Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

Unidad Iztapalapa

Construcción de una base de datos de videos aéreos y su análisis vía herramientas de IA

Presentado por:
Jorge Rafael Martínez Buenrostro

Asesora: Dra. Elizabeth Pérez Cortés

México, CDMX, a 11 de diciembre de 2025

Contenido

Lista de Figuras

Lista de Códigos

Capítulo 1

Introducción del Proyecto

La simulación de una red de comunicaciones en donde intervienen dispositivos personales de comunicación requiere contar con modelos que representen fielmente los patrones de movimiento de las personas. De otra manera, la utilidad de las conclusiones que se puedan obtener de esa simulación es limitada. Para avanzar hacia la definición de un modelo de movilidad humana grupal, se propone la construcción de una base de datos de videos aéreos —capturados por un dron— y su análisis mediante herramientas de IA, lo que permitirá determinar algunas características de la movilidad de interés.

En cuanto al proceso de diseño de un modelo de movilidad, es fundamental contar con una fuente de trazas que permita construir el modelo extrayendo las características necesarias.

El objetivo principal del proyecto es contar con una caracterización de los grupos humanos que se desplazan juntos, identificando las características estadísticas de los mismos. Entre los logros alcanzados se encuentran: la detección y el rastreo de individuos dentro de un video; la identificación de los patrones de movimiento y las interacciones entre individuos para la obtención de grupos; y la extracción de características relevantes sobre la movilidad y las interacciones grupales.

Capítulo 2

Marco teórico

2.1 Elementos de un modelo de movilidad

Un modelo de movilidad grupal debe considerar los siguientes elementos:

- **Agrupación**: Identificación de conjuntos de individuos que se desplazan de forma coordinada.
- **Trayectorias**: Patrones de movimiento a nivel individual y grupal.
- **Interacciones**: Dinámicas dentro del grupo y entre grupos.
- **Características estadísticas**: Velocidad, densidad, cohesión, dirección y separación entre individuos.

Recordando que una trayectoria individual tiene tres elementos básicos: *puntos de recorrido*, *tiempos de pausa* y *longitud de vuelo*. Dentro del contexto de este proyecto las trayectorias individuales y grupales no cuentan con puntos de recorrido como tal, ya que el área de grabación es un paso común para comunicar diferentes zonas dentro de la universidad. Por lo que se considera que no hay puntos de recorrido; sin embargo, un punto que es interesante es saber si los grupos se detienen en algún punto del recorrido, y de ser así por cuanto tiempo lo hacen. Bajo esta premisa este comportamiento es el tiempo de pausa.
Otra consideración importante es que cuando un grupo cambia su densidad de individuos, se considera que el grupo original deja de existir y uno o varios grupos con la nueva densidad son creados.

Capítulo 3

Objetivos y metodología

3.1 Objetivos

El objetivo principal del proyecto es contar con una caracterización de los grupos de humanos que se desplazan juntos. Identificando las características estadísticas de los mismos.

Los objetivos particulares son:

- Construir una base de datos de videos aéreos de grupos humanos.
- Usar un modelo de IA que permita identificar y caracterizar los grupos humanos en los videos.
- Analizar los patrones de movimiento y las interacciones entre los grupos humanos.

3.2 Metodología

El proceso se dividió en las siguientes etapas:

1. Captura y almacenamiento: Grabación de videos con un dron y organizacion en base de datos.

2. Preprocesamiento: Conversión de videos a frames y almacenamiento de metadatos.

3. Detección y seguimiento de individuos: Uso de YOLO para identificar y rastrear personas en los frames.

4. Extracción de características: Cálculo de atributos de movilidad grupal.

Capítulo 4

Desarrollo

El desarrollo de este proyecto se estructuró en cuatro fases interconectadas, optimizadas para el análisis eficiente y escalable de grandes volúmenes de datos de video. La principal modificación a la metodología inicial fue la transición de un procesamiento basado en frames a un procesamiento directo sobre el flujo de video (stream), eliminando cuellos de botella de E/S (Input/Output).

