Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

Máster Universitario en
Ingeniería Industrial

Trabajo de Fin de Máster

Análisis en tiempo real del flujo de pasajeros en las estaciones de tranvía mediante IA y visión por computador

Autor: Jaime Oliva García

Tutor: Cándido Caballero Gil

Cotutora: Sonia Díaz Santos

La Laguna, 6 de julio de 2024

La publicación de este Trabajo Fin de Máster solo implica que el estudiante ha obtenido al menos la nota mínima exigida para superar la asignatura correspondiente, no presupone que su contenido sea correcto, aunque si aplicable. En este sentido, la ULL no posee ningún tipo de responsabilidad hacia terceros por la aplicación total o parcial de los resultados obtenidos en este trabajo. También pone en conocimiento del lector que, según la ley de protección intelectual, los resultados son propiedad intelectual del alumno, siempre y cuando se haya procedido a los registros de propiedad intelectual o solicitud de patentes correspondientes con fecha anterior a su publicación.

D. **Cándido Caballero Gil**, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

Dña. **Sonia Díaz Santos**, alumna del Doctorado de Ingeniería Industrial, Informática y Medioambiental adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora.

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Análisis en tiempo real del flujo de pasajeros en las estaciones de tranvía mediante IA y visión por computador*”

ha sido realizada bajo su dirección por D. **Jaime Oliva García**,

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de julio de 2024

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a todas las personas que han hecho posible la realización de este trabajo y la finalización del máster.

En primer lugar, quiero agradecer a mi familia, especialmente a mis padres y a mi hermano, quienes siempre han estado a mi lado brindándome su apoyo incondicional, su amor y sus palabras de aliento en cada etapa de este proyecto.

Agradezco profundamente a todos mis profesores por todos estos años de enseñanza y dedicación. Su guía y conocimientos han sido fundamentales para mi formación académica y personal.

También quiero dar las gracias a mis compañeros, quienes a lo largo de estos años se han convertido en amigos. Su colaboración, apoyo y compañerismo han sido esenciales para superar juntos los desafíos y disfrutar de los logros.

A todos ustedes, gracias de corazón.

Licencia

Para esta obra se permite que se compartan adaptaciones mientras se compartan de la misma manera y NO se permiten usos comerciales de la obra.

Resumen

El presente trabajo aborda el desarrollo de un sistema de detección y seguimiento de personas utilizando la plataforma Jetson Nano, la cámara Raspberry PI V2.1 y modelos de inteligencia artificial basados en YOLOv8. El objetivo principal del proyecto es implementar un sistema capaz de detectar y contar personas en diversos entornos mediante el análisis de vídeos pregrabados. Para ello, se desarrollaron dos componentes principales: un script principal y un módulo de seguimiento.

El script principal se encarga de la captura de vídeo, la detección de objetos, la definición de áreas de interés y la visualización de los resultados. Se emplearon varios modelos de YOLOv8 (desde YOLOv8n hasta YOLOv8l), seleccionando finalmente el modelo YOLOv8s por su balance óptimo entre precisión y velocidad. El módulo de seguimiento, por su parte, rastrea las personas detectadas a lo largo de los fotogramas del vídeo, asignando identificadores únicos y manteniendo un seguimiento continuo.

Las pruebas se realizaron con dos vídeos pregrabados: uno de una plaza y otro de una estación de tren. En el primer caso, el sistema logró detectar y contar las personas con una precisión del 100 %. En el segundo caso, aunque la detección fue satisfactoria, el conteo no fue tan preciso debido a la menor calidad del vídeo y la mayor distancia de grabación.

En conclusión, los resultados obtenidos demuestran la eficacia del sistema desarrollado y sugieren futuras mejoras, como la implementación en tiempo real, la optimización de modelos y la detección de otros objetos, lo que expandiría las capacidades del sistema para aplicaciones de seguridad y monitorización de flujo de personas.

Palabras clave: Detección de personas, Seguimiento de objetos, Inteligencia artificial, Visión por computadora, Jetson Nano, YOLOv8, Procesamiento de vídeo, Raspberry Pi Cámara V2.1, Monitorización de flujo, Deep learning, Modelos de detección, Optimización de modelos, Tranvía

Abstract

The present work addresses the development of a person detection and tracking system using the Jetson Nano platform, Raspberry PI V2.1 camera and artificial intelligence models based on YOLOv8. The main objective of the project is to implement a system capable of detecting and counting people in various environments through the analysis of pre-recorded videos. To achieve this, two main components were developed: a main script and a tracking module.

The main script is responsible for video capture, object detection, definition of areas of interest, and visualization of results. Several YOLOv8 models (from YOLOv8n to YOLOv8l) were employed, with YOLOv8s finally being selected for its optimal balance between accuracy and speed. The tracking module, on the other hand, tracks the detected people across video frames, assigning unique identifiers and maintaining continuous tracking.

Tests were conducted with two pre-recorded videos: one of a plaza and another of a train station. In the first case, the system successfully detected and counted people with 100 % accuracy. In the second case, although the detection was satisfactory, the counting was not as accurate due to the lower video quality and greater recording distance.

In conclusion, the results obtained demonstrate the effectiveness of the developed system and suggest future improvements, such as real-time implementation, model optimization, and the detection of other objects, which would expand the system's capabilities for security and flow monitoring applications.

Keywords: Person detection, Object tracking, Artificial intelligence, Computer vision, Jetson Nano, YOLOv8, Video processing, Raspberry Pi Camera V2.1, Flow monitoring, Deep learning, Detection models, Model optimization, Tram

Índice general

Índice de Figuras

Índice de Tablas

Índice de Listings

Capítulo 1

Introducción

1.1. Antecedentes

El análisis del flujo de pasajeros en estaciones de tranvía ha sido un área de interés y estudio durante muchas décadas debido a su impacto en la eficiencia y calidad del transporte público. Este apartado proporciona una revisión de la historia y evolución del problema, el desarrollo de tecnologías relacionadas, estudios y proyectos previos, y el estado del arte en el campo.

1.1.1. Historia y Evolución del Problema

La gestión del flujo de pasajeros en las estaciones de tranvía y otros sistemas de transporte público ha sido un desafío persistente a lo largo de la historia. Con el crecimiento de las ciudades y la urbanización acelerada, el transporte público se ha convertido en una columna vertebral esencial para la movilidad urbana. A continuación, se presenta una revisión exhaustiva de la evolución histórica de este problema, desde los métodos tradicionales hasta las tecnologías avanzadas contemporáneas.

En las primeras etapas del transporte público, la gestión del flujo de pasajeros se basaba principalmente en métodos manuales y observaciones directas. Los operadores de tranvías y autobuses contaban pasajeros a mano, registrando los datos en hojas de papel como se puede ver en la Figura 1.1. Este método, aunque rudimentario, proporcionaba una visión básica del número de pasajeros y los patrones de uso. A medida que las ciudades crecían y la demanda de transporte público aumentaba, estos métodos manuales comenzaron a mostrar sus limitaciones. La capacidad para manejar grandes volúmenes de datos de pasajeros se volvió inadecuada, y la precisión de las observaciones manuales era baja, lo que dificultaba la planificación y gestión eficiente del servicio.

En las décadas de 1970 y 1980, se introdujeron los primeros sistemas

Figura 1.1: Puesto de cobrador años 70. Fuente: Google Imágenes [1]

automáticos de conteo de pasajeros (APC). Estos sistemas utilizaban sensores básicos, como barreras infrarrojas y tapetes sensibles a la presión, instalados en las entradas y salidas de los vehículos de transporte. Estos sensores podían contar el número de pasajeros que subían y bajaban del tranvía, proporcionando datos más precisos y en tiempo real que los métodos manuales. A pesar de las mejoras, estos primeros sistemas APC tenían sus limitaciones. La tecnología de sensores infrarrojos, por ejemplo, podía ser inexacta en condiciones de alta densidad de pasajeros y no podía distinguir entre diferentes tipos de objetos (por ejemplo, pasajeros con equipaje o carritos). Además, estos sistemas carecían de la capacidad de analizar patrones complejos de movimiento y comportamiento de los pasajeros.

Con el avance de la tecnología en las décadas de 1990 y 2000, los sistemas APC se volvieron más sofisticados. Se introdujeron sensores más avanzados, como cámaras de video y sistemas de radar, que podían proporcionar datos más precisos y detallados. Además, la proliferación de tecnologías de comunicación inalámbrica permitió la transmisión de datos en tiempo real a centros de control centralizados. Estos avances permitieron una mejor integración de los datos de pasajeros con otros sistemas de gestión del transporte, como la programación de horarios y la gestión de flotas. Los operadores de transporte podían ahora monitorear el flujo de pasajeros en tiempo real y tomar decisiones informadas para ajustar el servicio según la demanda.

La revolución digital y el auge del Big Data a partir de la década de 2010 transformaron radicalmente la gestión del flujo de pasajeros. Con la capacidad de procesar grandes volúmenes de datos en tiempo real,

los operadores de transporte comenzaron a utilizar análisis de Big Data para identificar patrones y tendencias en el uso del transporte público. La inteligencia artificial y el aprendizaje automático (machine learning) han desempeñado un papel crucial en esta transformación. Los algoritmos de aprendizaje profundo, en particular, han permitido el desarrollo de modelos predictivos que pueden anticipar la demanda de pasajeros y optimizar los recursos en consecuencia. Estos modelos pueden analizar una amplia variedad de datos, incluyendo datos históricos de pasajeros, condiciones meteorológicas, eventos especiales y más.

La visión por computador ha sido uno de los avances más significativos en la gestión del flujo de pasajeros. Utilizando cámaras de video y algoritmos avanzados de procesamiento de imágenes, los sistemas de visión por computador pueden detectar y seguir el movimiento de pasajeros en tiempo real. Esto ha permitido una mayor precisión en la identificación de congestiones y la optimización de los flujos de pasajeros. Los sistemas de visión por computador utilizan redes neuronales convolucionales (CNN) para el reconocimiento de imágenes y la detección de objetos. Estos sistemas pueden diferenciar entre pasajeros y otros objetos, como equipaje o bicicletas, y proporcionar datos detallados sobre el comportamiento de los pasajeros, como tiempos de espera, rutas preferidas y patrones de movimiento. En la Figura 1.2 se muestra una de las salas de monitoreo de trenes que se usa actualmente en Renfe.

Figura 1.2: Monitoreo actual de transporte público. Fuente: Google Imágenes [2]

En años recientes, se han llevado a cabo numerosos proyectos e implementaciones de sistemas avanzados de gestión del flujo de pasajeros en todo el mundo. Por ejemplo, en Nueva York, el proyecto ’MetroFlow’ utilizó visión por computador y análisis de Big Data para mejorar la gestión del

metro. Este proyecto demostró cómo los algoritmos de IA podían mejorar la precisión en la detección de pasajeros y proporcionar datos en tiempo real para la toma de decisiones operativas. En Londres, Transport for London (TfL) ha implementado sistemas avanzados de monitoreo en sus estaciones de metro y autobuses, utilizando una combinación de sensores, cámaras y análisis de Big Data. Estos sistemas han permitido a TfL mejorar la eficiencia del servicio, reducir tiempos de espera y mejorar la experiencia del usuario[10].

1.1.2. Contexto y Motivación del estudio

El transporte público es una pieza fundamental en la infraestructura de cualquier ciudad moderna. En particular, los sistemas de tranvía juegan un papel crucial en la movilidad urbana, ofreciendo una opción de transporte eficiente, sostenible y accesible. Sin embargo, uno de los desafíos más significativos que enfrentan las autoridades de transporte y los operadores de tranvía es la gestión efectiva del flujo de pasajeros. En muchas ciudades, la alta densidad de pasajeros durante las horas pico puede llevar a congestiones, retrasos y una experiencia de usuario negativa.

Las estaciones de tranvía, especialmente en áreas urbanas densamente pobladas, experimentan fluctuaciones significativas en el número de pasajeros a lo largo del día. Estos picos de demanda pueden provocar problemas operativos como la sobrecarga de los vagones, tiempos de espera prolongados y dificultades en el embarque y desembarque. Estos problemas no solo afectan la eficiencia del sistema de transporte, sino que también impactan negativamente en la satisfacción de los usuarios y en la percepción general del transporte público.

El análisis en tiempo real del flujo de pasajeros utilizando inteligencia artificial y visión por computador ofrece una solución innovadora y efectiva para abordar estos desafíos. Al implementar sistemas avanzados que pueden monitorear y analizar continuamente el flujo de pasajeros, es posible mejorar la gestión y operación de los servicios de tranvía de varias maneras. Con datos precisos en tiempo real, los operadores pueden ajustar la frecuencia de los tranvías y la asignación de vehículos de manera más eficiente, respondiendo dinámicamente a las fluctuaciones en la demanda, lo que resulta una mejor optimización de los recursos. Un sistema de transporte más eficiente y predecible mejora significativamente la satisfacción del usuario, reduciendo los tiempos de espera y proporcionando un viaje más cómodo y seguro. Por otro lado, cabe mencionar que la capacidad de

monitorear y analizar el flujo de pasajeros también mejora la seguridad, permitiendo una respuesta más rápida y eficaz en situaciones de emergencia o evacuaciones. En la Figura 1.3 se puede observar un ejemplo del sistema que dispone Renfe en una de sus paradas de tren.

