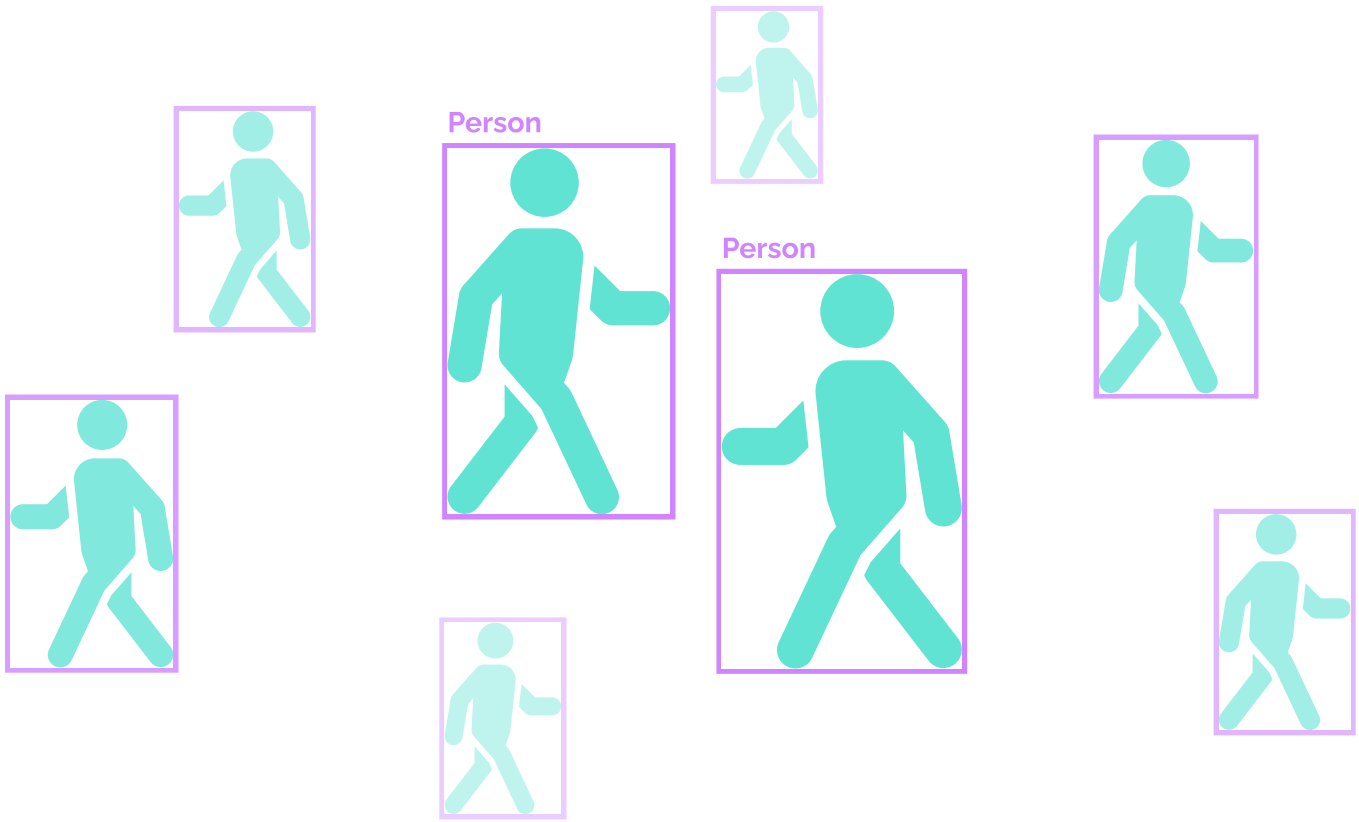
2023/24

Proyecto Final.
Conteo y Detección de Personas

Visión por Computador



Jose Ricardo Peña Seco

Sara González Ramírez

Índice

Introducción y objetivos

El presente estudio se focaliza en la identificación, recuento y análisis de la edad y género de individuos mediante el empleo de *YOLOv8* y *MediaPipe*. El propósito fundamental es poder examinar y supervisar el tránsito de personas dentro de un establecimiento, posibilitando la generación de estadísticas detalladas sobre la cantidad y tipología de personas que ingresan a lo largo del día.

La disposición de una cámara posicionada lateralmente permite la detección de personas que atraviesan la entrada del establecimiento, llevándose a cabo un seguimiento para determinar si están ingresando o saliendo del edificio. Una vez que una persona cruza la línea límite, indicando su entrada, se activa una segunda cámara ubicada frontalmente en la entrada. Esta cámara captura una imagen y realiza un análisis para determinar el género y la edad de la persona detectada.

Los datos recopilados se almacenan en una base de datos dedicada, la cual se utilizará para generar estadísticas detalladas sobre el flujo de personas en el establecimiento a lo largo del año. La información, tanto en forma de vídeos de las cámaras como de métricas y gráficos, estará accesible a través de un servidor web especialmente desarrollado para este propósito.

