Detección de Placas Vehiculares mediante uso de modelos de Deep Learnign y Machine Learning.

Harold David Vergel Sánchez[1], Yermarin Farid Viana Ossa[2], and Brallan Estiven Isaza[3]

[1,2,3]Universidad de Antioquia

Febrero 7, 2025

Abstract

This paper presents an artificial intelligence-based system for vehicle license plate detection and recognition. The system employs YOLOv11 for license plate localization and EasyOCR for character extraction. The YOLOv11 model was fine-tuned using a dataset of 15,000 training images and validated with 745 images, achieving a detection efficiency of approximately 95%. However, EasyOCR, used for character recognition, achieved only 9% effectiveness in identifying complete license plates, though individual characters were often recognizable. Challenges included low-quality images and language orientation limitations in EasyOCR. Despite these issues, corrections during code execution improved performance. Potential improvements to enhance accuracy are discussed.

Este trabajo presenta un sistema basado en inteligencia artificial para la detección y reconocimiento de matrículas vehiculares. El sistema utiliza YOLOv11 para localizar las matrículas y EasyOCR para extraer los caracteres. El modelo YOLOv11 se ajustó mediante un dataset de 15,000 imágenes de entrenamiento y 745 de validación, logrando una eficiencia de detección del 95%. Sin embargo, EasyOCR, empleado para el reconocimiento de caracteres, alcanzó solo un 9% de efectividad en la identificación de matrículas completas, aunque los caracteres individuales fueron reconocibles. Los desafíos incluyeron imágenes de baja calidad y limitaciones en la orientación del idioma en EasyOCR. A pesar de estos problemas, correcciones durante la ejecución del código mejoraron el rendimiento. Se discuten posibles mejoras para optimizar la precisión del sistema.

1 Introducción

El reconocimiento automático de placas vehiculares (Automatic License Plate Recognition, ALPR) es una tecnología ampliamente utilizada en aplicaciones de seguridad, control de tráfico, estacionamientos inteligentes y peajes automatizados. Tradicionalmente, estos sistemas requerían hardware especializado o intervención humana para analizar las imágenes capturadas, lo que limitaba su eficiencia y escalabilidad. Sin embargo, los avances en visión por computador e inteligencia artificial han permitido desarrollar modelos más precisos y rápidos para la detección y reconocimiento de placas a partir de imágenes.

Este proyecto propone el desarrollo de un sistema automatizado de detección y reconocimiento de placas vehiculares, utilizando técnicas avanzadas de visión por computador. Para la detección de la placa en la imagen, se empleó YOLOv11 (You Only Look Once v11), un modelo de aprendizaje profundo altamente eficiente para la detección de objetos. Una vez identificada la ubicación de la placa, se utilizó EasyOCR, una herramienta de reconocimiento óptico de caracteres (OCR) basada en redes neuronales, para extraer el código alfanumérico presente en la placa.

El presente trabajo describe en detalle el desarrollo del sistema, abarcando desde la recolección y preprocesamiento del dataset, hasta el entrenamiento del modelo YOLOv11, la implementación del OCR y la evaluación del rendimiento del sistema. Finalmente, se presentan los resultados obtenidos y se discuten los principales desafíos y posibles mejoras para optimizar la precisión del reconocimiento de placas en aplicaciones del mundo real.

2 Marco Teórico

2.1 EasyOCR

EasyOCR, cuyas siglas en inglés significan *Easy Optical Character Recognition*, es una librería diseñada para facilitar la extracción de texto a partir de imágenes. A diferencia de otras herramientas como Tesseract [1], EasyOCR se destaca por su eficiencia y precisión en el reconocimiento de texto en múltiples idiomas. Esta librería está construida sobre PyTorch [2], un framework de aprendizaje profundo desarrollado por Facebook AI Research (FAIR), conocido por su flexibilidad y capacidad de integración con GPUs para acelerar el procesamiento. Un tutorial detallado sobre su uso puede encontrarse en [3].

