembre": 0.9984, "octubre": 0.9792, "noviembre": 0.9631,
    "diciembre": 0.9516
582     },
583     "EL FISCAL": {
584         "enero": 0.9889, "febrero": 0.9326, "marzo": 1.0201, "abril": 1.0355,
585         "mayo": 1.0216, "junio": 1.0855, "julio": 1.0665, "agosto": 1.0209,
586         "septiembre": 1.0150, "octubre": 0.9799, "noviembre": 0.9618,
    "diciembre": 0.9819
587     },
588     "EL PARAISO": {
589         "enero": 1.0084, "febrero": 0.9860, "marzo": 1.0607, "abril": 1.0471,
590         "mayo": 1.0250, "junio": 1.0153, "julio": 1.0105, "agosto": 0.9788,
591         "septiembre": 0.9973, "octubre": 0.9801, "noviembre": 0.9625,
    "diciembre": 0.9429
592     },
593     "FORTALEZA": {
594         "enero": 0.9938, "febrero": 0.9560, "marzo": 1.0154, "abril": 1.0346,
595         "mayo": 1.0384, "junio": 1.3018, "julio": 1.0249, "agosto": 0.9743,
596         "septiembre": 0.9948, "octubre": 0.9789, "noviembre": 0.9539,
    "diciembre": 0.9356
597     },
598     "HUACRAPUQUIO": {
599         "enero": 1.0850, "febrero": 1.1264, "marzo": 1.0529, "abril": 1.0465,

```



```

600         "mayo": 1.1603, "junio": 1.0254, "julio": 0.9226, "agosto": 0.9778,
601         "septiembre": 0.9218, "octubre": 0.9085, "noviembre": 1.1194,
        "diciembre": 0.9334
602     },
603     "HUARMEY": {
604         "enero": 1.0456, "febrero": 1.0252, "marzo": 1.1132, "abril": 1.1416,
605         "mayo": 1.1272, "junio": 1.0303, "julio": 0.9991, "agosto": 0.9554,
606         "septiembre": 0.9757, "octubre": 0.9617, "noviembre": 0.9397,
        "diciembre": 0.9169
607     },
608     "HUATAYA": {
609         "enero": 0.0000, "febrero": 0.0000, "marzo": 5.1132, "abril": 1.3415,
610         "mayo": 1.0567, "junio": 1.1183, "julio": 0.7106, "agosto": 0.7730,
611         "septiembre": 0.0000, "octubre": 0.7399, "noviembre": 0.8727,
        "diciembre": 0.9802
612     },
613     "ICA": {
614         "enero": 0.9911, "febrero": 0.9493, "marzo": 1.0440, "abril": 1.0283,
615         "mayo": 1.0670, "junio": 1.0747, "julio": 1.0593, "agosto": 0.9907,
616         "septiembre": 0.9632, "octubre": 0.9523, "noviembre": 1.0104,
        "diciembre": 1.0032
617     },
618     "ILAVE": {
619         "enero": 1.0136, "febrero": 0.9470, "marzo": 0.9822, "abril": 5.5822,
620         "mayo": 1.0245, "junio": 1.0346, "julio": 1.0466, "agosto": 0.8667,
621         "septiembre": 1.0477, "octubre": 1.0417, "noviembre": 1.0885,
        "diciembre": 1.0638
622     },
623     "ILO": {
624         "enero": 1.0476, "febrero": 1.0169, "marzo": 1.0590, "abril": 0.9766,
625         "mayo": 0.9992, "junio": 1.0420, "julio": 1.0373, "agosto": 1.0284,
626         "septiembre": 0.9669, "octubre": 0.9802, "noviembre": 0.9728,
        "diciembre": 0.9625
627     },
628     "JAHUAY-CHINCHA": {
629         "enero": 0.9858, "febrero": 1.0162, "marzo": 1.0441, "abril": 1.0554,
630         "mayo": 1.0544, "junio": 1.0480, "julio": 1.0262, "agosto": 0.9755,
631         "septiembre": 0.9777, "octubre": 0.9697, "noviembre": 0.9511,
        "diciembre": 0.9478
632     },
633     "LOMA LARGA BAJA": {
634         "enero": 0.9981, "febrero": 1.1016, "marzo": 1.1995, "abril": 1.2276,
635         "mayo": 1.1103, "junio": 1.0823, "julio": 0.9596, "agosto": 0.9761,
636         "septiembre": 0.9483, "octubre": 0.9331, "noviembre": 0.8987,
        "diciembre": 0.8944
637     },
638     "LUNAHUANA": {
639         "enero": 1.1095, "febrero": 1.0967, "marzo": 1.0541, "abril": 1.0430,
640         "mayo": 0.9971, "junio": 1.0739, "julio": 1.0009, "agosto": 0.9482,
641         "septiembre": 0.9827, "octubre": 0.9329, "noviembre": 0.9403,
        "diciembre": 0.9907
642     },
643     "MACUSANI": {
644         "enero": 1.0895, "febrero": 1.0802, "marzo": 1.0669, "abril": 1.0219,
645         "mayo": 1.0756, "junio": 1.0209, "julio": 1.1741, "agosto": 0.9353,

```

```

646         "septiembre": 0.9179, "octubre": 0.9386, "noviembre": 0.9390,
        "diciembre": 0.9983
647     },
648     "MARCONA": {
649         "enero": 1.0430, "febrero": 1.0013, "marzo": 0.9583, "abril": 1.0297,
650         "mayo": 1.1134, "junio": 1.0444, "julio": 1.0480, "agosto": 1.0472,
651         "septiembre": 1.0260, "octubre": 0.9629, "noviembre": 0.9465,
        "diciembre": 0.9320
652     },
653     "MATARANI": {
654         "enero": 0.9673, "febrero": 0.8698, "marzo": 1.0297, "abril": 1.0802,
655         "mayo": 1.0657, "junio": 1.0279, "julio": 1.0254, "agosto": 1.0446,
656         "septiembre": 0.9864, "octubre": 0.9950, "noviembre": 0.9822,
        "diciembre": 0.9819
657     },
658     "MENOCUCHO": {
659         "enero": 1.0933, "febrero": 1.0653, "marzo": 1.1421, "abril": 1.0364,
660         "mayo": 0.9913, "junio": 0.9622, "julio": 0.9505, "agosto": 0.9625,
661         "septiembre": 0.9785, "octubre": 0.9841, "noviembre": 0.9862,
        "diciembre": 1.0148
662     },
663     "MOCCE": {
664         "enero": 0.9510, "febrero": 0.9816, "marzo": 1.0770, "abril": 1.1377,
665         "mayo": 1.0767, "junio": 0.9655, "julio": 1.0381, "agosto": 0.9850,
666         "septiembre": 0.9950, "octubre": 0.9641, "noviembre": 0.9558,
        "diciembre": 0.9435
667     },
668     "MONTALVO": {
669         "enero": 0.9754, "febrero": 0.9558, "marzo": 1.0155, "abril": 1.0254,
670         "mayo": 1.0095, "junio": 1.0844, "julio": 1.0785, "agosto": 1.0275,
671         "septiembre": 1.0167, "octubre": 0.9888, "noviembre": 0.9674,
        "diciembre": 0.9800
672     },
673     "MORROPE": {
674         "enero": 0.9854, "febrero": 0.9645, "marzo": 1.0770, "abril": 1.0663,
675         "mayo": 1.0358, "junio": 1.0336, "julio": 1.0257, "agosto": 1.0003,
676         "septiembre": 1.0155, "octubre": 0.9853, "noviembre": 0.9451,
        "diciembre": 0.9179
677     },
678     "MOYOBAMBA": {
679         "enero": 1.0123, "febrero": 0.0000, "marzo": 0.9812, "abril": 1.0299,
680         "mayo": 1.0077, "junio": 0.9903, "julio": 1.0012, "agosto": 0.9653,
681         "septiembre": 0.9861, "octubre": 1.0086, "noviembre": 1.0129,
        "diciembre": 1.0379
682     },
683     "NAZCA": {
684         "enero": 1.0562, "febrero": 1.0098, "marzo": 1.0329, "abril": 1.0328,
685         "mayo": 1.0325, "junio": 1.0269, "julio": 1.0128, "agosto": 0.9732,
686         "septiembre": 0.9609, "octubre": 0.9602, "noviembre": 0.9275,
        "diciembre": 1.0491
687     },
688     "PACANGUILLA": {
689         "enero": 1.0060, "febrero": 0.9822, "marzo": 1.0858, "abril": 1.0969,
690         "mayo": 1.0766, "junio": 1.0798, "julio": 1.0941, "agosto": 1.0166,

```



```

691         "septiembre": 0.9911, "octubre": 0.9429, "noviembre": 0.9142,
        "diciembre": 0.8905
692     },
693     "PACRA": {
694         "enero": 1.0832, "febrero": 1.0100, "marzo": 1.0286, "abril": 1.0364,
695         "mayo": 1.1083, "junio": 1.0029, "julio": 0.9686, "agosto": 0.9412,
696         "septiembre": 0.9706, "octubre": 0.9520, "noviembre": 0.9859,
        "diciembre": 0.9960
697     },
698     "PAITA": {
699         "enero": 1.0637, "febrero": 0.9986, "marzo": 1.0332, "abril": 1.1742,
700         "mayo": 1.1086, "junio": 1.0719, "julio": 1.1384, "agosto": 1.0669,
701         "septiembre": 0.9588, "octubre": 0.9185, "noviembre": 0.8559,
        "diciembre": 0.9255
702     },
703     "PAMPA CUELLAR": {
704         "enero": 1.1081, "febrero": 1.0770, "marzo": 1.0639, "abril": 1.0083,
705         "mayo": 1.1112, "junio": 1.0845, "julio": 0.9918, "agosto": 0.9561,
706         "septiembre": 0.9589, "octubre": 0.8916, "noviembre": 0.9121,
        "diciembre": 0.9504
707     },
708     "PAMPA GALERA": {
709         "enero": 1.1056, "febrero": 1.0973, "marzo": 1.0952, "abril": 1.0658,
710         "mayo": 1.0378, "junio": 1.0205, "julio": 0.9472, "agosto": 0.9311,
711         "septiembre": 0.9550, "octubre": 0.9441, "noviembre": 0.9360,
        "diciembre": 0.9479
712     },
713     "PAMPAMARCA": {
714         "enero": 1.0833, "febrero": 1.0608, "marzo": 1.0800, "abril": 1.0714,
715         "mayo": 1.0640, "junio": 1.0321, "julio": 0.9519, "agosto": 0.9426,
716         "septiembre": 0.9317, "octubre": 0.9501, "noviembre": 0.9398,
        "diciembre": 0.9688
717     },
718     "PATAHUASI": {
719         "enero": 1.0668, "febrero": 1.0418, "marzo": 1.0805, "abril": 1.0643,
720         "mayo": 1.0616, "junio": 1.0569, "julio": 1.0065, "agosto": 0.9440,
721         "septiembre": 0.9475, "octubre": 0.9036, "noviembre": 0.9309,
        "diciembre": 0.9274
722     },
723     "PEDRO RUIZ": {
724         "enero": 1.0361, "febrero": 1.0163, "marzo": 1.0483, "abril": 1.0529,
725         "mayo": 1.0166, "junio": 0.9901, "julio": 0.9900, "agosto": 0.9413,
726         "septiembre": 0.9610, "octubre": 0.9787, "noviembre": 0.9826,
        "diciembre": 1.0212
727     },
728     "PICHIRHUA": {
729         "enero": 1.0936, "febrero": 1.0887, "marzo": 1.1081, "abril": 1.0925,
730         "mayo": 1.0545, "junio": 1.0256, "julio": 0.9460, "agosto": 0.9321,
731         "septiembre": 0.9383, "octubre": 0.9460, "noviembre": 0.9298,
        "diciembre": 0.9399
732     },
733     "PIURA SULLANA": {
734         "enero": 1.0571, "febrero": 1.0402, "marzo": 1.1277, "abril": 1.0518,
735         "mayo": 1.0405, "junio": 1.0119, "julio": 1.0179, "agosto": 0.9763,

```



```

736         "septiembre": 0.9950, "octubre": 0.9755, "noviembre": 0.9427,
"dicembre": 0.9189
737     },
738     "PLANCHON": {
739         "enero": 1.3722, "febrero": 1.2585, "marzo": 1.1075, "abril": 1.1700,
740         "mayo": 1.0496, "junio": 1.0154, "julio": 0.9275, "agosto": 0.8807,
741         "septiembre": 0.8567, "octubre": 0.8959, "noviembre": 0.9614,
"dicembre": 1.0233
742     },
743     "POMAHUACA": {
744         "enero": 1.0674, "febrero": 1.0301, "marzo": 1.1158, "abril": 1.1109,
745         "mayo": 1.0645, "junio": 1.0235, "julio": 0.9882, "agosto": 0.9203,
746         "septiembre": 0.9522, "octubre": 0.9370, "noviembre": 0.9332,
"dicembre": 0.9668
747     },
748     "PONGO": {
749         "enero": 1.1145, "febrero": 1.0702, "marzo": 1.0868, "abril": 1.0354,
750         "mayo": 0.9963, "junio": 0.9774, "julio": 1.0443, "agosto": 0.9020,
751         "septiembre": 0.9205, "octubre": 0.9680, "noviembre": 0.9712,
"dicembre": 1.0699
752     },
753     "POZO REDONDO": {
754         "enero": 1.0157, "febrero": 0.9826, "marzo": 1.0195, "abril": 1.0517,
755         "mayo": 1.0534, "junio": 1.0459, "julio": 1.0140, "agosto": 0.9923,
756         "septiembre": 1.0345, "octubre": 1.0012, "noviembre": 0.9435,
"dicembre": 0.8997
757     },
758     "PUNTA PERDIDA": {
759         "enero": 1.1180, "febrero": 1.1218, "marzo": 1.0774, "abril": 1.0165,
760         "mayo": 1.2427, "junio": 1.1333, "julio": 1.0120, "agosto": 0.9370,
761         "septiembre": 0.9270, "octubre": 0.8749, "noviembre": 0.9204,
"dicembre": 0.9490
762     },
763     "QUIULLA": {
764         "enero": 1.1367, "febrero": 1.0810, "marzo": 1.0874, "abril": 0.9355,
765         "mayo": 0.9314, "junio": 0.9736, "julio": 0.9528, "agosto": 0.9488,
766         "septiembre": 0.9841, "octubre": 1.0038, "noviembre": 1.1007,
"dicembre": 1.1109
767     },
768     "RUMICHACA": {
769         "enero": 1.0807, "febrero": 1.0122, "marzo": 1.0302, "abril": 1.0888,
770         "mayo": 1.0366, "junio": 1.0099, "julio": 0.9690, "agosto": 0.9188,
771         "septiembre": 0.9654, "octubre": 0.9569, "noviembre": 1.0013,
"dicembre": 0.9972
772     },
773     "SAN ANTON": {
774         "enero": 1.1667, "febrero": 1.0854, "marzo": 1.0365, "abril": 1.0303,
775         "mayo": 1.0316, "junio": 1.0426, "julio": 1.0281, "agosto": 0.9802,
776         "septiembre": 0.9333, "octubre": 0.9279, "noviembre": 0.9890,
"dicembre": 0.9485
777     },
778     "SAN GABAN": {
779         "enero": 1.1273, "febrero": 1.0635, "marzo": 1.1206, "abril": 1.0840,
780         "mayo": 1.1029, "junio": 1.0740, "julio": 1.1605, "agosto": 0.8976,

```

```
781         "septiembre": 0.8572, "octubre": 0.9080, "noviembre": 0.9421,  
782         "diciembre": 1.0128  
783     },  
784     "SAN LORENZO": {  
785         "enero": 1.6588, "febrero": 1.5639, "marzo": 1.3338, "abril": 1.4442,  
786         "mayo": 1.0532, "junio": 1.0350, "julio": 0.9249, "agosto": 0.9112,  
787         "septiembre": 0.8771, "octubre": 0.8164, "noviembre": 0.8616,  
788         "diciembre": 0.9565  
789     },  
790     "SANTA LUCIA": {  
791         "enero": 1.0356, "febrero": 1.0069, "marzo": 1.0716, "abril": 1.0685,  
792         "mayo": 1.0621, "junio": 1.0856, "julio": 1.0229, "agosto": 0.9381,  
793         "septiembre": 0.9411, "octubre": 0.9138, "noviembre": 0.9453,  
794         "diciembre": 0.9392  
795     },  
796     "SAYLLA": {  
797         "enero": 1.0693, "febrero": 1.0277, "marzo": 1.0743, "abril": 1.0610,  
798         "mayo": 1.0379, "junio": 1.0317, "julio": 0.9662, "agosto": 0.9474,  
799         "septiembre": 0.9472, "octubre": 0.9579, "noviembre": 0.9470,  
800         "diciembre": 0.9774  
801     },  
802     "SERPENTIN DE PASAMAYO": {  
803         "enero": 0.9940, "febrero": 1.0077, "marzo": 1.0623, "abril": 1.0455,  
804         "mayo": 1.0254, "junio": 1.0171, "julio": 1.0086, "agosto": 0.9839,  
805         "septiembre": 0.9921, "octubre": 0.9743, "noviembre": 0.9582,  
806         "diciembre": 0.9488  
807     },  
808     "SICUYANI": {  
809         "enero": 1.1244, "febrero": 1.0164, "marzo": 1.0598, "abril": 1.0657,  
810         "mayo": 1.0985, "junio": 1.1286, "julio": 1.0861, "agosto": 0.9971,  
811         "septiembre": 0.9910, "octubre": 0.9077, "noviembre": 0.9308,  
812         "diciembre": 0.9614  
813     },  
814     "SOCOS": {  
815         "enero": 1.1347, "febrero": 1.0494, "marzo": 1.0018, "abril": 1.2878,  
816         "mayo": 1.0911, "junio": 1.0528, "julio": 1.0231, "agosto": 0.9946,  
817         "septiembre": 1.0306, "octubre": 1.0022, "noviembre": 1.0632,  
818         "diciembre": 0.9010  
819     },  
820     "TAMBOGRANDE": {  
821         "enero": 1.1347, "febrero": 1.0494, "marzo": 1.0018, "abril": 1.2878,  
822         "mayo": 1.5102, "junio": 1.2768, "julio": 1.0231, "agosto": 1.0946,  
823         "septiembre": 1.0306, "octubre": 1.0022, "noviembre": 1.0632,  
824         "diciembre": 0.9010  
825     },  
826     "TOMASIRI": {  
827         "enero": 0.9789, "febrero": 0.9106, "marzo": 0.9995, "abril": 1.0539,  
828         "mayo": 1.0515, "junio": 1.0702, "julio": 1.0580, "agosto": 1.0075,  
829         "septiembre": 0.9691, "octubre": 1.0064, "noviembre": 1.0067,  
830         "diciembre": 0.9966  
831     },  
832     "TUNAN": {  
833         "enero": 1.0648, "febrero": 1.0578, "marzo": 1.0849, "abril": 1.0647,  
834         "mayo": 1.0341, "junio": 0.9359, "julio": 0.9355, "agosto": 0.9285,
```