Sistema

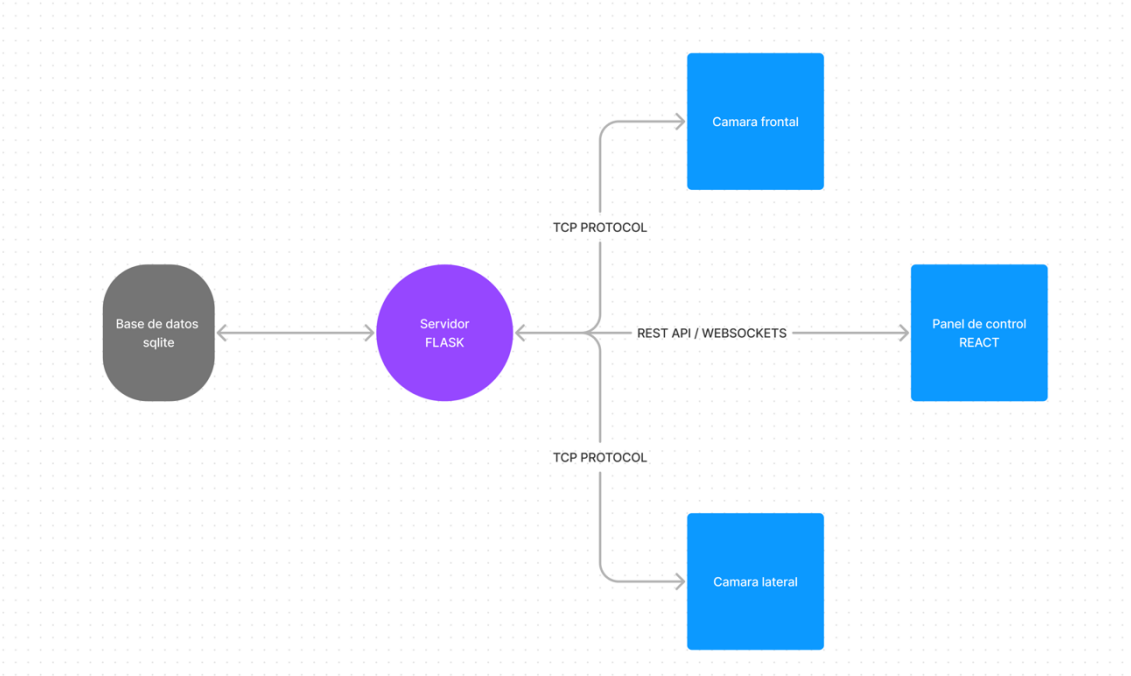
El ecosistema de nuestro proyecto se compone de cuatro aplicaciones interconectadas diseñadas para la captura y análisis de datos de visitantes, así como para la visualización de métricas en tiempo real:

1. **Aplicación Cliente - Cámara de Perfil**: Esta aplicación se encarga de monitorear las entradas y salidas de los visitantes, utilizando una cámara dedicada a detectar y registrar el flujo de personas.
2. **Aplicación Cliente - Cámara Frontal**: Orientada a la identificación demográfica, esta cámara se especializa en la detección de edad y género de los visitantes, proporcionando datos valiosos para análisis estadísticos.
3. **Aplicación Servidora Central**: Actúa como el núcleo del sistema, recibiendo datos de ambas cámaras. Procesa la información recibida, almacena los resultados en una base de datos y suministra estos datos a la interfaz de métricas. Además, facilita la transmisión de imágenes de las cámaras a través de HTTP.
4. **Aplicación Cliente - Panel de Control**: Esta interfaz permite a los usuarios visualizar las métricas capturadas, actualizadas en tiempo real mediante websockets. También proporciona acceso a las transmisiones de video en directo de las cámaras, utilizando un protocolo personalizado sobre TCP para la retransmisión de imágenes.

El funcionamiento del sistema es como sigue: Cada cámara mantiene un socket TCP abierto, a la espera de solicitudes para transmitir video. La aplicación servidora central se conecta a estas cámaras y ofrece un endpoint HTTP para acceder a las imágenes en vivo. Paralelamente, la aplicación web recupera las métricas y las transmisiones de video a través de esta interfaz. La API para las métricas sigue el estilo RESTful.

Cuando la cámara de perfil detecta un visitante entrando, envía una señal al servidor para que lo registre. El servidor, a su vez, solicita a la cámara frontal una captura instantánea para analizar y detectar las características faciales del visitante. Tras este análisis, registra la nueva entrada del cliente en la base de datos junto con la información demográfica obtenida y notifica al panel de control mediante websockets, lo que permite la actualización instantánea de las estadísticas visualizadas.

El esquema sería el siguiente:



Descripción Técnica

Panel de control

El panel de control se encarga de dos aspectos fundamentales: mostrar las métricas en tiempo real y la retransmisión de las cámaras.

La intención es que el dueño de la tienda pueda observar en todo momento las métricas de su negocio y a su vez controlar el mismo a través de la visualización de las cámaras

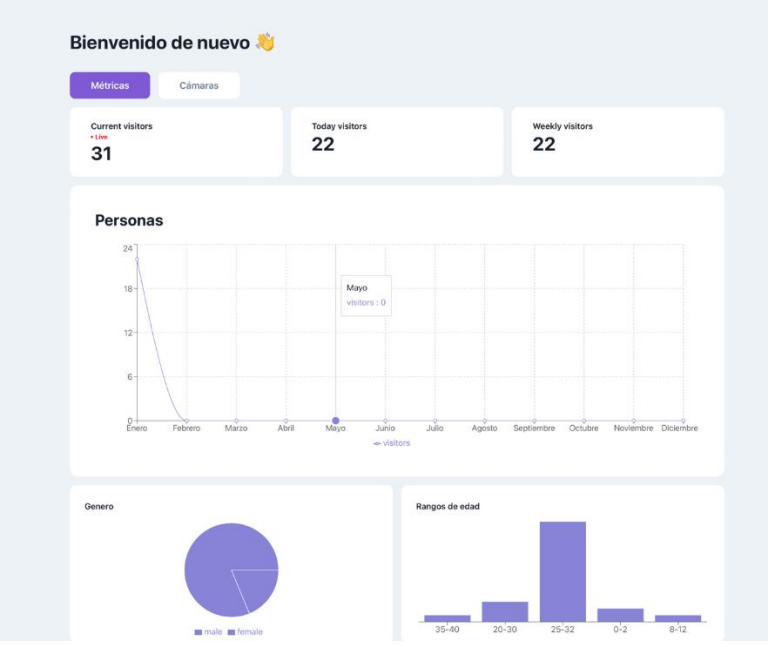
Este ha sido creado usando *Typescript*, un superset de javascript que añade tipado y una mejor orientación a objetos*, y React,* una librería para la creación de aplicaciones web*.* Para las gráficas se ha usado una librería llamada *recharts.*

