Low-cost Automated Parking Ticket System Using Computer Vision

Alejandrina Gonzalez
Computer Science
Stanford University
Palo Alto, USA
alegre@stanford.edu

Anand Lalwani
Electrical Engineering
Stanford Unviersity
Palo Alto, USA
anandl@stanford.edu

***Abstract*— We propose using a raspberry pi with camera and basic computer vision to automatically detect, recognize and characterize a license plate as it enters a parking garage. The plate can then be cross-referenced with an existing repository of valid parking permits, and if found in violation, can be ticketed accordingly.**

I. Introduction

Parking, especially at Stanford, is a rare resource. To help ease the burden of finding parking, heavy fines are currently imposed on violators - i.e. cars parked without a permit. However, finding the cars parked without a permit is a cumbersome and time consuming process that is both costly and not very effective. We propose a low cost solution (under 100$) that can automatically detect when a car enters the parking garage without a permit and notify the ticketing agency accordingly - thus aiding in the efforts in making sure parking is available for permitted users.

II. Description of Tasks

The overall project is broken down into 4 major aspects. Based on Ngyuen (2020) our project has the following structure:

1. Detect a car's license plate
2. Segmentation of characters on the license plate into individual characters
3. Neural network using Keras and softmax output to predict the characters and output a string
4. Integrating the above parts into one code that can run on a Raspberry Pi and test with real world cars
5. Cross referencing with Stanford transportation for access to updated sheet of permits

Each task can be done independently and we have completed all except for the last part. We have focused our deep learning exercise to task 3, and adopted existing models for task 1. We have also purchased all the components needed and integrated with Raspberry Pi 4 and OrCam Camera.

III. Datasets

The datasets used can be broken into three major categories, as needed by Task 1, Task 2 and Task 3.

A. Dataset for Task 1

For task 1, we obtained a small dataset of ~25 vehicle images from various countries. The figure input size is (12, 8) with 5 columns and 4 rows. All images are resized to width=224 and height=224 to keep our input vectors the same size. Each image is labelled with x,y,w,h for the bounding box identifying the license plate. Figure 1 is an example of the data.

Figure 1: example of image for Task 1 and Task 2. Image taken outside EVGR, Stanford, CA

B. Dataset for Task 2

For task 2, the dataset from task 1 along with OpenCV's inbuilt functions were used and therefore no new datasets were needed. Specifically, functions to identify box individual characters on the license plate were used.

Figure 2: final output of Task 2 from figure 1

C. *Dataset for Task 3*

For task 3, we have nearly 37,000 labelled images from 36 classes (36 characters). We further increased the dataset using data augmentation techniques that allowed to make the dataset more robust. Techniques such as zoom, blurring, rotation and shearing were used – following guidelines introduced in class. OpenCV's functions were used for data-augmentation

IV. Discussion

Task 1: Detect a car's license plate

The first task is to detect and draw a bounding box around the license plate identified in the image. For the task, we use a pre-trained W-POD model by Silva (2018) that has been trained on various countries' license plates. The method proposed by Silva is not only from various regions of the world but also accounts for photos taken from various angles, as is needed for a parking camera. Other popular networks that were considered was the YOLO neural network, however, the broader applicability of Silva's was more useful for our specific task.

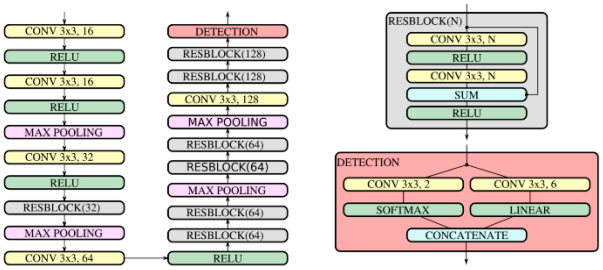

Figure 3: Architecture of WPOD-Net (Silva 2018).

Task 2: Segmentation of characters from license plate

To identify each character on the license plate before predicting it, we decided to use OpenCV's findContours function to draw bounding boxes across each individual character on the license plate. To aid findContours, we utilized image processing techniques that augment the image so it is easier to detect edges. Techniques used are - converting to an 8 bit scale (instead of 0/1), converting to a grayscale rather than color, and gaussian blurring to remove unwanted noise and detail. These techniques are inbuilt functions within OpenCV and thus do not require extensive codebases.

Task 3: Predicting characters using trained neural network

Majority of the work for the project focused on Task 3. To deploy our model on the 'edge', the model has to be relatively low-weight and quick to output – as required by the usecase. To achieve this, we start with the MobileNets (Howard 2017) model as our reference as it is low-weight, low-memory and low-power overall. The model was modified and trained on the dataset described above. The dataset was split 90-10, with 90% used for training set and 10% used for validation.

Significant amount of pre-processing of the data was required before training. From standardizing (normalizing) the input data to 80x80 to data augmentation to creating the labels to one-hot encoding labels. The code submitted includes all data preprocessing steps undertaken.

Transfer learning

We load the MobileNets architecture as is using Keras and replace the final layer with our own softmax output with 36 nodes (corresponding to the 36 classes/characters). The initial weights used were from the ImageNet class.

Model features used:

- Dropout: 0.5
- AveragePooling2D – size (3,3)
- 128 nodes used for Densor
- In between activation function: RELU
- Optimizer: ADAM
- Epochs: 30
- Learning rate: 0.0001
- Early stopping to prevent overfitting
- Loss function: categorical_crossentropy
- Final activation: Softmax
- Batch Size: 64

A common issue we ran into early on was dimensions. Our input images were not the same dimensions as required by the model, leading to considerable number of bugs. Another issue was computational resources – since we trained on Google Collab – a free limited GPU resource provided by Google, we had to do Task 3 on a different independent account to reset the computational budget provided by Google.

