ISBN: 978-628-96613-3-0. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2026.1.1.646

# Analysis of the computational performance of machine learning models for fatigue detection using facial analysis in an embedded system

Ricardo Yauri[1]; Antero Castro[1]; Rafael Espino[1]
[1]Universidad Tecnológica del Perú, Perú, *c24068@utp.edu.pe, C19828@utp.edu.pe, C22165@utp.edu.pe*

***Abstract*– Fatigue detection using facial analysis with computer vision and machine learning in embedded systems is a non-invasive solution, but it faces challenges in evaluating the computational resources used on hardware devices like a Raspberry Pi. Previous studies have investigated fatigue and drowsiness detection using facial features related to mouth and eye opening and computer vision-based classification models, without considering the computational performance of these solutions. This research aims to evaluate the performance of machine learning models that use facial point features for fatigue detection implemented on a Raspberry Pi, considering metrics such as CPU usage, memory, inference time, and processing speed. To this end, support vector machine, random forest, and decision tree models are evaluated to assess their computational performance. The results show that no single model is superior in all aspects, as decision trees demonstrate better detection of events like yawning (YAWN. Furthermore, the decision tree stands out for its lower latency and greater adaptation to real-time applications, so the selection of the model is based on a relationship between accuracy, latency and consumption of processing resources.***

***Keywords*-- Fatigue detection, facial analysis, computer vision, Mediapipe, Raspberry Pi.***

# Análisis del desempeño computacional de modelos de aprendizaje automático para detección de fatiga mediante análisis facial en un sistema embebido

Ricardo Yauri[1]; Antero Castro[1]; Rafael Espino[1]
[1]Universidad Tecnológica del Perú, Perú, *c24068@utp.edu.pe, C19828@utp.edu.pe, C22165@utp.edu.pe*

***Resumen*– La detección de fatiga mediante análisis facial usando visión por computadora y aprendizaje automático en sistemas embebidos es una solución no invasiva pero tiene como problemática los procedimientos de evaluación sobre los recursos computacionales utilizados en dispositivos de hardware como un Raspberry Pi. En estudios previos se ha realizado investigaciones sobre la detección de fatiga y somnolencia utilizando características faciales relacionadas a la apertura de la boca y los ojos y modelos de clasificación basados en visión por computadora, sin considerar el desempeño computacional de estas soluciones. Esta investigación tiene objetivo evaluar el desempeño de modelos de aprendizaje automático que utilizan características de puntos faciales para la detección de fatiga implementados en una Raspberry Pi, considerando métricas como uso de CPU, memoria, tiempo de inferencia y velocidad de procesamiento. Para ello se evalúan modelos de máquina de vectores de soporte, bosques aleatorios y árboles de decisión para evaluar su desempeño computacional. Los resultados muestran que no existe un modelo superior en todos los aspectos, ya que los árboles de decisión presentan mejor detección de eventos como bostezo (YAWN). Además, el árbol de decisión destaca por su menor latencia y mayor adaptación con aplicaciones en tiempo real, por lo que la selección del modelo se basa en una relación entre exactitud, latencia y consumo de recursos de procesamiento.***

***Palabras clave—Detección de fatiga, Análisis facial, visión por computadora, Mediapipe, Raspberry Pi***

## I. Introducción

La detección de la fatiga usando tecnologías de análisis facial tiene un impacto en la seguridad y comportamiento de las personas en contextos como la conducción o control de equipos en entornos industriales [1] [2]. En esta investigación, el término "fatiga" se analiza considerando el concepto particular de somnolencia como uno de sus componentes observables. Es por ello la visión por computadora y el aprendizaje automático surge como una tecnología no invasiva para analizar características faciales [3][4]. Por otro lado, los sistemas embebidos de placa única, como el Raspberry Pi, ha permitido la implementación de herramientas de detección de anomalías faciales considerando que recursos computacionales reducidos [5][6].

Además, se considera que la fatiga es un concepto más amplio que incluye dimensiones físicas y cognitivas, mientras que la somnolencia se manifiesta a través de variaciones de movimiento el rostro cuando una persona esta dormida o bostezando. Para ello se usan indicadores faciales como el bostezo y el cierre prolongado de los ojos.

Uno de los problemas relacionados a la detección de fatiga mediante análisis facial es la ejecución de modelos de aprendizaje automático en hardware embebido considerando las limitaciones del hardware. En este caso, se observa que se prioriza la precisión de los modelos, pero no se toma en cuenta otros parámetros como el uso de recursos computacionales, como el uso de CPU, memoria RAM, tiempo de inferencia y velocidad de procesamiento [7] [8]. Estas consideraciones dificultan la implementación de modelos de clasificación en dispositivos de hardware embebido, lo que afecta su comportamiento en aplicaciones en tiempo real.

En investigaciones previas se ha evaluado los procedimientos para la detección de fatiga y somnolencia usando características faciales como la apertura ocular (EAR) y la razón de apertura bucal (MAR), usando técnicas de visión por computadora y modelos de clasificación [9][10]. Por otro lado, otros sistemas describen el uso de señales fisiológicas que hace más complejo el sistema en comparación con el uso de cámaras [11]. En el caso de las herramientas tecnológicas, se describe el uso de software como OpenCV, MediaPipe y modelos de aprendizaje automático como máquina de vectores de soporte, árboles de decisión y bosques aleatorios, no evaluando su rendimiento computacional cuando se despliegan en sistema embebidos [12][13] .

Esta investigación muestra el análisis del comportamiento de modelos de aprendizaje automático para la detección de fatiga usando análisis facial en un sistema embebido. Se tiene como objetivo evaluar el comportamiento de diferentes modelos de aprendizaje automático implementados en una Raspberry Pi considerando métricas de rendimiento. Como objetivos específicos, se tiene: (i) adquirir y procesar puntos característicos del rostro utilizando MediaPipe; (ii) entrenar modelos de aprendizaje automático usando estas características; (iii) implementar los modelos en una Raspberry Pi; y (iv) analizar métricas como uso de CPU, memoria, tiempo de inferencia y velocidad de procesamiento.

