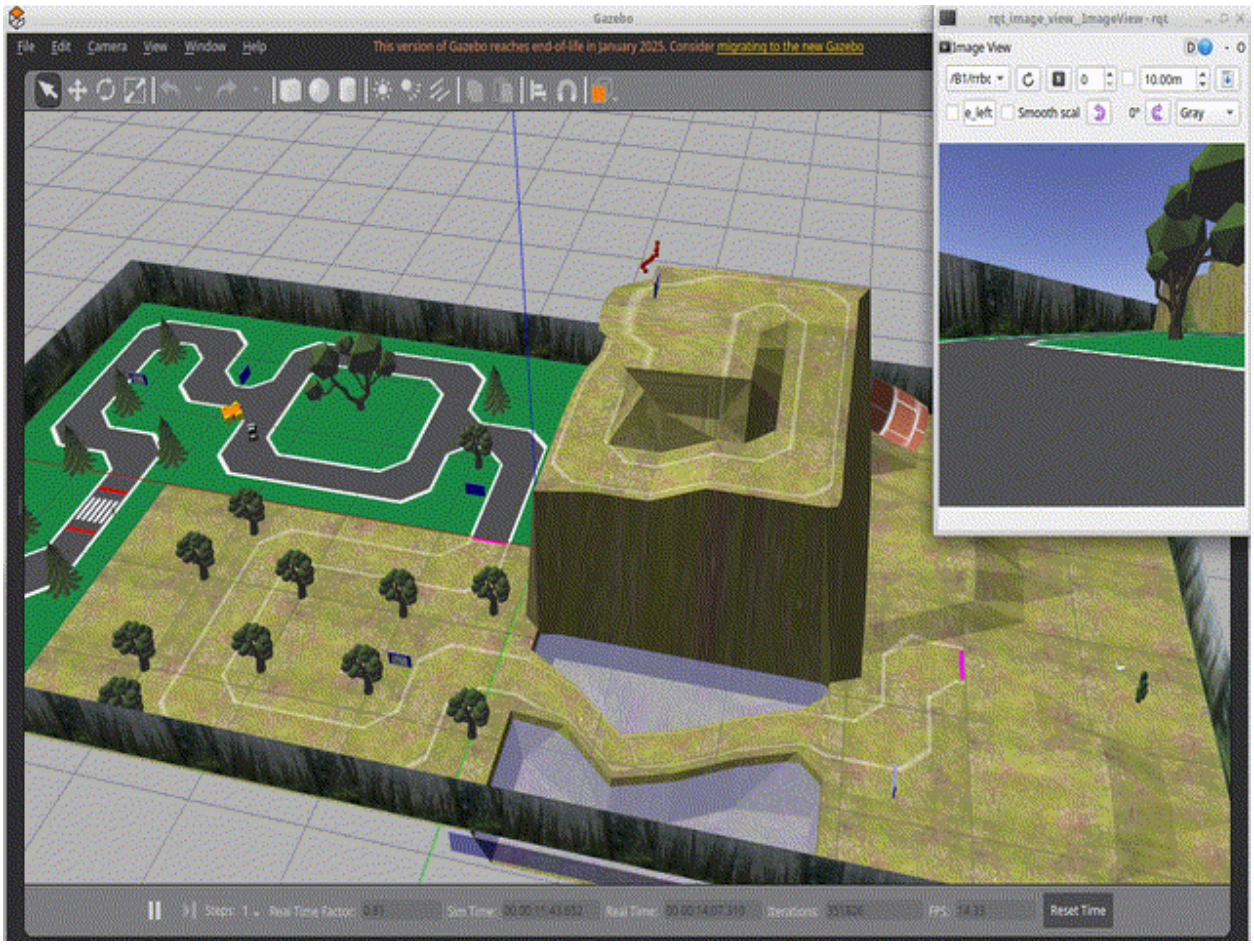

ENPH 353 Competition Analysis and Final Report

Harchetan Chohan & Coen Molyneux

December 2025

"A year spent in Artificial Intelligence is enough to make one believe in God"

- Alan Perlis

Table of Contents

1 Introduction

1.1 Competition Overview

The competition consisted of programming an autonomous robot to navigate a simulated world using the ROS framework. The simulated world was created using the physics simulation software, Gazebo. The course was divided into four main segments each with increased difficulty or nuance to navigate. These segments included paved road, grass terrain, and moving obstacles (see figure 1). In addition to navigating the course, our robot also had to be able to read clue boards along the way. The goal was to successfully navigate the course and read all these clue boards correctly to solve a mystery.

