University of Western Australia

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

THE UNIVERSITY OF WESTERN AUSTRALIA

Final Year Research Project GENG5511/ 5512:

DEEP LEARNING FOR MOBILE ROBOTS

Author:
Shuang-An Yu (Angelo)

Supervisor:
Professor Thomas Bräunl

18^{th} of October 2021

Word Count: 6013

DECLARATION

I declare that the work in this dissertation "Deep Learning for Mobile Robots" has been carried out by me for the completion of Master of Professional Engineering Degree at the University of Western Australia. This thesis does not copy or infringe any work by other individuals and institutions. Information from literature and similar works are acknowledged and cited in the references provided.

E-Signature

18/10/2021

Date

Abstract

This project aims to apply deep learning using images taken by cameras on mobile robots to improve autonomous driving. Scope of the project covers lane-detection and traffic sign recognition in a simulation environment. Past UWA projects focused mainly on image processing techniques such as edge-detection, and showed low accuracy in its neural network algorithms for lane-detection. However, literature states the limitations in image processing such as inaccuracy in detecting inconsistent lane markings, and suggests effectiveness in autonomous driving adaptation through deep learning. The research aims build onto existing neural network algorithms aided with image processing functions to achieve a fully autonomous mobile robot.

The proposed driving method focuses on convolutional neural networks through a python virtual environment with Tensorflow, OpenCV and open-source deep learning packages. Deep learning algorithms is implemented through Tensorflow models to train data obtained by images through OpenCV functions. Using analysis tools through Tensorflow, image classification accuracies of over 90% is seen with "Inception V3" and "PilotNet" convolutional neural network architectures. The trained neural network models are implemented by mobile robots to correctly manoeuvre along marked lanes in the virtual environment. Through testing and simulation, the mobile robot can also be adapted to a foreign environment with similar lane markings.

Major failure points of the robot include the inability to error correct once the robot has deviated from the lane markings. Similarly at certain locations of the road where low frequency of training data was taken. The solution retrains the model with an increased number of data points at the designated failure points to improve Tensorflow prediction accuracy.

The results show that the autonomous mobile robot was able to drive autonomously and follow lane markings. Troubles start to arise at points of intersections where there are low training data subsets. Future studies should include implementing a more complex system that allows distance detection of traffic signs for the mobile robot to travel and stop accurately. Additionally, Tensorflow models that require less computation time should be researched to reduce jerky movements caused by low frame rates.

Acronyms

CNN Convolutional Neural Network.

CPU Central Processing Unit.

DLNN Deep Learning Neural Networks.

FOV Field of View.

GTSRB German Traffic Sign Recognition Benchmark.

ROF Regions of Failure.

SAE The Society of Automotive Engineers.

UWA University of Western Australia.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1 Introduction to Deep Learning

In recent years, deep learning has grown in the field of autonomous driving. Deep learning is a method of machine learning that utilises complex data relations known as neural networks [11]. The ideology of neural networks are based off neurons in the human brain. With conventional computer learning algorithms, engineers are able to simulate human neurons through non-linear relationships [12]. A main category of deep learning commonly used in image processing is **CNN**, Convolutional Neural Networks [13]. Additional methods of deep learning techniques include recurrent neural networks (**RNN**), Long short term memory networks **LSTM** [13]. These neural networks can be used to train classifiers, which are trained machine algorithms that allow higher level operations such as handwriting identification and speech recognition [11]. Autonomous driving is a common implementation of image processing and deep learning. It consists of taking data from a wide range of input sensors to stimulate decision making.

Images taken from a simple camera can be used as inputs to deep learning processes. All visual images can be defined in terms of mathematical matrices through Hue Saturation Values **(HSV)**, Red Green Blue colour scales **(RGB)**, or gray-scales. Image processing are mathematical-based operations that enhances

images using these image matrices. Other techniques of image processing include colour space conversions and image noise removal through Gaussian filters [14]. Higher level image processing can allow shape analysis, edge smoothing and edge detection which are commonly used for operations such as autonomous object classification [15].

1.2 Problem Statement

Methods of computer vision may be limited by real world inconsistencies. For example, lanes markings may be inconsistent, markings may fade, this results in limited accuracy through image processing techniques to lane-detect and error-correct. Through previous literature, deep learning methods such as convolutional neural networks is proved to compensate for such outliers by applying adequate training and data sample-size.

The scope of this project covers deep learning algorithms to traffic-sign detection and autonomous lane-correction of mobile robots. The implementation is simulated the UWA Eyebot Simulation system (EyeSim) developed by Professor Thomas Braunl [16]. The main tools implemented are python-based packages such as Tensorflow and OpenCV for deep learning and image processing. Both are open source libraries that can be executed in python, C and C++.

1.3 Project Aim

The aim of the project is to explore methods of deep learning assisted with image processing techniques to improve on autonomous driving accuracy. The final goal is to design an accurate autonomous driving model that requires little to no human intervention.

The main project deliverables include:

1. Implement existing deep learning algorithms to an autonomous driving vehicle to allow lane detection within the EyeSim system
2. Implement existing deep learning algorithms for traffic sign recognition in the EyeSim system
3. Analyse and improve the performance of the trained neural network models through Data Augmentation[1]
4. Design and implement an autonomous driving model by integrating traffic sign and lane detection in the EyeSim system

1.4 Document Outline

The first chapter introduces the problem statement and aims of the project. Background information is provided to aid the reader to understanding the context of this project.

The second chapter outlines mathematical models relating to background information of the project. Past projects and State-of-the-art literature are discussed to guide project methodologies. This sets a baseline to results and discussion of the project.

Chapter three Methodologies cover the major steps and architectural frameworks to the project. Diagrams and explanations of the proposed model for lane and traffic detection. Moreover includes setup and programs to for the overall delivery.

[1]Data Augmentation: Training data adjustments through computer vision techniques or manual adjustments to acquire a larger, unbiased data set.

The fourth chapter contains detailed explanation on the model design and experimental procedures. This includes the proposed final model, software setup and implementation of training methodologies.

Results of proposed experiments and implementation are outlined in Chapter five. The section outlines the overall performance of the proposed model by assessing it with relevant literature.

Chapter 6 concludes the thesis by summarising the overall experiments and findings. Providing recommendations and remarks for future works in the area.

2. Literature Review

2.1 Deep Learning

Deep learning is defined as a subset of Artificial Intelligence and Machine learning. Artificial intelligence encompasses the broad study of self-learning and self-problem solving abilities for a computer. Where machine learning is computer learning techniques with or without human intervention [17]. The relation is shown in *figure* 2.1.

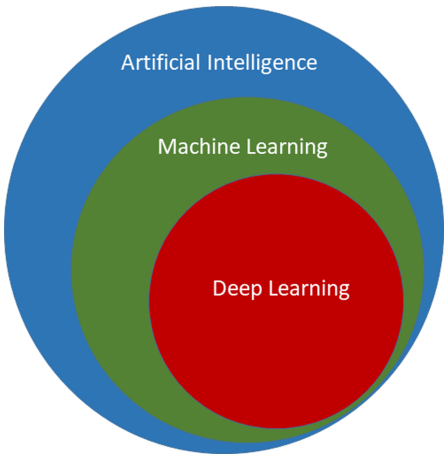

Figure 2.1: ***Relation between Artificial Intelligence, Machine Learning and Deep Learning***

Different to its higher level subsets, deep learning is defined by layered algorithms that allows the computer to compute lower level operations such as image and speech recognition [17]. The primary goal of deep learning algorithms is to calculate a regression path to identify between different images and

speech patterns.

2.1.1 Neural Networks

The basic structure of neural networks consists of three basic architectures, inputs, outputs and weights. Similar to axons in human brain cells, weights are the paths that the neural network information moves across; where different paths would indicate different weightings of importance [18]. The role of the neurons are to classify the identity of specific features [18], where the whole system is a network of so called 'classifiers' make decisions based on the synaptic relations between themselves [5].

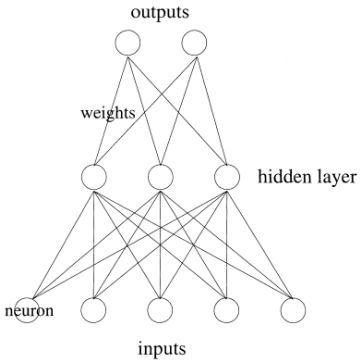

Figure 2.2: ***General architecture of a neural network adapted from [3]***

From *figure 2.2* a hidden middle layer is shown. These are hidden neurons that makes up the internal structure of the deep learning decision tree [18]. The layers can be abstracted as a series of lower level yes/no questions, for example: does this entity have a nose? does it have fur? With outputs being defined as a classified object.

2.1.2 Convolutional Neural Networks

CNN, convolutional neural networks is a type of neural network that utilises three different sub-processes for training. Those are: *feature extraction*, *feature*

mapping and *subsampling* [5]. The structure between the three processes can be visualised in *figure 2.3* below.

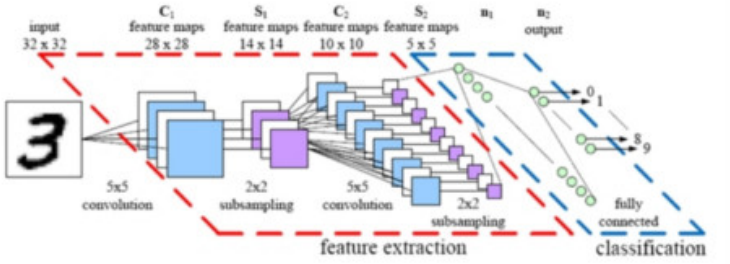

Figure 2.3: ***CNN architecture adapted from [4]***

To train the CNN classifier, individual neurons extracts features from pre-known inputs and between neurons. This is done within the hidden layers through mathematical filtering. In parallel, each hidden layer adjusts its weight path (decision making bias) within each kernel through convolutional operations and weight sharing; this is known as *feature mapping* [5], [18]. Subsampling is a process applied to each set of feature maps to reduce the mapping resolution, this allows the classifier to reduce training bias from insignificant low weighted features through averaging.