La motivación de la investigación se relaciona con la necesidad de evaluar modelos de aprendizaje automático y como afecta la eficiencia computacional. Por otro lado, el aporte del trabajo consiste en brindar una evaluación del desempeño de distintos modelos, generando información para

implementar soluciones de visión por computadora en un Raspberry Pi 4.

El artículo se organiza iniciando con el capítulo 1 de introducción, capítulo 2 con trabajos relacionados, capítulo 3 con la metodología propuesta, incluyendo la adquisición de datos, evaluación y despliegue de modelos, capítulo 4 de resultados y discusión y capítulo 5 de conclusiones y trabajos futuros.

## II. Trabajos Relacionados

Para la revisión de literatura, se realizó una búsqueda bibliográfica considerando documentos relacionados con detección de somnolencia y análisis de características faciales. De esta manera se pudo dar contexto al problema y sustentar la metodología usada. Se identificaron investigaciones relacionadas con visión por computadora, aprendizaje automático e integración en sistemas embebidos con hardware reducido, donde se obtienen métricas del comportamiento computacional.

En el caso de la detección de somnolencia o fatiga mediante modelos de aprendizaje automático se utilizan características faciales como la razón de apertura ocular y bucal, parpadeo y movimientos faciales [9] [14]. En algunos casos se usan modelos de clasificación como máquina de vectores de soporte, árboles de decisión o bosques aleatorios [15][16]. Además, la mayoría de estas investigaciones se enfocan en analizar la precisión del modelo, y no se analiza del consumo de recursos de hardware y su despliegue en hardware limitado.

La integración de modelos de aprendizaje automático con MediaPipe, es utilizado para extraer puntos característicos del rostro y métricas con estados de fatiga. Investigaciones previas indican este tipo de herramientas son óptimas por su capacidad de integración con Python y OpenCV [17][18]. Sin embargo, sólo se utiliza MediaPipe para preprocesamiento y no se analiza el impacto sobre el uso de recursos de hardware en sistemas embebidos.

En el caso de la implementación de modelos de aprendizaje automático en dispositivos como Raspberry Pi [19] [20] o microcontroladores para detección de gestos, se describe el mecanismo y procedimiento para ejecutar modelos simples en dispositivos de bajo consumo, pero limitados a pocos algoritmos específicos [21][22]. Por otro lado, en varios casos se usan conjuntos de datos basados con imágenes en ambientes simples, lo que varía respecto a enfoques que utilizan análisis facial mediante video en tiempo real.

Finalmente, se han analizado métricas del comportamiento de hardware embebido que integran algoritmos de visión computacional, los cuales incluyen el uso de CPU, consumo de memoria o tiempo de inferencia [23] [24]. Estas métricas se adquirieron en sistemas basados como Raspberry Pi, Jetson Nano o microcontroladores [25][26]. Sin embargo, aún existe una brecha respecto a análisis comparativo y su relación con el desempeño de modelos de aprendizaje automático para la detección de fatiga. Es por ello que esta investigación busca contribuir con la explicación metodológica del proceso de evaluación para caracterizar el comportamiento computacional de distintos modelos en un hardware con recursos limitados. Además, los estudios indican que la somnolencia corresponde a un estado fisiológico que puede ser generado por la fatiga siendo posible su identificación mediante eventos observables como cierre de ojos o bostezos.

## III. Proceso de Diseño del Sistema

La metodología de esta investigación tiene como primera etapa la adquisición de datos faciales usando Mediapipe para detección de características, seguido del almacenamiento de las métricas. Luego, los datos se utilizan para el entrenamiento y validación de modelos de aprendizaje automático SVM (máquina de vectores de soporte), RF (bosques aleatorios) y DT (árboles de decisión). Como etapa final estos modelos son desplegados en el Raspberry Pi 4 para evaluar su comportamiento (Fig. 1).

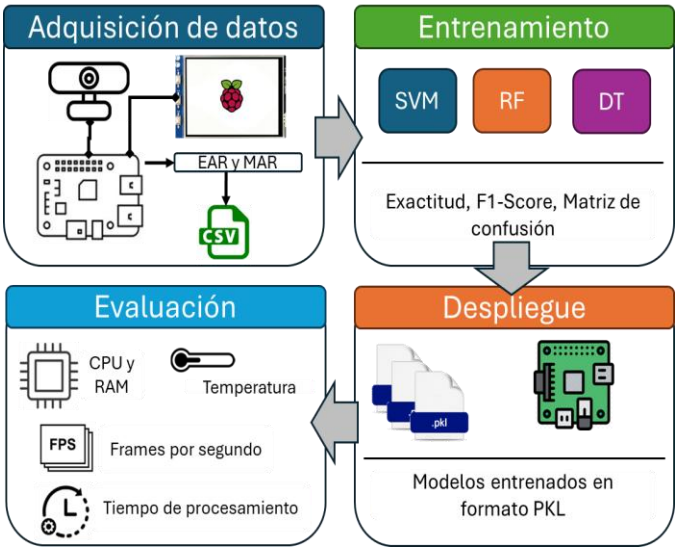



Fig. 1. Etapas del proceso de evaluación de modelos en Raspberry Pi

### *A. Etapa de adquisición de datos*

La adquisición de datos se realizó mediante un programa desarrollado en Python, desplegado en una Raspberry Pi 4, el cual integra las bibliotecas MediaPipe y OpenCV para el procesamiento de video. A partir de las imágenes capturadas por una cámara USB, el sistema detecta el rostro del usuario y extrae puntos característicos de los ojos y la boca, y se calculan los puntos de interés faciales, tales como la razón de apertura del ojo izquierdo, ojo derecho y la boca (Tabla I). Estas métricas, junto con la marca temporal y la etiqueta del evento, se almacenan en un archivo CSV considerando estos datos: timestamp, eyeLeft_ratio, eyeRight_ratio, mouth_ratio, label. Esta adquisición se realizó considerando eventos de bostezo (YAWN), dormido (SLEEP) y estado normal, para el

entrenamiento mediante un etiquetado manual usando comandos por teclado (n para normal, y para Yawn y S para sleep) (Fig. 2).