Métricas

Las métricas que hemos decidido añadir son las siguientes:

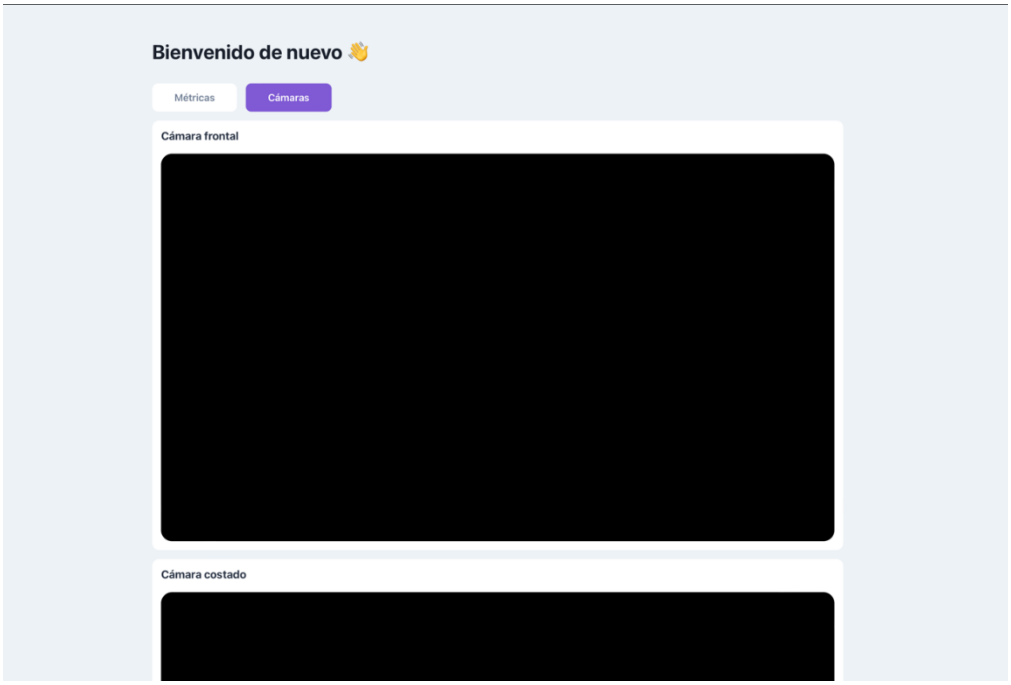
- Personas dentro de la tienda
- Visitas totales del día
- Visitas de la semana
- Gráfico de línea de las visitas a lo largo del año
- Gráfico circular de la distribución entre hombres y mujeres
- Gráfico de barras de la distribución de edad de los visitantes

Las métricas son obtenidas inicialmente por http usando una *Rest API*. Sin embargo, una vez cargadas se actualizan en tiempo real a través de las cámaras usando websockets. Para ello hemos utilizado socket.io.



Cámaras

Respecto las cámaras estas son mostradas en su correspondiente página. Hemos optado por poner la captura de pantalla con las cámaras desactivadas por motivos de privacidad. Las cámaras retransmiten a través de TCP, con un protocolo de capa de aplicación bastante simple que hemos desarrollado nosotros para la retransmisión de las imágenes. Para ello hemos usado la interfaz sockets ofrecida por el sistema operativo y accesible desde Python.



Cámara Perfil: Conteo y Detección de Personas

Este proceso se divide en dos partes: detección de las personas que capture la cámara y el conteo de cuántas entran y salen del lugar. Solo se puede contar una persona que pase a la vez por lo que, en esta fase, el proyecto solo está pensado para establecimientos o servicios cuyas entradas solo den capacidad para una persona.

Este código se encuentra dentro del repositorio vc-trabajo-final-cliente-deteccion-clientes.

Detección

Para la detección, se han desarrollado dos clases fundamentales dentro del archivo *detection.py*: **Person** y **YOLOPersonDetector**.

Clase Person

La primera clase, denominada Person, desempeña un papel esencial en la gestión individual de cada persona detectada. Su función principal consiste en almacenar información relevante sobre la persona, como las coordenadas de la caja delimitadora de su figura, la posición central, la trayectoria, la dirección del movimiento, tiempo de vida de la persona siendo detectada y un indicador que señala si ha superado el tiempo máximo de seguimiento.

Este seguimiento temporal se realiza mediante métodos como **age_one()**, que incrementa la edad de la persona en una unidad, y **is_time_out()**, que verifica si la persona ha excedido su tiempo de seguimiento máximo.

Asimismo, la clase posee el método **update_coords()** para actualizar las coordenadas de la persona y el método **calculate_dir()** para calcular su dirección de movimiento con relación a unos límites especificados.

Clase YOLOPersonDetector

La segunda clase, YOLOPersonDetector, se encarga de integrar el modelo YOLO para la detección de personas en una imagen.

Su método principal, **predict()**, acepta una imagen como entrada y devuelve una lista de las coordenadas de las personas detectadas. Este método utiliza la biblioteca *ultralytics* para realizar predicciones a partir del modelo YOLO, extrayendo las cajas delimitadoras de los objetos detectados y verificando la categoría del objeto para determinar si corresponde a una persona. En caso afirmativo, las coordenadas se agregan a la lista de las coordenadas de las personas detectadas y se devuelve como resultado.

Controlador y Contador

Se han diseñado dos clases fundamentales para gestionar la visualización de información y llevar a cabo el seguimiento y conteo de individuos detectados en cada frame. Estas clases, denominadas **Painter** y **PersonCounterController**, cumplen roles clave en el sistema y se detallan a continuación.

Clase Painter

La clase **Painter** se encarga de la presentación visual de la información en la imagen procesada. Su función principal es dibujar líneas representativas, contar el número de personas detectadas y resaltar visualmente la presencia de cada individuo en la imagen.

