Tecnológico Nacional de México

Sistema de detección de placas vehiculares.

Materia:
Tópicos de Inteligencia Artificial

Integrantes:
Dávila Bejarano Víctor Jesús
Flores Medina Martin

Profesor:
Mora Félix Zuriel Dathan

30 de Noviembre del 2025

Video demostración
`https://www.youtube.com/watch?v=ERlRj01u0CM`

Repositorio de GitHub
`https://github.com/MartinMFM/topicosIA`

Contents

1 Introducción

El Reconocimiento Automático de Matrículas (ALPR) se ha consolidado como un componente fundamental dentro de los Sistemas de Transporte Inteligente (ITS) [1]. Su aplicación es crítica para la gestión moderna de entornos urbanos, abarcando desde el cobro automático de peajes hasta el control de acceso y, de manera crucial para este proyecto, la aplicación de leyes de tránsito y el monitoreo de vehículos [2, 3].

En este contexto, el presente proyecto propone el desarrollo de un **Sistema de Detección de Matrículas y Gestión de Incidencias** diseñado para identificar vehículos estacionados en zonas prohibidas. La arquitectura del sistema se fundamenta en un enfoque secuencial validado por su robustez [3, 1]: inicialmente, un primer modelo detecta la matrícula completa; posteriormente, un segundo modelo localiza y recorta exclusivamente el área de los caracteres alfanuméricos. Esta segmentación precisa optimiza la entrada para la librería OCR, garantizando una extracción de datos más limpia y eficiente para su procesamiento en la aplicación móvil.

2 Objetivo General

Desarrollar un sistema integral de detección de matrículas y gestión de incidencias vehiculares mediante técnicas de visión artificial y tecnologías móviles, con el fin de automatizar la identificación de infracciones de estacionamiento y la consulta de datos de propietarios.

3 Objetivos Específicos

- **Desarrollar un modelo de detección de objetos** basado en la arquitectura YOLOv8 para localizar y extraer la región de la matrícula vehicular en imágenes capturadas en escenarios reales.
- **Implementar un modelo de segmentación secundario** entrenado para detectar y recortar exclusivamente el área de los caracteres alfanuméricos dentro de la placa, optimizando así la entrada de datos para el reconocimiento de texto.
- **Integrar un motor de Reconocimiento Óptico de Caracteres (OCR)** capaz de procesar la región segmentada de los caracteres para extraer y digitalizar la cadena de texto de la matrícula con alta precisión.
- **Construir una API RESTful y un sistema de base de datos** que orqueste la comunicación entre los modelos de inteligencia artificial y la interfaz de usuario, permitiendo la validación de matrículas y la recuperación de información del propietario.

- **Diseñar e implementar una aplicación móvil** utilizando el framework React Native, que provea a los usuarios autorizados de un mecanismo de autenticación seguro y una interfaz para la captura de evidencias y reporte de incidencias en tiempo real.

4 Descripción del problema

La gestión del control de acceso vehicular en lugares privados y públicos enfrenta desafíos operativos cuando depende de procesos manuales o sistemas rígidos. Actualmente, la identificación de vehículos que infringen normativas internas como estacionarse en zonas prohibidas, bloquear accesos o utilizar espacios reservados suele carecer de un seguimiento automatizado y eficiente. Esta falta de control dificulta la aplicación de sanciones progresivas, como la restricción del acceso.
Desde una perspectiva técnica, la detección de matrículas en escenarios móviles presenta dificultades significativas. A diferencia de los sistemas de peaje controlados, una aplicación móvil opera en entornos no restringidos, donde factores como la iluminación variable, ángulos de captura oblicuos y fondos complejos degradan el rendimiento de los algoritmos tradicionales [2]. Por tanto, el problema central radica en desarrollar un sistema capaz de reconocer matrículas con alta precisión en estas condiciones adversas para automatizar la gestión de incidencias y el control de acceso basado en el historial de comportamiento del usuario.

5 Justificación

La implementación de este Sistema de Detección de Matrículas y Gestión de Incidencias es fundamental para modernizar la seguridad y la administración de espacios vehiculares. Su principal valor reside en la automatización del proceso de sanción: al vincular la matrícula detectada con una base de datos de propietarios, el sistema permite aplicar reglas de negocio estrictas, como la denegación automática de acceso tras acumular tres reportes de infracción. Esto fomenta una cultura de respeto a las normativas internas y optimiza la labor del personal de seguridad. El uso de arquitecturas avanzadas como YOLO permite una detección robusta, eliminando la necesidad de pasos complejos de segmentación manual y rectificación de imágenes que ralentizan los sistemas antiguos [1, 2]. La integración de esta inteligencia artificial en una plataforma móvil (React Native) democratiza el acceso a herramientas de monitoreo avanzadas, permitiendo una gestión de tráfico y estacionamiento más ágil, precisa y justa.

