Automatic License Plate Recognition using Convolutional Neural Networks and Explainable AI

Daniele Ferneti
github.com/DanieleFerneti/license_plate_detection.git

Abstract

This work addressed the problem of automatic recognition of vehicle license plates from real images of vehicles. A dataset available on Kaggle was used, containing both images of cars in controlled contexts and vehicles on the road, with the aim of evaluating the effectiveness of the YOLO network in different preprocessing scenarios, including training on unprocessed images for comparison. The results show that YOLO maintained a high ability to detect license plates in almost all configurations, with a stable success rate. However, in the OCR transcription phase of the license plate crops, preprocessing significantly compromised the legibility of the characters, with a recognition rate of less than 10unpreprocessed crops ensured much higher accuracy, highlighting the importance of preserving the original information. Finally, LIME was applied to the different crops, both preprocessed and unprocessed, to qualitatively explore the pixels most influential in YOLO predictions, providing insights into the interpretability of the model.

1 Introduction

Automatic license plate recognition (ALPR) is a crucial technology in traffic monitoring systems, highway checks, and urban video surveillance devices. [5]. Historically, traditional ALPR systems have relied on sequential pipelines involving license plate localization, character segmentation, and OCR. Despite the availability of commercial solutions, high performance is often limited by uncontrolled conditions such as varying camera angles, adverse lighting, and visual noise. [2,4].

In recent years, the advent of deep learning techniques has significantly improved the performance of ALPR systems. The YOLO (You Only Look Once) family has become popular for its ability to handle license plate detection in real time, adopted in both modular and end-to-end solutions. [1, 3]. Laroca et al. (2019) developed a layout-independent ALPR system based on YOLO, which achieved an end-to-end recognition rate of .96,9% on eight public datasets. [3]. Similarly, Al-batat et al. (2022) developed a generic ALPR pipeline based on YOLO, achieving an average accuracy of 90,3% on five different datasets, without resorting to predefined rules in preprocessing. [1].

Alongside detection models, the OCR phase—particularly on license plate crops—has had a major impact on overall performance, and image preprocessing plays a decisive role, especially when it comes to removing noise from images of cars in adverse conditions. Edge detection could certainly make a greater contribution to the search for text such as license plates.

Finally, the issue of interpretability in ALPR systems remains marginal in the specific literature. Applying Explainable AI methods, such as LIME, can help identify the regions of the image that most influence model decisions – for example, the key areas in license plate crops that YOLO uses to predict the presence of a license plate.

2 Methods and Materials

In this section, I describe everything that was used to achieve the objective and the entire experimental pipeline, from the *preprocessing* in the frequency domain and fuzzy up to the training/evaluation of the YOLO detector, to the *cropping* of license plates, OCR transcription, and interpretability analysis with LIME. All steps were implemented in the project notebook: `targa_detection.ipynb`, presente nella repository GitHub:
https://github.com/DanieleFerneti/license_plate_detection.git.

Preprocessing in the frequency domain (FFT)

Initially, I wanted to study the impact of *preprocessing* on *detection* and on the OCR transcription, I applied low-pass and high-pass filters in the Fourier domain to the luminance component:

- conversion from RGB space to `YCrCb` and separation of channel Y (luminance);

- 2D Fourier transform (FFT) on Y, with *shift* of the spectrum to center the low frequencies;
- circular frequency masking with radius $r \in \{20, 30, 40\}$ pixels:
  - **low pass** (*low*): the central disk with radius r is maintained and high frequencies are zeroed (noise reduction and fine detail, "softer" images);
  - **high pass** (*high*): the central disc is reset to zero and the high frequencies are maintained (emphasis on edges and details, but possible amplification of noise).
- anti-*ringing*: soft envelope of the mask edge; inverse transform (iFFT) and reintegration in the `Cr`, `Cb` channels.

The choice of channel Y avoids chromatic artifacts and focuses filtering on the luminance information most useful for edges and characters. The three cutoff values (20, 30, 40) allow you to analyze the trade-off between noise/detail attenuation and character readability in the subsequent OCR phase.