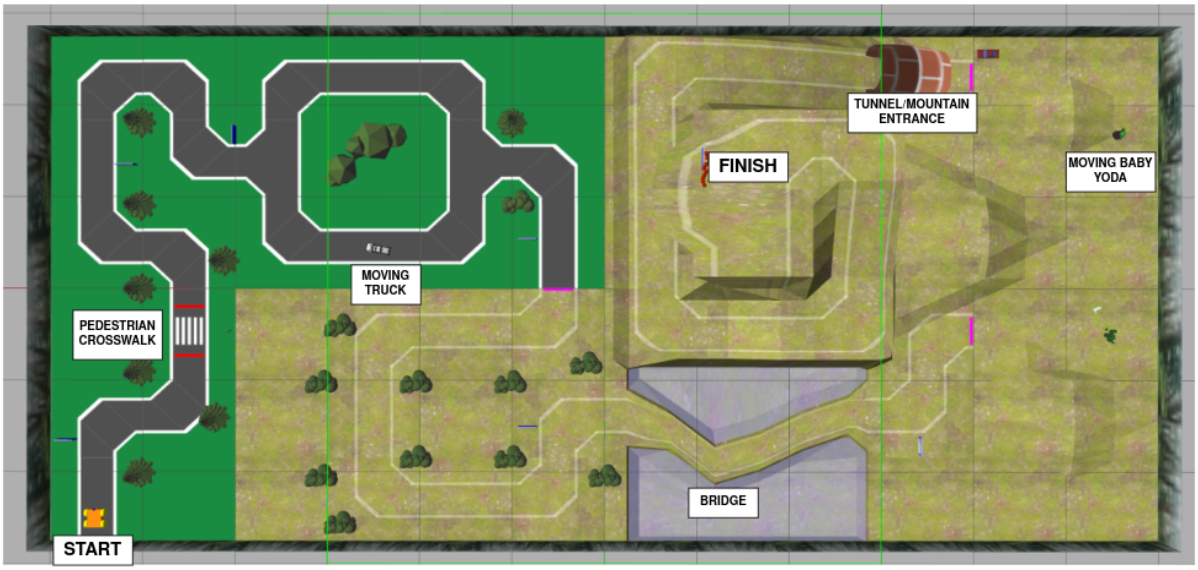

Figure 1: Map Layout

1.2 Contribution Split

Coen focused primarily on building the imitation learning driving control. This included training the convolution neural networks, building data collection and training pipelines, the driving state machine responsible for choosing drive modes.

Harchetan focussed on the detection of the signboards, and then the further reading and publishing of the clue keys and values on these signs.

