ISSN 1870-4069

Implementación y optimización del modelo YOLO para el reconocimiento de elementos de seguridad en tiempo real

Joctan Maceda Hernández, Yolanda Moyao Martínez, David Eduardo Pinto Avendaño, Beatriz Beltrán Martínez, José Andrés Vázquez Flores

Benemérita Universidad Autónoma de Puebla, México

joctan.maceda@alumno.buap.mx, yolanda.moyao@correo.buap.mx, david.pinto@correo.buap.mx, beatriz.beltran@correo.buap.mx, andres.vazquez@correo.buap.mx

Resumen. Este trabajo presenta un estudio detallado sobre la implementación y optimización de un sistema de reconocimiento de objetos en tiempo real basado en la arquitectura YOLOv10, aplicado a la clasificación de elementos de seguridad. La detección eficiente de entidades como policías, guardias nacionales y civiles armados en entornos urbanos es fundamental para sistemas de vigilancia, control público y toma de decisiones en tiempo real, especialmente en contextos de seguridad ciudadana. Para lograr este objetivo, se desarrolló un conjunto de datos personalizado, mejorado mediante la herramienta Grounding DINO para la generación automatizada de etiquetas, reduciendo significativamente la intervención manual. El modelo fue entrenado usando GPU en Google Colaboratory, aplicando técnicas de optimización que permitieron incrementar tanto su precisión como su velocidad. Se realizó una comparación entre YOLOv10 y versiones anteriores del mismo modelo, destacando las mejoras introducidas en términos de precisión, eficiencia y arquitectura. YOLOv10 se distingue por su enfoque de entrenamiento sin NMS (Non-Maximum Suppression), asignaciones de etiquetas duales y diseño holístico, consolidándose como una herramienta eficaz para tareas de reconocimiento en tiempo real donde la clasificación de elementos de seguridad es crítica.

Palabras clave: YOLO, detección de objetos, visión por computadora, inteligencia artificial, seguridad urbana.

Implementation and Optimization of the YOLO Model for Real-time Security Element Recognition

Abstract. This paper presents a detailed study on the implementation and optimization of a real-time object recognition system based on

the YOLOv10 architecture, applied specifically to the classification of security-related elements. The efficient detection of entities such as police officers, national guards, and armed civilians in urban environments is crucial for surveillance systems, public safety monitoring, and real-time decision-making, particularly in contexts of civil security. To achieve this, a custom dataset was developed and enhanced using the Grounding DINO tool for automated label generation, significantly reducing manual annotation effort. The model was trained using GPU resources in Google Colaboratory, applying optimization techniques that improved both accuracy and speed.8 A comparison was made between YOLOv10 and its previous versions, highlighting improvements in precision, efficiency, and network architecture. YOLOv10 stands out for its training approach without Non-Maximum Suppression (NMS), dual label assignments, and holistic design, establishing itself as an effective tool for real-time object detection tasks where classifying security-related elements is critical.

Keywords: YOLO, object detection, computer vision, artificial intelligence, urban security.

1. Introducción

La detección de objetos en tiempo real ha cobrado relevancia en múltiples áreas de la inteligencia artificial y la visión por computadora. Su implementación es fundamental en sectores como la seguridad, la automatización industrial y el análisis de tráfico. Tradicionalmente, los métodos de detección requerían múltiples pasos de procesamiento, lo que los hacía poco eficientes para aplicaciones en tiempo real. La introducción de YOLO (You Only Look Once) ha revolucionado este campo al permitir inferencias rápidas con una única pasada sobre la imagen.

En este trabajo, exploramos la implementación de YOLOv10 y su optimización para la detección de objetos en entornos urbanos, con un enfoque particular en la identificación de elementos de seguridad pública. Se utilizó Grounding DINO como herramienta para mejorar la calidad del dataset, automatizando parcialmente el proceso de anotación y reduciendo así la intervención manual. Las imágenes fueron inicialmente recolectadas sin etiquetas, y luego fueron procesadas por Grounding DINO para generar anotaciones automáticas que posteriormente fueron validadas manualmente, logrando una base de datos robusta para el entrenamiento del modelo. La Figura 1 muestra una representación esquemática del proceso de detección con YOLOv10. En este esquema se visualizan las etapas clave del modelo, desde la entrada de la imagen hasta la generación de predicciones finales. YOLOv10 emplea un enfoque optimizado que reduce la necesidad de post-procesamiento mediante la eliminación de NMS, lo que mejora tanto la precisión como la velocidad de detección.