6 Alcance

El alcance del proyecto comprende el desarrollo de una aplicación móvil que permita el reporte de incidencias vehiculares mediante la captura fotográfica

de matrículas utilizando la cámara del dispositivo. El desarrollo se limita estrictamente a la funcionalidad de registro, excluyendo la implementación de un sistema de gestión o panel administrativo para las incidencias, por lo que la administración de los datos se realizará directamente sobre la base de datos. Asimismo, debido a la alta demanda computacional que requieren los dos modelos de detección y el módulo de OCR para operar simultáneamente, y considerando la disponibilidad de recursos de infraestructura, la API se desplegará y ejecutará en un entorno local.

7 Desarrollo del Sistema

La implementación del sistema se estructuró en tres módulos principales: el procesamiento de inteligencia artificial, la interfaz móvil y la infraestructura de backend. A continuación, se detalla la metodología empleada para cada componente.

7.0.1 Preparación del Dataset

La conformación del conjunto de datos se realizó en dos etapas diferenciadas para atender los requerimientos de cada modelo. Para la detección general de la matrícula, se utilizó el recurso público *Large-License-Plate-Detection-Dataset* obtenido de Kaggle, el cual posee una estructura de directorios preconfigurada para la arquitectura YOLO (división en entrenamiento, prueba y validación).
Por el contrario, para el modelo de segmentación de caracteres, fue necesario construir un dataset personalizado debido a la especificidad de la tarea. Este proceso implicó el etiquetado manual de la región de interés (ROI) que abarca los caracteres alfanuméricos, utilizando la plataforma de visión artificial Roboflow para la anotación y gestión de las imágenes, tal como se ilustra en la Figura 1.

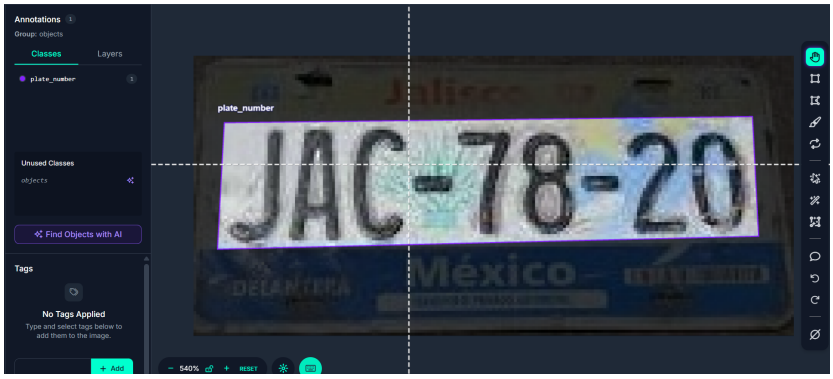

Figure 1: Proceso de etiquetado manual en la plataforma Roboflow para delimitar la región de los caracteres.

7.1 Entrenamiento de los Modelos de Visión (YOLOv8)

Para ambos modelos se seleccionó la arquitectura **YOLOv8s** (versión Small). Esta elección se fundamenta en su arquitectura ligera que permite tiempos de inferencia reducidos en dispositivos con recursos limitados, sin sacrificar significativamente la precisión media (mAP) [3].

7.1.1 Modelo 1: Detección de Placas Vehiculares

El primer modelo se fundamenta en la arquitectura YOLOv8s y tiene como objetivo prioritario aislar la matrícula del entorno vehicular complejo. El entrenamiento se ejecutó durante un ciclo de 50 épocas con un tamaño de lote de 24, utilizando la técnica de Precisión Mixta Automática (AMP) para optimizar el consumo de memoria en la GPU. Para garantizar la convergencia y estabilidad del modelo, se configuró una tasa de aprendizaje inicial de 0.01 con un momentum de 0.937, incorporando un periodo de *warmup* de 3 épocas. Un aspecto distintivo de esta configuración fue la personalización de la función de coste, donde se asignó un peso significativamente mayor a la pérdida de caja (*box loss*: 7.5) frente a la clasificación (*cls loss*: 0.5); esta decisión técnica fuerza al modelo a priorizar la exactitud de las coordenadas del recorte sobre la probabilidad de clase. Finalmente, se empleó un mecanismo de *Early Stopping* con una paciencia de 20 épocas para detener el proceso si la generalización no mejoraba, evitando así el sobreajuste.

