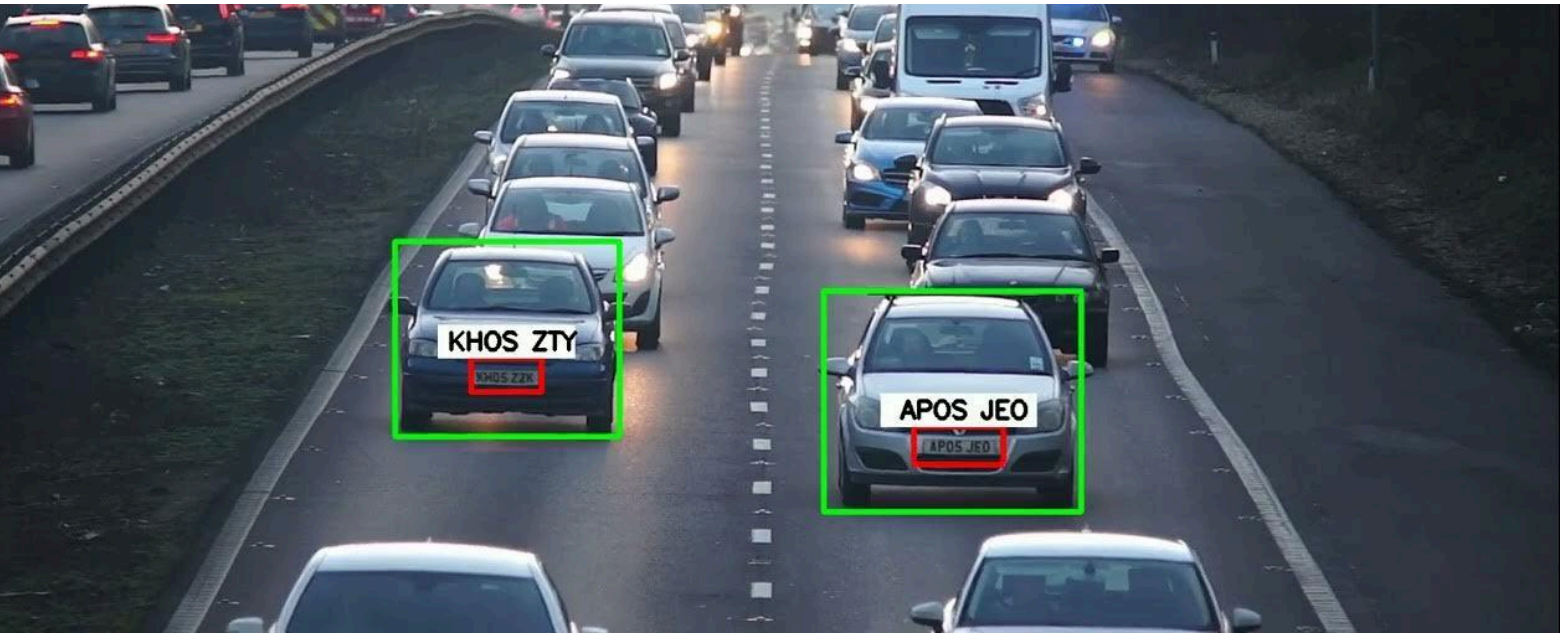
UAB
Universitat Autònoma de Barcelona

Automatic Number-Plate Recognition



Submitted by:

- Luis Domene García, 1673659
- Eric López Cervello, 1672981
- Marino Oliveros Blanco, 1668563

Vision & Learning

15-10-2024

1. Introduction

Project Overview

Motivation

The motivation behind this project stems from the increasing need for efficient, automated systems to manage traffic, security, and urban infrastructure. Automatic Number-Plate Recognition (ANPR) systems offer a solution to this by providing automated detection and recognition of vehicle license plates, which are essential for toll systems, parking control and access management.

These plates have a specific layout consisting 4 digits and 3 character letters in all caps using specific fonts ‘alte din 1451 mittelschrift’ family and specific layouts that usually consist of a white rectangle with a small fraction to the left containing a blue rectangle with the letter E for España/Spain. The white rectangle is always wider than tall but can change sizes depending on the vehicle type and model (e.g Motorcycles, Alfa Romeo cars).

Objectives and Goals

The primary objective of this project is to develop a functional ANPR system capable of recognizing Spanish license plates based on the current format (four numbers followed by three letters). The goals include:

- Building a pipeline that given an image detects license plates, segments characters and recognizes the four numbers alongside the 3 letters.
- Achieving high accuracy while ensuring the system works in different environments and lighting conditions.

Key Issues and Challenges

This project addresses several challenges:

1. **Image Quality and Environment**: Images may vary due to different lighting, weather conditions. Ensuring accurate detection in low light requires robust imaging techniques.
2. **Localization of License Plates**: Detecting license plates from vehicle images involves various factors, such as plate orientation and varying sizes. The system must be able to localize plates under such conditions.
3. **Character Segmentation and Noise**: The segmentation process is often hindered by noise in the image, particularly due to low-resolution.
4. **Character Recognition**: Developing a reliable machine learning model is crucial for accurate recognition.

Techniques and Methods

1. **Image Acquisition**: The first step is acquiring images using cameras. We have used our mobiles to get images. But also used other datasets in order to increase the amount of images and increase the variability of the camera features.
2. **Localization and Detection**: Two main approaches we’ve worked on (Image Processing + Template Matching), use of fine tuned YOLOv11. IntersectionOverUnion
3. **Segmentation of Characters**: - Eric
4. **Recognition of Characters**: we use EasyOCR, Tesseract (not so good), Paddle OCR

2. Methodology

In the methodology section we go deeper into the techniques and mechanics used to achieve our project.

