**UNIVERSIDAD TECNOLÓGICA BOLIVIANA**

**INGENIERÍA DE SISTEMAS**



**Desarrollo de un sistema de reconocimiento facial en tiempo real utilizando Python y una interfaz gráfica para la gestión de capturas y comparaciones**

**Estudiantes:**

* Fabio Camacho Encinas
* Juan Carlos Marca Alvarez
* Fátima Lucia Rocha Rospigliossi

**Materia:** Inteligencia Artificial

**LA PAZ – BOLIVIA**

**2024**

[1. Introducción 1](#_Toc184065364)

[2. Planteamiento del problema 2](#_Toc184065365)

[3. Objetivo general 2](#_Toc184065366)

[4. Objetivos específicos 2](#_Toc184065367)

[5. Alcance 2](#_Toc184065368)

[6. Límites 3](#_Toc184065369)

[7. Marco teórico 3](#_Toc184065370)

[8. Desarrollo 3](#_Toc184065371)

[9. Recomendaciones 3](#_Toc184065372)

[10. Conclusiones 3](#_Toc184065373)

Desarrollo de un sistema de reconocimiento facial en tiempo real utilizando Python y una interfaz gráfica para la gestión de capturas y comparaciones

# Introducción

El reconocimiento facial es una tecnología que ha evolucionado significativamente en las últimas décadas, encontrando aplicaciones en diversas áreas, como la seguridad, la atención personalizada, la autenticación biométrica y el entretenimiento. En esencia, esta tecnología permite identificar o verificar la identidad de una persona analizando sus características faciales, lo que la convierte en una herramienta poderosa y versátil.

El desarrollo de sistemas de reconocimiento facial, aunque antes era exclusivo de instituciones con acceso a tecnología avanzada, ahora está al alcance de desarrolladores gracias a bibliotecas y frameworks accesibles como OpenCV, Mediapipe y herramientas de desarrollo en Python. Estas herramientas ofrecen una forma eficiente de implementar algoritmos avanzados de detección y procesamiento de imágenes, incluso en dispositivos con recursos limitados.

Este proyecto aborda la implementación de un sistema básico pero funcional de reconocimiento facial en tiempo real. El sistema incluye la detección de rostros, la captura de imágenes, el almacenamiento de estas para comparación y el reconocimiento en tiempo real mediante una cámara web. Además, se desarrolla una interfaz gráfica utilizando Tkinter, lo que permite a los usuarios interactuar con el sistema de forma sencilla, sin necesidad de conocimientos técnicos avanzados.

El objetivo principal de este sistema es proporcionar una solución accesible para aquellos que deseen explorar el reconocimiento facial con fines educativos, de prueba o como base para proyectos más complejos. A diferencia de las plataformas comerciales o de alta gama, este sistema se centra en la simplicidad, ofreciendo una experiencia introductoria que puede servir como punto de partida para desarrolladores y estudiantes interesados en esta área.

Asimismo, el proyecto busca destacar las posibilidades que ofrecen las tecnologías de código abierto en la democratización de herramientas avanzadas. Al emplear bibliotecas ampliamente disponibles y una estructura de desarrollo modular, se sientan las bases para futuras extensiones, como la integración con algoritmos de aprendizaje profundo o la creación de bases de datos para múltiples usuarios.

En resumen, este proyecto no solo tiene el propósito de demostrar la viabilidad del reconocimiento facial utilizando herramientas accesibles, sino también de fomentar la curiosidad y la experimentación en el campo de la visión por computadora.

# Planteamiento del problema

En muchas aplicaciones que requieren reconocimiento facial, los sistemas disponibles suelen ser complejos, costosos o inaccesibles para usuarios con recursos limitados. Además, muchas soluciones no cuentan con interfaces simples que permitan a usuarios no técnicos interactuar fácilmente con la herramienta. Esto plantea la necesidad de desarrollar un sistema que sea práctico, accesible y funcional para propósitos educativos, experimentales o de pequeñas aplicaciones personales o empresariales.

# Objetivo general

Desarrollar un sistema de reconocimiento facial en tiempo real con Python, que permita capturar, guardar y comparar imágenes faciales, utilizando una interfaz gráfica intuitiva para facilitar su uso.