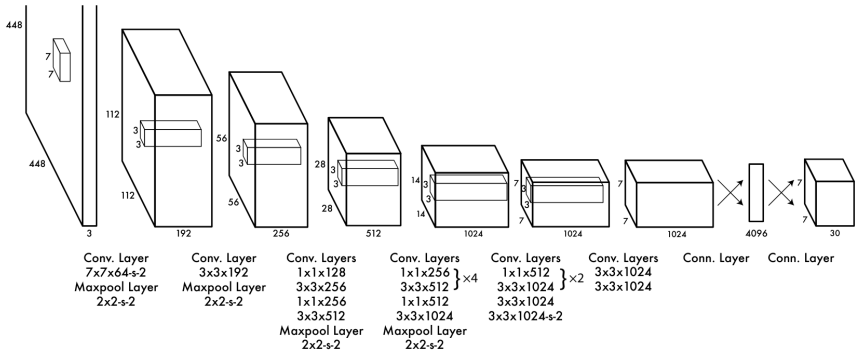

Fig. 1. Representación esquemática del proceso de detección con YOLOv10. Imagen tomada de [1].

2. Estado del arte

Las redes neuronales convolucionales (CNN) han sido fundamentales en la evolu-ción de la visión por computadora, permitiendo la extracción de características espaciales en imágenes mediante filtros y capas de pooling. En el contexto de la detección de objetos, los primeros enfoques se basaban en modelos como R-CNN (Girshick et al., 2014), que dividían el proceso en dos fases: generación de regiones de interés y clasificación de dichas regiones. Aunque estos modelos lograban buenos resultados, su tiempo de procesamiento era demasiado alto para aplicaciones en tiempo real (Ren et al., 2015). La llegada de YOLO en 2016 (Redmon, 2016) supuso una revolución en la detección de objetos, al tratar el problema como una regresión única sobre coordenadas y probabilidades de clase. En lugar de analizar múltiples regiones de una imagen por separado, YOLO divide la imagen en una cuadrícula y predice varias cajas delimitadoras por celda, mejorando drásticamente la velocidad de detección sin comprometer significativamente la precisión.

Esto permitió su adopción en aplicaciones críticas como la videovigilancia y los vehículos autónomos, donde la respuesta en tiempo real es esencial (Redmon and Farhadi, 2018). Desde su introducción, YOLO ha experimentado múltiples mejoras. YOLOv3 optimizó su arquitectura con Darknet-53, incrementando su precisión en detección de objetos pequeños. YOLOv4 introdujo CSPDarknet y técnicas avanzadas de aumento de datos para mejorar el rendimiento en escenarios complejos (Bochkovskiy et al., 2020). La versión YOLOv5, desarrollada por Ultralytics, se centró en la facilidad de uso y eficiencia computacional. La Figura 2 muestra la evolución de YOLO desde su primera versión hasta YOLOv10, destacando las mejoras clave en cada iteración. Esta evolución ha sido clave para lograr una detección más rápida y precisa, abordando los principales desafíos que enfrentan los modelos tradicionales.

Version	Date	Contributions	Structure
version 1	2015	Detector of single-shot objects	Dark web
version 2	2016	Multiscale training, dimensional clustering	Dark web
version 3	2018	SPP Block, Darknet-53	Dark web
version 4	2020	Mish-based activation, CSPDarknet-53 main structure	Dark web
v5	2020	Un anchored detection, SWISH-based activation, PANet	PyTorch
v6	2022	Self-care, detection of objects without anchoring	PyTorch
v7	2022	Transformers, E-ELAN Repaetional	PyTorch
v8	2023	GAN, unanchorted detections	PyTorch
v9	2024	Programmable gradient information (PGI), generalised efficient layer aggregation network (GELAN)	PyTorch
v10	2024	NMS-free training approach, dual-label assignments, holistic model design for greater accuracy and efficiency	PyTorch

Fig. 2. Evolución de YOLO desde YOLOv1 hasta YOLOv10.