Se utilizan diversos métodos, como **paint_lines()**, que dibuja líneas divisorias y texto en la imagen para etiquetar las líneas de delimitan los límites de control, y **paint_counter()**, que muestra un contador de personas en una esquina de la imagen. Además, el método **paint_person()** resalta visualmente cada persona detectada, marcando su caja delimitadora, centro y etiqueta.

Clase PersonCounterController

La segunda clase, **PersonCounterController**, opera como el controlador principal de la aplicación. Se encarga de orquestar la interacción entre el detector de personas, el presentador visual y de avisar al servidor de que una persona ha entrado al local.

Al recibir un frame de la cámara, redimensiona la imagen según las especificaciones proporcionadas y realiza predicciones de detección de personas mediante el objeto **YOLOPersonDetector**. Luego, utiliza el objeto **Painter** para visualizar líneas divisorias, contadores y la presencia de personas en la imagen.

El método **track()** se encarga de seguir el movimiento de las personas detectadas y calcular su dirección en relación con límites específicos en la imagen con el método realizado en la clase **Person**, **calculate_dir()**.

Por otro lado, el método **count()** actualiza el contador de personas en función de la dirección de movimiento de cada persona detectada y, a su vez, va enviando esta información al servidor, avisando a la **Cámara Frontal** de que tiene que realizar una foto cuando el contador suma 1.

Código Principal

En el script principal, llamado **main.py**, se establece la lógica central para la ejecución del sistema de conteo y seguimiento de personas.

En la primera sección, se importan las clases esenciales desde los módulos **control** y **camera**. La clase **PersonCounterController** se encarga del control del conteo de personas, mientras que **CameraServer** se utiliza para el envío de frames al servidor desde la cámara.

La función principal, **main()**, inicializa instancias de las clases mencionadas y configura parámetros clave como el tamaño de la imagen, límites de conteo, edad máxima y el modelo YOLO a utilizar. Posteriormente, se utiliza la cámara del sistema, inicializada mediante OpenCV, para capturar frames en un bucle continuo. Cada frame capturado es procesado por el objeto **PersonCounterController**, llevando a cabo la detección de personas, así como el seguimiento y conteo.

En resumen, **main.py** funciona como el punto de entrada del sistema, coordinando la adquisición y procesamiento de frames, y presentando visualmente los resultados de la detección y conteo de personas en tiempo real. La modularidad de las clases facilita una fácil expansión y mantenimiento del sistema.

Cámara Frontal: Análisis de Género y Edad

Este proceso realiza la detección de rostros y el análisis de género y edad a partir de imágenes capturadas por la cámara frontal.

Este código se encuentra dentro del repositorio vc-trabajo-final-cliente-person-qualifier.

Clasificador de Caras

Interfaz FaceDetector y Dataclasses

En el archivo **face_detector.py**, se ha diseñado una estructura modular para la detección de caras en imágenes. Se utilizan las bibliotecas **numpy** para manipulación de datos y **dataclasses** para definir clases inmutables. Además, se emplea la clase abstracta **ABC** del módulo **abc** para establecer una interfaz abstracta que las implementaciones específicas deben cumplir.

Tres clases de datos se definen utilizando la anotación **@dataclass**: **Point** para representar puntos en un plano, **BoundingBox** para delimitar objetos en una imagen, y **FaceDetectorResult** para almacenar los resultados de la detección de caras, incluyendo la imagen y la caja delimitadora.

La clase abstracta **FaceDetector** establece una interfaz que exige que cualquier implementación proporcione un método **detect()**. Este método recibe una imagen representada como un array de **numpy** y devuelve una lista de resultados de detección de caras.

Clase MediaPipeFaceDetector

En **mediapipe_face_detector.py**, la clase **MediaPipeFaceDetector** ofrece una implementación específica del detector de caras utilizando la biblioteca MediaPipe. A continuación, se describen detalladamente los métodos clave de esta implementación.

El método **detect()** implementa la detección de caras utilizando la biblioteca MediaPipe. Se inicia definiendo las opciones necesarias para el detector de caras, incluyendo el modelo específico (**blaze_face_short_range.tflite**). Luego, se crea un objeto detector utilizando estas opciones y se procesa la imagen mediante el método **detect()** del detector de caras. La información resultante se convierte en instancias de la clase **FaceDetectorResult** mediante el método **_convert_to_face_detector_result()**. Finalmente, se devuelve una lista de los resultados de la detección.

El método **_convert_to_face_detector_result()** transforma los resultados de la detección proporcionados por MediaPipe en instancias de la clase **FaceDetectorResult**. Recibe las coordenadas de las caras detectadas y las convierte en instancias de la clase **BoundingBox** que representa una caja delimitadora. Luego, para cada caja delimitadora, se utiliza el método **ImageUtils.crop()** para recortar la imagen original y se crea una instancia de **FaceDetectorResult** que contiene la imagen

recortada y la caja delimitadora correspondiente. La lista resultante de instancias de **FaceDetectorResult** se devuelve.

Clase ViolaJonesFaceDetector

La clase **ViolaJonesFaceDetector** implementa la detección de caras utilizando el clasificador de Viola-Jones mediante la biblioteca OpenCV. En el método **detect()**, se carga el clasificador previamente entrenado y se aplica para identificar caras potenciales en la imagen. Las coordenadas de las caras detectadas se convierten luego en instancias de la clase **FaceDetectorResult** mediante el método **_convert_to_face_detector_result()**.

El método **_convert_to_face_detector_result()** transforma las coordenadas de las caras en instancias de **BoundingBox** y utiliza el método **ImageUtils.crop()** para recortar la imagen original. Se crea una lista de instancias de **FaceDetectorResult** que contienen la imagen recortada y la caja delimitadora correspondiente.

Por otro lado, la función **_convert_image_to_gray** convierte una imagen a escala de grises utilizando la función **cv2.cvtColor()** de OpenCV.

