Práctica Profesional Supervisada

Herlan, Nicolas Federico

Arquitectura de IA especializada en el análisis conductual mediante fusión de características : Prueba de concepto basada en la detección de microsonrisas en imágenes

Instituto de Ingeniería y Agronomía

Carrera: Ingeniería en Informática

2025

INSTITUTO DE INGENIERÍA Y AGRONOMÍA

INGENIERÍA EN INFORMÁTICA

PROYECTO INTEGRADOR PROFESIONALIZANTE

Informe Final

Arquitectura de IA especializada en el análisis conductual mediante fusión de características: Prueba de concepto basada en la detección de microsonrisas en imágenes

Estudiante: Nicolas Federico Herlan

Entidad Receptora: Universidad Nacional Arturo Jauretche

Docente Supervisor: Carlos Schenone

Tutor Organizacional: Martín Morales

Florencio Varela, Diciembre 2025

Resumen

El presente trabajo aborda el diseño y validación de una prueba de concepto orientada a la detección automática de microsonrisas en imágenes, empleando técnicas de Inteligencia Artificial y aprendizaje automático. Las microexpresiones faciales, por su corta duración y carácter involuntario, representan uno de los desafíos más complejos en el campo del análisis de comportamiento no verbal. El proyecto propone un enfoque híbrido basado en la extracción de características geométricas faciales mediante landmarks y la construcción de un conjunto de datos propio especialmente etiquetado. A partir de este material, se desarrolla un modelo de clasificación supervisada capaz de distinguir entre microsonrisas y expresiones neutrales, evaluando su desempeño a través de métricas estandarizadas. Los resultados obtenidos muestran que es posible alcanzar una capacidad de discriminación adecuada aun en condiciones de recursos computacionales limitados, lo que demuestra la viabilidad técnica de la propuesta. Finalmente, el informe presenta la arquitectura diseñada, el proceso de entrenamiento, la validación experimental y las posibles líneas de mejora para trabajos futuros.

Palabras Clave: Detección de microsonrisas, Arquitectura de inteligencia artificial, Fusión de características, Análisis conductual, Landmarks faciales.

Abstract

AI architecture specialized in behavioral analysis through feature fusion: Proof of concept based on the detection of microsmiles in images.

This work presents the design and validation of a proof of concept aimed at the automatic detection of micro–smiles in images using Artificial Intelligence and machine learning techniques. Facial micro–expressions, due to their short duration and involuntary nature, constitute one of the most challenging problems in non–verbal behavior analysis. The project proposes a hybrid approach based on the extraction of geometric facial features through landmarks and the construction of a custom labeled dataset. Using this dataset, a supervised classification model is developed to distinguish micro–smiles from neutral expressions, and its performance is evaluated through standardized metrics. The results demonstrate that an effective level of discrimination can be achieved even under limited computational resources, proving the technical feasibility of the proposal. The report details the system architecture, the training process, the experimental validation, and future improvements.

Keywords: Microsmile Detection, Artificial Intelligence Architecture, Feature Fusion, Behavioral Analysis, Facial Landmarks.

Dedicatoria y agradecimientos

Este proyecto está dedicado a todas las personas que han sido una fuente de inspiración y apoyo. A aquellos que, con su dedicación, esfuerzo y amor por la materia, han hecho posible que este trabajo se materialice.

A mis seres queridos, cuya paciencia y comprensión han sido el aliento necesario para seguir en cada etapa de mi carrera. Gracias por estar siempre ahí.

Quiero expresar mi más sincero agradecimiento a todas las personas que han hecho posible la realización de este proyecto.

En primer lugar, a mi profesor tutor Carlos Schenone, por su orientación invaluable, por compartir su conocimiento y por su apoyo en cada fase del trabajo. Su capacidad para motivar y guiarme ha sido fundamental para el desarrollo de este proyecto.

Agradecer a la Universidad Nacional Arturo Jauretche por brindarme una educación excepcional, a todos los profesores que tuve a lo largo de mi recorrido académico y por supuesto a mis compañeros por brindarme siempre un espacio para compartir ideas, dudas. El apoyo ha sido clave para seguir avanzando.

Índice de contenido

1. Introducción

En los últimos años, el crecimiento acelerado de los sistemas de visión por computadora y del aprendizaje profundo ha generado avances significativos en la automatización de tareas complejas vinculadas al reconocimiento facial, la detección de emociones, el análisis conductual y los sistemas biométricos inteligentes. La disponibilidad de modelos preentrenados, arquitecturas optimizadas y bibliotecas de código abierto ha permitido que numerosos procesos tradicionalmente manuales puedan ser asistidos o reemplazados por herramientas basadas en inteligencia artificial (IA). Sin embargo, la aplicación práctica de estas tecnologías sigue enfrentando desafíos derivados de la variabilidad de los datos, las limitaciones computacionales y la demanda de sistemas más explicables y eficientes [1] [2].

En este marco, los sistemas de detección y análisis de microexpresiones faciales han cobrado interés en la comunidad científica debido a su potencial para identificar respuestas emocionales sutiles, rápidas e involuntarias asociadas con estados internos de la persona. Estas microexpresiones son relevantes en áreas como la interacción humano–máquina, la seguridad, el análisis conductual, la salud mental y la investigación sociocognitiva [3] [4]. No obstante, su detección automática continúa siendo un problema técnico desafiante, dado que suelen durar menos de 500 milisegundos, presentan baja intensidad muscular y requieren modelos capaces de identificar cambios geométricos mínimos en la configuración facial.

El presente proyecto surge como una respuesta a esta problemática, proponiendo el desarrollo de un sistema que permita detectar y analizar expresiones y microexpresiones faciales a partir de video, empleando arquitecturas livianas como MediaPipe Face Mesh, modelos de reconocimiento emocional como DeepFace, y técnicas de aprendizaje automático tradicional para la clasificación de emociones. Este enfoque integra eficiencia computacional, capacidad de procesamiento en tiempo real y una estructura explicable basada en distancias y variaciones geométricas entre puntos clave del rostro.

1.2. Planteamiento del problema técnico

La detección automática de microexpresiones se enfrenta a múltiples limitaciones técnicas que motivan la presente investigación. En primer lugar, los sistemas convencionales de reconocimiento emocional presentan dificultades para capturar variaciones de baja intensidad o transiciones muy rápidas entre estados emocionales. Los modelos de deep learning de gran escala logran buenos resultados en tareas complejas, pero requieren recursos computacionales elevados y grandes volúmenes de datos, lo que limita su implementación práctica en entornos con hardware convencional o de gama media [5].

A nivel metodológico, el análisis de microexpresiones requiere modelos sensibles a pequeñas variaciones geométricas en los landmarks faciales. Sin embargo, la mayoría de los datasets existentes presentan ejemplos limitados, en condiciones controladas y con una representación insuficiente de la variabilidad real en iluminación, postura, edad, rasgos faciales y entorno [6]. Esto provoca que los sistemas tradicionales pierdan precisión cuando se aplican a videos espontáneos capturados en ambientes naturales, especialmente si el rostro no está completamente alineado o si la calidad del video es inferior a la ideal.

Por otro lado, uno de los principales desafíos identificados es la necesidad de un sistema que combine:

1. bajo costo computacional,
2. capacidad de procesar video cuadro a cuadro en tiempo real,
3. interpretabilidad de las características extraídas,
4. posibilidad de integrarse a flujos prácticos de análisis audiovisual.

En el ámbito de este proyecto profesional, la problemática se presenta al intentar desarrollar una herramienta capaz de analizar imágenes de personas hablando, sonriendo o realizando microexpresiones sutiles, con el fin de detectar microsonrisas, medir la intensidad de las expresiones y evaluar su grado de espontaneidad. La complejidad recae no solo en capturar los cambios que ocurren en milisegundos, sino en garantizar que el sistema se mantenga robusto frente a variaciones naturales en el rostro, la iluminación y el movimiento.

1.3. Justificación del estudio

El estudio se justifica por la creciente demanda de herramientas automáticas capaces de analizar señales no verbales y expresiones faciales con alta precisión en contextos reales. Existe una necesidad concreta de desarrollar modelos livianos que puedan ser ejecutados sin requerir hardware especializado, permitiendo su uso en contextos educativos, investigativos y profesionales. El análisis de microexpresiones constituye un recurso valioso para la comprensión del comportamiento humano, y su detección automatizada puede contribuir al desarrollo de sistemas de comunicación asistida, interfaces inteligentes y herramientas de evaluación emocional.

Asimismo, desde la perspectiva metodológica señalada por Hernández Sampieri, el planteamiento del problema debe demostrar relevancia científica, práctica y social. En este caso, la investigación aporta relevancia científica al explorar mecanismos de detección geométrica y clasificación híbrida; relevancia práctica al proponer una solución de bajo costo computacional; y relevancia social al facilitar herramientas accesibles para estudiar emociones en ámbitos donde tradicionalmente no se cuenta con equipamiento especializado [7].

1.4. Objetivos de la investigación

Objetivo general

Desarrollar un sistema de detección y análisis de expresiones faciales y microexpresiones a partir de imágenes, utilizando modelos livianos y técnicas de aprendizaje automático para lograr una clasificación precisa y explicable de los eventos emocionales detectados.

Objetivos específicos

1. Estudiar los fundamentos teóricos necesarios para el desarrollo de un sistema de clasificación basado en la identificación y caracterización de patrones

gestuales representativos de las microexpresiones en general y las microsonrisas en particular.

2. Evaluar y seleccionar arquitecturas de procesamiento de video e imágenes considerando su capacidad para extraer características gestuales relevantes y la viabilidad de implementación dentro de las restricciones computacionales existentes.
3. Construir un conjunto de datos etiquetado de imágenes faciales que contengan ejemplos representativos de microsonrisas, asegurando su idoneidad para el entrenamiento y validación de un modelo de clasificación basado en aprendizaje automático.
4. Diseñar, entrenar y validar el modelo de clasificación de microsonrisas mediante la implementación de un pipeline completo que incluye la extracción de características (embeddings), el entrenamiento supervisado, el ajuste de hiperparámetros y la evaluación de rendimiento, resultando en la construcción de la prueba de concepto funcional.
5. Implementar un módulo de extracción de landmarks faciales mediante MediaPipe Face Mesh para capturar cambios geométricos cuadro a cuadro.
6. Integrar DeepFace para realizar el reconocimiento emocional y la comparación de embeddings.
7. Calcular métricas geométricas relevantes (distancias, aperturas y variaciones musculares faciales).
8. Diseñar y entrenar un modelo de clasificación (Random Forest) basado en las características extraídas.
9. Evaluar el desempeño del sistema mediante métricas estándar (accuracy, recall, precision, F1-score)
10. Identificar limitaciones, posibles fuentes de error y oportunidades de optimización para futuras implementaciones.

1.5. Alcances y delimitaciones

La presente investigación se circunscribe al análisis de imágenes previamente grabadas, procesadas localmente y con una única persona en cuadro. El sistema no incluye análisis multimodal (audio-emoción y análisis temporal), detección de

múltiples individuos simultáneamente ni sincronización avanzada entre modelos de visión y lenguaje. Tampoco aborda la detección en tiempo real con GPU de alto rendimiento, aunque sienta las bases para escalar hacia dicho escenario.

A su vez, la metodología se centra en la extracción geométrica y clasificación basada en aprendizaje automático clásico, no en el entrenamiento de redes profundas completas debido a las restricciones de hardware y al enfoque en la eficiencia computacional.

1.6. Estructura del documento

El siguiente capítulo aborda el marco teórico, exponiendo los fundamentos del análisis facial, las arquitecturas MediaPipe y DeepFace, el estado del arte en microexpresiones. Posteriormente, se describe la metodología adoptada, seguida por la implementación técnica, la discusión de resultados y las conclusiones.

2. Marco Teórico

2.1. Inteligencia Artificial (IA)

La Inteligencia Artificial (IA) es una rama de la informática cuyo objetivo es desarrollar sistemas capaces de ejecutar tareas que, tradicionalmente, requieren inteligencia humana. Esto incluye procesos como razonamiento, percepción, toma de decisiones, reconocimiento de patrones y capacidad de aprendizaje [8]. Desde una perspectiva clásica, la IA se dividía en enfoques basados en reglas (en los que el comportamiento del sistema se definía mediante lógica formal) y enfoques estadísticos, que permitían aprender a partir de datos. Con el avance del procesamiento computacional, el paradigma estadístico se consolidó como dominante, dado que permite modelar fenómenos complejos y no lineales con mayor flexibilidad [9], especialmente en áreas como la visión por computadora.

En el contexto del análisis facial, la IA resulta fundamental porque permite modelar relaciones sutiles y multidimensionales entre señales visuales (como texturas, geometría, distancias y variaciones temporales) que no pueden ser descritas mediante reglas rígidas. Un rostro humano presenta microvariaciones imposibles de capturar con programación explícita, por lo que se requiere un enfoque basado en aprendizaje automático (Machine Learning, ML) para comprenderlas y clasificarlas con precisión [10]. En el caso específico de las microsonrisas (gestos involuntarios, breves y extremadamente sutiles) los métodos de IA y ML se vuelven la única vía viable para detectar patrones que incluso pueden no ser percibidos por observadores humanos no entrenados [11].

2.2. Aprendizaje automático (ML)

El Aprendizaje Automático (ML) es un subcampo de la IA que se centra en el desarrollo de algoritmos capaces de aprender patrones a partir de datos, ajustando sus parámetros internos en lugar de seguir instrucciones rígidamente programadas [9]. En términos generales, el ML se divide en tres grandes categorías.

La primera de ellas es el aprendizaje supervisado, donde el modelo aprende a partir de ejemplos etiquetados. En este proyecto, por ejemplo, las entradas corresponden a características geométricas del rostro (a_upper, a_lower, ratio_aperture,

smile_strength, mouth_open, mouth_witdh y ,lower_lip_dist) y la etiqueta a predecir es *is_micro_smile* (0 o 1). El propósito es que el algoritmo pueda generalizar y asignar correctamente la etiqueta a datos nunca vistos. La segunda categoría es el aprendizaje no supervisado, que identifica patrones ocultos, agrupa datos o reduce dimensionalidad sin utilizar etiquetas previas; este enfoque no es central en el presente trabajo. Por último se encuentra el aprendizaje semi-supervisado *y por refuerzo*, que combinan información etiquetada, no etiquetada o recompensas; tampoco son utilizados en este trabajo.

Una de las principales ventajas del ML en el análisis de microsonrisas es su capacidad para integrar múltiples características geométricas del rostro (curvaturas, distancias y aperturas) y capturar relaciones no lineales entre ellas, algo esencial para detectar un gesto tan delicado y fugaz como una microsonrisa [10], [12].

2.3. Bosques Aleatorios (Random Forests)

El algoritmo de Random Forest es uno de los métodos más utilizados y confiables en Machine Learning para problemas de clasificación y regresión. Pertenece a la familia de los modelos de ensamble, que combinan múltiples modelos individuales para producir un resultado más robusto [13].

Funcionamiento general:

1. Construye muchos árboles de decisión independientes, cada uno entrenado con un subconjunto distinto del dataset mediante *bagging* (muestreo con reemplazo).
2. Cada árbol genera una predicción.
3. El bosque obtiene el resultado final mediante votación mayoritaria (clasificación) o promedio (regresión).

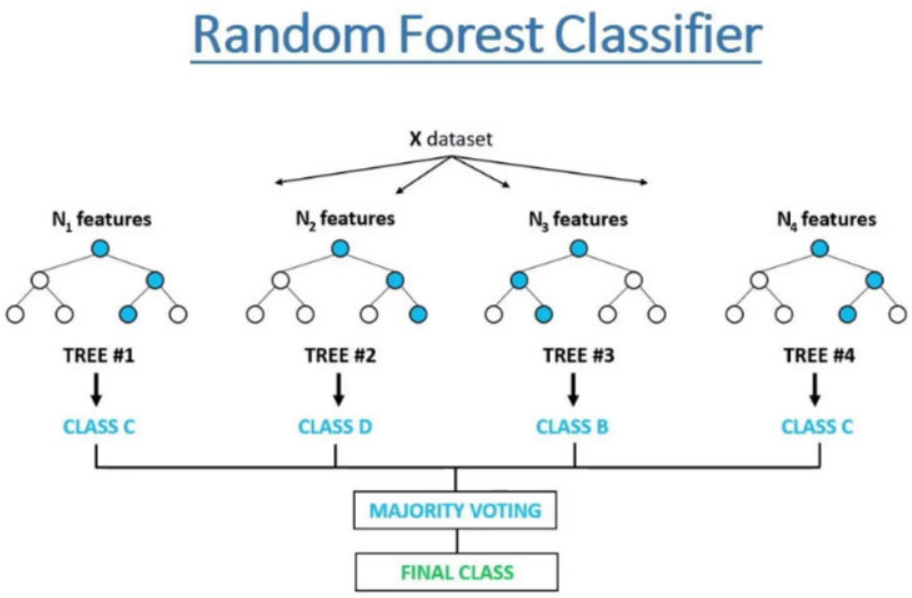

Figura 1. Diagrama de un Random Forest, Fuente: https://www.youtube.com/watch?v=goPiwckWE9M

A diferencia de modelos lineales, el Random Forest puede aprender interacciones no lineales entre características faciales sutiles [14]. Una microsonrisa no depende de un único punto del rostro:

- Una mínima curvatura del labio puede indicar sonrisa o neutralidad.
- La apertura de la boca puede indicar sonrisa o simplemente habla.
- Microgestos de ojos pueden reforzar o contradecir el gesto bucal.

Random Forest integra todos estos factores sin necesidad de reglas manuales, además, es altamente tolerante al ruido, ideal para datos obtenidos de landmarks, donde pueden existir fluctuaciones debido a iluminación, variaciones de ángulo o mínimos errores en el seguimiento facial [15]. Otra ventaja importante es que no requiere escalado estricto de variables, lo cual simplifica el preprocesamiento. En este tipo de datos geométricos algunas características tienen valores muy pequeños, otras están normalizadas y finalmente, otras dependen del tamaño del rostro.

La interpretabilidad también es un punto fuerte. Mediante la métrica de *feature importance*, es posible identificar cuáles rasgos (curvatura del labio, apertura, distancia entre comisuras) son los más determinantes para clasificar una microsonrisa [16]. Esto permite vincular los resultados computacionales con la teoría facial.

Finalmente, Random Forest funciona de manera eficiente en CPU, sin necesidad de GPU, lo que lo hace ideal para prototipos, entornos de bajo costo y aplicaciones de tiempo real.