EasyOCR funciona mediante la utilización de modelos pre-entrenados en PyTorch, los cuales han sido optimizados para la tarea de reconocimiento óptico de caracteres (OCR). Estos modelos son capaces de detectar y extraer texto de imágenes, además de proporcionar una métrica de confianza para cada detección. La librería permite seleccionar los idiomas específicos que se desean reconocer, descargando automáticamente las redes neuronales correspondientes para su uso. Esto la hace altamente adaptable a diferentes contextos lingüísticos.

Para implementar EasyOCR, es necesario instalar la librería utilizando `pip`, ya que no está integrada directamente en plataformas como Anaconda. El repositorio oficial de EasyOCR en GitHub [4] proporciona instrucciones detalladas para su instalación y configuración. Una vez instalada, se puede configurar para utilizar GPU, lo que acelera significativamente el procesamiento debido a la arquitectura paralela de las GPUs, optimizando las operaciones matemáticas requeridas para el OCR.

EasyOCR está basado en modelos de reconocimiento de texto entrenados principalmente en idiomas de tipo latino. Estos modelos han sido pre-entrenados con conjuntos de datos limitados, lo que les permite funcionar eficientemente en escenarios específicos [5]. Sin embargo, su rendimiento puede verse afectado si se aplica a idiomas o contextos no incluidos en su entrenamiento inicial.

El fine-tuning de EasyOCR implica ajustar los modelos pre-entrenados para adaptarlos a tareas específicas. Esto se logra mediante el reentrenamiento de las redes neuronales con nuevos conjuntos de datos que reflejen las características del problema particular. Este proceso permite mejorar la precisión del modelo en contextos específicos, aunque requiere un conocimiento profundo de PyTorch y acceso a datos etiquetados [5].

2.2 YOLO (You Only Look Once)

YOLO (You Only Look Once) es un algoritmo de detección de objetos conocido por su eficiencia y velocidad. A diferencia de los métodos tradicionales que requieren múltiples pasadas sobre una imagen, YOLO realiza la detección en una sola pasada, lo que lo hace significativamente más rápido. Este marco teórico abarca los conceptos clave, el funcionamiento, los parámetros, los usos y el proceso de fine-tuning de YOLO [6].

En cuanto a los conceptos básicos de YOLO, la detección de objetos implica identificar y localizar múltiples objetos dentro de una imagen. Esto difiere de la clasificación, que solo asigna una categoría a una imagen, y de la localización, que identifica la categoría y la ubicación de un solo objeto. Además, la codificación de bounding boxes se realiza mediante un vector de siete elementos que incluye:

1. **Probabilidad de presencia de objeto**: Valor entre 0 y 1.
2. **Coordenadas (x, y)**: Ubicación del centro del bounding box, normalizadas.
3. **Ancho y alto**: Dimensiones del bounding box, normalizadas.
4. **Clase del objeto**: Identificación de la categoría del objeto.

2.3 Operaciones clave en YOLO

2.3.1 Intersección sobre Unión (IoU)

La IoU mide la superposición entre un bounding box predicho y uno ideal. Se calcula como el área de intersección dividida por el área de unión. Un valor de 1 indica una coincidencia perfecta y un valor de 0 indica que no hubo intersección entre bounding boxes. Así, la IoU es un valor que oscila entre estas dos cantidades [7].

2.3.2 Supresión de No-Máximos (NMS)

NMS elimina los bounding boxes redundantes. Se conserva el bounding box con la mayor probabilidad y se descartan los que tienen una IoU alta con este, reduciendo así la redundancia [7].

2.4 Funcionamiento de YOLO

2.4.1 Entrenamiento

El entrenamiento de YOLO implica:

- Recolectar imágenes con objetos de interés.
- Marcar manualmente los bounding boxes ideales.
- Entrenar una red convolucional para predecir las coordenadas de los bounding boxes [6].

2.4.2 Predicción

Para realizar predicciones:

- Se divide la imagen en una grilla de subimágenes.
- Cada subimagen se procesa en la red convolucional.
- La red genera un volumen de predicciones.
- Se aplica NMS para limpiar los bounding boxes redundantes.
- Se obtiene la bounding box del objeto identificado [7].