# Objetivos específicos

* Implementar la detección facial en tiempo real utilizando la biblioteca Mediapipe para garantizar precisión y eficiencia.
* Desarrollar una interfaz gráfica con Tkinter que permita controlar las funcionalidades del sistema de manera intuitiva.
* Incorporar la funcionalidad de guardar capturas de imágenes para comparaciones futuras.
* Implementar un método para calcular y visualizar el porcentaje de similitud entre imágenes capturadas.
* Permitir el reconocimiento facial en tiempo real con alertas cuando se detecte una coincidencia satisfactoria.

# Alcance

* El sistema será capaz de detectar rostros en tiempo real mediante una cámara web.
* Permitirá guardar imágenes capturadas y compararlas con imágenes en tiempo real para calcular el nivel de similitud.
* La interfaz gráfica facilitará la interacción con el sistema, permitiendo controlar las funciones principales como encender/apagar la cámara, guardar capturas y realizar comparaciones.
* Se diseñará para aplicaciones educativas, experimentales o de demostración, no para entornos de seguridad críticos.

# Límites

* El sistema está limitado a una cámara web como dispositivo de captura, no incluye integración con otras fuentes de imágenes o videos.
* No se incorpora manejo de bases de datos para almacenar múltiples registros faciales.
* Solo está diseñado para funcionar en plataformas compatibles con Python y las bibliotecas utilizadas.

# Marco teórico

# Visión artificial

La visión artificial o visión por computadora es un campo de la inteligencia artificial que se encarga de desarrollar sistemas capaces de interpretar y comprender imágenes y videos para realizar tareas que, hasta ahora, solo los humanos podían llevar a cabo con su visión. La visión artificial permite que las computadoras "vean" y "entiendan" el mundo visual que las rodea, utilizando datos visuales provenientes de imágenes o secuencias de video. Esta disciplina abarca un conjunto de métodos y tecnologías que buscan emular o incluso superar las capacidades visuales humanas en ciertas tareas específicas, como la clasificación, detección y segmentación de objetos.

El objetivo principal de la visión artificial es permitir que las máquinas puedan interpretar y extraer información significativa de las imágenes, realizar tareas como el reconocimiento de patrones y tomar decisiones en función de esos datos visuales. En este marco, se combinan conocimientos de diferentes áreas, como la computación, el procesamiento de señales, la teoría de imágenes y el aprendizaje automático.

# Conceptos Básicos en Visión Artificial

Para comprender la visión artificial, es necesario familiarizarse con algunos conceptos clave:

* **Imagen Digital**: Una imagen digital es una representación bidimensional de información visual compuesta por píxeles, que son unidades discretas de información visual (cada píxel tiene un valor de color o intensidad).
* **Procesamiento de Imágenes**: Consiste en aplicar una serie de técnicas matemáticas y computacionales para transformar, analizar y extraer información útil de una imagen digital. Algunos procesos comunes incluyen el filtrado, la mejora de contrastes, la detección de bordes y la segmentación.
* **Segmentación**: La segmentación de una imagen es el proceso de dividir una imagen en regiones o partes de interés. En visión artificial, se busca separar los objetos dentro de una escena para facilitar su análisis.
* **Características Visuales**: Las características visuales son atributos importantes de una imagen que permiten identificar patrones o objetos. Algunas características comunes incluyen el color, la textura, la forma y el tamaño.
* **Reconocimiento de Patrones**: Es el proceso mediante el cual un sistema de visión artificial identifica patrones o estructuras en las imágenes para clasificarlas, reconocer objetos, personas o acciones específicas.
* **Modelos de Aprendizaje Automático**: En la visión artificial moderna, se utilizan modelos de machine learning (aprendizaje automático) y Deep learning (aprendizaje profundo), particularmente redes neuronales convolucionales (CNNs), para enseñar a las máquinas a reconocer patrones y realizar tareas complejas basadas en grandes volúmenes de datos visuales.