For our pipeline we used YOLO11 for localization and PaddleOCR for our recognition tasks. Our segmentation consisted of certain operations and contour finding. We will now do a deep dive into the architecture of the models used. Later on we will do a comparison on other models/approaches we tried and why our chosen approach results more adequate.

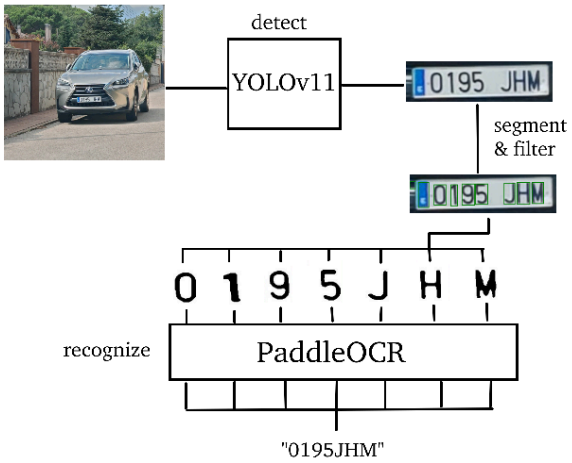


Figure 1. Pipeline of our approach to detect and recognize number plates

YOLO11

YouOnlyLookOnce11 is a computer vision model architecture developed by Ultralytics, the creators of the YOLOv5 and YOLOv8 models. YOLO11 supports object detection, segmentation, classification, keypoint detection, and oriented bounding box (OBB) detection. In our case we utilized it to find the bounding boxes around the license plates, which successfully worked. For YOLO training on bounding boxes you need labeled data which consists of a .txt file per image containing information about the position of the license plate with the following format <class_id> <x_center> <y_center> <width> <height>. These data annotations were obtained in Roboflow.

0 0.45 0.7171875 0.28125 0.053125

Figure 2. YOLO Annotation example

Architecture overview

YOLO11 was built on top of YOLOv8 (released a year prior), and was announced and released to the public in September 2024, because of its recency to find an accurate and detailed representation of its architecture resulted a bit challenging. We will now do an architectural overview of the YOLO11n; the nano version, the smallest of all versions, made to be easily utilized and available for most systems.

Key innovations from previous models include the C3k2 blocks for efficient feature extraction and C2PSA (Cross Stage Partial with Spatial Attention), improving spatial awareness while reducing computational load.

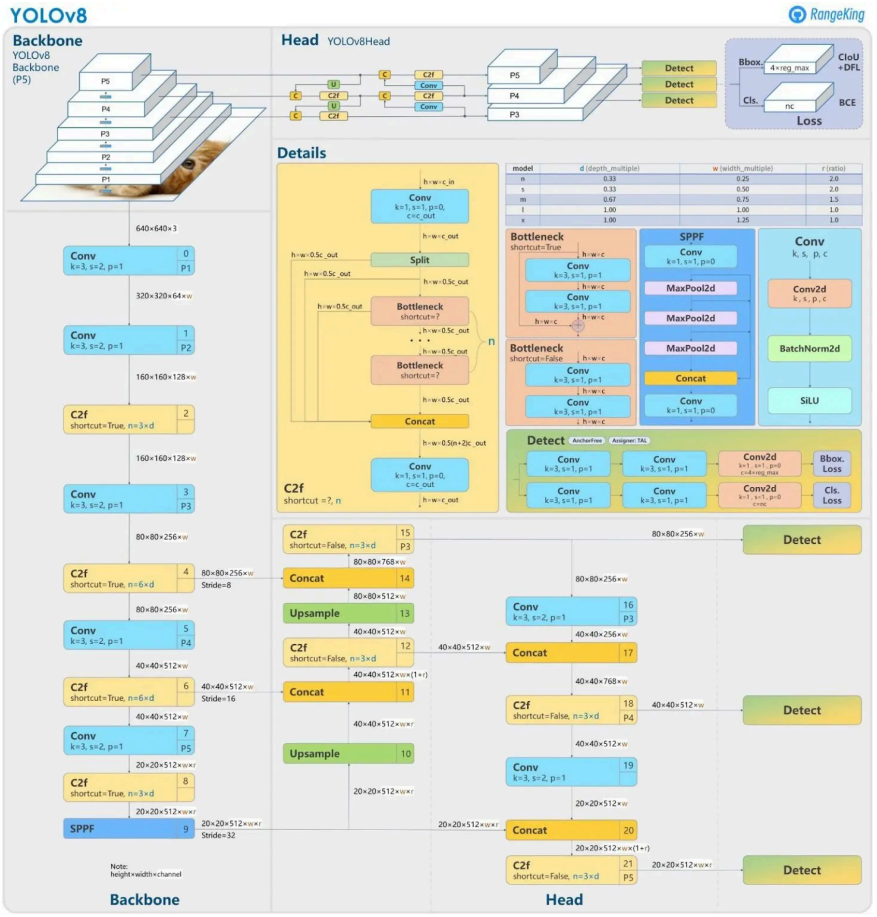


Figure 3. YOLOv8 model architecture overview

We will now describe the backbone, head and flow of the YOLO11 model.

<u>Backbone</u>

YOLO11 uses sequential layers for feature extraction, reducing spatial resolution while increasing channel depth. This backbone consists of convolutional, C3k2 and SPPF layers.

<u>Head</u>

Upsampling and Concatenation operations merge feature maps from different stages of the backbone:

1. Upsample from lower resolution feature maps and concatenate with higher resolution maps.
2. C3k2 blocks continue refining features after concatenation.

3. Downsampling occurs again to focus on medium and large objects.

The final Detect layer predicts object bounding boxes and classes using multi-scale features.