Figura 1.3: Sistema de cámaras de Renfe. Fuente: Google Imágenes [3]

En cuanto a mi interés por la realización del presente proyecto surge de la combinación de mi pasión por la inteligencia artificial y la automatización para contribuir en la mejora de procesos. Es por ello que la inteligencia artificial y la visión por computador representan áreas fascinantes para este proyecto y que están en constante evolución, con un potencial enorme para transformar diversos aspectos de nuestra vida cotidiana. Aplicar estas tecnologías al transporte público no solo ofrece una oportunidad para explorar innovaciones tecnológicas, sino que también permite abordar un problema real y tangible que afecta a millones de personas en las ciudades de todo el mundo. Académicamente, este proyecto me proporciona una plataforma para aplicar y expandir mis conocimientos en la propia inteligencia artificial y visión por computador, al tiempo que desarrollo habilidades prácticas en el análisis y resolución de problemas complejos, como es el análisis del flujo de pasajeros. Personalmente, creo firmemente en el poder de la tecnología para mejorar aquellos procesos que son repetitivos o que están presentes en el día a día de la vida urbana.

1.1.3. Objetivos del Trabajo de Fin de Máster

El objetivo principal de este Trabajo de Fin de Máster (TFM) es desarrollar un sistema basado en inteligencia artificial y visión por computador para el análisis en tiempo real del flujo de pasajeros en las estaciones de

tranvía. Este sistema tiene como finalidad mejorar la gestión operativa del transporte público, optimizar el uso de recursos, reducir congestiones y elevar la satisfacción en cuanto a la experiencia de los usuarios del tranvía de Tenerife. Se desarrollará un prototipo de sistema basado en una computadora NVIDIA Jetson Nano y una cámara de Raspberry Pi para analizar en tiempo real el flujo de pasajeros en las estaciones. Este sistema será probado inicialmente con vídeos pregrabados de flujo de personas para verificar su funcionalidad antes de su instalación en una parada de tranvía real.

Para alcanzar el objetivo general, se han definido los siguientes objetivos específicos:

- Diseñar e Implementar un Algoritmo de Detección y Seguimiento de Pasajeros: Crear un algoritmo robusto utilizando técnicas de visión por computador para detectar y seguir a los pasajeros en los vídeos de prueba. Este algoritmo debe ser capaz de operar en tiempo real y manejar diversas condiciones ambientales y de iluminación. Para ello será necesario utilizar modelos para el reconocimiento de imágenes y la detección de movimiento, además de implementar estos modelos en la plataforma Jetson Nano para su procesamiento en tiempo real.
- Implementar estos modelos en la plataforma Jetson Nano para su procesamiento en tiempo real: Configurar la Jetson Nano y la cámara de Raspberry Pi para capturar y procesar datos de vídeo en tiempo real. Asegurar una integración eficiente entre el hardware y el software. Desarrollar el software necesario para la captura y procesamiento de vídeo, optimizando el rendimiento para la plataforma hardware utilizada.
- Probar el Sistema Utilizando Vídeos Pregrabados de Flujo de Personas: Evaluar la funcionalidad y precisión del sistema mediante la utilización de vídeos pregrabados que simulan el flujo de pasajeros en una estación de tranvía. Para ello se seleccionan y utilizan vídeos de prueba representativos de diferentes escenarios y condiciones, además de analizar los resultados obtenidos por el sistema y realizar ajustes necesarios para mejorar su precisión y eficiencia.

Capítulo 2

Marco Teórico

2.1. Fundamentos de la visión por computador

La visión por computadora, también conocida como visión artificial o visión técnica, es una rama científica que abarca métodos para capturar, procesar, analizar y entender imágenes del mundo real con el objetivo de generar información numérica o simbólica que pueda ser procesada por un ordenador. De manera similar a cómo los seres humanos utilizan sus ojos y cerebros para interpretar su entorno, la visión por computadora busca dotar a los ordenadores de la capacidad de percibir y comprender imágenes o secuencias de imágenes, y actuar en consecuencia según la situación.

Esta capacidad de comprensión se logra a través de la integración de diversos campos como la geometría, la estadística, la física, entre otras disciplinas. La adquisición de datos puede realizarse mediante diferentes medios, tales como secuencias de imágenes, vistas capturadas por varias cámaras de video, o datos multidimensionales obtenidos de dispositivos como escáneres médicos. Numerosas tecnologías emplean la visión por computadora, entre las cuales se incluyen el reconocimiento de objetos, la detección de eventos, la reconstrucción de escenas (mapping) y la restauración de imágenes.

El objetivo principal de la visión por computadora es desarrollar estrategias automáticas para el reconocimiento de patrones complejos en imágenes provenientes de diversos dominios. Actualmente, muchas áreas se han beneficiado de estas técnicas. Un ejemplo notable es la robótica, donde los robots autónomos necesitan reconocer con precisión la ubicación de los objetos en su entorno para evitar colisiones. Esto a menudo se logra mediante el uso de sensores o cámaras, siendo estas últimas particularmente adecuadas para la aplicación de estrategias de visión por computadora.

Sin embargo, la robótica no es el único campo que se ha visto favorecido por estas técnicas. En el ámbito de la imagen médica, los sistemas de visión por computadora pueden reconocer patrones patológicos en imágenes específicas y diagnosticar enfermedades de manera automatizada. Otras áreas de aplicación incluyen los sistemas de seguridad, el seguimiento de objetos (como el seguimiento de un jugador en un partido de fútbol) y la detección de anomalías en piezas fabricadas en una cadena de producción, lo cual es crucial para el control de calidad.

Al aplicar los conceptos teóricos de la visión por computadora, siempre enfrentamos interferencias y problemas relacionados con el mundo real. Esto se debe a que nuestro entorno no es perfecto y los dispositivos de medición y captura también tienen sus limitaciones. Estos dispositivos introducen, en mayor o menor medida, distorsiones o ruidos que contaminan la imagen o muestra con la que deseamos trabajar. Entre estos problemas, identificamos los siguientes tipos de ruidos técnicos, entre otros:

2.1.1. Salt and Pepper

El ruido Salt and Pepper es un tipo de ruido impulsivo que causa que los píxeles afectados tomen un valor extremo, ya sea máximo (blanco) o mínimo (negro). El efecto de este ruido en una imagen en blanco y negro o en escala de grises es la aparición de diversos puntos blancos y negros dispersos aleatoriamente por la imagen. Este nombre se debe a que la imagen parece haber sido rociada con sal y pimienta. Se puede ver un ejemplo de dicho ruido en la Figura 2.1.

Este ruido puede surgir debido a problemas en los canales de transmisión de las imágenes. Para mitigar este ruido, se puede utilizar un filtro de promedio espacial, aunque esto puede resultar en una pérdida de definición de la imagen. Este método reduce el impacto del ruido al promediar los valores de los píxeles circundantes, suavizando la imagen. Sin embargo, es importante señalar que, aunque es útil, este filtro no siempre es el más confiable y puede haber implicaciones de seguridad, ya que las imágenes filtradas pueden ser susceptibles a la interceptación o espionaje si no se manejan adecuadamente.

2.1.2. Ruido Uniforme

El ruido uniforme se produce cuando el valor original del píxel distorsionado es sustituido por otro valor siguiendo una distribución uniforme en

Figura 2.1: Ejemplo de Salt and Pepper. Fuente: PDF [4]

el intervalo de valores posibles, es decir, desde el blanco hasta el negro. El efecto visual de este ruido es la percepción de interferencias en la imagen, similar a una imagen codificada. Se observa una pantalla superpuesta a la imagen llena de píxeles con valores aleatorios y uniformemente distribuidos como se puede apreciar en la Figura 2.2

Este ruido puede aparecer durante el proceso de cuantificación de una imagen. Para mitigar este ruido, se puede utilizar un filtro de promedio espacial, aunque esto puede conllevar una pérdida de definición en la imagen. Este método reduce el impacto del ruido promediando los valores de los píxeles circundantes, suavizando la imagen.

Imagen original: Lenna Imagen con ruido uniforme ($\sigma = 10{,}0$)

Figura 2.2: Ejemplo de Ruido uniforme. Fuente: PDF [5]

2.1.3. Ruido Gaussiano

El *ruido gaussiano* es un ruido derivado de los equipos de captura que sigue la distribución normal, representada por la fórmula:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

donde μ es la media y σ es la desviación estándar. El efecto en la imagen es similar al ruido uniforme, pero los valores del ruido no son tan abruptos, tendiendo más a tonos grises que a negros y blancos. Para mitigar este problema, se puede utilizar un filtro de promedio espacial con coeficientes gaussianos.

En todos los casos, para solventar cualquier tipo de ruido, deberemos aplicar algún tipo de filtro compatible con nuestro tipo de ruido. Los filtros de promedio espacial y gaussianos son algunas de las técnicas disponibles para reducir el impacto de los ruidos en las imágenes.

Existen otros tipos de interferencias debidas al contexto e interpretación de la imagen. A continuación, se enumeran algunas:

2.1.4. Punto de Vista

El punto de vista se refiere a la orientación de la imagen respecto al observador. Por ejemplo, un animal no se ve igual de frente que de espaldas, aunque sigue siendo el mismo animal. La iluminación también juega un papel crucial: la cantidad de luz que recibe el objeto puede afectar su apariencia. Un objeto debe ser distinguible independientemente de si tiene alguna cara oscura debido a la iluminación.

2.1.5. Oclusión

La oclusión ocurre cuando un objeto está parcialmente bloqueado por otro objeto, dificultando su extracción o análisis. Debemos ser capaces de interpretar que un objeto diferente está entre nuestro objeto a extraer y el observador.

2.1.6. Escala

La escala determina el tamaño de la imagen respecto al real. Un edificio no parecerá tener el mismo tamaño dependiendo de la imagen capturada (ya sea por distancia, ángulo, etc.). Debemos ser capaces de interpretar

que un objeto puede ser el mismo pese a que el tamaño en las fotografías pueda ser diferente.

2.1.7. Deformación

Un objeto puede estar deformado debido a múltiples factores (por ejemplo, la distorsión de una carretera en pleno verano) o simplemente debido a errores de captura, o posicionamiento y ángulo de la captura. Debemos interpretar que el objeto pertenece a una categoría a pesar de sus deformaciones.

2.1.8. Fondo Desordenado

Un objeto puede estar en un contexto desordenado y caótico, como por ejemplo un mosaico. Debemos ser capaces de distinguir el objeto entre el caos que lo envuelve.

2.1.9. Variaciones Dentro de una Misma Clase

Un tipo de objeto puede ser muy dispar a otro de su misma categoría. Por ejemplo, hay multitud de sillas diferentes, pero todas comparten los mismos rasgos característicos: cuatro patas, una superficie para sentarse y un respaldo.

Dependiendo de cómo solucionemos estos problemas (qué tipo de algoritmos y procesos apliquemos), nuestro programa o aplicación será más o menos eficiente y más o menos fiable.[11]

2.2. Introducción a la Inteligencia Artificial y el Aprendizaje Profundo (Deep Learning)

La inteligencia artificial es un campo multidisciplinario de la informática que se centra en la creación de sistemas capaces de realizar tareas que, normalmente, requieren inteligencia humana. Estas tareas incluyen el reconocimiento de voz, el procesamiento del lenguaje natural, la toma de decisiones, y la percepción visual, entre otras. La IA se ha convertido en una parte integral de la tecnología moderna y ha revolucionado numerosos campos, desde la medicina hasta el transporte.

La IA se divide en dos categorías principales: IA débil e IA fuerte. La IA débil, también conocida como IA estrecha, está diseñada para realizar

tareas específicas, como el reconocimiento facial o la recomendación de productos. Por otro lado, la IA fuerte, también conocida como IA general, tiene la capacidad de realizar cualquier tarea cognitiva que un ser humano pueda realizar. Aunque la IA fuerte sigue siendo un objetivo a largo plazo, la IA débil ha logrado avances significativos y se utiliza ampliamente en aplicaciones cotidianas.

Historia de la IA

La historia de la IA es un viaje a través de décadas de innovación y descubrimiento, marcado por altibajos, avances tecnológicos y un creciente entendimiento de la inteligencia misma. Los orígenes de la IA pueden rastrearse hasta la antigüedad, cuando filósofos y matemáticos comenzaron a contemplar la naturaleza del pensamiento y la posibilidad de que las máquinas pudieran replicarlo. Sin embargo, el desarrollo moderno de la IA comenzó a tomar forma en el siglo XX.

El término "inteligencia artificial"fue acuñado en 1956 por John McCarthy, Marvin Minsky, Nathaniel Rochester y Claude Shannon en la Conferencia de Dartmouth, considerada el punto de partida oficial del campo. Durante esta conferencia, se propuso la idea de que çada aspecto del aprendizaje o cualquier otra característica de la inteligencia puede, en principio, ser descrito tan precisamente que una máquina puede ser construida para simularlo". Este evento marcó el inicio de una nueva era de investigación y desarrollo en IA.

En las décadas de 1950 y 1960, los investigadores comenzaron a desarrollar los primeros programas de IA. Uno de los primeros logros fue el programa Logic Theorist, creado por Allen Newell y Herbert A. Simon en 1955. Este programa fue capaz de demostrar teoremas en lógica matemática y se considera uno de los primeros programas de IA verdaderamente funcionales. Otro avance significativo fue el programa General Problem Solver (GPS), también desarrollado por Newell y Simon, que intentaba resolver problemas de manera generalizada, utilizando una estrategia basada en heurísticas.

Durante este período, se desarrollaron también los primeros lenguajes de programación diseñados específicamente para la IA. LISP, creado por John McCarthy en 1958, se convirtió en uno de los lenguajes más importantes y duraderos en el campo de la IA. LISP permitió a los investigadores experimentar con nuevas ideas y algoritmos, sentando las bases para muchos avances futuros.