# Componentes de un Sistema de Visión Artificia

Un sistema de visión artificial consta de varios componentes fundamentales que interactúan entre sí para lograr una interpretación efectiva de las imágenes:

* **Captura de Imágenes**: El primer paso en cualquier sistema de visión artificial es la adquisición de imágenes. Esto se realiza mediante cámaras o sensores que convierten la información visual en datos digitales que pueden ser procesados por el sistema.
* **Preprocesamiento de Imágenes**: Las imágenes adquiridas suelen estar afectadas por ruido, distorsiones o malas condiciones de iluminación. El preprocesamiento mejora la calidad de las imágenes mediante técnicas como el filtrado, la corrección de color y la normalización.
* **Segmentación de Imágenes**: Una vez preprocesada la imagen, se realiza la segmentación, es decir, el proceso de dividir la imagen en regiones más pequeñas y manejables. Esto facilita la identificación de objetos o áreas de interés, como el fondo y los elementos que se desean analizar.
* **Extracción de Características**: La extracción de características es el proceso de identificar elementos clave dentro de una imagen (bordes, texturas, formas, etc.). Las características extraídas sirven como la base para la clasificación de objetos, el reconocimiento de patrones y otras tareas.
* **Reconocimiento y Clasificación**: Una vez que las características relevantes se han extraído de la imagen, los algoritmos de clasificación, como las redes neuronales profundas, se utilizan para identificar o clasificar los objetos presentes en la imagen. Esto permite asignar una etiqueta o categoría a los objetos identificados.
* **Toma de Decisiones y Acción**: En algunos casos, el sistema de visión artificial no solo tiene que identificar los objetos, sino también tomar decisiones basadas en esa información. Por ejemplo, en sistemas de conducción autónoma, el sistema debe identificar obstáculos y decidir cómo actuar (frenar, desviar, etc.).

# Aplicaciones de la Visión Artificial

La visión artificial tiene un amplio espectro de aplicaciones en diversos campos. Algunas de las más destacadas son:

* **Reconocimiento Facial**: Se utiliza en sistemas de seguridad y verificación de identidad, como en el desbloqueo de dispositivos móviles y la autenticación en entornos bancarios.
* **Vehículos Autónomos**: Los coches autónomos utilizan visión artificial para reconocer señales de tráfico, detectar obstáculos y entender el entorno de conducción.
* **Medicina**: En la medicina, la visión artificial se emplea para analizar imágenes médicas, como radiografías y resonancias magnéticas, para ayudar en el diagnóstico de enfermedades y la planificación de tratamientos.
* **Robótica**: Los robots utilizan visión artificial para percibir su entorno y realizar tareas como la manipulación de objetos, la navegación y la interacción con seres humanos.
* **Inspección Industrial**: En el ámbito industrial, los sistemas de visión artificial se emplean para detectar defectos en productos fabricados, asegurando la calidad en la producción.
* **Reconocimiento de Texto**: La visión artificial también se usa en el reconocimiento óptico de sus principales caracteres para convertir texto impreso en texto digital, lo que facilita la digitalización de documentos.

# Desarrollo

**Librerias**

**Cv2 (OpenCV)**: Cv2 es el módulo de OpenCV para Python. OpenCV es una librería de computación visual para el procesamiento de imágenes en Python. Esta biblioteca proporciona herramientas para realizar operaciones de procesamiento de imágenes, como el filtrado, la detección de bordes, el reconocimiento de características, el seguimiento de objetos, etc. Estas herramientas nos permiten desarrollar aplicaciones de visión artificial, como el reconocimiento facial, el seguimiento de objetos, etc.

Fuente: <https://imaginaformacion.com/tutoriales/opencv-en-python>

**Mediapipe**: Es un entorno de código abierto para construir aplicaciones que realicen inferencia de visión por computadora sobre datos sensoriales como video, imágenes o audio.

Fue creado para equipos de inteligencia artificial, aprendizaje automático y desarrolladores de software con aplicaciones listas para producción o prototipos como parte de trabajos de investigación.

El entorno Mediapipe permite implementar soluciones para detección facial, manos, gestos, iris, etc.