La adquisición de datos se realizó en un entorno de laboratorio, con fines de validación técnica para la validación del sistema. Las pruebas se realizaron sin asociar los datos a la identidad de las personas, usando sólo información derivada de características faciales.

TABLA I
CAMPOS DE DATOS ADQUIRIDOS

| Variable | Descripción | Tipo |
|---|---|---|
| timestamp | Marca temporal de la muestra | DateTime |
| eyeLeft_ratio | Razón de apertura del ojo izquierdo | Numérico |
| eyeRight_ratio | Razón de apertura del ojo derecho | Numérico |
| mouth_ratio | Razón de apertura bucal | Numérico |
| label | Estado detectado (NORMAL, YAWN, SLEEP) | Categórico |

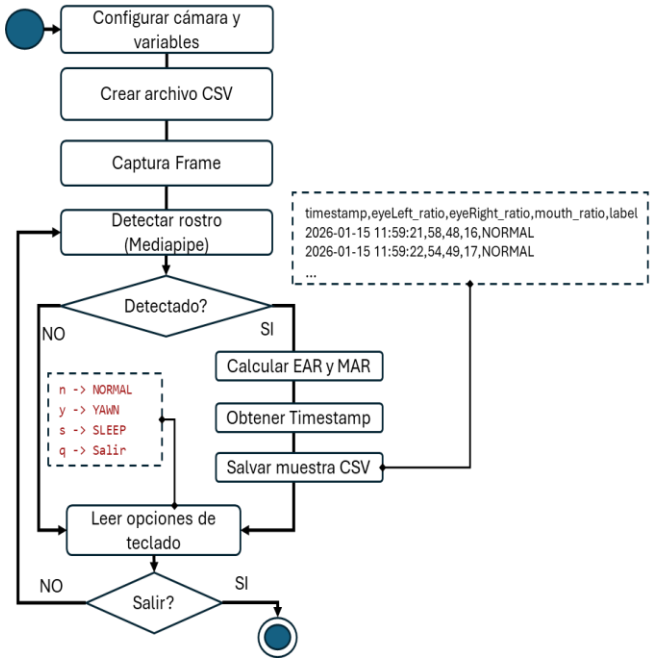


Fig. 2. Diagrama de flujo del proceso de adquisición

## B. Proceso de entrenamiento de modelos

Posteriormente se realizó el proceso de entrenamiento utilizando la biblioteca scikit-learn. En primer lugar se realizó un análisis exploratorio (EDA) para identificar patrones y valores atípicos. Además se implementó un pipeline de entrenamiento mediante validación cruzada y ajuste de hiperparámetros usando la función GridSearchCV de python. Después se evaluó el comportamiento cada modelo en base a la precisión, F1-score y matriz de confusión, comparando los resultados de clasificación y determinar cuales tienen mejor comportamiento durante el entrenamiento (Fig. 3).

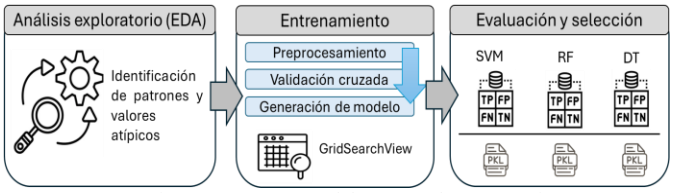


Fig. 3. Componentes del proceso de entrenamiento

Como resultado del proceso de entrenamiento se obtuvieron los hiperparámetros mediante la técnica de búsqueda GridSearchCV. En el caso del modelo SVM, el mejor comportamiento se alcanzó utilizando un kernel lineal con un valor de regularización C=10, para la separación lineal de las clases. Para el modelo Random Forest, la configuración consiste de 50 árboles y una profundidad de 10. Por otro lado, el Árbol de Decisión tuve el mejor comportamiento con un mínimo de dos muestras para la división de nodos (Tabla II).

Los resultados del proceso de entrenamiento muestran el comportamiento de los tres modelos obteniendo valores de exactitud (accuracy) superiores al 93 % para la clasificación de los eventos. Además, el modelo RF presenta el mejor comportamiento con la exactitud de 94.82 % y valores más altos de F1-score (Tabla III).

TABLA II
HIPERPARÁMETROS ÓPTIMOS DE LOS MODELOS ENTRENADOS

| Modelo de aprendizaje automático | Hiperparámetros óptimos obtenidos |
|---|---|
| SVM (Support Vector Machine) | C = 10, kernel = linear, gamma = scale |
| RF (Random Forest) | max_depth = 10, n_estimators = 50 |
| DT (Decision Tree) | max_depth = None, min_samples_split = 2 |

TABLA III
MÉTRICAS DEL PROCESO DE ENTRENAMIENTO DE MODELOS

| Model | Accuracy | Precision_macro | Recall_macro | F1_macro |
|---|---|---|---|---|
| SVM | 93.625498 | 93.771218 | 93.712184 | 93.699434 |
| DT | 94.223108 | 94.334365 | 94.297635 | 94.289201 |
| RF | 94.820717 | 94.913427 | 94.883086 | 94.877666 |

## C. Despliegue de los modelos en hardware embebido

Los modelos entrenados fueron almacenados en archivos del tipo pkl y luego son desplegados en la Raspberry Pi 4 para la clasificación de eventos de bostezo o dormido. Para la evaluación en hardware, cada modelo fue probado con la generación de 10 eventos de bostezo, 10 eventos de cierre de ojos y el resto con eventos normales. Además, la clase NORMAL es parte del proceso de clasificación multiclase, para lo cual se ha considerado evitar sesgos y tener una mejor referencia para interpretar los resultados obtenidos en el Raspberry Pi 4 (Fig. 4). El tamaño reducido de los datos de evaluación, se usan para un enfoque de validación inicial en entorno controlado, para obtener resultados representativos. En este contexto, el uso de un número limitado de eventos por clase es consistente con una etapa inicial de desarrollo.