A pesar de estos éxitos tempranos, la IA pronto enfrentó sus primeros desafíos significativos. A finales de la década de 1960 y principios de la de 1970, los investigadores se encontraron con el problema de la "explosión combinatoria", donde la cantidad de posibilidades a considerar en ciertos problemas crecía exponencialmente, haciendo que los programas fueran ineficientes e impracticables. Además, las expectativas iniciales sobre la rapidez con que se lograrían avances significativos en IA eran extremadamente optimistas, lo que llevó a una disminución en la financiación y el interés, un período conocido como el primer "invierno de la IA".

A pesar de estos desafíos, la investigación en IA continuó. En la década de 1970, se desarrollaron los sistemas expertos, que eran programas diseñados para emular la toma de decisiones humanas en dominios específicos. MYCIN, desarrollado en la Universidad de Stanford, fue uno de los sistemas expertos más famosos, utilizado para diagnosticar infecciones bacterianas y recomendar tratamientos. Estos sistemas demostraron que la IA podía ser útil en aplicaciones prácticas, aunque todavía estaba limitada a dominios muy específicos.

La década de 1980 vio un resurgimiento en la investigación de IA, impulsado por la proliferación de computadoras personales y avances en el hardware de computación. Se desarrollaron nuevas técnicas y algoritmos, y el campo se benefició de una renovada inversión tanto del sector público como del privado. Sin embargo, este período también enfrentó sus desafíos, y muchos sistemas de IA todavía carecían de la flexibilidad y adaptabilidad necesarias para aplicaciones más generales.

El verdadero renacimiento de la IA comenzó a finales de la década de 1990 y principios de la década de 2000, con la explosión de datos disponibles gracias a internet y el aumento de la potencia computacional. Este período marcó el inicio de la era del aprendizaje automático (machine learning) y, más específicamente, del aprendizaje profundo (deep learning). Las redes neuronales, que habían sido propuestas en la década de 1940 pero que habían caído en desgracia, resurgieron con fuerza gracias a la disponibilidad de grandes cantidades de datos y hardware más potente, como las unidades de procesamiento gráfico (GPU).

Uno de los hitos más importantes en esta nueva era fue la victoria de Deep Blue, una computadora de IBM, sobre el campeón mundial de ajedrez Garry Kasparov en 1997, la cual se aprecia en la Figura 2.3. Este evento mostró el poder de los sistemas de IA para superar a los humanos en tareas altamente complejas. A partir de entonces, los avances en IA se aceleraron rápidamente.

Figura 2.3: Gasparov vs Deep Blue. Fuente: Google Imágenes [6]

En 2012, un equipo de investigadores de la Universidad de Toronto, liderado por Geoffrey Hinton, presentó un modelo de red neuronal profunda que ganó el concurso de reconocimiento de imágenes ImageNet. Este avance demostró la eficacia del aprendizaje profundo para tareas de visión por computadora y marcó el comienzo de una nueva era en la IA. Las redes neuronales profundas comenzaron a ser aplicadas en una amplia variedad de campos, incluyendo el procesamiento del lenguaje natural, el reconocimiento de voz y la robótica.

Desde entonces, la IA ha avanzado a pasos agigantados. Empresas tecnológicas como Google, Facebook, Amazon y Microsoft han invertido miles de millones de dólares en investigación y desarrollo de IA, y han incorporado estas tecnologías en sus productos y servicios. Los asistentes virtuales, como Siri, Alexa y Google Assistant, se han vuelto comunes en los hogares, utilizando IA para entender y responder a las consultas de los usuarios. Los vehículos autónomos, impulsados por IA, están en desarrollo avanzado y prometen transformar la industria del transporte.

Hoy en día, la IA está en el centro de numerosas innovaciones tecnológicas. Desde la medicina, donde se utiliza para diagnosticar enfermedades y personalizar tratamientos, hasta la agricultura, donde optimiza la producción de cultivos, la IA está cambiando la forma en que vivimos y trabajamos. Sin embargo, a medida que la IA continúa avanzando, también plantea desafíos éticos y sociales significativos. La privacidad, la seguridad y el impacto en el empleo son cuestiones cruciales que deben abordarse a medida que seguimos integrando la IA en nuestras vidas[12].

2.2.1. Aprendizaje Automático (Machine Learning)

El aprendizaje automático (Machine Learning) es una subdisciplina de la inteligencia artificial que se centra en el desarrollo de algoritmos y técnicas que permiten a las computadoras aprender de los datos y mejorar su rendimiento con el tiempo sin ser explícitamente programadas para cada tarea específica. En esencia, el aprendizaje automático busca habilitar a las máquinas para que identifiquen patrones, hagan predicciones y tomen decisiones basadas en datos. Este enfoque ha transformado numerosos campos, desde la medicina hasta el comercio electrónico, pasando por la seguridad y la investigación científica.

El aprendizaje automático se basa en la premisa de que los sistemas pueden aprender a partir de los datos, identificar patrones y tomar decisiones con una mínima intervención humana. Esto se logra a través de la construcción de modelos matemáticos a partir de datos de entrenamiento. Una vez entrenados, estos modelos pueden aplicarse a nuevos datos para hacer predicciones o tomar decisiones.

Existen varios tipos de aprendizaje automático, cada uno con sus propios enfoques y aplicaciones:

1. Aprendizaje supervisado: En este enfoque, el modelo se entrena con un conjunto de datos etiquetados, es decir, datos que incluyen tanto las entradas como las salidas deseadas. El objetivo es aprender una función que mapea entradas a salidas, de modo que pueda predecir la salida correspondiente para nuevas entradas. Ejemplos de tareas de aprendizaje supervisado incluyen la clasificación (como el reconocimiento de imágenes) y la regresión (como la predicción de precios de viviendas).

2. Aprendizaje no supervisado: Aquí, el modelo se entrena con datos que no están etiquetados. En lugar de predecir una salida específica, el objetivo es descubrir estructuras o patrones ocultos en los datos. Las técnicas de aprendizaje no supervisado incluyen el clustering (agrupamiento) y la reducción de dimensionalidad. El clustering se utiliza para agrupar datos similares, mientras que la reducción de dimensionalidad se utiliza para simplificar los datos complejos mientras se preserva la mayor cantidad posible de información relevante.

3. Aprendizaje semisupervisado: Este enfoque es una combinación de aprendizaje supervisado y no supervisado. Utiliza un pequeño conjunto de datos etiquetados y un gran conjunto de datos no etiquetados para entrenar el modelo. Esto es útil cuando la obtención de datos etiquetados es costosa o laboriosa, pero los datos no etiquetados son abundantes.

4. Aprendizaje por refuerzo: En el aprendizaje por refuerzo, un agente aprende a tomar decisiones secuenciales interactuando con un entorno. El agente recibe recompensas o castigos en función de las acciones que realiza y su objetivo es maximizar la recompensa acumulada a lo largo del tiempo. Este enfoque se utiliza comúnmente en robótica, juegos y sistemas de recomendación.

El aprendizaje automático implica varios componentes y etapas clave:

Preprocesamiento de datos: Los datos en bruto a menudo contienen ruido, valores faltantes y otros problemas que deben ser tratados antes de que puedan ser utilizados para entrenar un modelo. El preprocesamiento de datos incluye la limpieza, normalización y transformación de los datos para mejorar la calidad del modelo.

Selección de características: No todas las características de los datos son igualmente importantes para el modelo. La selección de características implica identificar y seleccionar las características más relevantes que contribuyen significativamente a la predicción o clasificación del modelo.

Entrenamiento del modelo: Durante esta etapa, el modelo se entrena utilizando datos de entrenamiento. Los parámetros del modelo se ajustan iterativamente para minimizar el error en las predicciones. Este proceso puede implicar el uso de técnicas de optimización como el descenso de gradiente.

Evaluación del modelo: Una vez entrenado, el modelo se evalúa utilizando un conjunto de datos de validación o prueba que no se utilizaron durante el entrenamiento. Las métricas comunes para evaluar el rendimiento del modelo incluyen la precisión, la precisión, el recall, la curva ROC y la métrica F1.

Tuning del modelo: A menudo, los modelos pueden mejorarse ajustando sus hiperparámetros. Este proceso, conocido como ajuste de hiperparámetros, puede implicar la búsqueda de cuadrículas, la búsqueda aleatoria o técnicas más avanzadas como la optimización bayesiana.

Implementación y mantenimiento: Una vez que un modelo ha sido entrenado y evaluado, se implementa en un entorno de producción donde puede hacer predicciones sobre nuevos datos. El modelo debe ser monitoreado y mantenido para asegurarse de que siga siendo preciso y relevante a medida que cambian los datos y el entorno[13].

El aprendizaje automático ha tenido un impacto profundo en muchas áreas. En la medicina, se utiliza para el diagnóstico de enfermedades, la personalización de tratamientos y el descubrimiento de medicamentos. En el comercio electrónico, los sistemas de recomendación impulsados por

aprendizaje automático ayudan a personalizar la experiencia del usuario y aumentar las ventas. En la industria automotriz, el aprendizaje automático es fundamental para el desarrollo de vehículos autónomos. En el ámbito financiero, se utiliza para la detección de fraudes, la gestión de riesgos y la toma de decisiones de inversión.

A pesar de sus numerosos éxitos, el aprendizaje automático también enfrenta desafíos significativos. La necesidad de grandes volúmenes de datos de alta calidad es uno de los principales obstáculos, ya que los modelos precisos dependen de datos de entrenamiento representativos. Además, los modelos de aprendizaje automático a menudo se consideran cajas negras debido a su falta de interpretabilidad, lo que plantea desafíos en aplicaciones donde la transparencia y la explicación de las decisiones son cruciales. También existen preocupaciones éticas y de privacidad relacionadas con el uso de datos personales y las decisiones automatizadas.

2.2.2. Aprendizaje Profundo (Deep Learning)

El aprendizaje profundo (Deep Learning) es una subdisciplina del aprendizaje automático que se centra en el uso de redes neuronales artificiales con múltiples capas para modelar y resolver problemas complejos. Inspirado en la estructura y el funcionamiento del cerebro humano, el aprendizaje profundo ha revolucionado numerosos campos, desde la visión por computadora hasta el procesamiento del lenguaje natural, impulsado por su capacidad para aprender representaciones jerárquicas de datos.

Las redes neuronales profundas consisten en capas de neuronas artificiales interconectadas, donde cada capa aprende una representación diferente de los datos de entrada. Las capas superiores de la red capturan características de bajo nivel, como bordes y texturas en una imagen, mientras que las capas más profundas capturan características de alto nivel, como objetos y formas. Esta capacidad de aprender representaciones a múltiples niveles es lo que permite al aprendizaje profundo manejar tareas complejas de manera eficaz.

Una de las arquitecturas más comunes en el aprendizaje profundo es la red neuronal convolucional (CNN), especialmente eficaz para el procesamiento de datos con estructura de cuadrícula, como las imágenes. Las CNN utilizan capas convolucionales para extraer características espaciales y son ampliamente utilizadas en tareas de clasificación de imágenes, detección de objetos y segmentación semántica. Otra arquitectura clave es la red neuronal recurrente (RNN), diseñada para procesar datos secuenciales,

como el texto y las series temporales. Las RNN pueden mantener un estado interno que captura información histórica, lo que las hace efectivas para tareas como la traducción automática y la predicción de series temporales.

El aprendizaje profundo ha sido posible gracias a varios factores clave. Primero, la disponibilidad de grandes volúmenes de datos, como los generados por Internet y los dispositivos móviles, ha proporcionado el material necesario para entrenar redes neuronales profundas. Segundo, los avances en hardware, particularmente las unidades de procesamiento gráfico (GPU), han permitido el entrenamiento eficiente de modelos complejos en tiempos razonables. Finalmente, los avances en algoritmos y técnicas de optimización, como el descenso de gradiente estocástico y la normalización por lotes, han mejorado la eficacia y estabilidad del entrenamiento de redes profundas.

Las aplicaciones del aprendizaje profundo son vastas y diversas. En la visión por computadora, las CNN han permitido avances significativos en tareas como el reconocimiento de imágenes y la detección de objetos. En el procesamiento del lenguaje natural, las arquitecturas basadas en transformadores, como BERT y GPT, han revolucionado tareas como la generación de texto y la traducción automática. En el reconocimiento de voz, los modelos de aprendizaje profundo han mejorado la precisión de los sistemas de transcripción y asistentes virtuales.

Una de las técnicas más innovadoras en el aprendizaje profundo es el uso de redes generativas adversariales (GAN). Las GAN consisten en dos redes neuronales que compiten entre sí: un generador que crea datos falsos y un discriminador que intenta distinguir entre datos reales y falsos. Esta competencia lleva a la mejora continua de ambas redes, lo que permite generar datos sintéticos realistas, como imágenes y videos[14].

A pesar de sus éxitos, el aprendizaje profundo enfrenta varios desafíos. La interpretabilidad de los modelos sigue siendo un problema importante, ya que las redes neuronales profundas a menudo se consideran cajas negras. La necesidad de grandes volúmenes de datos etiquetados para el entrenamiento es otra limitación, especialmente en campos donde la recopilación de datos es costosa o difícil. Además, el alto costo computacional del entrenamiento de modelos profundos puede ser prohibitivo para algunas aplicaciones.

Para abordar estos desafíos, los investigadores están explorando varias direcciones. El aprendizaje auto-supervisado, donde los modelos se entrenan utilizando datos sin etiquetar, ha mostrado resultados prometedores y podría reducir la dependencia de los datos etiquetados. El aprendizaje

federado permite el entrenamiento de modelos en datos descentralizados, preservando la privacidad de los datos individuales. Además, se están desarrollando nuevas técnicas para mejorar la interpretabilidad de los modelos, proporcionando explicaciones más claras de cómo toman decisiones.

