UNIVERSIDAD MAYOR REAL Y PONTIFICIA SAN FRANCISCO XAVIER DE CHUQUISACA
FACULTAD DE TECNOLOGIA

ALGORITMOS SORT Y TRACKING APLICADO A DETECCION DE OBJETOS

Inteligencia Artificial II

Nombre: Daniel Andree Arancibia Aguilar

Carrera: Ingeniería en Ciencias de la computación

Docente: Ingeniero Carlos Walter Pacheco Lora

Introducción al modelo YOLOV11:
YOLOv11 es una versión evolucionada del modelo YOLO (You Only Look Once), diseñado específicamente para la detección de objetos en imágenes y videos en tiempo real. Forma parte de una familia de modelos ampliamente utilizados en visión por computadora por su capacidad de combinar precisión y velocidad en una sola arquitectura.

Características principales del YOLOv11:

- **Arquitectura optimizada:** Integra mejoras en la extracción de características, procesamiento multiescala y predicción final.
- **Diseño modular:** Compuesto por tres módulos principales: **Backbone**, **Neck**, y **Head**, que permiten un flujo eficiente y organizado.
- **Capacidad multiescala:** Detecta objetos de diferentes tamaños mediante predicciones especializadas en tres escalas (pequeña, mediana y grande).
- **Escalabilidad:** Ofrece parámetros ajustables como depth_multiple y width_multiple, que permiten adaptar el modelo a diferentes recursos de hardware y necesidades.

2. ¿Para qué sirve YOLOv11?
YOLOv11 se utiliza principalmente en tareas de **detección de objetos en tiempo real**, que consisten en identificar la posición (mediante cajas delimitadoras) y la clase (categoría) de uno o más objetos presentes en una imagen o un video.
Aplicaciones principales:

1. **Seguridad y vigilancia:**
    - Detección de intrusos, reconocimiento facial, y análisis en sistemas de videovigilancia.
2. **Automoción:**
    - Uso en vehículos autónomos para detectar peatones, señales de tráfico, y otros vehículos.
3. **Medicina:**
    - Localización de anomalías en imágenes médicas, como tumores en radiografías o resonancias magnéticas.
4. **Agricultura:**
    - Monitoreo de cultivos mediante drones para identificar plagas o anomalías.
5. **Industria y logística:**
    - Inspección automatizada en líneas de ensamblaje o clasificación de productos en almacenes.
6. **Tecnología de consumo:**
    - Implementación en aplicaciones móviles, cámaras inteligentes, y herramientas de realidad aumentada.

La combinación de alta velocidad y precisión hace que YOLOv11 sea ideal para tareas donde se requiere una respuesta inmediata.
3. ¿Cómo se construyó YOLOv11?

La construcción de YOLOv11 sigue principios fundamentales en visión por computadora y aprendizaje profundo, con un enfoque en la eficiencia y flexibilidad. El diseño de su arquitectura incluye los siguientes pasos y características:

1. Backbone: Extracción de Características
El **Backbone** es el módulo inicial que procesa la imagen de entrada para extraer características jerárquicas en diferentes escalas. El flujo de datos dentro del backbone y hacia el siguiente módulo se define por etapas cuidadosamente diseñadas.

