ENPH 353 Team 9 Final Report

Sarvan Gill
Anthony Ho

October - December 2019

1 Introduction

The competition takes place in a simulated parking lot environment. This parking lot has 2 loops, and inner and an outer loop, 2 crosswalks with pedestrians crossing at any given time, and a truck driving around the inner loop.

The objective of the competition is to control a robot that goes around this simulated parking lot, and checks if the parked cars have permission to park by reporting all the license plates and spot numbers of the cars. The robot is to do this without hitting the pedestrians or the truck and staying within the lanes, failure to do any of these resulted in point deduction.

Our strategy was as follows: The robot is to move around the outer loop twice, this way we can be sure that we get all of the outer plates, then if time permits, we will go into the inner loop, capture the closest license plate and then exit and stop our run at the next crosswalk so we do not get hit by the truck.

2 Data Processing Architecture

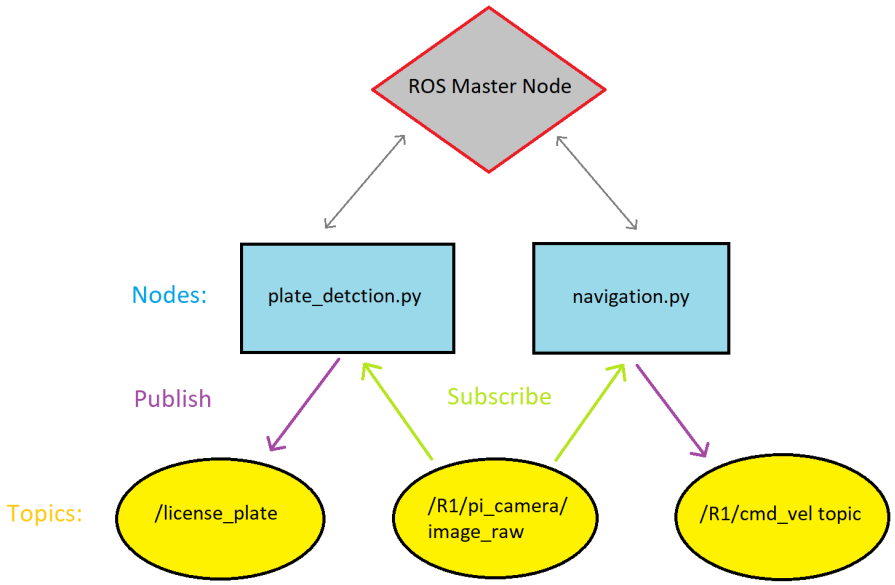


Figure 1: High level data processing architecture

One of the things that we knew we wanted to do was to separate the navigation and licence plate detection. The reason we wanted to separate these two actions is because we wanted to be able to continuously move and take pictures at the same time. If we had a script that tried to do both, without multithreading we would've needed to stop to take a picture of the license plate and then continue our navigation, this would be too time consuming.

So in the end we created two ROS nodes, as shown in Figure 1, one for navigation and one for plate detection.

For both nodes, we made use of computer vision with OpenCV to process the images received from the `/R1/pi_camera/image_raw` topic. In the plate detection we also used a convolutional neural network (CNN)

modelled with Keras to help process and read the images.

3 Navigation

As mentioned in the strategy, our plan was to go around the outer loop twice and then go inside the inner loop to get a single plate and stop at the next closest crosswalk. We saw this as 3 sub tasks: driving along the desired path, detecting crosswalks/pedestrians, and avoiding the truck.

3.1 Driving

The driving and lane following are easily split into two sections, the general lane following technique used on the outer loop, and a modified version used on the inner loop.

3.1.1 General Lane Following

As aforementioned, we used PID control to stay within the lane. In the simplest scenario (driving around the outside loop), this meant following the outside. Through experimentation, we found a value for the position of the center of the outside line, that put our robot in the center of the lane, this position is shown by the blue dot in the bottom right corners of Figure 2.

We then calculated the center and the width of the right line, by scanning from right to left at the desired height for white pixels. Once we found our first white pixel, we looked for the next black pixel, and from there it is trivial to find the center of the line and the width. The center is shown by the green dot in the bottom right corner of Figure 2.