2.4 MediaPipe FaceMesh

MediaPipe es un framework desarrollado por Google Research orientado al procesamiento multimodal de señales en tiempo real, incluyendo imágenes, video, audio y datos sensoriales. Su principal objetivo es proporcionar una infraestructura optimizada para el despliegue de modelos de Machine Learning en dispositivos de baja y mediana potencia, tales como notebooks sin GPU dedicada o dispositivos móviles [17]. Esto se logra mediante un sistema basado en *computational graphs*, donde cada nodo corresponde a una etapa del pipeline, permitiendo implementar modelos complejos con eficiencia computacional y estabilidad temporal.

Dentro de este ecosistema, MediaPipe Face Mesh destaca como una de las soluciones más avanzadas para el análisis facial. El módulo es capaz de detectar 468 landmarks 3D (Figura 2) en tiempo real con una precisión y velocidad que superan ampliamente a los enfoques tradicionales basados exclusivamente en redes convolucionales profundas instaladas de manera aislada. Esta eficiencia proviene de su arquitectura híbrida: el sistema emplea un modelo ligero de detección de rostro y un regresor especializado para estimar la malla facial completa, optimizados ambos para ejecutarse incluso en CPU sin degradación perceptible del rendimiento [17].

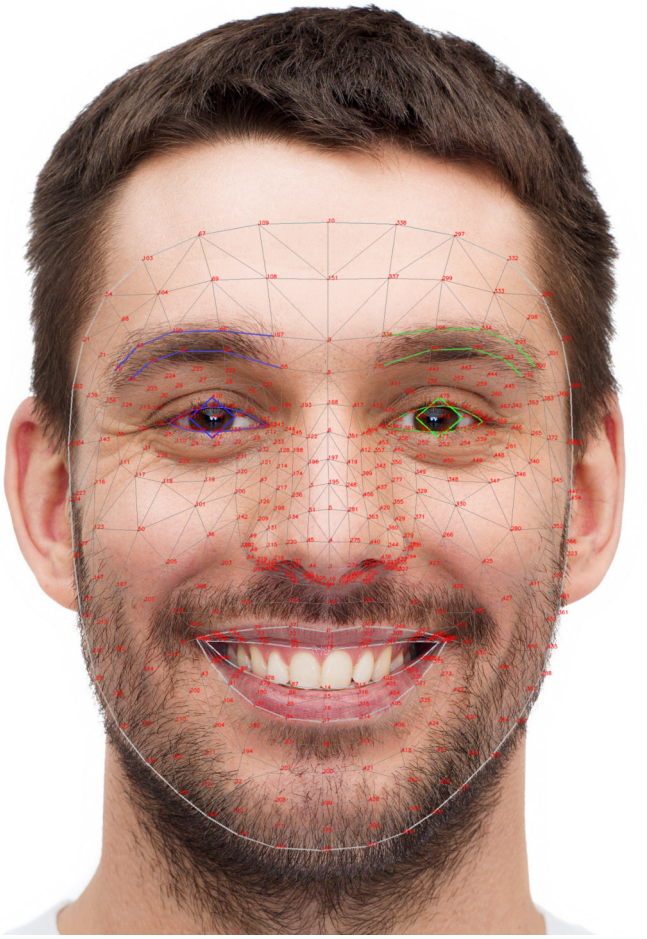

Figura 2. FaceMesh generado por Mediapipe Fuente: https://ai.google.dev/edge/mediapipe/solutions/vision/face_landmarker?hl=es-419

El funcionamiento del pipeline comienza con la captura del fotograma de entrada. MediaPipe realiza primero una etapa de detección de rostro seguida de un recorte (*face crop*) normalizado a una resolución fija de 256 × 256 × 3 píxeles, lo cual permite al modelo operar con entradas homogéneas independientemente de la distancia, la pose o la iluminación del sujeto. Una vez obtenida esta región, se introduce en un *feature extractor* convolucional altamente optimizado para tiempo real. Este módulo extrae representaciones jerárquicas que describen texturas, bordes, gradientes y estructuras espaciales del rostro a diferentes resoluciones internas, preservando así tanto el detalle local como la organización global de la cara [17].

A partir de estos mapas de características, el sistema implementa el modelo Attention Mesh, un avance presentado por Grishchenko et al. (2020), que incorpora

mecanismos de atención para priorizar regiones críticas del rostro. El pipeline divide la atención en tres áreas principales: los ojos e iris, la malla facial global y los labios. Cada región recibe un nivel distinto de precisión dependiendo de su importancia para la reconstrucción final. Por ejemplo, los ojos requieren un nivel excepcionalmente alto de detalle debido a que pequeñas variaciones producen cambios visuales significativos; la malla global necesita equilibrar detalle y estabilidad; los labios, por su parte, contienen información especialmente relevante para detectar microexpresiones y variaciones musculares asociadas a la sonrisa [18].

El mecanismo de atención permite redistribuir los recursos del modelo para asignar mayor capacidad predictiva a las áreas donde la variabilidad fisiológica es más compleja. En el pipeline visual se observa cómo, desde resoluciones intermedias como 24 × 24 × 32 o 64 × 64 × 32, se extraen regiones específicas del mapa de características que alimentan ramas independientes dedicadas a la predicción del iris, la geometría global del rostro o el contorno labial. Esta estructura permite modelar con precisión detalles que resultan difíciles de capturar con un procesamiento uniforme, manteniendo al mismo tiempo una velocidad de ejecución adecuada para sistemas en tiempo real [18].

Las predicciones generadas por las tres ramas del modelo se fusionan luego en una única representación tridimensional del rostro. Cada uno de los 468 puntos de la malla se estima en coordenadas normalizadas y es posteriormente estabilizado a través de un mecanismo de regresión entrenado con datos anotados manualmente y con muestras generadas sintéticamente, lo que incrementa la robustez frente a variaciones de iluminación, ruido de cámara y cambios leves de pose [19].

Este diseño permite que MediaPipe Face Mesh alcance velocidades de entre 30 y 60 FPS incluso en dispositivos móviles, convirtiéndose en un estándar industrial para aplicaciones de realidad aumentada, avatarización facial, seguimiento de expresiones en tiempo real, análisis emocional, y reconstrucción geométrica de alta precisión [17], [18], [19].

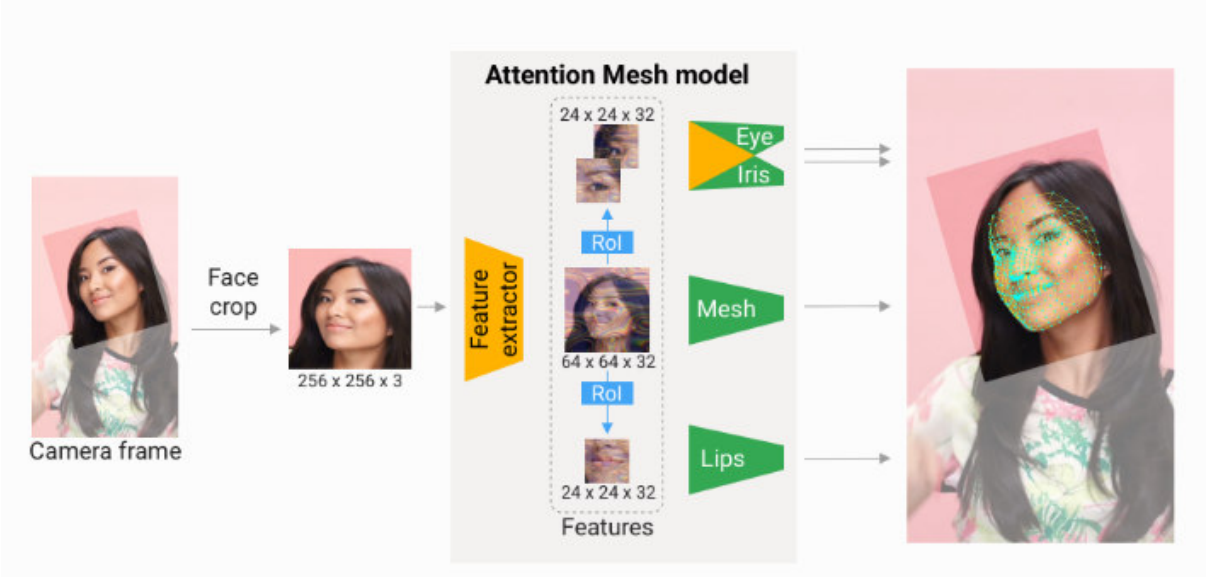

Figura 3. Pipeline de Mediapipe Fuente: Attention Mesh: High-fidelity Face Mesh Prediction in Real-time [18].

En el contexto específico del análisis de microsonrisas, MediaPipe ofrece varias ventajas fundamentales. Su capacidad para generar landmarks submilimétricos permite detectar variaciones faciales extremadamente sutiles, coherentes con la duración típica de una microexpresión (entre 1/25 y 1/3 de segundo). Además, su estabilidad temporal asegura que los movimientos minúsculos de labios, mejillas o ojos no se confundan con artefactos del modelo, lo cual resulta esencial cuando se buscan cambios de muy baja amplitud. Debido a estas características, MediaPipe constituye un componente crítico en el pipeline del proyecto, posibilitando la extracción confiable de información geométrica que posteriormente se utiliza como entrada para algoritmos de clasificación tradicionales, como los Bosques Aleatorios.

2.5. Deepface

DeepFace es un framework de análisis facial ampliamente utilizado para la estimación de identidad y la inferencia de atributos faciales. Su diseño se basa en arquitecturas previamente entrenadas, como VGG-Face, FaceNet, ArcFace y SFace, las cuales producen representaciones embebidas (embeddings) en espacios

vectoriales capaces de codificar información semántica del rostro, como expresión, estructura y consistencia geométrica [20]. Este framework fue originalmente concebido para reconocimiento facial robusto, pero sus módulos adicionales permiten realizar inferencia de emociones mediante clasificadores específicos entrenados sobre grandes repositorios de rostros etiquetados [21].

En el sistema propuesto, no se utiliza DeepFace como modelo de reconocimiento facial ni como extractor de características profundas debido a que su procesamiento es más costoso que las técnicas geométricas y, además, introduciría un sesgo metodológico incompatible con el enfoque clásico del proyecto. En cambio, su función se circunscribe a proporcionar una referencia emocional inicial en un único fotograma, antes de iniciar la extracción de características geométricas. Durante esta etapa, el programa analiza el primer fotograma estable en el que el rostro es detectado, obteniendo la emoción dominante reportada por DeepFace. Este valor se utiliza únicamente para decidir si los parámetros geométricos iniciales corresponden a un estado neutro o si deben ajustarse debido a la presencia de una sonrisa previa.

2.6. Estado del Arte

La investigación en reconocimiento automático de expresiones y microexpresiones faciales ha evolucionado de manera significativa en las últimas dos décadas, impulsada por avances en visión por computadora, modelado geométrico, representación temporal y métodos de clasificación robustos. El análisis de sonrisas, microsonrisas y gestos voluntarios e involuntarios del rostro constituye una de las áreas más estudiadas dentro del comportamiento no verbal, debido a su relevancia en psicología, interacción humano–computadora, seguridad y análisis afectivo.

El reconocimiento de expresiones faciales (FER, por sus siglas en inglés) ha evolucionado significativamente en las dos últimas décadas. Inicialmente, el enfoque se centraba en la detección de expresiones faciales básicas como la felicidad, la tristeza y la sorpresa. Sin embargo, los estudios más avanzados ahora se enfocan en expresiones sutiles e involuntarias, como las microexpresiones, que brindan una visión de las emociones suprimidas. Esta área ha crecido debido a las mejoras en

sistemas de visión, aprendizaje automático, aprendizaje profundo y modelado geométrico [22], [23].

La detección de sonrisas y microsonrisas es particularmente relevante, ya que estas expresiones faciales juegan un papel esencial en campos como la psicología, la interacción humano-computadora, la seguridad y la computación afectiva [24]. Las sonrisas, así como las expresiones faciales involuntarias, pueden indicar sentimientos de felicidad, incomodidad o incluso engaño, lo que hace que su detección precisa sea crucial en múltiples dominios [25].

Los avances significativos en el reconocimiento facial de emociones han sido respaldados por avances tanto en aprendizaje automático como en aprendizaje profundo. Los primeros métodos para el reconocimiento de sonrisas dependían en gran medida de características hechas a mano (por ejemplo, bordes, texturas y características geométricas de la cara) y clasificadores básicos como máquinas de soporte vectorial y Random Forest [22]. Estos métodos tuvieron éxito en entornos controlados, pero tuvieron dificultades con desafíos del mundo real, como variaciones en las condiciones de iluminación, la orientación facial y las oclusiones.

Con la llegada del aprendizaje profundo, particularmente de las redes neuronales convolucionales, la precisión en el reconocimiento de sonrisas y microexpresiones mejoró dramáticamente. Las CNN permiten la extracción automática de características de los datos de píxeles en bruto, eliminando la necesidad de ingeniería manual de características [22] [24]. Las técnicas híbridas que involucran redes neuronales recurrentes y redes generativas antagónicas han mejorado aún más la capacidad de analizar expresiones faciales dinámicas a lo largo del tiempo, haciendo que el reconocimiento de microexpresiones sea factible [22].

Las expresiones requieren cámaras de alta velocidad y técnicas avanzadas de interpolación para capturarlas y analizarlas de manera efectiva [24] [26]. Los investigadores han logrado avances significativos en el desarrollo de conjuntos de datos y algoritmos capaces de detectar estas expresiones fugaces, con resultados prometedores utilizando Multiple Kernel Learning y clasificadores Random Forests [24]. Sin embargo, la precisión humana en el reconocimiento sigue siendo baja, lo que resalta la necesidad de mejores soluciones algorítmicas.

Estudios, como los de Ouarti et al. [25], han explorado las diferencias entre sonrisas espontáneas y sonrisas posadas mediante el análisis de características espaciales, como el desplazamiento de los puntos de referencia faciales. Estas características son fundamentales para distinguir expresiones genuinas de las falsas, ya que las sonrisas espontáneas muestran un patrón de movimiento más simétrico y complejo que las sonrisas posadas. La duración, amplitud, velocidad de inicio, y simetría son características temporales importantes en el reconocimiento de expresiones posadas vs. espontáneas. Los estudios han mostrado que las sonrisas espontáneas tienden a tener un inicio y un final más lento en comparación con las sonrisas posadas [25] [26].

En conjunto, la literatura muestra que los avances en el reconocimiento de sonrisas y microsonrisas han dependido de métodos capaces de capturar variaciones geométricas sutiles, movimientos rápidos, contexto muscular circundante y relaciones no lineales entre múltiples rasgos faciales, lo que justifica plenamente el uso de estrategias basadas en landmarks, características temporales y clasificadores robustos como Random Forest en este proyecto.

3. Metodología y diseño del sistema

El diseño metodológico del sistema parte de la dificultad de acceder a datasets especializados en microexpresiones, los cuales suelen ser pequeños, con restricciones de acceso o limitados a contextos de laboratorio. Trabajar con microexpresiones espontáneas implica desafíos adicionales, ya que estas duran entre 1/25 y 1/5 de segundo y requieren una alta precisión para ser identificadas correctamente. Frente a este contexto, se optó por la construcción de un dataset propio, generado mediante la captura de video a través de una webcam convencional y el procesamiento de los frames utilizando la malla facial de MediaPipe FaceMesh [17][19]. Este enfoque permite controlar de manera exacta las condiciones de captura, asegurando coherencia en iluminación, distancia y posición. Además, al ser el autor del proyecto el sujeto de prueba, se facilita realizar múltiples iteraciones del mismo gesto para obtener variabilidad interna suficiente. Este procedimiento también es coherente con trabajos clásicos donde se generan datasets ad hoc para analizar expresiones bajo condiciones controladas, como CK+ y otros repositorios experimentales que comienzan con gestos posados antes de pasar a expresiones más naturales [1][3].

Para la construcción del dataset, se optaron dos caminos complementarios. El primer conjunto se etiqueta en base a métodos matemáticos y geométricos, utilizando distancias entre landmarks, índices normalizados, curvatura labial. Este tipo de enfoques ha sido utilizado históricamente en reconocimiento facial, especialmente en métodos geométricos basados en puntos claves del rostro [22][23]. El segundo modelo recurre a etiquetado manual en tiempo real: el usuario presiona una tecla cuando considera que está realizando una microsonrisa, generando así un dataset supervisado para comparar rendimiento y consistencia entre métodos automáticos y humanos.

La totalidad del sistema fue desarrollada en Python 3.9.13, debido a su ecosistema estable, maduro y fuertemente adoptado por la comunidad científica y de machine learning. Python es el lenguaje predominante para frameworks como TensorFlow, PyTorch, Mediapipe o DeepFace, siendo ampliamente respaldado para proyectos de visión por computadora y análisis estadístico. Python constituye la base del proyecto por su sintaxis clara, su facilidad de mantenimiento y su ecosistema científico rico en

librerías optimizadas para visión por computadora, machine learning y análisis numérico. A nivel académico, Python es considerado el estándar para prototipado rápido y experimentación en entornos científicos [27].

OpenCV se emplea para la captura de video, la conversión de colores, el redimensionamiento de frames y la visualización con anotaciones. Su motor está optimizado en C++ y expuesto a Python, permitiendo procesamiento en tiempo real a altos FPS incluso en hardware modesto. Históricamente, OpenCV ha sido la biblioteca dominante para tareas de visión computacional y procesamiento digital de imágenes, ampliamente utilizada en investigación y aplicaciones industriales [7][28].

MediaPipe FaceMesh se utiliza para extraer 468 puntos clave del rostro en cada frame, proporcionando una malla 3D extremadamente detallada. Su ventaja principal radica en su estabilidad temporal y su capacidad de funcionar en tiempo real, incluso sin GPU dedicada. Estas características lo convierten en una herramienta ideal para detectar microvariaciones faciales que aparecen durante una microsonrisa, tal como ha sido demostrado en los trabajos de Google Research que introducen FaceMesh y Attention Mesh [17][18][19].

DeepFace se integra en el sistema para obtener una segunda perspectiva del comportamiento facial, complementando el análisis geométrico con información emocional generada automáticamente. Esto permite verificar si un gesto corresponde efectivamente a una sonrisa o si está asociado a emociones alternativas como sorpresa o miedo. DeepFace se basa en arquitecturas de deep learning ampliamente reconocidas y utilizadas en estudios de verificación facial, como VGGFace2 y DeepFace de Facebook AI Research [20][21].