Estructura Interna del Backbone y Conexiones

1. **Entrada Inicial:**
    - La imagen de entrada de dimensiones 640×640×3640 \times 640 \times 3640×640×3 pasa por la primera capa convolucional (Conv), que aplica un filtro 3x3 con stride 2, reduciendo las dimensiones a 320×320×32320 \times 320 \times 32320×320×32. Esta operación actúa como una primera extracción de bordes y texturas básicas.
2. **Etapas Jerárquicas (P1 a P5):**
    - Cada etapa está formada por:
        - **Una capa de convolución (Conv)** para reducir las dimensiones espaciales.
        - **Bloques C3x2** que aplican operaciones residuales para enriquecer las características aprendidas.
        - Las conexiones internas de cada bloque C3x2 utilizan estructuras de tipo residual que fusionan la entrada y salida del bloque.
    - Ejemplo de flujo entre etapas:
        - La salida de P1 (320×320×32320 \times 320 \times 32320×320×32) alimenta directamente a P2, donde se reduce a 160×160×64160 \times 160 \times 64160×160×64, y así sucesivamente.
3. **Conexiones al Neck:**
    - Las salidas de múltiples niveles del backbone (80×80×25680 \times 80 \times 25680×80×256 de P3, 40×40×51240 \times 40 \times 51240×40×512 de P4, y 20×20×102420 \times 20 \times 102420×20×1024 de P5) se envían al **Neck**. Esto asegura que las características de diferentes escalas estén disponibles para el procesamiento multiescala en etapas posteriores.
4. **Bloque SPPF (Spatial Pyramid Pooling Fast):**
    - Después de P5, las características de tamaño 20×20×102420 \times 20 \times 102420×20×1024 pasan por el bloque SPPF, que utiliza operaciones de pooling piramidal con diferentes tamaños de ventana para agregar información global. Esto crea un resumen efectivo de las características sin pérdida de detalles críticos.

- Conexión: La salida del SPPF (20×20×102420 \times 20 \times 102420×20×1024) se envía directamente al **bloque final C2PSA**.

5. **Bloque C2PSA:**
   - Integra atención espacial y canalizada para destacar características importantes. La salida de este bloque se conecta al primer Concat del **Neck**.

2. Neck: Fusión Multiescala

El **Neck** conecta las características generadas en el backbone y las organiza en diferentes niveles para facilitar las predicciones en el módulo **Head**. El objetivo es combinar y procesar información multiescala a través de upsampling, concatenación y refinamiento.

Estructura Interna del Neck y Conexiones

1. **Entrada al Neck:**
   - Recibe tres entradas clave desde el backbone:
     - P3 (80×80×25680 \times 80 \times 25680×80×256).
     - P4 (40×40×51240 \times 40 \times 51240×40×512).
     - P5 (20×20×102420 \times 20 \times 102420×20×1024).
2. **Upsampling:**
   - La salida de P5 (20×20×102420 \times 20 \times 102420×20×1024) se somete a una operación de **upsampling**, duplicando el tamaño espacial a 40×40×102440 \times 40 \times 102440×40×1024.
   - Conexión: Esta salida escalada se concatena con P4 (40×40×51240 \times 40 \times 51240×40×512) utilizando un bloque Concat.
3. **Primera Concatenación (Concat):**
   - Combina la información escalada de P5 con las características de P4 (40×40×51240 \times 40 \times 51240×40×512), formando un tensor combinado de 40×40×153640 \times 40 \times 153640×40×1536.
   - Conexión: Este tensor se envía a un bloque C3x2 para procesar y refinar las características.
4. **Procesamiento de Características Refinadas:**
   - La salida del bloque C3x2 (40×40×51240 \times 40 \times 51240×40×512) se somete a un nuevo **upsampling**, ampliando su tamaño a 80×80×51280 \times 80 \times 51280×80×512.
   - Conexión: Se concatena con P3 (80×80×25680 \times 80 \times 25680×80×256).
5. **Segunda Concatenación (Concat):**
   - La salida escalada (80×80×51280 \times 80 \times 51280×80×512) y las características de P3 (80×80×25680 \times 80 \times 25680×80×256) se combinan, resultando en un tensor de 80×80×76880 \times 80 \times 76880×80×768.
   - Conexión: Este tensor se procesa a través de otro bloque C3x2 para generar una representación refinada que se envía directamente al **Head**.

3. Head: Predicciones Finales

El **Head** es responsable de generar las predicciones finales de clases, coordenadas de las cajas delimitadoras (bounding boxes), y las puntuaciones de confianza para cada objeto detectado. Este módulo utiliza las características refinadas provenientes del Neck.