Flow:

1. Backbone extracts features at different scales.
2. Head refines and combines feature maps.
3. Detect predicts bounding boxes and class probabilities at multiple scales for objects of varying sizes.

Notable modules:

- C3k2 Block: A custom residual block with two convolutional layers, likely based on the YOLOv5 C3 structure. It may involve a bottleneck structure and is focused on computational efficiency.

- C2PSA Block: a block that combines 2 convolutional layers with a spatial attention mechanism (possibly positional or pyramid spatial attention) to improve feature representation by focusing on important areas of the image.

- SPPF (Spatial Pyramid Pooling-Fast): A fast, efficient version of the Spatial Pyramid Pooling module, designed to capture multi-scale spatial information and improve detection of objects at various sizes.

These modules together contribute to the model's ability to efficiently extract rich, multi-scale features, which are crucial for accurate object detection. We show an example of our YOLO11 approach at work.

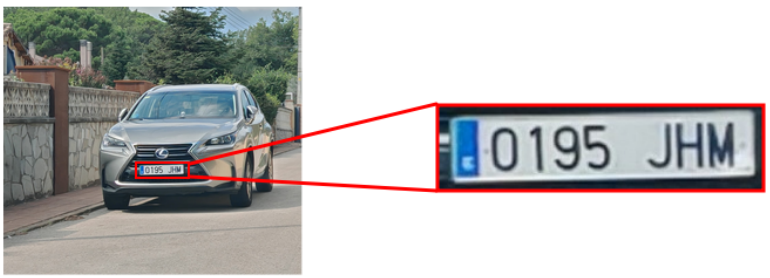


Figure 4. Example of the cropping localization/detection task

PaddleOCR

PaddleOCR is an open-source optical character recognition (OCR) tool developed by PaddlePaddle, designed for detecting and recognizing text in images with high accuracy. We used it for our recognition task once the plates were localized and segmented, focusing particularly on its ability to recognize the text from pre-segmented license plates.

While PaddleOCR offers a complete pipeline that includes text detection and recognition, our use of it is confined to the recognition phase, bypassing the detection and direction classification stages, that is why the architecture overview will be focused on its recognition sector.

Architecture Overview

At the core of PaddleOCR's recognition architecture are several deep learning models, each suited for handling different text characteristics:

- Convolutional Neural Networks (CNNs) extract visual features from the input image, allowing the system to capture patterns related to the shapes of characters.

- Recurrent Neural Networks (RNNs) model the sequential nature of text, enabling the recognition of words as continuous sequences.

- Attention Mechanism enhances recognition by focusing on the most relevant parts of the image, particularly useful for distorted or irregular text.

- Rectification Networks in models like RARE and SRN correct misaligned or curved text, which is common in real-world scenarios like license plates.

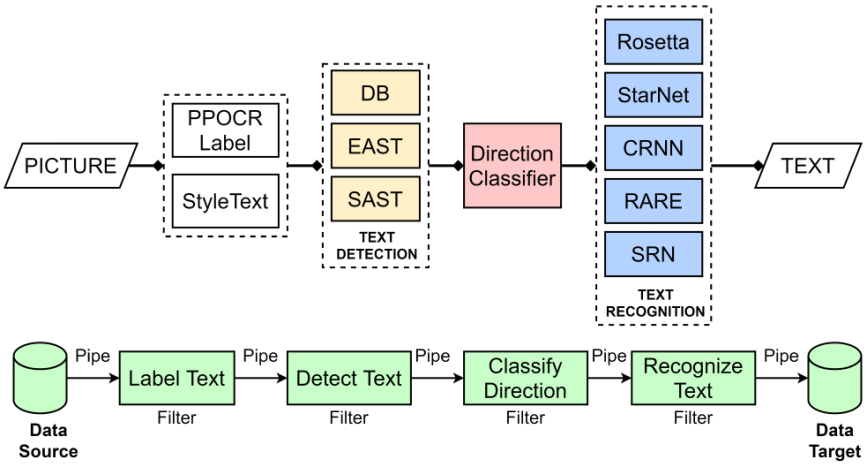


Figure 5: Complete PaddleOCR Pipeline Diagram

PPOCR

We chose PPOCR, an efficient and lightweight version of PaddleOCR that is optimized for performance and accuracy. PPOCR offers a straight, simple approach to OCR, making it highly suitable for real-time applications such as vehicle identification through license plate recognition. On more technical detail here is why it is appropriate for us:

1. Lightweight and Fast: PPOCR includes lightweight models such as PP-OCRv3, which are optimized for both speed and accuracy. This makes it highly suitable for real-time recognition tasks, like identifying license plates in traffic or surveillance systems, where speed is crucial, also we are trying to do the video part of the project so a 'reasonably fast' model is crucial.

2. High Accuracy: PPOCR's recognition models, including CRNN, RARE, and SRN, are specifically designed to handle a variety of text distortions, such as the slight curvature or noise found in license plates. The built-in rectification mechanisms in these models ensure that even misaligned text can be accurately recognized.

3. End-to-End Pipeline (Recognition Focus): Since we provide pre-segmented plate images, PPOCR's modular design allows us to bypass the text detection phase entirely and focus only on text recognition. This means the system is leaner and faster, as it only processes the cropped text regions.

4. Adaptability to Real-World Conditions: PPOCR is designed to work with text in various languages and fonts. Fonts are actually important for us as depending on the age and model of a vehicle the plate can have different fonts.

PPOCR is the ideal choice for our license plate recognition task because it balances speed (more will be said later on as comparisons with Pytesseract and SimpleOCR arise), accuracy, and efficiency. Figure 6 shows some example of segments that get feed to the PPOCR model.