This technique of semi-supervised neural networking works exceptionally well in image processing [17]. As it allows break-down of images into multiple layered kernels that operate on small number of pixels. It reduces the complexity of classification by limiting the number of selection parameters for the neurons with weight sharing, eliminating chances of over-fitting the classifier [11].

2.1.3 Mathematical Models of Classifiers (SVM)

The mathematical ideology of classifiers for neural networks are covered in [19] and [5]. The methodologies use a probabilistic approach with regression and weighting vectors to the networking of neurons. The main goal of this method

is to fit mathematical models that separates two different variable parameters. The basic relations can be explained through correlation equations [5]:

$$d = [\sum_{j=1}^{M} w_j x_j + \varepsilon] \tag{2.1}$$

x_j and w_j are the correlation of inputs and weight values to the model respectively; Where the output, d is the desired response and is defined as a 'convolution' of the weight matrix and input matrix. Different methods of regression such *Least mean square Method*[1] are based off fitting of weight matrix values through probabilistic equations of random variables.

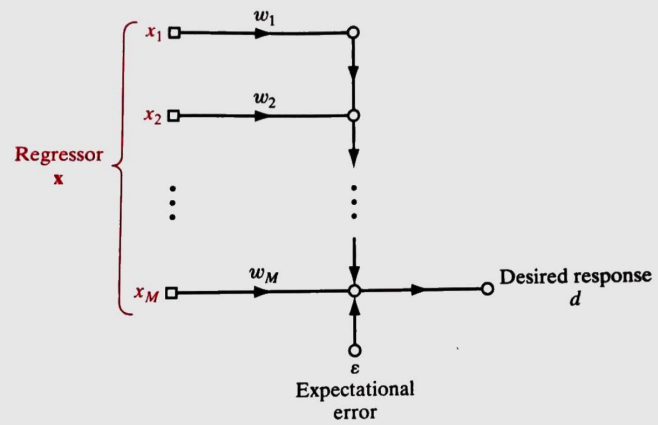

Figure 2.4: ***Linear Regression model of Random variables [5]***

A build onto regression is *Support Vector Machines* (**SVM**). SVM is a binary learning machine that solves pattern-classification and non-linear regression models [5]. In a way, SVM's acts as a basis to **CNN's** studied in the previous section, as SVM's are able to cover feature mapping, and kernel operations.

Figure 2.5 below shows pattern recognition structure with SVM. It consists of a non-linear mapping, followed by a linear regression to separate features that came from the input space. The output of the SVM attempts to divide features through a *hyperplane* visualised below. A *hyperplane* allows mathematical sep-

[1] Method of adaptive filtering

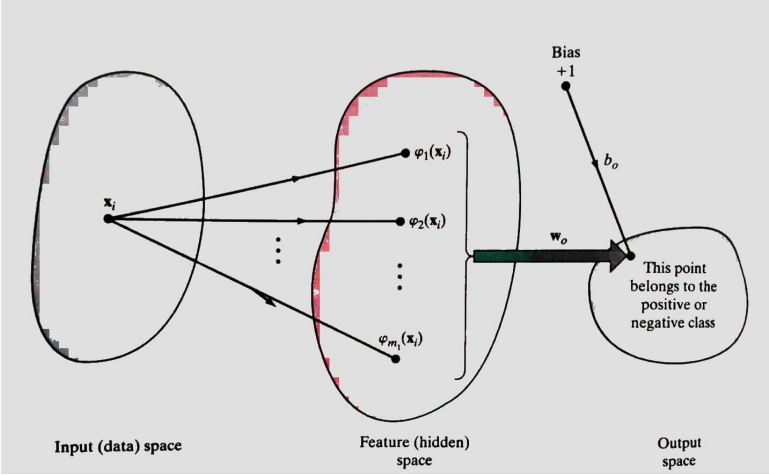

Figure 2.5: ***Pattern classifier architecture of SVM [5]***

aration of $\phi(\mathbf{x_i})$, feature vectors [5]. By finding a plane, it creates a threshold for classification of features such as in an image.

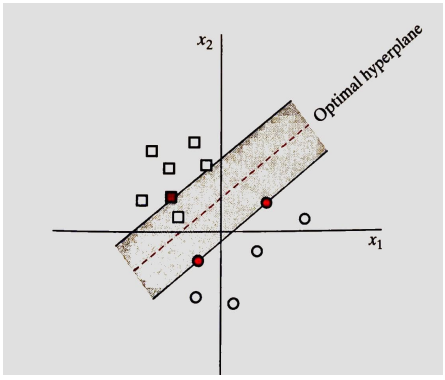

Figure 2.6: ***Hyperplane representation, the square and circles can be interpreted as different features to an image/pattern [5]***

2.2 Image Processing

2.2.1 Image Enhancement

Enhancement techniques covered in [20] suggests mathematical filtering techniques. Filters include: Low-pass filtering, High-pass filtering and Adaptive filtering. Filtering is good in removing noise and smoothing out images through blurring. Subsequently it also permits the opposite, increasing image noise to

sharpen edges of buildings and shapes in images (high-pass filtering) [20]. The mathematical operators of filtering are based of convolutions of matrices that often used in signal processing. Colour conversions also serve as basis to simplifying images for processing, examples include Gray-Scale Modification and *RGB to HSV colour-scale conversions*.

2.2.2 Shape and Edge Detection

Straight-line detection through Hough-transformations is a useful tool in findings straight edges at arbitrary locations [20]. This method of detection can also be used for specific functions such as vertical line detection. *Canny Edge Detection* is another method of processing that identifies unique lines and edges in an image. This form of operation is useful in determining edges of objects by contouring its perimeter [21].

Using edge-detection and simple grey-scaling, shape detection can be achieved to a high accuracy as shown in [22]. From contouring objects, its features (area, length, form factors etc.) could be analysed to group it into predefined shapes.

2.3 Past Relevant Literature

2.3.1 End to End Lane-Detection

Well recognised State-of-the-Art implementation of deep learning in autonomous lane correction include [6] by Nvidia in 2016. The proposed system composes of a CNN system that is trained and simulated to predict lane correction steering outputs for autonomous driving shown in *figure 2.7*. The network uses a 66×200 pixel sized input image as input into a driving simulator. The final

results was deep to have 98% accuracy in lane keeping.

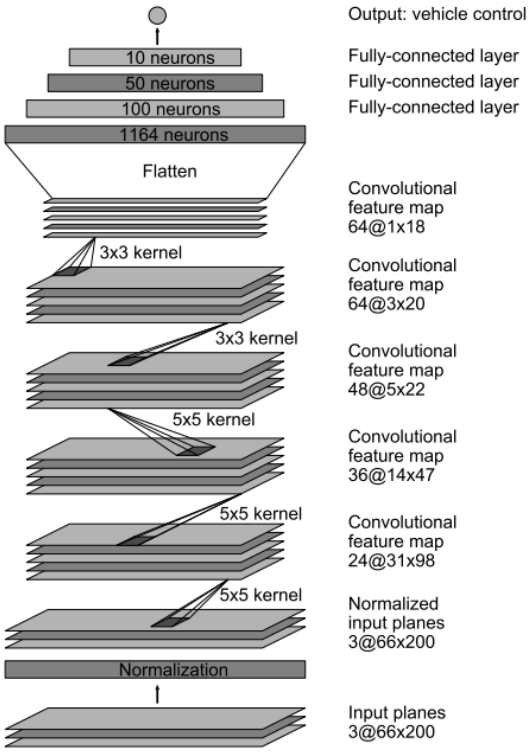

Figure 2.7: ***Nvidia Proposed CNN architecture for Autonomous Lane-Correction of vehicles [6]***

Similar studies and implementations of CNN lane-correction in vehicles include [23] [7] [24]. The general end to end driving methodology composes of data collection, training, and simulation through simple interface of trained CNN classifiers (*figure 2.8*)

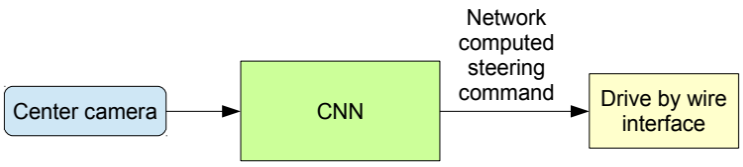

Figure 2.8: ***High level architecture overview of Lane-Detection system, Nvidia [6]***

Simulations were a implemented in real world road conditions. However, data images from the literature are limited to a few conditions: roads were either straight or turns, no evidence of lane changing [23]. Literature also states the

evident turning angle bias due to uneven spread of ”straight” and ”turn” images in the training data sets [6]. Lane markings such as dotted giveway lines, zebra crossings and broken lane markings were also inevident.

2.3.2 Traffic Sign Detection

The German Traffic Sign Benchmark study (GTSRB) in 2011 has a set a baseline to traffic sign recognition through The study of traffic sign detection through neural networks implemented by The German Benchmark [25] in 2011 has set a baseline to recent year literature. The deep learning model from [25] normalises images through image processing methods to a large training data sample size such as HOGs (histogram oriented graidents) and gray-scaling.

2.3.2.1 Pre-trained Classifiers

In 2018, the GTSRB baseline is improved on by [1] through Google's deep learning architecture *Inception-v3* with average accuracy of 99.09%. Inception-v3 is one of Google's pre-trained CNN allowing high accuracy of image detection proved in [1] [9].

