CSCI 1430 Final Project Report: License Plate Detector

Anonymous CVPR submission

Paper ID ****

1. Introduction

Our project aims to build a computer vision program that takes in images of cars with different backgrounds and orientations, detects (and outlines) the license plate, and reads its number if visible. Our main tasks were building a license plate detector using edge detection and feature extraction, then training a neural network model to read the plate input.

This task is challenging because of the nature of the kinds of images that can be provided. License plates have a variety of formats across different areas and countries, and coupling this with the fact that cars can further be in different orientations or visibility means our approach needs to accommodate a lot of variation in input. Our data comes from datasets with images of cars from the US, Europe, and Brazil, with annotated license plates.

Our project could have an important application in traffic systems. For instance, many toll systems could be improved by simply using CV to identify license plates rather than stopping each car. Furthermore, this has applications to law enforcement and vehicle management by the government.

2. Related Work

We used some online references and OpenCV tutorials for our work on edge detection and character segmentation. Our datasets were from prior studies on license plates. The neural network was developed without reference to external works.

3. Method

Our approach and code involved several components: license plate detection, license plate digit segmentation, and a forward pass through a neural network.

3.1. Datasets

Various forms of data were needed to gather the necessary information on license plates. There were three main datasets. The first dataset was pictures of cars with an arbitrary background, which was used for testing the bounding boxes our detector returns. The second was pictures of cropped license plates that were used for testing our segmentation algorithm that divided (i.e. segmented) a license plate picture into single characters. Once the segmentation script was complete, we used it to save many cropped images of single characters from license plates. These were used to train our neural network for character classification and can be considered our third dataset.

3.2. Plate Detector

Our approach utilizes several image filters, contours from OpenCV, and our segmentation algorithm to eliminate false positives. First, given an image of a car with an arbitrary background and visible license plate, we convert the image to gray scale, apply a Gaussian blur, and use a vertical Sobel filter to highlight vertical edges. We then use OpenCV's findContour method to find possible bounding boxes where all pixels on the border of a bounding box have nearly the same intensity. This gives us many possible bounding boxes, the cropped license plate being one of them. We then filter these boxes by some threshold of area and aspect ratio. Since license plates are typically white in background, and some other color for the text, we further filter these returned contours by the ratio of white pixels to total pixels in that contour. We use a bound of 0.45 as our second to last filter. Finally, for each of these last possible bounding boxes, we pass each to our segmentation algorithm that decomposes each character of a license plate into a cropped image with just that character. If the number of cropped character images returned by the segmentation algorithm is between three and eight inclusive, the majority of the time that is the true license plate. All other bounding boxes are eliminated.

3.3. Character Segmentation

Initially, we approached the task by using a neural network on the entire plates, but this produced bad results. This made sense, because the task was quite complex given the variance of license plates in our dataset. We then tried implementing segmentation of plates into characters to simplify

the overall task of getting the plate number to just the task of identifying one character at a time.

To do this, we first cleaned the images to a more easily processed version. We made the plate image greyscale, resized and applied a Gaussian blur, and equalized the histogram. By then applying a threshold, we got our cleaned image. The main algorithm involved applying a bitwise not on the cleaned image, finding the contours, and going through the contours to process each. On each, we found the bounding rectangle, found the area of the bounding box, and depending on whether the area was too small or large, we dropped the contour as needed with thresholds of 0.008 and 0.032 respectively relative to our entire image area. Otherwise, we drew a filled rectangle corresponding to the bounding box on the mask with white colour and stored the bounding box/image. The following code highlights our approach for each contour:

```
x,y,w,h = cv2.boundingRect(contour)
area = cv2.contourArea(contour)
center = (x + w/2, y + h/2)
if (float(area/img_area) > 0.008) and /
    (float(area/img_area) < 0.032) and h > w:
    x,y,w,h = x-4, y-4, w+8, h+8
    bounding_boxes.append((center, (x,y,w,h)))
    cv2.rectangle(char_mask,(x,y), /
        (x+w,y+h),255,-1)
```

This approach to image character segmentation allowed us to tackle the task of recognizing each character for each separated bounding box and image using our neural network with higher reliability than the initial approach of just one neural net.