Fuente: <https://viso.ai/computer-vision/mediapipe/>

**TKinter**: Es una biblioteca de lenguaje de programación Python que se utiliza para la creación y desarrollo de aplicaciones de escritorio. Esta librería proporciona soluciones para el posicionamiento de interfaces graficas de usuario en aplicaciones de escritorio.

PIL (Pillow): Es una biblioteca de procesamiento de imágenes en Python que proporciona funcionalidades para abrir, manipular y guardar imágenes en varios formatos. Permite realizar tareas como:

* Cargar imágenes desde archivos.
* Modificar el tamaño y la resolución de una imagen.
* Aplicar filtros y efectos a las imágenes.
* Realizar operaciones de recorte y rotación.
* Guardar imágenes en diferentes formatos, como JPEG, PNG, BMP, etc.

**Image**: El módulo de Image proporciona una clase con el mismo nombre que se utiliza para representar una imagen PIL. El módulo también proporciona una serie de funciones de fábrica, incluidas funciones para cargar imágenes desde archivos y crear nuevas imágenes.

Fuente: https://pillow.readthedocs.io/en/stable/reference/Image.html

**ImageTk**: Es un módulo en Python que se utiliza junto con la biblioteca Pillow (PIL) para facilitar la manipulación de imágenes en aplicaciones gráficas, especialmente aquellas creadas con Tkinter. Este módulo proporciona una clase que convierte objetos de imagen de Pillow en formatos que Tkinter puede utilizar fácilmente para mostrar imágenes en interfaces de usuario.

ImageTk actúa como un puente entre la manipulación de imágenes con Pillow y la visualización de estas imágenes en aplicaciones de Tkinter, permitiendo mostrar imágenes en ventanas, botones y otros componentes de interfaces graficas de usuario creadas con Tkinter

Fuente: https://pillow.readthedocs.io/en/stable/reference/ImageTk.html

**Numpy**: es una librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos.

Incorpora una nueva clase de objetos llamados arrays que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación.

La ventaja de Numpy frente a las listas predefinidas en Python es que el procesamiento de los arrays se realiza mucho más rápido (hasta 50 veces más) que las listas, lo cual la hace ideal para el procesamiento de vectores y matrices de grandes dimensiones.

Fuente: <https://aprendeconalf.es/docencia/python/manual/numpy/>

**Función encender\_webcam**

**Propósito general:** La función encender\_webcam tiene como objetivo inicializar la captura de video desde la cámara web del sistema y comenzar el flujo de procesamiento de los fotogramas en tiempo real.



**Declaración de variable global:**

La línea global cap indica que la variable cap, utilizada para manejar la captura de video, será accedida y modificada a nivel global. Esto permite que otras funciones del programa interactúen con la misma instancia de la cámara web.

**Inicialización de la captura de video:**

La línea cap = cv2.VideoCapture(0) crea un objeto de captura de video usando OpenCV (cv2), asignado a la variable global cap.

El parámetro 0 indica que se utiliza la cámara web predeterminada del sistema. Si existieran múltiples cámaras conectadas, podrían utilizarse otros índices como 1, 2, etc., para seleccionar una cámara específica.

**Inicio del procesamiento de fotogramas:**

La línea actualizar\_frame() llama a la función actualizar\_frame, lo que inicia el procesamiento continuo de los fotogramas capturados por la cámara. Esta función se encargará de realizar tareas como la detección de rostros y la actualización de la interfaz gráfica con los datos de video.

**Comportamiento esperado:**

Una vez ejecutada, la función activa la cámara web y pone en marcha el flujo de procesamiento en tiempo real.

Si la cámara ya estaba activa o en uso, el código no incluye una verificación explícita para manejar esa condición, por lo que podría ser necesario agregar validaciones adicionales para evitar conflictos.

**Función apagar\_webcam:**

**Propósito general:** La función apagar\_webcam tiene como objetivo detener la captura de video desde la cámara web y liberar los recursos asociados, además de limpiar el contenido visual del widget que muestra el video en la interfaz gráfica.



**Declaración de variable global:**

La línea global cap declara que la variable cap es global, permitiendo que la función acceda y modifique la instancia de captura de video inicializada previamente en encender\_webcam.

