Automated Multiple License Plate Recognition for Dash Cameras

David Peterson
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061
dpeterson23@vt.edu

Abstract

This project presents an efficient and reliable automated multiple license plate recognition (AMLPR) system designed for the dash camera use case. The proposed software is developed using version 3 of the state-of-the-art YOLO object detection algorithm combined with probabilistic inferencing to filter and obtain accurate license plate number predictions from a Python wrapper of Google's OCR engine. The software achieved 100% license plate detection accuracy, 48.87% average license plate number recognition accuracy, and an excellent efficiency of 0.685 seconds for the average processing time on an image from the UCSD license plate dataset. The overall performance of the proposed software is also examined on a custom dataset created using higher-quality hardware representative of modern-day dash camera abilities.
https://github.com/dpetersonVT23/ece4424-spr22-proj

1. Introduction

Over the last decade, an average of 700,000 hit-and-runs (17,000 fatal) were reported in the United States alone every year, causing millions of dollars in damage [1]. Hit-and-runs can occur when a vehicle is being driven or is parked, but the offender is rarely apprehended because their license plate is not acquired. All conventional dash cameras can record driving footage; however, a select few can automatically activate if an impact is detected while parked. Unfortunately, these videos can take a long time to process, have low quality, and have visibility hindrances such as weather and brightness.

Emerging technologies allow data to be transmitted in real-time from a dash camera to an application on a mobile device in an attempt to alert an end-user of a hit-and-run remotely. However, the transmission of video files is time-consuming, requires more processing power, and drains the small battery in the dash camera. Integrated AMLPR in a dash camera could automatically identify relevant data and reduce transmitted data to images, conserving memory and power usage while minimizing transmission times to the end-user and allowing the full recorded videos to be reserved for future review. By merging dynamic AMLPR software into a conventional dash camera, end-users could actively use artificial intelligence to identify, recognize, and save relevant license plate numbers when an accident occurs, ultimately facilitating the offender's capture because every second matters.

1.1. Background

In modern day applications, automated license plate recognition software is used in a variety of environments to facilitate automated vehicle data collection. Some of these environments include public roadways, private premises, and mountings on police squad cars [2]. Deployments of this technology allow end-users to determine when vehicles are present in a certain area and automatically identify them without having to search through or transmit hours of footage. However, as technology advances rapidly in modern society, the detection and recognition software are quickly becoming outdated and comparably inefficient to its successors. Updated sensors, cameras, and machine learning algorithms enable much greater performance for these widely deployed systems in terms of overall accuracy and efficiency. Additionally, the computational power required to run these systems is decreasing, meaning this software can be integrated into highly constrained systems, such as dash cameras, which has yet to reach the market.

1.2. Motivation

As previously discussed, when an offender damages another's property, such as a vehicle, and chooses to flee the scene of the accident for any reason, the victim is not usually aware of the damage until a later time. This lapse in time exponentially decreases the probability of being caught by law enforcement to ensure the offender compensates for the inflicted damage. Even if the victim has a monitoring system to protect their property, detecting and transmitting video evidence of the incidents in real-time is a challenge. Considering these hit-and-run incidents occur on vehicles an average of 2,000 times a day in the United States, causing thousands of dollars in damage in each occurrence, it is not rare for an individual to

experience the perspective of a victim at least one once in the period of owning any vehicle.

With the integration of AMLPR software into dash cameras that protect vehicles by monitoring activity around them at all times, a new level of reassurance is brought to the owner in the case of any accident. By enabling the camera to not only detect when an impact on the vehicle occurs, but also determine when the offending vehicle was in frame of the camera, this would enable the technology to quickly send an image to the victim, as opposed to a video. Using the image of the vehicle with the license plate in view, a computer prediction of the license plate number, and an immediate notification of the incident due to ease of transmission, the victim greatly increases their chances of having the offender brought to justice. Therefore, any individual that is concerned about the well-being of one of the most expensive assets they own, would care for this technology to add further protection wherever they travel in daily life. Additionally, the first time a device with this machine learning technology is used to assist in ensuring inflicted damages are recovered, the technology will have paid for itself ten-fold, thus increasing motivation for all consumers to make the investment in this technology.