4.1 Captura y almacenamiento

Esta etapa se encarga de la grabación física de los videos aéreos y su almacenamiento inicial en el servidor. Los videos son capturados mediante un dron en áreas de tránsito común dentro del campus universitario, y posteriormente son transferidos a un directorio designado para su procesamiento. La organización inicial de los archivos es fundamental para facilitar el análisis automatizado posterior.

4.2 Preprocesamiento y orquestación

En la metodología inicial, esta fase incluía la conversión intensiva de videos a secuencias de imágenes (frames). Para maximizar la eficiencia y reducir la latencia, esta etapa fue refactorizada, reemplazando la conversión explícita por un procesamiento directo concurrente sobre el archivo de video.

4.2.1 Implementación de la Concurrencia

Para manejar múltiples archivos de video simultáneamente, se implementó un sistema basado en `concurrent.futures.ThreadPoolExecutor` en Python. Este orquestador es el punto de entrada que:

- Escanea un directorio de entrada en busca de todos los archivos `.mp4`.
- Despacha cada video a un *worker thread* dedicado para su análisis completo.

- Utiliza la potencia de la GPU (NVIDIA RTX A5000) de manera eficiente, limitando el número de workers concurrentes a 4, un número óptimo para balancear la carga de VRAM del modelo YOLOv11x y el rendimiento del sistema.

4.3 Detección y seguimiento de individuos (Tracking Unificado)

Esta etapa es la base del análisis, encargada de identificar y mantener la identidad de cada persona a lo largo de las secuencias de video.

4.3.1 Arquitectura Híbrida YOLO + DeepSORT

Para lograr la detección y el seguimiento con alta precisión, se utilizó una arquitectura de dos etapas:

- **Detección de Objetos (YOLOv11x):**
  - Se empleó la versión *Extra Large* (`yolo11x.pt`) del modelo YOLOv11 (You Only Look Once). Este modelo fue seleccionado por su equilibrio entre alta precisión de localización (mAP) y velocidad de inferencia, esencial para el procesamiento en tiempo real.
  - El modelo fue configurado para detectar exclusivamente la clase `persona` (clase 0), optimizando el consumo de recursos.
- **Rastreo por Re-Identificación (DeepSORT):**
  - Los *bounding boxes* generados por YOLO son pasados al algoritmo DeepSORT (Deep Simple Online and Real-time Tracking). DeepSORT asigna un ID de seguimiento (*Track ID*) persistente a cada individuo.
  - El algoritmo se basa en el filtro de Kalman para predecir la posición futura y utiliza una red neuronal (como MobileNet, según la configuración) para extraer características de apariencia (*features*) que permiten re-identificar a una persona que ha sido ocluida o ha salido brevemente del campo de visión.

4.3.2 Persistencia de Datos

El resultado de este proceso (el *Track ID* y las coordenadas (x_1, y_1, x_2, y_2) para cada persona en cada frame) se guarda inmediatamente en la base de datos PostgreSQL en la tabla `FrameObjectDetection`.

4.4 Extracción y caracterización de grupos

Esta fase final emplea los datos de seguimiento guardados para identificar patrones de interacción social (grupos) y generar métricas estadísticas clave.

4.4.1 Algoritmo de Agrupamiento PDF

La lógica de agrupación se implementa a través de la clase `GroupTracker`, que replica el algoritmo de agrupamiento basado en:

- **Proximidad Espacial:** Se considera que dos individuos están en proximidad si la distancia entre sus centroides (calculada en píxeles) es menor a un umbral predefinido (ej. 100 píxeles).
- **Persistencia Temporal (τ):** Para que una interacción sea clasificada como un grupo, la proximidad debe mantenerse durante un número mínimo de frames (ej. 15 frames). Esto evita la detección de interacciones fugaces o cruces accidentales.

El algoritmo utiliza teoría de grafos, donde los individuos son nodos y los pares estables son aristas, para identificar los componentes conexos (grupos de 2 o más personas) en cada frame.