7.1.2 Modelo 2: Segmentación de Caracteres (ROI)

El segundo modelo presenta una configuración de entrenamiento diseñada para aislar exclusivamente la región alfanumérica, discriminando eficazmente elementos distractores comunes en las matrículas, tales como logotipos institucionales o nombres de estados. Dada la alta variabilidad en la tipografia y condiciones de los caracteres, se extendio el ciclo de aprendizaje a 100 épocas y se implementó una estrategia agresiva de aumento de datos (*Data Augmentation*). Esta estrategia incluyó la técnica de *Mosaic* con probabilidad de 1.0 para mejorar la detección de objetos pequeños y el contexto semántico, así como transformaciones geométricas (rotación de $\pm 5^{\circ}$, traslación de 0.1 y escalado de 0.2) y ajustes fotométricos en el espacio HSV (Saturación: 0.7, Valor: 0.4) para robustecer la inferencia frente a cambios de iluminación y perspectiva.

7.2 Desarrollo de la Aplicación Móvil

La interfaz de usuario se construyó utilizando **React Native**, permitiendo la captura de imágenes directamente desde la cámara del dispositivo. La aplicación implementa una lógica de pre-validación para asegurar que la imagen enviada tenga la calidad suficiente antes de ser transmitida a la API local.

7.3 Manual de Usuario y Guía de Operación

7.3.1 Acceso y Autenticación

Al iniciar la aplicación, se presenta la interfaz de autenticación. Los usuarios previamente registrados pueden acceder al sistema ingresando sus credenciales en los campos correspondientes, tal como se muestra en la Figura 2.

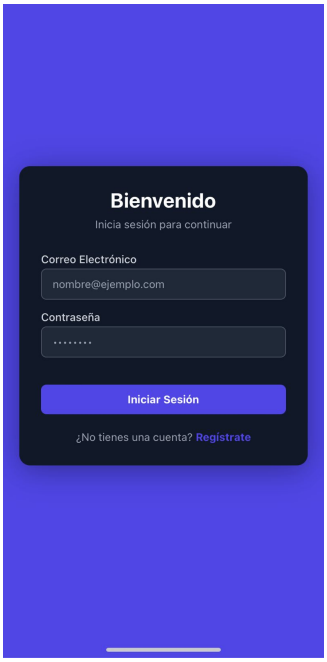

Figure 2: Interfaz de inicio de sesión.

En caso de no contar con una cuenta activa, el usuario debe acceder al módulo de registro (Figura 3). Para completar el proceso, es obligatorio proporcionar un nombre completo, una dirección de correo electrónico válida y definir una contraseña segura.

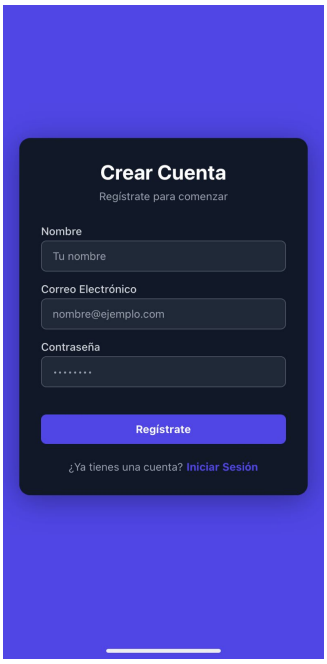

Figure 3: Formulario de registro de nuevos usuarios.

7.3.2 Panel Principal

El menú principal (Figura 4) centraliza las funcionalidades del sistema. Desde aquí, el usuario dispone de dos opciones operativas:

1. **Escanear Placa:** Para detección automática mediante visión artificial.
2. **Reportar Vehículo:** Para el registro manual de incidencias.

Adicionalmente, en la esquina superior derecha se ubica el botón para cerrar la sesión actual de manera segura.

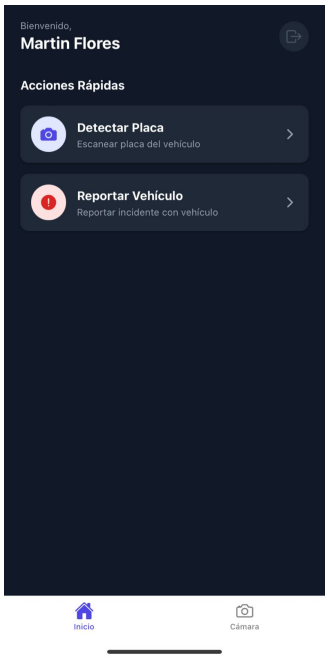

Figure 4: Pantalla principal y menú de opciones.