Estructura Interna del Head y Conexiones

1. **Ramas de Predicción:**
    - Hay tres ramas independientes, cada una correspondiente a una escala:
        - **P3 (80x80):** Maneja objetos pequeños.
        - **P4 (40x40):** Maneja objetos medianos.
        - **P5 (20x20):** Maneja objetos grandes.
2. **Procesamiento Final en Cada Rama:**
    - Las características de cada escala pasan por capas convolucionales finales para ajustar las predicciones.
    - Cada rama produce simultáneamente:
        - **Clases:** Predicción categórica para cada objeto.
        - **Bounding Boxes:** Coordenadas de las cajas delimitadoras.
        - **Confianza:** Probabilidad de que la detección sea correcta.
3. **Salida Multitarea:**
    - Las predicciones finales se combinan en un solo tensor que contiene las cajas, clases y puntuaciones de confianza para todos los objetos detectados en la imagen.

Conexiones entre Módulos

El flujo de datos y las conexiones entre módulos se pueden resumir como:

1. **Backbone al Neck:**
    - Tres salidas del backbone (P3, P4, P5) se conectan al Neck, proporcionando características multiescala para procesamiento adicional.
2. **Dentro del Neck:**
    - Las conexiones incluyen:
        - **Upsampling:** Escala las características a tamaños mayores.
        - **Concat:** Fusiona características de diferentes etapas del backbone.
        - **C3x2:** Refina las características concatenadas para una mejor representación.
3. **Neck al Head:**

El Neck envía características refinadas y combinadas de P3, P4 y P5 al Head para predicciones específicas de cada escala.

3. ¿Cómo se construyó YOLOv11?

La construcción de YOLOv11 sigue principios fundamentales en visión por computadora y aprendizaje profundo, con un enfoque en la eficiencia y flexibilidad. El diseño de su arquitectura incluye los siguientes pasos y características:

a. Arquitectura Modular:

YOLOv11 se compone de tres módulos principales:

- **Backbone:** Se utiliza para extraer características de bajo, medio y alto nivel de las imágenes.
- **Neck:** Actúa como un puente entre el Backbone y el Head, combinando características de diferentes escalas
- **Head:** Genera predicciones finales, incluyendo:

b. Escalabilidad:

Se introdujeron parámetros ajustables para adaptar el modelo a diferentes escenarios:

- **Depth Multiple (d):** Modifica la profundidad de los bloques residuales.
- **Width Multiple (w):** Ajusta el número de canales en las capas convolucionales.
- **Max Channels (mc):** Establece un límite superior para los canales, controlando los recursos necesarios.

c. Optimización Computacional:

- Se implementaron técnicas como convoluciones profundas, capas de atención, y procesamiento paralelo para maximizar la eficiencia.
- La arquitectura minimiza la cantidad de cálculos redundantes, lo que la hace ideal para aplicaciones en dispositivos con recursos limitados, como cámaras y drones.

d. Algoritmos de Entrenamiento:

El modelo se construyó usando aprendizaje supervisado. Esto significa que fue entrenado con un gran conjunto de imágenes etiquetadas que contienen las clases de objetos y las ubicaciones de las cajas delimitadoras.

4. ¿Con qué datos se entrenó YOLOv11?

YOLOv11 fue entrenado con un conjunto de datos extenso y diverso para garantizar un rendimiento robusto en tareas de detección de objetos en distintos contextos.

a. Conjuntos de Datos Principales:

1. **COCO (Common Objects in Context):**
    - Conjunto de datos estándar utilizado en detección de objetos.
    - Contiene más de 200,000 imágenes con 80 clases de objetos comunes, como personas, vehículos, animales, y electrodomésticos.
    - Proporciona anotaciones detalladas de las cajas delimitadoras y segmentación de objetos.
2. **PASCAL VOC:**
    - Conjunto de datos clásico en visión por computadora.
    - Contiene 20 clases de objetos y un menor volumen de datos en comparación con COCO, pero ofrece anotaciones precisas.
3. **Open Images Dataset:**

- Un conjunto de datos más grande que COCO, con millones de imágenes etiquetadas y más de 600 clases.
- Incluye anotaciones detalladas, como relaciones entre objetos y características adicionales.

b. Ampliación de Datos (Data Augmentation):
Para mejorar la robustez del modelo y prevenir el sobreajuste, se utilizaron técnicas de aumento de datos, como:

- **Transformaciones geométricas:** Rotación, escalado, y recorte.
- **Ajustes de color:** Variación en brillo, contraste, y saturación.
- **Introducción de ruido:** Perturbaciones aleatorias para simular condiciones del mundo real.