El programa en Python implementó la etapa de inferencia usando los modelos entrenados cargados desde archivos .pkl. ml_predict() que utiliza métricas faciales de EAR izquierdo, EAR derecho y MAR, para ejecutar la inferencia obteniendo el tiempo de predicción. Además la función detect_drowsiness() integra la salida del modelo con una lógica basada en contadores de frames consecutivos, para reducir falsos positivos (Fig. 5). El sistema propuesto detecta el evento de la fatiga de manera indirecta mediante eventos faciales asociados a la somnolencia, considerando el bostezo (YAWN) y el cierre de los ojos etiquetado con el identificador "SLEEP".

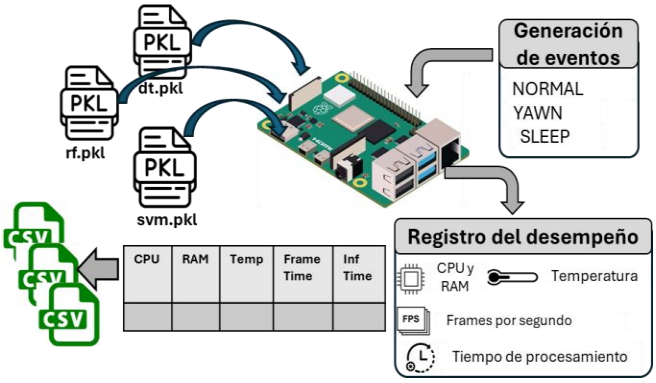



Fig. 4. Etapas de despliegue de modelos en el Raspberry Pi 4

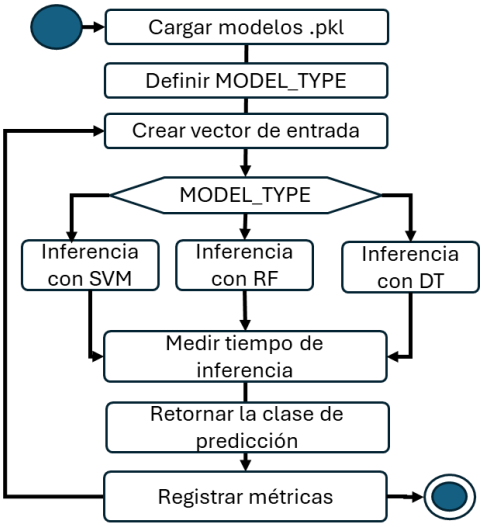



Fig. 5. Diagrama de flujo del programa de inferencia

## IV. Resultados y Discusión

Como parte del proceso de validación, los datos utilizados se obtuvieron en un entorno controlado para pruebas de funcionalidad técnica del sistema, y no para realizar la identificación de personas, para evitar el uso de datos personales sensibles biométricos. Las métricas obtenidas durante el despliegue de los modelos fueron utilizadas para realizar comparaciones del desempeño del hardware considerando indicadores de uso promedio de CPU, consumo de memoria, tiempo de inferencia, tiempo por fotograma y temperatura durante el proceso de clasificación de los modelos.

En la Fig. 6 se observa la comparación entre el número de eventos reales (línea roja) y los eventos detectados por cada modelo para las clases SLEEP y YAWN. El modelo RF logra una detección completa de los eventos de YAWN, mientras que SVM y DT presentan valores menores. El mejor desempeño del RF se basa en su naturaleza de combinar varios árboles de decisión. La detección de fatiga se realiza de manera indirecta mediante la detección de somnolencia pero no se consideran todos los aspectos no visibles. Además, para la clasificación multiclase se evita introducir sesgos en los resultados para evaluar las tres clases mediante las características faciales.

Las clases SLEEP y YAWN se basa en características faciales como elementos que, visualmente, pueden ser efectos de la somnolencia, lo que implica que cada evento activa condiciones dentro de las etapas de procesamiento. Esta diferenciación responde a operaciones de cálculo del sistema y no a una diferencia en la complejidad algorítmica de los modelos.

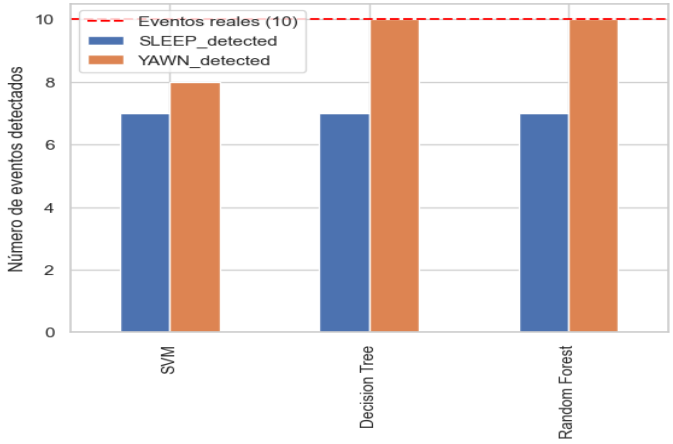



Fig. 6. Eventos detectados vs eventos reales

Los resultados presentados en la Tabla IV muestran una diferencia en términos de recall y F1-score, especialmente para la clase YAWN, donde el Árbol de Decisión (DT) y el Random Forest (RF) tienen un resultado del 100 %, superando al SVM. Esto indica que los modelos basados en DT son más eficientes para capturar patrones, mientras que el SVM presenta mayor dificultad para generalizar los cambios por los datos adquiridos para cada clase. Estos resultados muestran que los modelos basados en árboles tienen ventaja en la detección de eventos más evidentes como YAWN, mientras que la detección de SLEEP continúa siendo el más complejo