Table 2.1: Inception-v3 architecture implemented for traffic sign recognition in [1]

Layers	Maps & Size	Filter Size	Layers	Maps & Size	Filter Size
input	3 & 299 × 299	-	mixed3	768 & 17 × 17	1 × 1,3 × 3
conv2d_1	32 & 149 × 149	3 × 3	mixed4	768 & 17 × 17	1 × 1,3 × 3,1 × 7,7 × 1
conv2d_2	32 & 147 × 147	3 × 3	mixed5	768 & 17 × 17	1 × 1,3 × 3,1 × 7,7 × 1
conv2d_3	64 & 147 × 147	3 × 3	mixed6	768 & 17 × 17	1 × 1,3 × 3,1 × 7,7 × 1
conv2d_4	80 & 73 × 73	1 × 1	mixed7	768 & 17 × 17	1 × 1,3 × 3,1 × 7,7 × 1
conv2d_5	192 & 71 × 71	3 × 3	mixed8	1280 & 8 × 8	1 × 1,3 × 3,1 × 7,7 × 1
mixed01	256 & 35 × 35	1 × 1,3 × 3	mixed9	2048 & 8 × 8	1 × 1,3 × 3,1 × 3,3 × 1
mixed1	288 & 35 × 35	1 × 1,3 × 3,5 × 5	mixed10	2048 & 8 × 8	1 × 1,3 × 3,1 × 3,3 × 1
mixed2	288 & 35 × 35	1 × 1,3 × 3	output	62 & 62 × 1	Classifier

Table 2.1 shows mapping architecture of Inception-v3 based CNN. The classifier intakes a size 299 by 299 input and goes through high density pipeline of con-

volutional mapping and feature extraction. The filter size for the feature kernels run down to 1 by 1 pixel size to reduce training speed [1].

Many other different pre-trained neural networks exists and open-source to the public [9]. The highest accuracy deep learning neural network is DenseNet-201 [9], however downsides include substantially increased training times and requirements for high power GPU and CPU.

2.3.2.2 Data Augmentation

Data augmentation is commonly applied to deep learning models to reduce bias and increase robustness in image classification [1] [26]. Common methods of augmentation include increasing 'bad data' sample size [25] and addition of noise through Gaussian filters [26]. This is a major advantage over traditional image processing techniques. Whereby algorithms such as colour histograms and shape detection would have lower rate of image recognition due to its mathematical constraints [21].

Figure 2.9: ***Data Augmentation through increasing training data sample size through 'bad data', [1]***

From [1], data augmentation is applied to traffic signs. This includes collecting images of traffic signs that have glare, angled and damaged signs, moreover applying filters and flipping images. It allows enlargement of the training data

sample size to reduce bias in the model prediction. This method increases classifier reliability by increasing the variation in training data spectrum.

3. Methodology

This section of the report outlines the experimental procedures, software setup, hardware requirements and methodological approaches to data validation from State-of-the-art literature.

3.1 Experimental Overview

The fourth aim of the project is to design and implement a high level autonomous system. The ultimate goal is to produce a fully autonomous driving system defined by level five autonomy from SAE standards shown in Appendix A.1. The proposed system simulates autonomous robot at a level three system-high level of automation under specific geographical circumstances [10].

3.2 Experimental Procedure

The overall experimental procedure and design consists of:

1. Design of the autonomous driving system flow
2. Data collection in EyeSim environment (images, steering angle data)
3. Pre-processing of training data (data augmentation)
4. Training of CNN classifiers through python-tensorflow and deep learning packages

5. Implementation of autonomous driving system to EyeSim for analysis and debugging

6. Repetition from 1-5

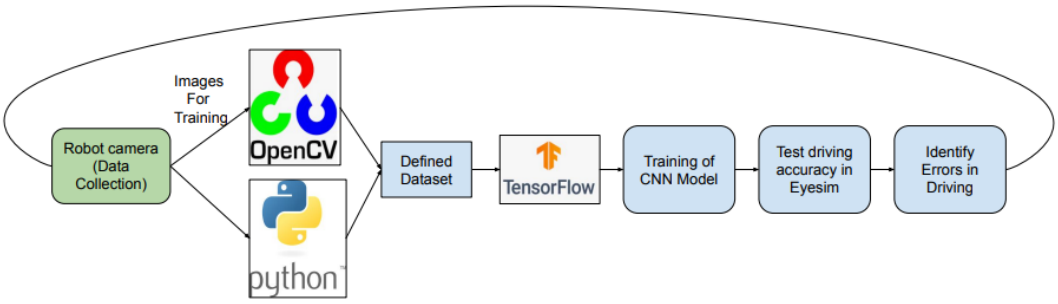

Figure 3.1: ***Experimental procedure to training and debugging robot***

3.3 EyeSim Simulation Environment

The EyeSim virtual environment replicates is used as the primary simulation program for the project. It provides a real-world driving scenario with road markings and traffic signs [16]. The system is able to imitate real-world physics such as obstacle collision and object inertia. Custom roads and traffic markings can be implemented to suit the experimental design.

The coding environment of the inbuilt mobile driving robots are primarily based-off C and C++ program language. Python can be run with the assistance of a Linux based programming shell such as through Big Sur[1], Ubuntu, or WSL[2]. From the *figure 3.2*, the necessary markings are developed in the EyeSim system. The main addition factor compared to past implementations of autonomous lane-detection include the stop lines, give way lines and zebra crossing markings. This allows the behaviour of lane-detection at different lane markings to

[1] Apple Mac operating system
[2] Windows Subsystem for Linux

Figure 3.2: ***EyeSim simulation environment with road signs, lane markings and driving robot***

be studied. As not many literature explores into the behaviour of model and lane-detection prediction at intersections that require decision making.

3.4 Software Environment Setup

3.4.1 Processor Speed

The main software environment setup is operated on the Apple M1 Mac book pro with ARM64 based CPU architecture. Computer processing speed and multi-core operation of the GPU (Graphics Processing Unit) is important to speed up the process of running CNN algorithms. The hardware and processor benchmarks provided in [27] states there is small difference in deep learning performance between medium and high thread CPU's (4 - 32 core CPU's). Therefore the following computer specifications listed in 3.1 are sufficient for CNN deep learning of the project.

Table 3.1: Computer specifications for project [2]

Part	Specifications
CPU	3.2GHz Clock 8-core CPU
GPU	8-core GPU 16-core Neural Engine
RAM	8GB

3.4.2 Python Environment

To integrate between the EyeSim system and terminal of the Apple Mac, Intel emulator must be installed to run all necessary python packages for the project. The emulator (Rosetta 2) allows interpretation of x86 bit compiled python packages to be used within the Apple Mac and EyeSim environment [28]. As of current time, the EyeSim software is not fully available for the M1 mac architecture. Therefore to prevent build errors, the author recommends to build the python and EyeSim environment through Rosetta 2 emulator.

The following python packages are implemented in past literature such as [23] [24] [1] [26] [29] and are necessary in running the deep learning and drive functions in EyeSim.

- Tensorflow - Python open source machine learning library
- Tensorboard - Tensorflow training visualisation tool
- Numpy - For array operations
- OpenCV - Python open source computer vision package library for image processing

These packages are installed under python 3 virtual environment. Additional packages such as *tensorflow-for-poets 2* is used to train Inception-v3 CNN. A full visualisation of the software setup can be seen in *figure 3.3*.

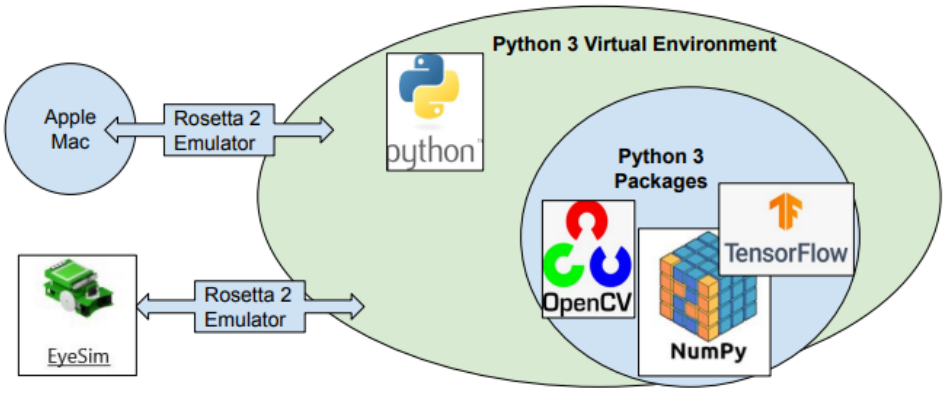

Figure 3.3: ***Software environment setup diagram***

3.5 Baseline Measures to Autonomous Driving

The validation of results is split into three different categories. Subsequently lane-detection, traffic recognition and autonomous driving (both lane and sign detection).

3.5.1 Traffic Sign Recognition Validation Methods

The validation of traffic sign recognition is commonly examined through training summaries of the deep learning model as shown in [9]. Additional factors such as training convergence rates and image validation time is compared. In the GTSRB [25], a 'new' data set of traffic signs are used to validate the accuracy of proposed deep learning models. For this project, training summaries of Inception-v3 will be used for analysis where the spread of data will be measured over 5000 epochs. Epochs are the number of iterations that passes through the training network that allows adjustments to hidden layer neuron weightings [30]. The training accuracy to epochs relation is calculated from the loss function. Percentage accuracy in data is inversely proportional to the cross-entropy loss function.

$$CrossEntropyLoss = [-\sum_{i=0}^{n} y_i \log y_{ihat}] \tag{3.1}$$

Equation 3.1 from [30] shows the cross-entropy relation to be the inverse logarithmic sum of input value y_i and the predicted output value y_{ihat}.

3.5.2 Lane-Detection Validation Methods

Many validation techniques are available for lane-detection and autonomous driving models. A qualitative form of measure is defined by the SAE J3016 standards [10]. The measures are set from level 0 with no autonomous features and safety to vehicles, up to level 5 reaching full autonomy that requires no human intervention [31].