YOLOv10 representa la iteración más reciente, incorporando mejoras en la arquitectura de la red neuronal, la gestión de anclas y la optimización de pesos preentrenados, ofreciendo un mejor balance entre velocidad y precisión (Ultralytics, 2024). Sin embargo, uno de los mayores desafíos en la detección de objetos sigue siendo la generación de datasets de entrenamiento bien etiquetados. Para abordar este problema, se ha explorado la integración de Grounding DINO, un modelo que permite la generación automatizada de etiquetas con menor intervención humana, reduciendo el costo y el tiempo de anotación de datos (Liu et al., 2023).

Desde YOLOv5 (Ultralytics), cada versión de YOLO ha sido lanzada con modelos de diferentes tamaños (nano, pequeño, mediano, grande y extragrande), adaptados para distintos niveles de precisión y velocidad de inferencia. YOLOv10 sigue esta tendencia y ofrece una variedad de modelos previamente entrenados, desarrollados por investigadores de la Universidad de Tsinghua. Estos modelos mejoran el rendimiento en términos de latencia y Average Precision (AP) en comparación con versiones anteriores. La Figura 3 muestra una comparación del rendimiento de YOLOv10 con versiones anteriores en términos de latencia y número de parámetros. Se puede observar que YOLOv10 ha sido optimizado para mejorar la precisión mientras reduce el tiempo de inferencia, lo que lo hace ideal para aplicaciones en tiempo real.

Además, los modelos previamente entrenados de YOLOv10 están disponibles en distintos tamaños, lo que permite seleccionar la mejor opción dependiendo de las necesidades del proyecto. La Figura 4 detalla estos modelos junto con sus características principales.

Estas optimizaciones han permitido que YOLOv10 supere a sus predecesores en precisión y eficiencia, consolidándose como una de las mejores opciones para tareas de detección en tiempo real.

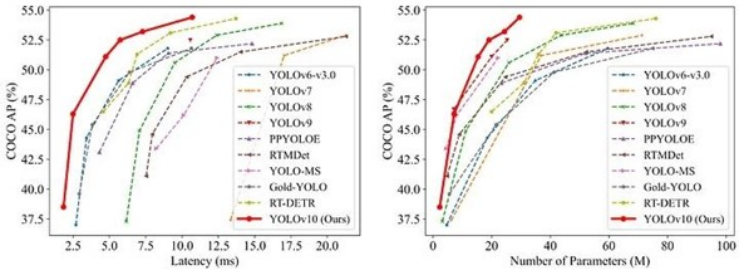

Fig. 3. Comparación de rendimiento: latencia (izquierda) y número de parámetros (derecha) en todos los modelos YOLO.

Tabla 1. Distribución de imágenes por fuente.

Fuente de Imágenes	**Cantidad**
Grabaciones de vigilancia	1,200
Fotografías urbanas	1,000
Material de código abierto	800
Total	**3,000**

3. Metodología

3.1. Selección del dataset

El conjunto de datos utilizado en este estudio fue generado con la plataforma Roboflow, conteniendo un total de 3,000 imágenes etiquetadas manualmente y refinadas con Grounding DINO. Se recopilaron imágenes de entornos urbanos, enfocándose en la detección de policías, guardias nacionales y civiles armados.

En la Tabla 1 se muestra la distribución de las imágenes recopiladas según su fuente.

Para garantizar un entrenamiento balanceado, se dividió el conjunto de datos en subconjuntos de entrenamiento, validación y prueba, como se muestra en la Tabla 2.

Además, el dataset contenía un total de tres clases de objetos como se muestra en la Tabla 3.