Respecto a los parámetros de YOLO, el número de capas varía según la versión, oscilando generalmente entre 24 y 106 capas. En las versiones más recientes, se estima que el número de capas podría estar entre 300 y 350. Además, el fine-tuning es un proceso crucial que implica ajustar un modelo preentrenado con un nuevo conjunto de datos específico, lo que permite adaptar el modelo a nuevas tareas o mejorar su rendimiento en dominios específicos [6].

2.5 Usos de YOLO

2.5.1 Detección de Objetos

El uso principal de YOLO es la detección de objetos en tiempo real. Es ampliamente utilizado en aplicaciones como vigilancia, vehículos autónomos y análisis de imágenes médicas [7].

2.5.2 Tipos de Uso

- **Detección en Tiempo Real**: Ideal para aplicaciones que requieren respuestas rápidas (control de placas vehiculares en un parqueadero atomatizado).
- **Análisis de Vídeo**: Detección de objetos en secuencias de vídeo (fotomultas o cámaras de vigilancia).
- **Aplicaciones Específicas**: Personalización para tareas específicas mediante fine-tuning (Seguridad Aeroportuaria y Control de Equipaje - Detección de objetos prohibidos, etc) [6].

3 Metodología

El presente proyecto se estructuró en dos fases metodológicas principales: detección de placas vehiculares y reconocimiento óptico de caracteres (OCR). En la primera fase, se implementó el modelo YOLOv11 (You Only Look Once), una arquitectura de red neuronal convolucional (CNN) ampliamente reconocida por su eficiencia en tareas de detección de objetos en tiempo real. Este modelo fue entrenado utilizando un conjunto de datos etiquetados, donde cada imagen contenía anotaciones en formato YOLO que especificaban las coordenadas de las placas vehiculares. Para encontrar las imágenes usamos la página Roboflow centrada en desarrollo de aplicaciones de visión computacional y descargamos el dataset proveniente del proyecto Placas_Vehiculares Computer Vision Project [8].

Se decide debido a la gran cantidad de datos que se tenían y con intención de obtimizar el tiempo de computo usar el modelo nano de YOLO versión 11 [9] y ejecutarlo dentro de google colab usando los recursos de

tarjeta gráfica (GPU T4) entregados por este, luego de ejecutar el entrenamiento con el método train de un modelo preentrenado, este entregaba una carpeta con diferente información sobre las predicciones y la tasa de aprendizaje que obtuvo el modelo respecto a las imágenes de validación, pasándose como parámetros del train 10 ciclos (epochs), un tamaño máximo de 640 pixeles y el parámetro de guardado de imágenes como verdadero.

El proceso de entrenamiento se optimizó mediante la métrica IoU (Intersection over Union), la cual cuantifica la superposición entre las regiones detectadas y las regiones reales de las placas, asegurando una alta precisión en la localización [10]. Adicionalmente, se aplicó una técnica de fine-tuning sobre los pesos preentrenados del modelo, lo que permitió adaptarlo específicamente al contexto de las placas vehiculares, mejorando así su rendimiento en la detección de regiones de interés (ROI) [11]. Esta información puede ser observada en los resultados mediante las gráficas que se verán más adelante.

De la carpeta que nos entrega ejecutar en reentrenamiento del modelo de YOLO se extrae best.pt como el mejor modelo de las 10 epochs y se utiliza de ahora en adelante como el identificador de las placas dentro de las imágenes, donde ahora pasamos a integrar este modelo entrenado con el modelo preentrenado de la librería EasyOCR para abordar el problema del reconocimiento de caracteres mediante, una herramienta basada en redes neuronales profundas especializada en la extracción de texto a partir de imágenes. Para garantizar la efectividad del reconocimiento, se aplicaron técnicas avanzadas de preprocesamiento de imágenes, incluyendo la transformación al espacio de color LAB y la aplicación de CLAHE (Contrast Limited Adaptive Histogram Equalization), las cuales mejoraron significativamente el contraste y la claridad de las regiones de texto [12].