**Liberación de recursos de la cámara:**

La condición if cap: verifica si cap ha sido inicializada, es decir, si la cámara está activa. Si esta condición es verdadera:

La línea cap.release() libera los recursos asociados a la cámara web, deteniendo efectivamente la captura de video.

La línea cap = None asigna None a la variable global cap, indicando que la cámara ya no está en uso y evitando futuras operaciones no deseadas con esta variable.

**Limpieza del widget de video:**

La línea lbl\_video.imgtk = None elimina la referencia a cualquier imagen previa asignada al widget lbl\_video, asegurándose de que no se muestre un fotograma anterior o residual.

La línea lbl\_video.configure(image='') establece la propiedad de imagen del widget a un valor vacío, limpiando visualmente el área donde se mostraba el video en la interfaz gráfica.

**Comportamiento esperado:**

Cuando se ejecuta esta función, la cámara web se detiene y su recurso es liberado para que pueda ser utilizado por otros programas. Además, la interfaz gráfica deja de mostrar el contenido capturado por la cámara, proporcionando un estado limpio para el widget de video.

**Relación con otras funciones:**

Esta función complementa a encender\_webcam, ya que asegura que los recursos de la cámara se manejen correctamente cuando la cámara ya no es necesaria. Es especialmente útil para evitar problemas como conflictos en el acceso a la cámara por parte de otros programas o errores al intentar reiniciar la captura de video.

**Función guardar\_captura:**

**Propósito general:** La función guardar\_captura permite capturar un fotograma actual de la cámara web, almacenarlo en una variable global para su uso posterior y guardarlo como un archivo de imagen en el disco.



**Declaración de variable global:**

La línea global captura\_guardada declara que la variable captura\_guardada será utilizada a nivel global para almacenar temporalmente el fotograma capturado.

**Verificación de la cámara activa:**

La condición if cap: verifica si la cámara está activa (es decir, si cap fue inicializada por encender\_webcam). Si la cámara no está activa, la función no ejecutará las siguientes acciones.

**Captura del fotograma:**

La línea ret, frame = cap.read() intenta capturar un fotograma desde la cámara web:

ret es un valor booleano que indica si la captura fue exitosa (True) o no (False).

frame contiene los datos del fotograma capturado en formato de arreglo NumPy.

La condición if ret: asegura que las acciones siguientes solo se ejecuten si la captura del fotograma fue exitosa.

**Almacenamiento de la captura:**

La línea captura\_guardada = frame guarda el fotograma capturado en la variable global captura\_guardada, permitiendo que este fotograma se utilice en otras funciones del programa, como la comparación o el reconocimiento facial.

**Guardado en disco:**

La línea cv2.imwrite('captura.png', frame) guarda el fotograma capturado como un archivo de imagen en formato PNG en el directorio actual del programa. El nombre del archivo es captura.png.

Si ya existe un archivo con ese nombre, será sobrescrito.

**Notificación al usuario:**

La línea messagebox.showinfo("Guardar Captura", "Captura guardada exitosamente.") muestra una ventana emergente al usuario con un mensaje que confirma que la captura fue guardada correctamente.

**Comportamiento esperado:**

Si la cámara está activa y la captura del fotograma es exitosa, se guardará una copia del fotograma tanto en la variable captura\_guardada como en un archivo de imagen en el disco. El usuario será notificado del éxito de la operación mediante un mensaje emergente.

Si la cámara no está activa o la captura falla, la función no realiza ninguna acción.

**Relación con otras funciones:**

Esta función trabaja en conjunto con comparar\_captura y iniciar\_reconocimiento, ya que utiliza el fotograma guardado en captura\_guardada como referencia para realizar comparaciones o reconocimiento facial.

**Función comparar\_captura:**

**Propósito general:** La función comparar\_captura compara un fotograma capturado en tiempo real desde la cámara web con una captura previamente guardada, calculando el porcentaje de similitud entre ambas imágenes.



**Verificación de la existencia de una captura guardada:**

La línea if captura\_guardada is None: verifica si la variable global captura\_guardada contiene una imagen previamente almacenada.