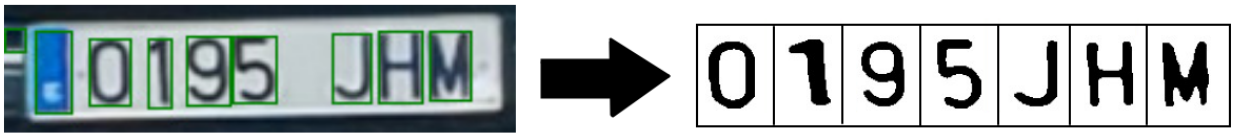


Figure 6. Cropped text regions to process

Figure 7. Pipeline implemented PPOCR recognised plate

2.1 Detection

We implemented two approaches in order to detect license plates in images. The first attempt was a meticulous process involving mathematical morphology operations followed by template matching. After observing that the quality of the detections was not good enough, we proceeded to use a pretrained model, YOLO, to do the task. It was our job to fine tune it on license plates detection.

Approach 1: Mathematical Morphology and Template Matching

This approach involves morphological image processing operations (MathMorph) and uses the result with template matching in order to detect the part of the image where the plate is. We've tried different combinations of operations but the procedure that shows best results with the template matching was the following:

1. Image Preprocessing and Resizing

The first step involves loading and resizing the images to a fixed resolution for uniformity and processing speed. We use the resize_and_pad function to adjust the images to an HD resolution (1280x720). This function maintains the aspect ratio of the original image while resizing and pads the image with black borders to fit the target size. This ensures that all images are processed at

the same resolution, which is critical for consistent performance in later steps.The objective of this step is to standardize the input images to ensure uniformity in the processing pipeline. It also makes the subsequent processing steps faster and more manageable.

2. Conversion to Grayscale and Smoothing

Once the images are resized, they are converted to grayscale, which simplifies the subsequent operations by reducing the image to a single channel. A Gaussian blur is then applied to remove noise and smooth the image. This operation is for reducing unwanted details that might complicate the detection of the license plate.

3. Black Hat Morphological Operation

The next step uses a morphological operation known as "black hat." This operation highlights dark regions on a light background, which is particularly useful for detecting the license plate, as it typically appears as a dark region surrounded by lighter areas.

The black hat operation enhances the contrast of the license plate by highlighting darker areas, making it easier to detect in the next steps.

4. Morphological Opening

Following the black hat operation, a morphological "open" operation is performed. This step helps remove small artifacts or noise around the detected dark regions. The open operation removes small white noise while preserving the structure of the darker regions, such as the license plate.

5. Rectangular Black Hat Operation

A second black hat operation is applied using a rectangular structuring element. This step is specifically designed to emphasize horizontal structures, such as the shape of a license plate. By using a rectangular kernel, this operation enhances the features that correspond to the license plate's dimensions, making it easier to isolate from the background.

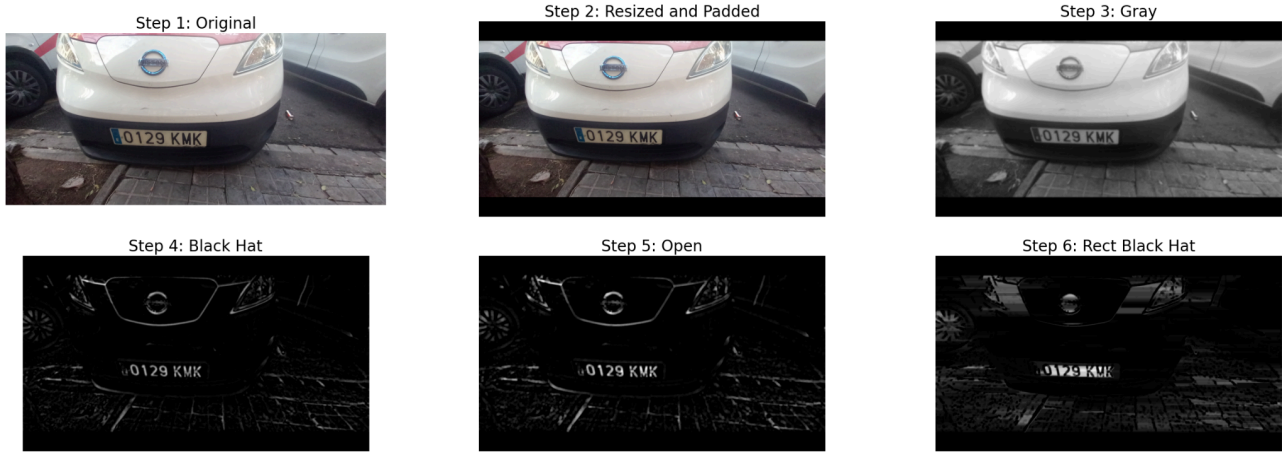


Figure 8. Pre-Processing Process

After these operations, the license plates should be highlighted, a simple template matching of a white rectangle should be enough to detect them. However, the results in real images turn to be very inconsistent for some reasons:

- The template does not match the variation in size of the different plates, so if the plate is too small or too big the template does not fit properly.
- A similar problem was found related to the angle variation of the plate. At the end the template was just a white rectangle so when the angle makes the plate look like something that is not a rectangle it had several problems.

To fix those problems some tests were performed. One of them was that given that we reduce the size of the image and it is quick to perform the processing + template matching we could use different template sizes and orientations then use the one with the highest matching score. But the approach didn't work as expected. In figures X,X and X you can observe some of the results along the images when the pre-processing is applied and the heatmap of the template matching.