Esta clase es una adición ya que realmente no se utiliza en el código final.

Descripción General de la Clase FaceQualifier

La clase **FaceQualifier** es una implementación en Python para la detección y clasificación de atributos faciales, específicamente edad y género. Se utiliza en conjunto con modelos de aprendizaje profundo pre-entrenados para analizar imágenes de rostros y estimar estas características. Esta clase es especialmente útil en aplicaciones que necesitan identificación demográfica de individuos, como análisis de clientes, sistemas de seguridad, o para fines estadísticos.

Descripción de Atributos y Métodos de FaceQualifier

Atributos

1. **Model Paths**: Rutas a los archivos de los modelos pre-entrenados y sus archivos de configuración para la detección de edad (**ageModel** y **ageProto**) y género (**genderModel** y **genderProto**).
2. **MODEL_MEAN_VALUES**: Una tupla de valores medios que se utilizan para normalizar los datos de la imagen durante el preprocesamiento.
3. **ageList**: Una lista de intervalos de edad utilizada para clasificar la edad estimada del rostro detectado.
4. **genderList**: Una lista con las clasificaciones de género ('m' para masculino y 'f' para femenino).
5. **ageNet**: Red neuronal para la clasificación de edad.
6. **genderNet**: Red neuronal para la clasificación de género.

Métodos

1. **detect**:
    - Toma una imagen de un rostro como entrada.
    - Procesa la imagen (a través de **blobFromImage**) ajustándola al tamaño requerido por los modelos y aplicando la normalización.
    - Utiliza **ageNet** y **genderNet** para predecir la edad y el género, respectivamente.
    - Devuelve una tupla con la clasificación de género y un rango de edad.

Utilidad de la Clase FaceQualifier

FaceQualifier es una herramienta poderosa para aplicaciones que requieren análisis demográfico rápido y preciso basado en imágenes de rostros. Al utilizar modelos de aprendizaje profundo, puede proporcionar estimaciones de edad y género en tiempo real, lo que es valioso en diversos campos como marketing, control de acceso, análisis de público en eventos, y más. La capacidad de esta clase para integrarse con sistemas de procesamiento de imágenes y cámaras en vivo la hace extremadamente útil para aplicaciones de visión por computadora y análisis de datos en tiempo real.

Tratado de Imagen

En el archivo **utils.py** se ha desarrollado la clase **ImageUtils**, la cual proporciona funciones útiles para el tratado de imágenes. A continuación, se explica cada función de forma detallada.

El método estático **crop()** recibe una imagen representada como un array de **numpy** y una instancia de la clase **BoundingBox**. Utilizando las coordenadas de la caja delimitadora, se recorta la región correspondiente de la imagen y se devuelve como un nuevo array de **numpy**. Este método es útil para extraer partes específicas de una imagen, como las caras detectadas.

El método **overlay_icons()** superpone un icono en una imagen en una posición específica. Toma la imagen original, la ruta del icono, un color, el tamaño del icono y las coordenadas del punto de superposición. El icono se lee y redimensiona según el tamaño proporcionado. Luego, se crea una máscara del icono donde los píxeles son 0, y se genera una capa de color del tamaño del icono. La capa de color se copia sobre la región de la imagen original especificada por las coordenadas del punto de superposición, respetando la máscara del icono. El resultado es una imagen con el icono superpuesto.

Código Principal

En el script principal de clasificación de caras, **main.py**, se realiza la detección y resaltado de caras en tiempo real utilizando las clases creadas con MediaPipe y transmite los fotogramas procesados a través de un servidor.

Comienza importando las bibliotecas necesarias, incluyendo OpenCV para la manipulación de video, el módulo de registro (**logging**), la implementación de detección de caras con MediaPipe (**MediaPipeFaceDetector**), y el detector de caras basado en el clasificador de Viola-Jones (**ViolaJonesFaceDetector**). También se importa la clase **CameraServer** para la transmisión de video.

Luego, se inicializa la captura de video desde la cámara utilizando **cv2.VideoCapture(0)**. Se crea una instancia de **CameraServer** que inicia un servidor para enviar los fotogramas capturados.

En el bucle principal (**while True**), se lee cada fotograma de la cámara. Se utiliza la instancia de **MediaPipeFaceDetector** para detectar caras en el fotograma y se resalta cada cara detectada con un rectángulo azul en la imagen original.

Posteriormente, el fotograma modificado se envía al servidor mediante el método **server.send_frame()**. Se maneja cualquier excepción que pueda ocurrir durante el proceso de envío, y se registra un mensaje de error si es necesario.

El bucle se ejecuta continuamente hasta que se interrumpe manualmente o se produce una excepción. Finalmente, se cierra la conexión del servidor con **server.close()**.

Servidor central

El servidor ha sido desarrollado con el framework Flask. Vamos a ir documentando la API expuesta para toda la funcionalidad de la aplicación.

Endpoints

Streaming

Endpoint: **/streaming/front_camera/**

- **Método HTTP**: GET
- **Descripción**: Este endpoint se encarga de transmitir en vivo el video de una cámara frontal. Es útil en escenarios donde se necesita monitorear o visualizar una transmisión en tiempo real desde una cámara específicamente designada como 'frontal', como podría ser en un sistema de seguridad, monitoreo de tráfico, o en cualquier aplicación que requiera visualizar en tiempo real lo que sucede en la parte frontal de una ubicación.

- **Funcionamiento**: Al recibir una solicitud GET a este endpoint, la aplicación busca en la base de datos la primera cámara clasificada como 'frontal'. Luego, utiliza la dirección IP y el puerto de esta cámara para crear una instancia de **NetworkCamera**, que es responsable de generar el flujo de video. Finalmente, responde a la solicitud con el flujo de video en tiempo real, utilizando un formato adecuado para transmisiones de video en vivo (**multipart/x-mixed-replace**).