4.4.2 Análisis de Métricas y Reporte Final

Una vez que el seguimiento y el agrupamiento de un video han finalizado, el *pipeline* procede a la generación del reporte. Esta fase consulta los datos recién almacenados en las tablas `GroupDetection` y `FrameObjectDetection` para calcular las siguientes estadísticas:

- **Identificación:** Número total de individuos únicos rastreados.
- **Grupos Únicos:** Cantidad total de identificadores de grupo diferentes encontrados.
- **Tamaño Promedio:** Tamaño promedio de los grupos detectados.
- **Duración de Grupos:** Identificación de los grupos más persistentes (número de frames en los que el grupo fue detectado).
- **Métricas de Grupo:** Aunque no se muestran en el reporte simple, el modelo de datos está diseñado para almacenar métricas de movilidad grupal, como la dispersión (varianza de las distancias entre miembros) y la velocidad promedio del centroide del grupo.

Todos los hallazgos estadísticos y el resumen del proceso se guardan en un documento de texto plano (`.txt`) con la nomenclatura `REPORTE_{video_name}.txt`, cumpliendo con el requisito final del proyecto.

Capítulo 5

Resultados

El siguiente capítulo presenta y analiza los resultados obtenidos tras la ejecución del *pipeline* de detección, rastreo y análisis grupal sobre la totalidad del conjunto de videos aéreos.

El sistema unificado de análisis (YOLOv11x + DeepSORT + Algoritmo PDF) fue ejecutado de forma concurrente sobre 8 secuencias de video, demostrando la eficacia del enfoque de procesamiento directo sobre el *stream* de video y la capacidad del sistema para escalar el análisis de manera automatizada.

5.1 Rendimiento y Volumen de Datos Procesados

El proceso concurrente analizó un volumen significativo de datos, validando la estabilidad y eficiencia del *pipeline* refactorizado.

Cuadro 5.1: Métricas generales del procesamiento de videos

Métrica	**Valor Total**
Total de Videos Analizados	8
Duración Total Analizada (Frames)	**236,785 frames**
Duración Total Estimada (Horas)	$\approx$ 2,19 horas (@30FPS)
Total de Identidades Únicas (Track IDs)	**4,097 personas**
Total de Grupos Únicos Detectados	**952 grupos**

El sistema logró procesar eficientemente más de 230,000 *frames* sin recurrir a la conversión previa a imágenes estáticas, confirmando la optimización del proceso de E/S. La configuración concurrente (`--max_workers 4`) en la NVIDIA RTX A5000 permitió una alta tasa de procesamiento por hora.

5.2 Resultados de Detección y Rastreo de Individuos

La combinación de YOLOv11x y DeepSORT demostró ser robusta para el seguimiento de personas en el entorno aéreo, logrando identificar y mantener la identidad de 4,097 personas únicas en el conjunto de datos.

Se observó una clara correlación entre la fecha de grabación y la densidad poblacional, lo que impactó directamente el volumen de detección:

- **Baja Densidad (05-02):** En los videos capturados el 2 de mayo (V1, V2, V3), el sistema rastreó un total combinado de **452 personas únicas**.
- **Alta Densidad (05-14 y 05-16):** Los días 14 y 16 de mayo registraron una afluencia mucho mayor, con un total de **1,945** y **1,700** personas únicas rastreadas, respectivamente.

Este resultado confirma la capacidad del modelo para gestionar entornos de tráfico bajo y alto, un factor clave para la generación de modelos de movilidad precisos.

5.3 Caracterización de Grupos Humanos

El algoritmo de agrupamiento, basado en proximidad y persistencia temporal, identificó un total de 952 agrupaciones sociales únicas.

5.3.1 Volumen de Agrupamiento

Los días de mayor densidad poblacional (14 y 16 de mayo) fueron los que generaron el mayor número de interacciones sociales estables:

Cuadro 5.2: Distribución de grupos detectados por fecha de captura

Fecha de Captura	**Videos**	**Personas Únicas**	**Grupos Detectados**	**Ratio Grupos/Personas**
05-02-2025	3	452	107	1:4.2
05-14-2025	3	1,945	382	1:5.1
05-16-2025	2	1,700	463	1:3.7
Promedio General				**1:4.3**

El ratio promedio indica que por cada **4.3 individuos únicos** detectados en el área de estudio, se formó al menos **un grupo** estable a lo largo de las secuencias. Este dato establece una métrica fundamental para modelar la interacción social en el ambiente observado.