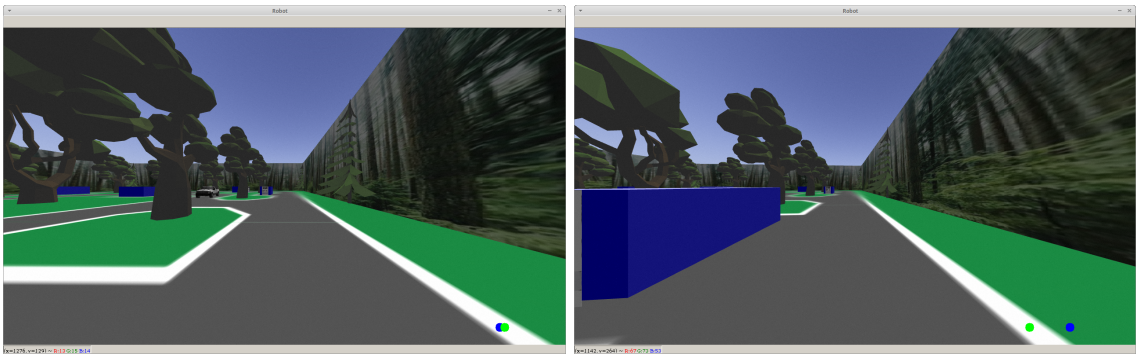


Figure 2: Centered Robot (Left), Off Centered Robot (Right)

The PID worked as follows. If the set point was within 10 pixels of the line, we tell the robot to go straight. Other wise if the set point was to the left of the line, we tell the robot to turn right and if it was to the right of the line we turned left. In the right picture of Figure 2, although the robot is looking straight down the lane, it is hugging the right line too closely so we turn left towards the line to center ourselves in the middle of the lane.

3.1.2 Inner Loop

By counting the number of crosswalks that we passed, we were able to determine how many laps we had completed. So, once we had crossed the 4^{th} cross walk we start to move into the inner loop. Moving into the inner loop revolved around counting gaps in the left hand side of the lane. Once we crossed the cross

walk, we line followed as described in Section 1.1.1, however we also looked for a gap in the left hand side of the lane, the gap occurs from the blue car blocking the white line, this can be seen in Figure 2.

After we passed this gap, we switched from following the outside line of the line to the inside or left side of the lane This took us around and into the inner loop. So shortly after the left frame shown in Figure 2, we would follow the left line and turn into into the inner loop. Once we saw the first license plate, following the left line also took us back out of the inner loop to stop the robot out of the trucks way.

3.2 Pedestrian Detection

Our method for detecting the pedestrian was to stop at the cross walk and continue when the pedestrian was not crossing. As the crosswalks were clearly indicated by a vibrant red, we simply looked for the color and once we saw it at a certain distance we stopped. We scan the area shown for red in the blue rectangle in Figure 3. After we decide that it is clear to go, we don't look for another crosswalk for 100 frames (this lets us ignore the red marker on the other side of the crosswalk).

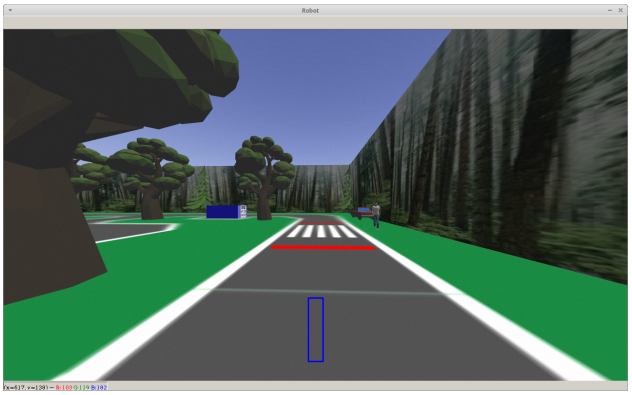


Figure 3: Searching for crosswalk

The methodology to detecting the pedestrian was to look at the amount of white pixels on the actual cross walk. The idea is that when the pedestrian crosses, he/she covers a portion of the crosswalk and hence lowers the amount of white pixels in that area. This is shown in the Figure 4, where the red box represents the area under consideration. As the pedestrian covers the cross walk in the right picture, there are less white pixels at that moment.