Endpoint: **/streaming/side_camera/**

- **Método HTTP**: GET
- **Descripción**: Similar al endpoint de la cámara frontal, este endpoint gestiona la transmisión en vivo de una cámara lateral. Este endpoint es especialmente útil en situaciones donde se requiere tener una vista lateral, como puede ser el monitoreo de áreas laterales de un edificio, vigilancia perimetral, o para obtener ángulos diferentes en un evento en vivo.
- **Funcionamiento**: Funciona de manera similar al endpoint de la cámara frontal. Al recibir una solicitud GET, busca en la base de datos la cámara etiquetada como 'lateral'. Crea una instancia de **NetworkCamera** con la información de esta cámara y luego transmite el video en vivo. La respuesta es un flujo de video en tiempo real, utilizando el mismo formato de MIME type para asegurar una transmisión continua y efectiva.

Estadísticas

Endpoint: **/visitors/male_female_ratio/**

- **Método HTTP**: GET
- **Descripción**: Este endpoint proporciona la proporción de visitantes masculinos y femeninos. Es útil para análisis demográficos o para entender la distribución de género entre los visitantes.
- **Funcionamiento**: Realiza una consulta a la base de datos para contar las entradas clasificadas como masculinas (**'m'**) y femeninas (**'f'**), y devuelve un JSON con la cuenta de cada género.

Endpoint: **/visitors/age_interval_count/**

- **Método HTTP**: GET
- **Descripción**: Devuelve el conteo de visitantes según intervalos de edad. Este endpoint es crucial para análisis demográficos detallados y para entender la distribución de edades entre los visitantes.
- **Funcionamiento**: Recupera todas las entradas, agrupa por intervalos de edad y cuenta la cantidad de entradas en cada grupo, devolviendo los resultados en un formato JSON.

Endpoint: **/entrances/**

- **Método HTTP**: GET
- **Descripción**: Proporciona una lista de todas las entradas registradas. Este endpoint es útil para obtener una visión general de todos los visitantes.
- **Funcionamiento**: Recupera todas las entradas de la base de datos y las devuelve en formato JSON.

Endpoint: **/entrances/** (con método POST)

- **Método HTTP**: POST
- **Descripción**: Permite la creación de un nuevo registro de entrada. Este endpoint es esencial para añadir nuevos datos de visitantes a la base de datos.
- **Funcionamiento**: Crea una nueva entrada en la base de datos con los datos proporcionados en la solicitud JSON y devuelve la entrada creada en formato JSON.

Endpoint: **/entrances/<int:entrance_id>/**

- **Método HTTP**: GET
- **Descripción**: Recupera detalles de una entrada específica, identificada por su ID. Este endpoint es útil para obtener información detallada sobre una entrada particular.
- **Funcionamiento**: Busca en la base de datos una entrada con el ID especificado y devuelve su información en formato JSON.

Endpoint: **/visitors/today/**

- **Método HTTP**: GET
- **Descripción**: Cuenta el número de visitantes del día actual. Este endpoint es útil para obtener estadísticas diarias.
- **Funcionamiento**: Filtra las entradas en la base de datos por la fecha actual y devuelve el conteo de estas.

Endpoint: **/visitors/week/**

- **Método HTTP**: GET
- **Descripción**: Calcula el número de visitantes en la última semana. Esencial para análisis semanales y tendencias a corto plazo.
- **Funcionamiento**: Filtra las entradas de la última semana y devuelve su cantidad.

Endpoint: **/visitors/per_month/**

- **Método HTTP**: GET
- **Descripción**: Devuelve el conteo de visitantes por cada mes del año actual. Este endpoint es valioso para análisis mensuales y comprensión de tendencias a lo largo del año.

- **Funcionamiento**: Calcula el número de visitantes para cada mes del año en curso y devuelve estos conteos en una lista.

Endpoint: **/visitors/current/**

- **Método HTTP**: GET
- **Descripción**: Proporciona el número actual de visitantes. Este endpoint es útil para saber cuántos visitantes hay en un momento dado.
- **Funcionamiento**: Suma los valores actuales de un contador de visitantes y devuelve el total.

Manejo y gestión de cámaras

Endpoint: **/** (con método GET)

- **Método HTTP**: GET
- **Descripción**: Este endpoint proporciona una lista de todas las cámaras registradas en el sistema. Es útil para obtener una visión general de las cámaras disponibles y sus configuraciones.
- **Funcionamiento**: Recupera todas las cámaras de la base de datos y las devuelve en formato JSON. Utiliza una programación funcional (mediante la clase **Stream**) para transformar cada cámara a su representación JSON.

Endpoint: **/** (con método POST)

- **Método HTTP**: POST
- **Descripción**: Permite agregar una nueva cámara al sistema o actualizar una existente basándose en su tipo. Este endpoint es crucial para la administración y configuración de cámaras en la aplicación.
- **Funcionamiento**:
    - Recibe datos de una cámara a través de una solicitud JSON (incluyendo IP, puerto y tipo).
    - Busca en la base de datos si ya existe una cámara con el mismo tipo.
    - Si la cámara no existe, crea una nueva entrada en la base de datos con los datos proporcionados y la devuelve en formato JSON.
    - Si la cámara ya existe, actualiza su dirección IP y puerto con los nuevos datos proporcionados y guarda los cambios en la base de datos, devolviendo la cámara actualizada en formato JSON.

Notificaciones y procesado en tiempo real

Endpoint: **/client_entered/** (con método POST)

- **Método HTTP**: POST
- **Descripción**: Este endpoint se activa cuando un cliente entra en el rango de una cámara o sensor. Se utiliza para registrar la entrada de un cliente, realizar predicciones (probablemente basadas en análisis de imagen o video) y emitir una actualización a través de SocketIO.
- **Funcionamiento**:
    - Invoca la función **make_prediction()** para obtener resultados relacionados con características del cliente (como género y rango de edad).
    - Crea un nuevo registro en la tabla **Entrances** con la marca de tiempo actual, género, intervalo de edad y una Imagen.
    - Incrementa el contador de visitantes mediante la creación de un objeto **Counter** con valor **1**.
    - Emite un mensaje a través de SocketIO al namespace '/visitors', lo que podría utilizarse para actualizar una interfaz de usuario en tiempo real.
    - Devuelve una respuesta JSON con el estado.