5.3.2 Persistencia de las Interacciones Sociales

Un análisis de los grupos más duraderos (Top 5 en cada video) reveló una gran variabilidad en la persistencia de las interacciones:

- **Grupos Altamente Persistentes:** En los videos de alta densidad (05-14 y 05-16), se detectaron grupos con duraciones excepcionalmente largas, destacando el **Grupo 84** (05-14-V2) con **6,093 frames** (aproximadamente 3 minutos y 23 segundos). Estos grupos representan interacciones de largo plazo, como personas permaneciendo estáticas, o viajando juntas a lo largo de la grabación.
- **Grupos de Interacción Breve:** En contraste, en el set de videos del 05-02 (baja densidad), el grupo más duradero solo alcanzó **467 frames** (aproximadamente 15 segundos), y el grupo menos duradero de la muestra fue de solo **7 frames** (Grupo 2, 05-02-V2), lo que es cercano al umbral de persistencia temporal ($\tau = 15$ frames) necesario para la confirmación.

Estos resultados sugieren que el *pipeline* es capaz de diferenciar entre **encuentros momentáneos** y **agrupaciones sociales estables**, siendo un elemento crucial para la simulación de redes de comunicación donde la duración del contacto es crítica.

Capítulo 6

Conclusiones y Trabajo futuro

6.1 Conclusiones del Proyecto

El proyecto ha logrado implementar y validar exitosamente un *pipeline* integral para la construcción de una base de datos de videos aéreos y el análisis automatizado de la movilidad y el comportamiento grupal. Los objetivos planteados han sido cubiertos, destacando las siguientes conclusiones clave:

6.1.1 Eficiencia y Escalabilidad del Sistema

La refactorización del *pipeline* hacia un procesamiento directo y concurrente de videos (en lugar de la previa conversión a *frames*) fue la decisión más crítica para la eficiencia.

- El sistema demostró ser altamente escalable, procesando 236,785 *frames* de video (equivalente a más de 2.19 horas) en un tiempo óptimo gracias al uso de *threading* concurrente y la capacidad de la GPU.
- La arquitectura modular permite la fácil integración o reemplazo de componentes clave (ej., cambiar el modelo YOLO o el algoritmo de *tracking*) sin necesidad de reescribir la lógica de orquestación.

6.1.2 Robustez del Análisis Detección-Rastreo

La implementación del enfoque híbrido YOLOv11x + DeepSORT garantizó la generación de trayectorias individuales con alta fidelidad, un requisito fundamental para el análisis de grupos.

- Se logró rastrear un total de 4,097 identidades únicas, incluso en escenarios de alta densidad poblacional.
- El rastreo fue lo suficientemente persistente para mantener la identidad individual a través de leves oclusiones o cruces, proporcionando la base de datos necesaria para la simulación de modelos de movilidad.

6.1.3 Valoración del Modelo de Agrupamiento

El algoritmo de agrupamiento, basado en la proximidad y persistencia temporal ($\tau = 15$ *frames*), demostró ser eficaz para caracterizar el comportamiento social.

- Se detectaron 952 grupos únicos, estableciendo una métrica de interacción: por cada 4.3 individuos únicos, se formó un grupo estable en el área de estudio.
- El análisis de la persistencia de los grupos reveló patrones importantes: mientras que los grupos de baja densidad tienden a ser fugaces (duraciones máximas de ≈15 segundos), los grupos en escenarios de alta densidad son extremadamente persistentes (duraciones de hasta ≈3.4 minutos), lo que apunta a la existencia de grupos estáticos o estructuras de tráfico lento en las áreas observadas.
- La capacidad de extraer métricas (como la dispersión y la duración) convierte la base de datos de video aéreo en una herramienta cuantitativa para calibrar modelos de movilidad y comunicación.