Figure 9. Results of Morph + Template Matching. Detection

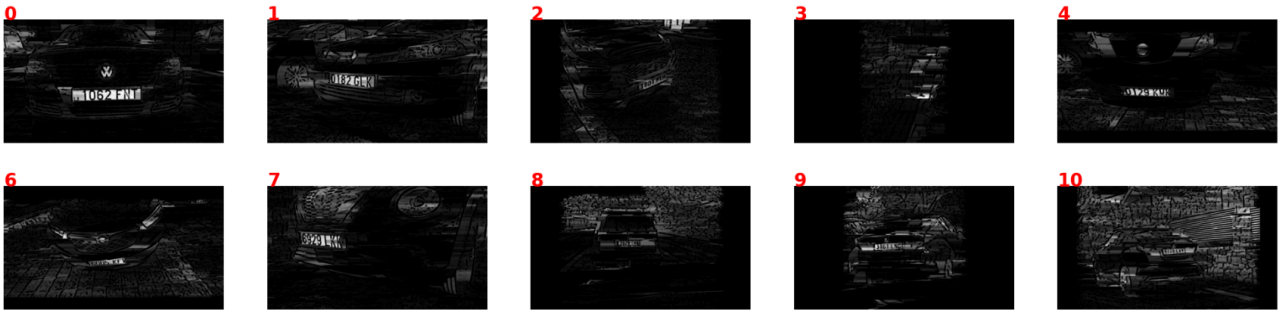


Figure 10. Results of Morph +Template. Pre-processing

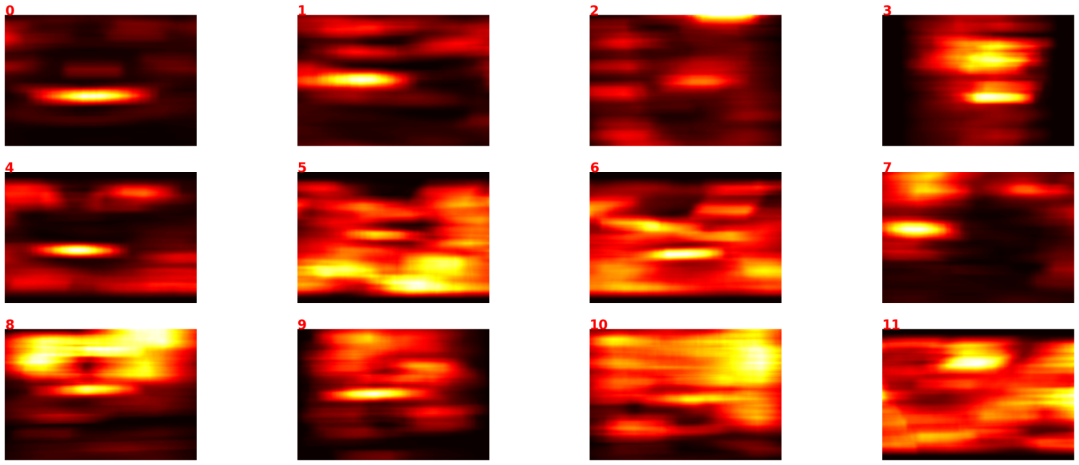


Figure 11. Results of Morph+Template. Template Matching Heatmap

Approach 2: YOLO V11

We also wanted to test how YOLO v11 would work in this task so we decided to fine tune a YOLO11 model to detect the plates and the results were outstanding compared with our approach. We test the performance of one approach and the other on test images. Because of the difference in performance we decided to use YOLO in the final pipeline, the model itself has been explained above.

2.2 Segmentation

Once the license plate is extracted from the image, the next step is to separate each character in order to later recognize them individually. To do so, we perform an elaborated segmentation process that finds regions of the license plate that could contain a character inside.

First, all license plates are resized to a constant shape of 200 pixels in width and 50 in height, in order for the next steps to be applied consistently across all possible license plates. While this is a small size, we found it to be enough resolution for the segmentation process, and makes the processing faster. Then, we convert it to grayscale and further binarize it using adaptive thresholding.

The next step to locate the characters is to find different shapes in the image, and we do so by searching for contours. We treat each different contour we find as a shape, and extract its bounding box. Many shapes can be found in spanish license plates apart from characters, including the leftmost blue box, nails and shadows. We have designed a meticulous method to filter them:

1. Keep shapes whose bounding box has an area between 70 - 800 pixels and are tall rectangles, meaning their aspect ratio (width / height) ranges from 0.1 - 2.0. These constants were determined through trial and error.
2. Remove holes inside characters like 6, 8, 9 and 0. These are defined as shapes of smaller area inside their parent shapes in the hierarchy obtained when finding the contours.
3. Remove blue shapes, like the usual blue box at the left of the license plate.
4. Remove blobs. Blobs are those shapes that remain after eroding them repetitively. Instead, erosion on characters would vanish them easily because they are formed by thin sticks. This is an heuristic assumption we have made.

We've observed after applying this segmentation process on different images, all characters' bounding boxes are kept, but still sometimes an undesired box remains, usually the leftmost blue box. While a specific filter is set for blue colors, these may still remain because of the night, shadows or other light phenomena that may turn it into very dark blue or even dark gray. However, this is not a problem as we will further apply a filter on the recognition step.

2.3 Recognition

The task of recognizing text in an image is called Optical Character Recognition. At first, we intended to train ourselves a custom model, like a convolutional neural net, to recognize single characters. However, we lacked a dataset of license plate characters appropriate to our task. Training on the EMNIST, a public dataset of hand-written characters, did not yield good results on license plates.

That's why we decided to use a pretrained model. After some research, the most successful open-source models for OCR are Tesseract, EasyOCR and PPOCR (PaddleOCR). We end up using PPOCR, but our decision is based on its proficient results after comparing it with the other two mentioned methods. This comparison is explained in more detail in the Experimental Design section.