Si no hay ninguna captura guardada, se muestra un mensaje de advertencia al usuario mediante messagebox.showwarning("Comparar Captura", "No hay captura guardada para comparar."), y la función finaliza sin ejecutar más acciones.

**Verificación de la cámara activa:**

La condición if cap: asegura que la cámara está activa y lista para capturar un nuevo fotograma. Si no lo está, la función no realiza ninguna acción.

**Captura del fotograma actual:**

La línea ret, frame = cap.read() intenta capturar un nuevo fotograma desde la cámara:

ret indica si la captura fue exitosa.

frame contiene los datos de la imagen capturada.

**Conversión a escala de grises:**

Ambas imágenes (la captura guardada y el fotograma actual) se convierten a escala de grises utilizando cv2.cvtColor:

gray\_captura = cv2.cvtColor(captura\_guardada, cv2.COLOR\_BGR2GRAY) convierte la captura guardada a escala de grises.

gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) convierte el fotograma capturado a escala de grises.

La conversión a escala de grises es necesaria para simplificar la comparación y reducir la complejidad computacional.

**Cálculo de la similitud:**

La línea similitud = cv2.matchTemplate(gray\_frame, gray\_captura, cv2.TM\_CCOEFF\_NORMED) utiliza la función cv2.matchTemplate para calcular la similitud entre las dos imágenes en escala de grises:

El método cv2.TM\_CCOEFF\_NORMED normaliza los resultados para obtener valores entre -1 y 1, donde 1 indica una coincidencia perfecta.

El resultado es una matriz que contiene los valores de similitud para diferentes desplazamientos entre las dos imágenes.

**Determinación del porcentaje de similitud:**

La línea porcentaje\_similitud = np.max(similitud) \* 100 calcula el porcentaje de similitud basado en el valor máximo de la matriz de resultados.

**Notificación del resultado:**

El porcentaje de similitud se muestra al usuario mediante messagebox.showinfo("Comparar Captura", f"Porcentaje de similitud: {porcentaje\_similitud:.2f}%").

**Comportamiento esperado:**

Si la cámara está activa y existe una captura guardada, el programa calcula la similitud entre ambas imágenes y muestra el resultado al usuario como un porcentaje.

Si no hay una captura guardada, el usuario es notificado mediante un mensaje de advertencia.

Si la cámara no está activa, no se realiza ninguna acción.

**Relación con otras funciones:**

Depende de guardar\_captura para tener una imagen de referencia almacenada en captura\_guardada.

También utiliza la captura en tiempo real proporcionada por cap, que se inicializa con encender\_webcam.

**Función iniciar\_reconocimiento:**

**Propósito general:** La función iniciar\_reconocimiento activa un proceso de comparación continua en tiempo real entre el fotograma capturado por la cámara web y una captura previamente guardada, notificando al usuario si se detecta una similitud suficiente.



**Verificación de la existencia de una captura guardada:**

La línea if captura\_guardada is None: asegura que existe una captura previamente guardada en la variable global captura\_guardada.

Si no hay ninguna captura guardada, se muestra un mensaje de advertencia al usuario mediante messagebox.showwarning("Iniciar Reconocimiento", "No hay captura guardada para comparar.").

Si la captura no existe, la función finaliza sin realizar ninguna acción adicional.

**Activación del reconocimiento:**

La línea reconocimiento\_activo = True establece la variable global reconocimiento\_activo como True, indicando que el reconocimiento facial en tiempo real está activo.

Llamada a la función de comparación en tiempo real:

La línea comparar\_en\_tiempo\_real() invoca otra función encargada de ejecutar el reconocimiento continuo (detallada más adelante). Esto permite realizar la comparación de imágenes en intervalos regulares mientras reconocimiento\_activo sea True.

**Comportamiento esperado:**

Si existe una captura guardada, la función inicia el proceso de reconocimiento facial en tiempo real.

Si no existe una captura guardada, el usuario es notificado mediante un mensaje de advertencia y el reconocimiento no comienza.

**Relación con otras funciones:**

Depende de guardar\_captura para tener una imagen de referencia almacenada en captura\_guardada.