Scikit-Learn se utiliza para entrenar los modelos supervisados, incluyendo SVM, Random Forest y regresión logística. La librería es considerada un estándar en machine learning tradicional, especialmente para datasets pequeños o medianos como los generados en este proyecto. Su diseño limpio, sus transformadores modulares y sus implementaciones altamente optimizadas han sido documentadas extensamente en la literatura técnica moderna [31].

Matplotlib permite generar gráficos temporales que muestran la evolución de medidas geométricas del rostro (distancias, ángulos, ratios). Este análisis visual es

esencial para validar patrones que caracterizan una microsonrisa, tal como se realiza en múltiples estudios que analizan series temporales de movimiento facial para distinguir expresiones genuinas de expresiones posadas [32].

Finalmente, Joblib se utiliza para guardar y cargar modelos de manera eficiente. A diferencia de pickle, Joblib está optimizado para objetos grandes como clasificadores entrenados, lo que permite reutilizar modelos sin necesidad de reentrenamiento. Este enfoque es ampliamente aplicado en flujos de trabajo científicos donde los modelos deben ser portables, reproducibles y fácilmente distribuidos [14][33].

4. Implementación

4.1 Construcción del conjunto de datos con embeddings

En primer lugar se explicará cómo se utiliza Mediapipe en el proyecto y cuales son los puntos de interés de FaceMesh para realizar en análisis las microsonrisas. El sistema depende de la detección precisa de landmarks faciales. Para ello se instancia el modelo FaceMesh, como se puede observar en el código siguiente:

```
mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(
    static_image_mode=False,
    max_num_faces=1,
    refine_landmarks=True,
    min_detection_confidence=0.7,
    min_tracking_confidence=0.7
)
```

El parámetro static_image_mode=False permite al modelo utilizar información temporal entre frames consecutivos, reduciendo la variabilidad abrupta de los landmarks y estabilizando la trayectoria de movimientos faciales sutiles. Esto es esencial porque las microsonrisas suelen ocurrir en lapsos inferiores a 500 ms y con amplitudes faciales mínimas, por lo que cualquier salto o jitter en la estimación puede generar falsos positivos o enmascarar variaciones reales. La restricción max_num_faces=1 se emplea para optimizar el rendimiento y asegurar que solo se analice al usuario principal en escena, eliminando interferencias de personas en segundo plano y simplificando la lógica de post-procesamiento. Dado que el análisis del sistema se centra en un único individuo, mantener esta limitación reduce la carga computacional y aumenta la estabilidad del seguimiento [29].

Uno de los elementos más críticos es el uso de refine_landmarks=True, que activa un modelo extendido capaz de mejorar la precisión de los landmarks en zonas clave como el contorno de los labios, los párpados y el iris. Este refinamiento es imprescindible para la detección de microsonrisas, porque dichas expresiones dependen de micromovimientos en la curvatura labial, elevación mínima de las comisuras y variaciones milimétricas en la tensión muscular peribucal. El modelo refinado de MediaPipe incorpora una red adicional entrenada con supervisión más

densa sobre estas regiones, lo que incrementa la resolución geométrica del mesh y permite capturar deformaciones que no serían detectables con el modelo estándar [29]

Los parámetros min_detection_confidence=0.7 y min_tracking_confidence=0.7 establecen umbrales mínimos de confiabilidad tanto para la detección inicial del rostro como para el seguimiento entre cuadros. Un valor inferior aumentaría la sensibilidad del sistema, pero también generaría mayor ruido debido a falsas detecciones en condiciones de iluminación variable o movimientos rápidos. Mantener ambos valores en 0.7 permite un equilibrio adecuado entre robustez y estabilidad, garantizando que los landmarks evolucionen suavemente a lo largo del tiempo y mantengan coherencia espacial incluso en secuencias prolongadas de vídeo [29]. Esta estabilidad es indispensable para medir cambios relativos como la apertura labial o la variación en la curvatura.

En el contexto específico del análisis de microsonrisas, la elección de FaceMesh refinado resulta obligatoria. Las microsonrisas se caracterizan por alteraciones mínimas en la curvatura labial y patrones de activación muscular que pueden durar apenas fracciones de segundo. Dado que estos cambios son extremadamente sutiles, los sistemas de landmarks tradicionales (que suelen presentar errores promedio de varios píxeles) no son adecuados para esta tarea. FaceMesh, por el contrario, ofrece una malla de 468 puntos con coherencia topológica y actualizaciones de alta frecuencia, lo que permite capturar deformaciones mínimas del contorno de la boca, incluso en condiciones subóptimas de grabación. Además, el modelo está optimizado para ejecutar en tiempo real en CPU y GPU, lo que habilita un análisis continuo y sin interrupciones, crucial para estudios de comportamiento facial y análisis afectivo automatizado [29].

En consecuencia, la combinación de los parámetros seleccionados y el uso del modelo refinado proporciona la precisión y estabilidad necesarias para medir microvariaciones faciales como la elevación relativa de comisuras, el cierre parcial o apertura mínima de labios y la tensión distribuida en el área peribucal. Estos aspectos convierten a MediaPipe FaceMesh en la herramienta más adecuada para

detectar microsonrisas en aplicaciones de visión computacional orientadas al comportamiento emocional.

La detección de microsonrisas se basa en la correcta selección y manipulación de landmarks faciales, dado que estas expresiones producen variaciones mínimas en la geometría del rostro. Para ello, el sistema define un conjunto preciso de puntos que describen el contorno del labio superior, el labio inferior, las comisuras y ciertas regiones del área ocular. Estos puntos son fundamentales para calcular medidas como curvaturas labiales, aperturas verticales y horizontales de la boca, así como distancias de referencia necesarias para la normalización. Entre los conjuntos de landmarks utilizados se incluyen:

```
UPPER_CURVE = [191, 80, 81, 82, 13, 312, 311, 310, 415]
LOWER_CURVE = [95, 88, 178, 87, 14, 317, 402, 318, 324]
```

Como se puede observar en la figura 4, estos puntos se distribuyen de manera simétrica alrededor del eje vertical de la boca y permiten capturar la geometría esencial de los labios durante el gesto de sonrisa. La elección de estos landmarks no es arbitraria: corresponden a regiones anatómicas directamente involucradas en los movimientos faciales asociados a la elevación de las comisuras y a la modificación de la curvatura labial, elementos clave en la manifestación de la sonrisa.

La fundamentación teórica de esta selección de landmarks y métricas se apoya directamente en el Facial Action Coding System (FACS). El FACS es un sistema descriptivo y anatómicamente basado para el análisis del movimiento facial humano, desarrollado por Paul Ekman y Wallace V. Friesen. A diferencia de los enfoques categóricos que asignan etiquetas emocionales globales, el FACS descompone las expresiones faciales en Unidades de Acción (Action Units, AU), cada una asociada a la contracción de uno o más músculos faciales específicos [23].

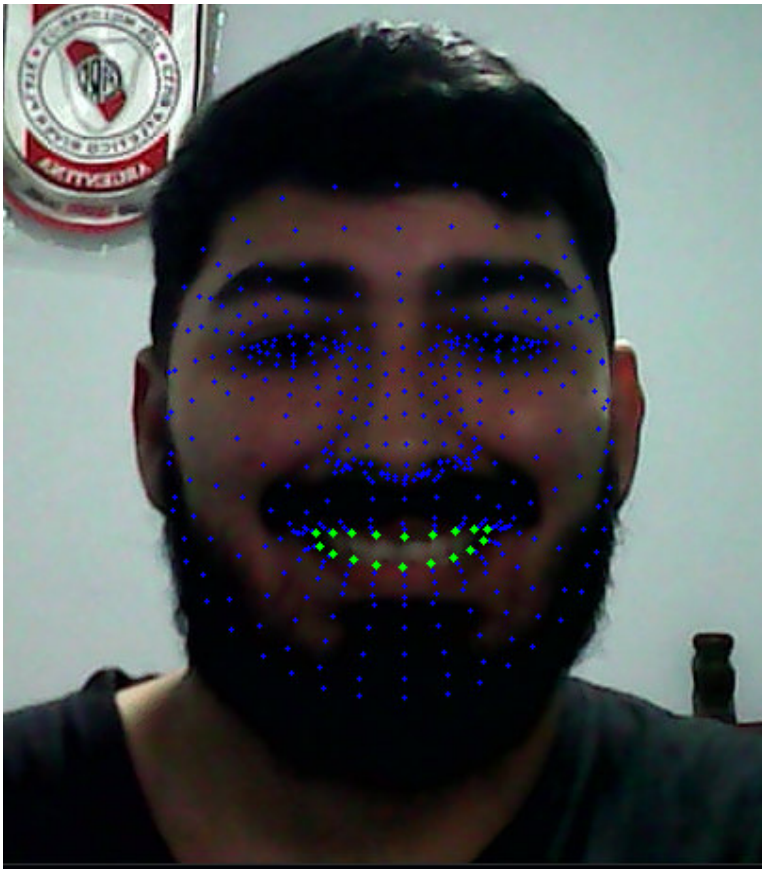

Figura 4. FaceMesh en verde se resaltan los UPPER_CURVE Points LOWER_CURVE Points. Fuente: producción propia a partir de Mediapipe.

En el caso de la sonrisa, y particularmente de la microsonrisa, la unidad de acción más relevante es la AU12 (Lip Corner Puller). Esta AU está asociada principalmente a la contracción del músculo cigomático mayor, responsable de elevar las comisuras labiales. En expresiones voluntarias intensas, la AU12 produce desplazamientos visibles y sostenidos; sin embargo, en microexpresiones, dicha activación es breve, de baja amplitud y frecuentemente incompleta [23].

A continuación destacamos otro grupo de landmarks importantes, representando el centro de los ojos, y la boca.

```
LEFT_EYE_CENTER = 468
RIGHT_EYE_CENTER = 473

MOUTH_LEFT = 78
MOUTH_RIGHT = 308
```

```
UPPER_CENTER = 13
LOWER_CENTER = 14
```

En la siguiente figura se pueden observar los landmarks anteriores.

Figura 5. FaceMesh en verde se resaltan los puntos LEFT_EYE_CENTER, RIGHT_EYE_CENTER, MOUTH_LEFT, MOUTH_RIGHT, UPPER_CENTER y LOWER_CENTER. Fuente: producción propia a partir de Mediapipe.

Para convertir los landmarks en medidas cuantificables, el sistema implementa funciones auxiliares diseñadas para capturar propiedades geométricas clave. Una de ellas calcula la curvatura del labio mediante un ajuste polinomial de segundo grado, como se muestra en la función siguiente:

```
def curvature(points):
    if len(points) < 3:
        return 0.0
    xs = np.array([p[0] for p in points], dtype=float)
    ys = np.array([p[1] for p in points], dtype=float)
    xs -= np.mean(xs)
    ys -= np.mean(ys)
    coeffs = np.polyfit(xs, -ys, 2)
    return coeffs[0] * 1000
```

La centralización de los datos evita desplazamientos irrelevantes, mientras que la inversión del eje Y asegura coherencia con la convención gráfica usada para representar curvaturas. El coeficiente cuadrático del polinomio describe la concavidad o convexidad del labio; multiplicarlo por 1000 amplifica la variación para hacerla más utilizable en la clasificación. Este método resulta particularmente efectivo para detectar microsonrisas debido a que estas producen cambios extremadamente sutiles en la forma labial, difíciles de capturar con distancias lineales simples.

Otro cálculo relevante es la distancia promedio entre puntos del labio inferior, obtenido mediante la función *mean_pairwise_distance(points)*, cuyo propósito es caracterizar la “inflación” o tensión de esta región. Esta medida es especialmente útil para discriminar entre una sonrisa y movimientos asociados al habla o tensión emocional. En conjunto, estas funciones permiten construir un set robusto de características faciales derivadas directamente de la geometría.

```
def mean_pairwise_distance(points):
    if len(points) < 2:
        return 0.0
       dists = [math.dist(points[i], points[i + 1]) for i in
range(len(points) - 1)]
    return np.mean(dists)
```

Las métricas que surgen en base a estas funciones son:

```
# --- Curvaturas globales ---
pts_upper = [(int(face_landmarks.landmark[i].x * w),
int(face_landmarks.landmark[i].y * h)) for i in UPPER_CURVE]
pts_lower = [(int(face_landmarks.landmark[i].x * w),
int(face_landmarks.landmark[i].y * h)) for i in LOWER_CURVE]

a_upper = curvature(pts_upper)
a_lower = curvature(pts_lower)
lower_lip_dist = mean_pairwise_distance(pts_lower)
smile_strength = (abs(a_upper) + abs(a_lower)) / 2
```

Un parámetro muy importante es la distancia entre los ojos, la cual funciona como referencia primaria de normalización. Esta distancia se obtiene extrayendo los puntos específicos correspondientes a los bordes inferiores de ambos ojos y calculando la distancia euclidiana:

```
# --- Ojos ---
left_eye_center = face_landmarks.landmark[LEFT_EYE_CENTER]
right_eye_center = face_landmarks.landmark[RIGHT_EYE_CENTER]
eye_distance = math.dist(
                (left_eye_center.x * w, left_eye_center.y * h),
                (right_eye_center.x * w, right_eye_center.y * h)
            )
```

Todas las métricas posteriores se dividen por esta distancia, garantizando que las medidas sean comparables entre individuos y entre diferentes sesiones del mismo individuo. La normalización evita que aperturas labiales pequeñas generen falsos positivos simplemente por estar capturadas muy cerca de la cámara.

Posteriormente, el sistema calcula la apertura vertical de la boca, parámetro fundamental para distinguir una microsonrisa cerrada de una sonrisa plena o del habla. Para ello se miden los puntos centrales del labio superior e inferior, como se muestra en las siguientes expresiones:

```
upper_lip = face_landmarks.landmark[UPPER_CENTER]
lower_lip = face_landmarks.landmark[LOWER_CENTER]
upper_center = (upper_lip.x * w, upper_lip.y * h)
lower_center = (lower_lip.x * w, lower_lip.y * h)
mouth_open = math.dist(upper_center, lower_center)
normalized_mouth_open = mouth_open / eye_distance
```

Una vez normalizado, este valor adquiere un significado interpretable: valores altos indican una boca abierta, típicamente relacionados con el habla o sorpresa; valores intermedios suelen corresponder a una sonrisa tenue; y valores muy bajos reflejan neutralidad o incluso presión labial. Para el análisis de microsonrisas, este parámetro es uno de los más relevantes, especialmente cuando se observa su variación temporal cuadro a cuadro.

De forma complementaria, también se calcula la apertura horizontal de la boca, parámetro directamente relacionado con el ensanchamiento lateral característico de las microsonrisas cerradas. Esta métrica se obtiene midiendo la distancia entre las comisuras:

```
left_mouth = face_landmarks.landmark[MOUTH_LEFT]
right_mouth = face_landmarks.landmark[MOUTH_RIGHT]
p_left = (left_mouth.x * w, left_mouth.y * h)
p_right = (right_mouth.x * w, right_mouth.y * h)
mouth_width = math.dist(p_left, p_right)
normalized_mouth_width = mouth_width / eye_distance
```

En conjunto, todos estos parámetros convierten la información bruta proporcionada por los *landmarks* en un perfil geométrico detallado. Este perfil, procesado en serie temporal, constituye la base matemática que permite al sistema distinguir de manera robusta una microsonrisa auténtica de otras configuraciones faciales similares. El diseño de estas métricas responde directamente al objetivo del proyecto: capturar expresiones extremadamente sutiles mediante mediciones objetivas, cuantificables y normalizadas, evitando depender de inferencias puramente emocionales o clasificadores preentrenados que no detecta microvariaciones.

El sistema incorpora un módulo de procesamiento emocional auxiliar basado en DeepFace, cuyo objetivo no es reemplazar el análisis geométrico de la microsonrisa [30], sino aportar una capa adicional de contexto que permite calibrar el estado emocional basal del sujeto antes de iniciar la medición. DeepFace se ejecuta únicamente en momentos específicos, principalmente durante el período de calibración inicial, donde se requiere determinar si la expresión facial predominante corresponde a un estado neutral o presenta activaciones emocionales previas que puedan sesgar las mediciones. El fragmento de código responsable de esta operación es el siguiente:

```
# --- Calibración —
if not is_calibrated:
   try:
          analysis = DeepFace.analyze(frame, actions=['emotion'],
enforce_detection=False)
       dominant_emotion = analysis[0]['dominant_emotion']
   except Exception:
       dominant_emotion = "neutral"
   if dominant_emotion == "neutral":
       mouth_width_base = normalized_mouth_width
       mouth_open_base = normalized_mouth_open
       lower_lip_base = lower_lip_dist
       is_calibrated = True
```

El objetivo central de este proceso es asegurar que el sistema no tome como referencia inicial una expresión que ya esté sesgada hacia la sonrisa o hacia alguna otra emoción. Para ello, DeepFace estima la emoción dominante del frame y devuelve etiquetas probabilísticas basadas en modelos preentrenados. Si la emoción detectada es “neutral”, el sistema asume que el rostro se encuentra en un estado neutral y registra un conjunto de medidas base: ancho de boca, distancias oculares y posición del labio inferior. Estas magnitudes sirven como “punto cero” o referencia geométrica sobre la cual se calculan las variaciones posteriores que determinan la presencia o ausencia de microsonrisas.

Esta verificación emocional cumple tres objetivos clave:

- Neutralidad inicial: evita que el sistema interprete como microsonrisa una variación mínima que se produce sobre un rostro que ya estaba parcialmente sonriendo.
- Corrección de sesgos de expresión: estabiliza el punto de referencia incluso en sujetos que, por características faciales naturales, poseen una curvatura labial elevada o tensiones faciales permanentes.
- Contexto de activación emocional general: permite distinguir microsonrisas genuinas de cambios faciales que aparecen cuando el sujeto experimenta emociones intensas como sorpresa, miedo o disgusto.

El uso de DeepFace en esta etapa es estratégico: se ejecuta solo cuando se necesita confirmar el estado base y no durante todo el video, lo que permite mantener una alta eficiencia computacional mientras se agrega una capa crítica de fiabilidad al proceso de calibración. Una vez establecido el punto neutral, el sistema continúa apoyándose principalmente en las medidas geométricas extraídas de FaceMesh, que constituyen la base numérica del detector de microsonrisas. De esta manera, el análisis emocional funciona como un mecanismo auxiliar que fortalece la robustez del sistema sin interferir con la lógica central basada en la dinámica facial cuantitativa.