1.3 Software Architecture

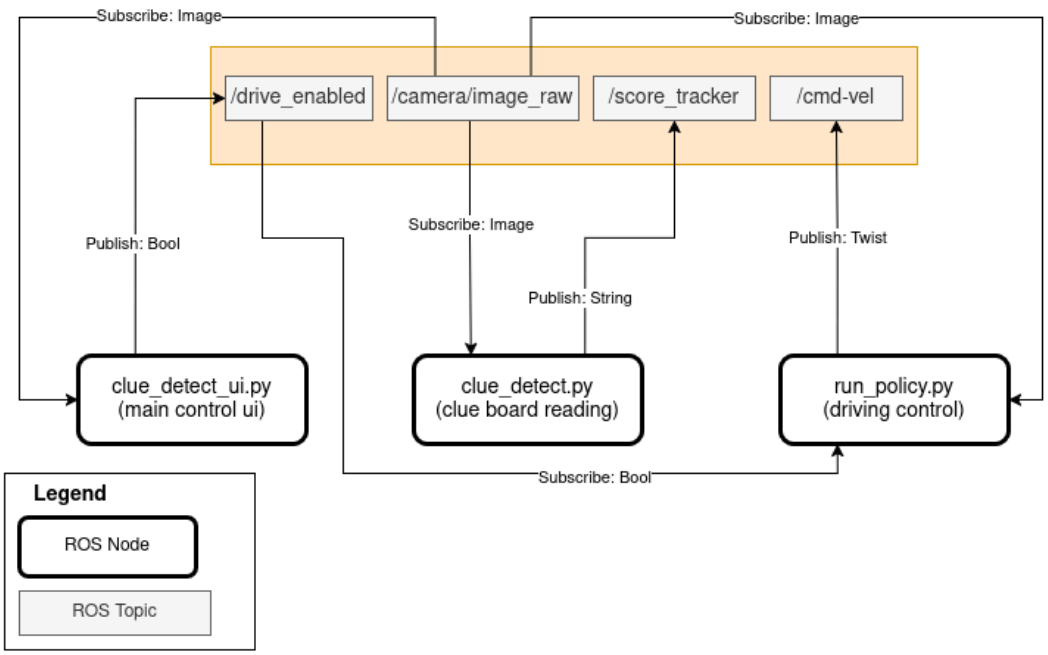

Figure 2: Software Architecture

2 Driving Control

2.1 Method

The driving control system was based on imitation learning using convolutional neural networks (CNNs). The approach was to train a CNN for each section of the map including pavement, grass, and mountain terrain (see figure 1).

Imitation learning involves training a neural network to replicate observed behavior. In this case, the networks were trained to mimic human driving in simulation. Thousands of images were collected during manual driving, and each image was paired with the corresponding driving command: turning left, right, or straight. The network learned to associate visual features in the image with the appropriate drive command.

Multiple networks were trained as opposed to one large network, for both computational reasons and piece of mind. A single model trained on the entire map would have required many

more parameters, which would greatly slow down how fast the model makes its decisions (inference time).

Additionally, training a single network on new sections of map risked breaking other previously working sections. For example the model could drive great on the pavement but then when I go to train it on the grass it now breaks the driving ability on the pavement. So by taking the approach of multiple networks once we had a section working we could just ‘set it and forget it’

2.2 Data Collection

Training data was collected by manually driving the robot on the desired path. During the driving, the camera images were recorded alongside the corresponding driving commands decided by the human. The images were saved and a csv file stored pairs of image name and drive command i.e 3252.png, LEFT.

Multiple datasets were collected for each driving surface. We also later collected datasets at different camera FOV’s which aided in clue board reading. We also collected smaller targeted datasets to enforce certain behaviors. For example when entering the roundabout the robot must turn left to avoid collision with the truck (see figure 1). To enforce this smaller datasets were collected of just that single turn and then fine tuning the working pavement model on that data.

2.3 Implementation

The implementation involved a state machine that detected the pink and red lines at the boundaries of each driving surface to toggle between models. See A-1 for flow chart

2.3.1 Pedestrian Detection

Using the red line at the pedestrian crosswalk (see figure 1), the state machine would stop the robot and subtract consecutive to image frames to detect if there is some difference across the pixels. If the pedestrian was not crossing theoretically the difference between frames would be zero. So by using difference in consecutive images we could wait for the pedestrian to first cross and then continue

2.3.2 Truck Detection

Due to time constraints we were not able to fully implement our original plan for truck detection. We were going to attempt to train a CNN on determining if the truck was on the right, by cropping the camera feed to only look at the right side of the roundabout and then to detect if the truck was present.

2.3.3 Baby Yoda Detection and Usage

To detect Baby Yoda, the camera image was cropped to a region and masked to isolate green pixels. Figure 3 the result mask alongside the coloured image. We used the number of white pixels in the masked image to determine where Baby Yoda was. We would wait for Baby Yoda to approach and then start walking away before we started driving. We could then train a CNN to follow Baby Yoda, as the path he takes goes to the tunnel.