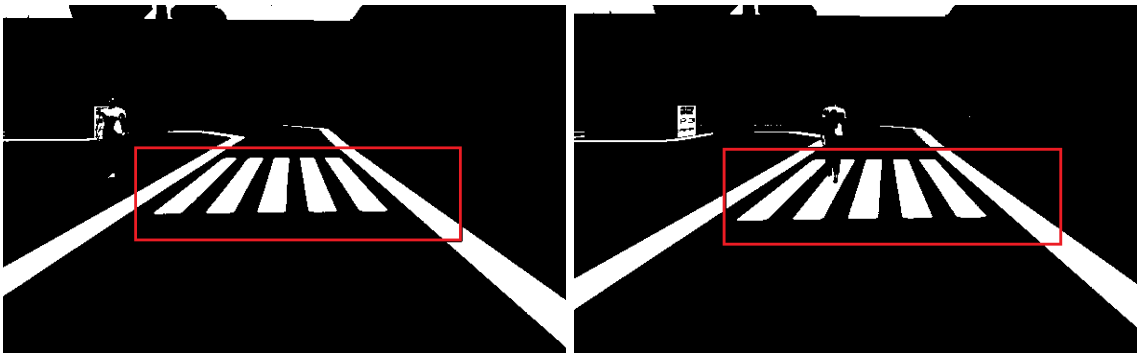


Figure 4: Clear crosswalk (Left), Pedestrian covering the crosswalk (Right)

Initially, we said that if the number of white pixels stayed above a threshold for 8 frames, the crosswalk was clear and the robot could proceed. However, this meant that at times both the robot and the pedestrian could start moving towards the crosswalk at the same time and collide. Thus, we decided to wait for the pedestrian to cross once before we moved forward.

3.3 Truck Detection

Detecting the truck was a matter of looking at the specific color on the truck and coming up with a loose mask based on this color, filtering contours by area, and then applying a tighter mask. The last mask was an extremely tight bound so that at most times the truck detection image was essentially a black screen with no contours. When the truck passed, even a the majority of it was filtered out. This is shown in Figure 5, where the truck is clearly in front, but the mask only reveals a few pixels surrounded by the red bounding box. However because the masks were so tight to the truck, we could safely say that if any contours were found the truck was in front of the robot.

If we were to do this again, we would probably look into a more robust way of object detection rather than looking at the color and shape of this specific truck. But with the time constraints, it did not make sense to improve upon a solution that was working consistently.

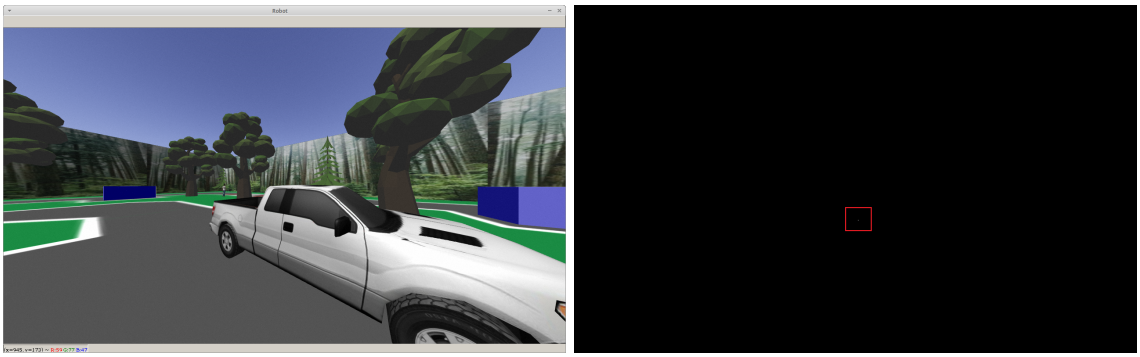


Figure 5: Robot view of truck(Left), Applied truck mask (Right)

If we detected that the truck was in front of us we simply told the robot to stop until the truck was out of view and then we continued along our way.