7.3.3 Módulo de Escaneo y Detección

Al seleccionar la opción de escaneo, el sistema activa la cámara del dispositivo. Es requisito indispensable conceder los permisos de uso de cámara para habilitar la captura. El usuario debe encuadrar la matrícula y presionar el botón de captura para iniciar el procesamiento de imagen, como se ilustra en la Figura 5.

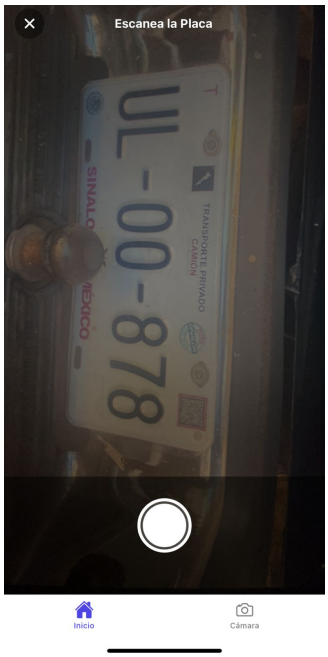

Figure 5: Interfaz de captura y escaneo de matrícula.

Tras el procesamiento de la imagen, el sistema consulta el estado de la matrícula en la base de datos. Si la identificación es exitosa, se desplegará la ficha técnica del propietario y se habilitará la función para generar un reporte, tal como se ilustra en la Figura 6.
Por el contrario, si la matrícula no se encuentra registrada, la aplicación omitirá la información del propietario y deshabilitará automáticamente el botón de reporte para prevenir registros inválidos (Figura 7).

Figure 6: Datos de vehículo registrado.

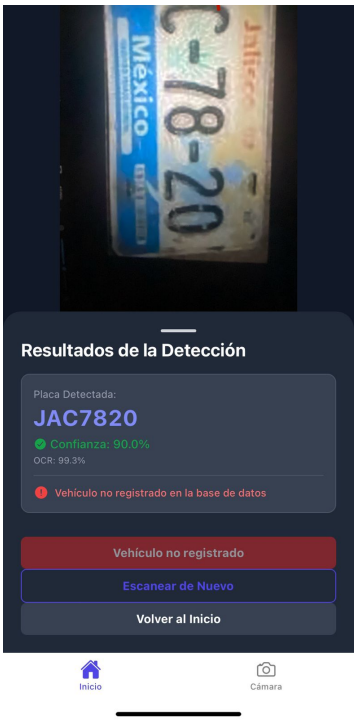

Figure 7: Matrícula no registrada.

7.3.4 Generación de Reportes

La interfaz de reportes precarga automáticamente la matrícula detectada y la geolocalización actual del dispositivo. El usuario únicamente debe ingresar los detalles de la infracción en el campo de descripción (Figura 8).

Figure 8: Formulario para el registro de la incidencia.

Al dar clic en el botón de enviar reporte, el sistema validará el estado del conductor. Se mostrará un mensaje de éxito si el reporte se creó correctamente (Figura 9); por el contrario, si el propietario ya acumula 3 reportes previos, el sistema denegará el acceso automáticamente impidiendo agregar nuevos reportes (Figura 10).

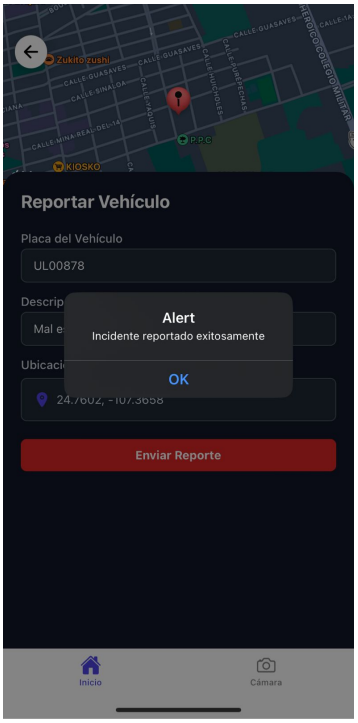

Figure 9: Confirmación de reporte exitoso.