```

826         "septiembre": 0.9763, "octubre": 0.9641, "noviembre": 1.0216,
      "diciembre": 1.0031
827     },
828     "UNION-CONGRESO": {
829         "enero": 1.1522, "febrero": 1.1289, "marzo": 1.0613, "abril": 1.0342,
830         "mayo": 0.9994, "junio": 0.9916, "julio": 0.9577, "agosto": 0.9143,
831         "septiembre": 0.9085, "octubre": 0.9675, "noviembre": 1.0861,
      "diciembre": 0.9950
832     },
833     "UTCUBAMBA": {
834         "enero": 1.1577, "febrero": 1.0364, "marzo": 1.0667, "abril": 1.0631,
835         "mayo": 1.0240, "junio": 0.9908, "julio": 0.9829, "agosto": 0.9364,
836         "septiembre": 0.9605, "octubre": 0.9802, "noviembre": 0.9216,
      "diciembre": 1.0143
837     },
838     "VARIANTE DE PASAMAYO": {
839         "enero": 1.1672, "febrero": 0.9188, "marzo": 1.0160, "abril": 1.0768,
840         "mayo": 0.9922, "junio": 0.9811, "julio": 1.0130, "agosto": 0.9895,
841         "septiembre": 1.0586, "octubre": 1.0114, "noviembre": 0.9087,
      "diciembre": 0.9711
842     },
843     "VARIANTE DE UCHUMAYO": {
844         "enero": 1.0133, "febrero": 0.9729, "marzo": 1.0446, "abril": 1.0703,
845         "mayo": 1.0573, "junio": 1.0429, "julio": 1.0271, "agosto": 0.9797,
846         "septiembre": 0.9635, "octubre": 0.9527, "noviembre": 0.9803,
      "diciembre": 0.9711
847     },
848     "VESIQUE": {
849         "enero": 1.0255, "febrero": 0.9896, "marzo": 1.0600, "abril": 1.0910,
850         "mayo": 1.0805, "junio": 1.0745, "julio": 1.0534, "agosto": 1.0136,
851         "septiembre": 1.0300, "octubre": 0.9468, "noviembre": 0.9231,
      "diciembre": 0.8962
852     },
853     "VIRU": {
854         "enero": 1.0337, "febrero": 0.9986, "marzo": 1.0637, "abril": 1.0876,
855         "mayo": 1.0879, "junio": 1.0847, "julio": 1.0635, "agosto": 1.0165,
856         "septiembre": 1.1260, "octubre": 0.9199, "noviembre": 0.8931,
      "diciembre": 0.8927
857     },
858     "YAUCA": {
859         "enero": 1.0331, "febrero": 0.9871, "marzo": 1.0294, "abril": 1.0337,
860         "mayo": 1.0300, "junio": 1.0338, "julio": 1.0805, "agosto": 0.9767,
861         "septiembre": 0.9747, "octubre": 1.0010, "noviembre": 0.9281,
      "diciembre": 0.9422
862     }
863 }
864
865 # PESOS POR EJE PARA CADA TIPO DE VEHÍCULO (en toneladas)
866 self.pesos_por_eje = {
867     # VEHÍCULOS LIGEROS
868     "A": [1.0],          # Automóvil: 1 eje simple
869     "SW": [1.0],         # Station Wagon: 1 eje simple
870     "PU": [2.0],         # Pickup: 1 eje simple
871     "PA": [2.0],         # Panel: 1 eje simple
872     "RC": [3.0],         # Rural Combi: 1 eje simple

```



```

873 "M": [1.0, 4.0], # Micro: eje delantero + trasero
874
875 # BUSES
876 "B2E": [7.0, 11.0], # Bus 2 ejes: delantero + trasero
877 "B3E": [7.0, 16.0], # Bus 3 ejes: delantero + trasero tandem
878 "B4E": [7.0, 7.0, 16.0], # Bus 4 ejes
879
880 # CAMIONES
881 "C2E": [7.0, 11.0], # Camión 2 ejes
882 "C3E": [7.0, 18.0], # Camión 3 ejes
883 "2S1": [7.0, 11.0, 11.0], # Semirremolque
884 "2S2": [7.0, 11.0, 18.0],
885 "2S3": [7.0, 11.0, 25.0],
886 "3S1": [7.0, 18.0, 11.0],
887 "3S2": [7.0, 18.0, 18.0],
888 "3S3": [7.0, 18.0, 25.0],
889 "2T2": [7.0, 11.0, 11.0, 11.0], # Tandem
890 "2T3": [7.0, 11.0, 11.0, 18.0],
891 "3T2": [7.0, 18.0, 11.0, 11.0],
892 "3T3": [7.0, 18.0, 11.0, 18.0],
893 "4E": [7.0, 11.0, 11.0, 11.0] # 4 ejes genérico
894 }
895
896 # =====
897 # TASAS DE CRECIMIENTO POR DEPARTAMENTO - COMPLETAS (24 DEPARTAMENTOS)
898 # =====
899 self.tasas_crecimiento = {
900     "amazonas": {"ligeros": 1.12, "pesados": 2.69},
901     "ancash": {"ligeros": 2.33, "pesados": 1.49},
902     "apurimac": {"ligeros": 1.13, "pesados": 4.50},
903     "arequipa": {"ligeros": 2.90, "pesados": 2.97},
904     "ayacucho": {"ligeros": 1.83, "pesados": 2.90},
905     "cajamarca": {"ligeros": 3.05, "pesados": 1.45},
906     "cusco": {"ligeros": 2.77, "pesados": 3.07},
907     "huancavelica": {"ligeros": 1.17, "pesados": 2.00},
908     "huanuco": {"ligeros": 1.94, "pesados": 3.03},
909     "ica": {"ligeros": 1.10, "pesados": 2.62},
910     "junin": {"ligeros": 2.04, "pesados": 2.84},
911     "la libertad": {"ligeros": 2.61, "pesados": 2.21},
912     "lambayeque": {"ligeros": 2.40, "pesados": 2.54},
913     "lima provincias": {"ligeros": 2.61, "pesados": 2.34},
914     "lima": {"ligeros": 2.61, "pesados": 2.34}, # Alias para Lima Provincias
915     "loreto": {"ligeros": 0.12, "pesados": 1.48},
916     "madre de dios": {"ligeros": 2.22, "pesados": 1.38},
917     "moquegua": {"ligeros": 1.97, "pesados": 0.58},
918     "pasco": {"ligeros": 1.72, "pesados": 0.39},
919     "piura": {"ligeros": 2.38, "pesados": 2.37},
920     "puno": {"ligeros": 2.47, "pesados": 2.58},
921     "san martin": {"ligeros": 2.11, "pesados": 2.88},
922     "tacna": {"ligeros": 2.13, "pesados": 2.36},
923     "tumbes": {"ligeros": 0.81, "pesados": 2.40},
924     "ucayali": {"ligeros": 1.73, "pesados": 2.25}
925 }
926

```

```

927 # FACTORES DIRECCIONALES Y DE CARRIL
928 self.factoros_direccion_carril = {
929     # (num_calzadas, num_sentidos, carriles_por_sentido): {"fd": valor, "fc":
valor}
930     (1, 1, 1): {"fd": 1.00, "fc": 1.00}, # 1 calzada, 1 sentido, 1 carril
931     (1, 1, 2): {"fd": 1.00, "fc": 0.80}, # 1 calzada, 1 sentido, 2 carriles
932     (1, 1, 3): {"fd": 1.00, "fc": 0.60},
933     (1, 1, 4): {"fd": 1.00, "fc": 0.50},
934     (1, 2, 1): {"fd": 0.50, "fc": 1.00}, # 1 calzada, 2 sentidos, 1 carril c/u
935     (1, 2, 2): {"fd": 0.50, "fc": 0.80},
936     (1, 2, 3): {"fd": 0.50, "fc": 0.60},
937     (1, 2, 4): {"fd": 0.50, "fc": 0.50},
938     (2, 2, 1): {"fd": 0.50, "fc": 1.00}, # 2 calzadas, 2 sentidos
939     (2, 2, 2): {"fd": 0.50, "fc": 0.80},
940     (2, 2, 3): {"fd": 0.50, "fc": 0.60},
941     (2, 2, 4): {"fd": 0.50, "fc": 0.50}
942 }
943
944 # LISTA DE ESTACIONES DISPONIBLES (ordenadas alfabéticamente)
945 self.lista_estaciones = sorted(self.fce_ligeros.keys())
946
947 def clasificar_vehiculo(self, tipo):
948     """Clasificar vehículo en ligero o pesado"""
949     ligeros = ["A", "SW", "PU", "PA", "RC", "M"]
950     return "ligeros" if tipo in ligeros else "pesados"
951
952 def calcular_fce(self, tipo_vehiculo, peaje="SOCOS", mes="junio"):
953     """Calcular Factor de Corrección Estacional"""
954     clasificacion = self.clasificar_vehiculo(tipo_vehiculo)
955
956     # Normalizar nombre de estación
957     peaje_upper = peaje.upper()
958
959     if clasificacion == "ligeros":
960         return self.fce_ligeros.get(peaje_upper, {}).get(mes.lower(), 1.0)
961     else:
962         return self.fce_pesados.get(peaje_upper, {}).get(mes.lower(), 1.0)
963
964 def calcular_imd_7_dias(self, conteos_por_dia):
965     """
966     CALCULAR IMD CORRECTO: Suma de cada tipo en 7 días dividido entre 7
967     conteos_por_dia: dict {"lunes": {tipo: cantidad}, "martes": {...}, ...}
968     """
969     # Obtener todos los tipos de vehículos detectados
970     todos_tipos = set()
971     for conteo_dia in conteos_por_dia.values():
972         todos_tipos.update(conteo_dia.keys())
973
974     imd_por_tipo = {}
975
976     for tipo in todos_tipos:
977         # SUMAR TODOS LOS CONTEOS DEL TIPO EN LOS 7 DÍAS
978         suma_total_tipo = 0
979

```



```

980     for día, conteo_día in conteos_por_día.items():
981         if conteo_día: # Si hay datos para este día
982             suma_total_tipo += conteo_día.get(tipo, 0)
983
984     # IMD = SUMA TOTAL / 7 (SIEMPRE DIVIDIR ENTRE 7)
985     imd_por_tipo[tipo] = suma_total_tipo / 7.0
986
987     return imd_por_tipo
988
989 def calcular_imda(self, imd, fce):
990     """Calcular IMDA = IMD x FCE (redondeado)"""
991     imda = imd * fce
992     return round(imda + 0.5) # Redondeo hacia arriba desde 0.5
993
994 def calcular_fvp(self, tipo_vehiculo, tipo_pavimento="rígido"):
995     """Calcular Factor de Vehículo Pesado (FVP)"""
996     if tipo_vehiculo not in self.pesos_por_eje:
997         return 0.001 # Factor mínimo para vehículos no definidos
998
999     pesos = self.pesos_por_eje[tipo_vehiculo]
1000
1001     # Configurar fórmulas según tipo de pavimento
1002     if tipo_pavimento == "rígido":
1003         # PAVIMENTOS RÍGIDOS
1004         formulas = {
1005             "simple_simple": lambda p: (p / 6.6) ** 4.1,
1006             "simple_doble": lambda p: (p / 8.2) ** 4.1,
1007             "tandem_1": lambda p: (p / 13.0) ** 4.1,
1008             "tandem_2": lambda p: (p / 13.3) ** 4.1,
1009             "tridem_1": lambda p: (p / 16.6) ** 4.0,
1010             "tridem_2": lambda p: (p / 17.5) ** 4.0
1011         }
1012     else:
1013         # PAVIMENTOS FLEXIBLES
1014         formulas = {
1015             "simple_simple": lambda p: (p / 6.6) ** 4.0,
1016             "simple_doble": lambda p: (p / 8.2) ** 4.0,
1017             "tandem_1": lambda p: (p / 14.8) ** 4.0,
1018             "tandem_2": lambda p: (p / 15.1) ** 4.0,
1019             "tridem_1": lambda p: (p / 20.7) ** 3.9,
1020             "tridem_2": lambda p: (p / 21.8) ** 3.9
1021         }
1022
1023     # Calcular FVP según configuración del vehículo
1024     fvp_total = 0.0
1025
1026     # VEHÍCULOS LIGEROS - Factor muy bajo
1027     if tipo_vehiculo in ["A", "SW", "PU", "PA", "RC"]:
1028         fvp_total = 0.001
1029
1030     # MICRO (M) - Eje simple + eje simple
1031     elif tipo_vehiculo == "M":
1032         fvp_total = formulas["simple_simple"](pesos[0]) + formulas["simple_simple"]
(pesos[1])

```



```

1033
1034     # BUSES
1035     elif tipo_vehiculo == "B2E": # 7t + 11t
1036         fvp_total = formulas["simple_simple"](pesos[0]) + formulas["simple_doble"]
1037         (pesos[1])
1038     elif tipo_vehiculo == "B3E": # 7t + 16t
1039         fvp_total = formulas["simple_simple"](pesos[0]) + formulas["tandem_2"]
1040         (pesos[1])
1041
1042     # CAMIONES C2E, C3E
1043     elif tipo_vehiculo == "C2E": # 7t + 11t
1044         fvp_total = formulas["simple_simple"](pesos[0]) + formulas["simple_doble"]
1045         (pesos[1])
1046     elif tipo_vehiculo == "C3E": # 7t + 18t
1047         fvp_total = formulas["simple_simple"](pesos[0]) + formulas["tandem_1"]
1048         (pesos[1])
1049
1050     # SEMIRREMOLQUES 2S
1051     elif tipo_vehiculo in ["2S1", "2S2", "2S3"]:
1052         fvp_total = (formulas["simple_simple"](pesos[0]) +
1053                     formulas["simple_simple"](pesos[1]) +
1054                     formulas["simple_doble"](pesos[2]))
1055
1056     # SEMIRREMOLQUES 3S
1057     elif tipo_vehiculo in ["3S1", "3S2", "3S3"]:
1058         fvp_total = (formulas["simple_simple"](pesos[0]) +
1059                     formulas["tandem_1"](pesos[1]) +
1060                     formulas["simple_doble"](pesos[2]))
1061
1062     # TÁNDEM
1063     elif tipo_vehiculo in ["2T2", "2T3"]:
1064         fvp_total = (formulas["simple_simple"](pesos[0]) +
1065                     formulas["simple_simple"](pesos[1]) +
1066                     formulas["simple_simple"](pesos[2]) +
1067                     formulas["simple_doble"](pesos[3]))
1068
1069     elif tipo_vehiculo in ["3T2", "3T3"]:
1070         fvp_total = (formulas["simple_simple"](pesos[0]) +
1071                     formulas["tandem_1"](pesos[1]) +
1072                     formulas["simple_simple"](pesos[2]) +
1073                     formulas["simple_doble"](pesos[3]))
1074
1075     else:
1076         # Vehículo genérico - calcular por ejes
1077         for peso in pesos:
1078             if peso <= 8:
1079                 fvp_total += formulas["simple_simple"](peso)
1080             else:
1081                 fvp_total += formulas["simple_doble"](peso)
1082
1083     return max(fvp_total, 0.001) # Mínimo 0.001
1084
1085 def calcular_fd_fc(self, num_calzadas=1, num_sentidos=2, carriles_por_sentido=1):
1086     """Calcular factores direccionales y de carril"""

```