4 Plate Identification

The plate identification portion of the project can be broken into two parts: plate detection and plate classification. The plate detection portion involved going through the raw image from the camera and extracting a usable photo of the license plate to be used in the classification. Plate classification involved finding each letter in the image of the plate and sending it through the convolutional neural network (CNN) to classify the letters and plates.

As the camera extracted images, we tried to find plates in every image rather than using one single image. This gave us a higher probability of finding a good image that the CNN could use to classify the individual letters. The plate detection was one class and the plate classification was another class. The plate detection class instantiated a plate classification object, where it sent the images that it had found.

4.1 Plate Detection

4.1.1 Locating License Plates

We had tested out a few methods of getting usable images of the plates and spot numbers, but we settled on using masks. With masks, it allowed us to isolate for a specific range of colours in HSV using `cv2.inRange`. While this method may not scale well with different colours, the constraints of the competition made this a viable method.

The original idea was to find all of the blue areas in the camera, then find the largest contours that arise due to these blue areas using a mask. However, we ran into some trouble trying to isolate the blue sky from the frame. For this reason, we decided to use a grey mask to isolate the light grey colours in the frame.

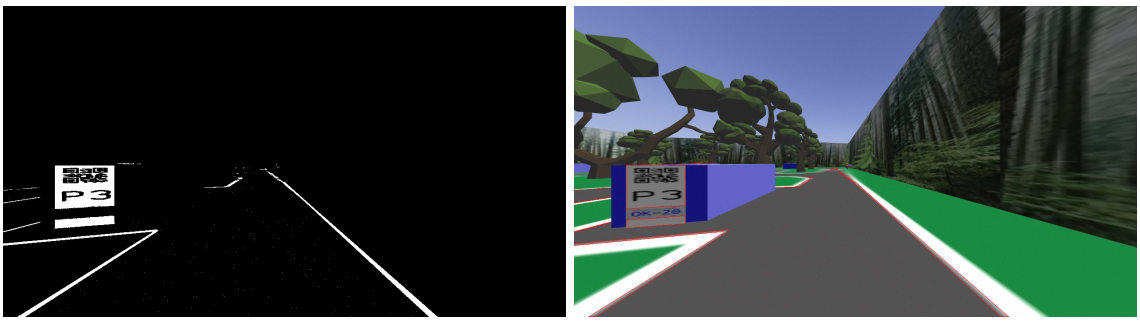


Figure 6: Mask isolating light grey (Left), Contours around the grey objects (Right)

This worked for the most part, but we did get a lot of extra contours from different objects, as seen in figure 6. For these extra contours, we made a set of requirements for the contours to pass. These requirements included being in the right range of widths, heights and aspect ratios (aspect ratio = height/width), being above a minimum contour area threshold and having the leftmost point not on the left edge of the image. This final requirement was such put into place as when the plate was identified to be on the very left side of the image, the resultant image would always be very skewed and cut off. This way, we would only be collecting good candidates for the license plate. From the contour of the license plate, we could get a bounding box and extract that image. The resultant image is below.

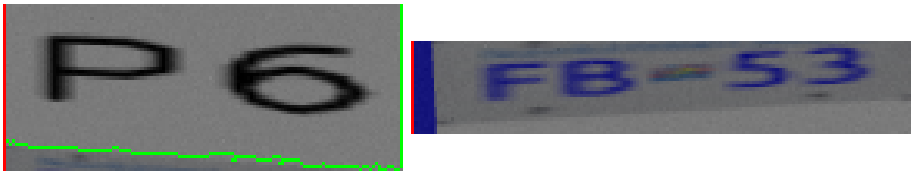


Figure 7: Images of the parking spot number and license plate before processing

Clearly, these images are very skewed. During time trials, we had actually used such an image and sent it to the plate classifier to extract the individual letters. This worked fine, but would result an a wrong plate quite often. Since we were taking many images of the same plate, this method would identify the license plate correctly at least once, but it would also result in many incorrect predictions for the same license plate.