6.2 Trabajo Futuro

El proyecto sienta bases sólidas y abre varias líneas de investigación y desarrollo para extender su utilidad:

6.2.1 Extensión del Análisis Comportamental

- **Identificación de roles y patrones:** Implementar análisis para diferenciar si un individuo está quieto, caminando solo o esperando. Esto añadiría una capa semántica al estado de la persona, crucial para modelos de comunicación que penalizan la movilidad.
- **Análisis de trayectorias:** Desarrollar módulos para calcular la trayectoria promedio de los grupos y el vector de velocidad promedio, enriqueciendo la caracterización del movimiento colectivo más allá de solo la existencia del grupo.

6.2.2 Optimización y Despliegue

- **Integración de base de datos espaciotemporal:** Migrar la base de datos a un esquema que aproveche extensiones espaciales (como PostGIS) para permitir consultas complejas basadas en geometría y tiempo (ej., "¿Cuántos grupos pasaron por el área X entre las 10:00 y las 10:15?").
- **Aceleración en GPU:** Investigar librerías de aceleración de video basadas en GPU (como NVIDIA's V-SDK o cuStream) para el decodificador de video, trasladando completamente el cuello de botella de E/S y *decoding* a la GPU.

6.2.3 Validación y Calibración del Modelo

- **Correlación con datos reales:** Utilizar los datos de agrupamiento obtenidos para calibrar y validar el modelo de movilidad de la red de comunicaciones. Por ejemplo, ajustar los parámetros de encuentro y desconexión en la simulación basados en las duraciones de grupo observadas (los 952 grupos detectados).
- **Estudio de robustez:** Evaluar cómo los cambios en los hiperparámetros del algoritmo de agrupamiento (τ y la distancia de proximidad) modifican la cantidad y duración de los grupos, permitiendo establecer los umbrales óptimos para diferentes escenarios de afluencia.

Apéndice A

Uso de Docker y Docker Compose

En la sección **??** se establece como requisito el uso de Docker y Docker Compose para la ejecución del proyecto. A continuación, se detallan las instrucciones necesarias para su instalación, ya que ambas herramientas son fundamentales para la implementación. Además, se describe el archivo `docker-compose.yml`, el cual permite crear un contenedor que incluye todas las dependencias requeridas para el correcto funcionamiento del sistema.

A.1 ¿Qué son Docker y Docker Compose?

Docker es una plataforma de virtualización ligera que permite desarrollar, empaquetar y ejecutar aplicaciones en contenedores aislados. Un contenedor incluye el código, las dependencias y configuraciones necesarias para que la aplicación se ejecute de manera consistente en cualquier entorno. Esto facilita la portabilidad, escalabilidad y despliegue de software.

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones multicontenedor mediante archivos de configuración YAML. A través de un solo archivo `docker-compose.yml`, es posible especificar los servicios, redes y volúmenes que componen una aplicación, simplificando así su orquestación.

Estas herramientas son fundamentales en este proyecto para garantizar que el entorno de ejecución sea replicable y controlado, independientemente del sistema operativo o configuración local del usuario.

A.2 Instalación en Linux (Ubuntu/Debian)

Para instalar Docker y Docker Compose en un sistema Linux basado en Debian o Ubuntu, siga los siguientes pasos:

1. Actualizar los paquetes del sistema:

```
1 sudo apt update
2 sudo apt upgrade
```

Código A.1: Actualizar el sistema.

2. Instalar Docker:

```
sudo apt install docker.io
sudo systemctl enable docker
sudo systemctl start docker
```

Código A.2: Instalar Docker.

3. Verificar que Docker está instalado correctamente:

```
docker --version
```

Código A.3: Verificar instalación de Docker.

4. Instalar Docker Compose:

```
sudo apt install docker-compose
```

Código A.4: Instalar Docker Compose.

5. Verificar la instalación:

```
docker-compose --version
```

Código A.5: Verificar instalación de Docker Compose.

A.3 Instalación en Windows

Para instalar Docker y Docker Compose en Windows, se recomienda utilizar Docker Desktop, que incluye ambas herramientas de forma integrada.