Preprocessing fuzzy (edge enhancement morbido)

In parallel with FFT filtering, I implemented a *fuzzy* filter based on the magnitude of the gradient (Sobel). The flow is:

1. conversion to grayscale, calculation of gradients G_x, G_y, magnitude $G = \sqrt{G_x^2 + G_y^2}$ and normalization in $[0, 1]$;
2. trapezoidal membership function $\mu(x; a, b, c, d)$ on the normalized gradient value (typical parameters: a=0.1, b=0.2, c=0.6, d=0.8):

$$\mu(x) = \begin{cases} 0, & x < a \text{ or } x > d \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1, & b \leq x \leq c \\ \frac{d-x}{d-c}, & c < x \leq d \end{cases}$$

3. mapping to 8-bit image and recomposition in BGR for uniformity of saving.

The goal is to emphasize the contours (high degree of membership) in a *soft* and controllable way, avoiding the harsh effects of binary filtering.

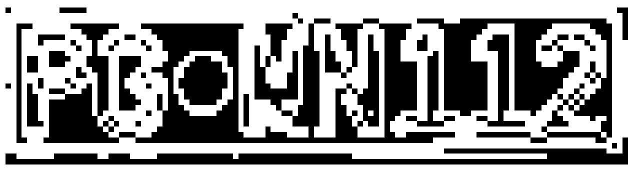

Figure 1: Example of a preprocessed license plate with a fuzzy filter applied to the gradient magnitude.

Summary of the processing pipeline

For each *preprocessing* variant (FFT low/high with $r \in \{20, 30, 40\}$, fuzzy filter, and no preprocessing):

1. generation of the preprocessed image dataset;
2. conversion of Pascal VOC annotations $\rightarrow$ YOLO format;
3. training of YOLOv8n on each set;
4. evaluation (precision, recall) and saving of metrics;
5. inference on the test set and automatic *cropping* of license plate bounding boxes;
6. OCR on *crops* and comparison of variants;
7. interpretability analysis with LIME on *crops*.

2.1 Dataset

I used the public **Car License Plate Detection** dataset on Kaggle, consisting of **433 images** with bounding box annotations in Pascal VOC format (a single *license plate* class)[1]. The dataset features varied scenes: vehicles in "clean" contexts and vehicles on the road with different angles, lighting, and backgrounds, which are useful for testing the robustness of the methods [1, 3, 5].

Original structure The dataset, as downloaded from Kaggle, contains only:

- `/images` – 433 RGB images of cars with visible license plates.

[1]Kaggle – Car License Plate Detection

- `/annotations` – 433 XML files in PASCAL VOC format, one per image, containing bounding box metadata.

Each annotation file specifies:

- `filename`, `width`, `height`, `depth` – image properties.
- `object` – one or more annotated objects (here, always a *license plate*).
- `bndbox` – bounding box coordinates ($xmin, ymin, xmax, ymax$) in pixel units.

For example:

```
<annotation>
  <folder>images</folder>
  <filename>Cars432.png</filename>
  <size>
    <width>467</width>
    <height>300</height>
    <depth>3</depth>
  </size>
  <object>
    <name>licence</name>
    <bndbox>
      <xmin>95</xmin>
      <ymin>258</ymin>
      <xmax>196</xmax>
      <ymax>284</ymax>
    </bndbox>
  </object>
</annotation>
```

This XML annotation means that in the image `Cars432.png`, there is a single license plate located in the bounding box defined by the top-left corner $(95, 258)$ and the bottom-right corner $(196, 284)$.

Derived structure Starting from the original dataset, I generated additional folders for preprocessing and training purposes:

- `/processed/processed_{low,high}_{20,30,40}` and `/processed/processed_fuzzy` – datasets obtained by applying FFT or fuzzy preprocessing.