2.3. Técnicas y algoritmos comunes en el análisis de vídeo y detección de objetos

El análisis de video y la detección de objetos son áreas cruciales en la visión por computadora, con aplicaciones que van desde la seguridad y vigilancia hasta los vehículos autónomos y la realidad aumentada. Estas técnicas implican la identificación y seguimiento de objetos de interés en secuencias de video en tiempo real o grabadas, permitiendo una interpretación y comprensión más profunda de la escena visual.

Existen varias técnicas y algoritmos comunes utilizados en el análisis de video y detección de objetos:

1. Redes Neuronales Convolucionales (CNN): Las CNN son la columna vertebral de muchos sistemas de detección de objetos. Estas redes están diseñadas para procesar datos en forma de cuadrícula, como las imágenes, y son capaces de aprender características espaciales a través de capas convolucionales. Las arquitecturas CNN populares incluyen AlexNet, VGG, ResNet y DenseNet, que han demostrado un rendimiento notable en tareas de clasificación y detección de imágenes.

2. You Only Look Once (YOLO): YOLO es un algoritmo de detección de objetos en tiempo real que trata la detección de objetos como un problema de regresión único. En lugar de seleccionar regiones de interés y pasar cada una a través de una CNN, YOLO pasa toda la imagen a través de la red, dividiéndola en una cuadrícula y prediciendo simultáneamente los cuadros delimitadores y las probabilidades de clase para cada región. YOLO es conocido por su velocidad y precisión, lo que lo hace adecuado para aplicaciones en tiempo real.

3. Single Shot MultiBox Detector (SSD): SSD es otro método popular para la detección de objetos en tiempo real. A diferencia de YOLO, SSD utiliza múltiples mapas de características de diferentes resoluciones para realizar predicciones de objetos, lo que le permite manejar objetos de varios tamaños de manera más efectiva. SSD es conocido por su equilibrio entre precisión y velocidad.

4. R-CNN y sus variantes: La familia de algoritmos R-CNN (Region-

based Convolutional Neural Networks) incluye R-CNN, Fast R-CNN y Faster R-CNN. Estos métodos combinan propuestas de regiones con CNN para detectar objetos. Faster R-CNN, en particular, introduce una red de propuestas de región (RPN) que mejora significativamente la velocidad de detección al generar propuestas de regiones de manera más eficiente.

5. Detectores basados en puntos clave: Los detectores basados en puntos clave, como CornerNet y CenterNet, se centran en detectar puntos clave de objetos (por ejemplo, esquinas o centros) y luego agrupar estos puntos clave para formar cuadros delimitadores. Estos métodos pueden ser efectivos para detectar objetos en escenas complejas donde los métodos basados en regiones pueden fallar.

6. Seguimiento de objetos: El seguimiento de objetos implica el seguimiento continuo de los objetos de interés a lo largo de una secuencia de video. Los algoritmos de seguimiento populares incluyen el filtro de partículas, el filtro de Kalman y los métodos de correlación como el MOSSE (Minimum Output Sum of Squared Error). Las técnicas más recientes combinan redes neuronales profundas con métodos tradicionales de seguimiento para mejorar la precisión y la robustez.

7. Redes Neuronales Recurrentes (RNN) y LSTM: Para el análisis de video que requiere la captura de dependencias temporales, como la acción y el reconocimiento de actividad, las RNN y sus variantes como LSTM (Long Short-Term Memory) son esenciales. Estas redes pueden procesar secuencias de datos y mantener información a lo largo del tiempo, lo que las hace adecuadas para tareas que involucran la dinámica temporal.

8. Redes Neuronales Generativas Adversariales (GAN): Las GAN se utilizan en el análisis de video para tareas como la generación de video, la mejora de resolución y la interpolación de fotogramas. En el contexto de la detección de objetos, las GAN pueden ayudar a generar datos de entrenamiento sintéticos para mejorar el rendimiento del modelo.

9. Segmentación de video: La segmentación de video va más allá de la detección de objetos para identificar y delinear cada píxel perteneciente a un objeto en cada fotograma. Las técnicas de segmentación de video incluyen métodos basados en CNN como Mask R-CNN y DeepLab, que pueden proporcionar segmentaciones precisas en tiempo real.

10. Métodos híbridos: Muchos sistemas modernos de análisis de video y detección de objetos combinan múltiples técnicas y algoritmos para aprovechar las fortalezas de cada uno. Por ejemplo, un sistema puede utilizar YOLO para la detección rápida de objetos y luego refinar las detecciones utilizando un modelo más preciso como Faster R-CNN[15].

2.4. Estado del arte en el análisis de flujo de pasajeros y su aplicación en transporte público

El análisis del flujo de pasajeros en el transporte público es un campo de investigación y desarrollo que ha cobrado gran relevancia en las últimas décadas. Este enfoque combina técnicas de inteligencia artificial, visión por computadora, sensores y análisis de datos para entender y mejorar la dinámica de los sistemas de transporte. La capacidad de monitorear y analizar el flujo de pasajeros en tiempo real proporciona información esencial para optimizar la operación y planificación del transporte público, mejorando la eficiencia y la experiencia del usuario.

Actualmente, el estado del arte en el análisis de flujo de pasajeros está marcado por la integración de tecnologías avanzadas y metodologías innovadoras. Los desarrollos recientes en este campo se pueden agrupar en varias categorías principales:

1. Visión por Computadora: La visión por computadora es una de las tecnologías más avanzadas utilizadas para el análisis del flujo de pasajeros. Las cámaras de video instaladas en estaciones y vehículos capturan imágenes y videos que se procesan utilizando algoritmos de visión por computadora. Las redes neuronales convolucionales (CNN) son especialmente efectivas para la detección y el seguimiento de pasajeros. Estos algoritmos pueden identificar y contar personas, analizar sus movimientos y detectar patrones de comportamiento. Los avances en el aprendizaje profundo han mejorado significativamente la precisión y robustez de estos sistemas, permitiendo una monitorización en tiempo real y una mejor gestión del flujo de pasajeros.

2. Sensores Avanzados: Además de las cámaras, se utilizan diversos tipos de sensores para el análisis del flujo de pasajeros. Los sensores infrarrojos y las etiquetas de identificación por radiofrecuencia (RFID) son métodos comunes. Los sensores infrarrojos pueden contar el número de personas que entran y salen de un área determinada, mientras que las etiquetas RFID permiten el seguimiento individual de pasajeros que portan tarjetas o dispositivos RFID. Estos sensores proporcionan datos precisos y pueden funcionar en condiciones donde las cámaras podrían no ser efectivas, como en situaciones de poca luz o con alta densidad de pasajeros.

3. Sistemas de Conteo Automático de Pasajeros (APC): Los sistemas APC están integrados en vehículos de transporte público y utilizan una

combinación de sensores y algoritmos para contar automáticamente el número de pasajeros. Estos sistemas ofrecen datos en tiempo real sobre la ocupación de los vehículos, ayudando a las autoridades de transporte a tomar decisiones informadas sobre la asignación de recursos y la programación de servicios. Los datos de los sistemas APC pueden integrarse con otras fuentes de datos para proporcionar una visión integral del flujo de pasajeros.

4. Análisis de Big Data: La recopilación y el análisis de grandes volúmenes de datos generados por diversas fuentes, como tarjetas de transporte, sensores y redes sociales, permite obtener una visión holística del flujo de pasajeros. Las técnicas de análisis de big data pueden identificar patrones y tendencias, predecir la demanda de transporte y apoyar la toma de decisiones estratégicas. Los algoritmos de aprendizaje automático aplicados a big data pueden analizar grandes conjuntos de datos en tiempo real, proporcionando información valiosa para la optimización del transporte público.

5. Modelado y Simulación: Los modelos matemáticos y las simulaciones por computadora se utilizan para estudiar y predecir el comportamiento del flujo de pasajeros en diferentes escenarios. Estos modelos pueden simular el impacto de cambios en la infraestructura, horarios y políticas de transporte, permitiendo a los planificadores evaluar diferentes estrategias antes de implementarlas. Las simulaciones ayudan a entender cómo los pasajeros interactúan con el sistema de transporte y cómo se pueden mejorar las operaciones para reducir la congestión y mejorar la eficiencia.

6. Aprendizaje Automático y Aprendizaje Profundo: Los algoritmos de aprendizaje automático y profundo se aplican para mejorar la precisión del análisis del flujo de pasajeros. Estos algoritmos pueden aprender de los datos históricos y en tiempo real, adaptándose a cambios en los patrones de comportamiento y proporcionando predicciones precisas sobre la demanda y el uso del transporte. Las redes neuronales recurrentes (RNN) y las redes de memoria a largo corto plazo (LSTM) son particularmente útiles para analizar secuencias temporales de datos y predecir el flujo de pasajeros en el futuro.

7. Aplicaciones en Transporte Público: La aplicación de estas tecnologías en el transporte público ha demostrado ser altamente beneficiosa. Los datos precisos sobre el flujo de pasajeros permiten a las autoridades de transporte optimizar la asignación de vehículos y personal, reducir tiempos de espera y mejorar la eficiencia operativa. El análisis en tiempo

real del flujo de pasajeros puede ayudar a reducir la congestión, mejorar la seguridad y proporcionar información actualizada a los usuarios sobre horarios y disponibilidad de servicios. La capacidad de predecir la demanda y analizar patrones de uso a largo plazo apoya la planificación estratégica, incluyendo la expansión de rutas, la mejora de infraestructuras y la implementación de políticas de transporte sostenibles. Además, el monitoreo continuo del flujo de pasajeros puede mejorar la seguridad y la gestión de emergencias, permitiendo una respuesta rápida a incidentes y la gestión eficaz de evacuaciones.

El análisis del flujo de pasajeros y su aplicación en el transporte público continúan evolucionando, impulsados por los avances en tecnología y la creciente disponibilidad de datos. A medida que las ciudades enfrentan desafíos crecientes en términos de movilidad y sostenibilidad, estas innovaciones jugarán un papel crucial en la creación de sistemas de transporte público más eficientes, seguros y orientados al usuario. El futuro del análisis del flujo de pasajeros promete mejoras continuas en precisión, velocidad y robustez, ampliando aún más sus aplicaciones en diversos campos y contribuyendo significativamente al desarrollo de infraestructuras urbanas inteligentes y sostenibles.

Capítulo 3

Metodología

La metodología elegida para el análisis del flujo de pasajeros en el transporte público se basa en el uso de una Jetson Nano junto con una cámara de Raspberry Pi. En el prototipo final, estas herramientas se utilizarán para capturar imágenes en tiempo real y realizar la detección de pasajeros. Sin embargo, para las fases iniciales de desarrollo y prueba, se emplearán vídeos pregrabados en lugar de la cámara en tiempo real.

3.1. Descripción general del enfoque metodológico

Para este trabajo en concreto se ha optado por seguir la siguiente estructura citada a continuación:

1. **Captura de Datos**

   Inicialmente, se utilizarán vídeos pregrabados de flujo de personas para entrenar y validar el sistema. Estos vídeos simulan las condiciones reales de una estación de tranvía y proporcionan una base sólida para el desarrollo del modelo. En la fase final del proyecto, se integrará una cámara de Raspberry Pi conectada a una Jetson Nano para capturar imágenes en tiempo real en la estación de tranvía.

2. **Detección de Pasajeros**

   Una vez obtenidas las imágenes, ya sean de vídeos pregrabados o capturadas en tiempo real, se procederá a la detección de pasajeros utilizando el modelo de detección de objetos YOLO (You Only Look Once). YOLO es conocido por su alta velocidad y precisión en la detección de objetos, lo que lo hace ideal para aplicaciones en tiempo real. El modelo YOLO será entrenado y ajustado para identificar pasajeros en las imágenes de la estación de tranvía, detectando y delimitando cada persona con cuadros de detección.

3. **Conteo de Pasajeros**

   Después de la detección de pasajeros, se realizará el conteo de personas. Este proceso implicará la identificación y contabilización de cada pasajero detectado en los cuadros delimitadores. El conteo se actualizará en tiempo real para proporcionar información precisa sobre el número de pasajeros presentes en cada momento.

4. **Implementación del Tracker**

   Para saber cuántas personas entran y salen de la estación, se implementará un sistema de seguimiento (tracker). Este sistema rastreará a cada pasajero detectado a lo largo del tiempo, utilizando algoritmos de seguimiento basada en la distancia euclidiana. El tracker permitirá mantener un conteo continuo y actualizado de las entradas y salidas de pasajeros, proporcionando datos valiosos sobre el flujo de personas en la estación.

5. **Análisis y Visualización de Datos**

   Los datos obtenidos del conteo y seguimiento de pasajeros se analizarán y visualizarán mediante paneles de control interactivos. Estos paneles mostrarán información en tiempo real sobre el número de pasajeros que entran y salen, ayudando a los operadores de transporte a gestionar mejor la capacidad y los recursos. Las visualizaciones incluirán gráficos del conteo de entradas y salidas.

6. **Evaluación y Mejora Continua**

   El sistema se evaluará continuamente para identificar áreas de mejora y optimizar su rendimiento. Además, la idea es que se realicen pruebas de campo y se recopilen comentarios de los usuarios para ajustar y mejorar el sistema. La retroalimentación se incorporará en ciclos iterativos para garantizar que el sistema se adapte a las necesidades cambiantes y mejore continuamente su precisión y eficiencia.