3.4. Neural Network

Once the characters of a license plate are segmented into a cropped 100 by 50 pixel grayscale image, we pass them to a neural network to predict what digit or letter the character is. We used Tensorflow to create our model, which takes in a 4D tensor representing a batch of images that are 100 by 50 pixels with a single color channel and returns a probability distribution over 36 possible characters (10 digits and 26 letters) for each image in the batch. Our model architecture is as follows:

- Convolution layer with 32 filters of size 2, a stride of 1, same padding, and ReLU activation
- Maxpooling layer of 2 by 2
- Flattening layer
- Dense layer of size 512 with ReLU activation
- Dropout layer of 0.5
- Dense layer of size 36 with softmax activation

We train our model for 6 epochs, batch size of 32, a learning rate of 0.001, Adam optimizer, and SparseCategoricalCrossentropy loss.

4. Results

4.1. Plate Detector

Without any discrete labels of correct bounding boxes, we were not able to numerically quantify the accuracy of our detector. However, from user testing, our algorithm seems fairly reliable. Figure 1 contains one example of the output before and after filtering. The left image contains the actual license plate and a returned bounding box but several nonsensical boxes as well. After passing each of these bounding boxes to our segmentation algorithm, we only maintain the bounding boxes that return three to eight segmented characters. In this case, only the license plate holds, so it is the only bounding box returned. We then crop this bounding box and pass its segmented characters to our neural network to read a prediction. However, one issue we ran into is that the cropped image returned by the detector is much lower in quality. This may lead to incorrect readings. For instance, the returned license plate returned in Figure 1 is actually YG9X2G but is read by our network as Y09X20. The two G characters are replaced with zeros likely because of image quality loss as well as the fact that our train dataset contained many more digits than letters. This necessary filtering and image quality loss makes the detector is the least reliable out of the three main steps of our project.

4.2. Character Segmentation

Similar to the detector, we did not have any labeled data of correct bounding boxes for segmentation, so we had to evaluate the segmentation algorithm by hand for the most part. This consisted of tests on many different cropped license plates and saving the segmented results to disc and examining each segmented image for reliability. An example of this decomposition can be seen in Figure 2, where we have the bounding boxes returned for each character for a single license plate and the first nine of that same license plate cropped and filtered as it will be passed into our neural network.

4.3. Neural Network

With a train set of size about 3000 and test set of about 300, we were able to get fairly decent results among the 36 possible character classes. We trained for our model for 6 epochs and our plot of loss overtime is shown in Figure 3. As can be shown, our model converges. For each epoch, we printed the current model's accuracy on the test set. These results are shown below. We were able to reach a peak accuracy of 78.4% on the test set. Epoch results:

```
EPOCH 0 | ACCURACY: 0.75769
EPOCH 1 | ACCURACY: 0.78461
EPOCH 2 | ACCURACY: 0.71923
EPOCH 3 | ACCURACY: 0.69999
EPOCH 4 | ACCURACY: 0.73076
EPOCH 5 | ACCURACY: 0.76538
```

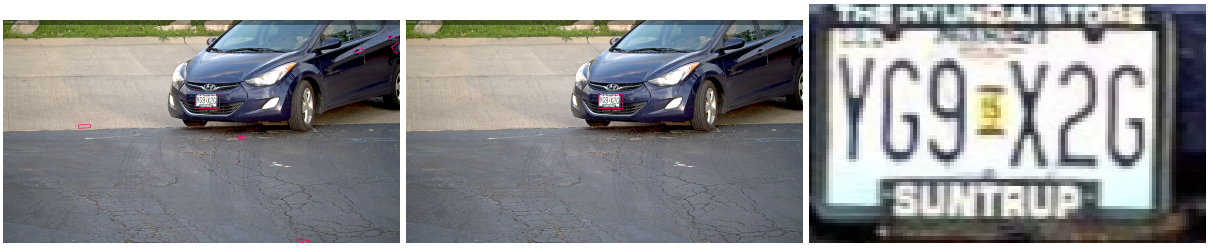

Figure 1. *Left:* Boxes returned before segmentation filtering. *Center:* After segmentation filtering. *Right:* Lower quality cropped plate

Figure 2. *Left:* Bounding boxes for characters. *Right:* Cropped and filtered first character.