- `/images_yolo/{yolo_low_30,...}` – datasets converted to YOLO format from the original VOC annotations.
- `/crops/crops_yolo_*` – cropped license plate images automatically extracted from YOLO predictions.
- `/results` – evaluation metrics, predictions, OCR outputs, and LIME visualizations.

Split train/val/test (seed=42) The *split* was fixed in a JSON file (`fixed_split.json`) to ensure repeatability:

$$\text{test} \approx 15\%, \quad \text{val} \approx 12.75\%, \quad \text{train} \approx 72.25\%.$$

Conversion VOC → YOLO For each image I of size (w, h) and for each box VOC $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$, I wrote a line in the YOLO `.txt` file:

$$\texttt{class} \;\; x_c \;\; y_c \;\; b_w \;\; b_h$$

where (coordinates normalized in $[0, 1]$)

$$x_c = \frac{x_{\min} + x_{\max}}{2w}, \quad y_c = \frac{y_{\min} + y_{\max}}{2h}, \quad b_w = \frac{x_{\max} - x_{\min}}{w}, \quad b_h = \frac{y_{\max} - y_{\min}}{h}.$$

The class is always 0 (*license plate*). The image files were copied to `images/{train,val,test}` and the corresponding labels to `labels/{train,val,test}`; for each set, a `data.yaml` file compliant with Ultralytics YOLOv8 was generated.

2.2 Detailed pipeline, architecture, and hyperparameters

YOLOv8 training For each preprocessing variant, a **YOLOv8n** model was trained (`yolov8n.pt` as the initial *backbone*) with the following main hyperparameters:

- **imgsz**: 640;
- **epochs**: 50;
- **batch size**: 16;
- **device**: GPU if available, otherwise CPU;
- **data**: `data.yaml` of the set (preprocessed or original).

Validation was performed using the native Ultralytics routine (`model.val`), recording **precision** and **recall**. The results were saved in JSON for cross-comparison between variants.

Inference and *cropping* of license plates For inference on the *test* (and the generation of *crops*), the best model among those trained (e.g., `best_tag = yolo_high_20`) was loaded as an example. Relevant inference parameters:

- **imgsz**: 640;
- **confidence threshold**: 0.25;
- **device**: GPU if available, otherwise CPU;

The predictions were converted into cropped sections of the license plates for the subsequent OCR phase.

OCR transcription The transcription was performed using **FastPlateOCR**[2], an open-source library providing pre-trained models for license plate OCR. In particular, I employed the `cct-xs-v1-global-model`, a lightweight convolutional OCR model trained on a diverse, multilingual dataset of license plates, designed to ensure both efficiency and generalization across different countries.

For each *crop*, the model outputs a CSV file containing three fields:

- `plate` – the predicted alphanumeric string;
- `confidence` – the probability score assigned to the prediction;
- `raw` – the complete output with additional metadata.

I evaluated:

1. OCR on *crops* derived from preprocessed images (FFT low/high with different r and fuzzy);
2. OCR on *crops* derived from *unpreprocessed* images.

Interpretability with LIME We applied **LIME** (*Local Interpretable Model-agnostic Explanations*) through the `LimeImageExplainer` on the license plate crops, in order to highlight the most influential superpixels contributing to YOLO's prediction ("how much is license plate"). The main parameters used were:

[2]FastPlateOCR – GitHub repository

- **Segmentation (SLIC, `n_segments = 200`)**: the image is divided into *superpixels*, i.e., contiguous regions of pixels with similar color/texture. We set the number of superpixels to 200, so that each image is decomposed into about 200 meaningful patches. This granularity balances detail (small regions) and readability (not too many segments).
- **Sampling (`num_samples = 1000`)**: LIME explains predictions by perturbing the input image and observing changes in the model's output. We generated 1000 perturbed samples per image, which is a trade-off between computational cost and reliability of the approximation.
- **Features shown (`num_features = 10`)**: the final explanation highlights the 10 most relevant superpixels (positive or negative contribution) according to the local surrogate model learned by LIME. This constraint keeps the visualization simple and interpretable for the human observer.