Durante el preprocesamiento del dataset, se aplicaron diversas técnicas de normalización para mejorar la calidad de las imágenes antes del entrenamiento. Estas incluyeron:

- **Corrección de color:** Ajuste del balance de blancos y normalización del histograma para mejorar el contraste.
- **Reducción de ruido:** Aplicación de filtros Gaussianos y mediana para eliminar ruido de la imagen.
- **Conversión a escala de grises:** En algunos casos, para mejorar la detección de contornos y texturas clave.

Model	Params (M)	FLOPs (G)	APval (%)	Latency (ms)	Latency (Forward) (ms)
YOLOv6-3.0-N	4.7	11.4	37.0	2.69	**1.76**
Gold-YOLO-N	5.6	12.1	**39.6**	2.92	1.82
YOLOv8-N	3.2	8.7	37.3	6.16	1.77
YOLOv10-N	**2.3**	**6.7**	39.5	**1.84**	1.79
YOLOv6-3.0-S	18.5	45.3	44.3	3.42	2.35
Gold-YOLO-S	21.5	46.0	45.4	3.82	2.73
YOLOv8-S	11.2	28.6	44.9	7.07	**2.33**
YOLOv10-S	**7.2**	**21.6**	**46.8**	**2.49**	2.39
RT-DETR-R18	20.0	60.0	46.5	**4.58**	**4.49**
YOLOv6-3.0-M	34.9	85.8	49.1	5.63	4.56
Gold-YOLO-M	41.3	87.5	49.8	6.38	5.45
YOLOv8-M	25.9	78.9	50.6	9.50	5.09
YOLOv10-M	**15.4**	**59.1**	**51.3**	4.74	4.63
YOLOv6-3.0-L	59.6	150.7	51.8	9.02	7.90
Gold-YOLO-L	75.1	151.7	51.8	10.65	9.78
YOLOv8-L	43.7	165.2	52.9	12.39	8.06
RT-DETR-R50	42.0	136.0	53.1	9.20	9.07
YOLOv10-L	**24.4**	**120.3**	**53.4**	**7.28**	**7.21**
YOLOv8-X	68.2	257.8	53.9	16.86	12.83
RT-DETR-R101	76.0	259.0	54.3	13.71	13.58
YOLOv10-X	**29.5**	**160.4**	**54.4**	**10.70**	**10.60**

Fig. 4. Modelos previamente entrenados disponibles para YOLOv10. Tabla tomada del sitio web de Ultralytics.

Además, se llevaron a cabo procesos de aumentación de datos, incluyendo rotaciones, cambios de iluminación y escalado, con el objetivo de robustecer la capacidad de generalización del modelo, generando un total de 9,000 imágenes después de la aumentación.

El dataset se almacenó en la nube utilizando Roboflow, lo que permitió una gestión eficiente y la posibilidad de realizar modificaciones en tiempo real sin necesidad de descargar archivos localmente. La API de Roboflow fue clave en la

Tabla 2. División del dataset.

Conjunto de Datos	**Cantidad de Imágenes**	**Porcentaje**
Entrenamiento	6,300	70 %
Validación	1,800	20 %
Prueba	900	10 %
Total	**9,000**	**100 %**

Tabla 3. Clases de objetos en el dataset.

Clases de Objetos	**Cantidad de Ejemplos**
Policías	3,000
Guardia Nacional	5,000
Civiles Armados	1,000

automatización del flujo de datos, asegurando que las imágenes estuvieran en el formato requerido para YOLOv10.

3.2. Entrenamiento del modelo

El modelo fue entrenado en Google Colaboratory utilizando una GPU Tesla T4. Los hiperparámetros, fueron configurados como se muestra en la Tabla 4.

Tabla 4. Hiperparámetros utilizados en el entrenamiento de YOLOv10.

Hiperparámetro	**Valor**	**Justificación**
Tasa de aprendizaje	0.01	Ajustada experimentalmente
Batch size	16	Balance entre memoria y estabilidad
Número de épocas	100	Evita sobreajuste
Tamaño de imagen	640 px	Balance entre precisión y velocidad

Estos valores fueron determinados mediante experimentación, utilizando una estrategia de ajuste progresivo de hiperparámetros. Se realizaron pruebas preliminares con distintos valores y se analizó su impacto en las métricas de rendimiento, como la precisión y la velocidad de inferencia.