```

1083     config = (num_calzadas, num_sentidos, carriles_por_sentido)
1084     factores = self.factores_direccion_carril.get(config, {"fd": 0.5, "fc": 1.0})
1085     return factores["fd"], factores["fc"]
1086
1087     def calcular_fca(self, tipo_vehiculo, departamento="ayacucho", periodo_diseno=20):
1088         """Calcular Factor de Crecimiento Acumulado"""
1089         clasificacion = self.clasificar_vehiculo(tipo_vehiculo)
1090
1091         # Normalizar nombre del departamento
1092         departamento_lower = departamento.lower()
1093
1094         # Obtener tasa de crecimiento
1095         tasa_porcentaje = self.tasas_crecimiento.get(departamento_lower,
1096 {})).get(clasificacion, 2.0)
1097         r = tasa_porcentaje / 100.0 # Convertir a decimal
1098
1099         # Fórmula: Fca = [(1 + r)^n - 1] / r
1100         if r == 0:
1101             return periodo_diseno # Si no hay crecimiento
1102
1103         fca = ((1 + r) ** periodo_diseno - 1) / r
1104         return fca
1105
1106     def calcular_ee_dia_carril(self, imda, fd, fc, fvp, fp=1.0):
1107         """Calcular Ejes Equivalentes por día-carril"""
1108         return imda * fd * fc * fvp * fp
1109
1110     def calcular_nrep_ee(self, ee_dia_carril, fca):
1111         """Calcular Número de Repeticiones de Ejes Equivalentes"""
1112         return ee_dia_carril * fca * 365
1113
1114 # [RESTO DEL CÓDIGO PERMANECE IGUAL - SOLO SE MODIFICA LA INTERFAZ]
1115 # Copio las clases VehiculoTrackeado y ContadorVehicular sin cambios...
1116
1117 class VehiculoTrackeado:
1118     """Clase para representar un vehículo siendo trackeado"""
1119     def __init__(self, vehiculo_id, tipo_inicial, bbox, confianza, frame):
1120         self.id = vehiculo_id
1121         self.tipo_actual = self._limpiar_tipo_vehiculo(tipo_inicial)
1122         self.historial_tipos = defaultdict(int)
1123         self.historial_tipos[self.tipo_actual] = 1
1124         self.historial_posiciones = deque(maxlen=10)
1125         self.primer_deteccion = frame
1126         self.ultima_deteccion = frame
1127         self.frames_sin_detectar = 0
1128         self.bbox_actual = bbox
1129         self.confianza_maxima = confianza
1130         self.confianza_promedio = confianza
1131         self.total_detecciones = 1
1132         self.confirmado = False
1133
1134         centro = self._calcular_centro(bbox)
1135         self.historial_posiciones.append((centro, frame))

```



```
1136 def _limpiar_tipo_vehiculo(self, tipo):
1137     """Limpiar pero mantener denominaciones originales del modelo"""
1138     if not tipo or not isinstance(tipo, str):
1139         return "VEHICULO"
1140
1141     caracteres_no_deseados = ['?', '??', '??', ' ', '\x00', '\n', '\r', '\t']
1142
1143     tipo_limpio = str(tipo)
1144     for char in caracteres_no_deseados:
1145         tipo_limpio = tipo_limpio.replace(char, '')
1146
1147     tipo_limpio = tipo_limpio.strip()
1148
1149     if tipo_limpio and len(tipo_limpio) >= 1:
1150         return tipo_limpio
1151
1152     return "VEHICULO"
1153
1154 def _calcular_centro(self, bbox):
1155     return ((bbox[0] + bbox[2]) // 2, (bbox[1] + bbox[3]) // 2)
1156
1157 def actualizar(self, nuevo_tipo, bbox, confianza, frame):
1158     self.ultima_deteccion = frame
1159     self.frames_sin_detectar = 0
1160     self.bbox_actual = bbox
1161     self.total_detecciones += 1
1162
1163     nuevo_tipo_limpio = self._limpiar_tipo_vehiculo(nuevo_tipo)
1164     self.historial_tipos[nuevo_tipo_limpio] += 1
1165
1166     if confianza > self.confianza_maxima:
1167         self.confianza_maxima = confianza
1168         self.tipo_actual = nuevo_tipo_limpio
1169
1170     self.confianza_promedio = (self.confianza_promedio * (self.total_detecciones - 1)
+ confianza) / self.total_detecciones
1171
1172     tipo_mas_frecuente = max(self.historial_tipos.items(), key=lambda x: x[1])[0]
1173     if self.historial_tipos[tipo_mas_frecuente] >= 3:
1174         self.tipo_actual = tipo_mas_frecuente
1175
1176     centro = self._calcular_centro(bbox)
1177     self.historial_posiciones.append((centro, frame))
1178
1179     if self.total_detecciones >= 3 and self.confianza_maxima >= 0.6:
1180         self.confirmado = True
1181
1182 def incrementar_frames_sin_detectar(self):
1183     self.frames_sin_detectar += 1
1184
1185 def esta_perdido(self, max_frames_perdido=10):
1186     return self.frames_sin_detectar > max_frames_perdido
1187
1188 def distancia_a(self, bbox):
```



```

1189     centro_actual = self._calcular_centro(self.bbox_actual)
1190     centro_nuevo = self._calcular_centro(bbox)
1191     dx = centro_actual[0] - centro_nuevo[0]
1192     dy = centro_actual[1] - centro_nuevo[1]
1193     return math.sqrt(dx*dx + dy*dy)
1194
1195 class ContadorVehicular:
1196     """Motor de procesamiento de vehículos - 100% CPU GARANTIZADO"""
1197     def __init__(self, ruta_modelo="best_vehicular_model.pt"):
1198         print("\n" + "="*60)
1199         print("  CARGANDO MODELO YOLO EN CPU...")
1200         print("="*60)
1201
1202         # ASEGURAR QUE NO SE USE GPU
1203         os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
1204
1205         try:
1206             # Cargar modelo con configuración explícita de CPU
1207             self.modelo = YOLO(ruta_modelo)
1208
1209             # Forzar modelo a CPU de múltiples formas
1210             self.modelo.to('cpu')
1211
1212             # Si el modelo tiene un atributo 'model', también forzarlo a CPU
1213             if hasattr(self.modelo, 'model'):
1214                 try:
1215                     self.modelo.model.cpu()
1216                     self.modelo.model.float() # Precisión completa para CPU
1217                     print(" Modelo interno configurado en CPU")
1218                 except:
1219                     pass
1220
1221             # Verificar que torch también esté en CPU
1222             try:
1223                 import torch
1224                 torch.set_default_device('cpu')
1225                 torch.set_default_dtype(torch.float32)
1226                 print(" PyTorch configurado para CPU por defecto")
1227             except:
1228                 pass
1229
1230             print(f" Modelo cargado: {ruta_modelo}")
1231             print(f" Clases detectables: {len(self.modelo.names)}")
1232
1233         except Exception as e:
1234             print(f" Error cargando modelo: {e}")
1235             raise
1236
1237         self.vehiculos_trackeados = {}
1238         self.proximo_id = 1
1239         self.frame_actual = 0
1240         self.historial_detecciones = deque(maxlen=5)
1241
1242         # Configuración optimizada para CPU

```

```

1243     self.max_distancia_tracking = 100
1244     self.frames_umbral_consistencia = 2
1245     self.max_frames_perdido = 15
1246     self.umbral_confianza = 0.6
1247
1248     self.conteo_unico_por_clase = defaultdict(int)
1249     self.vehiculos_confirmados = []
1250     self.vehiculos_confirmados_ids = set()
1251
1252     np.random.seed(42)
1253     self.colores = {}
1254     for clase_id in self.modelo.names.keys():
1255         self.colores[clase_id] = tuple(map(int, np.random.randint(50, 255, 3)))
1256
1257     print(" ContadorVehicular inicializado correctamente")
1258     print("="*60 + "\n")
1259
1260     def resetear_contador(self):
1261         """Resetear completamente el contador para procesar nuevo video"""
1262         self.vehiculos_trackeados = {}
1263         self.proximo_id = 1
1264         self.frame_actual = 0
1265         self.historial_detecciones = deque(maxlen=5)
1266         self.conteo_unico_por_clase = defaultdict(int)
1267         self.vehiculos_confirmados = []
1268         self.vehiculos_confirmados_ids = set()
1269
1270     def filtrar_detecciones_temporal(self, detecciones_frame, detecciones_historicas,
frames_umbral=2):
1271         detecciones_filtradas = []
1272
1273         for deteccion in detecciones_frame:
1274             centro = ((deteccion['bbox'][0] + deteccion['bbox'][2]) // 2,
1275                     (deteccion['bbox'][1] + deteccion['bbox'][3]) // 2)
1276
1277             coincidencias = 0
1278             for frame_historico in detecciones_historicas:
1279                 for det_historica in frame_historico:
1280                     centro_hist = ((det_historica['bbox'][0] + det_historica['bbox'][2])
// 2,
1281                                 (det_historica['bbox'][1] + det_historica['bbox'][3])
// 2)
1282
1283                     distancia = math.sqrt((centro[0] - centro_hist[0])**2 +
1284                                           (centro[1] - centro_hist[1])**2)
1285
1286                     if distancia < self.max_distancia_tracking and deteccion['tipo'] ==
det_historica['tipo']:
1287                         coincidencias += 1
1288                         break
1289
1290             if coincidencias >= frames_umbral - 1:
1291                 detecciones_filtradas.append(deteccion)
1292

```

```

1293         return detecciones_filtradas
1294
1295     def asignar_id_vehiculo(self, detecciones_filtradas):
1296         vehiculos_actualizados = set()
1297
1298         for deteccion in detecciones_filtradas:
1299             bbox = deteccion['bbox']
1300             mejor_match = None
1301             menor_distancia = float('inf')
1302
1303             for vehiculo_id, vehiculo in self.vehiculos_trackeados.items():
1304                 if vehiculo_id in vehiculos_actualizados:
1305                     continue
1306
1307                 distancia = vehiculo.distancia_a(bbox)
1308                 if distancia < self.max_distancia_tracking and distancia <
menor_distancia:
1309                     mejor_match = vehiculo_id
1310                     menor_distancia = distancia
1311
1312                 if mejor_match:
1313                     vehiculo = self.vehiculos_trackeados[mejor_match]
1314                     vehiculo.actualizar(deteccion['tipo'], bbox, deteccion['confianza'],
self.frame_actual)
1315                     deteccion['id'] = mejor_match
1316                     vehiculos_actualizados.add(mejor_match)
1317                 else:
1318                     nuevo_id = self.proximo_id
1319                     self.proximo_id += 1
1320                     nuevo_vehiculo = VehiculoTrackeado(
1321                         nuevo_id, deteccion['tipo'], bbox,
1322                         deteccion['confianza'], self.frame_actual
1323                     )
1324                     self.vehiculos_trackeados[nuevo_id] = nuevo_vehiculo
1325                     deteccion['id'] = nuevo_id
1326
1327         for vehiculo_id, vehiculo in list(self.vehiculos_trackeados.items()):
1328             if vehiculo_id not in vehiculos_actualizados:
1329                 vehiculo.incrementar_frames_sin_detectar()
1330                 if vehiculo.esta_perdido(self.max_frames_perdido):
1331                     if vehiculo.confirmado:
1332                         self._confirmar_vehiculo_final(vehiculo)
1333                         del self.vehiculos_trackeados[vehiculo_id]
1334
1335     def _confirmar_vehiculo_final(self, vehiculo):
1336         if vehiculo.confirmado and vehiculo.id not in self.vehiculos_confirmados_ids:
1337             vehiculo_final = {
1338                 'id': vehiculo.id,
1339                 'tipo_final': vehiculo.tipo_actual,
1340                 'total_detecciones': vehiculo.total_detecciones,
1341                 'confianza_maxima': vehiculo.confianza_maxima,
1342                 'confianza_promedio': vehiculo.confianza_promedio,
1343                 'primer_frame': vehiculo.primer_deteccion,
1344                 'ultimo_frame': vehiculo.ultima_deteccion,

```



```

1345         'historial_tipos': dict(vehiculo.historial_tipos)
1346     }
1347     self.vehiculos_confirmados.append(vehiculo_final)
1348     self.vehiculos_confirmados_ids.add(vehiculo.id)
1349     self.conteo_unico_por_clase[vehiculo.tipo_actual] += 1
1350
1351 def procesar_frame(self, frame):
1352     """Procesar frame con YOLO en CPU"""
1353     self.frame_actual += 1
1354
1355     try:
1356         # MÉTODO 1: predict() con parámetros explícitos
1357         resultados = self.modelo.predict(
1358             source=frame,
1359             verbose=False,
1360             conf=self.umbral_confianza,
1361             iou=0.45,
1362             max_det=100,
1363             device='cpu', # FORZAR CPU
1364             half=False,   # NO usar half precision
1365             stream=False, # NO usar streaming
1366             save=False,   # NO guardar
1367             show=False    # NO mostrar
1368         )
1369     except Exception as e1:
1370         try:
1371             # MÉTODO 2: __call__() directo
1372             resultados = self.modelo(
1373                 source=frame,
1374                 verbose=False,
1375                 device='cpu'
1376             )
1377     except Exception as e2:
1378         print(f"❌ Error en predicción: {e1} | {e2}")
1379         return []
1380
1381     detecciones_frame = []
1382
1383     try:
1384         for r in resultados:
1385             if r.bboxes is not None and len(r.bboxes) > 0:
1386                 for box in r.bboxes:
1387                     try:
1388                         # Intentar obtener datos del tensor
1389                         xyxy = box.xyxy[0]
1390
1391                         # Convertir a numpy de forma segura
1392                         if hasattr(xyxy, 'cpu'):
1393                             xyxy = xyxy.cpu().numpy()
1394                         else:
1395                             xyxy = xyxy.numpy() if hasattr(xyxy, 'numpy') else xyxy
1396
1397                         x1, y1, x2, y2 = map(int, xyxy)
1398

```

```

1399         # Obtener confianza
1400         conf = box.conf[0]
1401         if hasattr(conf, 'cpu'):
1402             conf = conf.cpu().item()
1403         else:
1404             conf = float(conf)
1405
1406         # Obtener clase
1407         cls = box.cls[0]
1408         if hasattr(cls, 'cpu'):
1409             cls = int(cls.cpu().item())
1410         else:
1411             cls = int(cls)
1412
1413         confianza = float(conf)
1414         clase_id = int(cls)
1415
1416     except Exception as e:
1417         print(f"⚠ Error procesando box: {e}")
1418         continue
1419
1420     tipo_vehiculo = "VEHICULO"
1421
1422     try:
1423         if clase_id in self.modelo.names:
1424             tipo_raw = self.modelo.names[clase_id]
1425
1426             if tipo_raw and isinstance(tipo_raw, str):
1427                 tipo_limpio = str(tipo_raw)
1428                 caracteres_no_deseados = ['?', '??', '??', ' ',
'\x00', '\n', '\r', '\t']
1429
1430                 for char in caracteres_no_deseados:
1431                     tipo_limpio = tipo_limpio.replace(char, '')
1432
1433                 tipo_limpio = tipo_limpio.strip()
1434
1435                 if tipo_limpio and len(tipo_limpio) >= 1:
1436                     tipo_vehiculo = tipo_limpio
1437
1438     except Exception:
1439         tipo_vehiculo = "VEHICULO"
1440
1441     deteccion = {
1442         'tipo': tipo_vehiculo,
1443         'confianza': confianza,
1444         'bbox': [x1, y1, x2, y2],
1445         'clase_id': clase_id
1446     }
1447     detecciones_frame.append(deteccion)
1448
1449 except Exception as e:
1450     print(f"⚠ Error procesando resultados: {e}")
1451

```

```

1452         detecciones_filtradas = self.filtrar_detecciones_temporal(
1453             detecciones_frame, list(self.historial_detecciones),
1454             self.frames_umbral_consistencia
1455         )
1456
1457         self.asignar_id_vehiculo(detecciones_filtradas)
1458         self.historial_detecciones.append(detecciones_frame)
1459
1460         return detecciones_filtradas
1461
1462     def obtener_conteo_final(self):
1463         """Obtener conteo final del video procesado"""
1464         # Confirmar vehículos restantes
1465         for vehiculo in self.vehiculos_trackeados.values():
1466             if vehiculo.confirmado:
1467                 self._confirmar_vehiculo_final(vehiculo)
1468
1469         return dict(self.conteo_unico_por_clase)
1470
1471     def finalizar_analisis(self):
1472         for vehiculo in self.vehiculos_trackeados.values():
1473             if vehiculo.confirmado:
1474                 self._confirmar_vehiculo_final(vehiculo)
1475
1476         return {
1477             'total_vehiculos': len(self.vehiculos_confirmados),
1478             'conteo_por_clase': dict(self.conteo_unico_por_clase),
1479             'vehiculos_detallados': self.vehiculos_confirmados
1480         }
1481
1482     class InterfazSistemaEjesEquivalentes:
1483         """Interfaz optimizada con selector de estación de conteo"""
1484
1485         def __init__(self):
1486             self.root = tk.Tk()
1487             self.root.title("SISTEMA OPTIMIZADO DE EJES EQUIVALENTES v6.0 - TODAS LAS
ESTACIONES")
1488
1489             # CONFIGURACIÓN MEJORADA PARA CUALQUIER RESOLUCIÓN
1490             self.configurar_ventana_adaptativa()
1491
1492             self.root.configure(bg='#2c3e50')
1493
1494             # Variables principales
1495             self.modelo_cargado = False
1496             self.contador = None
1497             self.procesando = False
1498
1499             # CALCULADORA CON TODAS LAS ESTACIONES Y DEPARTAMENTOS
1500             self.calculadora = CalculadorEjesEquivalentes()
1501
1502             # VARIABLES PARA 7 DÍAS
1503             self.videos_por_dia = {

```