Finalmente, se realiza una división de cada uno de las métricas por la métrica base para generar más estabilidad a la hora de determinar umbrales, como se muestra a continuación:

```
# --- Razones ---
ratio_mouth_width = normalized_mouth_width / mouth_width_base
ratio_mouth_open = normalized_mouth_open / mouth_open_base
ratio_lower_lip = lower_lip_dist / lower_lip_base
```

Es importante remarcar las diferentes limitaciones que tiene el modelo generado a partir de la construcción de este conjunto de datos. En primer lugar, pese a los intentos de normalización la distancia a la cámara es un factor crítico a la hora de detectar microsonrisas. En segundo lugar, el funcionamiento de DeepFace y Mediapipe tiene que ser bajo entornos bien iluminados. Finalmente la posición a la cámara tiene que ser de frente para una correcta detección de las librerías mencionadas anteriormente.

Una vez calculadas estas variables, el sistema exporta un dataset completo en formato CSV que registra la información frame-by-frame, permitiendo un análisis estadístico y supervisado extremadamente detallado. Cada fila del archivo contiene:

- **timestamp:** Momento exacto del frame en milisegundos, útil para análisis temporales y sincronización con el video.
- **frame:** Es el número de frame.
- **is_micro_smile:** Etiqueta binaria que indica si en ese instante específico se detectó una microsonrisa según las reglas definidas por el sistema (automáticas o manuales).
- **a_upper:** Representa la curvatura geométrica del labio superior
- **a_lower:** Representa la curvatura geométrica del labio inferior
- **smile_strength:** Índice construido a partir de las características anteriores. Indica la media entre a_upper y a_lower en cada frame. Con esto se puede analizar la intensidad de una sonrisa.
- **mouth_width:** Valor que representa la distancia que existe entre las comisuras de la boca. Clave para detectar ensanchamiento en una sonrisa.
- **mouth_open:** Valor que representa la distancia entre el labio superior y el labio inferior.
- **lower_lip_dist:** Metrica que hace referencia a la media de la distancia entre los puntos del labio inferior. Importante a la hora de analizar posibles tensiones en el labio inferior a la hora de sonreír.

El dataset general está compuesto por embeddings derivados de características geométricas específicas de la región facial, incluyendo curvaturas, aperturas normalizadas y distancias basadas en landmarks críticos. La presencia de la etiqueta binaria is_micro_smile habilita tareas de clasificación y detección. El

dataset fue producido mediante sesiones personalizadas de captura que aseguran variabilidad realista del gesto, preservando intensidades diversas y contextos auténticos de microexpresión.

La carga y preparación del dataset se realiza a partir de estos CSV. Durante la preparación se eliminan columnas irrelevantes como timestamp o frame y se balancea el conjunto de datos para evitar sesgos de clase, especialmente dado que las microsonrisas suelen representar una fracción menor del total de frames capturados. Posteriormente se realiza un muestreo estratificado para dividir los datos en subconjuntos de entrenamiento y prueba, asegurando que ambas particiones contengan proporciones representativas de eventos positivos y negativos.

```
records.append({
                "timestamp": current_time,
                "frame": len(records),
                "is_micro_smile": is_micro_smile,
                "a_upper": a_upper,
                "a_lower": a_lower,
                "mouth_width": ratio_mouth_width,
                "smile_strength": smile_strength,
                "mouth_open": ratio_mouth_open,
                "lower_lip_dist" : ratio_lower_lip
})
```

El preprocesamiento implica normalización, análisis exploratorio, estandarización de magnitudes y verificación de correlaciones entre variables. La normalización garantiza escalas comparables entre los diferentes indicadores faciales. El análisis exploratorio permite identificar patrones, anomalías y posibles redundancias. La verificación de correlaciones es esencial para evaluar la relación entre curvaturas, aperturas y el índice de microsonrisa, permitiendo confirmar la relevancia de las variables seleccionadas y asegurar la eficiencia del modelo final. Este conjunto de pasos constituye la base para entrenar clasificadores sólidos, capaces de detectar microsonrisas reales con precisión y estabilidad temporal.

4.1.1 Construcción del conjuntos de datos con etiquetado automático

El sistema implementa uno de los componentes más delicados del proyecto: la definición del criterio booleano que determina, en cada frame, si la persona está realizando una microsonrisa. A diferencia de una sonrisa completa o de una expresión neutra, la microsonrisa se caracteriza por variaciones sutiles, simultáneas y coordinadas en las estructuras labiales, cuya magnitud es pequeña pero consistente. Por ello, no es posible identificarla mediante un único parámetro; se requiere integrar una combinación ponderada de curvatura, apertura y tensión facial. El archivo main.py opera sobre este principio y construye un índice compuesto que evalúa la relación entre el labio superior e inferior, la apertura de la boca y el ensanchamiento lateral.

El proceso parte de la construcción de una medida compuesta de curvaturas. La curvatura del labio inferior suele ser el indicador más sensible a la aparición de una microsonrisa: incluso cambios muy pequeños en su convexidad ya reflejan una activación músculo cigomático leve. Por otro lado, la curvatura del labio inferior aporta información sobre tensiones compensatorias y movimientos de relajación.

```
is_upper = 1.5 < a_upper < 3
is_lower = 1.5 < a_lower < 3
a_coefficient = (is_upper and is_lower)
```

Estos valores no determinan la sonrisa directamente, pero permite verificar que las dos curvaturas se encuentran dentro de un rango compatible con una activación facial suave y simétrica, condición típica en microsonrisas. Este rango se eligió en base a los resultados obtenidos en las gráficas de la Figuras 6 y 7. Además su interpretación corresponde a que valores demasiado pequeños se correlacionan a expresiones completamente neutras, sin elevación del arco labial, mientras que valores demasiado grandes aparecen en sonrisas amplias o en movimientos exagerados, que exceden la sutil activación propia de una microexpresión. El coeficiente **a_coefficient** sintetiza esta condición, exigiendo que ambos labios estén activados pero sin alcanzar niveles altos.

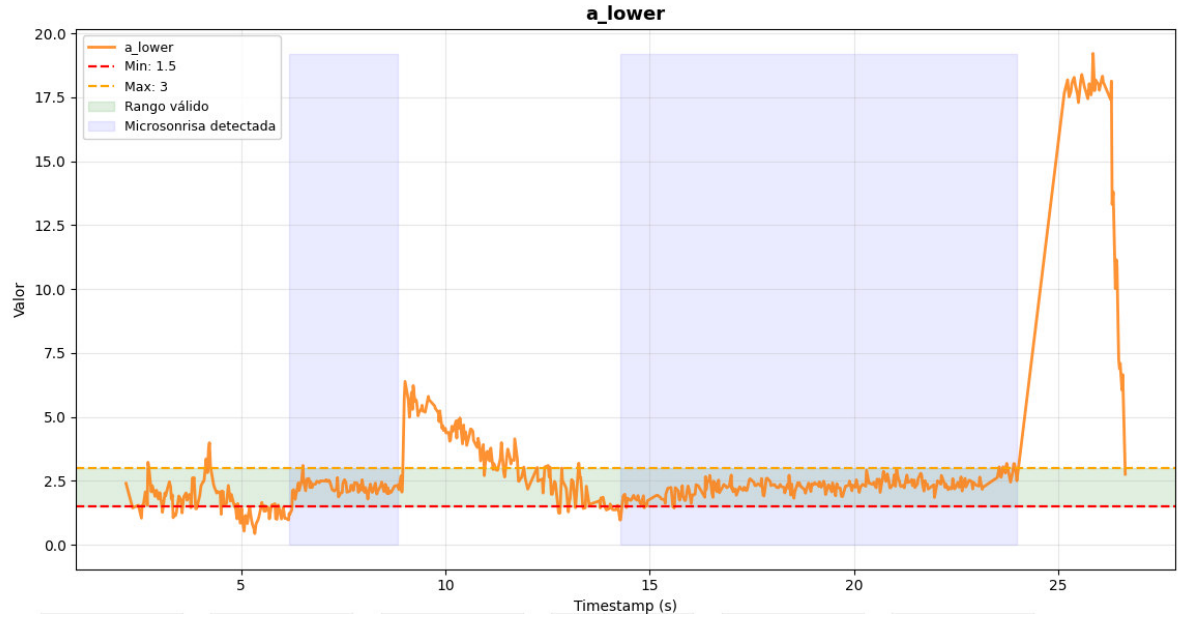

Figura 6. Gráfica de los valores de la métrica a_lower durante una sesión de etiquetado manual. Fuente: produccion propia a partir de matplotlib.

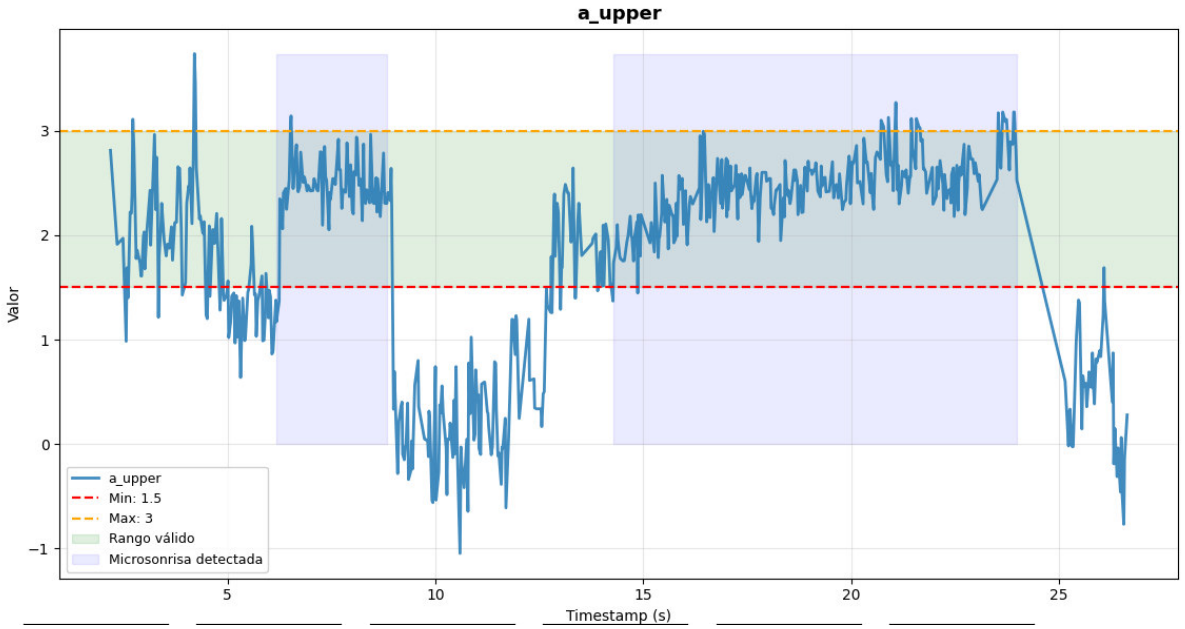

Figura 7. Gráfica de los valores de la métrica a_upper durante una sesión de etiquetado manual. Fuente: producción propia a partir de matplotlib.

El siguiente paso consiste en analizar la media entre las métricas **a_upper** y **a_lower** (métrica smile_strength). Valores altos de smile_strength sugieren una sonrisa amplia o clara, mientras que valores muy bajos indican neutralidad. El proyecto define como zona óptima para microsonrisa un rango estrecho entre 1 y 3, como se representa en la ecuación siguiente:

```
is_smile_strength = 1 < smile_strength < 3
```

En la siguiente figura, se pueden observar estos límites, destacando que la activación muscular es suficientemente evidente para ser registrada, pero no lo suficientemente intensa como para corresponder a una sonrisa completa.

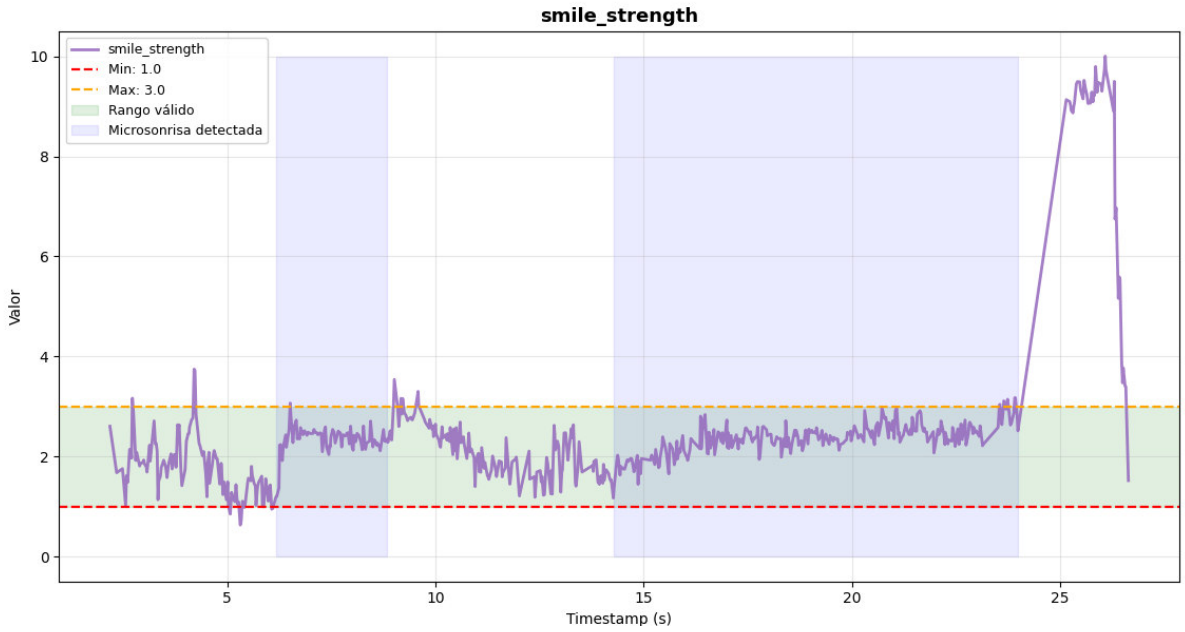

Figura 8. Gráfica de los valores de la métrica smile_strength durante una sesión de etiquetado manual. Fuente: producción propia a partir de matplotlib.

El sistema complementa las métricas de curvatura y activación labial con un conjunto de razones geométricas destinadas a validar la coherencia espacial del gesto. La microsonrisa, al ser una expresión sutil, requiere no sólo una elevación moderada de los labios, sino también una configuración equilibrada de distancias

verticales y horizontales. Por ello, se analiza el ratio del labio inferior, que debe ubicarse entre 1.05 y 1.4, cuyos límites se obtuvieron de la Figura 9.

```
is_lower_lip_dist = 1.05 < ratio_lower_lip  <  1.4
```

Este rango señala una elevación leve pero perceptible, es decir, valores menores corresponden a neutralidad absoluta, mientras que valores mayores representan movimientos demasiado amplios, incompatibles con la naturaleza discreta de una microexpresión.

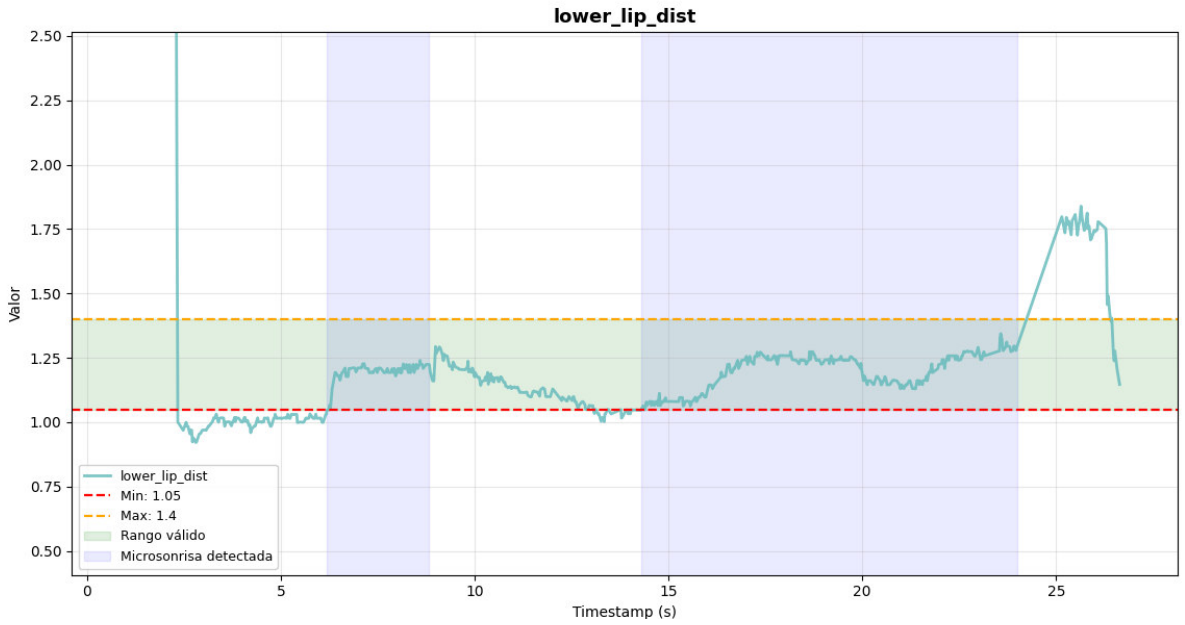

Figura 9. Gráfica de los valores de la métrica lower_lip_dist durante una sesión de etiquetado manual. Fuente: producción propia a partir de matplotlib.

De forma similar, el sistema controla el ensanchamiento lateral mediante el ratio de anchura de la boca. También aquí se utiliza un rango de 1.05 a 1.4 como criterio para asegurar una expansión moderada, como se puede observar en la Figura 10.

```
is_not_mouth_width = 1.05 < ratio_mouth_width < 1.4
```

Una microsonrisa auténtica implica una ligera tracción hacia los extremos, pero no una sonrisa franca o voluntaria. Si la boca permanece prácticamente inmóvil, el gesto no puede clasificarse como microsonrisa; si se ensancha demasiado, se aleja del patrón de baja intensidad característico.