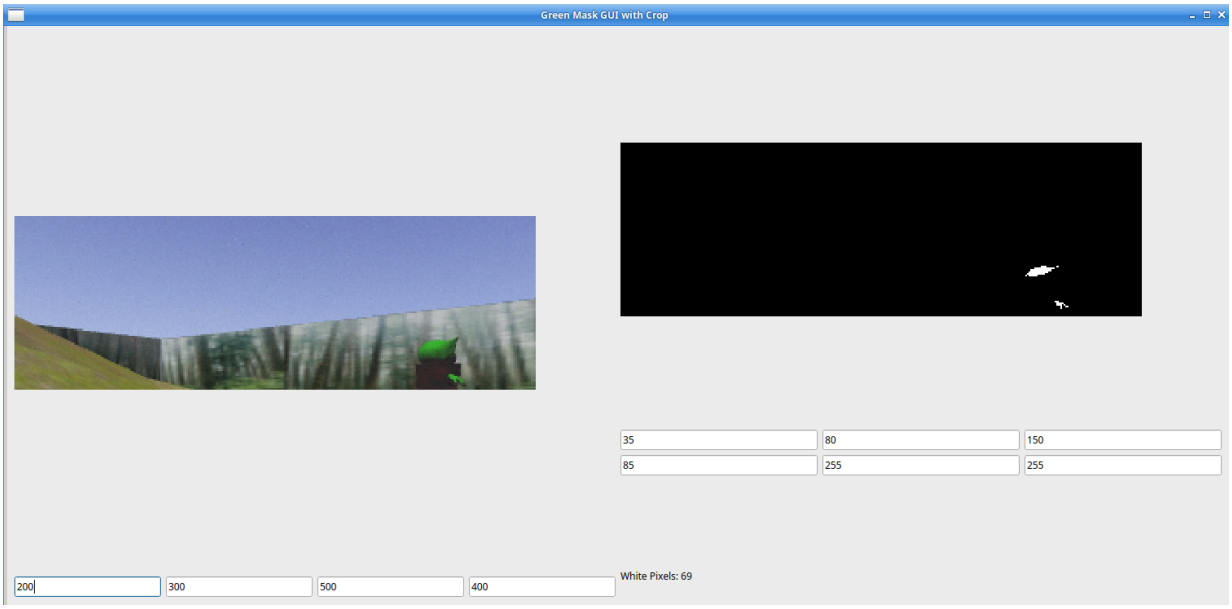

Figure 3: Baby Yoda Detection Tuning UI

2.4 Debugging

For debugging the main issue we encountered was the preprocessing of the images to feed into the network. Instead of giving the network the entire image we applied some classical techniques to enhance features and reduce noise in the image.

Since the pavement section had high contrast to begin with, all we had to do was crop out the sky in the camera feed, keeping all three colour channels (RGB). However the grass surface has a lot of noise, and much lower contrast. To counteract this we first applied a technique called CLAHE (Contrast Limited Adaptive Histogram Equalization), which essentially boosts local contrast while avoiding amplifying noise, and then followed that up by grey scaling the image.

Since all of these “classical” image techniques require thresholds we created custom UI’s to help determine them. See figure 3 as an example

3 Clue Board Detection and Reading

3.1 Method

The core of the method was first isolating the sign, by masking it from the surrounding environment by its unique blue colour. Once the contour for the sign was located, and it was verified that it had 4 edges, the 4 corners of the sign in the image frame were found and ran through a perspective transform to give us a frontal view of the signboard. The characters on the signboard could then also be found by running it again through a blue mask, to identify contours. Note that a maximum area is specified for the sign size, to avoid seeing the sky, and a minimum size and aspected ratio are specified for the characters to avoid catching multiple characters together and just noise. All the same, in finding a bound box that fits around each of these contours (with a bit of added padding to avoid cutting of contour edges), we would then find a frame fitting each of these characters. Then scaling these frames to a size appropriate to input into our CNN, after some preprocessing (i.e. gaussian blur), we could get the expected character and the associated confidence in the prediction of it.

3.2 Data Collection

Data for preparing the CNN to read the clue boards was both generated and collected. Figure 5 shows these two types of data side by side.

The generated data (about 16000 characters in the final iteration) was produced via the core script given for sign board generation in the competition package, with some modification to get a diverse range of words. To simulate the various imperfections in the images we would be capturing, various elements of randomized noise were added; namely, a significant amount of perspective distortion and brightness change was added, while some blur was also added. This generated data was used exclusively for the purposes of training the CNN for letters.

The collected data (only about 400 characters) was assembled by driving around the competition track and snapping images of the signboard via a GUI and capture algorithm that extracted the letters into individual images. These images were then manually labelled to match the letters they contained. These images (after some scaling and processing) were ultimately used for the purposes of validating/testing the CNN model, as if it succeeded with these images, it was likely to succeed in the competition as well.

3.3 Nuances in Implementation

3.3.1 CNN Implementation

The convolution neural network construction was quite standard, and Figure 6, 7, and 8 show some of the key features concerning its training and validation. Key features of the CNN training

include that it consists of 4 layers of increasing numbers of neurons, using a ReLU activation function, and has a 64 by 64 input parameter.

Some interesting approaches ultimately helped optimize it. Namely, dropouts of a meaningful magnitude (0.25 and increasing) were added in each layer, as were block normalizations (note this was a suggestion by AI, and it ultimately did not seem to affect performance, but did improve training speed). Moreover, some fail safes were added to prevent overfitting, and also reduce the learning rate by a factor of a half if the loss begins to plateau (the base learning rate was set to 1e-4).