2. Approach

The approach to creating the fully functioning AMLPR software that efficiently and accurately identifies and recognizes multiple license plate numbers simultaneously is to split the problem into incremental sub-problems. The core of automated license plate recognition software in machine learning is detecting the location of a license plate in an image or video frame, extracting the characters from the license plate, and performing an optical character recognition (OCR) algorithm to classify the characters into specific letters and numerical digits (all involving learned parameters) [3]. Additionally, processing the outputs for each stage accordingly using image processing and several post-classifier probability inferencing techniques proved to be crucial to increasing recognition accuracy (no learned parameters).

This detailed method of incrementally developing, tuning, and testing individual modules was believed to be successful because completing each module individually prior to combining them into a streamlined system would facilitate debugging and identification of any abnormal or unexpected behaviors. Overall, there is nothing entirely new in the approach taken to construct this system, however, the fine tuning of the models, image processing, and prediction filtering in the software is unique to this project.

2.1. Detection

To handle the license plate detection component of the AMLPR software, a convolutional neural network (CNN) was trained using the state-of-the-art algorithm named YOLO through the Darknet neural network framework, which specializes in efficient and real-time multiple object detection [5]. Although there are multiple versions of the YOLO architecture, version 3 (YOLOv3) was selected for this AMLPR project because version 3 has the most thorough documentation and later versions of the algorithm are based on version 3, using the Darknet-53 network as its backbone.

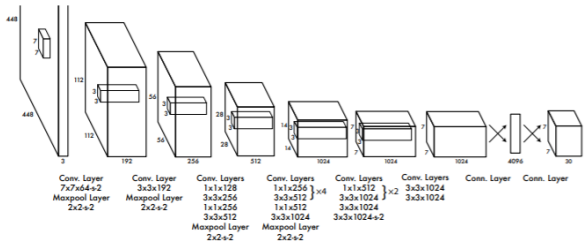

Figure 1: YOLOv3 Architecture with Darknet-53 Backbone [4]

YOLOv3 functions by scoring regions of an image through 53 layers, where higher-scoring regions are noted as positive detections of whatever class the regions most closely identify with, as visualized in Figure 1. Additionally, YOLOv3 is recognized for maintaining similar performance to alternatives while boasting substantial reductions in processing times, as highlighted in Figure 2, which is vital for an application that would process live data, such as a dash camera [5].

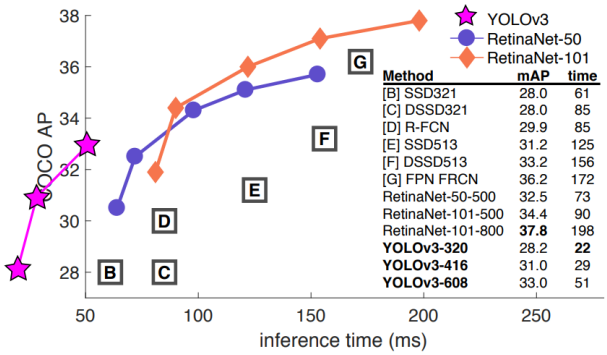

Figure 2: YOLOv3 Efficiency Comparison to Alternatives [5]

To train the YOLO model, the UCSD license plate dataset containing license plate images with location labels was acquired [6]. Data pre-processing was performed on this dataset to reformat the license plate location labels into the format that is compliant with the YOLO training requirements. Additionally, hyperparameters such as the learning rate, batch size, subdivisions, and burn-in were all tuned to optimize the training process because only one classification (license plate) is being trained, whereas the default parameters are set up to for multiple classes. To further accelerate the training process, a pre-trained

Darknet object detection model was utilized as a starting point as opposed to random model weights. The primary problem anticipated during this part of the project was the ability of the detection model to precisely identify the part of the image containing the license plate numbers. In Figure 3, the leftmost image shows an image cropped to the license plate with poor precision and the rightmost image shows an image cropped to the license plate with acceptable precision.

