
Detección de placas

Santiago Orjuela¹

Darwin Escobar²

¹ Instituto de Fisica, Facultad de Ciencias Exactas y Naturales, Universidad de Antioquia
Cl.67#53-108 Medellín, Colombia

E-mail: santiagoa.orjuela@udea.edu.co

E-mail: darwin.escobar@udea.edu.co

Key words. Machine Learning, Redes Neuronales, Redes Neuronales Convolucionales, Yolov5, Github

INTRODUCCIÓN

La detección de placas vehiculares es una tecnología ampliamente desarrollada en la actualidad, ya que se utiliza en diversos entornos, como centros comerciales, peajes y sistemas de control de acceso. Debido a su importancia, existen múltiples enfoques para abordar este problema.

En este informe, se presenta el desarrollo de un modelo de detección de placas de automóviles utilizando YOLOv5 (You Only Look Once version 5), un modelo de detección de objetos en tiempo real basado en redes neuronales convolucionales. Además, se empleó Roboflow para la clasificación automática de imágenes, organizándolas en conjuntos de entrenamiento, validación y prueba. Finalmente, se implementó una red neuronal profunda encargada de reconocer los caracteres de las matrículas detectadas.

1. Metodología

El problema general de la detección de los valores alfanuméricos en las placas se dividió en tres subproblemas, los cuales desarrollaremos a medida que los presentamos, en términos generales lo podemos ver en la imagen 1.

1.1. Detección de la Placa (Subproblema 1):

Para este problema, se nos proporcionó un dataset de 198 imágenes de automóviles en los cuales se puede

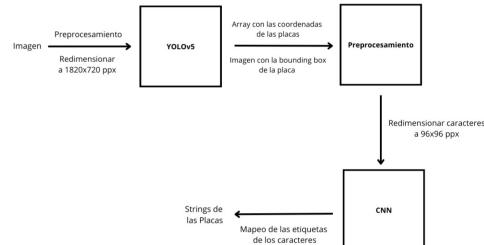


Fig. 1: Procedimiento general

observar al menos una placa. Sin embargo, algunas imágenes contienen una o más placas en el fondo, lo que hace que su detección sea más compleja. Por esta razón, nos enfocaremos únicamente en la detección de la placa principal, es decir, aquella que se ve claramente en la parte frontal del vehículo, como se muestra en la

Para siquiera comenzar a considerar el uso de un modelo, sabemos que primero debemos entrenarlo. Para ello, es necesario proporcionarle ejemplos etiquetados, lo que implica extraer las cuatro coordenadas que delimitan las placas en las diferentes imágenes. En el ámbito de la detección de objetos, esta delimitación se conoce como bounding box. La bounding box puede generarse manualmente utilizando herramientas como LabelImg o VGG Image Annotator. Sin embargo, optamos por una solución más sencilla mediante Roboflow, una plataforma especializada en visión por computadora. Roboflow nos



Fig. 2: Ejemplo de una de las imágenes del dataset

facilitó el proceso de etiquetado, organización y redimensionamiento de las imágenes. La organización de los datos sigue el formato que se muestra en la 3, y las imágenes fueron redimensionadas a una resolución de 1280x720 px.

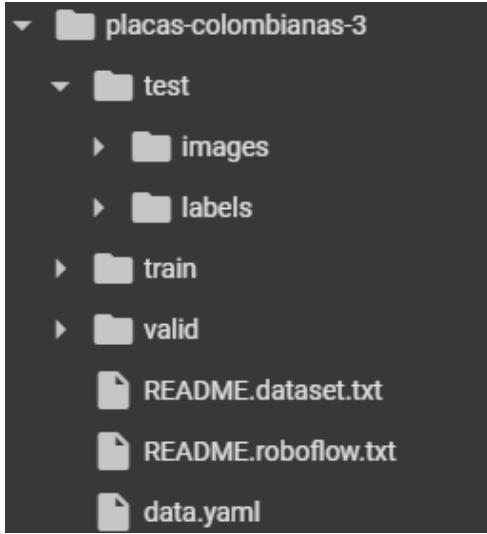


Fig. 3: Formato de YOLO

La organización de las imágenes de esta forma no es aleatoria, sino que es un requisito fundamental para el modelo que hemos decidido utilizar. YOLO (You Only Look Once) es un modelo de detección de objetos en tiempo real basado en redes neuronales convolucionales. Se le denomina "You Only Look Once" porque, a diferencia de otros métodos tradicionales, analiza toda la imagen en una sola pasada, en lugar de procesarla en múltiples etapas. Este modelo es el que utilizaremos para la detección de placas, y para su correcto funcionamiento, es imprescindible que las imágenes estén organizadas en el formato previamente mencionado. Por esta razón, como se ha señalado anteriormente, la forma en la que se estructuraron los datos no es aleatoria, sino una necesidad del sistema.