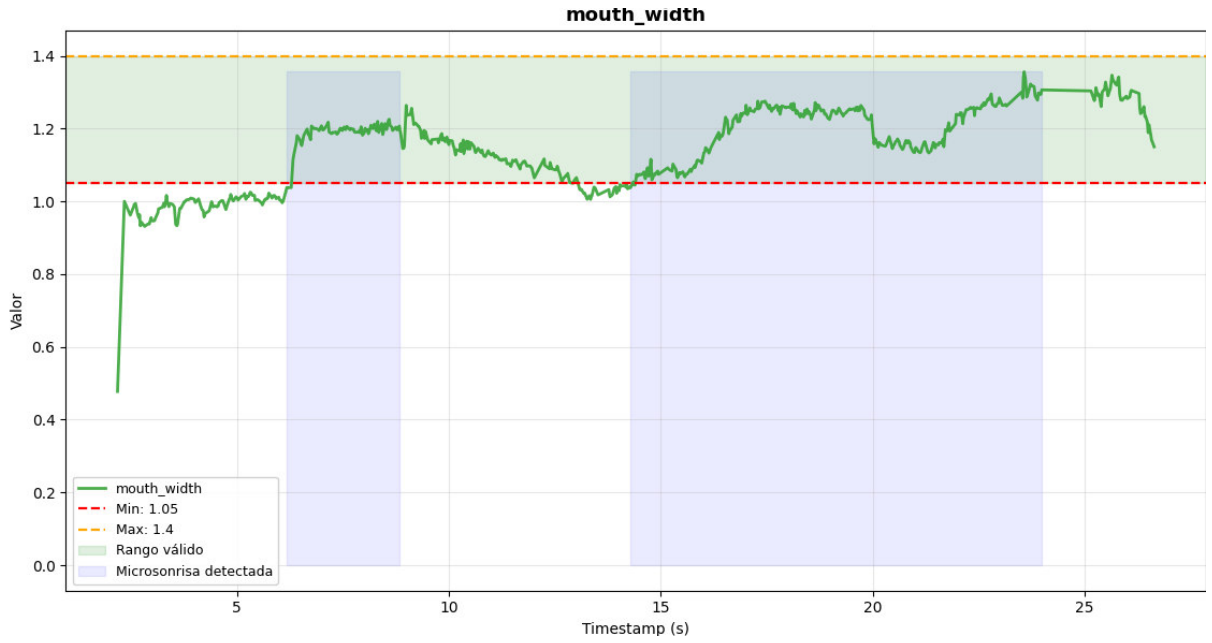

Figura 10. Gráfica de los valores de la métrica mouth_width durante una sesión de etiquetado manual. Fuente: producción propia a partir de matplotlib.

Por último, se evalúa la apertura vertical, cuya importancia radica en su capacidad para descartar sonrisas amplias o movimientos asociados al habla. La condición ratio_mouth_open < 1.5, garantiza que la separación entre los labios permanezca dentro de límites compatibles con un gesto tenue, dado que una apertura exagerada invalidaría la clasificación, incluso si las curvaturas y los ensanchamientos cumplen los criterios anteriores.

```
is_not_mouth_open = ratio_mouth_open < 2
```

En la Figura 11 se pueden observar los límites mencionados anteriormente.

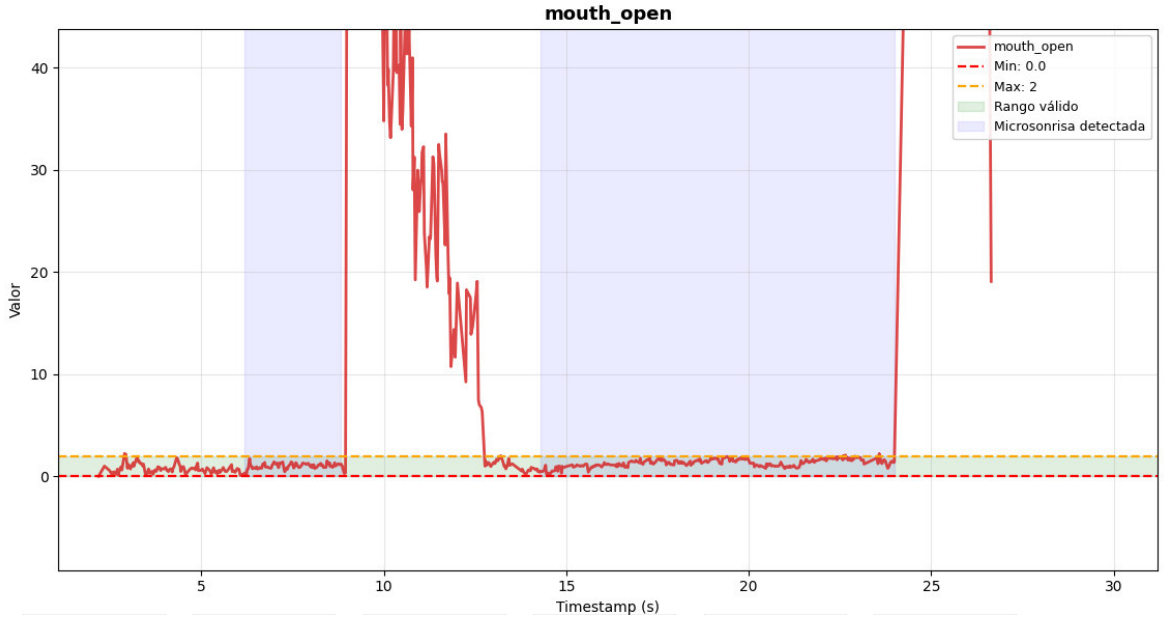

Figura 11. Gráfica de los valores de la métrica mouth_open durante una sesión de etiquetado manual. Fuente: producción propia a partir de matplotlib.

En conjunto, estas tres métricas funcionan como filtros de consistencia que aseguran que el movimiento observado conserve la delicadeza propia de una microsonrisa. La condición final queda definida en la siguiente expresión booleana:

```
detected_auto_micro = a_coefficient and is_lower_lip_dist and
is_not_mouth_open and is_not_mouth_width
```

Este criterio combina coherencia geométrica, intensidad moderada y baja apertura vertical. En términos interpretativos:

- Curvatura muy baja implica neutralidad.
- Curvatura media junto con ensanchamiento moderado y baja apertura vertical produce el patrón más fiable de microsonrisa.

- Curvatura alta acompañada de una apertura grande corresponde a sonrisas plenas, risas o gesticulaciones amplias, por lo que se excluye explícitamente del rango definido.

El resultado es un detector sensible a variaciones sutiles y capaz de diferenciar entre expresiones neutrales, microsonrisas genuinas y sonrisas completas, integrando de forma armónica la geometría facial con criterios fisiológicos basados en la literatura de microexpresiones.

4.1.2 Construcción del conjuntos de datos con etiquetado manual

El etiquetado manual permite forzar la asignación de la clase microsonrisa mediante una acción del usuario, anulando la decisión del sistema automático. En el código, cuando la opción manual_microsmile está activada, el valor de is_micro_smile se define exclusivamente a partir de micro_manual, independientemente de lo que haya detectado el algoritmo. En cambio, si el etiquetado manual no está habilitado, la variable is_micro_smile toma el valor proporcionado por la detección automática, tomando el valor de detected_auto_micro. Este mecanismo se utiliza para asegurar un etiquetado controlado y confiable, especialmente útil para la generación de datos de referencia, la corrección de errores de detección automática y la validación de modelos de reconocimiento de microexpresiones.

```
# Aplicar configuración de teclas:
          # - manual_microsmile (1): si True, fuerza microsonrisa manual
          # - micro_manual (2): si False, desactiva detección automática
          if manual_microsmile:
              is_micro_smile = micro_manual

          else:
              is_micro_smile = detected_auto_micro
```

4.2 Desarrollo del modelo de clasificación binaria

Se implementa un clasificador Random Forest debido a su buen rendimiento en datos tabulares, su robustez frente a ruido y su baja demanda computacional. El ajuste de hiperparámetros se realiza mediante GridSearchCV, optimizando parámetros como profundidad de árboles, número de estimadores, función criterio y tamaño mínimo de nodos. El archivo model.py constituye uno de los elementos centrales en la arquitectura del proyecto, ya que implementa el pipeline completo de entrenamiento, optimización y validación del modelo encargado de clasificar microsonrisas a partir de datos derivados de landmarks faciales. Este módulo concentra tanto los aspectos metodológicos del aprendizaje supervisado como las decisiones técnicas que permiten garantizar la reproducibilidad, el desempeño y la integridad matemática del clasificador. En las secciones siguientes se detalla el funcionamiento interno del código principal model.py, su fundamentación teórica y las decisiones metodológicas que guiaron su diseño. El propósito fundamental de model.py es realizar un entrenamiento riguroso sobre un dataset previamente generado por el módulo de captura y análisis facial (main.py). Este dataset contiene características geométricas del rostro, como curvaturas labiales, aperturas normalizadas, distancias relativas y métricas derivadas del movimiento sutil de los labios.

La estructura del código abarca cinco funciones principales:

1. Carga y preparación del dataset
2. Balanceo y muestreo estratificado de datos
3. Entrenamiento del modelo base (Random Forest)
4. Búsqueda exhaustiva de hiperparámetros (GridSearchCV)
5. Evaluación, generación de métricas y guardado del modelo final

La combinación de estas etapas constituye un pipeline profesional de aprendizaje automático aplicable en entornos reales.

```
df = pd.read_csv("./Smile-Debug/smile_debug_data_combined.csv")

size = 4000
```

```
# Filtrar por clase
smiles = df[df["is_micro_smile"] == 1]
no_smiles = df[df["is_micro_smile"] == 0]


# Calcular el máximo posible sin superar "size"
available_smiles = len(smiles)
available_no_smiles = len(no_smiles)
min_count = min(available_smiles, available_no_smiles, size)
```

El balanceo de clases implementado en model.py representa uno de los componentes clave del diseño del modelo. En tareas de clasificación binaria, especialmente cuando una clase es naturalmente más frecuente, la falta de balanceo puede conducir a modelos sesgados que se sesgan hacia la clase más común.

En el contexto de detección de microsonrisas, esto es especialmente crítico porque:

- los frames con microsonrisas verdaderas son una minoría del total,
- los frames sin microsonrisa (neutros) son la mayoría

El codigo model.py evita este problema mediante muestreo proporcional:

```
# Tomar muestras balanceadas
smiles_balanced = smiles.sample(n=min_count, random_state=42)
no_smiles_balanced = no_smiles.sample(n=min_count,
random_state=42)

# Combinar y mezclar
df_balanced = pd.concat([smiles_balanced,
no_smiles_balanced]).sample(frac=1, random_state=42)

X = df_balanced.drop(columns=["timestamp", "frame",
"is_micro_smile"])
y = df_balanced["is_micro_smile"]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)
```

El modelo Random Forest se beneficia enormemente de un dataset balanceado porque:

1. evita que los árboles se especialicen en clasificar mayorías,
2. fomenta una estructura de división más equilibrada,
3. aumenta el *recall* de la clase minoritaria,
4. reduce el sesgo inherente de los criterios de impureza (como Gini).

Una vez realizado el balanceo de clases y la selección de las características relevantes, el conjunto de datos resultante se divide en subconjuntos de entrenamiento y prueba. En particular, se aplica una partición estratificada utilizando un 80 % de los datos para entrenamiento y un 20 % para evaluación, preservando la proporción de clases en ambos subconjuntos. Esta división se implementa mediante la función train_test_split de scikit-learn [31], con un valor de random_state fijo para garantizar la reproducibilidad de los experimentos.

El modelo Random Forest utilizado se basa en un conjunto de árboles de decisión entrenados de manera independiente, cuya combinación permite obtener un clasificador robusto, estable frente al ruido y con capacidad para modelar patrones complejos presentes en los datos geométricos derivados de las microsonrisas. Para optimizar su desempeño, el script aplica una búsqueda exhaustiva de hiperparámetros mediante *Grid Search*, evaluando diferentes configuraciones del modelo con el fin de identificar aquella que maximice maximiza la exactitud y estabilidad del clasificador.

Los hiperparámetros incluidos en el param_grid cumplen funciones complementarias que influyen directamente en la capacidad del modelo para generalizar correctamente sin caer en sobreajuste o subajuste. A continuación, se describen y justifican en detalle, cada uno de estos parámetros [34].

```
rf = RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2'],
    'bootstrap': [True, False],
    'criterion': ['gini', 'entropy'],
}
```

1. n_estimators

El hiperparámetro n_estimators determina el número total de árboles que integran el bosque aleatorio. Dado que el Random Forest es un método basado en ensamble, un mayor número de árboles tiende a reducir la varianza del modelo al promediar múltiples decisiones independientes. Valores mayores como 200 o 300 árboles incrementan la estabilidad del clasificador frente a valores atípicos o ruido inherente a los landmarks faciales, aunque también aumentan el costo computacional. La inclusión de tres valores (100, 200 y 300) permite evaluar si el incremento en la cantidad de árboles ofrece mejoras significativas o si aparece un punto de rendimiento decreciente [34].

2. max_depth

El parámetro max_depth controla la profundidad máxima permitida para cada árbol individual. Una profundidad ilimitada (None) permite que el árbol crezca hasta clasificar perfectamente los datos de entrenamiento, lo cual genera un riesgo elevado de sobreajuste. En cambio, profundidades moderadas (10, 20 o 30) limitan la complejidad del árbol, fomentando reglas más generales y menos sensibles a variaciones pequeñas en los landmarks faciales. La exploración de estos valores permite identificar hasta qué punto la complejidad estructural del árbol es necesaria para capturar los patrones sutiles que caracterizan las microsonrisas [34].

3. min_samples_split

El hiperparámetro min_samples_split determina cuántas muestras debe tener un nodo antes de que el algoritmo intenta dividirlo. Valores bajos (por ejemplo, 2) generan árboles muy detallados, capaces de capturar microvariaciones específicas del dataset, mientras que valores más altos (5 o 10) promueven divisiones más robustas basadas en grupos más amplios de datos. En el contexto de señales faciales, donde existe fluctuación natural entre frames, aumentar este parámetro puede evitar divisiones espurias generadas por ruido, mejorando la capacidad de generalización del modelo [34].

4. min_samples_leaf

El parámetro min_samples_leaf fija la cantidad mínima de muestras que puede contener una hoja. Una hoja con un solo ejemplo puede resultar extremadamente sensible al ruido, especialmente en datasets basados en medidas geométricas derivadas de landmarks, donde pequeñas variaciones pueden inducir reglas de decisión artificialmente específicas. Valores mayores (como 2 o 4) obligan al modelo a considerar patrones más amplios, reduciendo la probabilidad de sobreajuste y favoreciendo una clasificación más estable en escenarios reales [34].

5. max_features

El parámetro max_features especifica cuántas características se consideran al evaluar cada división. Las opciones sqrt y log2 son estrategias estándar en Random Forest que buscan introducir aleatoriedad adicional en el proceso de construcción de los árboles. Esta aleatoriedad disminuye la correlación entre árboles individuales, lo que mejora la capacidad del ensamble para capturar patrones alternativos. Dado que las características derivadas de landmarks faciales suelen estar correlacionadas entre sí (por ejemplo, la apertura normalizada y la fuerza de la sonrisa), limitar el número de características por división contribuye a un ensamble más diverso y robusto [34].

6. bootstrap

El parámetro bootstrap controla si cada árbol se entrena con un subconjunto de datos elegido mediante muestreo con reemplazo. Cuando es True, cada árbol recibe una muestra ligeramente diferente del conjunto total de datos, lo cual es esencial para maximizar la diversidad del ensamble. La inclusión de False permite evaluar si, en un dataset pequeño o moderado, entrenar cada árbol con la totalidad de los datos puede mejorar la capacidad de aprendizaje. En el caso de este proyecto, probar ambos enfoques resulta fundamental debido a la naturaleza sutil y altamente específica de las microexpresiones faciales [34].

7. criterion

El parámetro criterion define la función utilizada para medir la calidad de una división del árbol.
Se incluyen dos opciones:

- **Gini**, que mide la impureza basándose en la probabilidad de clasificar incorrectamente un ejemplo si se etiquetara al azar según la distribución,
- **Entropía**, basada en la teoría de la información y que penaliza más fuertemente las distribuciones uniformes.

La entropía tiende a producir árboles más equilibrados y profundos, mientras que Gini es computacionalmente más eficiente y suele generar resultados similares. Dado que las microsonrisas son patrones extremadamente sutiles, evaluar ambos criterios permite identificar cuál de los dos produce divisiones más informativas en ese contexto [34].

El proceso de optimización de hiperparámetros constituye una fase crítica en cualquier proyecto de aprendizaje supervisado. Esta tarea se realiza mediante la clase GridSearchCV de Scikit-Learn. El objetivo central de esta técnica es explorar sistemáticamente diferentes combinaciones de hiperparámetros con el fin de identificar aquella configuración que maximiza el rendimiento del modelo [34].

A continuación, se presenta un análisis detallado de cada uno de los parámetros incluidos en la instancia:

```
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=3,               # validación cruzada con 3 particiones
    n_jobs=-1,          # usa todos los núcleos del CPU
    verbose=2,          # muestra progreso
    scoring='accuracy'  # métrica a optimizar
)
```

1. estimator

El parámetro estimator define el modelo base sobre el cual se realizará la búsqueda de hiperparámetros. En el contexto de GridSearchCV, este parámetro resulta fundamental ya que permite desacoplar la definición del modelo del proceso de optimización, separando claramente la lógica de aprendizaje del mecanismo de ajuste de parámetros. Esta separación metodológica facilita la reutilización del mismo esquema de búsqueda para distintos algoritmos de clasificación, mejora la modularidad del código y contribuye a la reproducibilidad del experimento, ya que el procedimiento de optimización permanece constante independientemente del modelo utilizado. [31]

2. param_grid

El parámetro param_grid contiene el conjunto de hiperparámetros junto con los valores que se desean evaluar para cada uno de ellos. Este componente define explícitamente el espacio de búsqueda que será explorado durante el proceso de optimización. Cada combinación posible dentro de la cuadrícula es evaluada de manera sistemática mediante validación cruzada, lo que garantiza una comparación homogénea entre modelos.

El diseño del grid constituye un aspecto metodológico clave, ya que determina el equilibrio entre la exhaustividad de la búsqueda, el costo computacional asociado y

la validez estadística de los resultados obtenidos. Un grid demasiado amplio puede resultar computacionalmente inviable, mientras que uno demasiado acotado puede impedir la identificación de configuraciones óptimas. En el marco de este proyecto, el grid se orienta a explorar en profundidad la complejidad del modelo, la profundidad de los árboles, los mecanismos de regularización implícita, la estrategia de muestreo de los datos y las funciones de impureza utilizadas durante la construcción de los árboles. Un diseño adecuado del espacio de búsqueda incrementa la probabilidad de obtener un clasificador estable, óptimo y reproducible. [31]

3. cv (cross-validation)

El parámetro cv=3 indica que el proceso de optimización se llevará a cabo utilizando validación cruzada con tres particiones, siguiendo el esquema de k-fold cross-validation. En este procedimiento, el conjunto de datos de entrenamiento se divide en tres subconjuntos. Para cada combinación de hiperparámetros, el modelo se entrena utilizando dos particiones y se evalúa sobre la restante. Este proceso se repite tres veces, alternando la partición de evaluación, y finalmente se calcula el promedio del desempeño obtenido. [31]

4. n_jobs

El parámetro n_jobs=-1 indica que se utilizarán todos los núcleos disponibles del CPU para ejecutar la búsqueda de hiperparámetros en paralelo. La optimización mediante Grid Search es inherentemente intensiva desde el punto de vista computacional, ya que cada combinación del grid implica entrenar y evaluar un modelo, y este proceso se repite tantas veces como particiones se definan en la validación cruzada. [31]

5. verbose

El parámetro verbose=2 controla el nivel de detalle de la información mostrada durante la ejecución del proceso de búsqueda. Si bien este parámetro no tiene impacto directo sobre el rendimiento del modelo entrenado, sí desempeña un rol importante en la trazabilidad y el monitoreo del experimento. [31]

6. scoring

En este caso, se utiliza **scoring='accuracy'**, lo que indica que la métrica empleada para comparar las distintas combinaciones de hiperparámetros será la exactitud. La Accuracy mide el porcentaje total de predicciones correctas realizadas por el modelo sobre el conjunto de evaluación y constituye una métrica ampliamente utilizada por su simplicidad e interpretación directa.