Activa la función comparar\_en\_tiempo\_real para realizar la comparación continua entre las imágenes.

**Función comparar\_en\_tiempo\_real:**

Propósito general: La función comparar\_en\_tiempo\_real realiza un reconocimiento facial en tiempo real mediante la comparación continua entre un fotograma capturado por la cámara y una captura previamente guardada. Este proceso se repite automáticamente mientras el reconocimiento esté activo.

**Verificación de condiciones iniciales:**

La función comienza evaluando si el reconocimiento está activo y si existe una captura guardada:

if reconocimiento\_activo and captura\_guardada is not None:

reconocimiento\_activo: Debe ser True, indicando que el usuario activó el reconocimiento mediante iniciar\_reconocimiento.

captura\_guardada: Debe contener una imagen válida que se haya guardado previamente con la función guardar\_captura.

**Captura del fotograma actual:**

Si las condiciones iniciales son verdaderas y la cámara está activa (if cap:), se lee un nuevo fotograma de video:

ret, frame = cap.read()

ret: Indica si la captura fue exitosa.

frame: Contiene los datos del fotograma capturado.

**Conversión a escala de grises:**

Para simplificar el proceso de comparación, tanto la imagen capturada como la guardada se convierten a escala de grises:

gray\_captura = cv2.cvtColor(captura\_guardada, cv2.COLOR\_BGR2GRAY)

gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

La escala de grises reduce la cantidad de información procesada, centrándose en las diferencias de luminosidad entre píxeles.

**Cálculo de similitud:**

La similitud entre las dos imágenes se evalúa mediante cv2.matchTemplate, utilizando el método de correlación normalizada:

similitud = cv2.matchTemplate(gray\_frame, gray\_captura, cv2.TM\_CCOEFF\_NORMED)

porcentaje\_similitud = np.max(similitud) \* 100

similitud: Es una matriz que contiene los valores de correlación.

np.max(similitud): Obtiene el valor máximo de correlación, representando la similitud más alta entre las dos imágenes.

porcentaje\_similitud: Convierte este valor a un porcentaje.

Notificación de reconocimiento exitoso:

Si el porcentaje de similitud supera o iguala el umbral del 95%:

if porcentaje\_similitud >= 95:

messagebox.showinfo("Reconocimiento Exitoso", "Reconocimiento exitoso")

reconocimiento\_activo = False

Muestra un mensaje al usuario indicando que el reconocimiento fue exitoso.

Detiene el reconocimiento desactivando la bandera reconocimiento\_activo.

**Repetición del proceso:**

Si el reconocimiento sigue activo (reconocimiento\_activo), se agenda una nueva llamada a la función tras 500 milisegundos:

if reconocimiento\_activo:

root.after(500, comparar\_en\_tiempo\_real)

root.after(500, comparar\_en\_tiempo\_real): Utiliza el temporizador de Tkinter para ejecutar la función nuevamente después del intervalo especificado.

**Comportamiento esperado:**

Compara continuamente los fotogramas capturados con la imagen guardada.

Detiene el proceso al alcanzar un porcentaje de similitud del 95% o más, notificando al usuario.

**Relación con otras funciones:**

Depende de guardar\_captura para obtener la imagen de referencia.

Es activada por iniciar\_reconocimiento y puede ser detenida indirectamente por detener\_reconocimiento.

**Función detener\_reconocimiento:**

**Propósito general:** La función detener\_reconocimiento permite al usuario detener el proceso de reconocimiento facial en tiempo real, interrumpiendo la ejecución continua de la función comparar\_en\_tiempo\_real.



**Desactivación del reconocimiento:**

La función modifica la variable global reconocimiento\_activo para detener el proceso de comparación en tiempo real:

global reconocimiento\_activo

reconocimiento\_activo = False

reconocimiento\_activo: Se asigna el valor False, indicando que el reconocimiento facial ya no está activo.

**Notificación al usuario:**

Una vez que se ha desactivado el reconocimiento, se muestra un mensaje al usuario para confirmar la acción:

messagebox.showinfo("Detener Reconocimiento", "Reconocimiento detenido.")

messagebox.showinfo: Utiliza un cuadro de diálogo informativo de Tkinter para comunicar que el reconocimiento ha sido detenido correctamente.