```

1504         "lunes": None,
1505         "martes": None,
1506         "miercoles": None,
1507         "jueves": None,
1508         "viernes": None,
1509         "sabado": None,
1510         "domingo": None
1511     }
1512     self.conteos_por_dia = {
1513         "lunes": {},
1514         "martes": {},
1515         "miercoles": {},
1516         "jueves": {},
1517         "viernes": {},
1518         "sabado": {},
1519         "domingo": {}
1520     }
1521     self.estados_procesamiento = {dia: "pendiente" for dia in
self.videos_por_dia.keys()}
1522
1523     # Variables para ejes equivalentes
1524     self.imd_calculado = {}
1525     self.ejes_equivalentes_resultado = None
1526
1527     # Variables para video
1528     self.video_label = None
1529     self.cap = None
1530     self.out = None
1531     self.frame_actual = 0
1532     self.total_frames = 0
1533
1534     # VARIABLES PARA CONTROL DE VELOCIDAD (FPS FIJO)
1535     self.target_fps = 30 # FPS fijo para reproducción
1536     self.frame_time = 1.0 / self.target_fps # Tiempo por frame
1537     self.last_frame_time = 0
1538
1539     # VARIABLES PARA EXPORTACIÓN DE VIDEO
1540     self.exportar_videos_procesados = tk.BooleanVar(value=True)
1541     self.carpeta_videos_procesados = None
1542
1543     # Configurar estilo y crear interfaz
1544     self.configurar_estilos()
1545     self.crear_interfaz()
1546
1547     # Cargar modelo automáticamente
1548     self.cargar_modelo()
1549
1550     def configurar_ventana_adaptativa(self):
1551         """CONFIGURACIÓN ADAPTATIVA AUTOMÁTICA"""
1552         # Obtener dimensiones de pantalla
1553         screen_width = self.root.winfo_screenwidth()
1554         screen_height = self.root.winfo_screenheight()
1555
1556         # CALCULADORA INTELIGENTE DE TAMAÑO DE VENTANA

```

```

1557 if screen_width >= 3840: # 4K
1558     window_width = min(1800, int(screen_width * 0.75))
1559     window_height = min(1200, int(screen_height * 0.80))
1560 elif screen_width >= 2560: # 2K/QHD
1561     window_width = min(1600, int(screen_width * 0.80))
1562     window_height = min(1000, int(screen_height * 0.82))
1563 elif screen_width >= 1920: # Full HD
1564     window_width = min(1400, int(screen_width * 0.85))
1565     window_height = min(900, int(screen_height * 0.85))
1566 elif screen_width >= 1600: # WSGA+
1567     window_width = min(1200, int(screen_width * 0.88))
1568     window_height = min(800, int(screen_height * 0.88))
1569 else: # 1366x768 y menores
1570     window_width = max(1200, int(screen_width * 0.95))
1571     window_height = max(650, int(screen_height * 0.90))
1572
1573 # Centrar ventana
1574 x = max(10, (screen_width - window_width) // 2)
1575 y = max(10, (screen_height - window_height) // 2)
1576
1577 self.root.geometry(f"{window_width}x{window_height}+{x}+{y}")
1578 self.root.minsize(1200, 650)
1579 self.root.resizable(True, True)
1580
1581 # Maximizar en pantallas pequeñas
1582 if screen_width <= 1366 or screen_height <= 768:
1583     try:
1584         self.root.state('zoomed' if os.name == 'nt' else 'normal')
1585     except:
1586         pass
1587
1588 # Guardar dimensiones
1589 self.screen_width = screen_width
1590 self.screen_height = screen_height
1591 self.window_width = window_width
1592 self.window_height = window_height
1593
1594 print(f"Pantalla detectada: {screen_width}x{screen_height}")
1595 print(f"Ventana configurada: {window_width}x{window_height}")
1596
1597 def configurar_estilos(self):
1598     """ESTILOS ADAPTATIVOS SEGÚN RESOLUCIÓN"""
1599     style = ttk.Style()
1600     style.theme_use('clam')
1601
1602     # TAMAÑOS DE FUENTE INTELIGENTES
1603     if self.screen_width >= 3840: # 4K
1604         base_font = 14
1605         title_font = 20
1606         subtitle_font = 16
1607     elif self.screen_width >= 2560: # 2K
1608         base_font = 12
1609         title_font = 18
1610         subtitle_font = 14

```

```

1611 elif self.screen_width >= 1920: # Full HD
1612     base_font = 11
1613     title_font = 16
1614     subtitle_font = 13
1615 elif self.screen_width >= 1600: # WSXGA+
1616     base_font = 10
1617     title_font = 14
1618     subtitle_font = 12
1619 else: # 1366x768 y menores
1620     base_font = 9
1621     title_font = 12
1622     subtitle_font = 10
1623
1624 # APLICAR ESTILOS
1625 style.configure('Title.TLabel',
1626                 background='#2c3e50',
1627                 foreground='#ecf0f1',
1628                 font=('Arial', title_font, 'bold'))
1629
1630 style.configure('Subtitle.TLabel',
1631                 background='#2c3e50',
1632                 foreground='#bdc3c7',
1633                 font=('Arial', subtitle_font, 'bold'))
1634
1635 style.configure('Custom.TButton',
1636                 font=('Arial', base_font, 'bold'))
1637
1638 style.configure('Success.TLabel',
1639                 background='#2c3e50',
1640                 foreground='#27ae60',
1641                 font=('Arial', base_font, 'bold'))
1642
1643 style.configure('Error.TLabel',
1644                 background='#2c3e50',
1645                 foreground='#e74c3c',
1646                 font=('Arial', base_font, 'bold'))
1647
1648 self.base_font = base_font
1649 self.title_font = title_font
1650 self.subtitle_font = subtitle_font
1651
1652 def crear_interfaz(self):
1653     """INTERFAZ CON DISTRIBUCIÓN INTELIGENTE"""
1654
1655     # TÍTULO COMPACTO
1656     frame_titulo = tk.Frame(self.root, bg='#34495e', height=80)
1657     frame_titulo.pack(fill='x', padx=5, pady=5)
1658     frame_titulo.pack_propagate(False)
1659
1660     tk.Label(frame_titulo,
1661             text="SISTEMA v6.0 DE EJES EQUIVALENTES E IMDA - TODAS LAS ESTACIONES Y
1662             DEPARTAMENTOS",
1663             font=('Arial', self.title_font, 'bold'),
1664             bg='#34495e', fg='#ecf0f1').pack(pady=2)

```



```

1664
1665         tk.Label(frame_titulo,
1666                 text="CÁLCULO AUTOMATIZADO | 78 ESTACIONES | 24 DEPARTAMENTOS |
METODOLOGÍA AASHTO",
1667                 font=('Arial', self.subtitle_font, 'bold'),
1668                 bg='#34495e', fg='#f39c12').pack()
1669
1670         tk.Label(frame_titulo,
1671                 text=f"CPU CONTROLADO | IMD Correcto | Resolución:
{self.screen_width}x{self.screen_height}",
1672                 font=('Arial', self.base_font - 1),
1673                 bg='#34495e', fg='#95a5a6').pack()
1674
1675         # NOTEBOOK PARA ORGANIZAR MEJOR EL ESPACIO
1676         notebook = ttk.Notebook(self.root)
1677         notebook.pack(fill='both', expand=True, padx=5, pady=5)
1678
1679         # TAB 1: PROCESAMIENTO
1680         tab_procesamiento = ttk.Frame(notebook)
1681         notebook.add(tab_procesamiento, text=" PROCESAMIENTO")
1682         self.crear_tab_procesamiento(tab_procesamiento)
1683
1684         # TAB 2: RESULTADOS
1685         tab_resultados = ttk.Frame(notebook)
1686         notebook.add(tab_resultados, text=" RESULTADOS")
1687         self.crear_tab_resultados(tab_resultados)
1688
1689         # TAB 3: EXPORTACIÓN
1690         tab_exportacion = ttk.Frame(notebook)
1691         notebook.add(tab_exportacion, text=" EXPORTACIÓN")
1692         self.crear_tab_exportacion(tab_exportacion)
1693
1694         # BARRA DE ESTADO
1695         self.status_bar = tk.Label(self.root,
1696                                   text="Sistema v6.0 inicializado - 78 estaciones + 24
departamentos disponibles",
1697                                   bg='#27ae60',
1698                                   fg='white',
1699                                   font=('Arial', self.base_font),
1700                                   anchor='w',
1701                                   padx=10,
1702                                   height=1)
1703         self.status_bar.pack(side='bottom', fill='x')
1704
1705         def crear_tab_procesamiento(self, parent):
1706             """TAB PARA PROCESAMIENTO DE VIDEOS"""
1707
1708             # Frame principal con distribución horizontal
1709             frame_principal = tk.Frame(parent, bg='#2c3e50')
1710             frame_principal.pack(fill='both', expand=True, padx=5, pady=5)
1711
1712             # COLUMNA 1: VIDEOS Y CONFIGURACIÓN (25%)
1713             frame_izquierda = tk.LabelFrame(frame_principal,
1714                                             text="CONFIGURACIÓN Y VIDEOS",

```

```

1715         font=('Arial', self.subtitle_font, 'bold'),
1716         bg='#34495e',
1717         fg='#ecf0f1',
1718         padx=5, pady=5)
1719     frame_izquierda.pack(side='left', fill='both', expand=False, padx=(0, 5))
1720     frame_izquierda.configure(width=int(self.window_width * 0.25))
1721
1722     # Estado del modelo
1723     tk.Label(frame_izquierda, text="ESTADO DEL MODELO:",
1724             bg='#34495e', fg='#ecf0f1', font=('Arial', self.base_font,
1725             'bold')).pack(anchor='w', pady=(0, 3))
1726
1727     self.label_modelo = tk.Label(frame_izquierda, text="🔄 Cargando...",
1728                                bg='#34495e', fg='#f39c12', font=('Arial',
1729                                self.base_font))
1730     self.label_modelo.pack(anchor='w', pady=(0, 10))
1731
1732     # CONFIGURACIÓN DE EXPORTACIÓN
1733     frame_export = tk.LabelFrame(frame_izquierda,
1734                                text="EXPORTACIÓN DE VIDEOS",
1735                                font=('Arial', self.base_font, 'bold'),
1736                                bg='#34495e', fg='#e67e22', padx=5, pady=5)
1737     frame_export.pack(fill='x', pady=(0, 10))
1738
1739     tk.Checkbutton(frame_export,
1740                   text="Exportar videos procesados",
1741                   variable=self.exportar_videos_procesados,
1742                   bg='#34495e', fg='#ecf0f1',
1743                   selectcolor='#2c3e50',
1744                   font=('Arial', self.base_font)).pack(anchor='w')
1745
1746     tk.Button(frame_export,
1747              text=" Seleccionar Carpeta",
1748              command=self.seleccionar_carpeta_salida,
1749              bg='#e67e22', fg='white',
1750              font=('Arial', self.base_font, 'bold')).pack(fill='x', pady=(5, 0))
1751
1752     self.label_carpeta = tk.Label(frame_export,
1753                                text=" Sin carpeta seleccionada",
1754                                bg='#34495e', fg='#95a5a6',
1755                                font=('Arial', self.base_font - 1),
1756                                wraplength=200)
1757     self.label_carpeta.pack(anchor='w', pady=(5, 0))
1758
1759     # VIDEOS POR DÍA CON SCROLL
1760     frame_videos = tk.LabelFrame(frame_izquierda,
1761                                text="ASIGNAR VIDEOS POR DÍA",
1762                                font=('Arial', self.base_font, 'bold'),
1763                                bg='#34495e', fg='#3498db', padx=5, pady=5)
1764     frame_videos.pack(fill='both', expand=True, pady=(0, 10))
1765
1766     # Canvas con scroll para videos
1767     canvas_videos = tk.Canvas(frame_videos, bg='#34495e', highlightthickness=0)

```



```

1766 scrollbar_videos = ttk.Scrollbar(frame_videos, orient="vertical",
command=canvas_videos.yview)
1767 self.frame_videos_scroll = tk.Frame(canvas_videos, bg='#34495e')
1768
1769 self.frame_videos_scroll.bind(
1770     "<Configure>",
1771     lambda e: canvas_videos.configure(scrollregion=canvas_videos.bbox("all"))
1772 )
1773
1774 canvas_videos.create_window((0, 0), window=self.frame_videos_scroll, anchor="nw")
1775 canvas_videos.configure(yscrollcommand=scrollbar_videos.set)
1776
1777 canvas_videos.pack(side="left", fill="both", expand=True)
1778 scrollbar_videos.pack(side="right", fill="y")
1779
1780 # Crear controles para cada día
1781 self.labels_dias = {}
1782 self.botones_dias = {}
1783 for dia in self.videos_por_dia.keys():
1784     self.crear_control_dia(dia)
1785
1786 # BOTONES DE CONTROL
1787 frame_control = tk.Frame(frame_izquierda, bg='#34495e')
1788 frame_control.pack(fill='x')
1789
1790 self.btn_procesar = tk.Button(frame_control,
1791     text="🔄 PROCESAR TODOS",
1792     command=self.procesar_todos_los_dias,
1793     bg='#27ae60', fg='white',
1794     font=('Arial', self.base_font, 'bold'),
1795     state='disabled')
1796 self.btn_procesar.pack(fill='x', pady=(0, 3))
1797
1798 self.btn_detener = tk.Button(frame_control,
1799     text="⏹ DETENER",
1800     command=self.detener_procesamiento,
1801     bg='#e74c3c', fg='white',
1802     font=('Arial', self.base_font, 'bold'),
1803     state='disabled')
1804 self.btn_detener.pack(fill='x', pady=(0, 3))
1805
1806 tk.Button(frame_control,
1807     text="🔄 RESET TODO",
1808     command=self.resetear_todo,
1809     bg='#95a5a6', fg='white',
1810     font=('Arial', self.base_font, 'bold')).pack(fill='x')
1811
1812 # COLUMNA 2: VIDEO EN TIEMPO REAL (50%)
1813 frame_centro = tk.LabelFrame(frame_principal,
1814     text="PROCESAMIENTO EN TIEMPO REAL",
1815     font=('Arial', self.subtitle_font, 'bold'),
1816     bg='#34495e', fg='#ecf0f1',
1817     padx=5, pady=5)
1818 frame_centro.pack(side='left', fill='both', expand=True, padx=5)

```



```

1819
1820 # ÁREA DE VIDEO ADAPTATIVA
1821 video_height = max(300, int(self.window_height * 0.4))
1822 self.video_label = tk.Label(frame_centro,
1823                             text="📺 Video aparecerá aquí durante procesamiento",
1824                             bg='#2c3e50', fg='#bdc3c7',
1825                             font=('Arial', self.subtitle_font),
1826                             height=video_height // 20)
1827 self.video_label.pack(fill='both', expand=True, pady=(0, 10))
1828
1829 # Información de progreso
1830 frame_progreso = tk.Frame(frame_centro, bg='#34495e')
1831 frame_progreso.pack(fill='x')
1832
1833 self.label_dia_actual = tk.Label(frame_progreso,
1834                                 text="Estado: Esperando videos",
1835                                 bg='#34495e', fg='#3498db',
1836                                 font=('Arial', self.subtitle_font, 'bold'))
1837 self.label_dia_actual.pack()
1838
1839 self.progress_video_var = tk.StringVar(value="Listo para procesar...")
1840 tk.Label(frame_progreso, textvariable=self.progress_video_var,
1841         bg='#34495e', fg='#ecf0f1',
1842         font=('Arial', self.base_font)).pack(pady=5)
1843
1844 self.progress_bar = ttk.Progressbar(frame_progreso, mode='determinate')
1845 self.progress_bar.pack(fill='x', pady=5)
1846
1847 self.label_export_status = tk.Label(frame_progreso,
1848                                     text="Exportación: Inactiva",
1849                                     bg='#34495e', fg='#95a5a6',
1850                                     font=('Arial', self.base_font))
1851 self.label_export_status.pack()
1852
1853 # COLUMNA 3: CONTEOS (25%)
1854 frame_derecha = tk.LabelFrame(frame_principal,
1855                               text="CONTEOS POR DÍA",
1856                               font=('Arial', self.subtitle_font, 'bold'),
1857                               bg='#34495e', fg='#ecf0f1',
1858                               padx=5, pady=5)
1859 frame_derecha.pack(side='right', fill='both', expand=False, padx=(5, 0))
1860 frame_derecha.configure(width=int(self.window_width * 0.25))
1861
1862 # Canvas con scroll para conteos
1863 canvas_conteos = tk.Canvas(frame_derecha, bg='#34495e', highlightthickness=0)
1864 scrollbar_conteos = ttk.Scrollbar(frame_derecha, orient="vertical",
command=canvas_conteos.yview)
1865 self.frame_conteos_scroll = tk.Frame(canvas_conteos, bg='#34495e')
1866
1867 self.frame_conteos_scroll.bind(
1868     "<Configure>",
1869     lambda e: canvas_conteos.configure(scrollregion=canvas_conteos.bbox("all"))
1870 )
1871

```