Additional qualitative measures include visual representation of high probability failure points [7]. Failure points of the race track are visually updated during each iteration of the neural network model, shown in *figure* 3.4. This method allows identification of system reliability through visual cues (red- high failure rate, black- high reliability rate).

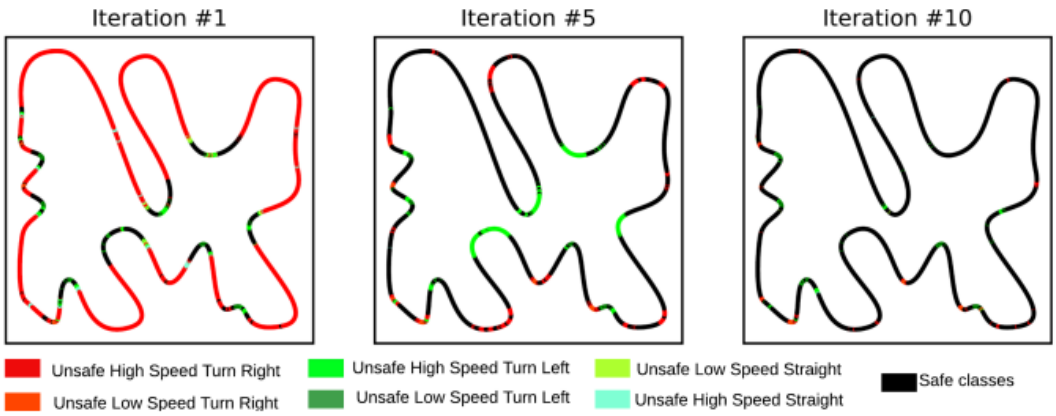

Figure 3.4: ***Visual data analysis method used in [7]***

From [6], a quantitative method of validation is proposed to calculate the per-

centage autonomy value.

$$Autonomy\,value = (1 - \frac{(number\,of\,interventions) \times 6\,seconds)}{elapsed\,time\,[seconds])}) \times 100 \qquad (3.2)$$

The value is based off the number of human interventions over the total driving time of the simulation vehicle. Interventions are due to errors caused by a wrong prediction of the CNN model, and defined by a human taking over the driving driving controls.

4. Design Implementation

This section outlines the overall experimental process proposed in *section 3* Methodology. Data collection and training progresses are aided with sample training images and flowcharts to explain important design in practices. To reduce complexity in interpreting the software architecture, table of summary with respect to software training files are presented in *table 4.4* with detailed descriptions.

4.1 Proposed Driving Model

The final high level driving model is shown in the *figure 4.1* . The system is based off driving architecture implemented in [6] back in *figure 2.8*.

During each frame of operation, the robot camera takes an image of resolution 320×240 pixels. A gray-scale filter is first applied to the image before storing it in a local file drive. The filtered image acts as the primary input to the driving system. This input image is segmented into two different regions of interest for image classification. The bottom half of the image for lane keeping, and the top half for traffic sign recognition. Both images are cropped into 320×96 pixel sized images. The sizing is further explained in *Section 4.4* Driving Implementation section.

Cropped images are inputted through to the two corresponding image classifiers to identify the correction action and output and of the vehicle. Output lane cor-

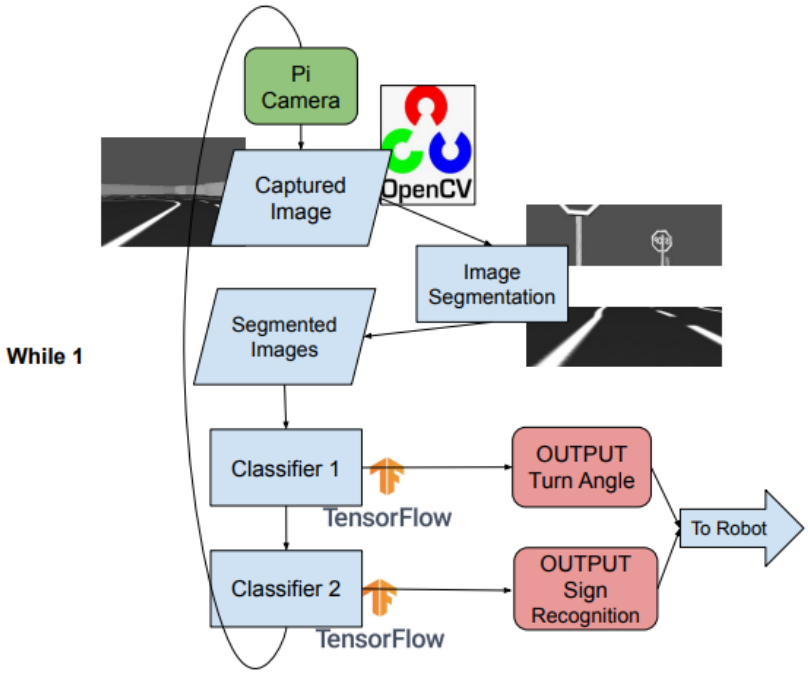

Figure 4.1: ***Propose high level architecture for autonomous driving vehicle***

rection classifier is the turning angle. Where as the resultant action of vehicle for example, stop at stop sign is predicted using Inception-v3 CNN. This loop is then repeated until the stop is terminated by the user.

4.2 Lane Detection Training

The trained model by [6] and [24] is used in the project to implement lane-detection. The training sequence is outlined in the flow chart below in *figure 4.2*.

4.2.1 Lane Data Collection

To efficiently collect training data for the robot, a while loop was used to iterate image capturing operations during manual operation of the robot. Simultaneously the robot turning angle was recorded and updated in a text file with the corresponding image label. For image training the robot was initially driven

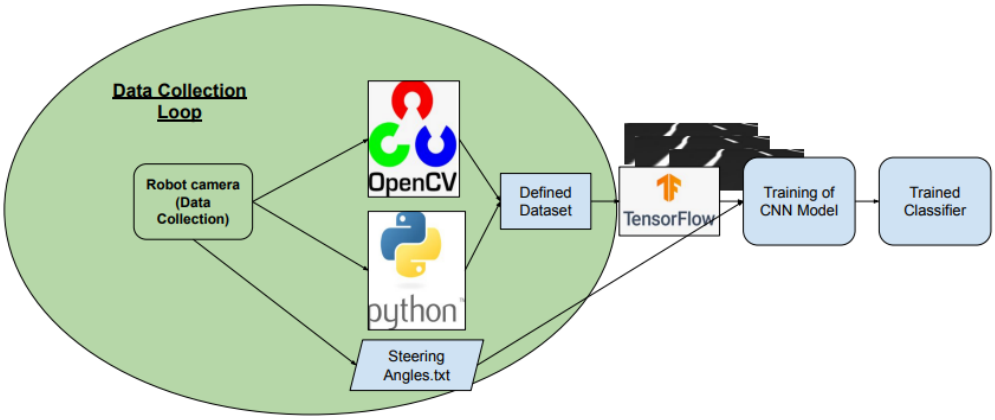

Figure 4.2: ***Data collection method for training sample collection for lane correction.***

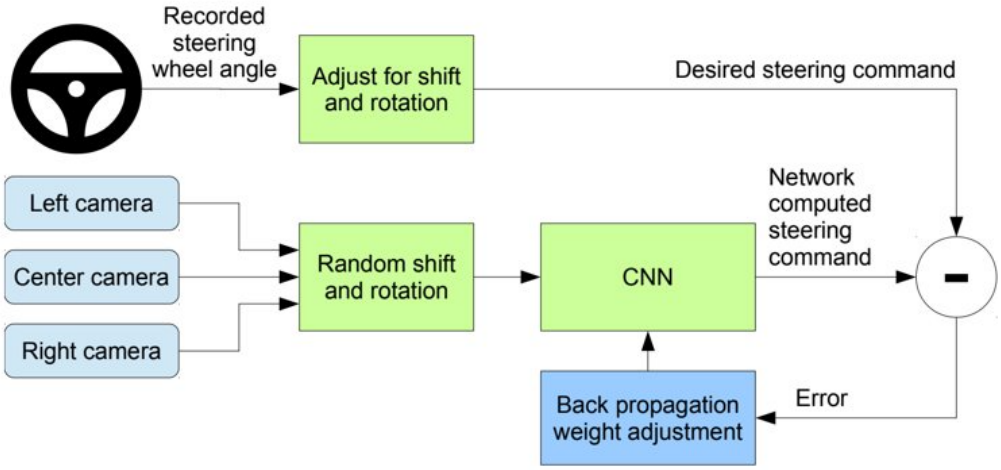

Figure 4.3: ***Data collection method proposed in [6]***

within simulation course twice collecting around 2000 PNG image set. Before each image is saved into the training directory, a gray-scale filter was applied to reduce RGB features and file size. Each image taken is around 4-9 Kilobytes.

To satisfy the input requirements defined by the training model constraint *figure 2.7* the images are cropped to 320×96 pixel resolution. The initial path taken by the robot during image collection is shown in *Appendix* A.2.

The list of different data sets taken for training for the experiment. As the file

size for each image is small, the largest data set of 6305 images took 74.3MB of storage space.

Table 4.1: Data sets taken for lane-detection training, *training data taken from different system

Data Set Ref.	No. of Images In Dataset	File Size (MB)
1*	2768*	90.7*
2	827	9.4
3	3168	37.6
4	6004	73.1
5	2205	27.4
6	4312	51.1
7	5881	68.4
8	6305	73.3

4.2.2 Lane Training

The processed data sets are trained with code adapted from [32]. By running the *train.py* in terminal command the tensorflow training model (model.py) is called upon. The training is ran over 30 epochs of the model. During each epoch iteration, weights of neuron layers are adjusted through *dataset.py and model.py* file. Where the final output classifier is a checkpoint file that allows the function to be called in tensorflow.