4.1.2 Unskewing the Image

After time trials, we decided that the quality of the images were the limiting factor of our CNN performance, so we decided to unskew the image to clean up the images. Using `cv2.getPerspectiveTransform` and `cv2.warpPerspective`, we could get an image of the license plate that was straightened out. The function `cv2.getPerspectiveTransform` gave us a transformation matrix based on the corners of the bounding box and where we want these corners to go.. These corners were found using `cv2.approxPolyDP`. These corners wreStraightening out the plates also meant that there was a cleaner distinction between letters, so overlapping letters were more likely to happen prior to the transformation. A before and after comparison is seen in figure 8, which is the cropped plate from figure 6.

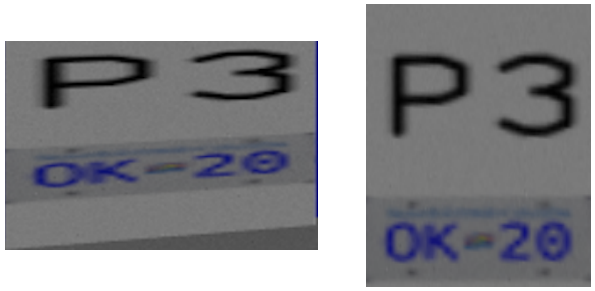


Figure 8: Image of the license plate before and after image processing

With these images, we split up the images of the parking spot numbers and the license plate. This was done just using a constant ratio of the height as place to be cut. Sometimes, if the skewed image was extra flattened, the images could have been cut in a way such that the top half of the license plate was included in the parking spot number image. Although rare, these images would not give good results, and the confidence when sent through the CNN model would not be very high. An example is shown in figure 9.

Figure 9: Example of a poorly cropped image

A solution to this would have been to find the contours of the license plate itself rather than a set height ratio crop. The cropped images were then sent to the image classification part of the license plate identification.

4.2 Plate Classification

The plate classification portion of the plate identification had three parts: the separation of each character, the training of the CNN and prediction using the CNN. For the CNN, we used the Keras library, which provided us with easy tools to build a CNN.

4.2.1 Separating Characters/Preparing Dataset

After receiving the images from the plate detection class, the first step was to separate the images into each individual letter and number. This was done as the CNN was trained on the individual alphanumeric characters. The separation of the characters was slightly different for the parking spot number and the license plate.

For both images, the same general process was used. A black and white image was used to get the contours of each letter. For the parking spot number, a simple black and white threshold was sufficient to giving us a clean black and white image. On the other hand, we used a blue mask for the license plate image, as the characters on the license plate image were blue. A blue mask for a license plate is shown below.

Figure 10: A license plate and the blue mask for the license plate

For the parking spot number image, the two largest contours by bounding box area were chosen, while four largest contours were chosen for the license plate image. The bounding boxes of these chosen contours were then cropped out to give us the resultant images. The contours were then sorted based on the x-coordinate of their top left point in the bounding box, as that would give us the letters in the correct order. Figure 11 has examples of these cropped letters in black and white.

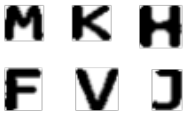


Figure 11: Examples of cropped letters

For the parking spot number, separating the letters was fairly simple and robust as the letters were fairly straightforward, as the letters were large with distinct separation. The license plate was much more complicated. As the letters were much smaller, the space between letters was much smaller. On top of that, the images would have some level of blur, causing the letters to merge with each other at times. This would cause the image to be classified as one letter, when it was really just two letters together. At one point, this was the major factor that was holding back the CNN performance. On top of that, both the first two characters and the last two characters were sometimes merged. This can be seen in figure 12.

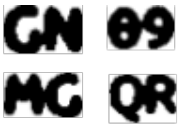


Figure 12: Examples of merged characters

The merged image problem was solved by noticing that along with the large contours of the letters, if letters were merged, then smaller contours were included in the four largest contours. As there was the possibility of two merged images in one license plate, we had to compare the two largest contours the two smallest contours. If these contours were a certain amount smaller than the largest contour, we would know

that we had a merge. These merged images were then split up down the middle of the image. Examples of merged characters is shown below.

