Parking with confidence using ANPR

Luis Gascon, Ethan Webb, Katherine Lim, Shannon Ioffe

August 13, 2024

Abstract

Existing smart parking systems utilized at Towson University are falling short of effective. The university would benefit economically and support the large student body of commuters by developing a smart parking system that accurately tracks parking metrics. Smart parking systems (which we shall refer to in this paper as 'SPS') address issues such as excess fuel expenditure and student tardiness. For this term project, we've set to create a REST API that returns the available parking spots for students and staff for a parking garage.

Introduction

Problem Definition

The current SPS that Towson University uses is ineffective at accurately measuring useful parking metrics. There is a need for a system that can report garage capacity data in real time so that commuter students can avoid circling around garages to look for parking spots, not knowing when parking is full. This is especially frustrating during peak times such as first week of classes and final exams.

Background and Motivation

The current implementation of a smart parking system at the Towson Town garage is ineffective at measuring relevant parking metrics that people care about. Other parking garages on campus, such as the Glen Parking Garage, have no parking system in place that keeps track of how many spaces are available. The only way for students to ensure there is parking is to drive around the garage, which can get very congested during times when classes are about to start, or around the time that classes have just ended. This is an unnecessary waste of time, as a parking system would eliminate the pointless circles around the garage since a student would know the garage is full beforehand.

Methodology

Existing Approaches to SPS

There are numerous approaches to SPS. Some employ networks of wireless motion sensors, IoT devices, or cameras, to name a few. According to a comprehensive review of SPS from ScienceDirect published in 2021, IoT-based smart parking systems are by far the most popular. These are systems that use devices connected over the internet to track parking occupancy in lots. The review also found that cameras are the most used data collection devices because motion sensors (such as IR and RFID) are sensitive to environmental changes, such as parking lots that are covered [1].

Automatic Number Plate Recognition (ANPR) is a commonly used approach in smart parking systems. ANPR collects license plate data to track parking traffic and duration. It uses a technique called Optical Character Recognition (OCR) to read license plates in from a camera and write them into digital format [2]. The most basic OCR software uses basic image processing techniques, and the most sophisticated OCR uses Neural Networks [3]. The more advanced the OCR software, the more accurate the parking system can be.

One deep learning approach is called Con-

volutional Neural Network (CNN). These are particularly good at image classification and object recognition. Because they are so good at recognizing spatial relationships, they can be incredibly useful for identifying license plates in both covered and open parking spaces. Another software approach is the Recurrent Neural Network (RNN). These are useful for handling sequential data, which makes them good at analyzing parking occupancy data in and incorporating relevant past parking data into the dataset [1].

Figure 1: *Applying gray scale*

Our Methodology

We aim to create a REST API using the Flask web micro-framework and the Flask-RESTful extension. This API will return the total count of student parking spots, enabling applications, to access this information.

Instead of analyzing a live video feed for object detection, we've decided to supply our program with a set of images containing various Maryland license plates to keep our experiment simple. The program then processes these images at random intervals to simulate vehicles entering and exiting. All image pre-processing is handled using the OpenCV Python library. We also have the option to choose from three different OCR engines: EasyOCR, Keras, and Tesseract.

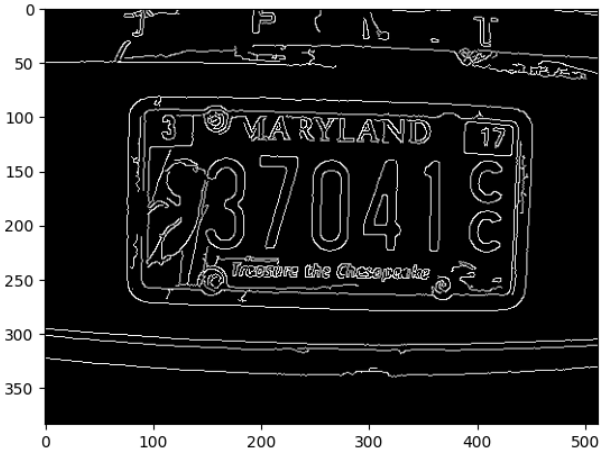

Figure 2: *Detecting edges*

Figure 3: *Cropped image after finding contours*

Image Preprocessing Color adds unnecessary complexity to the image, which in turn, introduces extra computational strain when processing images. We apply gray scale to remove colors. After applying gray scale, we reduce any image noise by applying a Gaussian blur to the image. Using a canny edge detector will allow assist in finding contour points of the license plate. The contour points are then sorted in ascending order and we filter through all the contour points by approximating whether or not a contour has 4 points, which indicates the 4 corners of the license plate.

We then use these contour points to crop in on the license plate to only focus on the license plate numbers. These pre-processing steps ensure the image is optimized for the next phase. Finally, we use Optical Character Recognition (OCR) to extract and detect text from the license plate image.

Overview of OCR Technologies

EasyOCR EasyOCR is a module in python that has the ability to find and extract text from photographs or from scanned documents. This can be used for analyzing images or for automated data entry. It is easy to implement since it simplifies the process of gathering the text from images, works very efficiently, and can handle a variety of typefaces and text layouts [4]. There is three main parts to EasyOCR: feature extraction, sequence labeling, decoding.