El entrenamiento se llevó a cabo en 3 etapas:

1. **Pre-entrenamiento:** Ajuste inicial del modelo con un subconjunto del dataset para validar la estabilidad de los hiperparámetros.
2. **Entrenamiento completo:** Optimización con el conjunto completo de imágenes.
3. **Validación:** Evaluación del rendimiento en un subconjunto independiente de imágenes no vistas por el modelo.

Se aplicaron estrategias de ajuste de hiperparámetros, como la reducción progresiva de la tasa de aprendizaje y el uso de técnicas de regularización como Dropout y Batch Normalization para mejorar la estabilidad del modelo.

3.3. Implementación técnica

El sistema de detección se compone de las siguientes etapas:

1. **Captura de imágenes y videos:** Implementación de un pipeline con OpenCV para la captura de imágenes en tiempo real desde fuentes de video en streaming.
2. **Preprocesamiento:** Aplicación de filtros para mejorar la calidad de las imágenes antes de la inferencia, incluyendo eliminación de ruido y ajuste de contraste.
3. **Inferencia con YOLOv10:** Uso de la biblioteca Ultralytics para ejecutar el modelo y obtener predicciones en tiempo real con optimización en GPU.
4. **Post-procesamiento:** Aplicación de algoritmos de supresión de no-máximos (NMS) para eliminar detecciones redundantes y mejorar la precisión de los resultados.
5. **Visualización de resultados:** Generación de gráficos interactivos con Matplotlib y Supervision para evaluar el desempeño del modelo en distintos escenarios.

Para validar la efectividad del modelo, se realizaron pruebas en entornos controlados y en escenarios del mundo real. Se implementó un sistema de evaluación basado en métricas de precisión, recall e IoU, garantizando una medición objetiva del desempeño del sistema de detección.

4. Resultados y análisis

Los modelos entrenados con datasets personalizados fueron comparados con modelos preentrenados. Los resultados, presentados en la Tabla 5, indican que YOLOv10 optimizado con Grounding DINO obtuvo una precisión del 85 %, con una mejora significativa en la detección de objetos específicos en entornos urbanos.

Tabla 5. Métricas de evaluación del modelo YOLOv10.

Métrica	**Valor**
Precisión	85 %
Recall	83 %
F1-score	84 %
IoU (Intersection over Union)	0.78

La Figura 5 muestra la matriz de confusión obtenida con un dataset de pesos preestablecidos sin nuestro entrenamiento, lo que permite visualizar la tasa de aciertos y errores. En contraste, la Figura 6 presenta la matriz de confusión obtenida con un dataset etiquetado con Grounding DINO, evidenciando una mejor distribución de las predicciones correctas. La Figura 7 ilustra la curva de confidencialidad, que representa la relación entre la confianza del modelo y la precisión en la detección. Por otro lado, la Figura 8 muestra la curva de precisión, donde se observa cómo varía la precisión del modelo en función de diferentes umbrales de confianza. Finalmente, la Figura 9 ejemplifica las detecciones realizadas con Grounding DINO y YOLOv10 en un entorno real.

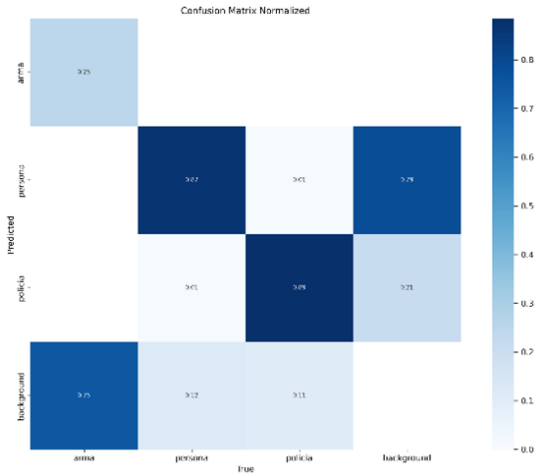

Fig. 5. Matriz de confusión obtenida con un dataset de pesos preestablecidos sin nuestro entrenamiento.