Figure 3: License Plate Detection Precision Comparison

The difference between the images in Figure 3 is the extra undesired text surrounding the license plate number that is included in the poorly cropped license plate image, such as the state name or motto. This additional text was anticipated to disrupt the efforts of the OCR algorithm to correctly identify the license plate number. Following initial observations of the trained YOLO model's performance, without any detection tuning, poor precision was observed on some of the testing data. Fine tuning was applied to the detection cropping to minimize extra undesired text and increase the precision of the detections to obtain results closer to the image shown on the right in Figure 3.

2.2. Image Processing

As previously discussed, the lower quality of the UCSD license plate images raised anticipated problems regarding the ability of OCR algorithm to recognize and correctly classify the characters, especially once the image is zoomed in and cropped to only the license plate. By applying several image processing techniques to these images to transform the cropped license plate into an OCR-friendly format, the performance of the OCR algorithm will be maximized. Additionally, the required supplementary tuning of the OCR algorithm will be minimized.

The image processing component of the AMLPR software encompasses five stages of modifications to the license plate image in a particular order. These stages include resizing, grayscale conversion, noise suppression, binarization, and dilation. During the development of the image processing component of the AMLPR software, several rounds of experimentation were performed using different approaches. The cv2 Python package documentation was primarily used as a resource for determining available image processing techniques [7]. Several different combinations of techniques with varying parameters in different orders were tested and resulted in the finalized process as previously highlighted. Firstly, the original license plate image is resized by a factor of four on both axes to generate more pixels and inherently more details in the image. An example of a UCSD license plate image at this state in the image processing is pictured in Figure 4.

Figure 4: Resized UCSD License Plate Image Initial State

Following the resizing, the colorspace of the image was converted to grayscale to facilitate the upcoming image thresholding (binarization). However, prior to the image thresholding, a gaussian blur is applied using a 5x5 kernel to remove image noise. Image thresholding is then applied to convert all pixels to black or white, using the OTSU threshold provided by the cv2 Python package. As a final step and attempt to remove undesired noise in the image, dilation is applied to the binarized image using a 3x3 kernel. The final result of a UCSD license plate image after the application of the discussed processing techniques is shown in Figure 5 and represents the input to the recognition module of the AMLPR software.

Figure 5: Processed UCSD License Plate Image Final State

2.3. Recognition

To handle the license plate number recognition element of the AMLPR software, the PyTesseract package was utilized with several optimizations to tune the algorithm for the license plate use case. PyTesseract serves as a "wrapper for Google's Tesseract-OCR engine" that can be optimized for different text applications [8]. The configurations made to the PyTesseract OCR algorithm include tuning the page segmentation mode and the OCR engine mode. The page segmentation mode is set to a value of 6, signifying the engine should look for a "single uniform block of text" (i.e., the license plate number) [8]. The OCR engine mode is set to a value of 3, which simply means using the resources that is available on the current runtime. The majority of the effort concerning the recognition aspect of this project is contained in the prediction filtering.

2.4. Prediction Filtering

The prediction filter serves a critical purpose as the last stage in the AMLPR software prior to outputting the finalized license plate number prediction. This filtering combines deterministic and probabilistic techniques to increase the likelihood that the final prediction is as close as possible to the actual license plate number. The problems anticipated from the raw output of the PyTesseract license plate number prediction are that any extra undesired numbers, symbols, logos, or edges may be included. For example, "1 4URR806 \" is the raw output for the processed license plate in Figure 5. The OCR algorithm has incorrectly identified the edges of the cropped license plate images as extra characters in the license plate number and promotes the reasoning for creating the prediction filtering.