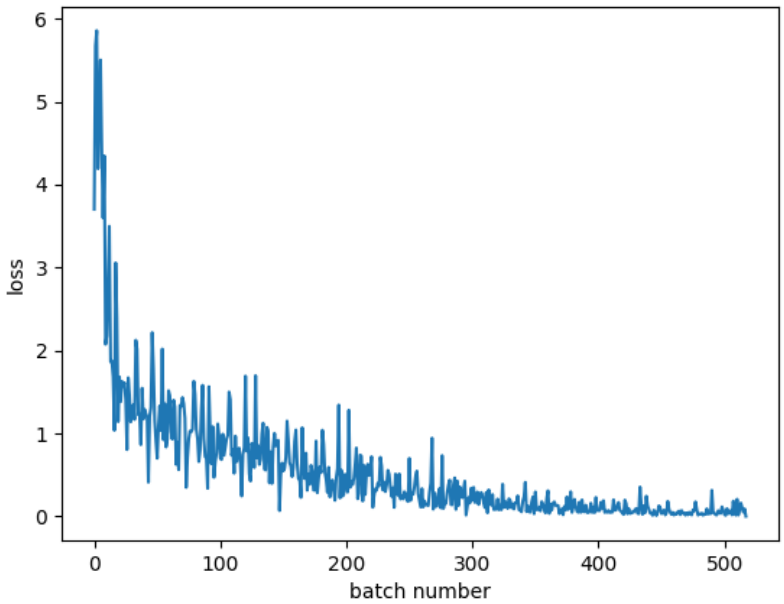

Figure 3. Graph of loss overtime

4.4. Discussion

Some of the components of our method had a variety of approaches that can be discussed. One is the setup of segmenting the license plates into their respective characters, rather than looking at the license plate as a whole, which was our initial approach. One trade-off was the speed and power needed for our program; processing each image in this way used more resources than the first method, and made it take longer to run and more difficult to ensure correctness. However, the benefit seemed to outweigh the cost, since doing this helped our model's accuracy a great deal by simplifying each individual task, despite there being more tasks overall (since there could be up to 8 image recognition values per plate). Our preprocess was also a big part of this improved accuracy, as segmentation allowed for each character to be extracted with more image clarity as opposed to entering an entire license plate image in which there are many variations throughout examples. We can think of this question as a choice between correctness and efficiency, which is a common issue in Deep Learning tasks.

Another part of our method that raised an interesting question with regards to the results was our model. It is difficult to determine a level that real applications of license plate detection would require to be considered reliable. We were able to improve our model through experimentation to high levels for an initial project, with accuracy across epochs upwards of 75%. With additional resources and more extensive datasets, it could be reasonable to aim for much higher accuracy so a model can not only read the characters correctly but the entire license plate. Our model was also designed with a tradeoff between efficiency and correctness. For the purposes of this project, we aimed to have a relatively simple model that could be trained quickly on a laptop. We did not want to overcomplicate it with additional parameters beyond what was needed to give good accuracy on the character model. In Deep Learning, this decision of simplifying a model versus adding potentially helpful layers is present in almost every task.

5. Conclusion

Thus, our project aimed to process and read license plates in a way that was very applicable to many real-world issues. By using edge detection, image processing, and a deep learning model, we were able to tackle the task of automating human license plate readings with a computer vision program.

This kind of program can prove very valuable moving forward, especially when fine tuned to high accuracy. The project demonstrates potential to add automation to traffic systems and law enforcement by giving humans a reliable way to quickly identify license plates involved in road incidents. This task has the potential to reduce much manual human effort in license plate management by simply requiring an image of the car to get the necessary information.

Appendix

Team contributions

Prithu Dasgupta Contributed to the plate detection, model fine-tuning, and neural network setup.

Daniel Park Contributed to the initial model setup, data gathering and preprocessing, and report.

Gokul Ajith Contributed to the data preprocessing, character segmentation, model training, and command-line processing.

Spencer Greene Contributed to the segmented dataset, character segmentation, and model optimization.