Ultimately, the CNN was incredibly efficient and effective after several iterations of training (with modifications to the input data processing parameters, and to the CNN training parameters).

3.3.2 Sign Reading Implementation

The core of the sign reading approach was laid out in 3.1, but the sign reading algorithm was effectively the master loop for our robot (as it both initiated and ended all robot processes), hence it is worth discussing further. A simple start timer button was what initiated the score tracker timer and also launched the policy node to driving. Likewise, a manual stop button was built in, but the main loop ended a termination sequence once the last clue key (“Bandit”) had been read (3 times just for safety).

Likewise, various shortcuts were made in an effort to compensate for the difficult nature of getting a good picture while moving. Namely, the clue keys were known to one of eight possible words, and hence they could be predicted given certain criteria (at least 4 letters detected, with 3 of a 60% confidence or greater). This made it so that the value was the only real worry about reading precisely, while the key could be quite quickly and roughly pieced together.

Additionally, processing the characters in batches with the CNN, as opposed to individually, greatly sped up this process. The requirements set on minimum confidence to publish were not very stringent (5% confidence average across all letters at least), but it was required that the confidence be higher than the previous reading to re-publish a new value to a given clue. Moreover, this called for certain bounds being put on the allowability of re-publishing even with lower confidences (as sometimes a letter was missed by the contour identification, but all the other letters were read very well, meaning the incorrect entry was published with a high confidence). Some hard filters were also added; for example, no clue reading could exceed 15 characters or have 4 or more of the same consecutive letter (this helped get rid of some of the errors being suffered).

On a final note, in an attempt to better the functionality of the clue detection and CNN, a CLAHE filter was applied to training and input data - this ultimately yielded worse performance, and was hence abandoned.

4 Conclusion

4.1 Performance

The robot performed very well in the competition, we placed 3rd. We detected all clues boards and misread two of them since they contained numbers and our CNN for reading clue boards was not trained on numbers.

Originally during time trials we used artisanal methods for driving. This worked well for pavement driving, but we decided to switch to imitation learning for competition. The thought behind this was that once we get the framework setup for one driving section, it should be more or less the copy paste for the other sections. Indeed this was the case, we used virtually the same notebook for training all sections of driving just with different image preprocessing components.

4.2 Areas for Improvement

Something we could have done differently was the ‘centralized’ command node. It ended up being the clue detector node, which in hindsight doesn’t make much sense. Rather it would be better to have a centralized node that takes in all the information (subscribes) and then distributes to the driving and clue detector nodes, these nodes return their results and then the centralized node publishes its decision.

We additionally had some major oversights, including the fact that again numbers were something that our CNN should have been able to recognize, and the use of an effectively strategic timing based approach to overcome the pickup in the roundabout. Likewise, some type of teleportation mechanism, to restart points, would have been a good idea, as we did have a fair danger of getting stuck/lost.