Training time varies with the sample size of the data sets. From training, it is shown to be a linear relationship from *figure 4.4*. The results do not show any exponential or non-linear correlation between the sample size of the training data and training time. This is good as it proves that even by doubling or quadrupling sample size of data sets, training is still feasible in terms of time and CPU/GPU strain.

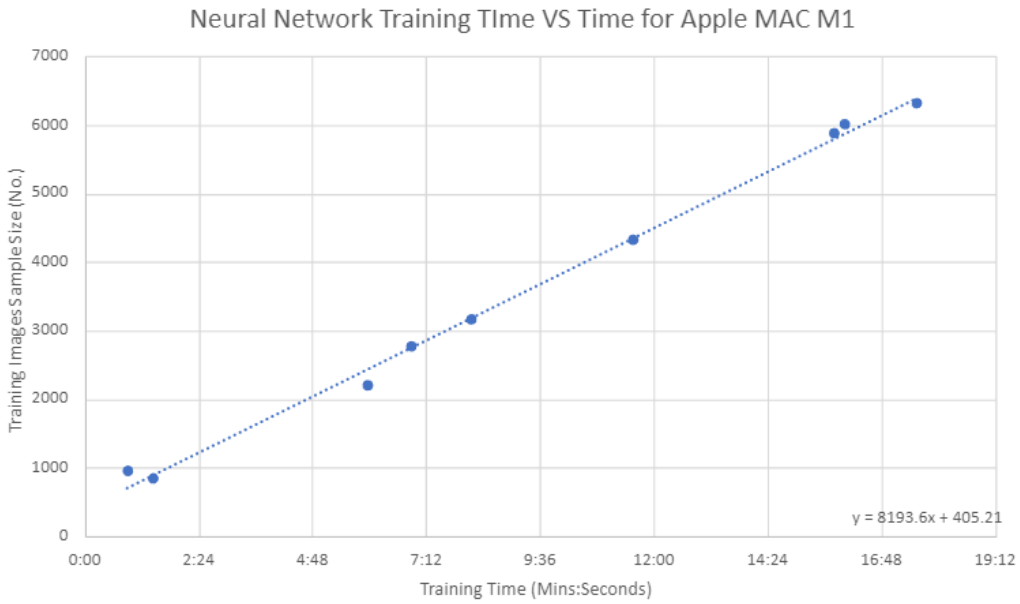

Figure 4.4: ***Neural network training time to data set sample size***

4.3 Traffic Recognition Training

4.3.1 Traffic Sign Data Collection

Compared to the lane-detection model, traffic sign recognition model required more pre-processing prior to training. The training code was adapted from the open-source Github repository (*tensorflow-for-poets2*) [33]. Requirements for training implementation include inputting the correct resolution data input, placing data sets in *classified buckets/ folders* and implementing sample sizes of at least 50 images per bucket.

Two different data sets were trained for the project. The first data set composed of 2 buckets, *"GO"* and *"STOP"* classification. The second data set broadened sign recognition network to 5 buckets, shown in *figure 4.5*. These buckets correlate to the *"GO"*, *"STOP"*, *"Parking"*, *"30Km/h"* and *"Speed Cancel"* signs respectively.

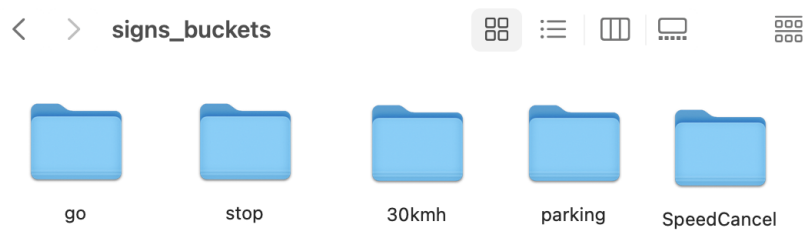

Figure 4.5: ***Buckets (files) with image data sets for neural network training***

Table 4.2: First set of collected training images

Bucket Label	Data Size	Percentage Portion
GO	764	81.2%
STOP	177	18.8%

Table 4.3: Second set of collected training images with 5 signs

Bucket Label	Data Size	Percentage Portion
GO	764	60.3%
STOP	177	14.0%
30Kmh	121	9.6%
Parking	66	5.2%
SpeedCancel	139	10.9%

4.3.2 Traffic Sign Recognition Training

As defined in *Table 2.1* by [1] and [34], Inception-v3 architecture requires 299× 299 sized inputs. The cropped 320×96 sized images are thereby resized before training use.

The collected training images were ran through *tensorflow-for-poets 2* package to generate a classifier file labelled as *retainedgraph.pb* [33]. To validate new images the *label.py* file calls upon the generated classifier file to predict the new image with an output string and prediction probabilities. The overall software flow for training and classification is shown in *figure 4.6*.

4.3.3 Data Augmentation Through Repetition

To increase the efficiency in collecting the training images, a python function allowing a manual 'start', 'stop' to the robot camera was necessary. This is different to collecting lane data where the robot could just run through the EyeSim road map without interruption. This manual operation allowed data augmentation at similar marking points to allow increased data pool for specific driving points. The function, named *"continue.py"*, was also helpful in debugging the classification model caused by low bucket frequency for both lane and traffic sign detection (discussed in *section 5*).

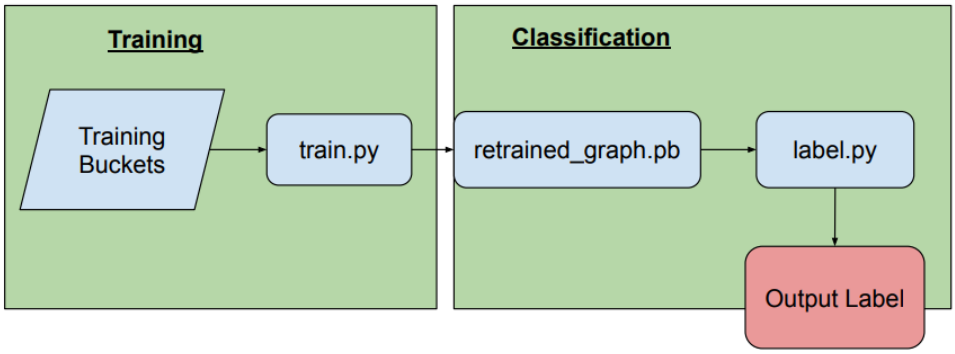

Figure 4.6: ***General software overflow of traffic sign training***

4.4 Driving Implementation

From *table 4.4*, the *cardrive.py* function implements a simple drive function. The EyeSim robot drives at a constant set speed while updating the turning angle at high frame rates. The robot code is tested by placing it parallel to the lane then running the driving file. Left lane driving is desired as defined by lane markings and training data where only left lane data is collected.

The proposed driving model from **section 4.1** was successfully implemented in EyeSim system using *cardriveautonomous.py*. Initially the models were ran indi-

vidually, one for lane-detection and one for traffic sign recognition. By applying a series pipeline to the robot, each classifier called is able to generate a predicted decision output for vehicle automation. Limitations to the model include the robot camera's FOV and processing speed explained in latter section 5.

4.5 Summary of Driving Functions

The following *table 4.4* is a summary of the main training files for the project. The categories are divided into:

- Data Collection, applies loops and image processing techniques to generate a training data set.
- CNN Training, applies files sourced from [33], [32] to train image classifiers.
- Autonomous Driving, simulation that applies trained classifier files to simple drive functions.

Table 4.4: List of major software files required for training and operating of deep learning robot.

Function	Function Category	File Name	Function Description
Lane Detection	Training	*collectimg.py*	Drives and collects images from EyeSim robot. Cropped images and steering angle text file is saved
Lane Detection	Training	*collectimg_continue.py*	Drives and adds images to existing data set, Allows user to pause the function recollect specific data points
Lane Detection	Training	*train.py from [32]*	Trains collected data set through CNN model, and produces output prediction file
Lane Detection	Classification	*model.ckpt*	Classifier model checkpoint file from CNN training output
Lane Detection	Classification	*cardrive.py*	Drives the model car without human interaction, uses checkpoint file to predict turning angle
Traffic Sign Recognition	Training	*collectimg_top.py*	Drives and collects images from EyeSim robot. Crops top section of images. Allows user to pause, and operate robot at desired data collection locations
Traffic Sign Recognition	Training	*retrain.py from [33]*	Takes collected bucket files for training, CNN architecture needs to be specified (Inception-v3)
Traffic Sign Recognition	Classification	*retrained_graph.pb*	Output prediction file for traffic sign recognition
Traffic Sign Recognition	Classification	*teststop.py*	Validation file for EyeSim traffic sign recognition, Calls the retrained file to predict new input image
Driving Implementation	(Autonomous Driving)	*autonomousdrive.py*	EyeSim robot driving file that implements both traffic sign recognition and lane-detection through series subroutines

5. Discussion and Results

This section outlines the overall performance and results of the EyeSim robot. Vehicle performance is analysed through proposed methodologies proposed in recent literature. Limitations and future insight to deep learning of mobile robots are also explored in this section.

5.1 Lane Detection Performance

From literature [6], lane detection was able to reach autonomous approximation of 98%. In other words, the vehicle produced 1 error every minute. Errors are

defined by the necessity of human intervention on the vehicle's steering angle due to prediction error. An example could be the vehicle going straight on a curved road.

From this project, final results are shown to have up to 100% accuracy in lane detection within a constrained environment. The visualisation of the path taken by the autonomous EyeSim robot is shown in *Appendix A.3*. It is seen that the trained robot is able to follow the lane markings accurately without deviating into other lanes.