Evaluating the accuracy on validation set and training set:

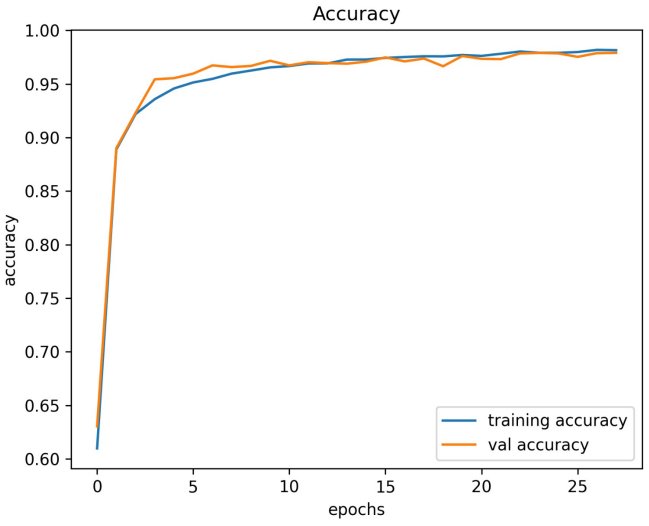

Figure 4: Accuracy for training set and validation set

As we can see the accuracy drastically increases from the first epoch to 10th epoch (up to 95%) and then plateaus near 97% accuracy on the training and validation set. The validation set overshoots the training accuracy in the earlier epochs (and in the previous runs, in the later epochs as well) indicating potential over-fitting. Further over-fitting was reduced by increasing the dropout layer from 0.3 to 0.5. The final accuracy at epoch 28 is 98.3% and 97.9% for training and validation, respectively. Note, we stopped at epoch 28 than epoch 30 as stipulated by the early stopping function.

The other metric we tracked is the loss as a function of epochs. The sharp drop in the loss function in the first few epochs with a gradual decrease after is characteristic of cross-entropy and has been witnessed across the class over various models. Similar to accuracy, overfitting was a concern earlier which was later fixed by increasing the drop out layer. The final loss at epoch 28 is 0.0512 and 0.0765 for training and validation loss respectively.

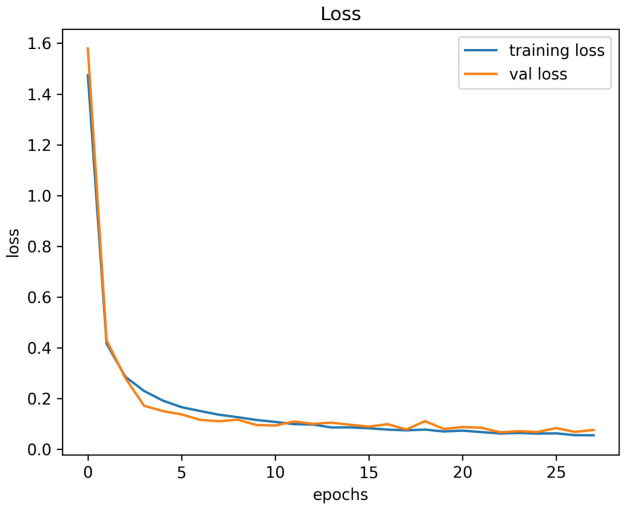

Figure 5: loss for training set and validation set

In summary, the newly trained MobileNets model works well to classify and predict the 36 classes of characters from the training set.

Figure 6: final string output prediction of license plate in figure 2.

Task 4: Hardware implementation

A Raspberry Pi 4 (Raspberry Pi, UK) was purchased along with an OrCam (China) 5 MP Pi compatible camera. Cana Kit (OR, USA) provided the power source and 64 GB SD card with NOOBs operating system preloaded. Further installation of libraries such as Tensor Flow, SCIPY, Keras etc. was done as well. Finally, registering the MAC address of the Raspberry Pi on Stanford Network was required for the Pi to access wifi across Stanford's internal wifi network

Raspberry Pi was chosen as it is low cost (total parts <$100), versatile and can run light weight ML algorithms on it's micro-processor. The camera purchased has autofocus and works well in well-lit conditions.

The trained model from task 3 along with further code to take the image, store locally, email final string output of predicted license code and delete image from local storage was added to the code base. The python notebook was then run locally on the pi and tested across various vehicles at Stanford's Thoborun Garage. A video demonstration of the same is provided along with the following images.

V. Conclusion

Task 5: Cross-referencing with Stanford Transportation's database

Unfortunately, due to privacy concerns by Stanford Transportation, we were not provided with access to Transportation's database of registered vehicles. Instead, we propose the following architecture to better integrate with Stanford's privacy:

1) Vehicle enters garage/parking lot
2) Camera takes image of car
3) The model proposed in this paper predicts the license plate of the car
4) The Raspberry Pi in real time emails the Stanford Transportation department the license plate of the car
5) The transportation department can run an internal script checking if the license plate of the car is registered in its data base. If it is not, it can notify parking enforcer of a potential infrigement.

We are currently in talks with the Transportation Department for a potential demonstration and pilot.

VI. Future work

Future work involves integrating with Stanford Transportation department, broadening the dataset of license plates to be more robust to moving images, creating a program to capture only when cars are in motion and finally a way to track number of vehicles in the parking garage and total number of spots available – thus allowing users to better understand ahead of time if there are any spots present in the parking garage.

VII. REFERNCES

[1] Silva, S. M., & Jung, C. R. (2018). License plate detection and recognition in unconstrained scenarios. In Proceedings of the European conference on computer vision (ECCV) (pp. 580-596).

[2] Ngyuen, Q. (2020): Detect and Recognize Vehicle's License Plate with Machine Learning and Python