The first step in the filtering is to whitelist certain characters that appear on license plates. These whitelisted characters include all digits and all capital letters in the alphabet, meaning any other classified characters, such as "\" are removed immediately. Referring to the PyTesseract configuration tuning, recall the page segmentation mode being set to a value of 6, to search for a "single uniform block of text." This does not mean to look for only one uniform block of text, but rather any individual groupings of text. This behavior causes the OCR algorithm to identify the black edges on either side of the license plate as two separate uniform blocks of text aside from the license plate number in the middle. This is further highlighted in Figure 6 where a poorly cropped license plate is inputted into the OCR algorithm. This input induces several recognitions of separate blocks of text to be identified including the inspection stickers, the state name, the license plate number, and the state motto.

Figure 6: Recognized Uniform Blocks of Text from PyTesseract

By using this specific page segmentation mode, the detailed PyTesseract output can be used to the software's advantage to analyze the confidence values for each individual block of text. Using a tuned confidence threshold to filter out predictions are statistically unlikely to be apart of the license plate number, extra undesired classifications are removed. For the provided example, "1", "4URR806", and "\" are all individual blocks of text. Utilizing the described filtering method, the original raw output of the processed license plate is reduced to "4URR806" for the final license plate number prediction.

2.5. Combined Software

Following the thorough development, tuning, and testing of each individual module that composes the AMLPR software, these modules were combined into a streamlined and user-interactive program that is executed on the commandline (licenseplate.py). For images, the program scans for multiple license plates and individually predicts each detected license plate number. The program concludes by providing the user the number of license plate detections, the predicted license plate numbers from the detections, and the option to view the cropped license plate detections. For videos, the program examines every second of the video by selecting frames based on the video framerate. These individual frames are examined similarly to an image, where the only difference is the video inputs utilize uniqueness checking in an attempt to filter out duplicate detected license plates (i.e., the license plate remains in view for more than a second during the video and gets detected in multiple reviewed frames). An example of a commandline interaction with the AMLPR software program is visualized in Figure 7.

```
[STATUS] Parsing commandline arguments...Done!
[STATUS] Loading YOLO model...Done!
[STATUS] Processing datasets/custom/images/IMG_6910.jpg...Done!
[RESULT] Processing time for datasets/custom/images/IMG_6910.jpg: 0.798 seconds
[RESULT] Number of detected license plates in datasets/custom/images/IMG_6910.jpg: 1
[RESULT] Predictions of recognized license plates: ['JTD0100']
[REQUEST] Would you like to visualize detections? [Y/n]:
```

Figure 7: Commandline Interaction with AMLPR Software

Further details for using the program on the commandline can be found in the usage section of the GitHub README, as discussed in the *Availability* section.

3. Experiments and Results

To measure success for the designed and developed AMLPR software, an evaluation script was written that corresponds to the evaluation metrics outlined in the project proposal. These objective evaluation metrics include license plate location detection accuracy, license plate number recognition accuracy, and overall AMLPR software efficiency. These metrics were measured based on the performance of the AMLPR software on the UCSD license plate dataset and additional observations were made on the performance of the AMLPR software on the custom license plate dataset. These objective evaluation metrics are derived from the evaluation protocol of a YOLOv2-based automatic license plate recognition system trained and tested on multiple license plate datasets, including the utilized UCSD dataset [9].

3.1. Experiment Descriptions

The experiments used to obtain the evaluation metrics for the created AMLPR software are fully contained in the evaluation script (evaluate.py). This file resembles the user-interactive program (licenseplate.py), where the main difference is all convenience features are stripped, leaving only the functionality to be measured. This evaluation script runs all the images from the UCSD license plate dataset through the program (246 images to be exact). The script stores detection accuracy, recognition accuracy, and processing time for each image and generates a summary of these metrics over all the images in the dataset at the conclusion of the program.