La elección de la métrica de evaluación es un aspecto crítico del proceso de optimización, ya que determina el criterio según el cual se seleccionará el mejor modelo. La Accuracy resulta apropiada cuando las clases se encuentran relativamente balanceadas y cuando el objetivo principal es maximizar el número total de aciertos. En el contexto de este proyecto, esta métrica permite evaluar de manera clara el desempeño global del clasificador en la detección de microsonrisas. [31]

Una vez terminada la fase de encontrar el modelo que mejor se ajusta se observan los siguientes gráficos tanto para la fase de entrenamiento como de prueba. Las matrices de confusión, representadas en las Figuras 12 y 13, tanto para los conjuntos de entrenamiento como de prueba presentan un desempeño perfecto del modelo. En ambos casos, las instancias de la clase *Micro Sonrisa* y *No Micro Sonrisa* fueron clasificadas correctamente sin presencia de falsos positivos ni falsos negativos. La matriz de confusión correspondiente al conjunto de entrenamiento (Figura 13) indica que las 3200 instancias de cada clase fueron etiquetadas correctamente. Esta situación refleja que el modelo ha aprendido a distinguir perfectamente los patrones definidos por las características seleccionadas dentro del conjunto de entrenamiento.

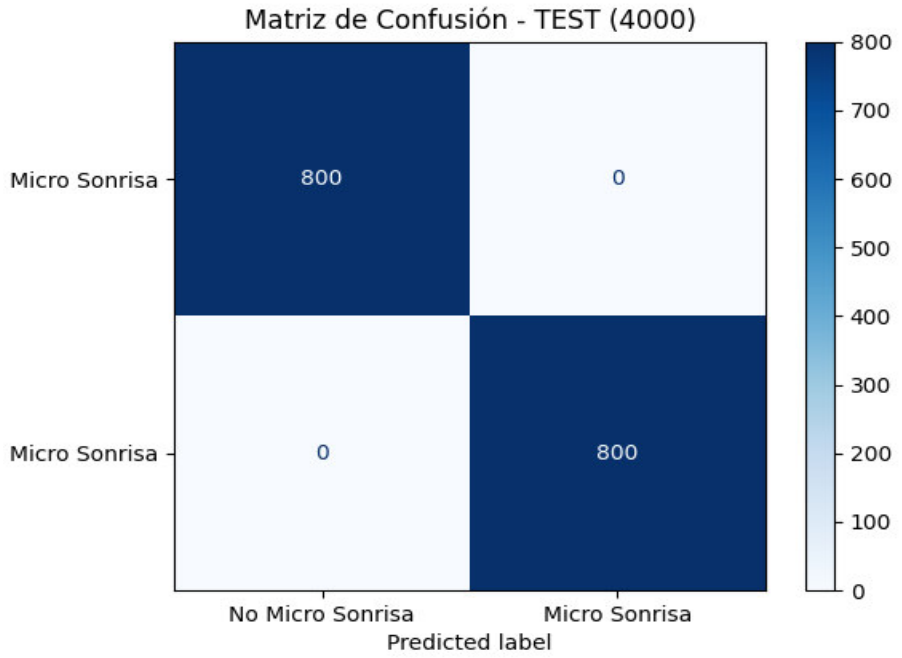

Figura 12. Matriz de confusión obtenida de la fase Test en el entrenamiento. Fuente: producción propia a partir de scikit-learn.

De forma análoga, la matriz de confusión del conjunto de pruebas, representada en la Figura 12, muestra que las 800 instancias de cada clase se clasificaron con total acierto. La ausencia de errores de clasificación en el conjunto de prueba sugiere una generalización sobresaliente del modelo, dado que no hubo degradación del desempeño fuera del conjunto de entrenamiento.

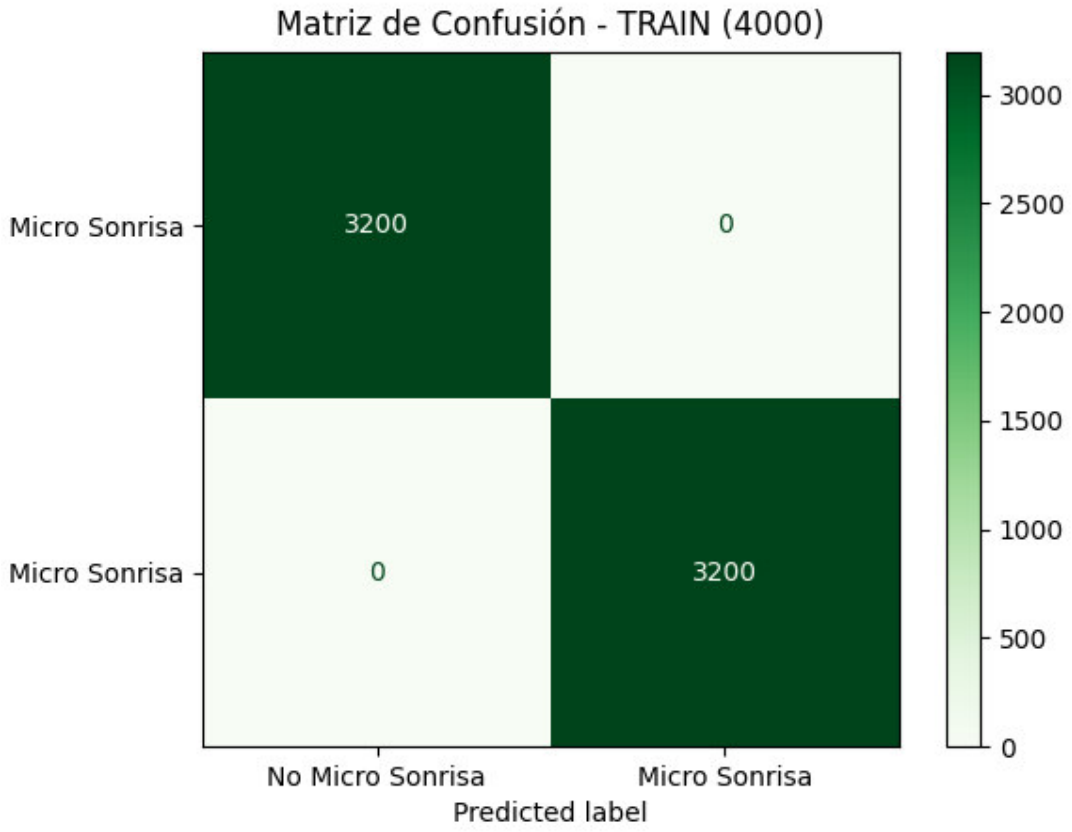

Figura 13. Matriz de confusión obtenida de la fase Train en el entrenamiento. Fuente: producción propia a partir de scikit-learn.

Los resultados obtenidos en las Figuras 14 y 15 muestran que el modelo alcanza valores de Accuracy, Precision, Recall y F1-Score iguales a 1.00, tanto en el conjunto de entrenamiento como en el conjunto de prueba. Este desempeño indica que el clasificador logró una separación perfecta entre las clases *Micro Sonrisa* y *No Micro Sonrisa* dentro del escenario experimental evaluado.

La Accuracy de 1.00 refleja que la totalidad de las instancias fueron correctamente clasificadas. Sin embargo, la literatura advierte que esta métrica, por sí sola, puede resultar insuficiente para evaluar clasificadores binarios, especialmente en presencia de distribuciones desbalanceadas o cuando los costos de error son asimétricos.

La Precision igual a 1.00 indica la ausencia total de falsos positivos, es decir, el modelo no etiquetó erróneamente muestras sin microsonrisa como positivas. De manera complementaria, el Recall también alcanza el valor máximo, lo que implica

que el modelo identificó correctamente todas las microsonrisas sin incurrir en falsos negativos.

La métrica F1-Score, al combinar Precision y Recall mediante un promedio armónico, confirma la consistencia del clasificador al obtener igualmente un valor de 1.00. Este indicador resulta particularmente adecuado para evaluar modelos cuando se requiere un balance entre ambos tipos de error, lo cual es fundamental en aplicaciones de análisis facial automático.

Finalmente, la coincidencia perfecta de estas métricas en entrenamiento y prueba sugiere que el modelo no presenta sobreajuste bajo las condiciones evaluadas, lo que indica una adecuada capacidad de generalización. Este comportamiento es coherente con las propiedades del algoritmo Random Forest, reconocido por su estabilidad y robustez frente al ruido en los datos.

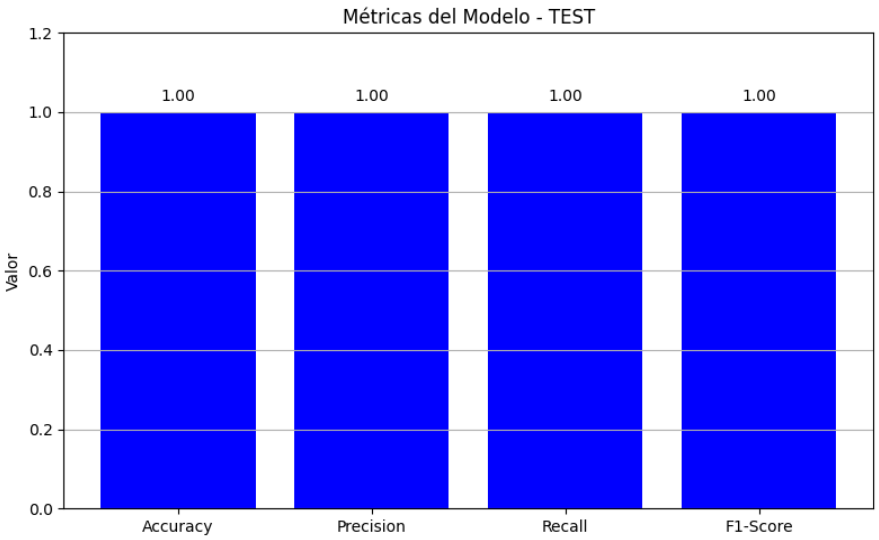

Figura 14. Métricas del modelo en la etapa de Test en el entrenamiento. Fuente: producción propia a partir de matplotlib.

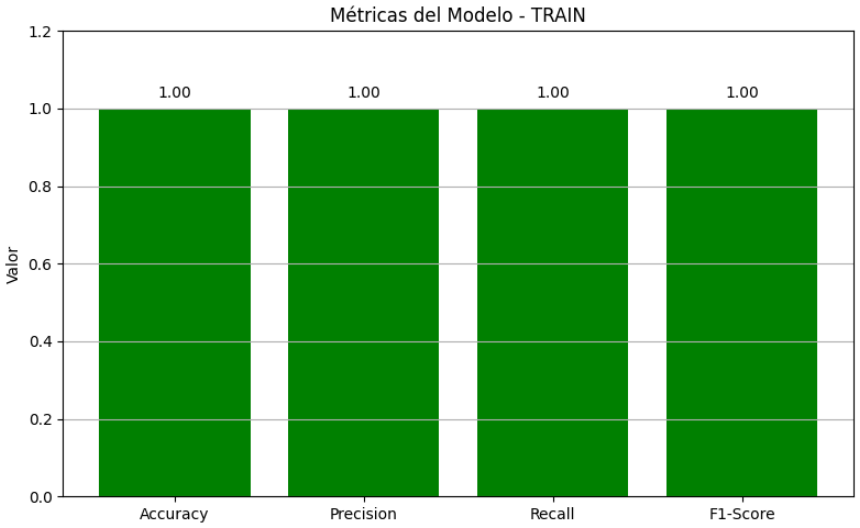

Figura 15. Métricas del modelo en la etapa Train en el entrenamiento. *Fuente: producción propia a partir de matplotlib.*

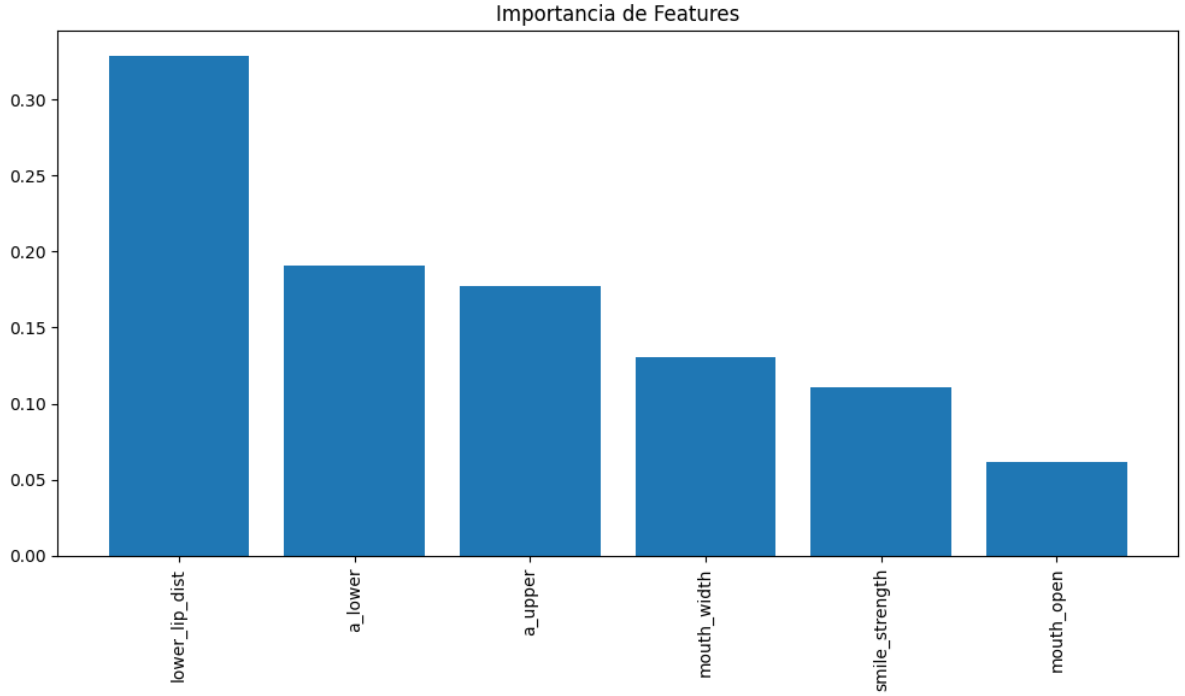

Figura 16. Importancia de las características en la etapa de entrenamiento. Fuente: producción propia a partir de matplotlib.

5. Interpretación de resultados

Luego de la etapa de entrenamiento y validación del modelo, y considerando los resultados altamente satisfactorios obtenidos durante dicha fase, se decidió avanzar hacia una etapa de inferencia, cuyo objetivo principal fue analizar el comportamiento del clasificador en un contexto más cercano a un escenario de aplicación real. Esta instancia resulta fundamental para evaluar la capacidad de generalización del modelo más allá de los datos utilizados durante el proceso de optimización.

En una primera instancia, la inferencia se llevó a cabo sobre la totalidad del conjunto de datos de etiquetado automático. Tal como se observa en la Figura 17, la matriz de confusión correspondiente a esta etapa evidencia un desempeño notablemente positivo. En particular, no se registran falsos positivos, lo que indica que el modelo no clasifica erróneamente instancias sin microsonrisa como microsonrisas. Asimismo, se observa únicamente la presencia de algunos falsos negativos, lo que sugiere que, en un número reducido de casos, el modelo no detecta microsonrisas presentes en los datos.

De manera complementaria, en la Figura 18 se presentan las métricas de evaluación obtenidas durante esta etapa de inferencia. Si bien los valores de Accuracy, Precision, Recall y F1-Score ya no alcanzan el valor máximo de 1.00, se mantienen muy próximos a dicho umbral, lo que confirma un alto nivel de desempeño del modelo bajo condiciones menos controladas que las del entrenamiento y prueba. Esta ligera disminución resulta esperable en escenarios de inferencia realista y no invalida la robustez del clasificador.