1. ¿Qué es DeepSORT con el embedding MobileNet?
DeepSORT (Simple Online and Realtime Tracking) es un algoritmo avanzado de seguimiento de objetos en tiempo real que extiende el enfoque original de SORT (Simple Online and Realtime Tracking). DeepSORT incorpora un sistema de **embeddings profundos** para mejorar la asociación de objetos a través de múltiples fotogramas, utilizando características visuales extraídas por una red neuronal, como **MobileNet**.

Características principales:

- **Basado en Kalman Filters:** Utiliza un filtro de Kalman para predecir la ubicación futura de los objetos rastreados en función de sus movimientos anteriores.
- **Asociación por Costo Mínimo:** Combina datos de detección (como bounding boxes) con características visuales para realizar asociaciones precisas entre los objetos rastreados y las detecciones en el siguiente fotograma.
- **Embeddings de Deep Learning:** En este caso, se utiliza **MobileNet** como red de extracción de características para generar embeddings compactos y distintivos de cada objeto.

Cuando MobileNet se emplea como extractor de características visuales, se obtiene una solución eficiente y ligera, ideal para dispositivos con recursos computacionales limitados o aplicaciones en tiempo real.

2. ¿Para qué sirve DeepSORT con el embedding MobileNet?
DeepSORT con MobileNet se utiliza principalmente en tareas de **seguimiento de objetos en tiempo real**, lo que significa rastrear continuamente un objeto específico en una secuencia de video. Este enfoque mejora la robustez frente a desafíos como oclusiones, objetos similares, y cambios de apariencia.

Aplicaciones principales:

1. **Seguridad y vigilancia:**
    - Seguimiento de personas en sistemas de videovigilancia.
    - Rastreo de vehículos en cámaras de tráfico.
2. **Automoción:**
    - Uso en vehículos autónomos para rastrear peatones y otros vehículos.
3. **Deportes:**
    - Análisis de movimiento y seguimiento de jugadores durante eventos deportivos.
4. **Retail y logística:**
    - Monitoreo de clientes en tiendas.
    - Seguimiento de productos en almacenes.
5. **Drones y robótica:**
    - Rastreo de objetos en misiones de navegación o reconocimiento.

El uso de MobileNet como extractor de características reduce significativamente el costo computacional, permitiendo implementar DeepSORT en dispositivos con recursos limitados, como drones y cámaras inteligentes.

3. ¿Cómo se construyó?

DeepSORT con el embedding MobileNet combina algoritmos de seguimiento tradicionales (filtro de Kalman y asignación de costos) con técnicas modernas de aprendizaje profundo. El flujo de construcción se detalla a continuación:

a. Estructura del Algoritmo DeepSORT

1. **Filtro de Kalman:**
    - Predice el estado futuro de los objetos rastreados (posición y velocidad).
    - Representa el estado de cada objeto como un vector que incluye posición, tamaño, y velocidad en el espacio 2D.
2. **Asignación de Costo (Hungarian Algorithm):**
    - Asocia las predicciones de los objetos rastreados con las detecciones actuales.
    - La asociación se basa en dos métricas:
        - **Distancia de Mahalanobis:** Evalúa la similitud en términos de movimiento.
        - **Distancia de Coseno en los Embeddings:** Evalúa la similitud visual entre el objeto rastreado y la nueva detección.
3. **Embeddings Visuales:**
    - DeepSORT utiliza características visuales para desambiguar objetos similares. En este caso, **MobileNet** genera un vector de características (embedding) único para cada detección.

b. MobileNet como Extractor de Embeddings

1. **MobileNet:**
    - Es una red neuronal convolucional ligera, diseñada para ejecutarse eficientemente en dispositivos con recursos limitados.
    - Genera embeddings compactos de alta calidad que representan características visuales de los objetos.
2. **Proceso de Integración:**
    - Cada detección (bounding box) se recorta de la imagen original.
    - MobileNet procesa el recorte y produce un vector de características visuales.
    - Este vector se utiliza para medir la similitud entre detecciones en diferentes fotogramas.