A Appendix

A-1 Driving State Machine

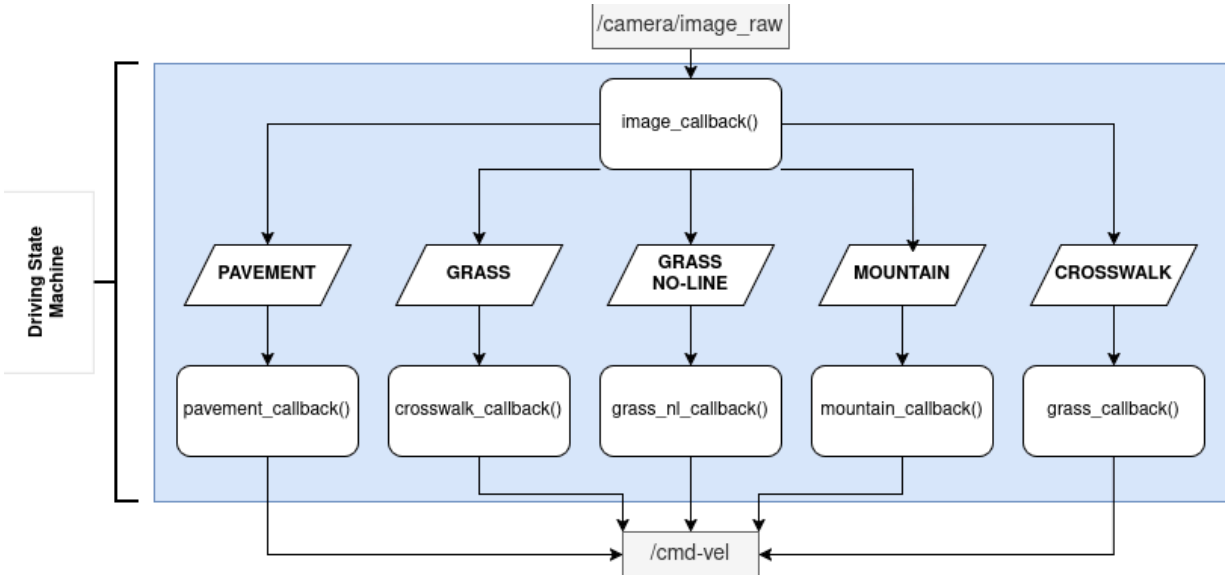

Figure 4: Driving State Machine Flow Chart

A-2 Generated and Collected Data Characters

Figure 5a: Generated Image Data - post processing

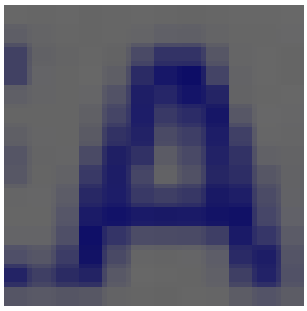

Figure 5b: Captured Image Data - pre-processing

A-3 CNN Character Recognition Training

Figure 6: Histogram showing distribution of generated training character data

```
Epoch 1/25
198/198 [==============================] - 11s 56ms/step - loss: 2.3160 - acc: 0.4018 - val_loss: 1.5794 - val_acc: 0.6584 - lr: 1.0000e-04
Epoch 2/25
198/198 [==============================] - 11s 54ms/step - loss: 1.0083 - acc: 0.7102 - val_loss: 0.6876 - val_acc: 0.8644 - lr: 1.0000e-04
Epoch 3/25
198/198 [==============================] - 10s 51ms/step - loss: 0.6461 - acc: 0.8173 - val_loss: 0.5331 - val_acc: 0.8954 - lr: 1.0000e-04
Epoch 4/25
198/198 [==============================] - 11s 56ms/step - loss: 0.4953 - acc: 0.8574 - val_loss: 0.3827 - val_acc: 0.9271 - lr: 1.0000e-04
Epoch 5/25
198/198 [==============================] - 11s 56ms/step - loss: 0.3836 - acc: 0.8986 - val_loss: 0.2957 - val_acc: 0.9430 - lr: 1.0000e-04
Epoch 6/25
198/198 [==============================] - 11s 57ms/step - loss: 0.3090 - acc: 0.9194 - val_loss: 0.2090 - val_acc: 0.9632 - lr: 1.0000e-04
Epoch 7/25
198/198 [==============================] - 11s 56ms/step - loss: 0.2701 - acc: 0.9285 - val_loss: 0.2017 - val_acc: 0.9620 - lr: 1.0000e-04
Epoch 8/25
198/198 [==============================] - 11s 57ms/step - loss: 0.2180 - acc: 0.9482 - val_loss: 0.1401 - val_acc: 0.9867 - lr: 1.0000e-04
Epoch 9/25
198/198 [==============================] - 11s 55ms/step - loss: 0.1881 - acc: 0.9591 - val_loss: 0.1201 - val_acc: 0.9861 - lr: 1.0000e-04
Epoch 10/25
198/198 [==============================] - 11s 55ms/step - loss: 0.1582 - acc: 0.9678 - val_loss: 0.1026 - val_acc: 0.9848 - lr: 1.0000e-04
Epoch 11/25
198/198 [==============================] - 11s 56ms/step - loss: 0.1483 - acc: 0.9672 - val_loss: 0.1121 - val_acc: 0.9804 - lr: 1.0000e-04
Epoch 12/25
198/198 [==============================] - 11s 56ms/step - loss: 0.1170 - acc: 0.9783 - val_loss: 0.0701 - val_acc: 0.9937 - lr: 1.0000e-04
Epoch 13/25
...
Epoch 24/25
198/198 [==============================] - 11s 55ms/step - loss: 0.0471 - acc: 0.9929 - val_loss: 0.0350 - val_acc: 0.9975 - lr: 1.0000e-04
Epoch 25/25
198/198 [==============================] - 11s 57ms/step - loss: 0.0434 - acc: 0.9938 - val_loss: 0.0307 - val_acc: 0.9956 - lr: 1.0000e-04
```