Sobre el comportamiento del uso de recursos de hardware en función del modelo desplegado en la Raspberry Pi, el DT tienen el tiempo promedio de inferencia más bajo (0.802 ms),

lo que se confirma con el valor más alto de FPS (8.09) y en un tiempo por frame inferior al de los otros modelos. Por otro lado el RF, tiene tiempo de inferencia (16.01 ms), lo que afecta el tiempo promedio por frame y en la reducción del FPS (7.45), debido a la necesidad de evaluar múltiples árboles por predicción. El SVM se ubica en un punto intermedio, con tiempos de inferencia bajos (1.69 ms), lo que indica una más carga por operaciones matemáticas continuas. Además, el uso de CPU y memoria RAM se mantiene sin cambios drásticos, indicando que el procesamiento de video es más crítico en el consumo de recursos del sistema (Tabla V).

TABLA IV
MÉTRICAS DE MODELOS DESPLEGADOS

| | Recall SLEEP (%) | Recall YAWN (%) | F1-score SLEEP (%) | F1-score YAWN (%) |
|---|---|---|---|---|
| SVM | 70.0 | 80.0 | 70.0 | 80.0 |
| DT | 70.0 | 100.0 | 70.0 | 100.0 |
| RF | 70.0 | 100.0 | 70.0 | 100.0 |

TABLA V
MÉTRICAS PROMEDIO DEL SISTEMA

| | Avg Frame Time (ms) | Avg Inference Time (ms) | Average FPS | Avg CPU (%) | Avg RAM (%) | Avg Temp (°C) |
|---|---|---|---|---|---|---|
| SVM | 124.83 | 1.69 | 8.01 | 53.83 | 13.921 | 62.785 |
| DT | 123.52 | 0.802 | 8.09 | 54.212 | 13.92 | 65.759 |
| RF | 134.106 | 16.01 | 7.45 | 52.757 | 14.037 | 64.441 |

La Fig. 7 muestra el tiempo promedio de procesamiento por Frame considerando el tipo de modelo (DT, RF y SVM) y clase detectada (NORMAL, SLEEP y YAWN), observando que se obtienen menores tiempos en la clase NORMAL, debido a que este estado no activa otros procesos o funciones de alertas o registro de eventos. Por otro lado, los estados SLEEP y YAWN aumentan el tiempo de procesamiento, por las funciones de inferencia y validación de frames continuos.

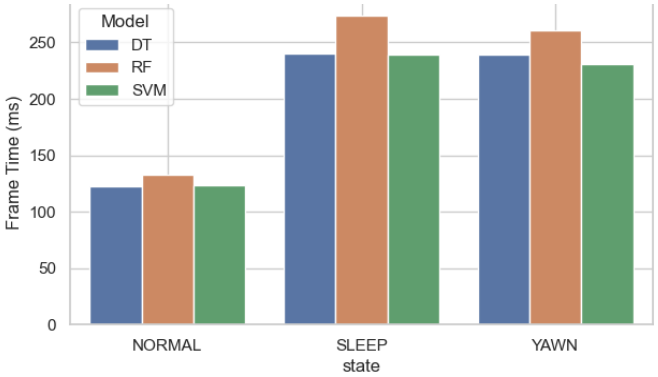



Fig. 7. Tiempo promedio de procesamiento de frame por modelo y clase

En el caso de RF, este tiene los mayores tiempos de procesamiento, debido a la evaluación de varios árboles de decisión aumentando el costo computacional cuando el evento es detectado. El SVM tiene los menores tiempos en SLEEP y YAWN, lo que indica que su inferencia es matemáticamente más eficiente que RF.

El modelo DT tiene la mayor parte de sus inferencias en valores bajos, muy sesgada con tiempos mínimos, lo que confirma su baja complejidad computacional. Por otro lado, el SVM muestra tiempo de inferencia mayores con dispersión intermedia. Estos resultados confirman que es necesario considerar un compromiso entre desempeño y costo computacional, donde RF, a pesar de clasificar mejor en ciertos casos (Tabla 3), agrega más carga temporal, siendo DT el modelo más eficiente desde el punto de vista de latencia de inferencia (Fig. 8).

La Fig. 9 muestra el tiempo promedio de inferencia por modelo y por clase observando que el RF presenta los mayores tiempos de inferencia en todas las clases, mientras que el que el DT mantiene tiempos menores. Asimismo, se observa un ligero incremento del tiempo de inferencia en las clases SLEEP y YAWN respecto a NORMAL, con los modelos RF y SVM. Estos indican que, aunque la inferencia no depende directamente de la clase a nivel algorítmico, la distribución de los datos por clase influye en el tiempo de procesamiento, confirmando que DT tiene la mejor relación entre velocidad y tiempo, mientras que RF tienen mejor comportamiento predictivo sacrificando la latencia.

