Universidad Galileo

SenseLink

Detección de Riesgos Urbanos en el Borde para Asistencia Visual

Joaquin Alonso Marroquin Amaya
20004254

Introducción

La discapacidad visual limita la autonomía y seguridad de muchas personas en Guatemala, donde la infraestructura urbana es poco accesible y representa numerosos riesgos. Más de **1.1 millones de guatemaltecos** presentan dificultades visuales, lo que evidencia la necesidad de tecnologías asistivas.

El proyecto **SenseLink** inició utilizando modelos multimodales en la nube, pero esta solución resultó inviable debido a **altas latencias (35–50 s)** y dependencia de Internet, lo cual comprometía la seguridad del usuario. Por ello, el enfoque se rediseñó hacia **computación en el borde**, buscando ejecutar el procesamiento directamente y sin conexión en una **Raspberry Pi Zero 2 W**.

El trabajo actual consiste en entrenar y optimizar un modelo **YOLO11n** para detección en tiempo real de elementos críticos del entorno urbano. El sistema identifica **7 clases esenciales**: semáforos (rojo, verde, amarillo), bache, escaleras, zebra y podotáctil. Con esto, se busca crear un asistente visual **rápido, privado, económico y seguro**, capaz de apoyar la movilidad de personas con discapacidad visual sin depender de servicios en la nube.

Ingeniería de Datos

El desempeño de un modelo de detección en el borde depende directamente de la calidad y equilibrio del dataset. Como no existía un conjunto de datos que cubriera todas las necesidades de navegación para personas con discapacidad visual en Guatemala, se optó por construir uno mediante la agregación y curación de múltiples fuentes.

Origen y Construcción del Dataset

Se consolidaron cuatro datasets de código abierto de Roboflow Universe, cada uno aportando categorías específicas del entorno urbano:

- **Semáforos**: para las clases semaforo_verde, _rojo y _amarillo.
- **Crosswalk Detection**: pasos peatonales (zebra).
- **Stair Detection**: escaleras y desniveles.
- **Smart Blind Stick**: obstáculos a nivel de suelo (bache, podotactil).

Se realizó una homologación de etiquetas para unificar las 7 clases objetivo: **semaforo_verde, semaforo_rojo, semaforo_amarillo, escaleras, zebra, podotactil, bache**.

Análisis Exploratorio y Desbalance

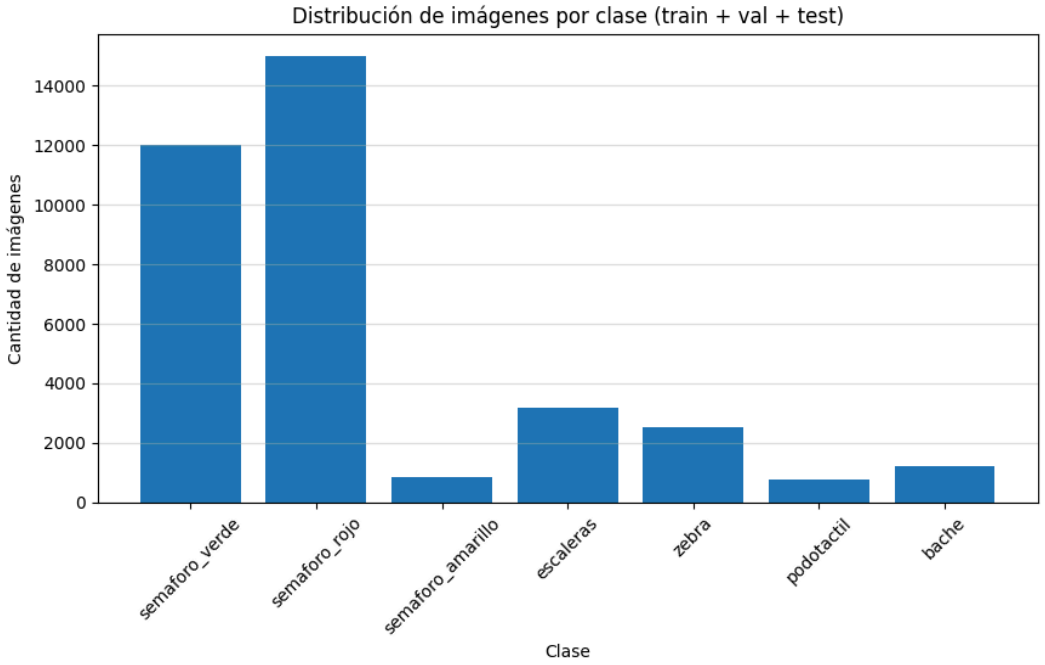
Un análisis previo al entrenamiento evidenció un desbalance significativo entre clases. Las imágenes de semáforos eran abundantes, mientras que las de baches y podotáctil eran escasas. La distribución inicial fue:

Distribución Inicial del Dataset (Sin procesar)

ID	Clase	Cantidad de Instancias	% del Total
0	semaforo_verde	12,004	~33.8%
1	semaforo_rojo	14,983	~42.2%
2	semaforo_amarillo	851	**2.4%**
3	escaleras	3,166	8.9%
4	zebra	2,529	7.1%
5	podotactil	770	**2.1%**
6	bache	1,224	**3.5%**
	TOTAL	**35,527**	**100%**

Esto generaba un sesgo crítico: las clases de semáforos representaban el 76% del dataset, mientras que bache y podotactil quedaban subrepresentadas. De haberse entrenado así, el modelo habría tendido a sobreajustarse a los semáforos e ignorar peligros del suelo. Este

análisis justificó el uso de una **estrategia de Data Augmentation Adaptativo**, descrita en la siguiente sección.



Estrategia de Data Augmentation

Para corregir el desbalance identificado, se implementó un algoritmo de **Data Augmentation Selectivo**, distinto de los enfoques tradicionales que aumentan todas las clases por igual. Aquí, los multiplicadores dependen de la prioridad y escasez de cada clase.

Algoritmo de Balanceo

El script de preprocesamiento analiza las etiquetas presentes en cada imagen y genera copias sintéticas según la prioridad definida. Las clases mayoritarias (semaforo_verde, rojo) no reciben aumentación, mientras que las minoritarias se multiplican agresivamente.

La distribución final quedó así:

ID	Clase	Cantidad Final (Imágenes)	Crecimiento Relativo	Prioridad de Aumentación
1	Semáforo Rojo	14,534	-	Nula (0x)
0	Semáforo Verde	10,374	-	Nula (0x)
2	**Semáforo Amarillo**	**8,294**	**+874%**	**Alta (12x)**
6	**Bache**	**7,605**	**+521%**	**Alta (8x)**
4	Zebra (Paso Peatonal)	5,459	+115%	Media (2x)
3	Escaleras	5,100	+61%	Media (1x)
5	Podotáctil	4,352	+465%	Alta (7x)
-	**TOTAL**	**55,718**	**-**	**-**

Transformaciones Aplicadas

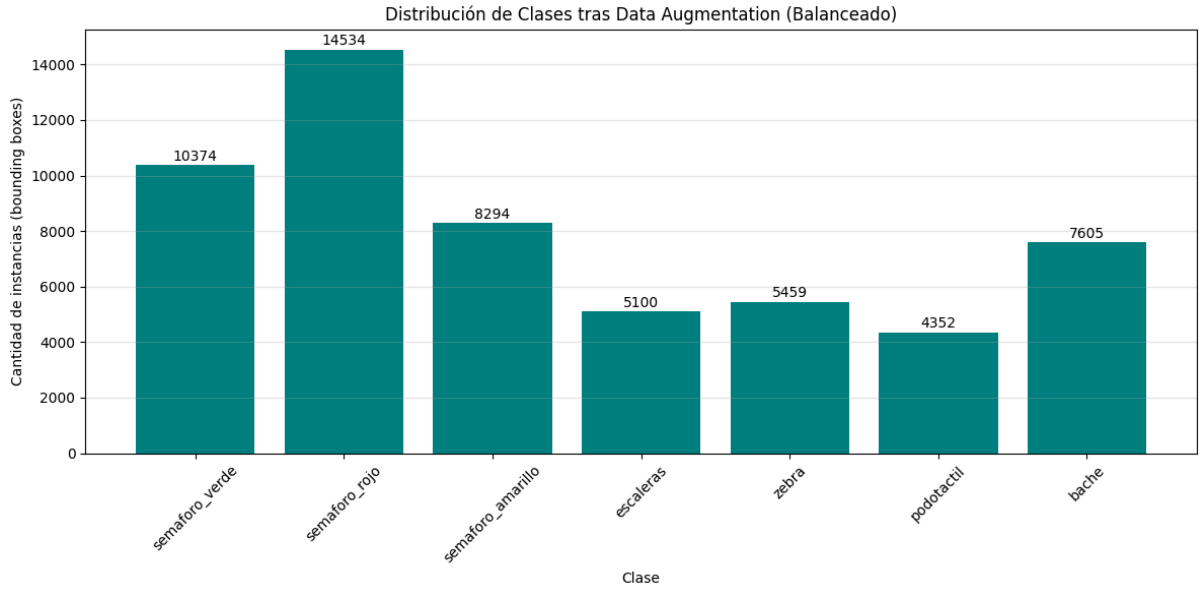
La aumentación se realizó con Albumentations, con un pipeline diseñado para simular condiciones reales de la calle:

- **Clima adverso**: RandomRain y RandomFog (15%).
- **Variaciones de sensor**: GaussNoise y RandomBrightnessContrast, imitando cámaras pequeñas como la de la Raspberry Pi.
- **Dinámica del usuario**: MotionBlur para simular el movimiento al caminar.

Estas transformaciones permiten que el modelo aprenda características más robustas sin alterar la integridad de las bounding boxes.

Dataset Final y Distribución

Tras la aumentación selectiva, el dataset alcanzó **55,661 instancias**, reduciendo la brecha entre clases. Bache pasó de 1,224 a 7,605 instancias y semáforo_amarillo de 851 a 8,294. Aunque no se buscó igualdad perfecta, todas las clases superaron ~4,000 instancias, suficientes para un entrenamiento equilibrado con YOLO.



Splits de Entrenamiento

Para preservar la validez de la evaluación:

- **Train (70%)**: Incluye todas las imágenes aumentadas.
- **Validation (20%)**: Solo datos reales, para ajuste de hiperparámetros.
- **Test (10%)**: Exclusivamente datos reales para la evaluación final.

Metodología y Arquitectura

La elección del modelo estuvo condicionada por las limitaciones de la **Raspberry Pi Zero 2 W**, cuyo CPU ARM Cortex-A53 y 512MB de RAM impiden usar detectores convencionales. Para identificar la arquitectura óptima, se realizó un *benchmark* con modelos preentrenados de la familia **YOLO11** (Ultralytics), evaluando latencia y uso de memoria:

- **YOLO11 (Base):** Preciso, pero demasiado lento y con bloqueos durante la inferencia.
- **YOLO11s (Small):** Más rápido, pero con un consumo de RAM cercano al límite del dispositivo.
- **YOLO11n (Nano):** La opción más ligera (~2.6M parámetros), con el mejor equilibrio entre velocidad y capacidad de detección.

Resultado: Se eligió **YOLO11n**, que libera suficientes recursos para procesos adicionales como TTS y captura de cámara, manteniendo detección confiable en tiempo real.

Entorno de Entrenamiento

El *fine-tuning* se realizó en la nube para aprovechar aceleración por GPU.

Hardware de Entrenamiento:

- **Plataforma:** Google Colab Pro
- **GPU:** NVIDIA Tesla T4 (16GB VRAM)
- **Frameworks:** PyTorch 2.x y Ultralytics YOLOv8/11

Experimentación y Resultados

Para validar la viabilidad del sistema en un dispositivo limitado como la Raspberry Pi Zero 2 W, se diseñó un experimento comparativo A/B para encontrar el equilibrio entre precisión y velocidad de respuesta.

Diseño del Experimento

Se entrenaron dos variantes de YOLO11n bajo las mismas condiciones de dataset y hardware (GPU T4), variando únicamente la resolución (`imgsz`) y el tamaño de lote (`batch_size`):

- **Configuración A (Alta Precisión):**
    - Resolución: 640×640
    - Batch size: 16
    - Objetivo: obtener el máximo mAP posible sin restricciones de hardware.

- **Configuración B (Optimizada para el Borde – RPi):**
    - Resolución: 320×320
    - Batch size: 32
    - Objetivo: maximizar la velocidad de inferencia con una pérdida controlada de precisión.

Ambos modelos se entrenaron por **50 épocas** usando *Early Stopping* (`patience=10`).