Now, once we have the extracted segments from the license plates, we apply OCR on each one. To facilitate the job to the model, we first pad the segment so that some space is left between the character and the borders. Although PPOCR by default also integrates the detection feature, we disable it since our assumption is that each segment is already the bounding box of a character. For each segment we obtain the predicted text and the confidence. We then filter those recognized characters whose confidence is below a specific threshold of 0.2. We found this value through trial and error. It is low enough to keep all recognized characters, but sufficiently high to filter those segments that don't contain, like for example the previously mentioned leftmost blue box that sometimes isn't filtered in the segmentation step.

Finally, once we have a recognized character for each segment, we put them together in a string and apply some post-processing. While Tesseract and EasyOCR allow for a whitelist to only recognize certain characters, PPOCR does not, so it is necessary to only keep the numbers from 0-9 and uppercase letters A-Z in the final recognized text.

3. Experimental Design

3.1 Dataset Description: (Source, key characteristics, annotation process)

Our dataset consists of 139 images of cars and its license plates; these images come in Frontal, Lateral and Vertical POVs. The dataset was obtained by physically taking pictures of cars, in different lightings… All of these images are of Spanish license plates.

Our dataset was annotated to classify the bounding boxes for the localization task, they were annotated with the YOLO11 format using Roboflow. Project id: total-license-plates.

The images were pre-processed to be at a 640 x 640 pixel size. We also used data augmentation on our dataset to end up with 333 images in total. The data augmentation process involved horizontal flips, a shear of 10º both horizontal and vertical and noise up to 0.18% of pixels (as recommended by Roboflow).

Our test data does not present any augmentation.

The dataset split consisted of 70% train, 10% validation and 20% test. We increased the size of the test to make sure we could at least test our model with about 20-30 images. After the augmentations (all occurring on the train set) we obtained 291 images for training, 14 for validation and 28 for testing.

Figure 12: Examples of image augmentation. The first ones are two darkened images that have been horizontally flipped. The third has been slightly rotated.

3.2 Experiments and Metrics

3.2.1 Experiments

Detection + Recognition
Before segmenting and recognizing each character one by one we tried to directly apply easyOCR on the crops of the detection just to see how it would work and could use it as a baseline.

The goal was to evaluate the performance of OCR (Optical Character Recognition) configurations on a small dataset of vehicle license plates using both English ('EN') and Spanish ('ES') settings. The test also compared results with and without preprocessing (such as convert to gray-scale, contrast enhancement (CLAHE), Bilateral Filters). The image-processing procedure can be seen in figures X.

Key Conclusions:
Preprocessing Enhances Results: Preprocessing improved OCR accuracy slightly for both Spanish and English configurations, but not enough to be considered a significant success.

Moderate Accuracy: The overall text similarity scores (around 0.5-0.6) indicate that the OCR failed to perform reliably on this dataset, even with preprocessing. It is fair to highlight that there were some samples that practically were pixels so the average distance on those samples damaged the final result.

Test Discarded: Due to the small dataset and unsatisfactory outcomes, the approach was discarded for future testing or real-world implementation.

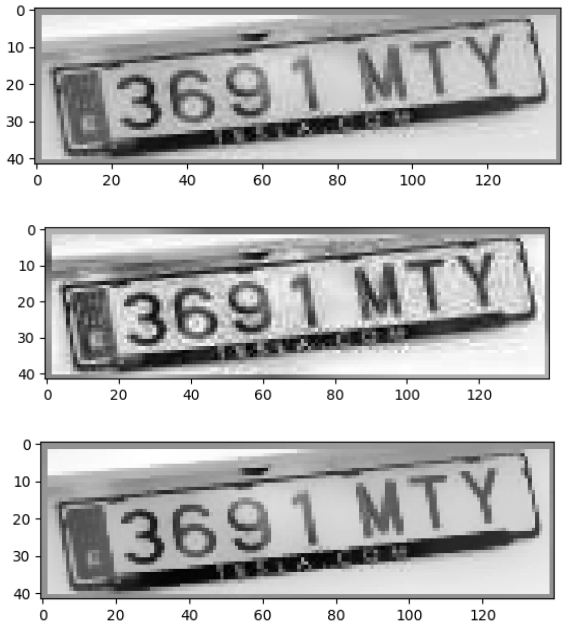


Figure 13. Image Processing.

Synthetic Dataset Generation Experiment

Although not used we adapted from a previous Deep Learning Project a Synthetic Dataset Generator using PIL and the fonts selected from the license plates: ‘alte din 1451 mittelschrift’, it generated a random assortment of 4 digits and 3 characters in a black font over a white rectangle of sometimes different sizes. We wanted to illustrate some of the results.

Figure 14. Synthetic Dataset Examples

YOLO vs MathMorph + Template Matching

Although it was crystal clear that YOLO was performing better than our approach we wanted to compare them in a qualitative way. So we performed an experiment comparing the approach on a 26 images test set. For both methods, we measured the number of True positives (detected a plate and was actually correct), False positives (made a prediction but there was no plate) and False negatives (no prediction but there was a plate). From this, we also compute the precision and recall. More information can be found in the Results section.

Comparing OCR methods

Our evaluation to compare the OCR methods consists of performing recognition on the full text of our license plates’ images extracted after detecting them with YOLO. Although in our pipeline the OCR step is applied character by character, we found it appropriate to evaluate the methods on the whole license plate, for the sake of comparing the three approaches, since we already have an annotated ground truth of the plates’ text.

We use three metrics to evaluate the methods: accuracy, ANLS and average time per image. Accuracy and ANLS are useful to know which has a better performance. The first tells us if the method perfectly recognizes license plates, while the second tells us how much the predictions approximate the ground truth, allowing for mistakes to be made. We decided to also measure the average time, since we consider that a good method should also be fast and efficient, so that it can be used in real-time applications.