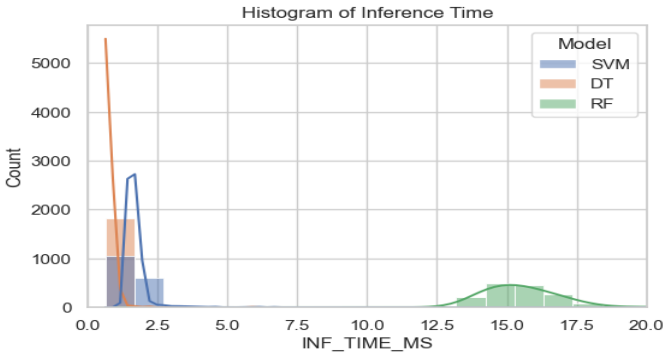



Fig. 8. Histograma del tiempo de inferencia por modelo

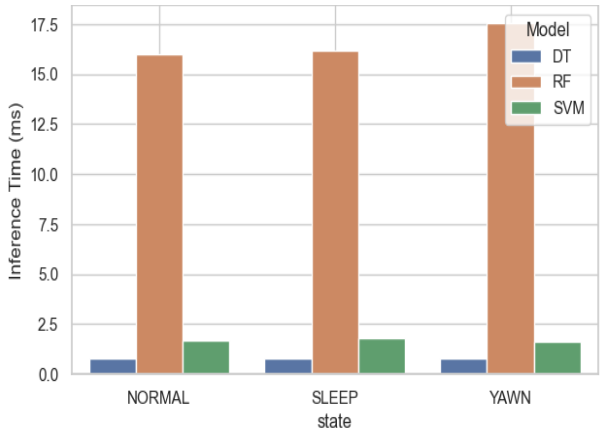
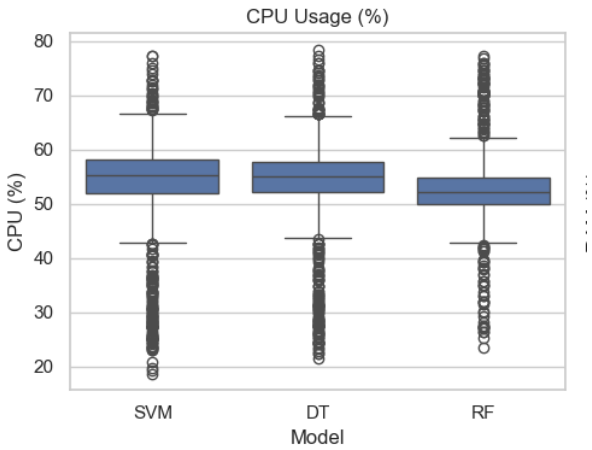


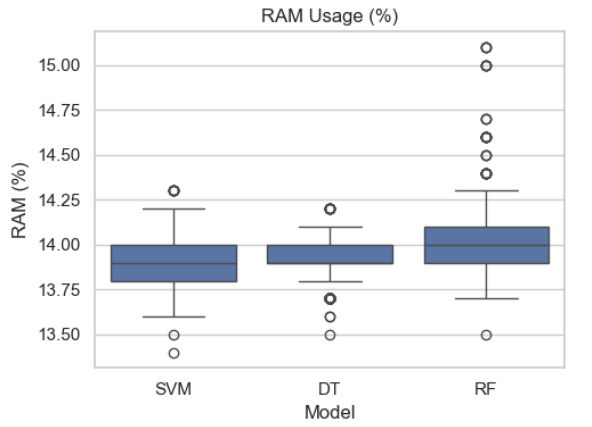

Fig. 9. Tiempo promedio de inferencia por modelo y clase

Las variaciones en el tiempo de inferencia no están asociadas a la complejidad de las clases, sino a las etapas del preprocesamiento y a la validación temporal de eventos. Además, etapas adicionales introducen diferencias en el tiempo total de ejecución. En el caso de la dispersión y distribución de datos de uso de CPU, en la Fig. 10 (a) se observa que los modelos SVM y DT tienen valores superiores y mayor dispersión, junto con más valores atípicos. Esta variabilidad puede ser causada debido a que para tareas continuas. como captura de video o cálculo de características es necesario el uso de más recursos. Por otro lado, RF tiene valores menores, lo que sugiere que, aunque los procesos de clasificación son más complejos, tienen una carga de CPU más estable.

En el caso del uso de memoria RAM, hay menor variabilidad en comparación con la CPU, lo que indica que la inferencia de los tres algoritmos mantiene un consumo de memoria constante en la Raspberry Pi. Sin embargo, RF tiene una pequeña tendencia a valores más altos y más outliers. Se infiere que la CPU es el recurso más crítico, mientras que la RAM permanece más constante.




(a)




(b)

Fig. 10. Distribución de datos para (a) uso de CPU y (b) memoria RAM

Los resultados de la Fig. 11 muestran que el consumo de CPU es mayor durante el estado NORMAL (alcanzando valores cercanos al 53-54%) en comparación con los estados de YAWN y SLEEP, lo que se mantiene constante para los tres modelos de clasificación. Este comportamiento se debe a que la mayor carga computacional no se relaciona sólo con la ejecución del modelo de clasificación, sino en la etapa de preprocesamiento de puntos de referencia faciales. Por otro lado, el uso de CPU se reduce durante la detección de bostezos o sueño sugiere que, una vez identificado el evento, el programa entra en rutinas de almacenamiento de información más ligeras, que reducen el uso del procesador.

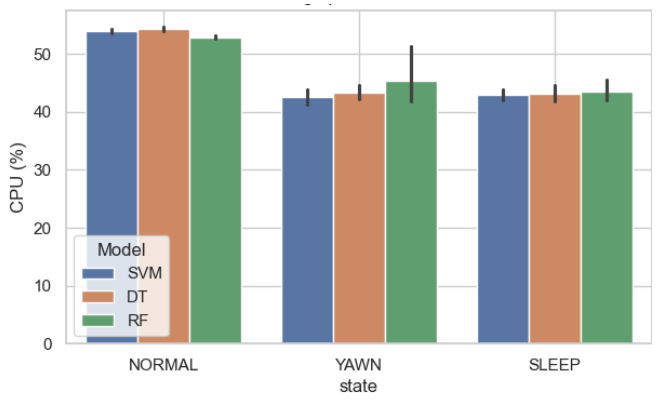



Fig. 11. Uso de CPU por modelo y clase

Los resultados obtenidos utilizan un conjunto de datos reducido, y se orienta sólo a evaluar de manera inicial la capacidad del modelo para diferenciar patrones faciales. Además los resultados son válidos para pruebas de concepto, demostrando la viabilidad del procedimiento de investigación. Es por ello que los modelos funcionan de manera uniforme para todas las clases.