At one point, the validation accuracy was down to 5% due to these merges. After fixing these merged characters, the validation accuracy was boosted up to the high 90% range. The parking spot numbers never had a problem with merged letters, due to its size. After finding the individual letters, each letter was standardized to a 30×30 image to be processed in the CNN.

4.2.2 Training the CNN

To train the CNN, data was generated to match the real life data that was received from the camera. Plates and parking spot numbers were generated with an assortment of random augmentations, such as random brightness levels, random blurs, randomly generated noise, as well as random heights and widths. Examples of these plates with random augmentations are shown below.

Figure 13: Examples of generated data

After splitting up each image, these images were turned to black and white. We had originally tried using grayscale images without any successful result. The switch to black and white instantly made both the training accuracy and the validation accuracy jump from ∼1% to ∼90%, thus we stayed with black and white images. Before feeding these images into the CNN, we used Keras' pre-processing library for additional data augmentation. Through `keras.preprocessing.image.ImageDataGenerator`, we specified different rotations, zooms and translations as additional augmentation to the training data.

For the CNN itself, we chose an 80-20 training to validation set split, as it had worked well in previous models. We used a learning rate of 0.00001, which was chosen through trial and error of different learning rates. The architecture of the CNN is shown in figure 14.

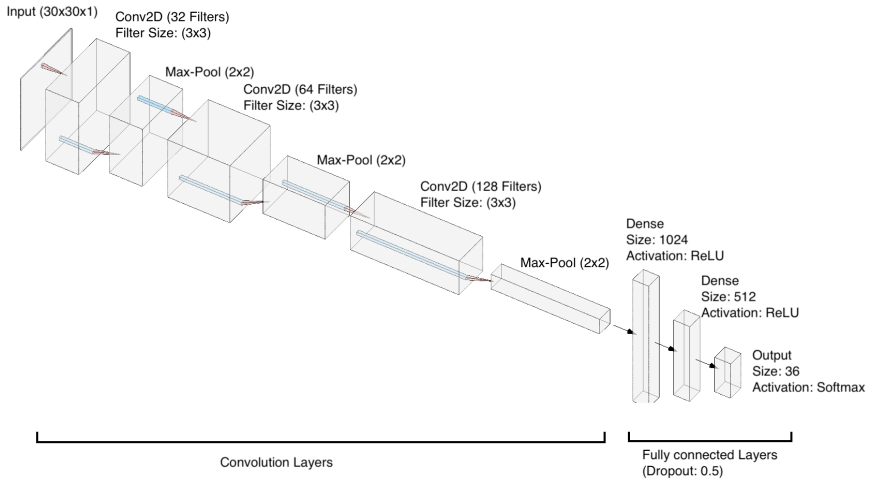
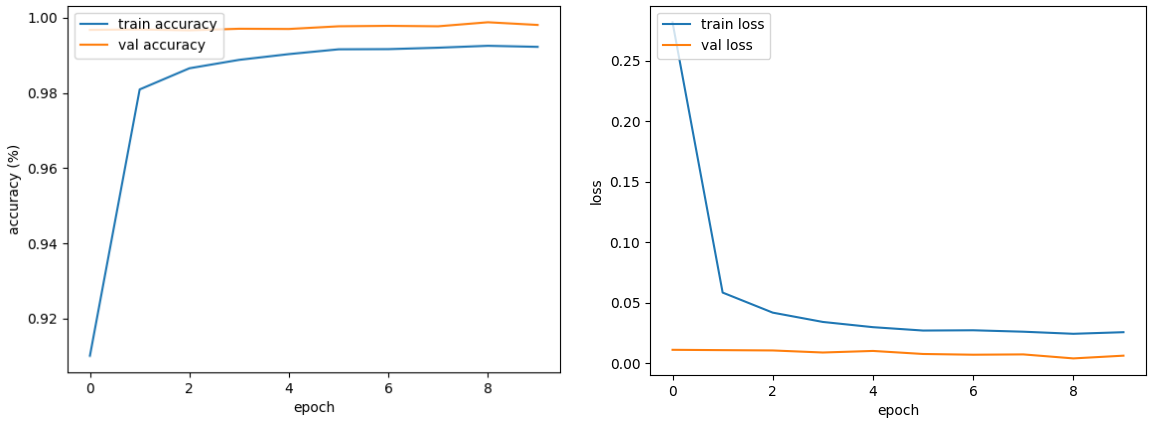


Figure 14: CNN Model Architecure

We originally had two convolutional layers and two fully, but we decided to increase it to three layers each when testing out the best configurations. The idea behind more convolutional layers that was to be able to identify more features of the characters. We were not getting good accuracy specifically on license plates, but we were getting good accuracy on the parking spot number.