   Este enfoque metodológico combina el uso de hardware específico, como la Jetson Nano y la cámara de Raspberry Pi, con técnicas avanzadas de visión por computadora y aprendizaje profundo. Al hacerlo, proporciona una solución robusta y eficiente para el análisis del flujo de pasajeros en el transporte público, mejorando la gestión operativa y la experiencia del usuario.

3.2. Selección de herramientas y tecnologías

3.2.1. Nvidia Jetson Nano B01

La Nvidia Jetson Nano B01 es una plataforma de computación de alto rendimiento diseñada para aplicaciones de inteligencia artificial y aprendizaje profundo. En la Figura 3.1 podemos ver su aspecto, esta pequeña pero potente computadora ofrece la capacidad de ejecutar redes neuronales complejas y realizar tareas de procesamiento de datos en tiempo real, todo en un formato compacto y energéticamente eficiente. A continuación, se detallan algunas de sus características más destacadas:

- **Procesador**: La Jetson Nano B01 cuenta con un procesador Nvidia Maxwell de 128 núcleos CUDA, lo que permite realizar cálculos de procesamiento paralelo de manera eficiente. También incluye un CPU ARM Cortex-A57 de cuatro núcleos.
- **Memoria**: Dispone de 4 GB de memoria LPDDR4, lo cual es adecuado para manejar tareas de IA y aprendizaje profundo.
- **Almacenamiento**: Tiene una ranura para tarjetas microSD para el almacenamiento del sistema operativo y los datos. También cuenta con interfaces de expansión para agregar almacenamiento adicional si es necesario.
- **Conectividad**: Ofrece varias opciones de conectividad, incluyendo:
  - 4 puertos USB 3.0.
  - 1 puerto USB 2.0 Micro-B.
  - 1 puerto Gigabit Ethernet.
  - 1 puerto HDMI.
  - 1 conector de alimentación de barril (5V 4A).
  - Conector GPIO de 40 pines para la integración con otros dispositivos y sensores.
- **Video y gráficos**: La Nvidia Jetson Nano B01 soporta video 4K a 30 fps y 1080p a 60 fps, lo que la hace ideal para aplicaciones de visión por computadora y procesamiento de video en tiempo real.
- **Sistemas operativos**: La Jetson Nano B01 es compatible con Nvidia JetPack SDK, que incluye un sistema operativo basado en Ubuntu

y una colección de herramientas y bibliotecas para el desarrollo de aplicaciones de IA y aprendizaje profundo.

- **Aplicaciones**: Es ideal para una amplia variedad de aplicaciones, incluyendo robots autónomos, drones, sistemas de videovigilancia inteligentes, análisis de video en tiempo real, dispositivos IoT (Internet de las cosas), y cualquier proyecto que requiera capacidades avanzadas de computación en el borde[16].

Figura 3.1: NVIDIA Jetson Nano B01. Fuente: Google Imágenes [7]

3.2.2. Cámara Raspberry Pi v2.1

La cámara Raspberry Pi v2.1 es un módulo de cámara de alta calidad diseñado específicamente para las computadoras de placa única Raspberry Pi o compatible con otras placas como la Jetson Nano. Ofrece una solución económica y eficiente para capturar imágenes y videos en una amplia variedad de proyectos. En la Figura 3.2 se muestra cómo es. A continuación se detallan las características principales de este módulo de cámara:

- **Sensor**: La cámara Raspberry Pi v2.1 está equipada con un sensor de imagen Sony IMX219. Este sensor CMOS es de tipo *back-illuminated* (retroiluminado), lo que mejora la sensibilidad a la luz y la calidad de la imagen.

- **Resolución**: El sensor IMX219 ofrece una resolución de 8 megapíxeles, capaz de capturar imágenes fijas de hasta 3280 x 2464 píxeles. Esto proporciona una gran cantidad de detalles en las imágenes capturadas.

- **Video**: El módulo de cámara puede grabar video en varias resoluciones y velocidades de fotogramas, incluyendo:
  - 1080p a 30 fps (fotogramas por segundo).
  - 720p a 60 fps.
  - 480p a 90 fps.

  Esta versatilidad hace que sea adecuado para aplicaciones que van desde videoconferencias hasta captura de video de alta velocidad.
- **Lente**: La cámara tiene una lente de enfoque fijo con una distancia focal de aproximadamente 3.04 mm y una apertura f/2.0, lo que permite una buena captación de luz y profundidad de campo.
- **Conectividad**: La cámara se conecta a la Raspberry Pi a través del puerto CSI (*Camera Serial Interface*), utilizando un cable de cinta incluido en el paquete. Esta conexión directa asegura una transferencia rápida y eficiente de datos entre la cámara y la Raspberry Pi.
- **Compatibilidad**: El módulo de cámara Raspberry Pi v2.1 es compatible con todas las versiones de la Raspberry Pi que tienen un puerto CSI, incluyendo la Raspberry Pi 1, 2, 3, 4 y Zero.
- **Tamaño y peso**: El módulo de cámara es compacto y ligero, lo que lo hace ideal para proyectos que requieren un dispositivo pequeño y discreto. Las dimensiones aproximadas son 23.86 x 25.4 x 9 mm y su peso es de solo unos pocos gramos.
- **Aplicaciones**: Gracias a sus capacidades y tamaño, la cámara Raspberry Pi v2.1 se puede utilizar en una amplia variedad de proyectos, tales como:[17]
  - Vigilancia y seguridad.
  - Proyectos de robótica.
  - Fotografía y video en *timelapse*.
  - Sistemas de visión artificial.
  - Aplicaciones de IoT (*Internet de las cosas*).

La cámara Raspberry Pi v2.1 es una herramienta versátil y potente para cualquier entusiasta, desarrollador o investigador que desee agregar capacidades de captura de imagen y video a sus proyectos basados en Raspberry Pi.

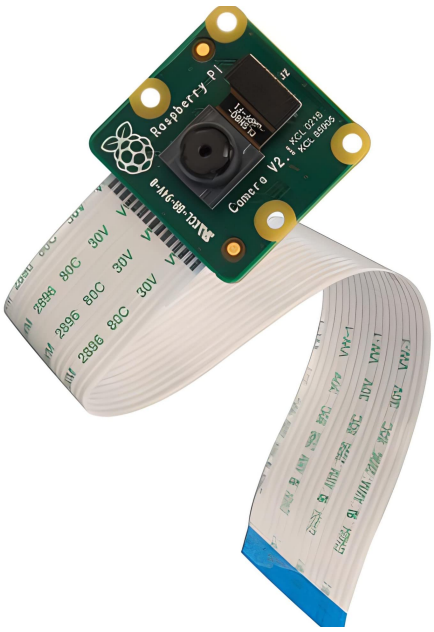

Figura 3.2: Cámara Raspberry Pi V2.1. Fuente: Google Imágenes [8]

3.2.3. Tecnologías Utilizadas

- **Lenguaje de Programación**: Python. Python es un lenguaje de programación de alto nivel, conocido por su sintaxis clara y su facilidad de uso. Es ampliamente utilizado en el desarrollo de aplicaciones de inteligencia artificial, procesamiento de datos, y muchas otras áreas debido a su extensa biblioteca estándar y el soporte de una gran comunidad.
- **Bibliotecas**:
  - **OpenCV**: Utilizada para el procesamiento de imágenes y videos. OpenCV (Open Source Computer Vision Library) es una biblioteca de visión por computadora de código abierto que proporciona más de 2,500 algoritmos optimizados. Es ideal para tareas como el reconocimiento facial, la detección de objetos y el procesamiento de imágenes en general.
  - **Pandas**: Utilizada para la manipulación y análisis de datos. Pandas es una biblioteca poderosa y flexible que proporciona estructuras de datos de alto rendimiento, como DataFrames, que son esenciales para el análisis y manipulación de datos tabulares y heterogéneos. Es muy popular en análisis de datos, estadística y aprendizaje automático.
  - **NumPy**: Utilizada para operaciones numéricas. NumPy (Numerical Python) es una biblioteca fundamental para la computación

científica en Python. Proporciona soporte para grandes arreglos y matrices multidimensionales, junto con una colección de funciones matemáticas de alto nivel para operar con estos datos de manera eficiente.

- **Ultralytics YOLO**: Utilizada para la detección de objetos. Ultralytics YOLO es una implementación avanzada del algoritmo YOLO (You Only Look Once), que es uno de los métodos más populares para la detección de objetos en tiempo real. Esta biblioteca permite realizar detecciones precisas y rápidas, siendo adecuada para aplicaciones en seguridad, automatización y análisis de video.

- **Modelo de Detección**: YOLOv8s. YOLOv8s es una versión mejorada del modelo YOLO, optimizada para proporcionar un equilibrio óptimo entre precisión y velocidad. Este modelo es capaz de identificar y localizar múltiples objetos dentro de una imagen o un video en tiempo real, lo que lo hace ideal para aplicaciones que requieren respuestas rápidas y precisas, en la Figura 3.3 se puede ver un ejemplo de la detección de personas y motocicletas[18].

Figura 3.3: Ejemplo de detección con el modelo YOLOv8. Fuente: Google Imágenes [9]

- **Algoritmo de Seguimiento**: Implementación personalizada basada en la distancia euclidiana. Este algoritmo de seguimiento es desarrollado a medida y se basa en el cálculo de la distancia euclidiana para rastrear objetos entre fotogramas consecutivos en un video. La distancia euclidiana mide la separación directa entre dos puntos en el espacio, lo que permite seguir de manera precisa los movimientos de los objetos detectados a lo largo del tiempo, manteniendo la consistencia de la identificación de los mismos objetos en diferentes fotogramas.

Capítulo 4

Desarrollo del proyecto

4.1. Arquitectura del Sistema

El sistema está diseñado con una arquitectura modular que permite una fácil integración y mantenimiento. Los componentes principales del sistema incluyen:

1. **Captura de Video**: La captura de video es el primer paso en la arquitectura del sistema. Para ello, se utiliza la biblioteca OpenCV, que proporciona herramientas robustas para la captura y procesamiento de video. OpenCV se encarga de capturar frames del video de entrada de manera continua, permitiendo el análisis en tiempo real. Este componente es crucial ya que proporciona los datos brutos sobre los que se realizarán las detecciones y seguimientos de objetos.

2. **Detección de Objetos**: Una vez capturados los frames, el siguiente paso es la detección de objetos. Aquí, el modelo YOLOv8 se emplea para identificar personas en cada frame. YOLO (You Only Look Once) es conocido por su capacidad de realizar detecciones rápidas y precisas, analizando la imagen completa en una sola pasada. En este proyecto, YOLOv8 se entrena específicamente para detectar personas, asegurando así una alta precisión en la identificación de individuos en el video.

3. **Seguimiento de Objetos**: Después de detectar las personas en los frames individuales, es necesario mantener la identificación de estas personas entre frames consecutivos. Para ello, se implementa un algoritmo de seguimiento personalizado basado en la distancia euclidiana. Este algoritmo compara las posiciones de las personas detectadas en un frame con las del siguiente, permitiendo seguir los movimientos de cada individuo de manera precisa a lo largo del tiempo.

4. **Definición de Áreas de Interés**: Para contabilizar las personas que entran y salen de una determinada área, se definen áreas específicas dentro del frame. Estas áreas de interés son esenciales para el análisis del flujo de personas. El sistema monitorea estas áreas para detectar cuando una persona cruza los límites definidos, permitiendo así contar de manera precisa las entradas y salidas.

5. **Visualización y Salida**: Finalmente, los resultados del procesamiento se visualizan en tiempo real. OpenCV se utiliza nuevamente para mostrar los frames del video con las detecciones y seguimientos superpuestos, permitiendo una fácil interpretación de los datos. Además, se generan estadísticas que indican el número de personas que han entrado y salido de las áreas de interés definidas. Estas estadísticas se pueden imprimir en pantalla o guardar en un archivo para un análisis posterior.

Cada uno de estos componentes se integra de manera modular, permitiendo que el sistema sea flexible y escalable. Esta arquitectura modular también facilita el mantenimiento y la actualización del sistema, ya que cada componente se puede modificar o mejorar de manera independiente sin afectar a los demás.

4.2. Descripción del código principal

El programa se divide en dos partes principales: el script principal y el módulo de seguimiento (tracker). Se pueden consultar los algoritmos al completo en el Apéndice. A continuación, se describe detalladamente cada una de estas partes, su función y cómo interactúan entre sí.