## V. Conclusiones

Se consiguió adquirir y procesar adecuadamente puntos característicos del rostro mediante MediaPipe y usar modelos de aprendizaje automático en una Raspberry Pi para su evaluación. Además, se realizó la comparación del desempeño de cada modelo considerando métricas como uso de CPU, memoria, tiempo de inferencia y velocidad de procesamiento, usadas para demostrar el impacto de la eficiencia computacional en sistemas embebidos con hardware reducido.

A partir de los resultados se observa que no hay un modelo superior en todos los aspectos, sino que cada uno tiene ventajas que deben son considerados según los requisitos de la aplicación. En relación al comportamiento de clasificación, los modelos basados en árboles (DT y RF) demostraron una mayor capacidad para detectar eventos como YAWN, mientras que el SVM mostró una menor capacidad para esta clase. A pesar de ello, la detección de la clase SLEEP se mantiene como el caso más complejo para todos los modelos,.

Por otro lado, considerando el consumo de recursos computacionales, el Árbol de Decisión (DT) es más eficiente,

por el menor tiempo promedio de inferencia, mayor FPS y menor latencia por frame, siendo adecuado para aplicaciones en tiempo real.

El sistema fue capaz de detectar eventos faciales relacionados a la somnolencia como componente de la fatiga, siendo una opción viable para monitoreo. Es por ello que la selección del modelo se debe basar en una relación entre exactitud, latencia y uso de recursos de CPU y RAM, donde el modelo RF es adecuado cuando se requiere clasificación más precisa de eventos, DT cuando se necesita menor latencia y SVM para un equilibrio entre velocidad de clasificación y modelos sencillos.

## AGRADECIMIENTO/RECONOCIMIENTO

Un agradecimiento a la Dirección de Investigación de la Universidad Tecnológica del Perú (UTP-Lima Sur), por el apoyo en el proyecto de Investigación financiado 2025, lo cual ha permitido el logro del presente estudio.

## REFERENCIAS

[1] I. Feliciani Merizio, F. R. Chavarette, and E. Fuzaro de Almeida, "Machine learning techniques for fault detection in rotating mechanical systems," *Eng. Comput.*, vol. 42, no. 3, pp. 1316–1334, Apr. 2025, doi: 10.1108/EC-09-2024-0896.

[2] C. Baccouch and C. Bahar, "Embedded System for ECG Signal Monitoring and Fatigue Detection in Elderly Individuals Using Machine Learning Models," *Int. J. Adv. Comput. Sci. Appl.*, vol. 16, no. 9, pp. 53–62, Sep. 2025, doi: 10.14569/IJACSA.2025.0160907.

[3] S. A. Haider, S. Prabha, C. A. Gomez-Cabello, S. Borna, A. Genovese, M. Trabilsy, A. Elegbede, J. F. Yang, A. Galvao, C. Tao, and A. J. Forte, "Facial Analysis for Plastic Surgery in the Era of Artificial Intelligence: A Comparative Evaluation of Multimodal Large Language Models," *J. Clin. Med.*, vol. 14, no. 10, p. 3484, May 2025, doi: 10.3390/jcm14103484.

[4] J. S. Kim, H. Ko, H. Woo, and W. S. Kim, "Lessons Learned from the Point-of-Care Use of a Facial Analysis Technology," *Ann. Child Neurol.*, vol. 31, no. 4, pp. 271–275, Oct. 2023, doi: 10.26815/acn.2023.00227.

[5] H. Giedra, T. Sledevič, and D. Matuzevičius, "Deploying Optimized Deep Vision Models for Eyeglasses Detection on Low-Power Platforms," *Electronics*, vol. 14, no. 14, p. 2796, Jul. 2025, doi: 10.3390/electronics14142796.

[6] S. Liawatimena and N. Isworo, "Annotated drowsiness detection dataset captured using Raspberry Pi 5," *Data Br.*, vol. 63, p. 112211, Dec. 2025, doi: 10.1016/j.dib.2025.112211.

[7] M. Ouloul, Z. Moutakki, A. Amghar, and K. Afdel, "Low-cost embedded facial recognition system based on overlapped local binary pattern," *e-Prime - Adv. Electr. Eng. Electron. Energy*, vol. 11, no. 1, p. 100924, Mar. 2025, doi: 10.1016/j.prime.2025.100924.

[8] B. Setiawan, I. Siradjuddin, A. D. W. Sumari, W. Widjanarko, E. Mandyatma, and D. F. Putradi, "Identification of CNN hyper-parameters for tobacco leaf quality classification on Nvidia Jetson Nano," *Eastern-European J. Enterp. Technol.*, vol. 6, no. 2 (126), pp. 17–24, Dec. 2023, doi: 10.15587/1729-4061.2023.289017.

[9] Y. Albadawi, A. AlRedhaei, and M. Takruri, "Real-Time Machine Learning-Based Driver Drowsiness Detection Using Visual Features," *J. Imaging*, vol. 9, no. 5, p. 91, Apr. 2023, doi: 10.3390/jimaging9050091.

[10] A. Amirbay, N. Baigabylov, A. Mukhanova, K. Mukhambetova, E. Zaitov, R. Burganova, K. Khusanova, and F. Akhmedova, "Autism detection using facial and motor analysis using machine learning," *Bull. Electr. Eng. Informatics*, vol. 14, no. 5, pp. 3985–4000, Oct. 2025, doi: 10.11591/eei.v14i5.10319.