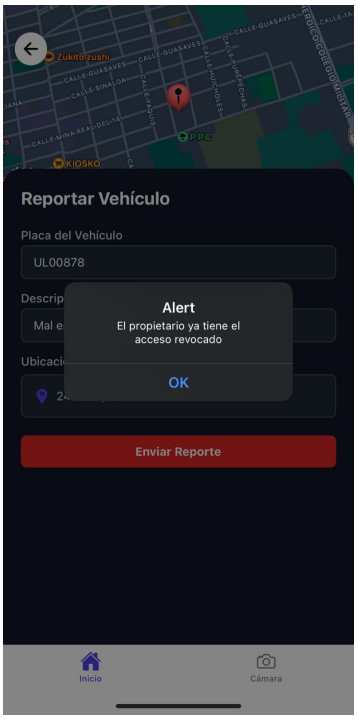

Figure 10: Alerta de restricción ya que el propietario se le revoco el acceso.

7.3.5 Guía de Solución de Problemas

En caso de experimentar dificultades durante la operación del sistema, consulte la Tabla 1 para identificar posibles causas y aplicar las medidas correctivas correspondientes.

Problema	Causa Probable	Solución Recomendada
La aplicación no detecta la matrícula o el escaneo demora demasiado.	Condiciones de iluminación insuficiente, distancia excesiva o ángulo de captura muy oblicuo.	Acérquese al vehículo (distancia óptima: 1 a 2 metros), active el flash del dispositivo o intente capturar la imagen desde un ángulo más frontal (90 grados).
Mensaje "Error de conexión" o "Servidor no disponible".	El dispositivo móvil no está conectado a la red local donde reside la API.	Verifique que el Wi-Fi esté activado y conectado a la misma red institucional que el servidor. Asegúrese de que la IP del servidor sea accesible.
Lectura incorrecta de caracteres (ej. confunde '8' con 'B').	La matrícula presenta suciedad, daños físicos, tornillos interferentes o sombras pronunciadas.	Limpie visualmente la zona de los caracteres. Si el problema persiste, realice una segunda captura variando ligeramente la posición para evitar reflejos.
La cámara muestra una pantalla negra al iniciar.	Permisos de seguridad denegados en el sistema operativo.	Cierre la aplicación, vaya a *Configuración ¿ Aplicaciones ¿ Permisos* y habilite el acceso a la Cámara.

Table 1: Matriz de diagnóstico y solución de errores comunes.

7.4 Documentación Técnica y Arquitectura del Sistema

A continuación, se presenta la documentación técnica del sistema desarrollado, detallando la arquitectura de software, el flujo de datos entre componentes y las especificaciones de infraestructura necesarias para el despliegue de la solución.

7.4.1 Arquitectura del Sistema

El sistema se ha diseñado siguiendo un patrón de arquitectura de tres capas con microservicios, lo que permite desacoplar la lógica de negocio, el procesamiento de imágenes y la gestión de datos. La arquitectura general se compone de los siguientes niveles:

1. **Capa de Presentación (Frontend):** Desarrollada en React Native, gestiona la interacción con el usuario, la captura de imágenes y la visualización de resultados.

2. **Capa de Lógica de Negocio (Backend Principal):** Implementada en Java con Spring Boot, actúa como orquestador central, gestionando la

autenticación, la persistencia de datos y la comunicación entre servicios.

3. **Servicio de Inteligencia Artificial (Microservicio):** Una API ligera en Python (Flask) dedicada exclusivamente al procesamiento pesado de imágenes (YOLO + OCR).

4. **Capa de Datos:** Base de datos relacional MySQL para el almacenamiento estructurado de usuarios, vehículos e incidencias.

7.4.2 Tecnologías Utilizadas

Para la implementación de los distintos módulos del sistema, se seleccionó un stack tecnológico robusto y moderno:

- **Backend:** Java 21, Spring Boot 3.3.5, Spring Data JPA y Hibernate para el mapeo objeto-relacional (ORM).
- **Inteligencia Artificial:** Python 3.9, Flask (API), Ultralytics YOLO (detección), EasyOCR (reconocimiento de texto), OpenCV y NumPy (preprocesamiento).
- **Frontend Móvil:** React Native 0.81, Expo, Zustand (gestión de estado) y Axios para peticiones HTTP.
- **Base de Datos:** MySQL 8.0+.

7.4.3 Implementación del Backend

El backend principal fue desarrollado utilizando Spring Boot, estructurado en controladores, servicios y repositorios. Este componente expone una API RESTful que maneja los siguientes endpoints principales:

- `/api/v1/auth`: Gestión de autenticación y registro de usuarios.
- `/api/v1/plates`: Controlador encargado de recibir la imagen desde el móvil y comunicarla al servicio de Python.
- `/api/v1/incidents`: Gestión del reporte de incidencias, vinculando la matrícula detectada con la geolocalización.