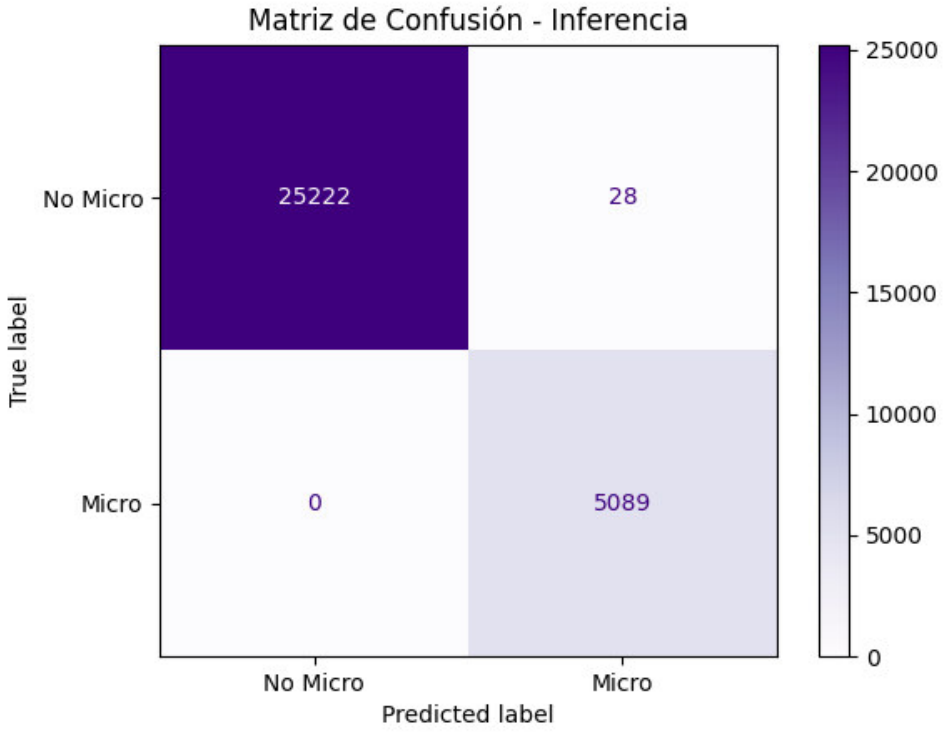

Figura 17. Matriz de confusión en la etapa de inferencia. *Fuente: producción propia a partir de scikit-learn.*

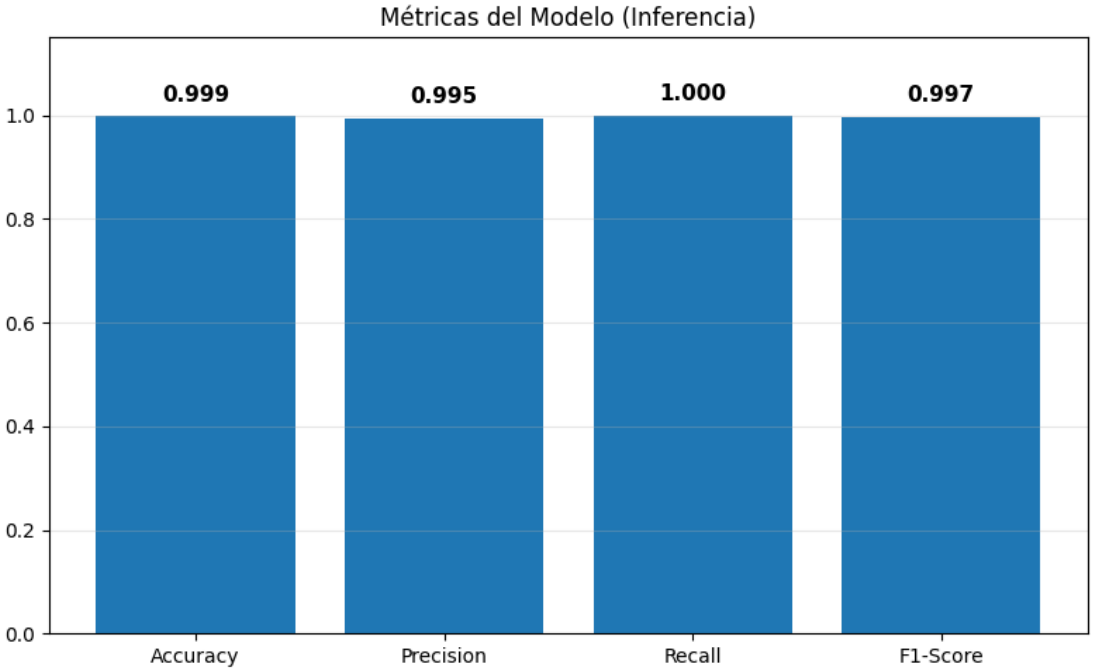

Figura 18. Métricas del modelo en la etapa de inferencia. Fuente: producción propia a partir de matplotlib.

A partir de los resultados favorables obtenidos sobre el conjunto de datos de etiquetado automático, se decidió realizar una segunda etapa de inferencia utilizando un conjunto de datos de etiquetado manual, con el objetivo de analizar el comportamiento del modelo frente a un conjunto alternativo y contrastar la validez del etiquetado automático frente a anotaciones más cercanas a la interpretación humana.

Los resultados de esta evaluación se presentan en las Figuras 19 y 20. En la Figura 19 se observa que las métricas obtenidas son inferiores a las alcanzadas sobre el conjunto de etiquetado automático, aunque continúan reflejando un desempeño global positivo. Esta reducción en la precisión del modelo puede atribuirse a la mayor variabilidad y subjetividad inherente al etiquetado manual, así como a posibles discrepancias entre los criterios geométricos utilizados durante el entrenamiento y la interpretación humana de las microsonrisas.

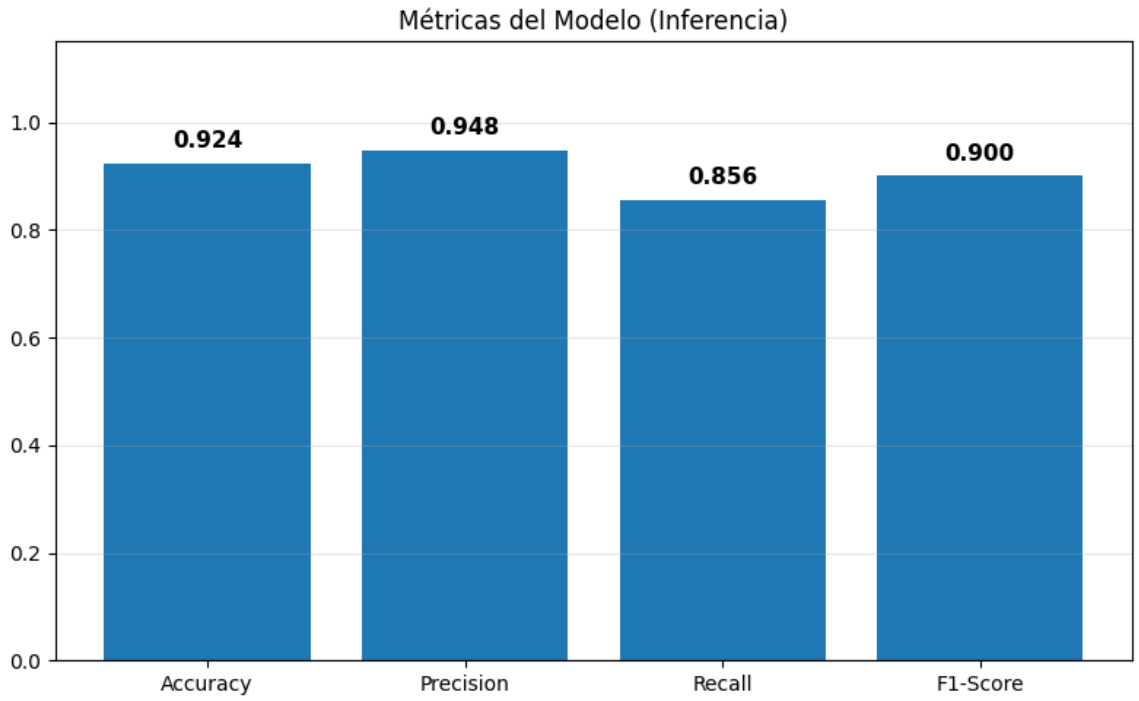

Figura 19. Métricas del modelo en la etapa de inferencia obtenidas con el conjunto de datos alternativo. *Fuente: producción propia a partir de matplotlib.*

Finalmente, la Figura 20 muestra la matriz de confusión correspondiente a esta etapa de inferencia, evidenciando que, si bien se incrementa la cantidad de errores de clasificación, el modelo conserva una capacidad significativa para distinguir entre las clases evaluadas. En conjunto, estos resultados permiten sustentar la validez del conjunto de datos de etiquetado automático, demostrando que las características geométricas utilizadas poseen un fundamento consistente y capturan información relevante del fenómeno estudiado.

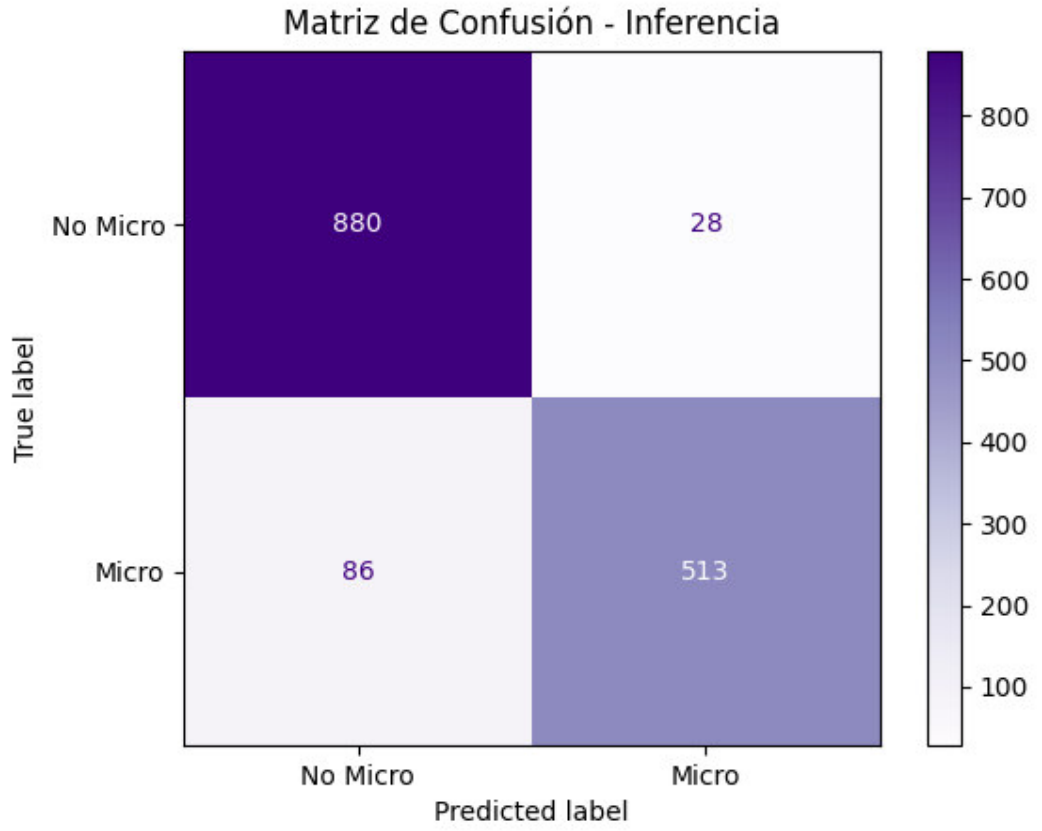

Figura 20. Resultados del modelo en la etapa de inferencia obtenidas con el conjunto de datos alternativo. Fuente: producción propia a partir de scikit-learn.

En consecuencia, la etapa de inferencia no solo confirma la robustez del modelo entrenado, sino que también respalda la metodología de generación automática de etiquetas como una aproximación viable y coherente para el análisis de microsonrisas, aportando una base sólida para futuras extensiones y aplicaciones del sistema propuesto.

6. Conclusiones

El desarrollo del presente proyecto permitió cumplir de manera satisfactoria los objetivos planteados, abordando tanto los fundamentos teóricos como los aspectos prácticos necesarios para la implementación de un sistema automático de detección de microsonrisas. En primer lugar, se realizó un estudio exhaustivo de los principios teóricos vinculados al reconocimiento de microexpresiones faciales, con especial énfasis en la identificación y caracterización de patrones gestuales sutiles, lo cual permitió establecer una base conceptual sólida para el diseño del sistema propuesto. En este marco, se evaluaron arquitecturas de procesamiento de imágenes y video considerando su capacidad para extraer características gestuales relevantes, así como su viabilidad de implementación bajo restricciones computacionales. Esta evaluación condujo a la adopción de un enfoque híbrido, que combina extracción de características geométricas con modelos de aprendizaje automático clásico, permitiendo un equilibrio adecuado entre precisión, interpretabilidad y eficiencia.

Uno de los aportes centrales del proyecto fue la construcción de un conjunto de datos etiquetados de imágenes faciales, compuesto por ejemplos representativos de microsonrisas y no microsonrisas. Este dataset, generado mediante un proceso de etiquetado automático y contrastado posteriormente con anotaciones manuales, demostró ser adecuado para el entrenamiento y la validación del modelo de clasificación, aportando consistencia y sustento empírico a la metodología propuesta.

A nivel de implementación, se diseñó, entrenó y validó un pipeline completo de clasificación, que abarca desde la captura y procesamiento de los datos hasta la predicción final. Dicho pipeline incluye la extracción de embeddings y métricas geométricas, el entrenamiento supervisado del clasificador, el ajuste sistemático de hiperparametros mediante validación cruzada y la evaluación del rendimiento a través de métricas estándar. La correcta integración de estas etapas permitió la construcción de una prueba de concepto funcional, capaz de detectar microsonrisas con un desempeño competitivo.

En particular, se implementó un módulo de extracción de landmarks faciales utilizando MediaPipe Face Mesh, lo que permitió capturar de forma precisa los cambios geométricos cuadro a cuadro en la región facial. Complementariamente, se integró DeepFace para tareas de reconocimiento emocional y comparación de embeddings, enriqueciendo la representación facial y aportando información adicional al proceso de clasificación. A partir de estos datos, se calcularon métricas geométricas relevantes (como aperturas y variaciones musculares faciales) que demostraron ser altamente útiles en la detección de microsonrisas.

El modelo de clasificación basado en Random Forest, entrenado sobre las características extraídas, mostró un desempeño sobresaliente durante las etapas de entrenamiento y prueba, y mantuvo resultados sólidos durante la fase de inferencia. La evaluación del sistema mediante métricas como Accuracy, Precision, Recall y F1-score permitió verificar no solo la exactitud global del modelo, sino también su equilibrio frente a los distintos tipos de error. Los resultados obtenidos en la inferencia sobre conjuntos de datos de etiquetado manual refuerzan la validez del enfoque propuesto y la coherencia del etiquetado automático. No obstante, el análisis de resultados también permitió identificar limitaciones inherentes al sistema, principalmente relacionadas con las condiciones de adquisición de los datos. El correcto funcionamiento del modelo se ve influenciado por factores como el nivel de iluminación, la distancia entre el sujeto y la cámara y la posición y orientación del rostro. Estas restricciones impactan directamente en la precisión de la detección de landmarks y, en consecuencia, en la calidad de las características geométricas extraídas. Reconocer estas limitaciones resulta fundamental para contextualizar los resultados y delimitar el alcance del sistema desarrollado.

En conjunto, el proyecto confirma la viabilidad técnica de un sistema ligero de detección de microsonrisas, capaz de operar bajo restricciones de recursos y con un alto grado de interpretabilidad. Los resultados obtenidos, junto con la identificación de fuentes de error y oportunidades de mejora, sientan una base sólida para futuras implementaciones más robustas y generalizables, orientadas a escenarios reales y aplicaciones avanzadas de análisis facial automático.

Como primera línea de trabajo futuro, se propone la expansión del enfoque hacia la detección y clasificación de otros gestos y microexpresiones faciales cómo se han estudiado en diferentes estudios relacionados [23] [24], tales como cejas elevadas, contracciones orbiculares, asimetrías labiales o movimientos sutiles de la región periocular. La generalización del pipeline desarrollado permitiría reutilizar la metodología de extracción geométrica y el esquema de clasificación para abarcar un conjunto más amplio de gestos faciales, avanzando hacia un sistema multimodal de análisis gestual.

En segundo lugar, se plantea la ampliación y diversificación del conjunto de datos, incorporando mayor número de sujetos, variabilidad en condiciones de iluminación, distintas distancias a la cámara y múltiples orientaciones del rostro. Esta ampliación resulta fundamental para mejorar la capacidad de generalización del modelo y reducir su dependencia de escenarios controlados.

Adicionalmente, se considera relevante la incorporación de modelos con componente temporal, como redes LSTM o arquitecturas basadas en convoluciones tridimensionales [22], que permitan modelar la evolución dinámica de los gestos faciales a lo largo del tiempo. Esta extensión resulta especialmente adecuada para el análisis de microexpresiones, cuya principal característica es su brevedad y variación temporal.

Finalmente, como paso orientado a la validación práctica del sistema, se propone el desarrollo de una interfaz gráfica de usuario (GUI) que permita visualizar en tiempo real la detección de landmarks, las métricas geométricas calculadas y la predicción del modelo. Esta interfaz facilitaría la interacción con el sistema, la evaluación cualitativa de los resultados y la demostración de la prueba de concepto, además de constituir una base para futuras aplicaciones en contextos de análisis emocional, investigación académica o interacción humano-computadora.

7. Bibliografía

[1] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar e I. Matthews, "The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression," en *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, 2010, pp. 94-101. doi: 10.1109/CVPRW.2010.5543262.

[2] R. A. Calvo y S. D'Mello, "Affect Detection: An Interdisciplinary Review of Models, Methods, and Their Applications," *IEEE Trans. Affect. Comput.*, vol. 1, no. 1, pp. 18-37, ene.-jun. 2010. doi: 10.1109/T-AFFC.2010.1.

[3] X. Li, T. Pfister, X. Huang, G. Zhao y M. Pietikäinen, "A spontaneous micro-expression database: Inducement, collection and baseline," en *Proc. IEEE Int. Conf. Workshops Autom. Face Gesture Recognit. (FG)*, 2013, pp. 1-6. doi: 10.1109/FG.2013.6553743.

[4] X. Ben, Y. Ren, J. Zhang, S.-J. Wang, K. Kpalma y W. Meng, "Video-based facial micro-expression analysis: A survey of datasets, features and algorithms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5826-5846, sept. 2022. doi: 10.1109/TPAMI.2021.3067464.

[5] Y. Zong, X. Huang, W. Zheng, Z. Cui y G. Zhao, "Learning from hierarchical spatiotemporal descriptors for micro-expression recognition," *IEEE Trans. Multimedia*, vol. 20, no. 11, pp. 3160-3172, nov. 2018. doi: 10.1109/TMM.2018.2837882.

[6] J. M. Girard, J. F. Cohn, M. H. Mahoor, S. M. Mavadati y D. P. Rosenwald, "Social risk and depression: Evidence from manual and automatic facial expression analysis," en *Proc. IEEE Int. Conf. Autom. Face Gesture Recognit. (FG)*, 2013, pp. 1-8. doi: 10.1109/FG.2013.6553749.

[7] M. Pantic y L. J. M. Rothkrantz, "Toward an affect-sensitive multimodal human-computer interaction," *Proc. IEEE*, vol. 91, no. 9, pp. 1370-1390, sept. 2003. doi: 10.1109/JPROC.2003.817122.

[8] S. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper

Saddle River, NJ, USA: Prentice Hall, 2010.

[9] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[10] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.

[11] P. Ekman y E. Rosenberg, *What the Face Reveals*. Oxford, U.K.: Oxford Univ. Press, 2005.

[12] R. O. Duda, P. E. Hart y D. G. Stork, *Pattern Classification*, 2nd ed. New York, NY, USA: Wiley-Interscience, 2001.