1. Acceder al sitio oficial: https://www.docker.com/products/docker-desktop/
2. Descargar el instalador correspondiente para Windows.
3. Ejecutar el instalador y seguir el asistente de instalación.
4. Reiniciar el sistema si es necesario.
5. Verificar que Docker y Docker Compose estén correctamente instalados desde la terminal de Windows (PowerShell o CMD):

```
docker --version
docker-compose --version
```

Código A.6: Verificar instalación de Docker y Docker Compose.

Nota: Docker Desktop requiere que la virtualización esté habilitada en la BIOS del sistema. También es necesario contar con Windows 10 o superior.

A.4 Descripción del archivo `docker-compose.yml`

El archivo `docker-compose.yml` permite definir y configurar el entorno de ejecución del proyecto utilizando contenedores de Docker. A continuación, se presenta su contenido y una explicación de cada uno de sus elementos:

```
version: "3.8"

services:
group-analysis:
image: nvidia/cuda:12.3.1-base-ubuntu22.04
container_name: group-analysis
tty: true
stdin_open: true
deploy:
resources:
reservations:
devices:
- driver: nvidia
count: all
capabilities: [gpu]
volumes:
- ./:/app
- ../Videos:/app/Videos:cached
working_dir: /app
environment:
- NVIDIA_VISIBLE_DEVICES=all
- NVIDIA_DRIVER_CAPABILITIES=compute,utility
- PYTHONPATH=/app
- DB_HOST=postgres
- DB_PORT=5432
- DB_NAME=group_analysis_db
- DB_USER=postgres
- DB_PASSWORD=postgres123
command: >
sh -c "
apt-get␣update␣&&
apt-get␣install␣-y␣python3-pip␣libgl1-mesa-glx␣libglib2.0-0␣
    postgresql-client␣&&
update-alternatives␣--install␣/usr/bin/python␣python␣/usr/bin/
    python3␣1␣&&
pip3␣install␣--no-cache-dir␣torch␣torchvision␣--index-url␣https
    ://download.pytorch.org/whl/cu121␣&&
pip3␣install␣--no-cache-dir␣-r␣requirements.txt␣&&
echo␣'Esperando␣a␣que␣PostgreSQL␣es ␣listo...'␣&&
until␣pg_isready␣-h␣postgres␣-p␣5432␣-U␣postgres;␣do
echo␣'PostgreSQL␣no␣est␣listo␣-␣esperando...'
sleep␣2
done␣&&
echo␣'PostgreSQL␣est␣listo'␣&&
```

```
pip3 install -e . &&
python3 /app/src/database/create_models.py &&
echo 'Contenedor inicializado correctamente' &&
tail -f /dev/null
"
depends_on:
postgres:
condition: service_healthy
networks:
- data-network

postgres:
image: postgres:15-alpine
container_name: postgres-group-analysis
restart: always
environment:
POSTGRES_DB: group_analysis_db
POSTGRES_USER: postgres
POSTGRES_PASSWORD: postgres123
ports:
- "5433:5432"
volumes:
- postgres_data:/var/lib/postgresql/data
networks:
- data-network
healthcheck:
test: ["CMD-SHELL", "pg_isready -U postgres"]
interval: 10s
timeout: 5s
retries: 5
start_period: 30s

adminer:
image: adminer:latest
container_name: adminer-group-analysis
restart: always
ports:
- "8081:8080"
depends_on:
- postgres
networks:
- data-network

volumes:
postgres_data:

networks:
data-network:
driver: bridge
```

Código A.7: Archivo docker-compose.yml

A continuación se explica el propósito de cada sección:

- **Información General**
  - Versión: Docker Compose 3.8
  - Tres servicios: group-analysis, postgres, adminer
  - Una red personalizada: data-network
  - Un volumen persistente: postgres_data
- **Servicio: group-analysis**
  - **Imagen**: nvidia/cuda:12.3.1-base-ubuntu22.04
  - **Propósito**: Contenedor principal para análisis con GPU
  - **Configuración GPU**:
    - Acceso a todas las GPUs disponibles
    - Capacidades: compute y utility
  - **Volúmenes**:
    - Monta directorio actual en /app
    - Monta ../Videos en /app/Videos (con cache)
  - **Variables de entorno**:
    - Configuración NVIDIA para GPU
    - Variables de conexión a PostgreSQL
    - PYTHONPATH establecido en /app
  - **Comando de inicialización**:
    - Instala dependencias del sistema
    - Instala PyTorch con soporte CUDA 12.1
    - Instala requirements.txt
    - Espera a que PostgreSQL esté disponible
    - Instala el proyecto en modo desarrollo
    - Crea modelos de base de datos
    - Mantiene el contenedor ejecutándose
  - **Dependencias**: Espera a que postgres esté saludable
- **Servicio: postgres**
  - **Imagen**: postgres:15-alpine (versión ligera)
  - **Base de datos**: group_analysis_db
  - **Credenciales**: usuario: postgres, contraseña: postgres123
  - **Puertos**: 5433:5432 (acceso externo en 5433)
  - **Volumen persistente**: postgres_data para almacenamiento
  - **Healthcheck**: Verifica cada 10s si PostgreSQL está listo
  - **Reinicio automático**: siempre

- **Servicio: adminer**
  - **Imagen**: adminer:latest
  - **Propósito**: Interfaz web para administrar PostgreSQL
  - **Puertos**: 8081:8080 (acceso en http://localhost:8081)
  - **Dependencias**: Requiere que postgres esté ejecutándose
- **Configuración de Redes**
  - **Red**: data-network (driver bridge)
  - Todos los servicios comparten la misma red
  - Comunicación interna por nombres de servicio
- **Configuración de Volúmenes**
  - **Volumen**: postgres_data
  - Propósito: Persistencia de datos de PostgreSQL
  - Almacenamiento: /var/lib/postgresql/data en el contenedor
- **Flujo de Ejecución**
  a. Se inicia la red data-network
  b. Se crea el volumen postgres_data
  c. Se inicia PostgreSQL con healthcheck
  d. Cuando PostgreSQL está saludable, se inicia group-analysis
  e. Se inicia Adminer (depende solo de que PostgreSQL esté corriendo)
  f. El contenedor group-analysis ejecuta su script de inicialización
- **Uso de GPU**
  - Requiere NVIDIA Container Toolkit instalado en el host
  - Driver NVIDIA compatible
  - GPUs visibles para contenedores Docker
- **Puertos Externos**
  - PostgreSQL: 5433 (mapeado al 5432 interno)
  - Adminer: 8081 (mapeado al 8080 interno)
  - No hay puertos expuestos para group-analysis
- **Aplicación Principal**
  - Framework: PyTorch con CUDA
  - Propósito: Análisis de video/grupos (computer vision)
  - Dependencias: OpenGL (libgl1-mesa-glx) para visualización
  - Estructura: Proyecto Python instalable (setup.py)

Esta configuración crea un entorno completo de desarrollo que incluye la aplicación de análisis de datos, una base de datos PostgreSQL y una herramienta de administración web, todos interconectados y fácilmente replicables en cualquier sistema que tenga Docker instalado.

A.5 Scripts de control del contenedor

Para facilitar el manejo del contenedor durante el desarrollo del proyecto, se han creado tres scripts auxiliares en Bash que automatizan las operaciones más comunes: iniciar, reiniciar y detener el contenedor.

A.5.1 `start_container.sh`

Este script verifica si el contenedor `data-analysis` ya se encuentra en ejecución. En caso de que no esté activo, lo inicia utilizando `docker-compose up -d`. Posteriormente, ejecuta el archivo `main.py` dentro del contenedor.

```
1  #!/bin/bash
2
3  if ! docker ps --filter "name=^/group-analysis$" --
       filter "status=running" | grep -q data-analysis;
       then
4  echo "Contenedor␣no␣esta␣corriendo.␣Levantando␣con␣
       docker-compose..."
5  docker-compose up -d
6  echo "Esperando␣que␣se␣instalen␣las␣dependencias..."
7  while ! docker exec data-analysis pip show colorama &>
       /dev/null; do
8  sleep 2
9  done
10 echo "Dependencias␣instaladas␣correctamente."
11 else
12 echo "Contenedor␣ya␣esta␣corriendo.␣Usando␣instancia␣
       existente."
13 fi
14
15 echo "Ejecutando␣script..."
16 docker exec -it group-analysis python3 /app/src/main.py
```

Código A.8: Script para iniciar el contenedor.