Análisis Comparativo

La muestra la evaluación entre el modelo base (A) y la versión optimizada para la raspberry pi (B) usando el conjunto de prueba.

Comparativa de Rendimiento: Modelo A vs. Modelo B

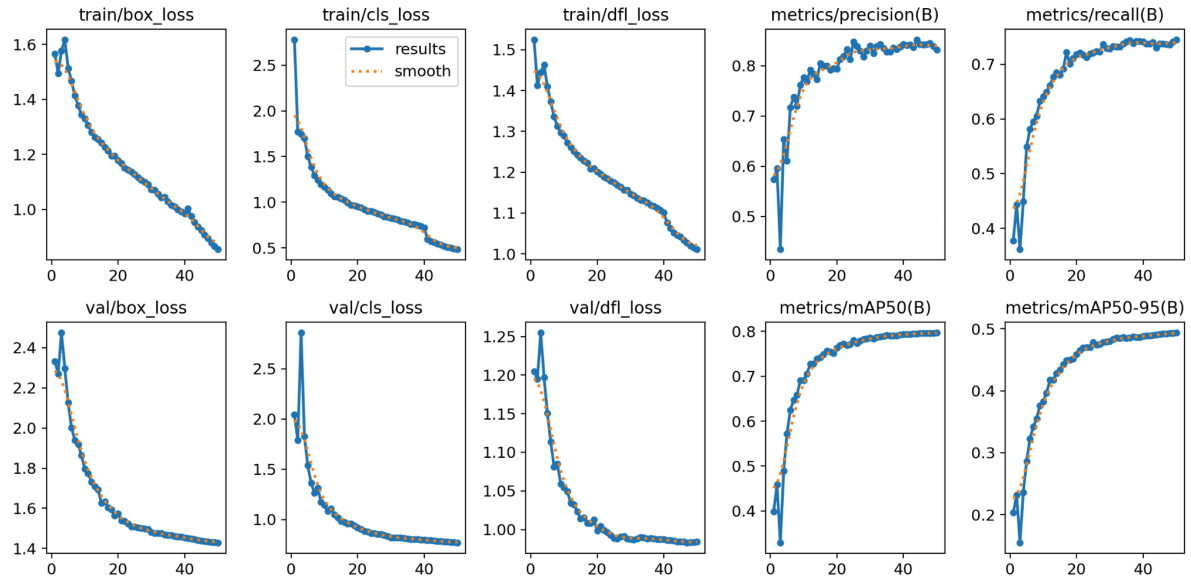
Métrica	Modelo A (640px)	Modelo B (320px)	Dif. Abs.	Impacto
mAP@50-95	0.579	0.494	-0.085	-14.7%
mAP@50	0.883	0.797	-0.086	-9.7%
Precisión	0.870	0.832	-0.038	-4.4%
Recall	0.848	0.745	-0.103	-12.1%
Inferencia (T4)	7.95 ms	7.72 ms	-0.23 ms	Marginal
Carga de entrada	409,600 px	102,400 px	-307,200 px	-75%

Interpretación:
Reducir la resolución de 640px a 320px provocó una caída esperada en sensibilidad, destacando el descenso en Recall. Sin embargo, la Precisión se mantuvo estable, indicando que cuando el modelo B detecta un objeto, lo hace con alta confianza.

Justificación:
La mejora clave no provino de la velocidad en GPU (casi igual), sino de la reducción del 75% en la densidad de píxeles. Esta disminución cuadrática es crítica para que la Raspberry Pi Zero 2 W alcance una velocidad de inferencia útil, convirtiendo al modelo B en la opción más adecuada para despliegues en el borde.