5.1.1 Autonomous Driving Correction and Training Bias

As stated in the methodologies section 3, and from *Appendix A.2* training snapshot. The robot was ran in the left lanes, simulating a 'perfect' drive path for the training data. The resultant outcome for this left-lane training bias was exhibited in the output prediction classifier, where the robot would turn into the left lane if deviated or placed into the right lane.

Figure 5.1: EyeSim robot is placed in 'wrong lane'

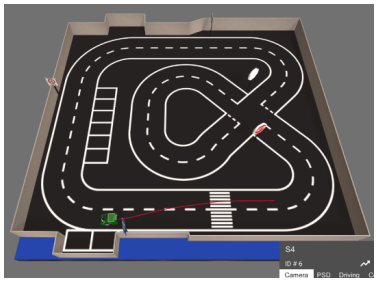

Figure 5.2: Robot exhibits lane change correction due to training bias

An additional factor that affects the lane change correction is **image cropping**. If the robot is placed in the right lane, by cropping the image, the trained robot is able to detect the segmented image as a "left turn". This is likely due

to a low cross-correlation value between images taken from a ”left turn” and what the camera sees in the right lane [35]. As a result the robot will exhibit this error correction behaviour 100% of the time on straight roads.

5.1.2 Data Augmentation: Deb-bugging

Autonomous lane detection classifier will not produce 100% accurate predictions by simply running iterations of driving data. Bias towards going straight is often shown in driving. The solution is to increase the proportion of turning data to remove bias in classifiers [6].

A major problem the EyeSim robot faced was turning at give-way lines and zebra crossings where it was suppose to continue straight *Appendix A.5* and *A.4*. The initial solution was to increase the amount of training data by running the robot through the course generating two times the sample data size (from 800 to 1500 training images). No significant performance were shown. The next step was to increase the training set to 3168 images, 4 times the original size. The results small improvements at zebra crossings lane centering. However, there were little to no improvement in approaching straight at give-way lines and intersections *Appendix A.6*.

Table 5.1: Frequency of failures at points of interest. Frequency range: (Always (100% of the time), Frequent (75%), Sometimes (50%), Seldom (<25%), No failures (0%))

Data Size	Failure in Lane	Failure in Intersection	Failure at Zebra Crossing
827	Sometimes	Frequent	Frequent
1500	Seldom	Always	Always
3168	No Failures	Always	Seldom

The training outcomes are presented in *Table 5.1* , where the failure rates of the

EyeSim robot is defined as a qualitative measure.

5.1.2.1 Lane Detection Optimisation

The solution to the issue of turning was to increase the percentage data sample size at the failure points identified from previous models. These high probability failure points are defined as **ROF** (Regions of Failure) and were identified as:

- Zebra crossings: correct robot action to move straight.
- Intersections: correct robot action to move straight. Includes give ways lines and stop lines.

The aim for this experiment is to determine an **optimal proportion** of normal ”Lane” marking to ”**ROF**” ratio to increase autonomous lane correction accuracy. Where ”Lane” marking is defined by vehicle driving straight or turning under normal lane conditions, this excludes lane markings such as zebra crossings and dotted give way lines.

From *Table 4.4*, the *collecontinue_continue.py* file was used to add on data from the previous models. Additional data was collected by manually moving the robot to the points of interests and running the image collection function. This step is repeated around 3 times at each intersection increasing data proportion of images at the points of interest. Each iteration records around 50 images shown in *figure 5.3*.
To test the trained classifiers, the robot was simulated both the outer and inner lanes on the left lane. Pass or fail was recorded with respect to the points of interest and its ability to correct turning angle (for example moving straight at zebra crossing).

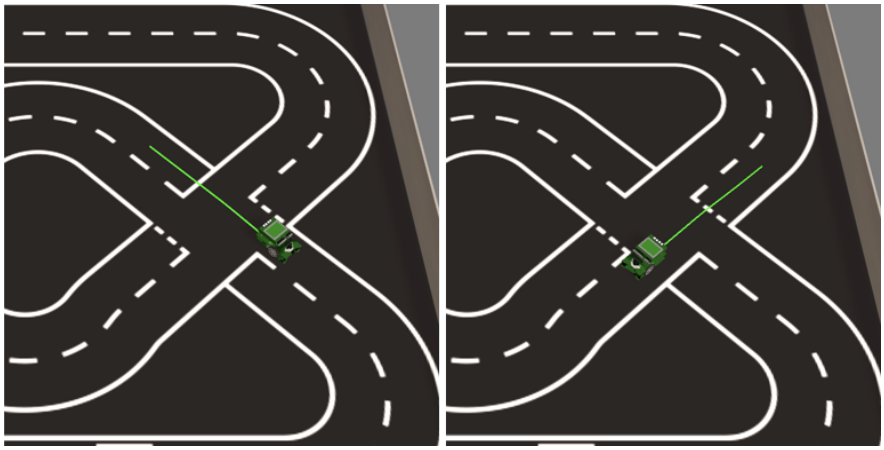

Figure 5.3: ***Increase data portion at regions with high failure rates***

5.1.2.2 Optimal Training Data Proportion

The summary of training data proportions are presented in the below tables. **ROF** proportions ranges from 20% to 38.1%. With 20% as the baseline value no additional ROF images given. Qualitative measures of autonomous driving performance are taken for each of the ROF sample proportions, displayed in *Table 5.2*.

Optimal Training: Sample Proportion 1		
Image Type	Data Size Approx.	Portion (%)
TOTAL	3168	100%
Lane	∼2519	71.5%
ROF (crossings, give ways)	∼649	20.5%

Optimal Training: Sample Proportion 2		
Image Type	Data Size Approx.	Portion (%)
TOTAL	2205	100
Lane	∼1476	66.9%
ROF (crossings, give ways)	∼729	33.1%

Optimal Training: Sample Proportion 3			Optimal Training: Sample Proportion 4		
Image Type	Data Size Approx.	Portion (%)	Image Type	Data Size Approx.	Portion (%)
TOTAL	4312	100	TOTAL	6305	100
Lane	∼2952	68.0%	Lane	∼3900	61.9%
ROF (crossings, give ways)	∼1360	31.5%	ROF (crossings, give ways)	∼2405	38.1%

Table 5.2: Frequency of failures at points of interest. Frequency range: (Always(100% of the time), Frequent (75%), Sometimes (50%), Seldom (<25%), No failures (0%))

Data Size	ROF Proportion	Failure in Lane	Failure in Intersection	Failure at Zebra Crossing
3168	20.5%	No Failures	Always	Sometimes
4312	33.1%	No Failures	Sometimes	No Failures
2205	31.5%	Seldom	Sometimes	No Failures
6305	**38.1%**	**No Failures**	**No Failures**	**No Failures**

Shown in *Table 5.2*, the most effect proportion of data for the EyeSim robot is shown to be approximately 38.1%. Using this image training proportion, the robot is able to maneuver through the course without any failure. This is true for both inner and outer lanes *Appendix A.3 and A.7*. In conclusion, by increasing the training sample size at high frequency at only the ROFs, the CNN model is able to adapt to complex environments. From this paper the ratio of ROF's to sample size of training data is shown to be approximately 38.1%.

5.1.3 Validation Through Foreign Environments

To test the validity of the trained model, the robot was run on a different course that is unknown to the model. The factors altered in the new environment include colour of the road, lane markings, lane width and turn angles.

The model was initially tested on the follow track adapted from [8].

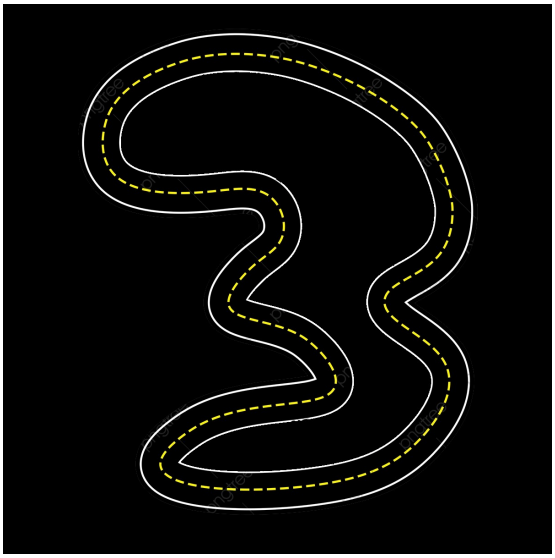

Figure 5.4: ***New Track for lane detection validation***

Features to the new track include:

- Darker road marking
- Wider lanes
- Sharper reflex angle turns
- Modified lane markings (wider, different colour)

The results of driving are suggestive to literature such as [6] and [23], where the robot is able to follow majority of the lane markings without failure until the critical turn angles displayed in *figure 5.5*.

Figure 5.5: ***ROF where EyeSim robot starts to fail***

From the training data the maximum and minimum angles are shown to be 35 and -25 degrees respectively. Where the distribution is made up of greater than

50% ”straight lane” data *figure 5.6*. Where from the figure the majority of the turning angles are distributed between -15 and +25 degrees.