El script principal es el componente central del programa y se encarga de varias tareas cruciales, que incluyen la captura de vídeo, la detección de objetos, la definición de áreas de interés y la visualización de los resultados en tiempo real. Aquí se detalla cada una de estas funciones:

4.2.1. Importación de Bibliotecas Necesarias

Se importan las bibliotecas necesarias para el procesamiento de vídeo, la manipulación de datos y la detección de objetos.

```
1    import cv2
2    import pandas as pd
3    import numpy as np
4    from ultralytics import YOLO
```

```
5     from tracker import Tracker
```

Listing 4.1: Importación de bibliotecas

4.2.2. Carga del Modelo YOLOv8

Se carga el modelo YOLOv8, que se utilizará para la detección de objetos en los fotogramas del vídeo.

```
1     model = YOLO('yolov8s.pt')
```

Listing 4.2: Carga del modelo YOLOv8

4.2.3. Definición de Áreas de Interés

Se definen las áreas de interés (AOI) en el vídeo, que se utilizarán para determinar la entrada y salida de personas. Estas áreas están delineadas por polígonos.

```
1     area1 = [(621, 720), (585, 720), (472, 400), (486, 400)]
2     area2 = [(696, 720), (656, 720), (498, 400), (512, 400)]
```

Listing 4.3: Definición de áreas de interés

4.2.4. Función para Obtener Coordenadas RGB

Esta función se utiliza para capturar las coordenadas RGB cuando se mueve el ratón sobre la ventana de visualización. Aunque está comentada, puede ser útil para depuración.

```
1     def RGB(event, x, y, flags, param):
2     if event == cv2.EVENT_MOUSEMOVE:
3         colorsBGR = [x, y]
4     #   print(colorsBGR)  # Uncomment to print BGR coordinates
```

Listing 4.4: Función para obtener coordenadas RGB

4.2.5. Configuración de la Ventana de Visualización

Se configura la ventana de visualización y se establece una función de callback para el ratón.

```
1     cv2.namedWindow('RGB')
2     cv2.setMouseCallback('RGB', RGB)
```

Listing 4.5: Configuración de la ventana de visualización

4.2.6. Apertura del Vídeo

Se abre el archivo de vídeo que se procesará.

```
cap = cv2.VideoCapture('estacion_tren.mp4')
```

Listing 4.6: Apertura del vídeo

4.2.7. Lectura del Archivo de Clases COCO

Se lee un archivo que contiene las clases del modelo COCO, lo que permite identificar las clases de los objetos detectados.

```
my_file = open("coco.txt", "r")
data = my_file.read()
class_list = data.split("\n")
```

Listing 4.7: Lectura del archivo de clases COCO

4.2.8. Inicialización del Contador de Fotogramas y Tracker

Se inicializa un contador de fotogramas y un objeto tracker para seguir a las personas detectadas.

```
count = 0
tracker = Tracker()
```

Listing 4.8: Inicialización del contador de fotogramas y tracker

4.2.9. Diccionarios y Conjuntos para Seguimiento de Personas

Se crean estructuras de datos para rastrear las personas que entran y salen de las áreas de interés.

```
people_entering = {}
entering = set()
people_exiting = {}
exiting = set()
```

Listing 4.9: Diccionarios y conjuntos para seguimiento de personas

4.2.10. Configuración para la Creación del Vídeo de Salida

Se configura el códec y el archivo de salida para guardar el vídeo procesado.

```
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output.mp4', fourcc, 10.0, (1280, 720))
```

Listing 4.10: Configuración para la creación del vídeo de salida

4.2.11. Bucle Principal de Procesamiento de Vídeo

Este bucle procesa el vídeo fotograma a fotograma. Se detallan las principales tareas dentro del bucle:

Lectura de un Fotograma

Se lee un fotograma del vídeo.

```
    ret, frame = cap.read()
    if not ret:
        break
```

Listing 4.11: Lectura de un fotograma

Procesamiento de Cada Segundo Fotograma

Se incrementa el contador de fotogramas y se procesan solo los fotogramas pares para reducir la carga de procesamiento.

```
    count += 1
    if count % 2 != 0:
        continue
```

Listing 4.12: Procesamiento de cada segundo fotograma

Redimensionamiento del Fotograma

Se redimensiona el fotograma a una resolución fija.

```
    frame = cv2.resize(frame, (1280, 720))
```

Listing 4.13: Redimensionamiento del fotograma

Detección de Objetos

Se utiliza el modelo YOLO para detectar objetos en el fotograma y se convierte el resultado a un DataFrame para facilitar su manipulación.

```
    results = model.predict(frame)
    a = results[0].boxes.data
    px = pd.DataFrame(a).astype("float")
```

Listing 4.14: Detección de objetos

Dibujo de Áreas de Interés

Se dibujan los polígonos que delimitan las áreas de interés en el fotograma.

```
    cv2.polylines(frame, [np.array(area1, np.int32)], True, (255, 0, 0), 2)
    cv2.polylines(frame, [np.array(area2, np.int32)], True, (255, 0, 0), 2)
```

Listing 4.15: Dibujo de áreas de interés

Filtrado de Detecciones de Personas

Se filtran las detecciones para quedarse solo con las personas y se almacenan en una lista.

```
    for index, row in px.iterrows():
        x1 = int(row[0])
        y1 = int(row[1])
        x2 = int(row[2])
        y2 = int(row[3])
        d = int(row[5])
        c = class_list[d]
        if 'person' in c:
            list.append([x1, y1, x2, y2])
```

Listing 4.16: Filtrado de detecciones de personas

Actualización del Tracker

Se actualiza el tracker con las nuevas detecciones de personas.

```
    bbox_id = tracker.update(list)
```

Listing 4.17: Actualización del tracker

Detección de Entrada y Salida

Se utiliza el tracker para determinar si una persona entra o sale de las áreas de interés.

```
    for bbox in bbox_id:
    x3, y3, x4, y4, id = bbox
    # Check if a person is entering
    results1 = cv2.pointPolygonTest(np.array(area1, np.int32), ((x4, y4)),False)
    if results1 >= 0:
        people_entering[id] = (x4, y4)
        if id in people_entering:
        results2 = cv2.pointPolygonTest(np.array(area2, np.int32),((x4, y4)),False)
    if results2 >= 0:
        entering.add(id)

    # Check if a person is exiting
    results3 = cv2.pointPolygonTest(np.array(area2, np.int32), ((x4, y4)),False)
    if results3 >= 0:
        people_exiting[id] = (x4, y4)
        if id in people_exiting:
            results4 = cv2.pointPolygonTest(np.array(area1, np.int32), ((x4, y4)),False)
    if results4 >= 0:
```

```
19      exiting.add(id)
```

Listing 4.18: Detección de entrada y salida

Dibujo de Rectángulos y IDs

Se dibujan rectángulos alrededor de las personas detectadas y se muestra su ID.

```
1      cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)
2      cv2.circle(frame, (x4, y4), 2, (255, 0, 255), -1)
3      cv2.putText(frame, str(id), (x3, y3), cv2.FONT_HERSHEY_COMPLEX,(0.5),(255, 255, 255),
           1)
```

Listing 4.19: Dibujo de rectángulos y IDs

Visualización de Contadores de Entrada y Salida

Se muestran los contadores de personas que entran y salen en el fotograma.

```
1      text_enter = 'entering: ' + str(len(entering))
2      text_exit = 'exiting: ' + str(len(exiting))
3      cv2.putText(frame, text_enter, (50, 50), cv2.FONT_HERSHEY_COMPLEX, (0.75),(255, 255,
           255), 1)
4      cv2.putText(frame, text_exit, (50, 75), cv2.FONT_HERSHEY_COMPLEX, (0.75),(255, 255,
           255), 1)
```

Listing 4.20: Visualización de contadores de entrada y salida

Escritura y Visualización del Fotograma

Se escribe el fotograma procesado en el archivo de salida y se muestra en la ventana de visualización.

```
1      out.write(frame)
2      cv2.imshow("RGB", frame)
3      if cv2.waitKey(1) & 0xFF == 27:
4          break
```

Listing 4.21: Escritura y visualización del fotograma

4.2.12. Liberación de Recursos

Se liberan todos los recursos al finalizar el procesamiento del vídeo.

```
1      cap.release()
2      out.release()
3      cv2.destroyAllWindows()
```

Listing 4.22: Liberación de recursos

4.2.13. Tracker

El módulo de seguimiento, denominado “tracker”, es una parte fundamental del programa que se encarga de rastrear las personas detectadas a lo largo de los fotogramas del vídeo. Este módulo se invoca desde el script principal y proporciona varias funcionalidades clave:

Inicialización del Tracker

Se crea una instancia de la clase Tracker, que es responsable de gestionar el seguimiento de los objetos.

```
1     tracker = Tracker()
```

Listing 4.23: Inicialización del tracker

4.2.14. Actualización del Tracker con Nuevas Detecciones

En cada fotograma, las nuevas detecciones de personas se pasan al tracker para actualizar el estado de los objetos seguidos. El tracker asigna un identificador único a cada persona y mantiene su estado a lo largo del tiempo.

```
1     bbox_id = tracker.update(list)
```

Listing 4.24: Actualización del tracker con nuevas detecciones

4.2.15. Determinación de Entrada y Salida

El tracker utiliza las áreas de interés definidas en el script principal para determinar cuándo una persona entra o sale de una zona específica. Esto se hace comparando las coordenadas de los objetos seguidos con las áreas de interés.

```
1     results1 = cv2.pointPolygonTest(np.array(area1, np.int32),((x4, y4)),False)
2     if results1 >= 0:
3         people_entering[id] = (x4, y4)
4         if id in people_entering:
5             results2 = cv2.pointPolygonTest(np.array(area2, np.int32), ((x4, y4)),False)
6     if results2 >= 0:
7         entering.add(id)
8
9     results3 = cv2.pointPolygonTest(np.array(area2, np.int32), ((x4, y4)),False)
10    if results3 >= 0:
11        people_exiting[id] = (x4, y4)
12        if id in people_exiting:
13            results4 = cv2.pointPolygonTest(np.array(area1, np.int32), ((x4, y4)),False)
14    if results4 >= 0:
15    exiting.add(id)
```

Listing 4.25: Determinación de entrada y salida

4.2.16. Visualización de Resultados

Aunque la mayor parte del procesamiento visual se realiza en el script principal, el tracker proporciona la información necesaria para dibujar las cajas delimitadoras y los identificadores sobre cada persona.

```
1     cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)
2     cv2.circle(frame, (x4, y4), 2, (255, 0, 255), -1)
3     cv2.putText(frame, str(id), (x3, y3), cv2.FONT_HERSHEY_COMPLEX, (0.5), (255, 255,
          255), 1)
```

Listing 4.26: Visualización de resultados

4.3. Descripción del Código del Tracker

4.3.1. Importación de la Biblioteca math

Se importa la biblioteca math que se utiliza para calcular la distancia euclidiana entre los centros de los objetos detectados.

```
1     import math
```

Listing 4.27: Importación de la biblioteca `math`

4.3.2. Definición de la Clase Tracker

La clase Tracker se utiliza para gestionar el seguimiento de los objetos detectados. A continuación, se describen los métodos y atributos de esta clase.

4.3.3. Inicialización del Tracker

El método `__init__` es el constructor de la clase. Inicializa dos atributos: `center_points` y `id_count`.

- `center_points`: Un diccionario que almacena las posiciones centrales de los objetos detectados.
- `id_count`: Un contador que se incrementa cada vez que se detecta un nuevo objeto, proporcionando un identificador único para cada objeto.

```
    class Tracker:
    def __init__(self):
    # Store the center positions of the objects
    self.center_points = {}
    # Keep the count of the IDs
    # each time a new object id detected, the count will increase by one
    self.id_count = 0
```

Listing 4.28: Inicialización del Tracker

4.3.4. Método update

El método update se encarga de actualizar las posiciones de los objetos detectados y asignarles identificadores únicos.

Entrada del Método

- `objects_rect`: Una lista de coordenadas de los rectángulos delimitadores de los objetos detectados.

Salida del Método

- `objects_bbs_ids`: Una lista de rectángulos delimitadores de los objetos junto con sus identificadores únicos.

Cálculo del Centro de los Objetos

Se calcula el punto central de cada objeto detectado.

```
    for rect in objects_rect:
    x, y, w, h = rect
    cx = (x + x + w) // 2
    cy = (y + y + h) // 2
```

Listing 4.29: Cálculo del centro de los objetos

Verificación de Objetos Existentes

Se verifica si el objeto ya ha sido detectado anteriormente comparando la distancia euclidiana entre el centro del objeto actual y los centros de los objetos almacenados.

```
    same_object_detected = False
    for id, pt in self.center_points.items():
        dist = math.hypot(cx - pt[0], cy - pt[1])
        if dist < 40:
            self.center_points[id] = (cx, cy)
            objects_bbs_ids.append([x, y, w, h, id])
```

```
            same_object_detected = True
            break
```

Listing 4.30: Verificación de objetos existentes

Asignación de Identificadores a Nuevos Objetos

Si el objeto no ha sido detectado previamente, se le asigna un nuevo identificador.

```
    if same_object_detected is False:
        self.center_points[self.id_count] = (cx, cy)
        objects_bbs_ids.append([x, y, w, h, self.id_count])
        self.id_count += 1
```

Listing 4.31: Asignación de identificadores a nuevos objetos

Limpieza del Diccionario de Puntos Centrales

Se eliminan los identificadores de los objetos que ya no están presentes en la nueva lista de detecciones.

```
    new_center_points = {}
    for obj_bb_id in objects_bbs_ids:
        _, _, _, _, object_id = obj_bb_id
        center = self.center_points[object_id]
        new_center_points[object_id] = center

        self.center_points = new_center_points.copy()
```

Listing 4.32: Limpieza del diccionario de puntos centrales

Retorno de los Rectángulos y IDs de los Objetos

El método retorna la lista de rectángulos delimitadores de los objetos junto con sus identificadores únicos.

```
    return objects_bbs_ids
```

Listing 4.33: Retorno de los rectángulos y IDs de los objetos

4.4. Interacción entre el Script Principal y el Tracker

La interacción entre el script principal y el tracker es esencial para el correcto funcionamiento del programa. El script principal realiza la captura de vídeo, la detección de personas y la definición de áreas de interés. Luego, pasa las detecciones al tracker, que se encarga de mantener un seguimiento continuo de las personas detectadas.

Esta colaboración permite al programa no solo detectar personas en cada fotograma, sino también rastrear sus movimientos a lo largo del tiempo y determinar sus interacciones con las áreas de interés. Esto es especialmente útil para aplicaciones de seguridad, análisis de comportamiento y monitorización de flujos de personas.