[11] N. Dachoponchai, Y. Wongsawat, and J. Arnin, "Predictive Analysis of Driver Drowsiness Progression: Multi-Level Drowsiness Classification Using Physiological Signals," in *2024 Asia Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, IEEE, Dec. 2024, pp. 1–6. doi: 10.1109/APSIPAASC63619.2025.10848692.

[12] S. Vidyanathan Dixith, S. Jadhav, Y. Kim, and N. Jayakumar, "Deep Learning-Based Drowsiness Detection System for Driver's Safety," *IEEE Access*, vol. 13, no. 1, pp. 154080–154102, 2025, doi: 10.1109/ACCESS.2025.3604728.

[13] J. R. Theivadas and S. Ponnan, "VigilEye: Machine learning-powered driver fatigue recognition for safer roads," *Meas. Sensors*, vol. 33, no. 1, p. 101186, Jun. 2024, doi: 10.1016/j.measen.2024.101186.

[14] C. Xu, W. Huang, J. Liu, and L. Li, "Detecting Driver Drowsiness Using Hybrid Facial Features and Ensemble Learning," *Information*, vol. 16, no. 4, p. 294, Apr. 2025, doi: 10.3390/info16040294.

[15] Z. Ji, X. Xie, E. Jiang, Y. Wang, B. Min, S. Yang, Y. Chen, and D. Pons, "Integrating DRN-RF with computer vision for detection of control room operator's mental fatigue," *PLoS One*, vol. 20, no. 4, p. e0320780, Apr. 2025, doi: 10.1371/journal.pone.0320780.

[16] Z. Wang, X. Guan, D. Li, C. Jiang, Y. Bai, D. Yang, and L. He, "Study on Muscle Fatigue Classification for Manual Lifting by Fusing sEMG and MMG Signals," *Sensors*, vol. 25, no. 16, p. 5023, Aug. 2025, doi: 10.3390/s25165023.

[17] M. Kim and G. Choi, "STFTransNet: A Transformer Based Spatial Temporal Fusion Network for Enhanced Multimodal Driver Inattention State Recognition System," *Sensors*, vol. 25, no. 18, p. 5819, Sep. 2025, doi: 10.3390/s25185819.

[18] A. Zaman, P. Chatterjee, and R. Sharma, "Real-Time Drivers' Drowsiness Detection: A Deep Learning Approach," in *2024 2nd International Conference on Artificial Intelligence, Blockchain, and Internet of Things (AIBThings)*, IEEE, Sep. 2024, pp. 1–5. doi: 10.1109/AIBThings63359.2024.10861885.

[19] S. Suhana, B. Prathap Rudra, K. Narang, and I. Anto, "IoT Based Emergency Vehicle Detection using YOLOv8," *J. Autom. Mob. Robot. Intell. Syst.*, vol. 19, no. 2, pp. 79–88, Jun. 2025, doi: 10.14313/jamris-2025-018.

[20] K. N. V. Satyanarayana, R. G. Prasanna, R. R. K. S. Satwik, P. Rupchand, P. Srihaas, and S. Bhuvanapriya, "Smart Traffic Management for Emergency Vehicles Using YOLOv8 Algorithm," in *Multidisciplinary Approaches to AI, Data, and Innovation for a Smarter World*, IGI Global, 2025, pp. 491–510. doi: 10.4018/979-8-3693-9375-8.ch029.

[21] F. Khan, A. Khan, T. Ali, T. Shahzad, T. Mazhar, S. Khan, M. A. Khan, and H. Hamam, "IoT-Driven Pollution Detection System for Indoor and Outdoor Environments," *Comput. Mater. Contin.*, vol. 86, no. 2, pp. 1–27, 2026, doi: 10.32604/cmc.2025.068228.

[22] M. I. Al Tameemi, S. T. Hasson, and H. Al-Libawy, "Real-Time Road Traffic Condition Detection System Implemented on Limited Resources Microcontroller Using Lite Deep Learning Techniques," *Int. J. Intell. Eng. Syst.*, vol. 18, no. 7, pp. 592–608, Aug. 2025, doi: 10.22266/ijies2025.0831.38.

[23] A. K. Pathak, A. K. Singh, P. Kumar, V. Bhatia, and O. Krejcar, "Real-time anti-sleep alert algorithm to prevent road accidents to ensure road safety," *Front. Futur. Transp.*, vol. 6, no. 1, p. 10, Mar. 2025, doi: 10.3389/ffutr.2025.1545411.

[24] D. A. N. Gookyi, F. A. Wulnye, E. A. E. Arthur, R. K. Ahiadormey, J. O. Agyemang, K. O.-B. O. Agyekum, and R. Gyaang, "TinyML for smart agriculture: Comparative analysis of TinyML platforms and practical deployment for maize leaf disease identification," *Smart Agric. Technol.*, vol. 8, no. 1, p. 100490, Aug. 2024, doi: 10.1016/j.atech.2024.100490.

[25] F. H. Kamaru Zaman, K. M. Ng, and S. A. Che Abdullah, "Comparative Analysis of Vision Transformers and CNN Models for Driver Fatigue Classification," *IIUM Eng. J.*, vol. 26, no. 2, pp. 169–186, May 2025, doi: 10.31436/iiumej.v26i2.3488.

[26] N. Aishwarya, G. S. Yaythish Kannaa, and K. Seemakurthy, "YOLOSkin: A fusion framework for improved skin cancer diagnosis using YOLO detectors on Nvidia Jetson Nano," *Biomed. Signal Process. Control*, vol. 100, no. 1, p. 107093, Feb. 2025, doi: 10.1016/j.bspc.2024.107093.