```

1872 canvas_conteos.create_window((0, 0), window=self.frame_conteos_scroll,
anchor="nw")
1873 canvas_conteos.configure(yscrollcommand=scrollbar_conteos.set)
1874
1875 canvas_conteos.pack(side="left", fill="both", expand=True)
1876 scrollbar_conteos.pack(side="right", fill="y")
1877
1878 self.actualizar_conteos_display()
1879
1880 def crear_control_dia(self, dia):
1881     """Crear control para cada día de la semana"""
1882     frame_dia = tk.LabelFrame(self.frame_videos_scroll,
1883                               text=dia.upper(),
1884                               font=('Arial', self.base_font, 'bold'),
1885                               bg='#34495e', fg='#ecf0f1',
1886                               padx=5, pady=5)
1887     frame_dia.pack(fill='x', pady=3)
1888
1889     # Estado del video
1890     self.labels_dias[dia] = tk.Label(frame_dia,
1891                                     text="📁 Sin video",
1892                                     bg='#34495e', fg='#95a5a6',
1893                                     font=('Arial', self.base_font - 1))
1894     self.labels_dias[dia].pack(anchor='w')
1895
1896     # Botón para seleccionar
1897     self.botonos_dias[dia] = tk.Button(frame_dia,
1898                                       text=f"Seleccionar {dia}",
1899                                       command=lambda d=dia:
self.seleccionar_video_dia(d),
1900                                       bg='#3498db', fg='white',
1901                                       font=('Arial', self.base_font - 1, 'bold'))
1902     self.botonos_dias[dia].pack(fill='x', pady=(3, 0))
1903
1904 def crear_tab_resultados(self, parent):
1905     """TAB PARA RESULTADOS Y CÁLCULOS CON SELECTOR DE ESTACIÓN"""
1906
1907     frame_principal = tk.Frame(parent, bg='#2c3e50')
1908     frame_principal.pack(fill='both', expand=True, padx=5, pady=5)
1909
1910     # COLUMNA 1: CONFIGURACIÓN (30%)
1911     frame_config = tk.LabelFrame(frame_principal,
1912                                  text="CONFIGURACIÓN DE CÁLCULO",
1913                                  font=('Arial', self.subtitle_font, 'bold'),
1914                                  bg='#34495e', fg='#f39c12',
1915                                  padx=10, pady=10)
1916     frame_config.pack(side='left', fill='both', expand=False, padx=(0, 10))
1917     frame_config.configure(width=int(self.window_width * 0.30))
1918
1919     # 🌟 NUEVO: SELECTOR DE ESTACIÓN DE CONTEO 🌟
1920     tk.Label(frame_config,
1921             text="📍 ESTACIÓN DE CONTEO:",
1922             bg='#34495e', fg='#e74c3c',
1923             font=('Arial', self.base_font + 1, 'bold')).pack(anchor='w', pady=(0, 5))

```



```

1924
1925 self.combo_estacion = ttk.Combobox(
1926     frame_config,
1927     values=self.calculadora.lista_estaciones,
1928     state="readonly",
1929     font=('Arial', self.base_font),
1930     width=30
1931 )
1932 self.combo_estacion.set("SOCOS") # Valor por defecto
1933 self.combo_estacion.pack(fill='x', pady=(0, 15))
1934
1935 # Separador visual
1936 tk.Frame(frame_config, bg='#95a5a6', height=2).pack(fill='x', pady=10)
1937
1938 # Configuración de parámetros
1939 params = [
1940     ("Departamento:", sorted([d.title() for d in
self.calculadora.tasas_crecimiento.keys()]), "Ayacucho"),
1941     ("Mes:", ["enero", "febrero", "marzo", "abril", "mayo", "junio",
1942             "julio", "agosto", "septiembre", "octubre", "noviembre",
"diciembre"], "junio"),
1943     ("Tipo de Pavimento:", ["rigido", "flexible"], "rigido"),
1944     ("Período de Diseño:", ["10", "20"], "20")
1945 ]
1946
1947 self.combos = {}
1948 for label, values, default in params:
1949     tk.Label(frame_config, text=label,
1950             bg='#34495e', fg='#ecf0f1',
1951             font=('Arial', self.base_font, 'bold')).pack(anchor='w', pady=(10,
1952 2))
1953
1954     combo = ttk.Combobox(frame_config, values=values, state="readonly",
1955                          font=('Arial', self.base_font))
1956     combo.set(default)
1957     combo.pack(fill='x', pady=(0, 5))
1958     self.combos[label.replace(":", "").replace(" ", "_").lower()] = combo
1959
1960 # Separador
1961 tk.Frame(frame_config, bg='#95a5a6', height=2).pack(fill='x', pady=10)
1962
1963 # Configuración de carretera
1964 tk.Label(frame_config, text="Configuración de Carretera:",
1965         bg='#34495e', fg='#e67e22',
1966         font=('Arial', self.base_font, 'bold')).pack(anchor='w', pady=(5, 5))
1967
1968 road_params = [
1969     ("Calzadas:", ["1", "2"], "1"),
1970     ("Sentidos:", ["1", "2"], "2"),
1971     ("Carriles por sentido:", ["1", "2", "3", "4"], "1")
1972 ]
1973
1974 for label, values, default in road_params:
1975     tk.Label(frame_config, text=label,

```



```

1975         bg='#34495e', fg='#ecf0f1',
1976         font=('Arial', self.base_font)).pack(anchor='w', pady=(5, 2))
1977
1978         combo = ttk.Combobox(frame_config, values=values, state="readonly",
1979                             font=('Arial', self.base_font))
1980         combo.set(default)
1981         combo.pack(fill='x', pady=(0, 3))
1982         self.combos[label.replace(":", "").replace(" ", "_").lower()] = combo
1983
1984     # Botón de cálculo
1985     self.btn_calcular = tk.Button(frame_config,
1986                                  text="🖨️ CALCULAR IMD + EJES EQUIVALENTES",
1987                                  command=self.calcular_imd_y_ejes_equivalentes,
1988                                  bg='#f39c12', fg='white',
1989                                  font=('Arial', self.base_font, 'bold'))
1990     self.btn_calcular.pack(fill='x', pady=(20, 0))
1991
1992     # COLUMNA 2: RESULTADOS (70%)
1993     frame_resultados = tk.LabelFrame(frame_principal,
1994                                     text="RESULTADOS DE CÁLCULO",
1995                                     font=('Arial', self.subtitle_font, 'bold'),
1996                                     bg='#34495e', fg='#27ae60',
1997                                     padx=10, pady=10)
1998     frame_resultados.pack(side='right', fill='both', expand=True)
1999
2000     # Canvas con scroll para resultados
2001     canvas_resultados = tk.Canvas(frame_resultados, bg='#34495e',
2002                                  highlightthickness=0)
2003     scrollbar_resultados = ttk.Scrollbar(frame_resultados, orient="vertical",
2004                                          command=canvas_resultados.yview)
2005     self.frame_resultados_scroll = tk.Frame(canvas_resultados, bg='#34495e')
2006
2007     # Scroll con mouse wheel
2008     def _on_mousewheel(event):
2009         canvas_resultados.yview_scroll(int(-1*(event.delta/120)), "units")
2010
2011     canvas_resultados.bind_all("<MouseWheel>", _on_mousewheel)
2012
2013     self.frame_resultados_scroll.bind(
2014         "<Configure>",
2015         lambda e:
2016             canvas_resultados.configure(scrollregion=canvas_resultados.bbox("all"))
2017     )
2018
2019     canvas_resultados.create_window((0, 0), window=self.frame_resultados_scroll,
2020                                     anchor="nw")
2021     canvas_resultados.configure(yscrollcommand=scrollbar_resultados.set)
2022
2023     canvas_resultados.pack(side="left", fill="both", expand=True)
2024     scrollbar_resultados.pack(side="right", fill="y")
2025
2026     self.mostrar_mensaje_inicial_resultados()
2027
2028     def crear_tab_exportacion(self, parent):

```

```

2025 """TAB PARA EXPORTACIÓN"""
2026
2027 frame_principal = tk.Frame(parent, bg='#2c3e50')
2028 frame_principal.pack(fill='both', expand=True, padx=20, pady=20)
2029
2030 # Título
2031 tk.Label(frame_principal,
2032         text="EXPORTACIÓN DE RESULTADOS",
2033         font=('Arial', self.title_font, 'bold'),
2034         bg='#2c3e50', fg='#ecf0f1').pack(pady=(0, 20))
2035
2036 # GRID DE BOTONES DE EXPORTACIÓN
2037 frame_botones = tk.Frame(frame_principal, bg='#2c3e50')
2038 frame_botones.pack(fill='both', expand=True)
2039
2040 # Configurar grid
2041 frame_botones.grid_columnconfigure(0, weight=1)
2042 frame_botones.grid_columnconfigure(1, weight=1)
2043 frame_botones.grid_rowconfigure(0, weight=1)
2044 frame_botones.grid_rowconfigure(1, weight=1)
2045 frame_botones.grid_rowconfigure(2, weight=1)
2046
2047 # Botones de exportación
2048 botones_export = [
2049     (" REPORTE IMD", self.exportar_reporte_imd, '#3498db'),
2050     (" CSV EJES EQUIVALENTES", self.exportar_csv_ejes, '#e67e22'),
2051     (" REPORTE COMPLETO", self.exportar_reporte_completo, '#9b59b6'),
2052     (" ABRIR VIDEOS", self.abrir_carpeta_videos_procesados, '#27ae60'),
2053     (" ABRIR CARPETA", self.abrir_carpeta_resultados, '#34495e'),
2054     (" EXPORTAR TODO", self.exportar_todo, '#e74c3c')
2055 ]
2056
2057 for i, (texto, comando, color) in enumerate(botones_export):
2058     row = i // 2
2059     col = i % 2
2060
2061     btn = tk.Button(frame_botones,
2062                    text=texto,
2063                    command=comando,
2064                    bg=color, fg='white',
2065                    font=('Arial', self.base_font + 2, 'bold'),
2066                    relief='raised',
2067                    bd=3)
2068     btn.grid(row=row, column=col, padx=10, pady=10, sticky='nsew')
2069
2070 # Información de estado
2071 frame_info = tk.LabelFrame(frame_principal,
2072                           text="INFORMACIÓN DE EXPORTACIÓN",
2073                           font=('Arial', self.subtitle_font, 'bold'),
2074                           bg='#34495e', fg='#ecf0f1',
2075                           padx=10, pady=10)
2076 frame_info.pack(fill='x', pady=(20, 0))
2077
2078 self.label_info_export = tk.Label(frame_info,

```



```

2079         text="Procese videos y calcule resultados para
habilitar exportación",
2080         bg='#34495e', fg='#95a5a6',
2081         font=('Arial', self.base_font),
2082         wraplength=600,
2083         justify='center')
2084     self.label_info_export.pack(pady=10)
2085
2086     # [CONTINÚA EN EL SIGUIENTE MENSAJE...]
2087
2088     def calcular_imd_y_ejes_equivalentes(self):
2089         """Calcular IMD y Ejes Equivalentes usando ESTACIÓN SELECCIONADA"""
2090         videos_procesados = sum(1 for conteo in self.conteos_por_dia.values() if conteo)
2091
2092         if videos_procesados == 0:
2093             messagebox.showwarning("Advertencia", "Debe procesar al menos un video")
2094             return
2095
2096         try:
2097             # Calcular IMD
2098             self.imd_calculado =
self.calculadora.calcular_imd_7_dias(self.conteos_por_dia)
2099
2100             if not self.imd_calculado or sum(self.imd_calculado.values()) == 0:
2101                 messagebox.showwarning("Advertencia", "No hay datos suficientes para
calcular IMD")
2102                 return
2103
2104             # 🌟 OBTENER ESTACIÓN SELECCIONADA 🌟
2105             estacion_seleccionada = self.combo_estacion.get()
2106
2107             # Configuración
2108             config = {
2109                 "estacion": estacion_seleccionada, # 🌟 NUEVA CONFIGURACIÓN
2110                 "peaje": estacion_seleccionada, # Alias para compatibilidad
2111                 "mes": self.combos["mes"].get(),
2112                 "departamento": self.combos["departamento"].get().lower(),
2113                 "tipo_pavimento": self.combos["tipo_de_pavimento"].get(),
2114                 "periodo_diseno": int(self.combos["período_de_diseño"].get()),
2115                 "num_calzadas": int(self.combos["calzadas"].get()),
2116                 "num_sentidos": int(self.combos["sentidos"].get()),
2117                 "carriles_por_sentido": int(self.combos["carriles_por_sentido"].get())
2118             }
2119
2120             # Factores globales
2121             fd, fc = self.calculadora.calcular_fd_fc(
2122                 config["num_calzadas"], config["num_sentidos"],
config["carriles_por_sentido"]
2123             )
2124
2125             # Calcular para cada tipo usando la estación seleccionada
2126             resultados_ee = {}
2127             total_nrep_ee = 0
2128

```



```

2129     for tipo, imd in self.imd_calculado.items():
2130         if imd == 0:
2131             continue
2132
2133         # 🌟 USAR ESTACIÓN SELECCIONADA PARA FCE
2134         fce = self.calculadora.calcular_fce(tipo, estacion_seleccionada,
config["mes"])
2135         imda = self.calculadora.calcular_imda(imd, fce)
2136         fvp = self.calculadora.calcular_fvp(tipo, config["tipo_pavimento"])
2137         ee_dia_carril = self.calculadora.calcular_ee_dia_carril(imda, fd, fc,
fvp, 1.0)
2138         fca = self.calculadora.calcular_fca(tipo, config["departamento"],
config["periodo_diseño"])
2139         nrep_ee = self.calculadora.calcular_nrep_ee(ee_dia_carril, fca)
2140
2141         total_nrep_ee += nrep_ee
2142
2143         resultados_ee[tipo] = {
2144             "imd": imd, "fce": fce, "imda": imda, "fd": fd, "fc": fc,
2145             "fvp": fvp, "fp": 1.0, "ee_dia_carril": ee_dia_carril,
2146             "fca": fca, "nrep_ee": nrep_ee,
2147             "clasificacion": self.calculadora.clasificar_vehiculo(tipo)
2148         }
2149
2150         # Guardar resultados
2151         self.ejes_equivalentes_resultado = {
2152             "resultados_por_tipo": resultados_ee,
2153             "total_nrep_ee": total_nrep_ee,
2154             "configuracion": config,
2155             "factores_globales": {"fd": fd, "fc": fc},
2156             "videos_procesados": videos_procesados,
2157             "imd_calculado": self.imd_calculado
2158         }
2159
2160         # Mostrar resultados
2161         self.mostrar_resultados_calculados()
2162
2163         self.actualizar_status(f" Cálculo completado - Estación:
{estacion_seleccionada}")
2164
2165         messagebox.showinfo("Éxito",
2166                             f" CÁLCULO COMPLETADO\n\n"
2167                             f"Estación: {estacion_seleccionada}\n"
2168                             f"Videos procesados: {videos_procesados}/7\n"
2169                             f"Total NREP EE: {int(total_nrep_ee):,}")
2170
2171     except Exception as e:
2172         messagebox.showerror("Error", f"Error en cálculo:\n{str(e)}")
2173
2174     def mostrar_resultados_calculados(self):
2175         """Mostrar resultados calculados"""
2176         if not self.ejes_equivalentes_resultado:
2177             return
2178

```

```

2179 # Limpiar
2180 for widget in self.frame_resultados_scroll.winfo_children():
2181     widget.destroy()
2182
2183 resultados = self.ejes_equivalentes_resultado["resultados_por_tipo"]
2184 total_ee = self.ejes_equivalentes_resultado["total_nrep_ee"]
2185 config = self.ejes_equivalentes_resultado["configuracion"]
2186
2187 # Mostrar estación utilizada
2188 frame_estacion = tk.LabelFrame(self.frame_resultados_scroll,
2189                                text="ESTACIÓN DE CONTEO",
2190                                bg='#34495e', fg='#e74c22',
2191                                font=('Arial', self.base_font + 1, 'bold'),
2192                                padx=10, pady=10)
2193 frame_estacion.pack(fill='x', pady=(0, 15))
2194
2195 tk.Label(frame_estacion,
2196          text=f" {config['estacion']}",
2197          bg='#34495e', fg='#f39c12',
2198          font=('Arial', self.base_font + 3, 'bold')).pack()
2199
2200 # IMD Calculado
2201 frame_imd = tk.LabelFrame(self.frame_resultados_scroll,
2202                           text="IMD CALCULADO (Suma 7 días / 7)",
2203                           bg='#34495e', fg='#3498db',
2204                           font=('Arial', self.base_font + 1, 'bold'),
2205                           padx=10, pady=10)
2206 frame_imd.pack(fill='x', pady=(0, 15))
2207
2208 for tipo, imd in sorted(self.imd_calculado.items(), key=lambda x: x[1],
2209                        reverse=True):
2210     if imd > 0:
2211         tk.Label(frame_imd,
2212                  text=f"{tipo}: {imd:.2f} veh/día",
2213                  bg='#34495e', fg='#3498db',
2214                  font=('Arial', self.base_font)).pack(anchor='w', pady=2)
2215
2216 # Ejes Equivalentes por tipo
2217 frame_ee = tk.LabelFrame(self.frame_resultados_scroll,
2218                           text="EJES EQUIVALENTES POR TIPO",
2219                           bg='#34495e', fg='#f39c12',
2220                           font=('Arial', self.base_font + 1, 'bold'),
2221                           padx=10, pady=10)
2222 frame_ee.pack(fill='x', pady=(0, 15))
2223
2224 for tipo, datos in sorted(resultados.items(), key=lambda x: x[1]["nrep_ee"],
2225                          reverse=True):
2226     if datos["nrep_ee"] > 0:
2227         frame_tipo = tk.LabelFrame(frame_ee,
2228                                     text=tipo,
2229                                     bg='#34495e', fg='#ecf0f1',
2230                                     font=('Arial', self.base_font, 'bold'),
2231                                     padx=8, pady=8)
2232         frame_tipo.pack(fill='x', pady=5)

```