EasyOCR fue configurado para basarse en el reconocimiento de caracteres del español y el inglés, y se implementó un proceso de normalización del texto detectado, eliminando espacios innecesarios y convirtiendo todos los caracteres a mayúsculas, lo que facilitó la comparación con las placas de test que en este caso se usaron 100 de estas debidamente etiquetadas de forma manual. Finalmente, se evaluó la precisión del sistema mediante un análisis comparativo entre las placas detectadas y las placas reales, calculando un porcentaje de efectividad.

4 Resultados y Discusión

Analizando los resultados de cada una de las partes trabajadas en el proyecto tenemos:

4.1 Fine-Tuning del molelo YOLO nano versión 11

Al realizar el entrenamiento del modelo con el dataset de 14.958 imágenes ubicado en /Placas_Vehiculares/train/images obtenemos una carpeta que contiene el modelo reentrenado y del cuál podemos analizar en la *Figura 1* las Métricas de entrenamiento y validación del nuevo modelo reentrenado. Cada una de las gráficas ahí representadas corresponde a:

1. **train/box_loss:** Representa la pérdida (error) asociada a la predicción de las coordenadas de los bounding boxes durante el entrenamiento. Un valor más bajo indica que el modelo está prediciendo mejor la ubicación de los objetos.

2. **train/cls_loss:** Corresponde a la pérdida asociada a la clasificación de los objetos durante el entrenamiento. Un valor más bajo indica que el modelo está clasificando correctamente los objetos.

3. **train/dfl_loss:** Es la pérdida relacionada con la distribución de las predicciones de los bounding boxes. Este término es específico de YOLOv8 y se refiere a la precisión en la distribución de las coordenadas.

4. **metrics/precision(B):** Mide la precisión del modelo en la detección de objetos. Un valor más alto indica que el modelo está detectando correctamente los objetos sin muchos falsos positivos.

5. **metrics/recall(B):** Mide la capacidad del modelo para detectar todos los objetos relevantes. Un valor más alto indica que el modelo está encontrando la mayoría de los objetos en la imagen.

6. **val/box_loss:** Es la pérdida asociada a la predicción de las coordenadas de los bounding boxes durante la validación. Un valor más bajo indica un mejor rendimiento en la validación.

7. **val/cls_loss:** Corresponde a la pérdida de clasificación durante la validación. Un valor más bajo indica una mejor clasificación en el conjunto de validación.

8. **val/dfl_loss:** Es la pérdida relacionada con la distribución de las predicciones de los bounding boxes durante la validación.

9. **metrics/mAP50(B):** Es el valor de mAP (mean Average Precision) con un umbral de IoU de 0.50. Mide la precisión promedio del modelo en la detección de objetos.

10. **metrics/mAP50-95(B):** Es el valor de mAP promedio calculado sobre múltiples umbrales de IoU, desde 0.50 hasta 0.95. Proporciona una medida más robusta del rendimiento del modelo.