**Relación con otras funciones:**

Se utiliza para finalizar el proceso iniciado por iniciar\_reconocimiento.

Detiene la ejecución cíclica de comparar\_en\_tiempo\_real al cambiar el estado de reconocimiento\_activo.

**Comportamiento esperado:**

Interrumpe inmediatamente cualquier comparación en curso.

Garantiza que no se realicen nuevas evaluaciones de similitud hasta que el reconocimiento sea activado nuevamente.

**Función actualizar\_frame:**

**Propósito general:** La función actualizar\_frame tiene como objetivo principal capturar y procesar los fotogramas de video desde la cámara web en tiempo real, utilizando Mediapipe para la detección de rostros, y actualizar el contenido visual en una ventana de la interfaz gráfica.



**Referencia a la variable global cap:**

La función comienza declarando que la variable cap es global, lo que permite acceder a la captura de video inicializada previamente por la función encender\_webcam.



**Verificación de captura activa:**

La línea if cap: asegura que cap ha sido correctamente inicializada, es decir, que la cámara está activa y lista para capturar fotogramas. Si esto es cierto, se ejecutará el código contenido dentro de esta condición.

**Captura del fotograma:**

La línea ret, frame = cap.read() realiza una operación de asignación doble. La función cap.read() devuelve dos valores:

ret: un valor booleano que indica si la captura del fotograma fue exitosa (True) o si falló (False).

frame: un arreglo de tipo numpy que contiene los datos en formato de imagen del fotograma capturado.

**Conversión de color del fotograma:**

La línea img\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB) convierte el fotograma capturado (almacenado en frame) del formato de color BGR (Blue-Green-Red) al formato RGB (Red-Green-Blue). Esta conversión es necesaria porque Mediapipe y la biblioteca PIL esperan las imágenes en formato RGB para procesarlas correctamente.

**Procesamiento de detección de rostros con Mediapipe:**

La línea results = mp\_face\_detection.process(img\_rgb) utiliza la instancia de Mediapipe Face Detection (mp\_face\_detection) para procesar el fotograma convertido a RGB. Este procesamiento devuelve un objeto results que contiene la información de detección, incluyendo las coordenadas de los rostros encontrados, si es que existen.

**Dibujado de las detecciones en el fotograma original:**

El bloque if results.detections: verifica si se detectaron rostros en el fotograma. Si el atributo detections del objeto results contiene datos, se ejecutará el bloque de código interno.

Dentro de este bloque, for detection in results.detections: recorre cada detección individual en el fotograma.

La línea mp\_drawing.draw\_detection(frame, detection) utiliza Mediapipe Drawing Utilities (mp\_drawing) para dibujar un marco y otros elementos visuales directamente en el fotograma original (frame) sobre los rostros detectados.

**Conversión del fotograma para compatibilidad con Tkinter:**

La línea img = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)) convierte nuevamente el fotograma procesado a formato RGB, utilizando OpenCV (cv2) para la conversión, y luego lo convierte a un objeto Image de la biblioteca PIL. Esto es necesario para que el fotograma pueda ser utilizado en Tkinter.

**Creación de imagen para Tkinter:**

La línea imgtk = ImageTk.PhotoImage(image=img) crea un objeto PhotoImage, que es el formato que Tkinter requiere para mostrar imágenes en su interfaz.

**Actualización del widget de video:**

La línea lbl\_video.imgtk = imgtk asigna la imagen procesada a la propiedad imgtk del widget lbl\_video, asegurándose de que la referencia a la imagen no se pierda.

La línea lbl\_video.configure(image=imgtk) establece la imagen en el widget lbl\_video, actualizando así el contenido visual mostrado en la interfaz.

**Programación de la siguiente actualización:**

La línea lbl\_video.after(10, actualizar\_frame) programa la ejecución de la función actualizar\_frame nuevamente después de un intervalo de 10 milisegundos. Esto crea un ciclo continuo que permite la actualización en tiempo real de los fotogramas de video en la interfaz gráfica.

# Recomendaciones

# Conclusiones