c. Flujo General del Algoritmo

1. En el primer fotograma:
    - Se generan detecciones iniciales (bounding boxes).
    - MobileNet extrae embeddings visuales para cada detección.
    - Cada detección se asigna a un nuevo ID de rastreo.
2. En fotogramas subsiguientes:
    - El filtro de Kalman predice la ubicación futura de cada objeto rastreado.
    - Las detecciones actuales se asocian con las predicciones utilizando métricas de similitud basadas en movimiento y visuales.
    - Si no se encuentra una asociación para una detección, se crea un nuevo objeto rastreado.
    - Si un objeto rastreado no se asocia con detecciones durante varios fotogramas, se elimina.

4. ¿Con qué datos se entrenó?

a. Datos para el Modelo de Embedding (MobileNet)

MobileNet fue preentrenado en conjuntos de datos estándar para clasificación de imágenes:

1. **ImageNet:**
    - Un conjunto de datos extenso con más de 1 millón de imágenes y 1,000 clases.
    - Proporciona características visuales genéricas útiles para identificar objetos en videos.

b. Datos para el Seguimiento (DeepSORT)

DeepSORT no requiere un entrenamiento directo para su algoritmo de seguimiento, pero se evalúa y ajusta utilizando conjuntos de datos de seguimiento etiquetados. Ejemplos:

1. **MOT Challenge (Multiple Object Tracking):**
    - Conjunto de datos estándar para evaluación de algoritmos de seguimiento.

- Incluye videos anotados con múltiples objetos en movimiento, como personas y vehículos.
- Las métricas principales incluyen MOTA (Multiple Object Tracking Accuracy) y MOTP (Precision).

c. Ajuste Fino para Aplicaciones Específicas

DeepSORT con MobileNet puede ajustarse a dominios específicos mediante **fine-tuning**:

- Ejemplo: Si se aplica a un entorno de tráfico, MobileNet puede ajustarse utilizando conjuntos de datos de vehículos como **UA-DETRAC**.
- Este ajuste permite mejorar la capacidad del modelo para generar embeddings más distintivos en el contexto particular.

Implementación y Uso del Sistema de Detección y Seguimiento en Tiempo Real

1. Introducción

Este proyecto tiene como objetivo implementar un sistema de detección y seguimiento de objetos en tiempo real utilizando modelos de aprendizaje profundo. Para ello, se combinan dos tecnologías principales: **YOLO (You Only Look Once)** para la detección y **DeepSORT** para el seguimiento. La idea es identificar objetos en un video y asignarles un identificador único que permita rastrearlos a lo largo del tiempo.

2. Componentes del Sistema

El sistema está dividido en tres módulos principales:

1. **Código principal (yolo_detection_tracking.py)**: Este archivo orquesta el funcionamiento del sistema. Captura los frames de la cámara en tiempo real, utiliza YOLO para detectar objetos y pasa las detecciones a DeepSORT para realizar el seguimiento. Al final, muestra los resultados en un video con los cuadros delimitadores e identificadores de los objetos rastreados.
2. **Detector YOLO (yolo_detector.py)**: Este módulo implementa el detector YOLO, que es responsable de encontrar objetos en cada frame. Aquí se filtran las detecciones para quedarse solo con objetos de interés (en este caso, personas) y se eliminan cuadros redundantes mediante un proceso de **supresión de no máximos (NMS)**.
3. **Rastreador DeepSORT (tracker.py)**: DeepSORT es el encargado de asociar las detecciones entre frames consecutivos y asignarles un identificador único. Esto permite que el sistema "recuerde" a cada objeto y pueda seguirlo incluso si desaparece brevemente o si hay solapamiento con otros objetos.