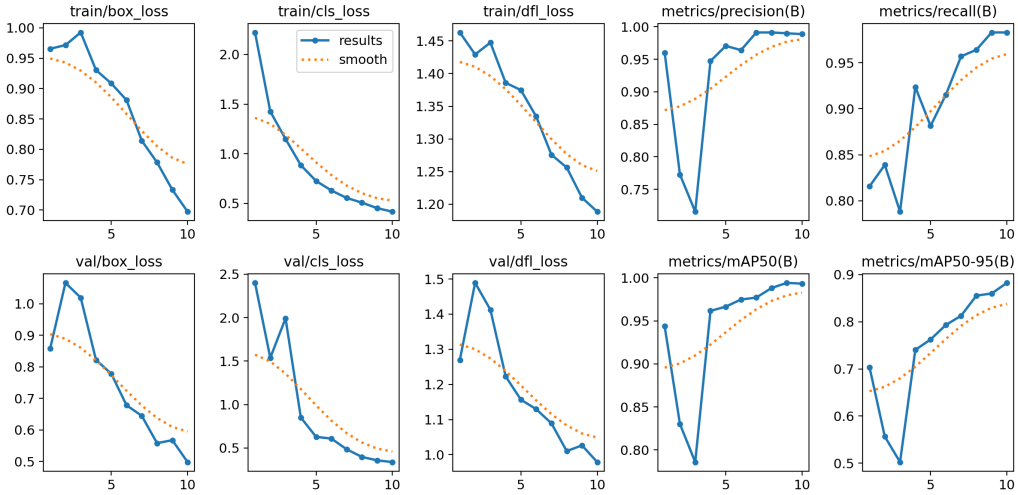

Figure 1: Métricas de entrenamiento y validación del modelo YOLO.

Al analizar como se comporta el modelo de YOLOn11 al ser reentrenado con los datos nuevos vemos que las gráficas se comportan adecuadamente disminuyendo con cada época el valor de box_loss, cls_loss, dlf_loss, donde la cercanía entre los valores finales en ambas gráficas tanto las asociadas en los datos de entrenamiento y validación especialmente en cls y dlf muestran que no existe overfiting en el modelo.

Ahora bien, si nos vamos a la comparativa que realiza el entrenamiento con los datos de validación obtenemos la *Figura 2* que nos muestra la comparación de cuantos errores tuvo el modelo reentrenado con respescto a la 745 imágenes etiquedatas.

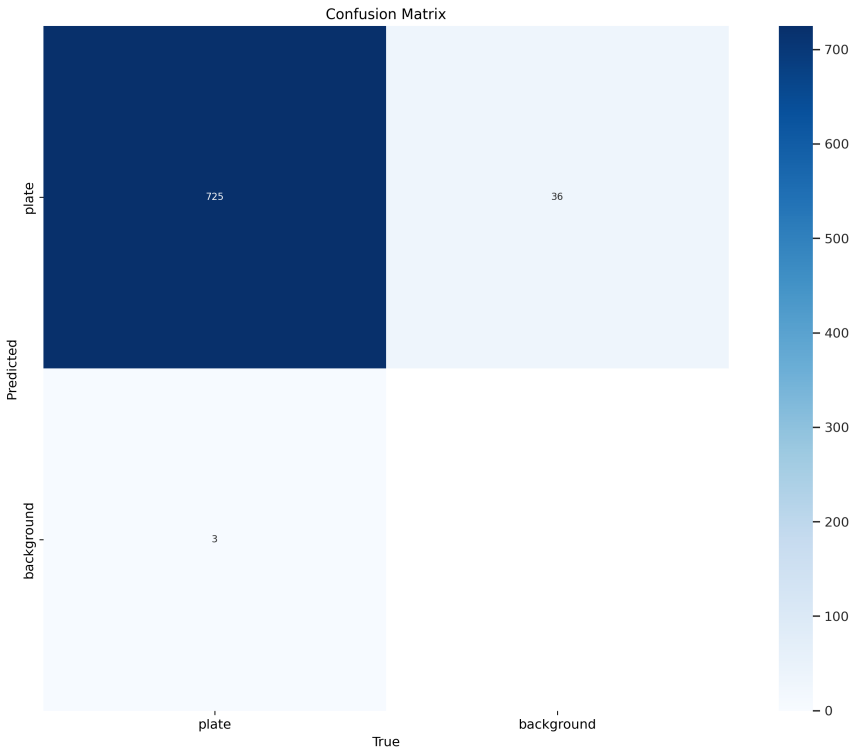

Figure 2: Matriz de Confusión

Vemos a partir de esto que de las 745 imágenes el modelo detectó 764 placas en estos de las cuales 36 fondos fueron identificados como placas (FP) y 3 placas fueron identificadas como fondo (FN), lo que nos da un porcentaje de efectividad del modelo del 95% aproximadamente. La duración del entrenamiento fue de 0.9 horas.

Ahora bien, para las imágenes de ejemplo vamos directamente a la otra parte del modelo relacionada con la detección de caracteres.