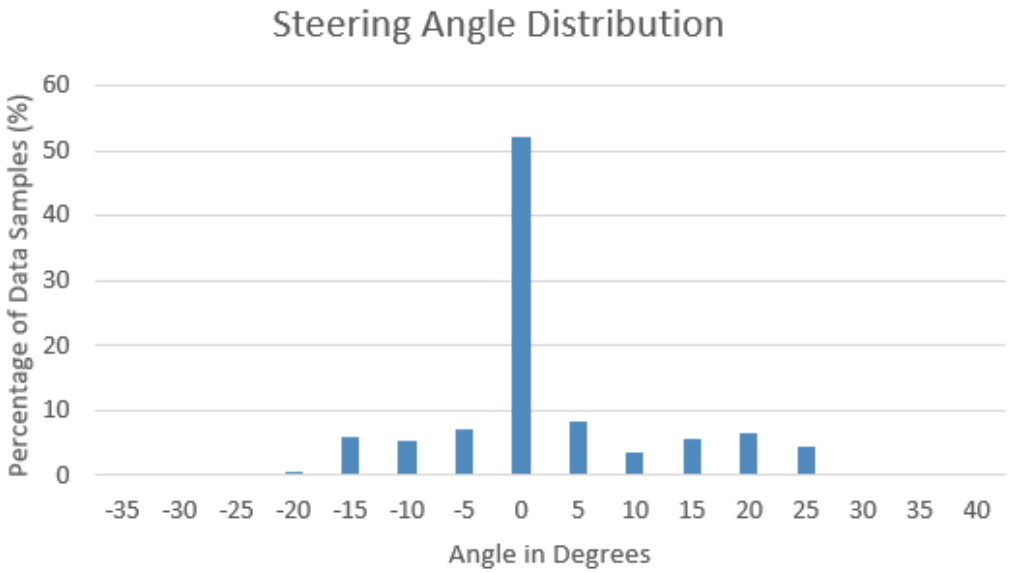

Figure 5.6: ***Robot's turning angles from training data***

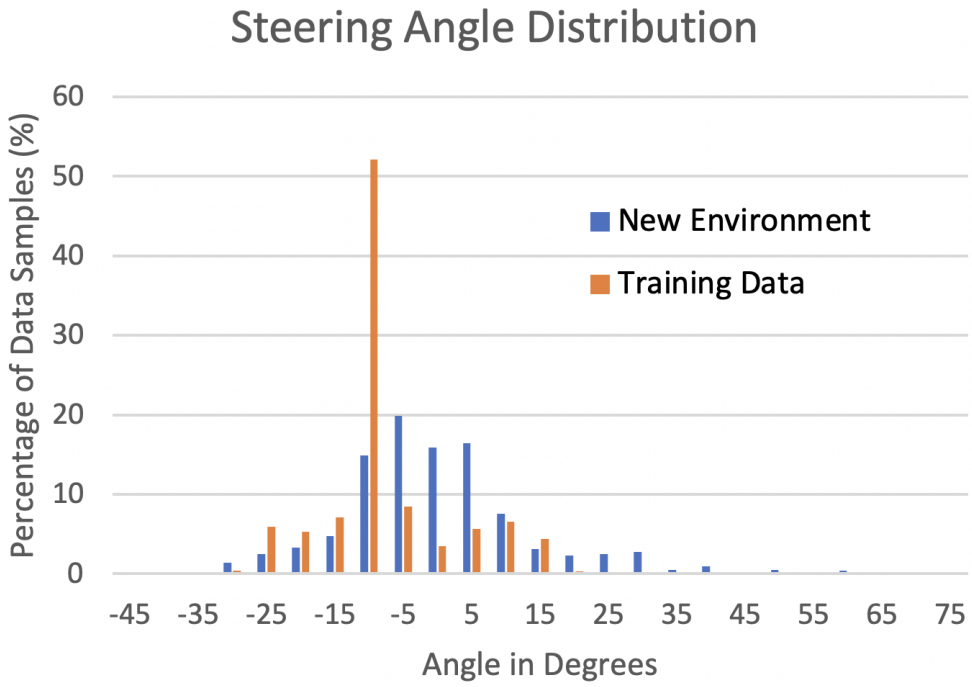

Figure 5.7: ***Robot's turning angles from training data compared to average angle distribution from the New Test Environment***

From *figure 5.7*, it could be seen that 3.72% of the required turning angles for the new environment lies above and below the training data range. Predictions outside the scope of training data is thereby not possible for the model resulting in failure to turn. To allow the model to adapt for this particular environment, higher proportion of data at reflex angle turns must be repeated to reduce bias [6].

On the other hand the EyeSim robot is able to adapt to a foreign environment with changes to lane width, markings and colour tone of road. This is proved with a different environment shown in *figure 5.8* where the distribution of turn angles are noticeably lower and correlate to within the range of trained CNN model for lane detection.

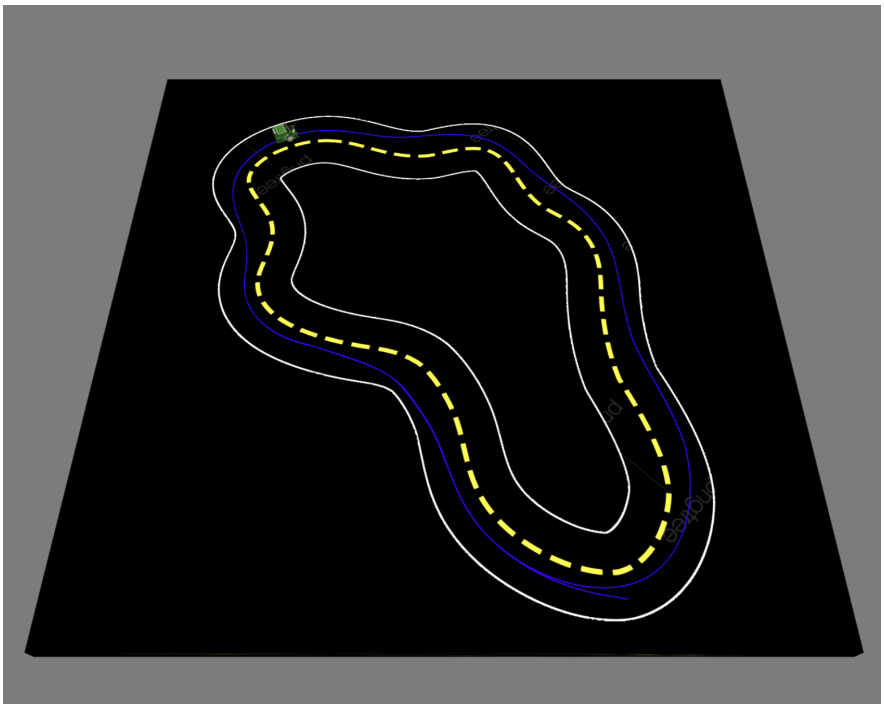

Figure 5.8: ***Successful adaptation of EyeSim robot to new environment (Race track adapted from [8]***

5.1.4 Traffic Sign Performance

Two major classifications should be performed by the vehicle during autonomous driving. This is defined by the vehicle's ability to recognise and stop at a stop sign or whether it is safe to move forward. For this, the system will have to differentiate between two sets of buckets. "Stop" and "Go". Further building of the model included 5 buckets, correlating to the 5 different traffic signs in the EyeSim racetrack. Two different traffic sign models were trained, with training results covered in the next section.

5.1.4.1 Training Accuracy

Using the data shown from *Table 4.2 and 4.3*, training results showed accuracies of 98.8% and 97.5% respectively with the Inception-v3 CNN architecture. This closely aligns to past literature averages of around $\sim$98% [1].
Appendix A.9 A.10

5.1.4.2 Inception-v3 Limitations

To test the trained model, validation images were collected and ran through the classifier. Tests showed high accuracy in predicting the type of sign in the FOV (Field of View) of the robot. Moreover, real images were inputted to test the model capacity. Depending on the image complexity, the model was able to correctly determine the corresponding traffic sign with a lower confidence interval *figure 5.10*

One major limitation to the Inception-v3 architecture is the processing speed of the GPU. Evaluation time of images take anywhere from 1-1.8 seconds (0.5-1 frames per second) varying with image complexity. From [9] the performance of different pre-trained neural networks, with Inception having GPU memory issues from expensive GPU computations. For future works, it is suggested that

Figure 5.9: ***Successful recognition of real life stop sign with low prediction confidence interval.***

a different pre-trained classifier is selected for traffic sign classification. These include networks such as MobileNetV2 and SqueezeNet proposed by [9].

Type of DLNN	Average Training Accuracy [-]	Average Verification Accuracy [-]	Average Training Time of Single Net [h]	Remarks
AlexNet	0.998	0.871	0.18	–
DenseNet-201	1	0.979	10.16	GPU low memory
GoogLeNet	1	0.937	0.18	–
Inception-ResNetV2	0.995	0.949	1.34	GPU low memory
InceptionV3	1	0.952	0.50	GPU low memory
MobileNetV2	1	0.966	0.36	–
NASNetMobile	1	0.958	1.87	–
ResNet-18	1	0.971	0.21	–
ResNet-50	1	0.964	0.37	–
ResNet-101	0.997	0.899	0.94	–
ShuffleNet	1	0.953	0.29	–
SqueezeNet	0.999	0.931	0.19	–
VGG-16	0.938	0.717	1.36	GPU low memory
VGG-19	1	0.695	3.81	GPU low memory
Xception	1	0.961	0.51	GPU low memory

Figure 5.10: ***Performance of Pre-trained DLNNs [9]***

5.1.5 Autonomous Driving Performance

The proposed autonomous driving vehicle proposed in **Section** 3 Methodology was successfully implemented. A simple *for loop* was implemented by the robot to stop when a stop signs are detected. Two trained models from **Section 5.1.4** and **5.1.2.1** were pipe-lined. The EyeSim robot was able to drive within the lanes and stop at the stop sign.

5.1.5.1 Autonomous Driving Limitations

Output model is very restricted in terms of driving performance. The primary factor that makes the model hard to debug is the system operation time of inception-v3 classifier. The slow operation time results in long waiting times for the robot between each operation decision, often ending up with slow stop start drives between each frame.

The second factor is the FOV of robot camera. The angle placement of the robot camera is relatively low, therefore the traffic signs is not visible when the vehicle is at the stop line. Future work may include integration of depth perception through computer vision and deep learning such as from [36] [37] [38].

Final factor that restricts the model shown from literature and project outcome is adaption to foreign environments. As shown from lane correction models, addition of unknown factors to image lane-detection and image classification will often result in system failure. Where one way to improve the CNN model is to implement a wider spectrum of training data.