Endpoint: **/client_exited/** (con método POST)

- **Método HTTP**: POST
- **Descripción**: Este endpoint se utiliza cuando se detecta la salida de un cliente. Es importante para mantener un recuento actualizado de los visitantes presentes en el establecimiento.
- **Funcionamiento**:
    - Decrementa el contador de visitantes creando un objeto **Counter** con valor **-1**.
    - Al igual que en el endpoint anterior, emite un mensaje a través de SocketIO al namespace '/visitors'. Esto es utilizado para notificar a una interfaz de usuario la salida de un visitante.
    - Devuelve una respuesta JSON indicando que la operación fue exitosa.

Modelos

Descripción General del Modelo Camera

El modelo **Camera** es una representación de datos de cámaras de vigilancia o transmisión dentro de una aplicación. Su propósito principal es almacenar y gestionar la información esencial de las cámaras utilizadas en el sistema, como su ubicación en la red (IP y puerto) y el tipo de cámara. Este modelo es crucial para operaciones que involucran la configuración y el acceso a las cámaras en tiempo real, permitiendo así a la aplicación interactuar con las cámaras para obtener transmisiones de video, realizar monitoreo, o cualquier otra función relacionada.

Descripción de Atributos y Funciones

Atributos

1. **IP (Dirección IP)**: Almacena la dirección IP de la cámara. Este atributo es esencial para identificar la cámara en la red, permitiendo a la aplicación saber dónde conectarse para acceder a la transmisión de video o a la configuración de la cámara.
2. **Puerto**: Indica el puerto a través del cual la cámara está accesible. Este detalle es necesario para establecer una conexión efectiva con la cámara, ya que diferentes cámaras pueden operar en distintos puertos dentro de la misma red.
3. **Tipo de Cámara (cam_type)**: Define el tipo o la categoría de la cámara, como 'frontal', 'lateral', etc. Este atributo ayuda a categorizar las cámaras dentro de la aplicación, permitiendo su uso específico según el contexto o la ubicación.

Funciones

1. **to_json**: Convierte la información de la cámara en un formato JSON. Esta función es especialmente útil para la comunicación de datos a través de APIs o interfaces de usuario, ya que convierte los detalles de la cámara en un formato fácilmente manejable y transmisible.

Descripción General del Modelo Entraces

El modelo **Entrances** es diseñado para registrar y gestionar las entradas de visitantes o clientes en un entorno, como un establecimiento comercial, un evento, o una instalación de seguridad. Cada registro representa una instancia individual de entrada, capturando detalles clave como el momento de la entrada, información demográfica y una imagen asociada.

Descripción de Atributos y Funciones de Entrances

Atributos

1. **Timestamp (Marca de Tiempo)**: Registra el momento exacto en que ocurrió la entrada. Es crucial para el seguimiento temporal de las visitas y para análisis estadísticos basados en el tiempo.

2. **Genre (Género)**: Almacena información sobre el género del visitante. Este dato es útil para análisis demográficos y para comprender la distribución de género de los visitantes.
3. **Age Interval (Intervalo de Edad)**: Indica el rango de edad del visitante. Es importante para análisis demográficos y para entender la composición de edades de los visitantes.
4. **Image (Imagen)**: Un campo para almacenar una imagen, posiblemente del visitante, en formato binario. Este puede ser utilizado para propósitos de identificación o registro visual.

Funciones

1. **to_json**: Esta función convierte la información del registro de entrada en un formato JSON, facilitando la transferencia y el manejo de los datos en aplicaciones web o interfaces de usuario.

Descripción General del Modelo Counter

El modelo **Counter** se utiliza para llevar un conteo acumulativo de eventos, como el número de visitantes que entran o salen de un lugar. Cada registro en este modelo representa un cambio en el conteo total, permitiendo un seguimiento dinámico del número de personas presentes en un momento dado.

Descripción de Atributos de Counter

1. **Value (Valor)**: Representa el cambio en el conteo, que puede ser positivo (por ejemplo, un visitante entra) o negativo (un visitante sale).
2. **Timestamp (Marca de Tiempo)**: Registra el momento en que ocurrió el cambio en el conteo, permitiendo análisis basados en el tiempo de los patrones de tráfico de visitantes.

Descripción de Atributos y Métodos de NetworkCamera

Atributos

1. **IP (Dirección IP)**: La dirección IP de la cámara de red.
2. **Port (Puerto)**: El puerto a través del cual se establece la conexión con la cámara.
3. **Buffer Size (Tamaño de Buffer)**: El tamaño del buffer utilizado al recibir datos del socket, importante para la gestión eficiente de la transmisión de datos.

Métodos

1. **connect**: Establece la conexión socket con la cámara utilizando la IP y el puerto proporcionados. Utiliza el protocolo TCP (SOCK_STREAM) para una transmisión confiable.
2. **close**: Cierra la conexión socket con la cámara.

3. **_receive_frame**: Método privado que gestiona la recepción de un solo cuadro de video de la cámara. Recibe los datos del frame en paquetes, los reconstruye y los decodifica para su uso posterior. En caso de un error, devuelve un cuadro negro por defecto.
4. **get_frame**: Procesa el frame recibido ajustando su tamaño y convirtiéndolo a un formato JPEG. Esto lo hace adecuado para su transmisión o visualización en diferentes contextos.
5. **generator**: Un generador que produce continuamente cuadros de video. Cada cuadro se formatea adecuadamente para su transmisión como parte de una respuesta HTTP, siguiendo el estándar MIME para transmisiones de video en tiempo real.