Análisis de Convergencia

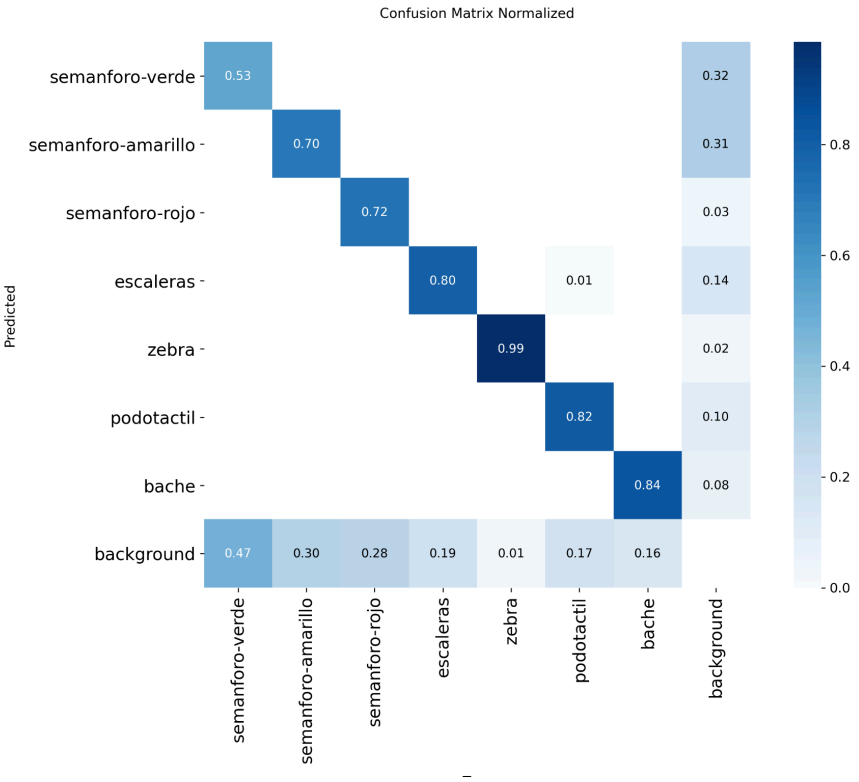
Para asegurar que no existiera sobreajuste, se evaluaron las curvas de aprendizaje del modelo B.



Las pérdidas de caja y clasificación, tanto en entrenamiento como en validación, muestran una convergencia estable sin divergencias en las últimas épocas. Esto confirma que la regularización y el Data Augmentation (ruido, clima sintético) permitieron una correcta generalización a escenarios urbanos reales.

Análisis de Errores (Matriz de Confusión)

La matriz de confusión normalizada revela los errores principales del modelo.



- **Semáforos:** Excelente separación entre verde y rojo. Existe ligera confusión entre amarillo y rojo por saturación lumínica y la baja resolución.

- **Baches:** Continúan siendo la clase más difícil debido al bajo contraste y falta de bordes definidos. Aun así, el balanceo del dataset (aumentación ×8) mejoró significativamente el desempeño, logrando detecciones suficientes para emitir alertas de proximidad.

Conclusiones y Trabajo Futuro

Conclusiones

El desarrollo de **SenseLink-AI** confirma que es posible ejecutar modelos modernos de visión artificial en hardware de borde limitado, evitando la dependencia de la nube en aplicaciones de seguridad.

Conclusiones principales:

1. **Viabilidad del Edge Computing:**
   La arquitectura **YOLO11n** a 320×320px funciona con tiempos de respuesta adecuados en la **Raspberry Pi Zero 2 W**, permitiendo un sistema de alerta completamente *offline* y sin riesgos de privacidad o latencia variable.
2. **Importancia de la Ingeniería de Datos:**
   El **Data Augmentation Selectivo** (Albumentations), con aumentos dinámicos y simulaciones climáticas, corrigió el desequilibrio del dataset y mejoró la detección de clases críticas pero poco frecuentes, como *baches* y *semáforos amarillos*.
3. **Compromiso Precisión-Velocidad:**
   El análisis demostró que, para dispositivos de asistencia, la rapidez de inferencia es más importante que la máxima precisión. Reducir la resolución fue un sacrificio controlado para lograr alertas oportunas.

Trabajo Futuro

Para evolucionar hacia un producto final, se plantean tres líneas principales:

1. **Optimización del Formato de Inferencia:**
   Migrar el modelo `.pt` a **NCNN** o **TensorFlow Lite Micro**, aplicando cuantización INT8 para aumentar significativamente los FPS en la Raspberry Pi.
2. **Aceleración por Hardware:**
   Integrar un coprocesador como **Google Coral USB Accelerator**, permitiendo usar modelos de mayor resolución o liberar la CPU para tareas de audio y navegación.
3. **Expansión del Dataset:**
   Recolectar datos locales de Guatemala para reflejar su infraestructura real (señalización, aceras, condiciones del entorno) y mejorar la generalización del sistema.