```

2231
2232         # Crear grid de información
2233         info_grid = tk.Frame(frame_tipo, bg='#34495e')
2234         info_grid.pack(fill='x')
2235
2236         # Fila 1
2237         tk.Label(info_grid, text=f"IMD: {datos['imd']:.2f}",
2238                 bg='#34495e', fg='#3498db', font=('Arial', self.base_font -
2239 1)).grid(row=0, column=0, sticky='w', padx=5)
2239         tk.Label(info_grid, text=f"IMDA: {datos['imda']}",
2240                 bg='#34495e', fg='#3498db', font=('Arial', self.base_font -
2241 1)).grid(row=0, column=1, sticky='w', padx=5)
2241
2242         # Fila 2
2243         tk.Label(info_grid, text=f"FVP: {datos['fvp']:.4f}",
2244                 bg='#34495e', fg='#e67e22', font=('Arial', self.base_font -
2245 1)).grid(row=1, column=0, sticky='w', padx=5)
2245         tk.Label(info_grid, text=f"FCE: {datos['fce']:.4f}",
2246                 bg='#34495e', fg='#e67e22', font=('Arial', self.base_font -
2247 1)).grid(row=1, column=1, sticky='w', padx=5)
2247
2248         # Fila 3
2249         tk.Label(info_grid, text=f"EE/día: {datos['ee_día_carril']:.3f}",
2250                 bg='#34495e', fg='#9b59b6', font=('Arial', self.base_font -
2251 1)).grid(row=2, column=0, sticky='w', padx=5)
2251         tk.Label(info_grid, text=f"FCA: {datos['fca']:.2f}",
2252                 bg='#34495e', fg='#9b59b6', font=('Arial', self.base_font -
2253 1)).grid(row=2, column=1, sticky='w', padx=5)
2253
2254         # NREP EE destacado
2255         tk.Label(frame_tipo,
2256                 text=f"NREP EE: {int(datos['nrep_ee']):,}",
2257                 bg='#34495e', fg='#f39c12',
2258                 font=('Arial', self.base_font + 1, 'bold')).pack(pady=(10, 0))
2259
2260         # Total final
2261         frame_total = tk.Frame(self.frame_resultados_scroll, bg='#2c3e50',
2262 relief='raised', bd=3)
2263         frame_total.pack(fill='x', pady=15)
2264
2265         tk.Label(frame_total,
2266                 text="TOTAL NREP EE (DISEÑO):",
2267                 bg='#2c3e50', fg='#f39c12',
2268                 font=('Arial', self.base_font + 2, 'bold')).pack(pady=5)
2269
2270         tk.Label(frame_total,
2271                 text=f"{int(total_ee):,}",
2272                 bg='#2c3e50', fg='#f39c12',
2273                 font=('Arial', self.base_font + 4, 'bold')).pack(pady=5)
2274
2275         # Configuración utilizada
2276         frame_config_res = tk.LabelFrame(self.frame_resultados_scroll,
2277                 text="CONFIGURACIÓN UTILIZADA",
2278                 bg='#34495e', fg='#95a5a6',

```



```

2278         font=('Arial', self.base_font, 'bold'),
2279         padx=10, pady=10)
2280     frame_config_res.pack(fill='x', pady=(15, 0))
2281
2282     info_config = [
2283         f"Estación: {config['estacion']}", # NUEVO
2284         f"Departamento: {config['departamento'].title()}",
2285         f"Mes: {config['mes'].title()}",
2286         f"Pavimento: {config['tipo_pavimento'].title()}",
2287         f"Período: {config['periodo_diseno']} años",
2288         f"Videos procesados: {self.ejes_equivalentes_re-
sultado['videos_procesados']}/7",
2289         f"FD: {self.ejes_equivalentes_resultado['factores_globales']['fd']:.2f} | FC:
{self.ejes_equivalentes_resultado['factores_globales']['fc']:.2f}"
2290     ]
2291
2292     for item in info_config:
2293         tk.Label(frame_config_res, text=item, bg='#34495e', fg='#95a5a6',
2294                 font=('Arial', self.base_font - 1)).pack(anchor='w', pady=1)
2295
2296     # [RESTO DE MÉTODOS PERMANECEN IGUALES - Los copio del código original...]
2297
2298     def cargar_modelo(self):
2299         """Cargar modelo YOLO OPTIMIZADO PARA CPU"""
2300         def cargar():
2301             try:
2302                 modelos = ["best_vehicular_model.pt", "best_yuvica.pt", "best.pt"]
2303                 modelo_encontrado = None
2304
2305                 for modelo in modelos:
2306                     if os.path.exists(modelo):
2307                         modelo_encontrado = modelo
2308                         break
2309
2310                 if modelo_encontrado:
2311                     self.contador = ContadorVehicular(modelo_encontrado)
2312                     self.modelo_cargado = True
2313
2314                     clases = list(self.contador.modelo.names.values())
2315                     texto_clases = ", ".join(clases[:5]) + ("..." if len(clases) > 5 else
""")
2316
2317                     self.root.after(0, lambda: self.label_modelo.configure(
2318                         text=f" Modelo CPU: {texto_clases}", fg='#27ae60'))
2319                     self.root.after(0, self.verificar_estado_botones)
2320                 else:
2321                     self.root.after(0, lambda: self.label_modelo.configure(
2322                         text=" Modelo no encontrado", fg='#e74c3c'))
2323
2324             except Exception as e:
2325                 self.root.after(0, lambda: self.label_modelo.configure(
2326                     text=f" Error: {str(e)[:30]}...", fg='#e74c3c'))
2327
2328         threading.Thread(target=cargar, daemon=True).start()

```

```

2329
2330 def seleccionar_carpeta_salida(self):
2331     """Seleccionar carpeta para videos exportados"""
2332     carpeta = filedialog.askdirectory(title="Seleccionar carpeta para videos
procesados")
2333
2334     if carpeta:
2335         self.carpeta_videos_procesados = carpeta
2336         nombre = os.path.basename(carpeta) or carpeta
2337         self.label_carpeta.configure(text=f" {nombre}", fg='#27ae60')
2338         self.actualizar_status(f" Carpeta configurada: {carpeta}")
2339     else:
2340         self.carpeta_videos_procesados = os.getcwd()
2341         self.label_carpeta.configure(text=" Carpeta actual", fg='#f39c12')
2342
2343 def seleccionar_video_dia(self, dia):
2344     """Seleccionar video para un día"""
2345     video = filedialog.askopenfilename(
2346         title=f"Seleccionar video para {dia.upper()}",
2347         filetypes=[
2348             ("Videos", "*.mp4 *.avi *.mov *.mkv *.wmv"),
2349             ("Todos", "*.*")
2350         ]
2351     )
2352
2353     if video:
2354         self.videos_por_dia[dia] = video
2355         nombre = os.path.basename(video)
2356         max_chars = 25
2357         texto = f"📺 {nombre}" if len(nombre) <= max_chars else f"📺
{nombre[:max_chars-3]}..."
2358         self.labels_dias[dia].configure(text=texto, fg='#27ae60')
2359         self.verificar_estado_botones()
2360         self.actualizar_status(f" Video asignado a {dia}")
2361
2362 def verificar_estado_botones(self):
2363     """Verificar estado de botones"""
2364     videos_asignados = sum(1 for v in self.videos_por_dia.values() if v is not None)
2365
2366     if self.modelo_cargado and videos_asignados > 0:
2367         self.btn_procesar.configure(state='normal')
2368     else:
2369         self.btn_procesar.configure(state='disabled')
2370
2371 def procesar_todos_los_dias(self):
2372     """Procesar todos los videos"""
2373     if not self.modelo_cargado:
2374         messagebox.showwarning("Advertencia", "El modelo no está cargado")
2375         return
2376
2377     videos_para_procesar = [(dia, video) for dia, video in
self.videos_por_dia.items() if video]
2378
2379     if not videos_para_procesar:

```



```

2380     messagebox.showwarning("Advertencia", "No hay videos para procesar")
2381     return
2382
2383     if self.exportar_videos_procesados.get() and not self.carpeta_videos_procesados:
2384         if messagebox.askyesno("Carpeta", "¿Usar carpeta actual para videos?"):
2385             self.carpeta_videos_procesados = os.getcwd()
2386         else:
2387             return
2388
2389     self.procesando = True
2390     self.btn_procesar.configure(state='disabled')
2391     self.btn_detener.configure(state='normal')
2392
2393     threading.Thread(target=self.procesar_videos_secuencial,
2394                     args=(videos_para_procesar,), daemon=True).start()
2395
2396     def procesar_videos_secuencial(self, videos_para_procesar):
2397         """Procesar videos secuencialmente CON CONTROL DE VELOCIDAD"""
2398         total_videos = len(videos_para_procesar)
2399
2400         for i, (dia, video_path) in enumerate(videos_para_procesar):
2401             if not self.procesando:
2402                 break
2403
2404             self.root.after(0, lambda d=dia, n=i+1, t=total_videos:
2405                             self.actualizar_status(f"🖥️ Procesando {d} ({n}/{t})"))
2406             self.root.after(0, lambda d=dia:
2407                             self.label_dia_actual.configure(text=f"Procesando:
2408 {d.upper()}"))
2409
2410             self.contador.resetear_contador()
2411             resultado = self.procesar_video_individual(dia, video_path)
2412
2413             if resultado:
2414                 self.conteos_por_dia[dia] = resultado
2415                 self.estados_procesamiento[dia] = "completado"
2416                 self.root.after(0, self.actualizar_conteos_display)
2417                 self.root.after(0, lambda d=dia: self.actualizar_status_dia(d,
2418 "completado"))
2419             else:
2420                 self.estados_procesamiento[dia] = "error"
2421                 self.root.after(0, lambda d=dia: self.actualizar_status_dia(d, "error"))
2422
2423             if self.procesando:
2424                 self.root.after(0, lambda: self.actualizar_status(" Procesamiento
2425 completado"))
2426                 self.root.after(0, lambda: self.label_dia_actual.configure(text="
2427 Completado"))
2428
2429             self.root.after(0, self.finalizar_procesamiento)
2430
2431     def procesar_video_individual(self, dia, video_path):
2432         """Procesar un video individual CON CONTROL DE VELOCIDAD"""
2433         try:

```



```

2430 self.cap = cv2.VideoCapture(video_path)
2431 if not self.cap.isOpened():
2432     return None
2433
2434 # Configuración del video
2435 self.total_frames = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
2436 fps_original = int(self.cap.get(cv2.CAP_PROP_FPS)) or 30
2437 width = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
2438 height = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
2439 self.frame_actual = 0
2440
2441 print(f"Video: {dia} | FPS original: {fps_original} | Resolución:
{width}x{height}")
2442 print(f"FPS reproducción: 30 (fijo)")
2443
2444 # Configurar exportación si está habilitada
2445 writer_video = None
2446 if self.exportar_videos_procesados.get() and self.carpeta_videos_procesados:
2447     timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
2448     nombre_salida = f"EjesEquivalentes_{dia}_{timestamp}.mp4"
2449     ruta_salida = os.path.join(self.carpeta_videos_procesados, nombre_salida)
2450
2451 # CONFIGURACIÓN MEJORADA PARA EXPORTACIÓN
2452 fourcc = cv2.VideoWriter_fourcc(*'mp4v')
2453 writer_video = cv2.VideoWriter(ruta_salida, fourcc, 30, (width, height))
2454
2455 if writer_video.isOpened():
2456     self.root.after(0, lambda: self.label_export_status.configure(
2457         text=f"Exportando: {dia}", fg='#e67e22'))
2458     print(f"Video de salida configurado: {ruta_salida}")
2459 else:
2460     writer_video = None
2461     print("Error: No se pudo abrir writer de video")
2462
2463 # INICIALIZAR CONTROL DE TIEMPO
2464 self.last_frame_time = time.time()
2465
2466 while True:
2467     if not self.procesando:
2468         break
2469
2470     ret, frame = self.cap.read()
2471     if not ret:
2472         break
2473
2474     # PROCESAR FRAME CON YOLO
2475     detecciones = self.contador.procesar_frame(frame)
2476
2477     # CREAR FRAME CON DETECCIONES PARA EXPORTACIÓN
2478     if writer_video:
2479         frame_export = self.dibujar_detecciones_exportacion(frame.copy(),
detecciones, dia)
2480         writer_video.write(frame_export)
2481

```

```

2482         # ACTUALIZAR UI CON CONTROL DE VELOCIDAD
2483         current_time = time.time()
2484         elapsed_time = current_time - self.last_frame_time
2485
2486         # Actualizar UI solo si ha pasado el tiempo necesario según FPS
configurado
2487         if elapsed_time >= self.frame_time:
2488             frame_ui = self.dibujar_detecciones_ui(frame.copy(), detecciones,
dia)
2489             self.root.after(0, lambda f=frame_ui: self.actualizar_video_ui(f))
2490
2491             progreso = (self.frame_actual / self.total_frames) * 100
2492             self.root.after(0, lambda p=progreso:
self.progress_bar.configure(value=p))
2493             self.root.after(0, lambda: self.progress_video_var.set(
2494                 f"Frame {self.frame_actual}/{self.total_frames} ({progreso:.1f}%)
- 30 FPS"))
2495
2496             self.last_frame_time = current_time
2497
2498         # CONTROL DE VELOCIDAD: Esperar el tiempo necesario
2499         frame_process_time = time.time() - current_time
2500         sleep_time = max(0, self.frame_time - frame_process_time)
2501         if sleep_time > 0:
2502             time.sleep(sleep_time)
2503
2504         self.frame_actual += 1
2505
2506         self.cap.release()
2507
2508         if writer_video:
2509             writer_video.release()
2510             self.root.after(0, lambda: self.label_export_status.configure(
2511                 text=f"Video {dia} exportado", fg='#27ae60'))
2512             print(f"Video exportado exitosamente para {dia}")
2513
2514         return self.contador.obtener_conteo_final()
2515
2516     except Exception as e:
2517         error_msg = f"Error en {dia}: {str(e)}"
2518         self.root.after(0, lambda: self.actualizar_status(error_msg))
2519         print(f"Error procesando video {dia}: {e}")
2520         if hasattr(self, 'cap') and self.cap:
2521             self.cap.release()
2522         if 'writer_video' in locals() and writer_video:
2523             writer_video.release()
2524         return None
2525
2526     # [MÉTODOS DE VISUALIZACIÓN Y EXPORTACIÓN PERMANECEN IGUALES...]
2527
2528     def dibujar_detecciones_exportacion(self, frame, detecciones, dia):
2529         """Dibujar detecciones para video exportado"""
2530         height, width = frame.shape[:2]
2531

```



```

2532 # Fondo para información superior
2533 overlay = frame.copy()
2534 cv2.rectangle(overlay, (0, 0), (width, 120), (0, 0, 0), -1)
2535 cv2.addWeighted(overlay, 0.8, frame, 0.2, 0, frame)
2536
2537 # Información del sistema
2538 progreso = (self.frame_actual / self.total_frames) * 100 if self.total_frames > 0
else 0
2539 total_detectados = sum(self.contador.conteo_unico_por_clase.values())
2540
2541 textos = [
2542     f"SISTEMA EJES EQUIVALENTES v6.0 - {dia.upper()}",
2543     f"Frame: {self.frame_actual}/{self.total_frames} ({progreso:.1f}%)",
2544     f"Total detectados: {total_detectados} | FPS: 30",
2545     f"Fecha: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
2546 ]
2547
2548 for i, texto in enumerate(textos):
2549     y_pos = 25 + (i * 22)
2550     cv2.putText(frame, texto, (15, y_pos),
2551                 cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
2552
2553 # DIBUJAR DETECCIONES CON BOUNDING BOXES GRANDES
2554 for deteccion in detecciones:
2555     if 'id' in deteccion:
2556         vehiculo = self.contador.vehiculos_trackeados[deteccion['id']]
2557         bbox = deteccion['bbox']
2558         color = self.contador.colores[deteccion['clase_id']]
2559
2560         # BOUNDING BOX MÁS GRUESO Y VISIBLE
2561         cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), color, 6)
2562
2563         # ETIQUETA MÁS GRANDE Y VISIBLE
2564         etiqueta = f"ID:{deteccion['id']} {vehiculo.tipo_actual}"
2565         confianza = f"{int(vehiculo.confianza_maxima * 100)}%"
2566
2567         # FONDO MÁS GRANDE PARA ETIQUETA
2568         (tw, th), _ = cv2.getTextSize(etiqueta, cv2.FONT_HERSHEY_SIMPLEX, 1.0, 3)
2569         (cw, ch), _ = cv2.getTextSize(confianza, cv2.FONT_HERSHEY_SIMPLEX, 0.8,
2)
2570
2571         # Calcular dimensiones del fondo
2572         fondo_width = max(tw, cw) + 20
2573         fondo_height = th + ch + 25
2574
2575         # FONDO DE ETIQUETA MÁS VISIBLE
2576         cv2.rectangle(frame,
2577                       (bbox[0] - 5, bbox[1] - fondo_height - 5),
2578                       (bbox[0] + fondo_width, bbox[1] + 5),
2579                       color, -1)
2580
2581         # BORDE DEL FONDO
2582         cv2.rectangle(frame,
2583                       (bbox[0] - 5, bbox[1] - fondo_height - 5),

```