La lógica de detección de placas (`PlateDetectionService`) implementa un flujo de normalización de datos. Una vez que el servicio de Python retorna el texto crudo del OCR, el backend realiza múltiples intentos de búsqueda en la base de datos (búsqueda exacta, normalización de formato mexicano y búsqueda parcial) para asegurar la correcta identificación del vehículo propietario.

7.4.4 Microservicio de Detección (Python API)

El procesamiento de imágenes se delegó a un microservicio independiente en Python para evitar bloquear el hilo principal de la aplicación Java. El flujo de procesamiento en este servicio es el siguiente:

1. **Recepción y Preprocesamiento:** La imagen recibida se somete a mejoras de contraste (CLAHE) y reducción de ruido.
2. **Detección de Placa:** Se utiliza el modelo YOLOv8 entrenado (descrito previamente en la sección 7.1) para recortar la región de la placa.
3. **Extracción de Caracteres:** Se aplica el segundo modelo de segmentación para aislar los números.
4. **OCR y Limpieza:** EasyOCR digitaliza el texto, seguido de una limpieza mediante expresiones regulares para filtrar caracteres no alfanuméricos, ajustándose al formato vehicular estándar.

7.4.5 Base de Datos

El almacenamiento de información se gestiona mediante un esquema relacional en MySQL que garantiza la integridad referencial de los datos. La estructura y las relaciones entre las entidades se ilustran en la Figura 11, conformando un modelo que comprende las siguientes tablas principales:

- **usuarios:** Almacena las credenciales y datos de los usuarios autorizados para reportar incidencias.
- **vehiculos:** Contiene el registro de matrículas y su relación con los propietarios. Se utiliza un índice en el campo `matricula` para optimizar las búsquedas.
- **propietarios:** Almacena la información de contacto (nombre, email, teléfono) necesaria para las notificaciones.
- **incidencias:** Registra el historial de infracciones, incluyendo la geolocalización (latitud/longitud), descripción, fecha y las relaciones con el vehículo y el usuario reportante.

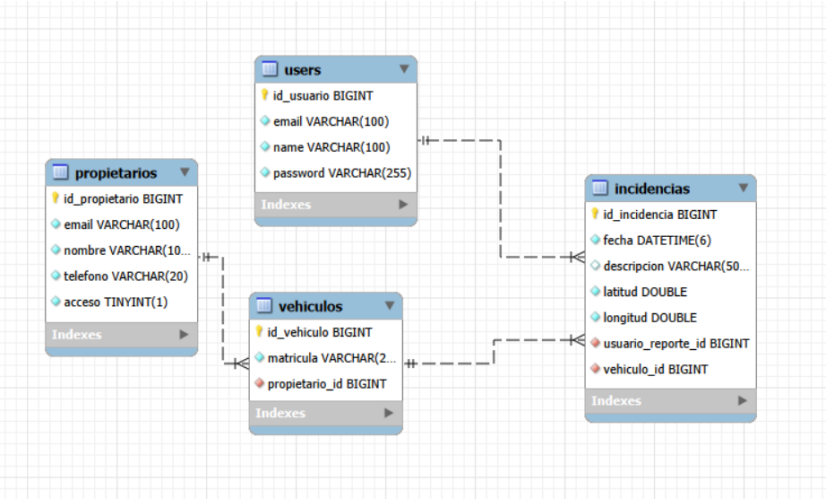

Figure 11: Diagrama Entidad-Relación (DER) del sistema.

7.4.6 Flujo de Operación

El ciclo completo de una detección exitosa, definido en la documentación técnica del proyecto, sigue la siguiente secuencia:

1. El usuario captura la fotografía desde la aplicación móvil.
2. La imagen es enviada al Backend Java, que la redirige al servicio Python.
3. Python procesa la imagen y retorna el texto de la matrícula y el nivel de confianza.
4. Java consulta la base de datos; si el vehículo existe, recupera la información del propietario.
5. La aplicación móvil muestra los datos y permite generar un reporte de incidencia, el cual se guarda con la ubicación GPS actual del dispositivo.