We had tested out different configurations and we settled on the one we currently have as it had the highest validation accuracy alongside good real-life accuracy. Other models had shown good validation accuracy but were not as well-suited to the actual competition. The sizes and number of filters, as well as the sizes of the fully connected were all based on common numbers of models that other people had used online. We found that 10 epochs was a good number, as more epochs would go further into overfitting. A diagram of the final CNN model's training/validation accuracy graph as well as the loss graph is shown in figure 15.



(a) Model accuracy over epochs

(b) Model loss over epochs

Figure 15: Graphs characterizing the training of the CNN model

From figure 15, the validation accuracy is consistently much higher than the training accuracy, and the opposite for the loss. This started happening when we added Keras' image pre-processing before training the CNN with the data. The model was trained with 18,000 pictures of license plates and 6,000 pictures of parking spot numbers, so 84,000 total characters.

4.2.3 Image Classification with the CNN

With the model saved, we could load up the model and use it to predict the letters. Originally for time trials, we used the `model.predict_classes` function provided by Keras to give us the output vector of the string. However, we found that if the prediction was uncertain, it would give us a wrong output. This was a reason why we were printing off many different wrong values of the license plate.

For the competition, we ended up using the `model.predict` function, which outputs the arrays of the probabilities. Using these probabilities, if the max probability of any of the characters fell below a certain threshold, we would not consider that entry. After this change, we very rarely got a completely wrong plate. However, if none of the pictures taken were good, there was the chance that we did not get a string for that plate at all. I would estimate the failure to get any plate at 1 in every 20 plates. It was rare, as our strategy was to go around the outside loop twice, thus having two opportunities to take many pictures of the plate. Thus, the model was very likely to pick up any plates that it did not get the first time around.

After receiving the list with the indices of the maximum probability using `np.argmax`, we converted the

indices into a string that the ROS node could publish. We kept a dictionary with the parking spot number as the key and a 2-tuple as the value. This tuple contained the string of the license plate, as well as the probabilities of each of the letters. Each probability was used to compare against other instances predictions on the same plate. If the prediction for the letter was different than the current guess, and the probability of the prediction was higher, then we would change the prediction of the license plate. This very rarely happened, as the minimum probability threshold often meant that prediction that got through was already at a high probability.

During the actual competition, the model failed to pick up one of the outer ring license plates in parking spot 4 even as it had went around twice. We take between 5 and 10 images every time we pass by each car. This meant that the system had failed to classify the plate around 20 times, which is a rare occasion. Better image pre-processing could have been the solution, as the image could have been skewed even after the transforming the picture. It could have also been due to the wobbling of the car when driving, as the pictures it produces while turning are more skewed and tilted.

5 Results and Conclusion

In the competition, we received 43/57 points by correctly reporting five outer plates and one inner plate.

Unfortunately, parking spot 4 did not reach the confidence threshold on either one of our passes. So, we were also slightly disappointed that we did not get all of the outer plates as we were doing so consistently in the practice runs before hand. If we were to do this again, we would have set the threshold much lower, such that we would report a larger quantity of thresholds, and improve our image pre-processing.

In terms of general software construction practices, we learnt about the importance of well structured code and thorough testing. Although we did multiple test runs to test functionality, we did not do timed tests until the day of the competition. Initially, the inner loop was just an extra that we had added in just in case the plate detection could work in one round. However, we did not think that we would have enough time to go for both inner plates. When the time was indeed extended, In the end we think the robot did have enough time to go for the last inner plate, however, we did not have enough time to modify our code to do so. So in the future more thorough testing and better code structure is something that we will be more cautious about.

Overall, we managed to detect objects and create a fairly consistent CNN and are proud of our accomplishments.