Feature extraction is accomplished though Resnet (Residual Networks), which are a type of CNN (convolutional neural network). This makes the network trainable by resolving the problem of vanishing gradients while still having a good depth. VGG (Visual Geometry Group) which is another type of CNN is also involved feature extraction and VGG decreases the number of parameters and computation [4].

After feature extraction is sequence labeling which uses an LSTM. Decoding is next after sequence labeling and is accomplished by a loss function CTC (Connectionist Temporal Classification). In handwriting or in speech recognition there can be some uncertainty between the input data and the labels. CTC places a blank label with the existing labels [4]. Additionally, EasyOCR supports over 80 different languages. EasyOCR can be used for license plate recognition but works best for clear pictures of license plates.

Keras KerasOCR and TensorFlow can be used to detect the digits 0-9 and letters A-Z from images. The MINST database is already built into Keras and TensorFlow, but to recognize the letters A-Z the NIST Special Database 19 must be used. Keras and Tensor flow are used train the OCR model. In license plate recognition, a ResNet implementation of PyImageSearch is also used. Keras performs better than both EasyOCR and Tesseract when examining low quality images [4].

Tesseract Tesseract can also be used to extract text from images. This model uses the line finding algorithm which involve blob filtering and line construction that allows a skewed page to be recognized.

After the lines have been fit from the line finding algorithm the baselines are better fit using a technique called quadratic spline. Tesseract aims to separate the characters in the words first if there is a fixed amount of space between each character. This gets more difficult however, if there is varied spacing and if the characters are joined. If this is the case, then the word is reevaluated after the word recognition phase [5].

When a word has an unsatisfactory evaluation, the parts with the least confidence are chopped based on candidate points. If the result is still unsatisfactory after the chops, then the associator sorts through possible combinations of chopped blobs to form candidate characters.

To determine what the character may be, the class pruner creates a shortlist of character classes that the unknown might belong to. The unknown characters are compared to prototype classes and their similarity is calculated to yield the result. The classifier was not trained on damaged characters. [5]

Comparison Out of all three OCRs explored, we decided to implement EasyOCR because its a lightweight framework with enough features for this type of experiment. However, it is important to recognize the strengths of the other OCRs. Keras seems to perform the best on lower quality images compared to EasyOCR and Tesseract. Keras can be useful if the image has inconsistent fonts or colors. Additionally it can read all image file types (.jpeg,.png, .gif, .bmp, .tiff) [4].

Challenges Most of the Maryland license plates that we used for testing contained extra text such as the state name and a website URL. There's also the issue with vertical text since EasyOCR doesn't read vertically aligned text.

Figure 4: *Maryland License Plate with a lot of text*

To overcome these obstacles, we've decided to rely upon fuzzy string matching with the help of a Python library called the thefuzz. A fuzziness of a string is determined by its Levenshtein distance, or the difference between two strings. It is used to identify the closest correct word to a misspelled input, and in our case, the misspelled input is what EasyOCR returns.

Experiments

We created a SQLite database containing the following entries:

licensePlate	isStudent
2AC7692	0
3AR9282	1
36815CE	1
37041CC	0
40741CB	1
WU TANG	0

Our program reads a series of images in random order, perform pre-processing and it then feeds it to EasyOCR. If the approximate contours cannot be found, we instantly discard that image and consider it invalid. EasyOCR returns a list of strings due to the many different text present on a license plate.

`thefuzz` approximately finds a match for each string from the list that EasyOCR returns and all the license plate numbers in the database. If a match is found, we check whether the approximately matched database entry is a student or not.

Results

Setting the ratio threshold at 70% netted accurate results. Visiting the URL endpoint of our API showed the number of parking spots available Unfortunately, some images were trickier to work with as our program fails to pinpoint the contour points of the license plate, despite collecting all contours.

Table 1: *Fuzziness Similarity*

Text Comparison		
Text from OCR	Database entry	Similarity Ratio
27692	2AC7692	83%
27692	37041CC	17%
MARYLAND	3AR9282	27%
389282	3AR9282	77%

Conclusion

Within the confines of our experiment, we were able to accurately determine which license plates being read belonged to a student. There is no doubt in my mind that this program isn't optimized enough to be run with a low-powered device, such as a Rasperry Pi. In an attempt to detect as much contours as much as possible, our program doesn't sort by a top n number of contours. Our program doesn't scale well either as comparing each strings obtained by EasyOCR by each database entry has the worst case runtime of $O(n^2)$. We could improve upon that in the future by rejecting common string patterns such as "Maryland".

References

[1] H. Fahim A. Mehedi and A. Muhtasim. "Smart parking systems: comprehensive review based on various aspects". In: (2021).

[2] M. Du S. Ibrahim and M. Shehata. "IEEE Transactions on Circuits and Systems for Video Technology". In: *IEEE Xplore* (2013), pp. 311–325.

[3] A. Rosebrock. *Pyimagesearch*. `https://pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/`. [Accessed 10-05-2024]. Sept. 2020.

[4] Thanga Sami. *Tesseract vs Keras-OCR vs EasyOCR — thangasami.medium.com*. `https://thangasami.medium.com/tesseract-vs-keras-ocr-vs-easyocr-ec8500b9455b`. [Accessed 29-04-2024].

[5] R. Smith. "An Overview of the Tesseract OCR Engine". In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 2. 2007, pp. 629–633. DOI: `10.1109/ICDAR.2007.4376991`.