Figure 7: CNN training loss and val_loss (as well as accuracies) with respect to epochs

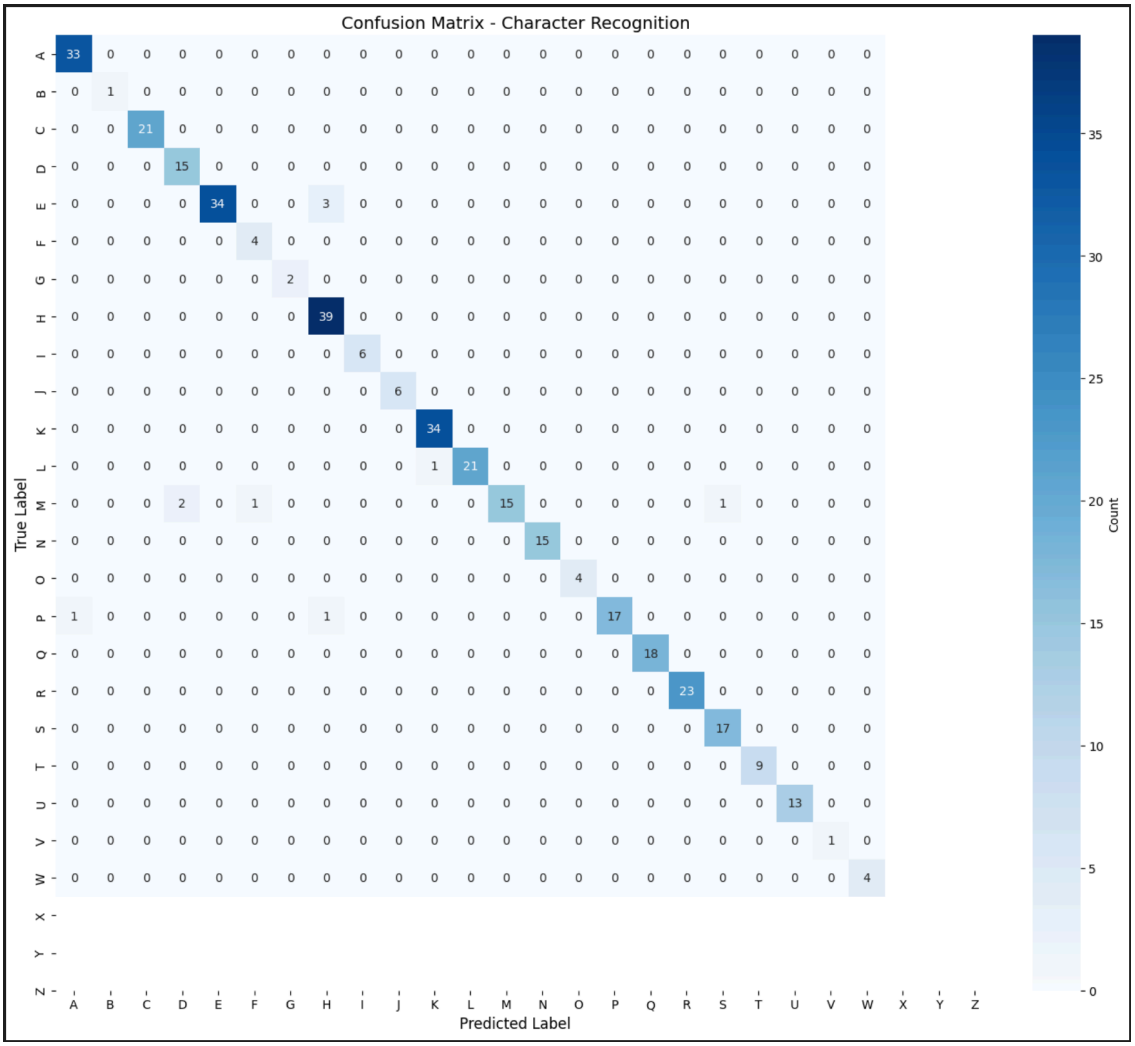

Figure 8: Confusion matrix showing the letters being confused for one another by the CNN - note, that visually inspection of these failed cases ultimately revealed that the photo captured for testing was just particularly poor in the cases of misrecognition.