[13] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, pp. 5–32, 2001. doi: 10.1023/A:1010933404324.

[14] G. James, D. Witten, T. Hastie y R. Tibshirani, *An Introduction to Statistical Learning*. New York, NY, USA: Springer, 2013.

[15] P. Geurts, D. Ernst y L. Wehenkel, “Extremely randomized trees,” *Mach. Learn.*, vol. 63, pp. 3–42, 2006. doi: 10.1007/s10994-006-6226-1.

[16] A. Liaw y M. Wiener, “Classification and regression by randomForest,” *R News*, vol. 2, no. 3, pp. 18–22, 2002.

[17] F. Lugaresi *et al.*, “MediaPipe: A framework for building perception pipelines,” *arXiv preprint arXiv:1906.08172*, 2019.

[18] I. Grishchenko *et al.*, “Attention Mesh: High-Fidelity Facial Landmark Tracking Using Attention Mechanisms,” Google Research, 2020.

[19] J. Bazarevsky *et al.*, “Real-time facial surface geometry from monocular video on mobile GPUs,” Google AI, 2019.

[20] Q. Cao, L. Shen, W. Xie, O. M. Parkhi y A. Zisserman, “VGGFace2: A dataset for recognising faces across pose and age,” en *Proc. IEEE Int. Conf. Autom. Face Gesture Recognit.*, 2018, pp. 67–74. doi: 10.1109/FG.2018.00020.

[21] S. Taigman, M. Yang, M. Ranzato y L. Wolf, “DeepFace: Closing the gap to human-level performance in face verification,” en *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2014, pp. 1701–1708. doi: 10.1109/CVPR.2014.220.

[22] O. Ali Hassen, D. Farhan, Y. Y. Gromov, K. Sheoran, y G. Dhand, “Computer vision of smile detection based on machine and deep learning approach,” *J. Cybersecurity Inf. Manag.*, vol. 16, no. 1, pp. 208–230, 2025. doi: 10.54216/JCIM.160115.

[23] I. Kotsia y I. Pitas, “Facial expression recognition in image sequences using geometric deformation features,” IEEE Trans. Image Process., vol. 16, no. 1, pp. 172–187, 2007. doi: 10.1109/TIP.2006.884954.

[24] T. Pfister, X. Li, G. Zhao, y M. Pietikäinen, “Recognising spontaneous facial micro-expressions,” IEEE Trans. Affective Comput., vol. 7, no. 1, pp. 1–12, 2016. doi: 10.1109/TAFFC.2015.2487369.

[25] B. Mandal y N. Ouarti, “Spontaneous vs. posed smiles - can we tell the difference?,” arXiv preprint arXiv:1605.07026, 2016.

[26] Q. Gan, S. Nie, S. Wang, y Q. Ji, “Differentiating between posed and spontaneous expressions with latent regression Bayesian network,” Proc. AAAI Conf. Artif. Intell., vol. 31, no. 1, pp. 4039–4046, 2017. doi: 10.1609/aaai.v31i1.11064.

[27] Python Software Foundation, “Python Documentation,” *Python.org*, 2024. Disponible en: https://www.python.org/doc/

[28] OpenCV Team, “OpenCV Documentation,” *OpenCV.org*, 2024. Disponible en: https://docs.opencv.org/

[29] Google AI Research, “MediaPipe Documentation,” *Google Developers*, 2024. Disponible en: https://developers.google.com/mediapipe

[30] S. Serengil, “DeepFace Library Documentation,” *GitHub Repository*, 2024. Disponible en: https://github.com/serengil/deepface

[31] Scikit-Learn Developers, “Scikit-Learn User Guide,” *Scikit-learn.org*, 2024. Disponible en: https://scikit-learn.org/stable/

[32] Matplotlib Development Team, “Matplotlib Documentation,” *Matplotlib.org*, 2024. Disponible en: https://matplotlib.org/stable/

[33] Joblib Developers, “Joblib Documentation,” *ReadTheDocs*, 2024. Disponible en: https://joblib.readthedocs.io/en/latest/

[34] Zhu N, Zhu C, Zhou L, Zhu Y, Zhang X. Optimization of the Random Forest Hyperparameters for Power Industrial Control Systems Intrusion Detection Using an Improved Grid Search Algorithm. Applied Sciences. 2022; 12(20):10456. https://doi.org/10.3390/app122010456

8. Anexos

A continuación se presenta el código del proyecto “main.py”

```
import cv2
import mediapipe as mp
import numpy as np
import math
import pandas as pd
import time
import hashlib
from datetime import datetime
from deepface import DeepFace
import os

mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(
    static_image_mode=False,
    max_num_faces=1,
    refine_landmarks=True,
    min_detection_confidence=0.7,
    min_tracking_confidence=0.7
)

# --- Puntos clave ---
UPPER_CURVE = [191, 80, 81, 82, 13, 312, 311, 310, 415]
LOWER_CURVE = [95, 88, 178, 87, 14, 317, 402, 318, 324]

LEFT_EYE_CENTER = 33
RIGHT_EYE_CENTER = 263

MOUTH_LEFT = 78
MOUTH_RIGHT = 308

UPPER_CENTER = 13
LOWER_CENTER = 14


# --- Funciones auxiliares ---
def curvature(points):
    if len(points) < 3:
        return 0.0
```

```
    xs = np.array([p[0] for p in points], dtype=float)
    ys = np.array([p[1] for p in points], dtype=float)
    xs -= np.mean(xs)
    ys -= np.mean(ys)
    coeffs = np.polyfit(xs, -ys, 2)
    return coeffs[0] * 1000

def mean_pairwise_distance(points):
    if len(points) < 2:
        return 0.0
    dists = [math.dist(points[i], points[i + 1]) for i in
range(len(points) - 1)]
    return np.mean(dists)


# --- Fuente de video ---
cap = cv2.VideoCapture(0)
timestamp_str = datetime.now().strftime("%Y%m%d_%H%M%S")
hash_id = hashlib.md5(timestamp_str.encode()).hexdigest()[:8]
os.makedirs(f"./Frames/{hash_id}", exist_ok=True)
os.makedirs("./Frames", exist_ok=True)
os.makedirs("./Videos", exist_ok=True)
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
fps = int(cap.get(cv2.CAP_PROP_FPS)) or 30
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)) or 640
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)) or 480
out = cv2.VideoWriter(f'./Videos/{hash_id}.mp4', fourcc, fps,
(width, height))

# --- Variables base ---
records = []
manual_microsmile = True        # toggled with key '1' (configura
si hay microsonrisa manual)
micro_manual = False            # toggled with key '2'
(activa/desactiva detección automática)
is_calibrated = False
left_eye_base = right_eye_base = mouth_width_base = lower_lip_base
= mouth_open_base = 1

start_time = time.time()
while cap.isOpened():
    ret, frame = cap.read()
```

```
    if not ret:
        break

    current_time = time.time() - start_time
    h, w, _ = frame.shape
    frame = cv2.flip(frame, 1)
    out.write(frame) # <--- guarda el frame en el video
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = face_mesh.process(rgb)

    is_micro_smile = False

    if results.multi_face_landmarks:
        for face_landmarks in results.multi_face_landmarks:
            # --- Ojos ---
            left_eye_center =
face_landmarks.landmark[LEFT_EYE_CENTER]
            right_eye_center =
face_landmarks.landmark[RIGHT_EYE_CENTER]
            eye_distance = math.dist(
                (left_eye_center.x * w, left_eye_center.y * h),
                (right_eye_center.x * w, right_eye_center.y * h)
            )

            # --- Distancia horizontal entre labios (apertura) ---
            left_mouth = face_landmarks.landmark[MOUTH_LEFT]
            right_mouth = face_landmarks.landmark[MOUTH_RIGHT]
            p_left = (left_mouth.x * w, left_mouth.y * h)
            p_right = (right_mouth.x * w, right_mouth.y * h)
            mouth_width = math.dist(p_left, p_right)

            # --- Distancia vertical entre labios (apertura) ---
            upper_lip = face_landmarks.landmark[UPPER_CENTER]
            lower_lip = face_landmarks.landmark[LOWER_CENTER]
            upper_center = (upper_lip.x * w, upper_lip.y * h)
            lower_center = (lower_lip.x * w, lower_lip.y * h)
            mouth_open = math.dist(upper_center, lower_center)

            # --- Curvaturas globales ---
            pts_upper = [(int(face_landmarks.landmark[i].x * w),
                          int(face_landmarks.landmark[i].y * h))
for i in UPPER_CURVE]
```

```
            pts_lower = [(int(face_landmarks.landmark[i].x * w),
                          int(face_landmarks.landmark[i].y * h))
for i in LOWER_CURVE]

            a_upper = curvature(pts_upper)
            a_lower = curvature(pts_lower)
            lower_lip_dist = mean_pairwise_distance(pts_lower)

            # --- Normalización ---
            normalized_mouth_width = mouth_width / eye_distance
            normalized_mouth_open = mouth_open / eye_distance

            # --- Calibración ---
            if not is_calibrated:
                try:
                    analysis = DeepFace.analyze(frame,
actions=['emotion'], enforce_detection=False)
                    dominant_emotion =
analysis[0]['dominant_emotion']
                except Exception:
                    dominant_emotion = "neutral"
                if dominant_emotion == "neutral":
                    mouth_width_base = normalized_mouth_width
                    mouth_open_base = normalized_mouth_open
                    lower_lip_base = lower_lip_dist
                    is_calibrated = True

            # --- Razones ---
            ratio_mouth_width = normalized_mouth_width /
mouth_width_base
            ratio_mouth_open = normalized_mouth_open /
mouth_open_base
            ratio_lower_lip = lower_lip_dist / lower_lip_base

            smile_strength = (abs(a_upper) + abs(a_lower)) / 2

            # --- Sonrisa global ---
            is_upper = 1.5 < a_upper < 3
            is_lower = 1.5 < a_lower < 3
            a_coefficient = (is_upper and is_lower)
            is_smile_strength = 1 < smile_strength < 3
            is_lower_lip_dist = 1.05 < ratio_lower_lip  <  1.4
```

```
            is_not_mouth_width = 1.05 < ratio_mouth_width < 1.4
            is_not_mouth_open = ratio_mouth_open < 2

            detected_auto_micro =  (
                a_coefficient and
                is_smile_strength and
                is_lower_lip_dist and
                is_not_mouth_width and
                is_not_mouth_open
            )
            # Aplicar configuración de teclas:
            # - manual_microsmile (1): si True, fuerza
microsonrisa manual
            # - micro_manual (2): si False, desactiva detección
automática
            if manual_microsmile:
                is_micro_smile = micro_manual

            else:
                is_micro_smile = detected_auto_micro

            # --- Guardar frame ---
            records.append({
                "timestamp": current_time,
                "frame": len(records),
                "is_micro_smile": is_micro_smile,
                "a_upper": a_upper,
                "a_lower": a_lower,
                "mouth_width": ratio_mouth_width,
                "smile_strength": smile_strength,
                "mouth_open": ratio_mouth_open,
                "lower_lip_dist" : ratio_lower_lip
            })

            # --- Mostrar datos --- (estado manual y automático)
            y0, dy = 40, 22
            values = [
                f"Manual (tecla 1): {'ON' if manual_microsmile
else 'OFF'}",
                f"Auto micro (tecla 2): {'ENABLED' if micro_manual
else 'DISABLED'}",
                f"Micro sonriendo (actual): {is_micro_smile}"
```

```
            ]

            metrics = [
                f"Manual: {'ON' if manual_microsmile else 'OFF'}",
                f"micro (tecla 2): {'ENABLED' if micro_manual else
'DISABLED'}",
                f"a_upper : {a_upper:.6f}",
                f"a_lower : {a_lower:.6f}",
                f"lip_dist : {ratio_lower_lip:.6f}",
                f"mouth_width: {ratio_mouth_width:.6f}",
                f"mouth_open: {ratio_mouth_open:.6f}",
                f"smile_strength: {smile_strength:.6f}"
            ]

            for i, text in enumerate(metrics):
                cv2.putText(frame, text, (30, y0 + i * dy),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0,
255, 0) if is_micro_smile else (0, 0, 255), 1)

            # Color de puntos
            color = (0, 255, 0) if is_micro_smile else (0, 0, 255)
            for p in pts_upper + pts_lower:
                cv2.circle(frame, p, 1, color, -1)


cv2.imwrite(f"./Frames/{hash_id}/frame_{len(records):04d}.jpg",
frame)
    cv2.imshow(f"Smile {hash_id}", frame)
    key = cv2.waitKey(1) & 0xFF
    if key == 27:  # ESC
        break
    elif key == ord('2'):
        # activa/desactiva detección automática de microsonrisas
        micro_manual = not micro_manual

cap.release()
out.release()
cv2.destroyAllWindows()

# Exportar CSV
df = pd.DataFrame(records)
df.to_csv(f"./Smile-Debug/smile_debug_{'manual' if
```

```
manual_microsmile else 'auto'}_{hash_id}.csv", index=False)
print(f"✅ {len(df)} frames procesados y guardados en
Smile-Debug/smile_debug_data_{hash_id}.csv")
```

A continuación se presenta el código del modelado “model.py”

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay,
    accuracy_score, precision_score, recall_score, f1_score
)
import matplotlib.pyplot as plt
import asyncio
import time
import numpy as np

model_type = "Manual" #"Manual" a "Auto"

def graphs(best_rf, X_train, X_test, y_train, y_test, size,
graph_path):

    # 1) Predicciones
    y_pred_train = best_rf.predict(X_train)
    y_pred_test = best_rf.predict(X_test)
    y_scores = best_rf.predict_proba(X_test)[:, 1]

    # 2) Matriz de Confusión TRAIN
    cm_train = confusion_matrix(y_train, y_pred_train)
    disp_train = ConfusionMatrixDisplay(cm_train,
display_labels=["No Micro Sonrisa", "Micro Sonrisa"])
    disp_train.plot(cmap="Greens", values_format="d")
    plt.title(f"Matriz de Confusión - TRAIN ({size})")
    plt.savefig(f"{graph_path}/matriz_confusion_train_{size}.png")
    plt.close()

    # 3) Matriz de Confusión TEST
```

```
    cm_test = confusion_matrix(y_test, y_pred_test)
    disp_test = ConfusionMatrixDisplay(cm_test,
display_labels=["No Micro Sonrisa", "Micro Sonrisa"])
    disp_test.plot(cmap="Blues", values_format="d")
    plt.title(f"Matriz de Confusión - TEST ({size})")
    plt.savefig(f"{graph_path}/matriz_confusion_test_{size}.png")
    plt.close()


    # 4) Importancia de Features
    importances = best_rf.feature_importances_
    indices = np.argsort(importances)[::-1]
    features = X_train.columns

    plt.figure(figsize=(10, 6))
    plt.bar(range(len(importances)), importances[indices])
    plt.xticks(range(len(importances)), features[indices],
rotation=90)
    plt.title("Importancia de Features")
    plt.tight_layout()
    plt.savefig(f"{graph_path}/feature_importance_{size}.png")
    plt.close()

    # 5A) Gráfico de barras de métricas TRAIN
    metrics_train = {
        "Accuracy": accuracy_score(y_train, y_pred_train),
        "Precision": precision_score(y_train, y_pred_train),
        "Recall": recall_score(y_train, y_pred_train),
        "F1-Score": f1_score(y_train, y_pred_train)
    }

    plt.figure(figsize=(8, 5))
    bars_train = plt.bar(metrics_train.keys(),
metrics_train.values(), color='green')

    for bar in bars_train:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2, height + 0.02,
f"{height:.2f}",
                 ha='center', va='bottom', fontsize=10)

    plt.ylim(0, 1.2)
```

```
    plt.title("Métricas del Modelo - TRAIN")
    plt.ylabel("Valor")
    plt.grid(axis='y')
    plt.tight_layout()
    plt.savefig(f"{graph_path}/metricas_train_{size}.png")
    plt.close()


    # 5B) Gráfico de barras de métricas TEST
    metrics_test = {
        "Accuracy": accuracy_score(y_test, y_pred_test),
        "Precision": precision_score(y_test, y_pred_test),
        "Recall": recall_score(y_test, y_pred_test),
        "F1-Score": f1_score(y_test, y_pred_test)
    }

    plt.figure(figsize=(8, 5))
    bars_test = plt.bar(metrics_test.keys(),
metrics_test.values(), color='blue')

    # Agregar valores numéricos arriba de cada barra
    for bar in bars_test:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2, height + 0.02,
f"{height:.2f}",
                 ha='center', va='bottom', fontsize=10)

    plt.ylim(0, 1.2)
    plt.title("Métricas del Modelo - TEST")
    plt.ylabel("Valor")
    plt.grid(axis='y')
    plt.tight_layout()
    plt.savefig(f"{graph_path}/metricas_test_{size}.png")
    plt.close()



df = pd.read_csv("./Smile-Debug/smile_debug_manual_combined.csv")

size = 4000

# Filtrar por clase
```

```
smiles = df[df["is_micro_smile"] == 1]
no_smiles = df[df["is_micro_smile"] == 0]

# Calcular el máximo posible sin superar "size"
available_smiles = len(smiles)
available_no_smiles = len(no_smiles)
min_count = min(available_smiles, available_no_smiles, size)
size = min_count

graph_path = f"./Graphs/{model_type}/Size {size}"
os.makedirs(graph_path, exist_ok=True)

# Tomar muestras balanceadas
smiles_balanced = smiles.sample(n=min_count, random_state=42)
no_smiles_balanced = no_smiles.sample(n=min_count,
random_state=42)

# Combinar y mezclar
df_balanced = pd.concat([smiles_balanced,
no_smiles_balanced]).sample(frac=1, random_state=42)

print(f"📊 Usando {min_count} muestras por clase (total {min_count
* 2})")


X = df_balanced.drop(columns=["timestamp", "frame",
"is_micro_smile"])
y = df_balanced["is_micro_smile"]


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)


rf = RandomForestClassifier(random_state=42)


param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
```

```
    'max_features': ['sqrt', 'log2'],
    'bootstrap': [True, False],
    'criterion': ['gini', 'entropy'],
}


grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=3,                # validación cruzada con 3 particiones
    n_jobs=-1,           # usa todos los núcleos del CPU
    verbose=2,           # muestra progreso
    scoring='accuracy'  # métrica a optimizar
)



inittime = time.time()
grid_search.fit(X_train, y_train)

endtime = time.time()

best_rf = grid_search.best_estimator_


graphs(best_rf, X_train, X_test, y_train, y_test, size,
graph_path)


import joblib
joblib.dump(best_rf,
f"./Models/model_rf_{model_type.lower()}_{size}.pkl")
```