El modelo YOLOv5 fue extraído del repositorio oficial de GitHub (<https://github.com/ultralytics/yolov5>). En específico, utilizamos el modelo `yolov5n6.pt`, seleccionado debido al tamaño de las imágenes mencionadas anteriormente. Con todos estos elementos preparados, procedimos a entrenar el modelo en Google Colab, utilizando la GPU T4 gratuita proporcionada por la plataforma. Tras el proceso de entrenamiento, obtuvimos los resultados que se muestran en la imagen 4, y el resumen general del modelo en la imagen 5. En términos generales, el modelo presenta las siguientes características:

- 206 capas en su arquitectura.
- Más de 3 millones de parámetros entrenables.
- 4.2 mil millones de operaciones de punto flotante por segundo (4.2 GFLOPs).
- Precisión del 84%, lo que indica que la mayoría de las detecciones son correctas.
- Recall del 93.5%, lo que significa que el modelo detecta correctamente el 93.5% de los objetos presentes en las imágenes.
- $mAP@50 = 0.954$, lo que indica un alto rendimiento en la detección con un umbral de IoU de 0.5. $mAP@50-95 = 0.799$, reflejando la precisión media en distintos umbrales de IoU.

Finalmente, todos los resultados fueron almacenados en una carpeta denominada "runs", donde se encuentran las imágenes con las placas detectadas, como se muestra en la 6.

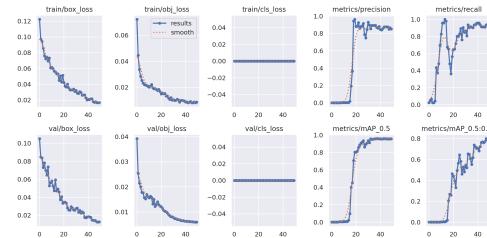


Fig. 4: Resultados del modelo YOLOv5

```
Model summary: 206 layers, 3007756 parameters, 0 gradients, 4.2 GFLOPs
Class Images Instances P R mAP@50 mAP@95 100% 2/2 [00:00:00:00, 2.58it/s]
Results saved to runs/train/exp/
```

Fig. 5: Resumen del modelo

Ya teniendo el modelo entrenado, podemos usar las imágenes de testeо, el modelo no entrega entonces la imagen con las placas que detectó y las bounding boxes, con esto podemos recortar la imagen y tener solo la imagen de la placa como se muestra en 7



Fig. 6: Ejemplos de placas detectadas



Fig. 7: Imagen de una placa recortada a partir de YOLOv5

1.2. Segmentación de Caracteres (Subproblema 2):

Una vez obtenida la imagen de la placa vehicular aislada, el siguiente desafío consiste en segmentar cada carácter de manera individual, es decir, extraer regiones de la imagen que contengan exclusivamente una letra o un número. Para ello, es posible aplicar diversas técnicas de procesamiento digital de imágenes, como la binarización mediante el método de Otsu y el análisis de proyecciones horizontales y verticales.

En este trabajo, se desarrolló una función en Python utilizando OpenCV (Open Source Computer Vision Library), denominada segmentar caracteres, cuyo propósito es recibir la imagen de la placa como entrada y devolver una lista de imágenes, cada una correspondiente a un carácter segmentado, como se ilustra en la Figura 8. El procedimiento de segmentación se compone de varias etapas. En primer lugar, se recorta la imagen de la placa, preservando únicamente el 75% superior de su altura, con el fin de eliminar posibles inscripciones adicionales en la parte inferior, como el nombre de la ciudad de emisión. A continuación, la imagen se convierte a escala de grises y se aplica un factor de escalamiento para mejorar la visibilidad de los caracteres. Posteriormente, se reduce el ruido mediante filtrado gaussiano y se suavizan los bordes, transformando los caracteres en regiones blancas sobre un fondo negro. La binarización se realiza empleando el método de Otsu, que permite determinar de manera óptima el umbral de segmentación sin intervención manual. Una vez obtenida la imagen binarizada, se identifican los contornos de los caracteres y se ordenan de izquierda a

derecha en función de su posición en la imagen. Finalmente, se extraen los caracteres individuales, se les añade un margen para mejorar su reconocimiento y se almacenan en una lista para su posterior procesamiento.