7.5 Instalación y Configuración del Entorno

Para la replicación del entorno de desarrollo y el despliegue del sistema, es necesario cumplir con los siguientes prerrequisitos de software: Java 21 JDK, Maven 3.8+, Python 3.9+, Node.js 18+ y MySQL 8.0. A continuación, se detalla el procedimiento técnico para cada componente:

7.5.1 Configuración de Base de Datos

El sistema requiere una instancia local de MySQL. Aunque Hibernate gestiona la creación de tablas, la base de datos y el usuario deben provisionarse manualmente mediante el siguiente script SQL:

```
CREATE DATABASE license_plate_detection_dev;
CREATE USER 'root'@'localhost' IDENTIFIED BY 'dev_password';
GRANT ALL PRIVILEGES ON license_plate_detection_dev.* TO 'root'@'localhost';
FLUSH PRIVILEGES;
```

7.5.2 Despliegue del Backend (Spring Boot)

El servicio principal actúa como núcleo del sistema. Se debe compilar el proyecto con Maven y asegurar que el archivo `application-dev.properties` contenga las credenciales configuradas previamente.

```
cd matriculas
mvn clean compile
mvn spring-boot:run
```

7.5.3 Activación del Microservicio de IA

Para aislar las dependencias de visión artificial (YOLO, EasyOCR), se recomienda utilizar un entorno virtual de Python. Este servicio se ejecuta en el puerto 10000 para evitar conflictos con el backend principal.

```
cd API
python -m venv venv
# Windows: .\venv\Scripts\activate
pip install -r requirements.txt
python app.py
```

7.5.4 Ejecución del Frontend Móvil

La aplicación cliente utiliza Expo. Es crítico actualizar la configuración de red en `src/api/api.js` con la dirección IP local de la máquina anfitriona para permitir la conexión desde dispositivos físicos.

```
cd frontend-matriculas
npm install
npx expo start
```

8 Resultados de los modelos

La evaluación del sistema se realizó analizando el desempeño individual de cada modelo en el pipeline. Se utilizaron métricas estándar en la detección de objetos: Precisión, Recall (Sensibilidad) y mAP (Mean Average Precision), junto con el análisis de las curvas de pérdida (loss) para verificar la convergencia del entrenamiento.

8.1 Desempeño del Modelo 1: Detección de Placas

El primer modelo, encargado de localizar la matrícula en la imagen completa, demostró un aprendizaje estable durante las 50 épocas de entrenamiento. Como se observa en la Figura 12, las curvas de pérdida de entrenamiento (*train/box_loss*, *train/cls_loss*) muestran un descenso constante, indicando que la red neuronal aprendió efectivamente a generalizar las características visuales de las placas.

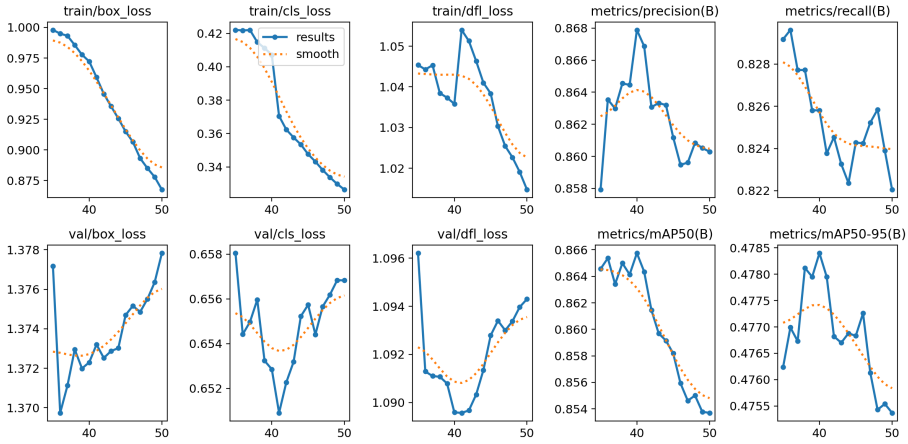

Figure 12: Curvas de aprendizaje y métricas de desempeño del modelo de detección de placas.

Al analizar las métricas de validación, se destaca que el modelo alcanza una precisión media (mAP50) cercana al **0.86**, lo cual indica una alta fiabilidad en la localización del objeto. Sin embargo, se observa en las gráficas de validación (*val/box_loss*) un ligero incremento en las últimas épocas, lo que sugiere un inicio de sobreajuste (overfitting). A pesar de esto, la precisión general se mantiene alta, validando la capacidad del modelo para operar en entornos diversos. La efectividad de este modelo en escenarios reales se evidencia en la Figura 13, donde es capaz de detectar matrículas en condiciones complejas, incluyendo vehículos en movimiento, múltiples autos en una misma escena y variaciones de iluminación.