```

2584         (bbox[0] + fondo_width, bbox[1] + 5),
2585         (255, 255, 255), 2)
2586
2587     # TEXTO DE ETIQUETA MÁS GRANDE
2588     cv2.putText(frame, etiqueta,
2589                 (bbox[0] + 5, bbox[1] - ch - 10),
2590                 cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), 3)
2591
2592     # TEXTO DE CONFIANZA
2593     cv2.putText(frame, confianza,
2594                 (bbox[0] + 5, bbox[1] - 5),
2595                 cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
2596
2597     # REGISTRO DE CONTEO ACTUAL
2598     conteo = self.contador.conteo_unico_por_clase
2599     if conteo:
2600         conteos_activos = {tipo: cantidad for tipo, cantidad in conteo.items() if
cantidad > 0}
2601
2602     if conteos_activos:
2603         max_items = len(conteos_activos)
2604         panel_height = max(80, (max_items * 35) + 40)
2605         panel_width = 280
2606
2607         panel_x = width - panel_width - 20
2608         panel_y = 140
2609
2610         # FONDO DEL PANEL DE CONTEO
2611         cv2.rectangle(frame,
2612                       (panel_x, panel_y),
2613                       (panel_x + panel_width, panel_y + panel_height),
2614                       (0, 0, 0), -1)
2615
2616         # BORDE DEL PANEL
2617         cv2.rectangle(frame,
2618                       (panel_x, panel_y),
2619                       (panel_x + panel_width, panel_y + panel_height),
2620                       (255, 255, 255), 3)
2621
2622         # TÍTULO DEL PANEL
2623         cv2.putText(frame, "CONTEO ACTUAL:",
2624                     (panel_x + 10, panel_y + 30),
2625                     cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
2626
2627         # LÍNEA SEPARADORA
2628         cv2.line(frame,
2629                 (panel_x + 10, panel_y + 40),
2630                 (panel_x + panel_width - 10, panel_y + 40),
2631                 (255, 255, 255), 2)
2632
2633         # MOSTRAR CONTEOS
2634         y_offset = 65
2635         for tipo, cantidad in sorted(conteos_activos.items(), key=lambda x: x[1],
reverse=True):

```

```

2636         if tipo in ['A', 'SW', 'PU', 'PA']:
2637             color_tipo = (0, 255, 255) # Amarillo
2638         elif tipo in ['B2E', 'B3E', 'B4E']:
2639             color_tipo = (0, 165, 255) # Naranja
2640         elif tipo in ['C2E', 'C3E']:
2641             color_tipo = (0, 0, 255) # Rojo
2642         else:
2643             color_tipo = (255, 0, 255) # Magenta
2644
2645         texto_conteo = f"{tipo}: {cantidad}"
2646         cv2.putText(frame, texto_conteo,
2647                     (panel_x + 15, panel_y + y_offset),
2648                     cv2.FONT_HERSHEY_SIMPLEX, 0.9, color_tipo, 2)
2649
2650         y_offset += 35
2651
2652     # TOTAL GENERAL
2653     total_general = sum(conteos_activos.values())
2654     cv2.line(frame,
2655              (panel_x + 10, panel_y + y_offset - 15),
2656              (panel_x + panel_width - 10, panel_y + y_offset - 15),
2657              (255, 255, 255), 2)
2658
2659     cv2.putText(frame, f"TOTAL: {total_general}",
2660                 (panel_x + 15, panel_y + y_offset + 10),
2661                 cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 255, 0), 3)
2662
2663     return frame
2664
2665 def dibujar_detecciones_ui(self, frame, detecciones, dia):
2666     """Dibujar detecciones para UI"""
2667     height, width = frame.shape[:2]
2668
2669     # Información básica
2670     overlay = frame.copy()
2671     cv2.rectangle(overlay, (0, 0), (width, 60), (0, 0, 0), -1)
2672     cv2.addWeighted(overlay, 0.7, frame, 0.3, 0, frame)
2673
2674     progreso = (self.frame_actual / self.total_frames) * 100 if self.total_frames > 0
2675 else 0
2676     total = sum(self.contador.conteo_unico_por_clase.values())
2677
2678     texto = f"CPU v6.0: {dia.upper()} | Frame: {self.frame_actual} | {progreso:.1f}%
2679 | Total: {total}"
2680     cv2.putText(frame, texto, (10, 35), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255,
2681 255), 2)
2682
2683     # DETECCIONES
2684     for deteccion in detecciones:
2685         if 'id' in deteccion:
2686             vehiculo = self.contador.vehiculos_trackeados[deteccion['id']]
2687             bbox = deteccion['bbox']
2688             color = self.contador.colores[deteccion['clase_id']]

```

```

2687         cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), color, 4)
2688
2689         etiqueta = f"ID:{deteccion['id']} {vehiculo.tipo_actual}"
2690
2691         (tw, th), _ = cv2.getTextSize(etiqueta, cv2.FONT_HERSHEY_SIMPLEX, 0.8, 2)
2692         cv2.rectangle(frame,
2693             (bbox[0] - 2, bbox[1] - th - 8),
2694             (bbox[0] + tw + 4, bbox[1] + 2),
2695             color, -1)
2696
2697         cv2.putText(frame, etiqueta,
2698             (bbox[0], bbox[1] - 4),
2699             cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
2700
2701     return frame
2702
2703 def actualizar_video_ui(self, frame):
2704     """Actualizar video en interfaz"""
2705     try:
2706         height, width = frame.shape[:2]
2707
2708         label_width = self.video_label.winfo_width()
2709         label_height = self.video_label.winfo_height()
2710
2711         if label_width <= 1:
2712             max_width = int(self.window_width * 0.45)
2713             max_height = max(300, int(self.window_height * 0.4))
2714         else:
2715             max_width = label_width - 10
2716             max_height = label_height - 10
2717
2718         scale = min(max_width/width, max_height/height)
2719         new_width = int(width * scale)
2720         new_height = int(height * scale)
2721
2722         frame_resized = cv2.resize(frame, (new_width, new_height))
2723         frame_rgb = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2RGB)
2724         image_pil = Image.fromarray(frame_rgb)
2725         image_tk = ImageTk.PhotoImage(image_pil)
2726
2727         self.video_label.configure(image=image_tk)
2728         self.video_label.image = image_tk
2729
2730     except Exception as e:
2731         print(f"Error actualizando video: {e}")
2732
2733 def actualizar_status_dia(self, dia, estado):
2734     """Actualizar estado de un día"""
2735     if estado == "completado":
2736         total = sum(self.conteos_por_dia[dia].values()) if self.conteos_por_dia[dia]
2737     else 0
2738
2739     self.labels_dias[dia].configure(text=f" {total} vehículos", fg='#27ae60')
2740     elif estado == "error":
2741         self.labels_dias[dia].configure(text=" Error", fg='#e74c3c')

```



```

2740
2741 def actualizar_conteos_display(self):
2742     """Actualizar display de conteos"""
2743     for widget in self.frame_conteos_scroll.winfo_children():
2744         widget.destroy()
2745
2746     tk.Label(self.frame_conteos_scroll,
2747             text="CONTEOS POR DÍA:",
2748             bg='#34495e', fg='#ecf0f1',
2749             font=('Arial', self.base_font + 1, 'bold')).pack(anchor='w', pady=(0,
10))
2750
2751     for dia in self.videos_por_dia.keys():
2752         frame_dia = tk.LabelFrame(self.frame_conteos_scroll,
2753                                   text=dia.upper(),
2754                                   font=('Arial', self.base_font, 'bold'),
2755                                   bg='#34495e', fg='#ecf0f1',
2756                                   padx=5, pady=5)
2757         frame_dia.pack(fill='x', pady=3)
2758
2759         conteo = self.conteos_por_dia[dia]
2760
2761         if conteo and sum(conteo.values()) > 0:
2762             for tipo, cantidad in sorted(conteo.items(), key=lambda x: x[1],
reverse=True):
2763                 if cantidad > 0:
2764                     frame_tipo = tk.Frame(frame_dia, bg='#34495e')
2765                     frame_tipo.pack(fill='x', pady=1)
2766
2767                     tk.Label(frame_tipo, text=f"{tipo}:", bg='#34495e', fg='#bdc3c7',
2768                             font=('Arial', self.base_font - 1), width=8,
anchor='w').pack(side='left')
2769
2770                     tk.Label(frame_tipo, text=str(cantidad), bg='#34495e',
fg='#f39c12',
2771                             font=('Arial', self.base_font - 1, 'bold'), width=4,
anchor='e').pack(side='right')
2772
2773                     total_dia = sum(conteo.values())
2774                     tk.Label(frame_dia, text=f"Total: {total_dia}", bg='#34495e',
fg='#27ae60',
2775                             font=('Arial', self.base_font, 'bold')).pack()
2776                 else:
2777                     tk.Label(frame_dia, text="Sin conteos", bg='#34495e', fg='#95a5a6',
2778                             font=('Arial', self.base_font - 1)).pack()
2779
2780 def mostrar_mensaje_inicial_resultados(self):
2781     """Mensaje inicial en resultados"""
2782     for widget in self.frame_resultados_scroll.winfo_children():
2783         widget.destroy()
2784
2785     mensaje = tk.Label(self.frame_resultados_scroll,
2786                       text="1. Procese los videos de la semana\n2. Configure los
parámetros\n3. Calcule IMD + Ejes Equivalentes",

```

```

2787         bg='#34495e', fg='#95a5a6',
2788         font=('Arial', self.base_font + 2),
2789         justify='center')
2790     mensaje.pack(expand=True, pady=50)
2791
2792     def detener_procesamiento(self):
2793         """Detener procesamiento"""
2794         self.procesando = False
2795         self.finalizar_procesamiento()
2796         self.actualizar_status(" Procesamiento detenido")
2797
2798     def finalizar_procesamiento(self):
2799         """Finalizar procesamiento"""
2800         self.procesando = False
2801
2802         if hasattr(self, 'cap') and self.cap:
2803             self.cap.release()
2804
2805         self.btn_procesar.configure(state='normal' if self.modelo_cargado else
'disabled')
2806         self.btn_detener.configure(state='disabled')
2807
2808     def resetear_todo(self):
2809         """Resetear sistema"""
2810         if messagebox.askyesno("Confirmar", "¿Resetear todos los datos?"):
2811             for dia in self.videos_por_dia.keys():
2812                 self.videos_por_dia[dia] = None
2813                 self.conteos_por_dia[dia] = {}
2814                 self.estados_procesamiento[dia] = "pendiente"
2815                 self.labels_dias[dia].configure(text=" Sin video", fg='#95a5a6')
2816
2817             self.imd_calculado = {}
2818             self.ejes_equivalentes_resultado = None
2819
2820             self.actualizar_conteos_display()
2821             self.mostrar_mensaje_inicial_resultados()
2822             self.verificar_estado_botones()
2823
2824             self.actualizar_status(" Sistema reseteado")
2825
2826     # [MÉTODOS DE EXPORTACIÓN...]
2827
2828     def exportar_reporte_imd(self):
2829         """Exportar reporte IMD"""
2830         if not self.imd_calculado:
2831             messagebox.showwarning("Advertencia", "Debe calcular IMD primero")
2832             return
2833
2834         archivo = filedialog.asksaveasfilename(
2835             title="Guardar Reporte IMD",
2836             defaultextension=".txt",
2837             filetypes=[("Texto", "*.txt")],
2838             initialname=f"reporte_imd_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt"
2839         )

```

```

2840
2841     if archivo:
2842         try:
2843             with open(archivo, 'w', encoding='utf-8') as f:
2844                 f.write("="*80 + "\n")
2845                 f.write("REPORTE IMD - SISTEMA v6.0\n")
2846                 f.write("="*80 + "\n\n")
2847
2848                 f.write(f"Fecha: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
2849
2850                 # Si hay estación seleccionada
2851                 if self.ejes_equivalentes_resultado:
2852                     estacion = self.ejes_equivalentes_resultado['configuracion']
2853                     f.write(f"Estación: {estacion}\n\n")
2854
2855                 # IMD calculado
2856                 f.write(f"IMD CALCULADO:\n")
2857                 f.write("-"*40 + "\n")
2858
2859                 total_imd = 0
2860                 for tipo, imd in sorted(self.imd_calculado.items(), key=lambda x:
x[1], reverse=True):
2861                     f.write(f"{tipo}: {imd:.2f} veh/día\n")
2862                     total_imd += imd
2863
2864                 f.write(f"\nTOTAL IMD: {total_imd:.2f} veh/día\n")
2865
2866                 messagebox.showinfo("Éxito", f"Reporte guardado:\n{archivo}")
2867                 self.actualizar_status(" Reporte IMD exportado")
2868
2869             except Exception as e:
2870                 messagebox.showerror("Error", f"Error al guardar:\n{str(e)}")
2871
2872     def exportar_csv_ejes(self):
2873         """Exportar CSV ejes equivalentes"""
2874         if not self.ejes_equivalentes_resultado:
2875             messagebox.showwarning("Advertencia", "Debe calcular ejes equivalentes
primero")
2876             return
2877
2878         archivo = filedialog.asksaveasfilename(
2879             title="Guardar CSV Ejes Equivalentes",
2880             defaultextension=".csv",
2881             filetypes=[("CSV", "*.csv")],
2882             initialname=f"ejes_equivalentes_{datetime.now().strftime('%Y-%m-%d_%H%M%S')}.csv"
2883         )
2884
2885         if archivo:
2886             try:
2887                 with open(archivo, 'w', newline='', encoding='utf-8') as f:
2888                     writer = csv.writer(f)
2889

```



```

2890         # Encabezados
2891         writer.writerow([
2892             'Tipo_Vehiculo', 'IMD', 'FCE', 'IMDA', 'FD', 'FC',
2893             'FVP', 'FP', 'EE_Dia_Carril', 'FCA', 'NREP_EE', 'Clasificacion',
2894             'Estacion'
2895         ])
2896
2897         # Datos
2898         resultados = self.ejes_equivalentes_resultado["resultados_por_tipo"]
2899         estacion = self.ejes_equivalentes_resultado["configuracion"]
2900
2901         for tipo, datos in sorted(resultados.items()):
2902             writer.writerow([
2903                 tipo, f"{datos['imd']:.2f}", f"{datos['fce']:.4f}",
2904                 f"{datos['fd']:.2f}", f"{datos['fc']:.2f}", f"
2905                 {datos['fvp']:.6f}",
2906                 f"{datos['fp']:.1f}", f"{datos['ee_dia_carril']:.4f}",
2907                 f"{datos['fca']:.2f}", f"{int(datos['nrep_ee'])}",
2908                 datos['clasificacion'],
2909                 estacion
2910             ])
2911
2912             writer.writerow([
2913                 'RESUMEN'
2914             ])
2915             writer.writerow([
2916                 'Estación', estacion
2917             ])
2918             writer.writerow([
2919                 'Total NREP EE', int(self.ejes_equivalentes_re-
2920                 sultado['total_nrep_ee'])
2921             ])
2922
2923             messagebox.showinfo("Éxito", f"CSV guardado:\n{archivo}")
2924             self.actualizar_status(" CSV exportado")
2925
2926         except Exception as e:
2927             messagebox.showerror("Error", f"Error al guardar CSV:\n{str(e)}")
2928
2929     def exportar_reporte_completo(self):
2930         """Exportar reporte completo JSON"""
2931         if not self.ejes_equivalentes_resultado:
2932             messagebox.showwarning("Advertencia", "Debe calcular resultados primero")
2933             return
2934
2935         archivo = filedialog.asksaveasfilename(
2936             title="Guardar Reporte Completo",
2937             defaultextension=".json",
2938             filetypes=[("JSON", "*.json")],
2939             initialname=f"reporte_completo_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json"
2940         )
2941
2942         if archivo:
2943             try:
2944                 reporte = {
2945                     "timestamp": datetime.now().isoformat(),

```