Fig. 8: Letras separadas de la placa

1.3. Reconocimiento de Caracteres (Subproblema 3):

Ahora que tenemos las diferentes letras, vamos a introducir estas imágenes en una red neuronal convolucional (CNN), un modelo simple con 9 capas: 4 convolucionales, 1 MaxPooling2D para la reducción de dimensiones y optimización de cálculos, una capa Dropout para mejorar la generalización del modelo evitando la dependencia de neuronas específicas, una capa Flatten para conectar las capas convolucionales con las capas densas y, por último, dos capas densas, alcanzando un total de 1,712,148 parámetros entrenados con un rendimiento mostrado en la imagen 9.

Cabe aclarar que, si bien esta fue la red neuronal seleccionada, también probamos otra arquitectura preentrenada que reentrenamos para nuestro caso específico. En particular, evaluamos MobileNetV2, pero su desempeño fue significativamente inferior al de nuestra red más pequeña.

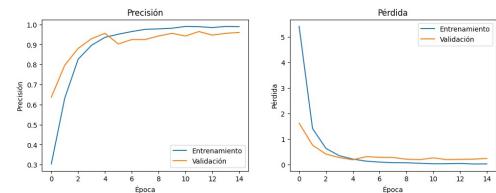


Fig. 9: Características del modelo convolucional

Este subproblema fue el que generó el mayor número de dificultades. Inicialmente, utilizamos bases de datos disponibles en la web con más de 3,000 caracteres, incluyendo los números del 0 al 9 y las letras de la A a la Z. Entrenamos el modelo simple, pero al aplicarlo a nuestras imágenes de caracteres, su rendimiento no fue eficiente. Aunque el modelo parecía desempeñarse bien en los datos de entrenamiento y prueba, fallaba considerablemente con nuestras imágenes.

Por lo tanto, decidimos construir nuestro propio conjunto de datos. Para ello, extrajimos los caracteres de todas las imágenes de placas y los recortamos como se muestra en la imagen 8. Luego, almacenamos

estos caracteres en Google Drive, organizándolos en carpetas según su etiqueta correspondiente. Con este nuevo conjunto de datos, reentrenamos la red neuronal y logramos una mejora significativa en su desempeño.

Aun así, el modelo sigue presentando algunas confusiones, especialmente entre caracteres visualmente similares como 0, O, G y D, así como entre 1 e I, como se observa en la imagen 10. Sin embargo, en otros casos, el modelo logra reconocer los caracteres de manera precisa y sin errores.

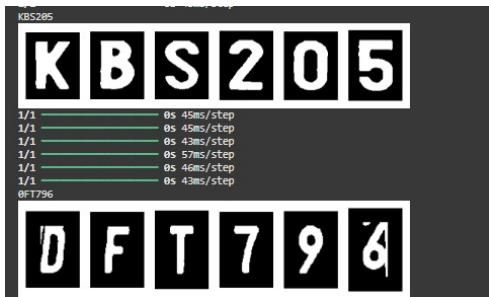


Fig. 10: OCR

2. Conclusiones

- El uso de YOLOv5 resultó ser una estrategia efectiva para la identificación de placas en imágenes de automóviles, alcanzando un alto nivel de precisión y recall en la detección.

- los modelos preentrenados en conjuntos de datos externos no eran lo suficientemente efectivos en las imágenes propias del proyecto. La creación de un dataset personalizado permitió mejorar sustancialmente el desempeño del modelo, aunque persistieron confusiones en caracteres con formas similares.

- En términos generales, el sistema desarrollado logró buenos resultados en la detección y reconocimiento de placas, aunque aún presenta oportunidades de mejora. La optimización del modelo, el refinamiento del conjunto de datos y la implementación de técnicas adicionales de preprocesamiento podrían contribuir a reducir los errores en la identificación de caracteres.

3. Referencias

1. Ultralytics. (s.f.). YOLOv5 by Ultralytics. GitHub. <https://github.com/ultralytics/yolov5>
2. Roboflow. (s.f.). Roboflow: Herramientas de visión por computadora para desarrolladores y empresas.
3. TensorFlow Developers. (s.f.). TensorFlow (Versión 2.18.0) [Biblioteca de Python]. Google. <https://www.tensorflow.org/>
4. Chollet, F., & colaboradores. (2015). Keras (versión X.X.X) [Biblioteca de software]. Recuperado de <https://github.com/keras-team/keras>