MSA 8650 Computer Vision Project Report

Serkan Comu, Chris Lee, Jiahui Li

October 7, 2020

Motivation

We want to explore multiple Optical Character Recognition (OCR) builds and systems to extract the text, Vehicle Registration Numbers (VRNs), from images of license plates.

Data

The Dataset contains 1821 images, captured by a commercial license plate reader.

- Context Images: Images of size 800x600 pixels depicting the vehicle and its surroundings.
- IR_Patch Images: Images of size 280x80 pixels depicting the cropped license plate.
- XML File: File that contains the metadata which contains the "true" VRNs.

Methods

In this project, we used three approaches to the read the license plate: AWS Rekognition, OpenCV and Tesseract, and with a pre-trained Deep Neural Network (DNN), LPRNet. In all three methods, we used Levenshtein's Distance to determine and evaluate the accuracy. Levenshtein's Distance measures the difference between two strings/sequences. In our case, we used Levenshtein's Distance on the "True" value of the VRNs, provided from the metadata, and on the predicted VRNs, provided from our methods.

- **AWS Rekognition**: A cloud service from Amazon to identify objects, people, text, and etc. from images and videos. We followed the code provided to set up AWS account, create S3 bucket, load images and perform the image analysis with AWS Rekognition.

- **OpenCV and Tesseract**: OpenCV is an open source computer vision library, which includes both classic and state-of-computer vision algorithms. Tesseract is an Optical Character Recognition engine (OCR). We used the OpenCV to process the image and applied Tesseract to recognize text. Since we are trying to predict American License plates, we configured the OCR to read the images as a single line text with a whitelist of alphanumeric values only.

1. We first applied Tesseract's OCR image to string function on the original cropped images in the IR_Patch folder. This process gave insight on what Tesseract was able to or not able to detect perfectly with no image enhancements. This first process gave a decent number of exact matches, but about a third of the license plates were read completely wrong. Average accuracy was ~32.93%.

2. The original images contained a lot more than needed. The next process, we created a textspotter with the open source files "textbox.prototxt" and "TextBoxes_icdar.caffemodel". We first inverted the image to make contouring easier. We then used canny function to detect the edges and the contour function to find and draw the max contour of the contour with the greatest area. We had difficulty in this part to extract the most useful top seven contours. We then used the textspotter on the images to crop the cropped image so that we pass less unnecessary image portions through Tesseract. After cropping the images again, we passed it through Tesseract and

received a higher average accuracy of 39.29%. This second process, we got a lot less exact matches, but also a lot less completely wrong matches.

3. On the final process, we a lot of different image enhancements approaches on the newly cropped images from process 2. We played around with morphological transformations like erosion and dilation, but those didn't improve accuracy. We tried reducing noise with blurring with a normalized box filter, a gaussian filter, a median filter, and a bilateral filter. The bilateral filter took the longest and the gaussian filter provided the best accuracy. We then made sure the images were in grayscale then converted the images to binary images using global thresholding, adaptive thresholding, and otsu's thresholding. Global thresholding lowered our accuracy, most probably because a lot of our images had different lighting conditions. Adaptive Thresholding gave us an improved accuracy of about ~47.52%.

- **LPRNet**: An end-to-end method for automatic license plate recognition without preliminary character segmentation, and with high performance (96%) on Chinese license plates.

For the LPRNet approach, we mostly used the code that was provided. After some debugging of the code to make it work with our data, we started the program running. First, we read the images and their metadata into python, then we ran the images through 40 different layers in our backbone to get some outputs. Afterwards, we used four of these outputs and ran them though a separate convolutional network to get our results. These results are then recorded and compared to the original metadata to get test accuracy results.

Our model gets all its weights from sirius-ai's LPRNet approach and therefore has biases that we may not want. The model comes pre-trained on images that are usually colored and always have a uniform convention: a Chinese character first followed by a letter and a dot. Our data does not have these characteristics; however, we thought it would be interesting to see how a neural network trained on one thing would behave when subjected to a very similar object but slightly different.

Because of these biases, we had an accuracy of 42 percent. Most of our results were off by 2, 3 and 4 because most predictions omitted the first character and replaced it with a Chinese character leading to a 1 distance right off the bat.

This experiment shows that deep neural networks are very training dependent. It is not reasonable to "buy" a deep neural network solution and expect it to work on your specific test case. To achieve a 96 percent testing accuracy the original creator of the script used 1.7M training images to train their network. This points to the most important side of deep learning. It is theoretically the best solution to this image problem. However, the neural network needs to be trained with images that have the same context. If we had 200k images taken from the same exact camera with all American license plates we would have had much better accuracy. However, 200k images are not cheap and the time to train such a neural net is very long.

Challenges with the Data and Processes

1) Most license plates that were falsely recognized came from photos with uneven lighting. A majority of those photos were taken during the hours of sunrise (between 6am and 9am). They bring color distortion of the features, and consequently leads to false detection.

2) Amongst the misread license plates, the images also included those with a lot of obstruction, such as truck towing a car. Orientation and skewness are problems with images as well. Read images as a single line to extract text.

3) Some characters having similar shapes are indistinguishable, especially when the license plates are blurring and degradation. For example, digit 0 and letter O, digit 1 and letter I, letter M and letter W, letter M and letter H, letter Q and O, letter D and O, etc.

4) Low quality images provided, which are not readable by the human eye.

5) Some license plates images contain irrelevant content that caused our predictions to predict more than 7 characters. A challenge was depicting and drawing the 7 largest contours to increase accuracy.

Accuracy Comparison

Methods	Average Accuracy	Exact Match	Mode (based on Distance)	Standard Deviation (based on Distance)
OpenCV Tesseract	**47.52%**	**7.80%**	**2**	**2.27**
Amazon Rekognition	**88.33%**	**61.34%**	**0**	**1.45**
DNN-LPRNet pre trained	**42.65%**	**1.2%**	**3**	**1.73**

Discussion

AWS Rekognition

Pros:

- No ML/DL experience is required. You just have to provide the images, and the services will detect the text from them.
- No labels are needed.
- Faster to train.
- Achieve the highest performance among three methods.
- Low cost to time spent ratio.

Cons:

- No parameters can be modified.
- Confidence Score is not that helpful to filter the correct predictions.
- Predicted duplicate labels sometime.

OpenCV and Tesseract

Pros:

- Wide range of Image Enhancing techniques and solutions.
- Building Training Set is extremely easy.
- Open source and very customizable.

Cons:

- Works best with grayscale images.
- Images needs to be self or preprocessed to improve performance.

Deep Learning Neural Network

Pros:

- Most customizable.
- Can be made to your exact specification.
- Fast after training.
- Feasibly the highest performance.

Cons:

- Takes a long time and a lot of data to train.
- Or uses a pretrained neural network that is not your exact situation.
- Requires the most coding and math knowledge to get working.

Use Cases:

AWS:

Quick and simple to setup; AWS is useful as a permanent and expensive middling solution to most problems. AWS can also be used on a time sensitive project to give teams setting up the Deep learning net more time and data, to later replace AWS with a long-term cheaper approach.

OpenCV and Tesseract:

OpenCV and Tesseract are top notch open source libraries. They do not have much notable disadvantages, but the main disadvantage we have discovered during this project is that it takes a long time to run so it will not be useful in a real time solution, but useful for a non-billable project

Deep Learning Neural Network:

DLNN's need to be completely custom to be more accurate than AWS, so it is mostly useful in projects that are not time sensitive. Since training a neural network by definition, is telling the neural network what to look for, pretrained networks are likely to have many biases that your data does not share with them.