4.2 Implementación Librería EasyOCR para detección de caracteres.

La librería EasyOCR permite detectar texto en imágenes por lo que al implementarla junto con el modelo anterior de selección de placas vemos que la mayoría de las placas son correctamente detectadas, no todos los caracteres son identificados o algunas partes de la placas como los bordes o remaches son identificados deformando el caracter que ahí se encontraba o agregando uno inexistente, tal como se aprecia en *Figura 3*

(a) Algunos caracteres no identificados.

(b) Añadido caracter falso.

Figure 3: Imágenes con más o menos caracteres de los reales.

O por otro lado, se encontró que casi toda la placa era correctamente identificada salvo un caracter que podía ser confundido entre letra o número, tal como se muestra en la *Figura 4*

(a) Número entendido como letra.

(b) Añadido caracter falso.

Figure 4: Imágenes con caracteres confundiendo números y letras.

Sin embargo, el modelo detectó algunas placas de forma completa, con un porcentaje de efectividad del 9%. Algunas de estas se pueden ver en *Figure 5*.

(a) Placa nacional reconocida.

(b) Placa internacional reconocida.

Figure 5: Imágenes con placas reconocidas totalmente.

Se intentó reentrenar el modelo de EasyOCR para que no funcionara en base a un idioma sino que se trabajara la estructura alfanumérica de las placas como idioma no latín, se etiquetaron algunas imágenes, sin embargo, no pudo realizarse debido a que requería un conocimiento avanzado de Pytorch.

5 Conclusiones

1. El reentrenamiento del modelo de YOLO alcanzó una gran efectividad, sin embargo, algunas diferencias entre los valores de box_loss para entrenamiento y validación podrían ser mejoradas implementando un modelo más fuerte como YOLO11s que pasa los parámetros de 2.6 a 9.4, es decir, vuelve más complejo el modelo. [10]

2. El porcentaje de efectividad de la integración con EasyOCR reflejó un problema importante en el reconocimiento de caracteres, debido a que no se hizo separación estos y resltó en un sezgo idiomático. Se podría hacer una separación de caracteres usando YOLO pero en su implementación de Segmentación y posteriormente implementar a cada uno de los caracteres separados una detección mediante EasyOCR.

6 Referencias

References

[1] Smith, R. (2007). An Overview of the Tesseract OCR Engine. *Ninth International Conference on Document Analysis and Recognition (ICDAR)*.

[2] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32.

[3] EasyOCR Tutorial. (2020). *YouTube*. Recuperado de `https://www.youtube.com/watch?v=C99ZNbodPb8`

[4] JaidedAI. (2023). *EasyOCR: Ready-to-use OCR with 80+ supported languages and all popular writing scripts*. GitHub. Recuperado de `https://github.com/JaidedAI/EasyOCR`

[5] Jaderberg, M., Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition. *arXiv preprint arXiv:1406.2227*.

[6] YOLO (detección de objetos) ¡EXPLICADO!. (2024). *YouTube*. Recuperado de `https://www.youtube.com/watch?v=ntoRvLgejUY`

[7] Ultralystic YOLOv11. *Ultralystic YOLO Vision*. GitHub. Recuperado de `https://github.com/ultralytics/ultralytics?tab=readme-ov-file`

[8] Yeison, *Placas_Vehiculares Dataset*, Open Source Dataset, Roboflow Universe, Roboflow, 2024. Disponible en: `https://universe.roboflow.com/yeison-89e3n/placas_vehiculares`.

[9] A. Wong, M. Famuori, M. J. Shafiee, F. Li, B. Chwyl, J. Chung, *YOLO Nano: a Highly Compact You Only Look Once Convolutional Neural Network for Object Detection*, Submitted on 3 Oct 2019.

[10] Ultralytics. (2024). *YOLO11 Documentation*. Recuperado de https://docs.ultralytics.com/models/yolo11/

[11] Ultralytics. (2024). *Ultralytics YOLO11 GitHub Repository*. Recuperado de https://github.com/ultralytics/ultralytics

[12] LearnOpenCV. (2022). *Intersection Over Union (IoU) in Object Detection and Segmentation*. Recuperado de https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/