Capítulo 5

Resultados

5.1. Resultados

En este capítulo se presentan los resultados obtenidos de la implementación y ejecución del programa descrito anteriormente. Se realizaron pruebas con dos vídeos pregrabados para evaluar la eficacia del sistema de detección y conteo de personas.

5.1.1. Prueba con Vídeo de una Plaza

El primer vídeo utilizado para las pruebas mostraba a personas paseando por una plaza. Se implementó la programación planteada y se ejecutó el sistema para este vídeo. El objetivo era detectar cuántas personas entraban y salían de la plaza. Durante las pruebas, se utilizaron varios modelos de YOLOv8, desde (el modelo más sencillo y a la vez rápido de YOLOV8) hasta YOLOv8l (el segundo modelo más preciso de la gama). Sin embargo, se optó finalmente por el modelo YOLOv8s, ya que ofrecía el mejor balance entre precisión en la detección y velocidad de procesamiento. Los resultados obtenidos fueron satisfactorios, logrando una detección precisa de las personas en la plaza. Además, el sistema logró un conteo de flujo de personas con una precisión del 100 % en las áreas de interés establecidas. En la Figura 5.1 se muestra un ejemplo de la ejecución del programa realizado.

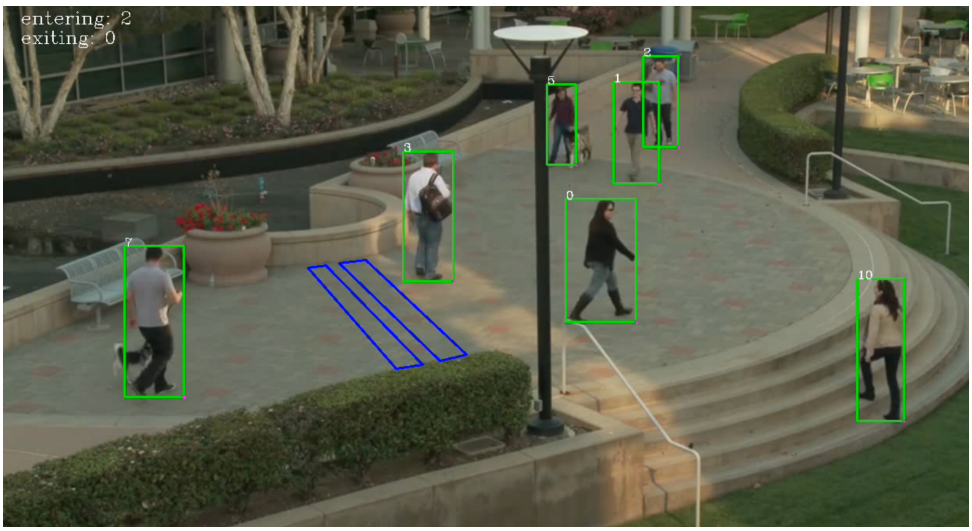

Figura 5.1: Extracto de ejecución del programa para el Vídeo de la Plaza. Fuente: Creación propia

5.1.2. Prueba con Vídeo de una Estación de Tren

En relación al título y objetivo del proyecto, se probó el sistema con un vídeo pregrabado de una estación de tren. Al igual que en la prueba anterior, se utilizaron varios modelos de YOLOv8, optando finalmente por el modelo YOLOv8s debido a su balance óptimo entre precisión y velocidad. Los resultados obtenidos fueron satisfactorios en cuanto a la detección de personas. Sin embargo, debido a la menor calidad del vídeo y la mayor distancia desde la cual se grabó, el conteo de personas no fue tan exacto como en el vídeo de la plaza, no mostrando la robustez apreciada en el caso anterior. Cabe destacar que este trabajo se trata de una prueba de concepto para el análisis del flujo de pasajeros, por lo que tiene una amplia gama de mejora. En la Figura 5.2 se puede ver un ejemplo de la ejecución del programa mencionado anteriormente.

Figura 5.2: Extracto de ejecución del programa para el Vídeo de la Estación de Tren. Fuente: Creación propia

5.1.3. Enlace a los Vídeos

Los vídeos utilizados para las pruebas pueden encontrarse en los siguientes enlaces:

- Vídeo de la Plaza
- Vídeo de la Estación de Tren

Capítulo 6

Conclusiones y líneas futuras

6.1. Resumen y Conclusiones

En este trabajo se ha desarrollado un sistema de detección y seguimiento de personas utilizando modelos YOLOv8. El objetivo principal del proyecto fue implementar un sistema capaz de detectar y contar personas en diferentes entornos mediante el análisis de videos pregrabados. Para lograr esto, se desarrollaron dos componentes principales: el script principal y el módulo de seguimiento.

El script principal fue responsable de la captura de video, la detección de objetos, la definición de áreas de interés y la visualización de los resultados. Se utilizaron varios modelos YOLOv8 para la detección de objetos, incluyendo YOLOv8n, YOLOv8s, YOLOv8m y YOLOv8l. Después de probar cada uno de estos modelos, se determinó que el modelo YOLOv8s ofrecía el mejor equilibrio entre precisión y velocidad, por lo que se eligió este modelo para las pruebas finales.

El módulo de seguimiento, por otro lado, se encargó de rastrear a las personas detectadas a lo largo de los fotogramas del video. Este módulo asignó identificadores únicos a cada persona y rastreó continuamente sus posiciones, lo cual fue esencial para contar cuántas personas entraron y salieron de las áreas de interés definidas.

Se realizaron pruebas con dos videos pregrabados. El primer video mostraba a personas caminando por una plaza. El sistema implementado detectó y contó a las personas con un 100 % de precisión, demostrando su efectividad en entornos controlados con buena calidad de video y distancia de grabación adecuada. El segundo video fue grabado en una estación de tren y presentó un escenario más desafiante debido a la menor calidad del video y a una mayor distancia de grabación. A pesar de estas dificultades, el sistema logró resultados satisfactorios en la detección de personas, aunque el conteo no fue tan preciso como en el video de la plaza.

Los resultados obtenidos demuestran que el sistema desarrollado es efectivo para detectar y rastrear personas en diferentes entornos. El modelo YOLOv8s proporcionó un equilibrio óptimo entre precisión y velocidad, lo que lo hace adecuado para aplicaciones en tiempo real. Sin embargo, se observó que la calidad del video influye significativamente en la precisión del conteo de personas. Se obtienen mejores resultados con videos de alta calidad y distancias de grabación más cortas. Además, el sistema mostró su capacidad para adaptarse a diferentes escenarios, evidenciando su versatilidad y potencial para aplicaciones de seguridad y monitoreo de flujo de personas.

Con base en los resultados obtenidos, se identifican varias líneas de trabajo futuras para mejorar y ampliar el sistema desarrollado. Una de las principales áreas de mejora es la calidad del video utilizado. El uso de videos de mayor calidad podría aumentar la precisión en la detección y el conteo de personas. Además, se podría explorar la optimización de los modelos, ya sea utilizando nuevas versiones de YOLOv8 o ajustando los parámetros de entrenamiento para mejorar aún más los resultados.

Otra línea de trabajo futura es la implementación del sistema en tiempo real. Hasta ahora, las pruebas se han realizado con videos pregrabados, pero un siguiente paso lógico sería adaptar y probar el sistema en tiempo real, permitiendo el monitoreo en vivo del flujo de personas en diferentes entornos. Además, el sistema podría expandirse para detectar y monitorear otros tipos de objetos relevantes dependiendo del contexto, como bicicletas, equipajes o vehículos.

En conclusión, los resultados obtenidos hasta ahora son prometedores y muestran el potencial del sistema desarrollado. Las líneas futuras propuestas buscan mejorar y ampliar las capacidades del sistema, contribuyendo a su aplicación en escenarios más complejos y desafiantes.

6.2. Líneas Futuras

A partir de los resultados obtenidos, se identifican varias líneas futuras de trabajo para mejorar y expandir el sistema desarrollado:

6.2.1. Mejora de la Calidad del Vídeo

Una de las limitaciones encontradas fue la calidad a la hora de encontrar vídeos para la realización de la prueba de la estación de tren. Mejorar la calidad de los vídeos utilizados podría aumentar la precisión en la detección

y conteo de personas, mejorando así el rendimiento del algoritmo.

6.2.2. Optimización del Modelo

Aunque el modelo YOLOv8s ofreció un buen balance entre precisión y velocidad, explorar nuevas versiones del modelo o ajustar los parámetros de entrenamiento podría mejorar aún más los resultados. Además, la placa NVIDIA Jetson Nano dispone de ciertas limitaciones de alimentación que reducen el rendimiento de la programación realizada. Obteniendo el máximo rendimiento de la Jetson Nano se podrían implementar modelos más complejos y precisos para la detección en condiciones adversas.

6.2.3. Implementación en Tiempo Real

El sistema se probó con vídeos pregrabados, pero un siguiente paso lógico sería implementar y probar el sistema en tiempo real, permitiendo una monitorización en directo del flujo de personas en diferentes entornos, y finalmente llegados al punto de que se obtenga un sistema perfectamente funcional, implementearlo en cada una de las paradas del trayecto del tranvía.

6.2.4. Detección de Otros Objetos

Si bien el enfoque principal del proyecto fue la detección de personas, el sistema podría expandirse para detectar y monitorear otros tipos de objetos relevantes según el contexto, como bicicletas, equipaje, vehículos o señales de tráfico.

En resumen, los resultados obtenidos hasta ahora son prometedores y muestran el potencial del sistema desarrollado. Las líneas futuras propuestas buscan mejorar y expandir las capacidades del sistema, contribuyendo a su aplicación en escenarios más complejos y diversos.

6.3. Agradecimientos

Esta investigación resulta del Proyecto Estratégico SCITALA (C064/23), fruto del convenio de colaboración suscrito entre el Instituto Nacional de Ciberseguridad (INCIBE) y la Universidad de La Laguna. Esta iniciativa se realiza en el marco de los fondos del Plan de Recuperación, Transformación y Resiliencia, financiados por la Unión Europea (Next Generation).

Capítulo 7

Summary and Conclusions

In this work, a person detection and tracking system has been developed using YOLOv8 models. The main objective of the project was to implement a system capable of detecting and counting people in different environments by analyzing pre-recorded videos. To achieve this, two main components were developed: the main script and the tracking module.

The main script was responsible for video capture, object detection, defining areas of interest, and visualizing the results. Several YOLOv8 models were used for object detection, including YOLOv8n, YOLOv8s, YOLOv8m, and YOLOv8l. After testing each of these models, it was determined that the YOLOv8s model offered the best balance between accuracy and speed, so this model was chosen for the final tests.

The tracking module, on the other hand, was responsible for tracking the detected people across the video frames. This module assigned unique identifiers to each person and continuously tracked their positions, which was essential for counting how many people entered and exited the defined areas of interest.

Tests were conducted with two pre-recorded videos. The first video showed people walking through a plaza. The implemented system successfully detected and counted people with 100 % accuracy, demonstrating its effectiveness in controlled environments with good video quality and appropriate recording distance. The second video was recorded at a train station and presented a more challenging scenario due to lower video quality and greater recording distance. Despite these difficulties, the system achieved satisfactory results in person detection, although the counting was not as accurate as in the plaza video.

The results obtained demonstrate that the developed system is effective for detecting and tracking people in different environments. The YOLOv8s model provided an optimal balance between accuracy and speed, making it suitable for real-time applications. However, it was observed that video

quality significantly influences the accuracy of people counting. Better results are obtained with high-quality videos and shorter recording distances. Additionally, the system showed its ability to adapt to different scenarios, evidencing its versatility and potential for security and people flow monitoring applications.

Based on the results obtained, several future lines of work are identified to improve and expand the developed system. One of the main areas of improvement is the quality of the video used. Using higher quality videos could increase the accuracy in detecting and counting people. Additionally, model optimization could be explored, either by using new versions of YOLOv8 or by adjusting training parameters to further improve results.

Another future line of work is the implementation of the system in real-time. So far, tests have been conducted with pre-recorded videos, but a logical next step would be to adapt and test the system in real-time, allowing live monitoring of people flow in different environments. Furthermore, the system could be expanded to detect and monitor other types of relevant objects depending on the context, such as bicycles, luggage, or vehicles.

In conclusion, the results obtained so far are promising and show the potential of the developed system. The proposed future lines aim to improve and expand the system’s capabilities, contributing to its application in more complex and challenging scenarios.

Capítulo 8

Presupuesto

8.1. Desglose del presupuesto material

A continuación se desglosa el presupuesto con cada material utilizado y su correspondiente costo:

Item	Costo Unitario (€)
Jetson Nano	89.10
Tarjeta SD de 256 GB	44.99
Cámara Raspberry Pi V2.1	26.99
Carcasa para Jetson Nano	17.99
Cargador para Jetson Nano	13.49
Total	**192.56**

Tabla 8.1: Resumen de los costos del presupuesto material

8.2. Descripción Detallada del presupuesto material

1. **Jetson Nano**: La NVIDIA Jetson Nano fue la pieza central de hardware utilizada en el proyecto, proporcionando el poder de cómputo necesario para realizar tareas de inteligencia artificial y procesamiento de vídeo en tiempo real.

2. **Tarjeta SD de 256 GB**: Esta tarjeta proporcionó el almacenamiento necesario para el sistema operativo, las bibliotecas de software y los datos del proyecto, incluyendo los vídeos utilizados para las pruebas.

3. **Cámara Raspberry Pi V2.1**: Se utilizará esta cámara para capturar el vídeo necesario para las pruebas de detección y seguimiento de personas, debido a su compatibilidad con la Jetson Nano y su alta calidad de imagen.