A.5.2 `restart_container.sh`

Este script reinicia completamente el contenedor (equivalente a detenerlo y volverlo a levantar), lo cual resulta útil cuando se han modificado archivos como `requirements.txt` o `setup.py`. Tras reiniciar, vuelve a ejecutar el archivo principal del proyecto.

```
#!/bin/bash
docker restart group-analysis
sleep 2
docker exec -it group-analysis python3 /app/src/main.py
```

Código A.9: Script para reiniciar el contenedor.

A.5.3 `stop_container.sh`

Este script detiene y elimina el contenedor junto con los volúmenes asociados. Debe utilizarse con precaución, ya que elimina todas las dependencias instaladas en el entorno del contenedor. Solo es necesario en casos donde se requiere limpiar completamente el entorno.

```
#!/bin/bash
docker-compose down --volumes
```

Código A.10: Script para detener y eliminar el contenedor y sus volúmenes.

A.6 Proceso de uso y desarrollo del contenedor

A continuación se describe el flujo recomendado para desarrollar y ejecutar el sistema dentro del contenedor de Docker:

1. Verifique que Docker y Docker Compose están instalados (Apéndice A.6).
   Nota para usuarios de Windows: Si se utiliza Windows como sistema operativo, se deben usar los archivos con extensión `.bat` en lugar de `.sh`, y deben ser ejecutados desde la terminal de Windows (por ejemplo, CMD o PowerShell).
2. Asigne permisos de ejecución a los scripts:

```
chmod +x start_container.sh
   restart_container.sh stop_container.sh
```

Código A.11: Dar permisos de ejecución a los scripts.

3. Para iniciar el contenedor y ejecutar el proyecto con los cambios más recientes del código fuente:

```
./start_container.sh
```

Código A.12: Iniciar contenedor y ejecutar el proyecto.

4. Si se realizan cambios en las dependencias o archivos de configuración del entorno (como `requirements.txt`), utilice:

```
./restart_container.sh
```

Código A.13: Reiniciar el contenedor completamente.

5. Para detener el contenedor y eliminar todos los volúmenes asociados:

```
./stop_container.sh
```

Código A.14: Eliminar el contenedor y limpiar el entorno.

Este conjunto de scripts permite un desarrollo ágil dentro del contenedor, ya que los cambios realizados en el código fuente local se reflejan de inmediato gracias al uso de `volumes`. Además, se reduce la necesidad de ejecutar manualmente comandos repetitivos, facilitando el trabajo del usuario final y asegurando la correcta ejecución del proyecto.

Para poder ejecutar un script dentro del contenedor, y dejarlo corriendo en segundo plano sin necesidad de una conexión SSH activa. Hay que instalar tmux dentro del contenedor, como se muestra a continuación:

```
sudo docker exec -u root -it group-analysis bash
apt update && apt install tmux -y
```

Una vez instalado tmux dentro del contenedor se pueden usar los siguiente comandos:

1. **Crear una sesión tmux**

```
sudo docker exec -it group-analysis tmux
    new-session -d -s {SessionName} 'cd /
    app && python3 {PythonScript}.py'
```

2. **Listas las sesiones tmux activas**

```
sudo docker exec -it group-analysis tmux
    list-sessions
```

3. **Conectarse a la sesión tmux creada** Para poder salir de la sesión sin detener el proceso, presione `Ctrl + b` y luego `d` (de detach).

```
sudo docker exec -it group-analysis tmux
    attach -t {SessionName}
```

Referencias

[1] Autor referencia 1

[2] Autor referencia 2

[3] Autor referencia 3