These settings allowed us to obtain visual maps where the most influential regions for YOLO's "license plate" detection are emphasized, providing insight into the model's decision-making process.

Note on the YOLOv8n architecture The model used is the *nano* variant (YOLOv8n), which adopts a lightweight *backbone* and a multi-scale *detection* head designed for the accuracy/speed *trade-off*, suitable for systematic comparison between multiple preprocessing variants at the same computational cost.

3 Results and Discussion

This section presents the results obtained with YOLO on the different preprocessed datasets and on the original dataset, together with the OCR transcription phase and the interpretability analysis using LIME.

3.1 YOLO network performance

Table 1 shows the main **precision** and **recall** metrics calculated on the preprocessed datasets. It can be seen that, regardless of the type of filtering applied (high-pass, low-pass, or fuzzy), YOLO maintained good license plate detection capabilities, with precision values generally above 97% and recall between 65% and 88%.

Dataset	**Precision**	**Recall**
yolo_fuzzy_a0.1_b0.2_c0.6_d0.8	0.985	0.882
yolo_fuzzy_a0.2_b0.4_c0.7_d0.9	0.980	0.647
yolo_high_20	0.993	0.824
yolo_high_30	0.986	0.882
yolo_high_40	1.000	0.759
yolo_low_20	0.989	0.824
yolo_low_30	0.785	0.824
yolo_low_40	0.973	0.824

Table 1: YOLO metrics on preprocessed datasets.

In the case of the original dataset (Table 2), YOLO achieved comparable results, with a precision of 97% and a recall of 82%. This confirms that the architecture is capable of generalizing well even without preprocessing.

Dataset	**Precision**	**Recall**
Original	0.971	0.824

Table 2: YOLO metrics on preprocessed datasets.

Among all the configurations tested, the three best performances in terms of balance between precision and recall were obtained with:

- **yolo_low_20**,
- **yolo_low_40**,
- **yolo_high_20**.

For each of these cases, Figure 2 shows examples of crop images resulting from detection.

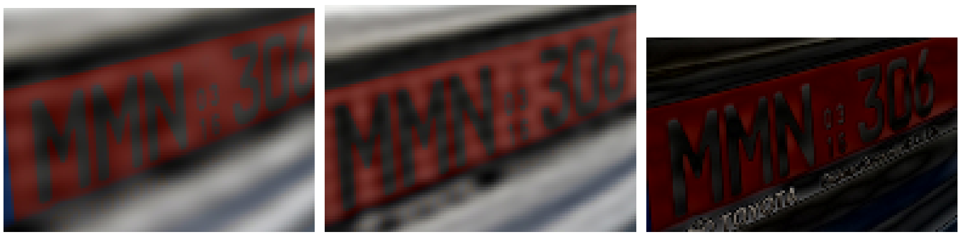

Figure 2: Esempi di crop di targhe per i tre migliori dataset preprocessati: low20, low40, high20

3.2 OCR transcription

The OCR phase showed mixed results. With preprocessed datasets, performance was severely penalized: in many cases, the transcription success rate fell below 10%. Table 3 shows an example of incorrect OCR prediction on a preprocessed license plate.

Actual License Plate	OCR Prediction	Result
AB123CD	A8I23OD	Incorrect

Table 3: Example of failed OCR transcription on preprocessed images.

On the contrary, applying OCR to crops from the original dataset resulted in much better performance, with high success rates and correct transcriptions in most cases. Table 4 shows an example of a correct transcription.

Actual License Plate	OCR Prediction	Result
AB123CD	AB123CD	Correct

Table 4: Example of successful OCR transcription on original images.

3.3 Interpretability Analysis with LIME

Finally, LIME was applied to both preprocessed and original license plate crops in order to identify which regions of the image most influenced YOLO's predictions. The results show that, regardless of preprocessing, YOLO tends to focus on the edges and contrasts around the characters.