4. **Carcasa para Jetson Nano**: Para proteger la Jetson Nano durante el desarrollo y las pruebas, se utilizó una carcasa que aseguraba que el hardware no sufriera daños físicos.

5. **Cargador para Jetson Nano**: Un adaptador de corriente fiable fue esencial para asegurar el funcionamiento continuo de la Jetson Nano durante las sesiones de desarrollo y pruebas.

8.3. Desglose del presupuesto del desarrollo

A continuación se desglosa el presupuesto correspondiente al desarrollo de la implementación del proyecto desde investigación o planificación, hasta preparación de presentación, suponiendo un mínimo de costo para la mano de obra de 40€/hora:

Fases	**Horas**	**Costo (€)**
Planificación	10	400
Investigación	50	2.000
Diseño	50	2.000
Desarrollo e implementación	100	4.000
Redacción del documento	50	2.000
Preparación de presentación	20	800
Total	**280**	**11.200**

Tabla 8.2: Resumen de los costos del presupuesto para el desarrollo

8.4. Descripción Detallada del presupuesto para el desarrollo

1. **Planificación**: En esta fase se definieron los objetivos del proyecto, se elaboró el cronograma de trabajo, y se planificaron las tareas a realizar. El costo representa la dedicación del tiempo necesario para planificar el proyecto de manera efectiva.

2. **Investigación**: Esta fase incluye la búsqueda y revisión de literatura científica relevante, el análisis de tecnologías y metodologías existentes, y la identificación de los requisitos del proyecto.

3. **Diseño**: Durante esta fase, se desarrollaron los modelos y arquitecturas necesarios para el sistema de detección y seguimiento de personas.

4. **Desarrollo e implementación**: Esta fase abarca la codificación del sistema utilizando la plataforma Jetson Nano, la implementación efectiva del diseño planteado y los modelos YOLOv8. Incluye pruebas y depuración del código.

5. **Redacción del documento**: Consiste en la elaboración del documento final del trabajo de fin de máster, incluyendo la escritura de capítulos, la preparación de la información, tablas o figuras, y la revisión y edición del contenido.

6. **Preparación de presentación**: Esta fase corresponde con la preparación de las diapositivas y materiales necesarios para la presentación del proyecto. También incluye el tiempo invertido en practicar y perfeccionar la presentación

8.5. Conclusión del costo para el proyecto

El costo total de este proyecto asciende a 11.392,56 €. Este presupuesto refleja los componentes clave necesarios para el desarrollo del sistema de detección y seguimiento de personas utilizando la plataforma Jetson Nano. Todos los componentes fueron seleccionados por su compatibilidad, rendimiento y costo-efectividad, permitiendo la realización exitosa del proyecto dentro de un marco de presupuesto razonable. Además, se desglosa el tiempo aproximado de la duración de cada fase para hacer posible la implementacion de este proyecto.

Bibliografía

[1] Google Blogger. Imagen de sueño. https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgYqMSb4NZxLXZKBmd31M05dD-kCbOuSbYuDfAcq185H7xAw2HbK1rWnz-mFZdtO0rVXTKV-Ly2EXQG-lHIsH2fsEhFwQ_xBtI8-Gi4srJX6Ghag8LQwaPPwnvDwQTUer8/s1600/sue%C3%B1o.jpg, 2024.

[2] El Sol Noticias. Cámaras de monitoreo en trenes. https://elsolnoticias.com.ar/wp-content/uploads/2023/04/Camaras-monitoreo-trenes.jpg, 2024.

[3] El Independiente. Cientos de cámaras e inteligencia artificial para luchar contra el covid en las estaciones de tren. https://www.elindependiente.com/economia/2020/06/04/cientos-de-camaras-e-inteligencia-artificial-para-luchar-contra-el-covid-en-la 2024.

[4] ScienceAsia. Pdf document from scienceasia. https://www.scienceasia.org/2016.42.n1/scias42_28.pdf, 2016.

[5] Francisco M. Gálvez García. Tema 5: Ruido en imágenes. https://www.uco.es/users/ma1fegan/2015-2016/vision/Temas/ruido.pdf, 2016.

[6] ChessBase. Veinte aniversario: Kasparov vs. deep blue. https://es.chessbase.com/post/veinte-aniversario-kasparov-vs-deep-blue, 2024.

[7] Amazon. Yahboom jetson nano 4gb developer kit. https://www.amazon.es/Yahboom-Jetson-Nano-4GB-Developer/dp/B09T37PPRF, 2024.

[8] Amazon. Raspberry pi camera module 8mp. https://www.amazon.es/Raspberry-Pi-Camera-Module-8MP/dp/B01ER2SKFS, 2024.

[9] Viso.ai. Guía de yolov8. https://viso.ai/deep-learning/yolov8-guide/, 2024.

[10] Andrew Schroeder, Kate Mitchell, and John Smith. The evolution of passenger counting technologies in public transport: From manual to automated systems. *Transport Reviews*, 2019.

[11] Wikipedia contributors. Visión artificial – wikipedia, the free encyclopedia, 2024.

[12] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, New Jersey, USA, 4th edition, 2021. Sección sobre la historia de la inteligencia artificial.

[13] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, Berlin, Heidelberg, 1st edition, 2006. Sección sobre los fundamentos del aprendizaje automático.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. Sección sobre los fundamentos y aplicaciones del aprendizaje profundo.

[15] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, London, 2010. Sección sobre análisis de video y detección de objetos.

[16] NVIDIA Corporation. Nvidia jetson nano: Product development. https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/product-development/, 2024.

[17] Raspberry Pi. Cámara v2 para raspberry pi. https://raspberrypi.cl/producto/camara-v2-para-raspberry-pi/, 2024.

[18] YouTube. Tutoriales de yolov8. https://youtube.com/playlist?list=PLZCA39VpuaZZ1cjH4vEIdXIb0dCpZs3Y5&si=2IeXhOQTEXJrcrXI, 2024.

Apéndice A

Apéndice con los códigos realizados

A.1. Algoritmo Principal

```
/*****************************************************************************
*
* script_principal .py
*
*****************************************************************************
*
* AUTORES
*   Jaime Oliva García
*
* FECHA
*   03/07/2024
*
* DESCRIPCION
*   Este script principal se encarga de la captura de vídeo, detección de objetos,
*   definición de áreas de interés y visualización de resultados utilizando la
*   plataforma Jetson Nano y el modelo YOLOv8.
*
*****************************************************************************/

# Import the necessary libraries
import cv2
import pandas as pd
import numpy as np
from ultralytics import YOLO
from tracker import Tracker  # Import the Tracker class from the tracker module

# Load the YOLOv8 model
model = YOLO('yolov8s.pt')

# Define the areas of interest for detecting entry and exit
area1 = [(621, 720), (585, 720), (472, 400), (486, 400)]
area2 = [(696, 720), (656, 720), (498, 400), (512, 400)]

# Function to get RGB coordinates when moving the mouse over the window
def RGB(event, x, y, flags, param):
if event == cv2.EVENT_MOUSEMOVE:
colorsBGR = [x, y]
# print(colorsBGR)  # Uncomment to print BGR coordinates
```

```
# Configure the display window and mouse callback function
cv2.namedWindow('RGB')
cv2.setMouseCallback('RGB', RGB)

# Open the train station video
cap = cv2.VideoCapture('estacion_tren.mp4')

# Read the COCO class file
my_file = open("coco.txt", "r")
data = my_file.read()
class_list = data.split("\n")

count = 0  # Frame counter

# Initialize the Tracker object
tracker = Tracker()

# Dictionaries and sets to track people entering and exiting
people_entering = {}
entering = set()
people_exiting = {}
exiting = set()

# Configuration for the output video creation
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output.mp4', fourcc, 10.0, (1280, 720))

# Main loop to process the video
while True:
ret, frame = cap.read()  # Read a frame
if not ret:
break  # Exit the loop if there are no more frames
count += 1
if count % 2 != 0:
continue  # Process every second frame
frame = cv2.resize(frame, (1280, 720))  # Resize the frame
results = model.predict(frame)  # Make predictions with YOLO
a = results[0].boxes.data
px = pd.DataFrame(a).astype("float")  # Convert results to a DataFrame
list = []
# Draw the areas of interest on the frame
cv2.polylines(frame, [np.array(area1, np.int32)], True, (255, 0, 0), 2)
cv2.polylines(frame, [np.array(area2, np.int32)], True, (255, 0, 0), 2)

for index, row in px.iterrows():
x1 = int(row[0])
y1 = int(row[1])
x2 = int(row[2])
y2 = int(row[3])
d = int(row[5])
c = class_list[d]
if 'person' in c:
list.append([x1, y1, x2, y2])
bbox_id = tracker.update(list)  # Update the tracker with new detections
for bbox in bbox_id:
x3, y3, x4, y4, id = bbox

# Check if a person is entering
```

```
results1 = cv2.pointPolygonTest(np.array(area1, np.int32), ((x4, y4)), False)
if results1 >= 0:
people_entering[id] = (x4, y4)
if id in people_entering:
results2 = cv2.pointPolygonTest(np.array(area2, np.int32), ((x4, y4)), False)
if results2 >= 0:
entering.add(id)

# Check if a person is exiting
results3 = cv2.pointPolygonTest(np.array(area2, np.int32), ((x4, y4)), False)
if results3 >= 0:
people_exiting[id] = (x4, y4)
if id in people_exiting:
results4 = cv2.pointPolygonTest(np.array(area1, np.int32), ((x4, y4)), False)
if results4 >= 0:
exiting.add(id)
# Draw the rectangle and ID of the person on the frame
cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)
cv2.circle(frame, (x4, y4), 2, (255, 0, 255), -1)
cv2.putText(frame, str(id), (x3, y3), cv2.FONT_HERSHEY_COMPLEX, (0.5), (255, 255, 255), 1)

# Display the counts of people entering and exiting on the frame
text_enter = 'entering: ' + str(len(entering))
text_exit = 'exiting: ' + str(len(exiting))
cv2.putText(frame, text_enter, (50, 50), cv2.FONT_HERSHEY_COMPLEX, (0.75), (255, 255, 255), 1)
cv2.putText(frame, text_exit, (50, 75), cv2.FONT_HERSHEY_COMPLEX, (0.75), (255, 255, 255), 1)

out.write(frame)  # Write the frame to the output video
cv2.imshow("RGB", frame)  # Display the frame in the window
if cv2.waitKey(1) & 0xFF == 27:  # Exit the loop if 'ESC' key is pressed
break

# Release the resources
cap.release()
out.release()
cv2.destroyAllWindows()
```

A.2. Algoritmo del Tracker

```
/*****************************************************************************
*
* Fichero .h
*
 *****************************************************************************
*
* AUTORES
*   Jaime Oliva García
*
* FECHA
*   03/07/2024
*
* DESCRIPCION
```

```
* Este código define una clase Tracker que se utiliza para realizar el
* seguimiento de objetos detectados en fotogramas de vídeo. El objetivo
* principal es asignar identificadores únicos a los objetos detectados y
* mantener un registro de sus posiciones a medida que se mueven a través de
* los fotogramas.
*
*******************************************************************************/
# Import the necessary libraries
import math

class Tracker:
def __init__(self):
# Store the center positions of the objects
self.center_points = {}
# Keep the count of the IDs
# Each time a new object is detected, the count will increase by one
self.id_count = 0

def update(self, objects_rect):
# List to store object bounding boxes and IDs
objects_bbs_ids = []

# Get the center point of the new object
for rect in objects_rect:
x, y, w, h = rect
cx = (x + x + w) // 2  # Calculate the center point in the x-axis
cy = (y + y + h) // 2  # Calculate the center point in the y-axis

# Find out if that object was detected already
same_object_detected = False
for id, pt in self.center_points.items():
# Calculate the distance between the current center point and the stored center points
dist = math.hypot(cx - pt[0], cy - pt[1])

# If the distance is less than 40, we consider it the same object
if dist < 40:
# Update the center position of the object
self.center_points[id] = (cx, cy)
# Add the bounding box and ID of the object to the list
objects_bbs_ids.append([x, y, w, h, id])
same_object_detected = True
break

# If it’s a new object, we assign an ID to that object
if not same_object_detected:
self.center_points[self.id_count] = (cx, cy)
objects_bbs_ids.append([x, y, w, h, self.id_count])
self.id_count += 1

# Clean the dictionary of center points to remove IDs not used anymore
new_center_points = {}
for obj_bb_id in objects_bbs_ids:
_, _, _, _, object_id = obj_bb_id
center = self.center_points[object_id]
new_center_points[object_id] = center

# Update dictionary with IDs not used removed
self.center_points = new_center_points.copy()
```

```
# Return the list of bounding boxes and IDs of the objects
return objects_bbs_ids
```

Apéndice B

Mención al artículo enviado a Congreso

B.1. Congreso CISIS24

Como parte del trabajo realizado en este proyecto, se ha enviado a un congreso internacional el artículo titulado “Real-Time Passenger Flow Analysis in Tram Stations Using AI and Computer Vision”. A continuación se presentan los detalles del artículo y del congreso:

- **Título del artículo**: Real-Time Passenger Flow Analysis in Tram Stations Using AI and Computer Vision.
- **Autores**: Sonia Díaz-Santos, Jaime Oliva-García, Pino Caballero-Gil, Cándido Caballero-Gil.
- **Congreso**: 17th International Conference on Computational Intelligence in Security for Information Systems (CISIS).
- **Fecha y lugar**: Salamanca, España. 9-11 de octubre de 2024.

La participación en este congreso permitirá compartir los hallazgos con la comunidad científica y recibir retroalimentación de expertos en el campo de la inteligencia artificial y la visión por computador.