Limitaciones y soluciones

Limitaciones

1. **Riesgo de Detección en Grupo**: Cuando dos o más personas pasan juntas y muy cerca una de la otra, el sistema actual corre el riesgo de no diferenciarlas y, por ende, de contarlas como una única entidad. Esto limita la fiabilidad del sistema en entornos de alto tráfico o en situaciones donde las personas tienden a moverse en grupos, como en eventos o en horas pico en comercios.
2. **Reconocimiento de Múltiples Personas**: La incapacidad para distinguir entre varias personas en una misma escena significa que el sistema no puede realizar un seguimiento preciso cuando se presentan situaciones de multitudes o aglomeraciones.
3. **Selección de la Persona para Detección de Cara**: Durante la detección facial, si hay varias personas en el encuadre, el sistema puede no garantizar que la cara que se analiza corresponda a la persona que está cruzando la puerta. Esto podría resultar en la captura de datos demográficos incorrectos o incompletos, lo cual afectaría la calidad y el valor de las métricas recopiladas.

Consideraciones y Posibles Soluciones

- **Mejorar la Detección de Múltiples Objetos**: Integrar y entrenar algoritmos de visión computarizada que sean capaces de detectar y diferenciar múltiples objetos simultáneamente podría ser una solución a esta limitación.
- **Implementar Mecanismos de Control de Flujo**: Establecer mecanismos físicos o digitales para controlar el flujo de personas y asegurar que pasen de una en una mejoraría la precisión de la detección sin necesidad de cambiar el sistema actual.

- **Sistemas de Detección de Zona**: Implementar un sistema que primero detecte la presencia de una persona en una zona específica antes de la puerta y luego active la cámara de perfil únicamente cuando se tiene la certeza de que una sola persona está en la zona de detección.
- **Feedback Visual o Auditivo**: Proporcionar indicaciones visuales o auditivas para guiar a las personas a pasar de una en una podría ser una solución práctica y de bajo costo.

Posibles Ampliaciones

Seguimiento y Detección de Múltiples Personas

Implementación: Para lograr el seguimiento y detección de múltiples personas, se podrían utilizar algoritmos de visión computarizada más avanzados que diferencien e identifiquen a varias personas en una imagen o vídeo simultáneamente.

Desarrollo: La integración de estas capacidades requeriría entrenar un modelo con un conjunto de datos extenso y diverso para garantizar precisión y robustez en diversos escenarios. Además, el sistema debería ser capaz de manejar y procesar los datos adicionales generados por el seguimiento de múltiples objetivos, lo cual podría implicar una actualización de la infraestructura de hardware y red para soportar un mayor flujo de datos.

Impacto: Esta ampliación permitiría un análisis demográfico más complejo y detallado, así como mejoras en las estrategias de marketing y seguridad al poder rastrear flujos de movimiento y patrones de comportamiento dentro del establecimiento.

Conectar las cámaras a través de una aplicación móvil

Implementación: Desarrollar una aplicación móvil que permita acceder a las transmisiones de las cámaras y visualizar las métricas en tiempo real. Esto podría realizarse a través de tecnologías como React Native para la construcción de la aplicación y el uso de WebSockets para mantener una comunicación en tiempo real con el servidor central.

Desarrollo: La aplicación móvil requeriría de una interfaz de usuario intuitiva y eficiente que permita a los usuarios interactuar con las cámaras y los datos. Además, sería necesario implementar medidas de seguridad robustas para proteger la transmisión de video y los datos sensibles de los usuarios.

Impacto: La movilidad y la facilidad de acceso a los datos y las imágenes en tiempo real desde cualquier ubicación ofrecerían una flexibilidad y conveniencia significativas para los usuarios, tales como gerentes o equipos de seguridad, que necesiten mantenerse informados sobre la actividad en sus negocios.

Tecnologías Usadas

1. **Poetry**: Es una herramienta moderna de gestión de dependencias y entornos virtuales en Python. Facilita la instalación precisa de paquetes y sus versiones, garantizando la consistencia del entorno de desarrollo y producción.
2. **Flask**: Un framework web ligero y flexible en Python que proporciona las herramientas necesarias para construir APIs REST de forma rápida y sencilla, permitiendo la creación de servicios web escalables.
3. **OpenCV (Open Source Computer Vision Library)**: Una biblioteca robusta de código abierto especializada en algoritmos de visión por computadora y procesamiento de imágenes, que permite implementar captura y análisis de imágenes en tiempo real con alto rendimiento.
4. **Socket.IO**: Es una biblioteca que permite la comunicación en tiempo real entre clientes y servidores web mediante WebSockets, facilitando actualizaciones instantáneas y bidireccionales.
5. **SQLite**: Un sistema de gestión de bases de datos relacional ligero, que se destaca por su simplicidad y eficiencia, ideal para dispositivos con recursos limitados o aplicaciones que requieren una solución de almacenamiento de datos más sencilla.
6. **React**: Una biblioteca de JavaScript para construir interfaces de usuario de forma eficiente y con capacidad de reacción, que permite crear vistas ricas y dinámicas en aplicaciones web.
7. **MediaPipe**: Un framework de procesamiento de medios cruzado que ofrece soluciones de vanguardia para la detección facial y el seguimiento, ideal para aplicaciones interactivas en tiempo real.
8. **YoloV8 (You Only Look Once)**: La última iteración de la serie YOLO, una red neuronal de detección de objetos altamente eficiente que permite la identificación y clasificación en tiempo real de personas y objetos dentro de un entorno.

Repositorios

1. Repositorio Conteo y Detección de Caras. vc-trabajo-final-cliente-deteccion-clientes.
2. Repositorio Clasificador de Caras. vc-trabajo-final-cliente-person-qualifier.
3. Repositorio Backend. vc-trabajo-final-backend.
4. Repositorio Frontend. vc-trabajo-final-frontend.