3.2.2 Metrics

Detection/Segmentation

For the evaluation of both detection and segmentation steps, we used Intersection over Union (IoU) as the primary metric. We compare the two detection methods, YOLO and Mathematical Morphology + Template Matching, as it effectively measures the overlap between predicted and ground truth bounding boxes. IoU is robust for evaluating object detection performance and was also applied to segmentation tasks in this study, where it was used to assess how accurately the characters were segmented from the vehicle license plates. This consistent use of IoU allowed for a direct comparison of both detection and segmentation performance across different methods.

Intersection over Union (IoU)

IoU measures the overlap between the predicted bounding box and the ground truth bounding box. The IoU score ranges from 0 to 1, where 1 means perfect overlap and 0 means no overlap at all.

$$IoU = Area\ of\ Intersection\ /\ Area\ of\ Union$$

Area of Intersection: The region where both the predicted bounding box and the ground truth bounding box overlap.
Area of Union: The total area covered by both the predicted bounding box and the ground truth bounding box combined.

IoU is a critical metric because it not only checks whether the model predicted the presence of an object but also evaluates how closely the detected bounding box matches the actual object's location.

Threshold for IoU: a threshold (e.g., IoU ≥ 0.5) is set to determine whether a prediction is considered a "correct" detection.

Recognition:

The evaluation of the recognition step is actually the evaluation of the whole pipeline, since this is the last step that depends on the performance of the previous ones. We used different metrics to evaluate the predicted text against the ground truth text, and for simplicity, all of them range from 0-1. Here is an overview of each and why we used them:

Accuracy

The accuracy in this task is simply the rate of correct license plate predictions, assuming correct predictions are those that are exactly equal to their ground truth. This metric is a simple and direct way to know whether the model performs well or not. However, it is very strict: if a prediction only differs from the ground truth in one character, accuracy will count that sample as 0, making the

overall score drop significantly. That's why we decided to look for a more tolerant metric that allows us to know if our model's predictions differ much from the actual text.

Normalized Levenshtein Similarity (NLS)
This metric, inspired by the ANLS proposed by [Biten et al., ICCV'19][1], is designed to smooth OCR recognition mistakes by applying a slight penalty to cases where the recognized output is close to the correct answer but not fully accurate. It uses the Levenshtein distance (or edit distance) between the predicted string and the ground truth string. The edit distance is the number of operations required to transform the first string into the second. By default, the replace operation counts as 2, while the insert and delete as 1.
ANLS was designed for question answering tasks, and while it involves some averaging and a threshold, for our purpose it was enough to take just the fundamental part, shown in this formula:

$$1 - \frac{Levenshtein(pred, gt)}{max(len(pred),len(gt))}$$

This is computed for each text string, and to evaluate on the whole test set we take the average.

Character-level evaluation:
In addition to ANLS and accuracy, we also evaluated the predicted strings in a character-level manner, using traditional metrics like Precision, Recall and F1-score. This was done with the aim of discovering which characters were the more problematic that the recognition model confused, but they can also serve as alternative scores to evaluate the overall model performance:

- Precision: The proportion of correctly recognized characters out of all characters predicted by the model. It is computed as: P = TP / (TP + FP)
- Recall: The proportion of correctly recognized characters out of all actual characters in the ground truth: P = TP / (TP + FN)
- F1-Score: The harmonic mean of precision and recall, giving a balanced measure of the model's accuracy.

We also compute a normalized confusion matrix showing the correct and incorrect character predictions, providing detailed insights into specific recognition errors. Normalizing it makes it easier to identify the confusion in characters regardless of the times they appear in license plates. Having the confusion matrix, it is easy to find which are the characters that the OCR model most mistakens.

4. Results

In this section we present the results obtained for the different experiments we conducted, explained in the previous section, as well as the results of the evaluation of the whole pipeline.

4.1 YOLO vs MathMorph + Template Matching

Table 1 shows the results of this experiment. The MathMorph approach has a very low precision (38.46%) compared to YOLO, meaning a high number of incorrect predictions. Its recall is equally low (38.46%), indicating that many actual license plates were missed. The large number of false positives and false negatives shows that MathMorph struggles to consistently detect license plates correctly. Even by inspection the False Positives of YOLO are indeed an incorrect annotation because they're images where there are more than one plate, so the result could have been even better.

Metric	YOLO	MathMorph + Template Matching
Total Ground Truth	26	26
Total Predictions	28	26
True Positives	26	10
False Positives	2	16
False Negatives	0	16
Precision (%)	92,86	38,46
Recall (%)	100	38,46

Table 1. Ground truth, positives and negatives, and precision and recall of both detection approaches.

Figure 15. Results of Morph +Template Matching Test Set

Figure 16. Results of YOLO Test Set

Figure 17. YOLO False Positives

4.2 Comparing OCR methods

Table 2 shows the results obtained for the comparison of the OCR methods. We can see how PPOCR obtains a much better performance, but it isn't the fastest one. We decided that the big gap in performance was more important and therefore we selected this method for our pipeline.

	Accuracy	**ANLS**	**Average Time**
Tesseract	0.108	0.579	0.208
EasyOCR	0.174	0.625	0.094
PPOCR	0.563	0.799	0.132

Table 2. Metrics obtained on the evaluation of different OCR methods on the full text of license plates

4.3 Recognition and whole pipeline

As previously explained, the result of the recognition is the final result of our pipeline. We evaluated the predicted strings using Accuracy and NLS, and also performed a character-level evaluation using precision, recall and F1 score. We also include the average confidence of all license plate recognitions. These results are from the test set we created, but we also report the evaluation made on the set of images that was originally provided to us in the Virtual Campus, consisting of the set of Frontal pictures and Lateral pictures. These are shown in Table 3. We also provide, for our test set, a box plot showing the NLS distribution in Figure 17 left, and a confusion matrix at the character level in Figure 17 right.