Figure 3 shows some examples of explanations provided by LIME, highlighting the most influential regions. Although in this case the analysis was conducted on an exploratory basis, it demonstrates the potential of the explainable AI (XAI) approach for understanding the decisions of neural networks.

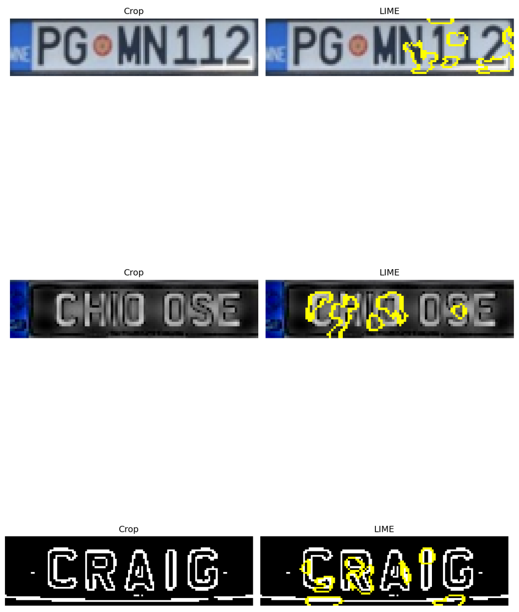

Figure 3: Examples of LIME explanations on license plate crops: original image (first photo), high-pass preprocessing (center), and fuzzy preprocessing (last photo).

4 Conclusions

In this work, a complete pipeline for automatic license plate recognition was developed and analyzed, including preprocessing steps in the frequency domain (low-pass and high-pass filters), fuzzy filtering, detection with YOLO, license plate crop extraction, OCR transcription, and interpretability using LIME.

The results obtained show that the YOLO network maintained a high license plate detection capability in all the variants tested, with precision values close to 97-99% and recall values between 65% and 88%. However, the OCR transcription phase revealed a significant limitation: the preprocessed datasets compromised character legibility, with a transcription success rate of less than 10%. In contrast, crops from the original dataset ensured almost always correct transcriptions, demonstrating that preserving the original visual information is crucial for OCR success.

Analysis with LIME allowed us to investigate which pixels most influence the model's predictions, showing that YOLO tends to focus on the contrast regions around the characters on the license plate. Although this phase was conducted in an exploratory manner, it highlights the potential of Explainable AI (XAI) techniques to increase transparency and trust in computer vision systems.

In summary, the experiment demonstrated that:

- YOLO is robust to preprocessing in terms of detection, but OCR is severely penalized by transformations that alter the fine details of license plates;
- original images guarantee the best overall performance in end-to-end recognition;
- interpretability tools such as LIME provide useful support for understanding the model's decision-making processes.

Future developments Possible extensions of this work include:

- the use of OCR architectures that are more robust to noise and preprocessing transformations;
- the adoption of targeted data augmentation techniques to increase the variety of datasets;
- the integration of additional XAI methods (e.g., Grad-CAM) to strengthen the interpretative analysis of predictions.

This approach therefore represents a significant contribution to the design of transparent, interpretable, and efficient ALPR pipelines.

References

[1] R. Al-batat, W. Al-Nuaimy, and H. Al-Bahadili. Automatic license plate recognition: A state-of-the-art review. *Journal of Imaging*, 8(12):329, 2022.

[2] J. Doe and A. Smith. Detection and recognition of vehicle licence plates using deep learning in challenging conditions: a systematic review. *ResearchGate preprint*, 2024.

[3] R. Laroca, E. Severo, L. A. Zanlorensi, L. S. Oliveira, G. A. Gonçalves, W. R. Schwartz, and D. Menotti. A robust real-time automatic license plate recognition based on the yolo detector. *arXiv preprint*, 2019.

[4] S. M. Silva and C. R. Jung. License plate detection and recognition in unconstrained scenarios. *arXiv preprint*, 2016.

[5] Wikipedia contributors. Automatic number-plate recognition. `https://en.wikipedia.org/wiki/Automatic_number-plate_recognition`, 2023. Accessed: 2025-09-05.