The detection accuracy is calculated by keeping track of the correct number of detections being made for each image (IOU greater than 0.5 with the ground-truth bounding box). For the UCSD license plate dataset, there is one license plate visible in each image; therefore, there should be one detection for each image. The recognition accuracy is calculated in two ways: relative accuracy and perfect accuracy. Relative accuracy represents the relative accuracy of the predicted license plate number to the actual license plate number (i.e., HAY7028 and HAY7018 would yield a high relative accuracy). Perfect accuracy represents the predicted license plate number perfectly matching the actual license plate number. Lastly, to determine the efficiency of the software, the processing time is measured the between input of the image to the object detection model to the output of the filtered license plate number.

3.2. Experiment Results

For the license plate location detection accuracy, the results were very pleasing and demonstrate success for this component of the software. While high performance was expected of the YOLOv3 object detection model due to the simple classification task of a single class, the produced 100% detection rate of all license plates in the UCSD dataset was very pleasing. A comparison to the YOLOv2 based license plate detector in [9] is visualized in Table 1, which performed with similar reliability in detecting 100% of license plates in the UCSD dataset, the primary benefits of the updated YOLO version are apparent in the efficiency experiments. The efficiency tests for the entire system will be discussed following the recognition accuracy results.

Table 1. System Comparison of License Plate Location Detection Accuracy

System	[9]	Proposed
Detection Accuracy	100%	100%

For the license plate number recognition accuracy, the results were acceptable and partially demonstrate success for this component of the software, but there is agreeably room for improvement. As previously discussed, the recognition accuracy was measured in two methods, the first being relative accuracy. Compared to the commercial Sighthound and OpenALPR systems, shown in Table 2, the proposed recognition accuracy falls short on performance on the UCSD license plate dataset.

Table 2. System Comparison of License Plate Number Recognition Accuracy

System	Sighthound	OpenALPR	Proposed
Recognition Accuracy	92.5%	78.6%	48.87%

Additionally, out of the 246 license plate images in the UCSD dataset, only seven were perfectly recognized (2.85%). If more time was available to further improve the performance of this component and the overall recognition accuracy, a custom OCR model would be trained on the specific font of the United States license plates (Gothic Penitentiary Fill). Another possible method that may be useful to improve this component in the AMLPR software is to perform additional edge detection and confirm the number of character edges match the number of characters extracted from the uniform blocks of text under the specified page segmentation mode.

For the overall AMLPR software efficiency, the results were very pleasing and clearly demonstrate success for this aspect of the software. Over five full runs of the AMLPR evaluation script, each making a full pass over the UCSD license plate dataset, the minimum, maximum, and average processing times were all calculated appropriately. It is important to note that results for the processing times to measure the efficiency of the program will be dependent on the specifications of the machine and other programs currently active on the machine. For the machine the evaluation script was run on (i9-8950HK and NVIDIA GTX 1050 Ti), over all 5 runs of the evaluation script, the minimum processing time for an image was 0.549 seconds, the maximum processing time for an image was 1.365 seconds, and the average processing time for an image over all 5 runs was 0.685 seconds. The detailed results of the processing times for all 5 runs of the evaluation script on the described machine are summarized in Table 3.

Table 3. AMLPR Software Processing Times

Run #	Minimum (s)	Maximum (s)	Average (s)
1	0.572	1.324	0.671
2	0.559	1.365	0.678
3	0.598	1.365	0.712
4	0.594	1.323	0.712
5	0.549	1.313	0.654

As a final overall comparison, the license plate detection and recognition system in [9] which utilized YOLOv2 for detection and CR-NET for recognition yielded processing times seven times slower than the proposed system, shown in Table 4. It is important to note that several factors may contribute to this difference in efficiency, including machine specifications and the fact that [9] included license plate layout classification in this processing time. Nonetheless, the efficiency difference of the updated object detection model highlights the machine learning advances in updated versions in addition to the overall excellent performance of the proposed AMLPR software.