Estos resultados evidencian que el uso de Grounding DINO para el etiquetado automático contribuyó a mejorar la calidad del entrenamiento, lo que se traduce en una mayor precisión y menor tasa de falsos positivos. Las métricas obtenidas demuestran que YOLOv10 optimizado con este enfoque es una solución eficiente para la detección de objetos en tiempo real.

5. Discusión

La comparación de YOLOv10 con sus versiones anteriores revela que:

- YOLOv3: Introdujo Darknet-53, mejorando la detección de objetos pequeños.
- YOLOv4: Incorporó CSPDarknet y aumento de datos avanzado para escenarios complejos.

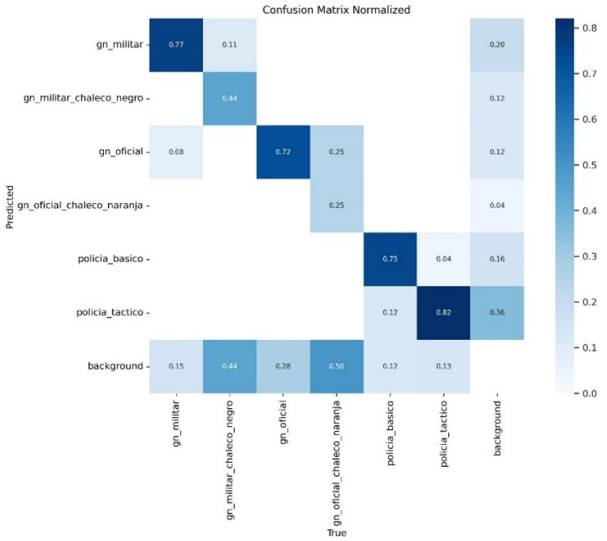

Fig. 6. Matriz de confusión obtenida con un dataset etiquetado con Grounding DINO.

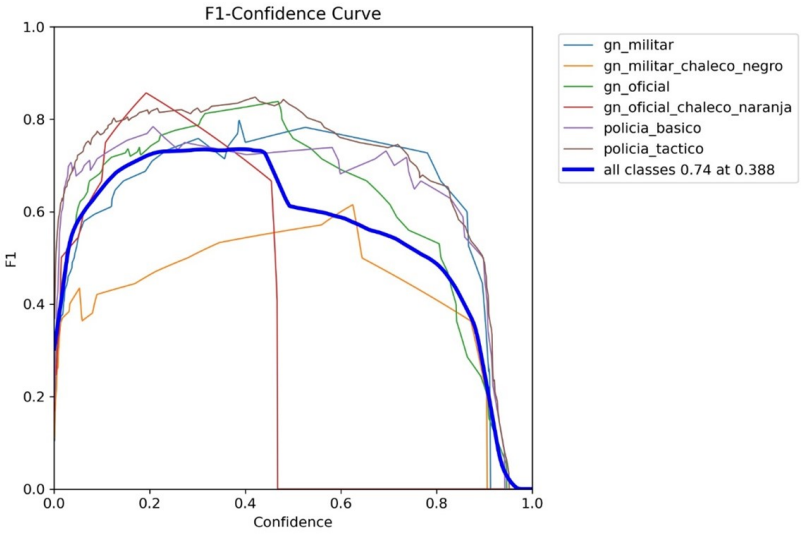

Fig. 7. Gráfica de curva de confidencialidad de la Figura 6.

- YOLOv5: Enfocado en la eficiencia computacional y facilidad de uso.
- YOLOv10: Implementa entrenamiento sin NMS, asignaciones de etiquetas duales y optimización de pesos, logrando una mayor precisión y velocidad.