Subset	**Acc.**	**NLS**	**Avg. Conf**	**Char. F1**
Test set	0.5714	0.8352	0.8358	0.8439
Frontal	0.8666	0.9756	0.9426	0.9712
Lateral	0.7059	0.9480	0.9288	0.9060

Table 3. Metrics of the evaluation of our pipeline on our Test set and on the Frontal and Lateral sets provided in the Virtual Campus.

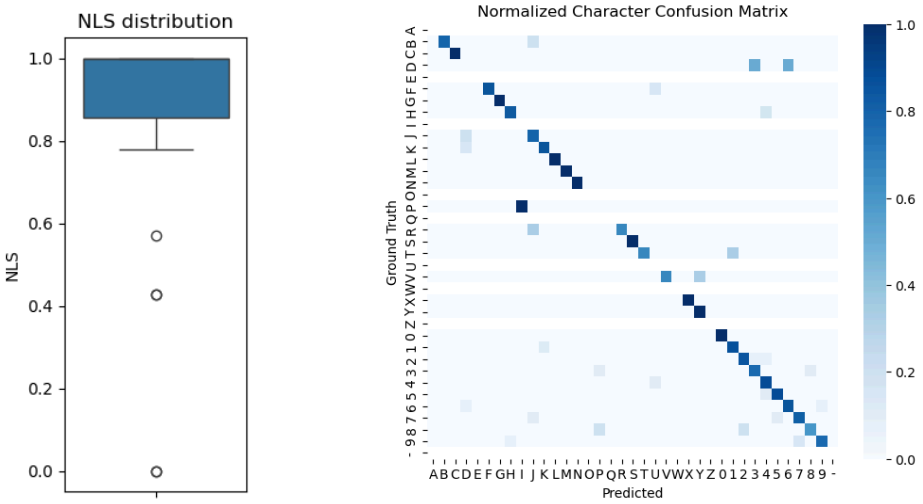


Figure 18. To the left, a box plot of the distribution of NLS on the Test set. To the right, the normalized character confusion matrix on the Test set.

The result of our pipeline is the coordinates of the bounding box where the license plate is located, along with the recognized text and its confidence (which is easily computed as the average of the confidence of each character). Since the process is repeated for each license plate YOLO detects in an image, our pipeline is able to recognize text of multiple plates in a single image, as shown in Figure 18.

Figure 19: Multiple license plates recognized by our pipeline in a single image.

5. Discussion and Conclusions

The results on the test set reveal that our pipeline has difficulties predicting the license plates' text with exact precision. The accuracy is 0.57, which means that slightly less than half of the license plates in the test set are wrongly predicted, but the NLS, which is much higher, shows that it is because of small mistakes. The character F1 score supports this observation. In the box plot we can see that there are a few outliers where the model really struggles to make a prediction. The confusion matrix shows how there are few letters and numbers that are usually mistaken with others, but overall they are mostly correctly predicted, as indicated by the prominent diagonal line. Despite this, we believe that accuracy should be taken more into account if the intention is to apply this pipeline into serious applications where the result needs to be as accurate as possible.

We've also identified that the most commonly confused characters are the following: 9 with 7; B with J; D with 3; D with 6; and F with U. This was unexpected, since they are visually very different from each other, we deduce it is related with the internal representation that our OCR model has of characters.

We believe our pipeline to recognize license plates could be further enhanced in the future. There are some improvements that could be done in the different steps that would likely increase the performance of the models.

First, our dataset contains quality images, as they are diverse and range from very easy cases to very difficult ones. However it is a small set that should be increased with more spanish license plates for a better fine tuning of the YOLO model, and a bigger test set to evaluate with more precision the models.

Second, the preprocessing of the images could be further curated. Currently, in our pipeline, we apply preprocessing steps before the segmentation and recognition steps. While YOLO is pretrained on raw and very diverse images, we have thought of passing our images without any preprocessing. However, there are some cases, for instance when it's dark and it is difficult to see the license plate, where maybe some light correction could be made.

As a last improvement, in this case for our evaluation, is to perform K-folding. This method consists of selecting different random subsets of the dataset to make the test set, and evaluating the different methods on each one. Otherwise, the metrics on a single subset would be biased to the type of data that the subset contains. We did not apply this technique since our dataset was not big enough for this operation to be meaningful, but this should be required to do once our dataset gets bigger.

To sum up, we have used a pretrained model, YOLO, and fine tuned on our dataset, achieving an almost perfect detection of license plates. It was the preferred method after comparing it with a mathematical morphology approach. We have then developed a segmentation technique that is able to accurately identify possible characters in a license plate. Then, with a pretrained OCR model like PPOCR, selected after a thorough comparison, we are able to recognize each of these characters. Putting them together, we obtain the resulting text that our pipeline recognizes in the license plate, achieving a good performance but with some character mistakes that should be taken into account if the pipeline is to be used in high-risk real life systems.

6. References

[1] ANLS: Biten, Ali Furkan, et al. "Scene text visual question answering." Proceedings of the IEEE/CVF international conference on computer vision. 2019. https://arxiv.org/pdf/1905.13648
[2] WER, CER, and MER - Testing with Kolena. https://docs.kolena.com/metrics/wer-cer-mer/
[3] You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640 [cs.CV]
[4] PP-OCR: A Practical Ultra Lightweight OCR System arXiv:2009.09941
[5] Template Matching: https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html