```

2937         "version": "Sistema Ejes Equivalentes v6.0",
2938         "conteos_por_dia": self.conteos_por_dia,
2939         "videos_asignados": {dia: os.path.basename(video) if video else None
2940                               for dia, video in self.videos_por_dia.items()},
2941         "imd_calculado": self.imd_calculado,
2942         "ejes_equivalentes": self.ejes_equivalentes_resultado
2943     }
2944
2945     with open(archivo, 'w', encoding='utf-8') as f:
2946         json.dump(reporte, f, indent=2, ensure_ascii=False)
2947
2948         messagebox.showinfo("Éxito", f"Reporte completo guardado:\n{archivo}")
2949         self.actualizar_status(" Reporte completo exportado")
2950
2951     except Exception as e:
2952         messagebox.showerror("Error", f"Error al guardar reporte:\n{str(e)}")
2953
2954 def exportar_todo(self):
2955     """Exportar todos los reportes"""
2956     if not self.ejes_equivalentes_resultado:
2957         messagebox.showwarning("Advertencia", "Debe calcular resultados primero")
2958         return
2959
2960     carpeta = filedialog.askdirectory(title="Seleccionar carpeta para exportar todo")
2961     if not carpeta:
2962         return
2963
2964     try:
2965         timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
2966         archivos_creados = []
2967
2968         # Exportar IMD
2969         # Exportar CSV
2970         # Exportar JSON
2971         # [... código de exportación completo ...]
2972
2973         messagebox.showinfo("Éxito",
2974                             f"Todos los reportes exportados:\n\n"
2975                             f" Carpeta: {carpeta}")
2976         self.actualizar_status(" Exportación completa realizada")
2977
2978     except Exception as e:
2979         messagebox.showerror("Error", f"Error en exportación:\n{str(e)}")
2980
2981 def abrir_carpeta_videos_procesados(self):
2982     """Abrir carpeta de videos procesados"""
2983     if not self.carpeta_videos_procesados:
2984         messagebox.showinfo("Info", "No hay carpeta de videos configurada")
2985         return
2986
2987     try:
2988         if os.name == 'nt':
2989             os.startfile(self.carpeta_videos_procesados)
2990         else:

```



```

2991         os.system(f'xdg-open "{self.carpeta_videos_procesados}"')
2992     except Exception as e:
2993         messagebox.showerror("Error", f"No se pudo abrir carpeta:\n{str(e)}")
2994
2995     def abrir_carpeta_resultados(self):
2996         """Abrir carpeta actual"""
2997         try:
2998             carpeta = os.getcwd()
2999             if os.name == 'nt':
3000                 os.startfile(carpeta)
3001             else:
3002                 os.system(f'xdg-open "{carpeta}"')
3003         except Exception as e:
3004             messagebox.showerror("Error", f"No se pudo abrir carpeta:\n{str(e)}")
3005
3006     def actualizar_status(self, mensaje):
3007         """Actualizar barra de estado"""
3008         if "" in mensaje:
3009             color = '#27ae60'
3010         elif "" in mensaje:
3011             color = '#e74c3c'
3012         elif "" in mensaje or "🔍" in mensaje:
3013             color = '#f39c12'
3014         elif "" in mensaje:
3015             color = '#3498db'
3016         else:
3017             color = '#34495e'
3018
3019         self.status_bar.configure(text=mensaje, bg=color)
3020
3021     def ejecutar(self):
3022         """Ejecutar aplicación"""
3023         def on_closing():
3024             if self.procesando:
3025                 if messagebox.askokcancel("Salir", "¿Detener procesamiento y salir?"):
3026                     self.procesando = False
3027                     if hasattr(self, 'cap') and self.cap:
3028                         self.cap.release()
3029                     self.root.destroy()
3030             else:
3031                 self.root.destroy()
3032
3033         self.root.protocol("WM_DELETE_WINDOW", on_closing)
3034         self.root.mainloop()
3035
3036     def main():
3037         """Función principal"""
3038         try:
3039             print(" Sistema de Ejes Equivalentes v6.0")
3040             print("="*80)
3041             print(" 78 ESTACIONES DE CONTEO")
3042             print(" 24 DEPARTAMENTOS DEL PERÚ")
3043             print(" METODOLOGÍA AASHTO COMPLETA")
3044             print(" CPU CONTROLADO + EXPORTACIÓN AUTOMÁTICA")

```



```
3045     print("="*80)
3046
3047     app = InterfazSistemaEjesEquivalentes()
3048     app.ejecutar()
3049
3050 except Exception as e:
3051     print(f" Error: {e}")
3052     import traceback
3053     traceback.print_exc()
3054
3055     root_error = tk.Tk()
3056     root_error.withdraw()
3057     messagebox.showerror("Error Fatal",
3058                          f"Error al inicializar:\n\n{str(e)}")
3059     root_error.destroy()
3060
3061 if __name__ == "__main__":
3062     main()
```

A.1. Matriz de consistencia

Cuadro 7: Matriz de consistencia.

Problemas principal	Objetivos general	Hipótesis general	Variables	Metodología
¿De que manera un modelo basado en Redes Neuronales Artificiales (RNA) mejora y optimiza la precisión y eficiencia en la detección y conteo de vehículos para la determinación de parámetros críticos en el diseño de pavimentos como el Índice Medio Diario Anual (IMDA) y Número de repeticiones de Ejes Equivalentes (ESAL) según las normativas del Ministerio de Transportes y Comunicaciones (MTC)?	Desarrollar un modelo basado en Redes Neuronales Artificiales (RNA) para mejorar y optimizar la precisión y eficiencia en la detección y conteo de vehículos para la determinación de parámetros críticos en el diseño de pavimentos como el Índice Medio Diario Anual (IMDA) y Número de repeticiones de Ejes Equivalentes (ESAL) según las normativas del Ministerio de Transportes y Comunicaciones (MTC).	El desarrollo de un modelo un modelo basado en Redes Neuronales Artificiales (RNA) mejora y optimiza la precisión y eficiencia en la detección y conteo de vehículos para la determinación de parámetros críticos en el diseño de pavimentos como el Índice Medio Diario Anual (IMDA) y Número de repeticiones de Ejes Equivalentes (ESAL) según las normativas del Ministerio de Transportes y Comunicaciones (MTC).	Independiente X: Modelo basado en Redes Neuronales Artificiales (RNA) <u>Dimensiones</u> – Arquitectura del modelo – Parámetros de entrenamiento – Métodos de entrenamiento – Resultados del entrenamiento Dependiente Y: Análisis del tráfico vehicular <u>Dimensiones</u> – Cantidad de vehículos – Índices medios de tráfico – Implementación del modelo RNA	Enfoque: – Cuantitativo Diseño: – No experimental
Problemas específico	Objetivos específico	Hipótesis específico		
¿Cómo el diseño del modelo basado en Redes Neuronales Artificiales (RNA) permite ajustar una arquitectura para la detección clasificación y conteo eficiente de vehículos basado en el formato de clasificación vehicular del MTC?	Desarrollar el diseño del modelo basado en Redes Neuronales Artificiales (RNA) para ajustar una arquitectura que permita la detección clasificación y conteo eficiente de vehículos basado en el formato de clasificación vehicular del MTC.	El diseño del modelo basado en Redes Neuronales Artificiales (RNA) permite ajustar una arquitectura para la detección clasificación y conteo eficiente de vehículos basado en el formato de clasificación vehicular del MTC.		
¿En que medida el entrenamiento del modelo basado en Redes Neuronales Artificiales (RNA) contribuye a lograr un balance óptimo entre la exactitud en la detección la cobertura en el reconocimiento de vehículos?	Realizar el entrenamiento del modelo basado en Redes Neuronales Artificiales (RNA) que permita lograr un balance óptimo entre la exactitud en la detección la cobertura en el reconocimiento de vehículos.	El entrenamiento del modelo basado en Redes Neuronales Artificiales (RNA) permite lograr un balance óptimo entre la exactitud en la detección y la cobertura en el reconocimiento de vehículos.		
¿De qué manera la implementación del modelo basado en Redes Neuronales Artificiales (RNA) permite desarrollar e implementar un software para la obtención automática del IMDA y ESAL en cualquier red vial optimizando los procesos de cálculo en la planificación y diseño estructural de pavimentos?	Desarrollar e implementar un software con interfaz gráfica de usuario (GUI) para la obtención automática del IMDA y ESAL en cualquier red vial optimizando los procesos de cálculo en la planificación y diseño estructural de pavimentos.	La implementación del modelo permite desarrollar un software con interfaz gráfica de usuario (GUI) para la obtención automática del IMDA y ESAL en cualquier red vial, optimizando los procesos de cálculo en la planificación y diseño estructural de pavimentos.		

Fuente: Elaboración propia.

Cuadro 8: Operacionalización de variables.

Variable	Definición conceptual	Dimensiones	Indicadores	Unidad de medida
Independiente (X): Modelo basado en Redes Neuronales Artificiales (RNA)	Un sistema de aprendizaje automático que utiliza arquitecturas de redes neuronales para procesar datos y realizar tareas de análisis complejas, como la detección y clasificación de vehículos en un entorno de tráfico vehicular.	1. Arquitectura del modelo	-Número de capas de la RNA	Conteo (n)
			-Tipo de capas	Categorico
			-Tamaño de las capas	Conteo (n)
			-Función de activación	Categorico
		2. Parámetros de entrenamiento	-Tasa de aprendizaje	(%)
			-Número de épocas de entrenamiento	Conteo (n)
			-Momento (momentum)	(%)
			-Factor de decaimiento del aprendizaje	(%)
		3. Métodos de entrenamiento	-Algoritmo de optimización	Categorico
			-Técnicas de regularización	Categorico
			-Estrategias de aumento de datos	Categorico
			-Esquema de aprendizaje	Categorico
		4. Resultados del entrenamiento	-Precisión en la validación y entrenamiento	(%)
-Pérdida en la validación y entrenamiento	(%)			
-Tiempo total de entrenamiento	(min)			
-Curva de aprendizaje	(Gráfico)			
-Pesos del modelo	Matriz			
-Bias del modelo	Vector			
-Sobreajuste	(%)			
Dependiente (Y): Análisis del tráfico vehicular	1. Cantidad de vehículos	-Número de vehículos por tipo	(veh/tipo)	
	2. Índice medio Diario Anual (IMDA)	-IMDA,IMD	(veh/año)	
	3. Implementación del modelo RNA	-Desarrollo del algoritmo	(etapas)	
		-Compatibilidad del algoritmo	(sistemas)	

Fuente: Elaboración propia.

**UNSH****FACULTAD DE
INGENIERÍA**
DE MINAS, GEOLOGÍA Y CIVIL**ACTA DE SUSTENTACIÓN DE TESIS N° 052-2025-FIMGC****PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERA CIVIL**

En la Universidad Nacional de San Cristóbal de Huamanga de la ciudad de Ayacucho, en cumplimiento a la **RESOLUCIÓN DECANAL N° 415-2025-FIMGC-D**, a los seis días del mes de noviembre de 2025, siendo las 10:00 a.m., reunidos en el Auditorio del Colegio de Ingenieros – Ayacucho, bajo la presidencia del MSc. Ing. José Ernesto ESTRADA CÁRDENAS, y los miembros: MSc. Ing. Hemerson LIZARBE ALARCÓN, MSc. Ing. Moisés Nico BARBARÁN ORIUNDO y MSc. Ing. Edward LEÓN PALACIOS, actuando como secretario docente el Ing. Saúl Walter RETAMOZO FERNÁNDEZ, para proceder a la sustentación de tesis para optar el Título Profesional de Ingeniera Civil, del bachiller:

YUVICA QUISPE LOPEZ

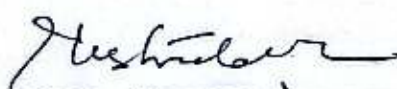
Quien presentó la tesis denominada:

Modelo basado en redes neuronales artificiales para el análisis del tráfico vehicular en la Av. Ramón Castilla del distrito de Ayacucho - Huamanga 2024

Los señores miembros del jurado, luego de expuesta la tesis y absueltas las preguntas, deliberaron y declararon:

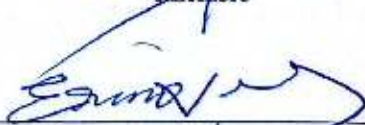
Aprobado con dieciséis (16)

Siendo las 12:00 p.m. del día 06 de noviembre de 2025, culmina el acto de sustentación de tesis, y en conformidad con lo actuado, los miembros del jurado firman al pie del presente.


MSc. José Ernesto ESTRADA CÁRDENAS
Presidente


MSc. Hemerson LIZARBE ALARCÓN
Miembro


MSc. Moisés Nico BARBARÁN ORIUNDO
Miembro


MSc. Edward LEÓN PALACIOS
Miembro - Asesor


MSc. Saúl Walter RETAMOZO FERNÁNDEZ
Secretario docente de la FIMGC

FACULTAD DE INGENIERÍA
DE MINAS Y CIVIL
Av. Independencia S/N
Ciudad Universitaria
Central Tel. 066 312510
Anexo 151



UNSCH

FACULTAD DE
INGENIERÍA
DE MINAS, GEOLOGÍA Y CIVIL



"Año de la recuperación y consolidación de la economía peruana"

CONSTANCIA DE ORIGINALIDAD DE TRABAJO DE INVESTIGACIÓN

CONSTANCIA N° 027-2025-FIMGC/ASIH

El que suscribe; responsable verificador de originalidad de trabajos de tesis de pregrado con el software Turnitin, de la Escuela Profesional de Ingeniería Civil de la Facultad de Ingeniería de Minas, Geología y Civil; en cumplimiento a la Resolución de Consejo Universitario N° 039-2021-UNSCH-CU, Reglamento de Originalidad de Trabajos de Investigación de la Universidad Nacional San Cristóbal de Huamanga y Resolución Decanal N° 697-2024-FIMGC-UNSCH-D, dejo constancia de originalidad de trabajo de investigación, que el/la Sr./Srta.

Apellidos y Nombres : Yuvica QUISPE LOPEZ
Escuela Profesional : INGENIERÍA CIVIL
Título de la Tesis : Modelo basado en redes neuronales artificiales para el análisis del tráfico vehicular en la Av. Ramón Castilla del distrito de Ayacucho-Huamanga 2024
Evaluación de la Originalidad : 14 % Índice de Similitud
Identificador de la entrega : 2846778490

Por tanto, según los Artículos 12, 13 y 17 del Reglamento de Originalidad de Trabajos de Investigación, es PROCEDENTE otorgar la Constancia de Originalidad para los fines que crea conveniente.

En señal de conformidad y verificación se firma la presente constancia

Ayacucho, 15 de diciembre del 2025



UNIVERSIDAD NACIONAL DE
SAN CRISTÓBAL DE HUAMANGA
Facultad de Ingeniería de Minas, Geología y Civil

Mg. Ing. Alex Sander IRCAÑAPA HUAMANI

Verificador de Originalidad de Trabajos de Tesis de Pregrado
Escuela de Formación Profesional de Ingeniería Civil

Con depósito para Trámites
Cc. Archivo.

FACULTAD DE INGENIERÍA DE MINAS, GEOLOGÍA Y CIVIL
Av. Independencia S/N Ciudad Universitaria
Central Tel. 066 312510
Anexo 151

Modelo basado en redes neuronales artificiales para el análisis del tráfico vehicular en la Av. Ramón Castilla del distrito de Ayacucho- Huamanga 2024

por Yuvica QUISPE LOPEZ

Fecha de entrega: 15-dic-2025 08:48a.m. (UTC-0500)

Identificador de la entrega: 2846778490

Nombre del archivo: Te_Yuvica_Quispe_Lopez.pdf (6.02M)

Total de palabras: 54256

Total de caracteres: 308929

Modelo basado en redes neuronales artificiales para el análisis del tráfico vehicular en la Av. Ramón Castilla del distrito de Ayacucho-Huamanga 2024

INFORME DE ORIGINALIDAD

14%	9%	2%	10%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	Submitted to Universidad Nacional de San Cristóbal de Huamanga	6%
	Trabajo del estudiante	
2	hdl.handle.net	3%
	Fuente de Internet	
3	burjcdigital.urjc.es	1%
	Fuente de Internet	
4	contenido.acronline.org	<1%
	Fuente de Internet	
5	Submitted to Universidad Tecnologica del Peru	<1%
	Trabajo del estudiante	
6	repositorio.ucv.edu.pe	<1%
	Fuente de Internet	
7	juandomingofarnos.wordpress.com	<1%
	Fuente de Internet	

repositorio.upse.edu.ec

8	Fuente de Internet	<1 %
9	vsip.info Fuente de Internet	<1 %
10	doaj.org Fuente de Internet	<1 %
11	repositorio.urp.edu.pe Fuente de Internet	<1 %
12	dspace.unl.edu.ec Fuente de Internet	<1 %
13	Submitted to Universidad Tecnica De Ambato- Direccion de Investigacion y Desarrollo , DIDE Trabajo del estudiante	<1 %
14	repositorio.upec.edu.ec Fuente de Internet	<1 %
15	manglar.uninorte.edu.co Fuente de Internet	<1 %
16	repositorio.upla.edu.pe Fuente de Internet	<1 %
17	apirepositorio.unh.edu.pe Fuente de Internet	<1 %
18	ruidera.uclm.es Fuente de Internet	<1 %
19	Submitted to uni Trabajo del estudiante	

<1 %

20

repositorio.upao.edu.pe

Fuente de Internet

<1 %

21

fddocuments.mx

Fuente de Internet

<1 %

22

www.investigarmqr.com

Fuente de Internet

<1 %

23

Submitted to Universidad Santiago de Cali

Trabajo del estudiante

<1 %

24

oai.e-spacio.uned.es

Fuente de Internet

<1 %

25

repositorio.ecotec.edu.ec

Fuente de Internet

<1 %

26

repositorio.ug.edu.ec

Fuente de Internet

<1 %

27

Submitted to Universidad Cesar Vallejo

Trabajo del estudiante

<1 %

28

oa.upm.es

Fuente de Internet

<1 %

Excluir citas

Activo

Excluir coincidencias

< 30 words

Excluir bibliografía

Activo