3. Cómo Funciona el Sistema

1. **Inicialización**:
   - Se carga el modelo YOLO desde un archivo preentrenado y se configuran los parámetros de confianza e IoU para la detección.
   - Se inicializa DeepSORT con valores ajustados para maximizar la precisión del seguimiento.
2. **Captura de Frames**:

    - El sistema utiliza la cámara en tiempo real como fuente de video. Se configura la resolución de la cámara para garantizar un buen rendimiento.
3. **Detección de Objetos**:
    - YOLO procesa cada frame para detectar objetos. Se filtran los resultados para incluir solo objetos de interés (por ejemplo, personas).
    - Se aplica una supresión de no máximos adicional para eliminar detecciones redundantes.
4. **Seguimiento de Objetos**:
    - Las detecciones procesadas se envían a DeepSORT, que asigna un ID único a cada objeto y rastrea su posición en frames posteriores.
    - Los parámetros de DeepSORT están configurados para minimizar la reasignación de IDs y mantener la consistencia.
5. **Visualización**:
    - Los cuadros delimitadores e identificadores de los objetos rastreados se dibujan sobre el video en tiempo real.
    - Se calcula y muestra el FPS actual para monitorear el rendimiento del sistema.
6. **Finalización**:
    - El sistema se detiene al presionar la tecla q o ESC, liberando los recursos de la cámara y cerrando las ventanas de visualización.

4. Configuración y Uso

1. **Requisitos del Sistema**:
    - **Librerías necesarias**:
        - ultralytics (para YOLO).
        - deep_sort_realtime (para DeepSORT).
        - opencv-python (para captura de video y visualización).
        - numpy (para manipulación de matrices).
    - **Hardware recomendado**:
        - Una GPU con soporte CUDA para un procesamiento más rápido.
        - CPU con al menos 4 núcleos si se trabaja sin GPU.
2. **Cómo ejecutar el sistema**:
    - Asegúrate de tener instalada la cámara y que funcione correctamente.
    - Configura las rutas del modelo YOLO (yolo11n.pt) en el archivo principal.
    - Ejecuta el programa principal (yolo_detection_tracking.py) en tu entorno Python.

5. Resultados

Durante la ejecución, el sistema realiza detecciones y rastrea objetos con los siguientes detalles:

- Los objetos detectados se identifican con cuadros delimitadores y un ID único.

- El sistema muestra un promedio de **1.82 FPS** en pruebas realizadas en una máquina sin GPU, lo que demuestra que funciona pero podría optimizarse para mayor fluidez.
- Los IDs se mantienen consistentes para la mayoría de los objetos, aunque pueden reasignarse si el objeto desaparece por mucho tiempo o se detecta de forma inconsistente.

6. Limitaciones

- **Rendimiento**: En hardware sin GPU, el sistema tiene un rendimiento limitado, especialmente en escenas con múltiples objetos.
- **Reasignación de IDs**: Aunque se configuró DeepSORT para minimizar este problema, aún puede ocurrir en escenarios muy dinámicos o con detecciones inconsistentes.
- **Solo una clase detectada**: Actualmente, el sistema está diseñado para detectar únicamente personas. Expandirlo a múltiples clases requeriría ajustes en los filtros y visualizaciones.

7. Posibles Mejoras

- **Optimización del rendimiento**:
  - Usar un modelo YOLO más ligero, como YOLOv8-nano, para aumentar los FPS.
  - Implementar procesamiento paralelo para manejar la detección y el seguimiento de forma más eficiente.
- **Soporte para múltiples clases**:
  - Ampliar la lista de clases objetivo y mostrar diferentes colores o etiquetas según el tipo de objeto detectado.
- **Visualización más rica**:
  - Agregar líneas de trayectoria para mostrar el movimiento de los objetos a lo largo del tiempo.
  - Incluir información adicional, como velocidad o dirección estimada.

8. Conclusión

Este sistema combina técnicas avanzadas de detección y seguimiento para rastrear objetos en tiempo real. Aunque tiene limitaciones, como el rendimiento en hardware sin GPU, es un excelente punto de partida para proyectos relacionados con análisis de video, vigilancia o interacción en tiempo real con objetos en movimiento.
El uso de YOLO para detección y DeepSORT para seguimiento garantiza precisión y estabilidad, mientras que su modularidad permite futuras mejoras y adaptaciones a diferentes casos de uso.

Bibliografía:

https://github.com/ultralytics/
https://github.com/nwojke/deep_sort/tree/master/deep_sort