Table 4. System Comparison of Average License Plate Location Detection and Recognition Efficiency

System	[9]	Proposed
Efficiency (s)	5.079	0.685

3.3. Custom Dataset Results

The custom dataset is composed of high-quality images and videos primarily containing Virginia license plates. These images and videos are taking at different distances from the license plates and with varying numbers of license plates in view of the camera to simulate a dash camera. After running the AMLPR software on each image and video in the custom dataset, there were several interesting observations and insights made concerning differences in performance relative to the UCSD dataset. Firstly, once a plate is detected, the recognition software was much more likely to perfectly predict the license plate number relative to the UCSD dataset. Additionally, for videos, the software intermittently had issues not correctly identifying the presence of duplicated license plates. Lastly, images that contained license plates with severely different formatting than those in the training data, such as yellow-gold New York license plates, often had failed detections.

4. Availability

All relevant and contributing files that were involved in the development of the AMLPR software as described in this report, including datasets, development notebooks, and more, can be found on GitHub in a public repository released under an open-source MIT license. This GitHub repository is linked at [10] and is thoroughly documented in the corresponding README. The only elements not solely contained on the GitHub repository are the trained YOLO object detection model weights. These weights are stored on Zenodo in a zip file (linked in the README) due to the weight file sizes being too large for GitHub.

5. Reproducibility

Others can reproduce my results by examining and following the guidance laid out in the development notebooks (for Jupyter Notebooks or Google Colabs), found in the "notebooks" directory of the GitHub repository. These notebooks detail the steps, thought process, and all utilized resources (all included in the GitHub repository and cited) that contributed to the development of every module in the proposed software. The training and testing datasets sourced from the UCSD license plate dataset archive are freely provided and available with appropriate license inclusion. The custom dataset comprised of images and videos with Virginia license plates are also freely provided and available with similar appropriate license inclusion. Lastly, all model parameters and performance results are fully reproducible with the provided resources, development code, and utilized references organized on the GitHub repository.

References

[1] W. Horrey, B. Tefft, and L. Arnold, "Hit-and-Run Crashes: Prevalence, Contributing Factors and Countermeasures," *AAA Foundation for Traffic Safety*, 07-May-2019. [Online]. Available: https://aaafoundation.org/hit-and-run-crashes-prevalence-contributing-factors-and-countermeasures/.

[2] K. Gullo and J. Kelley, "Automated License Plate Readers (alprs)," *Electronic Frontier Foundation*, 15-May-2020. [Online]. Available: https://www.eff.org/pages/automated-license-plate-readers-alpr.

[3] Rosebrock, "OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python," *PyImageSearch*, 05-Nov-2021. [Online]. Available: https://pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/.

[4] S. Shakhadri, "Implementation of YOLOv3: Simplified," *Analytics Vidhya*, 01-Jul-2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/implementation-of-yolov3-simplified/.

[5] J. Redmon and A. Farhadi, "Yolov3: An Incremental Improvement." [Online]. Available: https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[6] L. Dlagnekov and S. Belongie, "UCSD/Calit2 Car License Plate, Make and Model Database," *LPR and MMR Database Information*, 01-Dec-2007. [Online]. Available: http://vision.ucsd.edu/belongie-grp/research/carRec/car_data.html

[7] "Image filtering," *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html.

[8] "PyTesseract," *PyPI*. [Online]. Available: https://pypi.org/project/pytesseract/.

[9] R. Laroca, L. Zanlorensi, G. Goncalves, E. Todt, W. Schwartz, and D. Menotti, "An Efficient and Layout-Independent Automatic License Plate Recognition System Based on the YOLO Detector." [Online]. Available: https://www.inf.ufpr.br/lazjunior/papers/laroca2021efficient.pdf.

[10] D. Peterson, "ECE 4424 - Spring 2022 Project." [Online]. Available: https://github.com/dpetersonVT23/ece4424-spr22-proj