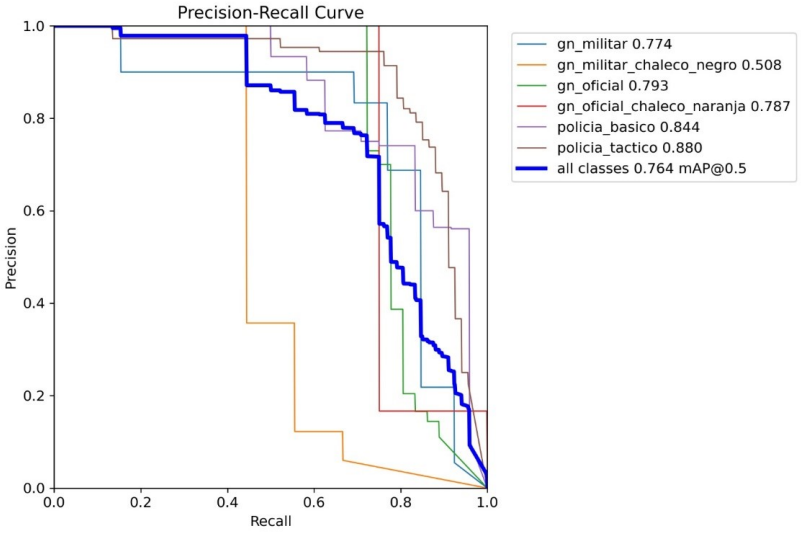

Fig. 8. Gráfica de curva de precisión de la Figura 6.

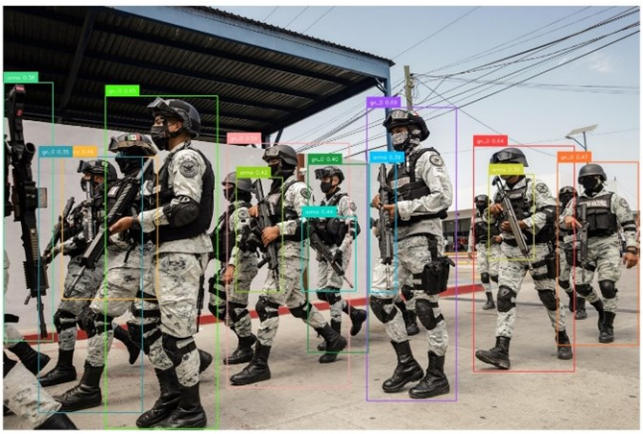

Fig. 9. Ejemplo de detecciones con Grounding DINO y YOLOv10.

La integración de Grounding DINO permitió una mejora en la calidad del dataset, reduciendo falsos positivos y mejorando la anotación de imágenes de entrenamiento. Además, la reducción del tiempo de etiquetado manual aceleró el proceso de entrenamiento y mejoró la capacidad del modelo para adaptarse a entornos urbanos complejos.

6. Conclusión

Este trabajo demostró que la implementación de YOLOv10 para la detección de objetos en tiempo real es altamente eficiente. La combinación con Grounding DINO permitió mejorar la calidad del entrenamiento, reduciendo la intervención manual en la creación del dataset y optimizando el rendimiento en entornos urbanos.

Para trabajos futuros, se propone:

- Optimizar el modelo en entornos no controlados.
- Integrar arquitecturas híbridas que combinen YOLOv10 con modelos basados en transformadores.
- Ampliar el dataset con imágenes de diferentes contextos urbanos para mejorar la generalización.

Referencias

1. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788 (2016)
2. Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y.M.: YOLOv4: Optimal Speed and Accuracy of Object Detection. Preprint at arXiv:2004.10934 [cs.CV] (2020)
3. Liu, S. et al.: Grounding DINO: Marrying DINO with Grounded Pre-training for Open-Set Object Detection. In: Leonardis, A., Ricci, E., Roth, S., Russakovsky, O., Sattler, T., Varol, G. (eds) Computer Vision – ECCV 2024, Lecture Notes in Computer Science, vol 15105. Springer, Cham (2024) https://doi.org/10.1007/978-3-031-72970-6_3
4. Lin, T.Y. et al.: Microsoft COCO: Common Objects in Context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds) Computer Vision – ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham (2014) https://doi.org/10.1007/978-3-319-10602-1_48
5. Ultralytics Team: YOLOv10 Documentation (2024) https://docs.ultralytics.com/models/yolov10/
6. Roboflow Inc.: Roboflow: Build, Train, Deploy Custom Computer Vision Models. (2024) https://roboflow.com/