Figure 13: Resultados visuales de validación. El modelo identifica correctamente múltiples placas en escenarios de tráfico denso y diferentes distancias.

8.2 Desempeño del Modelo 2: Segmentación de Caracteres

A diferencia del detector de placas, el modelo de segmentación de caracteres presenta un comportamiento de aprendizaje más dinámico, tal como se aprecia en la Figura 14. Las curvas de pérdida (*train/box_loss*) muestran un descenso abrupto en las primeras épocas, indicando una rápida asimilación de las características básicas de los caracteres.

Sin embargo, se observa una notable oscilación en las métricas de validación (*val/loss* y *metrics/mAP*). Este fenómeno es atribuible a dos factores técnicos intencionales descritos en la metodología:

1. La aplicación agresiva de *Mosaic Augmentation* y transformaciones geométricas, que dificultan el entrenamiento para forzar al modelo a aprender características más robustas.

2. La alta variabilidad intrínseca de los caracteres (diferentes tipografías y estados de conservación) frente a un tamaño de lote (*batch size*) reducido de 16.

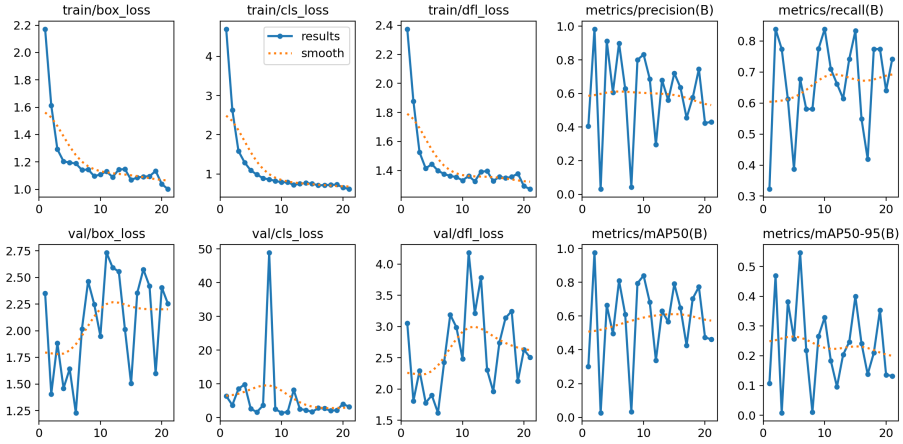

Figure 14: Métricas de entrenamiento del modelo de segmentación. La volatilidad en la validación refleja la complejidad de discriminar caracteres individuales frente a elementos distractores.

A pesar de la volatilidad numérica, la validación cualitativa presentada en la Figura 15 confirma el éxito del objetivo principal del proyecto: la discriminación semántica. El modelo logra ignorar exitosamente elementos de "ruido" como nombres de estados (e.g., "CALIFORNIA", "NORTH CAROLINA"), marcos de concesionarios y logotipos, delimitando exclusivamente la secuencia alfanumérica principal. Esta limpieza previa es el factor determinante que permite al OCR posterior operar con una tasa de error mínima.

Figure 15: Segmentación de la Región de Interés (ROI). Se observa cómo el modelo aísla los números principales ignorando textos secundarios como el estado o el país, optimizando la entrada para el OCR.

8.3 Conclusiones

Este proyecto nos permite llevar un mejor manejo del acceso vehicular al facilitar el crear un reporte utilizando vision artificial para la placa. Se nos presentaron varias dificultades a lo largo del desarrollo, como que no encontramos ningun servicio gratuido para alojar nuestra api debido a su consumo, y en las etapas iniciales al tener solo un modelo para detectar la placa al usar el ocr nos detectaba tambien los caracteres del estado que se encuentra en la placa, por lo que se decision el usar un segundo modelo para detectar el area de los numeros de la placa.

References

[1] Rayson Laroca et al. "A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018.

[2] Heng Sun et al. "License Plate Detection and Recognition Based on the YOLO Detector and CRNN-12". In: *International Conference on Information Science and Cloud Computing (ICSINC 2018)*. Vol. 494. Lecture Notes in Electrical Engineering. Springer, 2019, pp. 66–74. DOI: 10.1007/978-981-13-1733-0_9.

[3] Yongjie Zou et al. "License plate detection and recognition based on YOLOv3 and ILPRNET". In: *Signal, Image and Video Processing* (2021). DOI: 10.1007/s11760-021-01981-8.