6. Conclusion

6.1 Conclusion

The project scope covers autonomous operation of vehicles through the UWA EyeSim virtual environment. The main topics of research included lane-detection and traffic sign recognition through State-of-the-art methods of deep learning such as CNN. The aim of the project was to implement a high accuracy autonomous driving model proposed in section 3 of the report.

The final model showed success in autonomous driving within a confined environment through EyeSim simulations. Through data augmentation and balancing ROF proportions in training data, the system is able to adapt to foreign environments with a 100% autonomy value as defined by [6]. However this was limited to a large number of constraints such as driving angles, colour and lane markings of the road.

The model was also able to detect and differentiate different traffic signs with high accuracies. Results showed values of up to 98.8% through inception-v3 model. Limitations to traffic sign classification included low frame rate and model prediction time.

Deep learning algorithms provides a cheap reliable solution to autonomous driv-

ing. In many cases the deep learning method surpasses traditional computer vision techniques of autonomous driving. However, many opportunities to further improve deep learning of vehicles due to the lowered performance reliability in foreign environments.

6.2 Future Works

According section 5.1.4 and 5.1.2.1 of the report, many limitations exist with the current models autonomous driving and deep learning. Constraints to the model is caused by computational times of CNN architecture like Inception-v3 and serial pipelines of the driving model.

Future studies should include research into image classification models with reduced computational time. A system that implements parallel operation of CNN models is also desired to increase safety integrity levels and reliability of the robot. Additionally, due to the camera's field of view, traffic signs are not recognised at the exact location of the stop line. Therefore, distance prediction through deep learning and image processing would highly improve the driving experience to traffic sign detection.

A. Appendix: Figures

A.1 SAE Levels of Automation

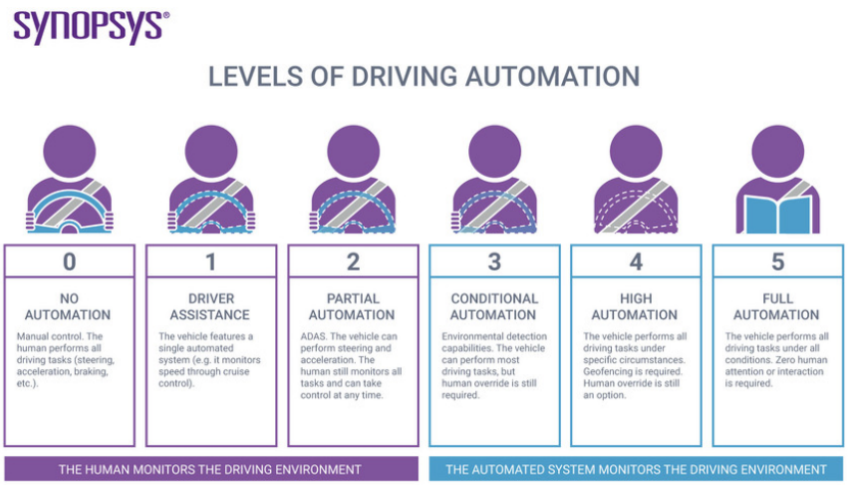

Figure A.1: *SAE levels of Automation by [10]*

A.2 Initial Lane-Detection Training Path

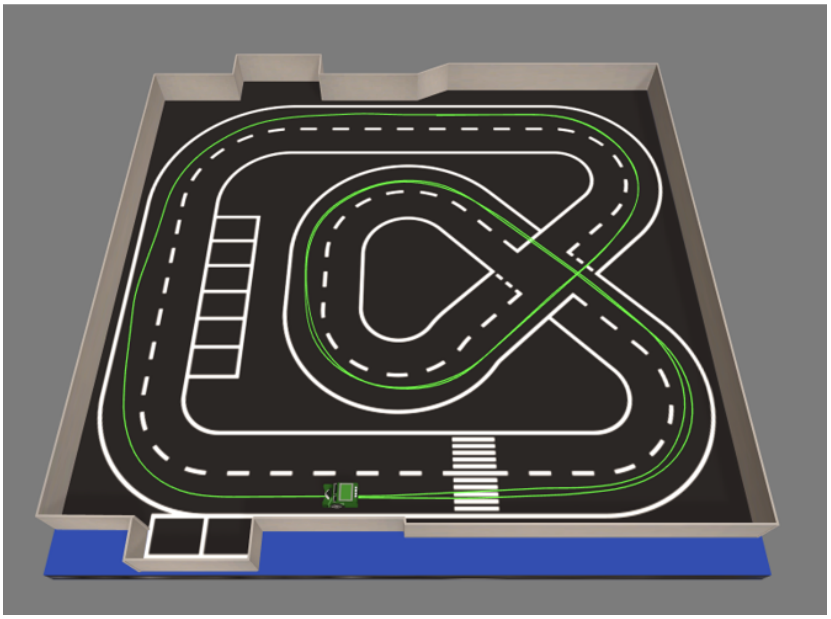

Figure A.2: *Path taken by robot during image collection for deep learning*

A.3 Lane-Detection Validation Run

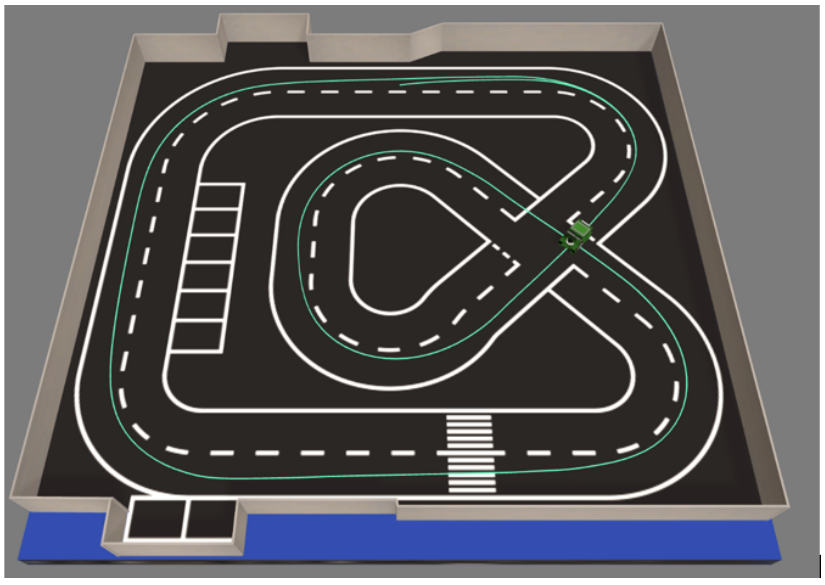

Figure A.3: *Path taken by CNN trained Autonomous robot in EyeSim within the same training environment*

A.4 Lane-Detection failure: 827 Images

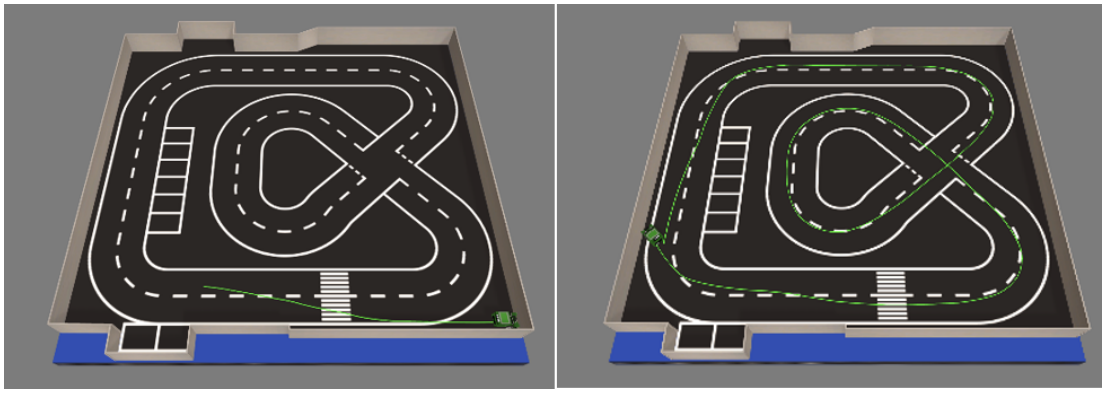

Figure A.4: *Initial trained robot with 800 images and no data repetition*

A.5 Lane-Detection failure: 1500 Images

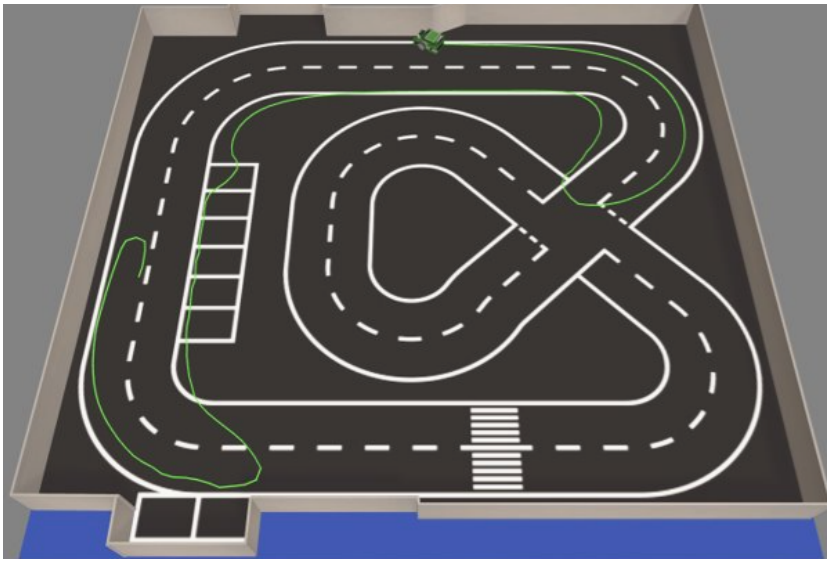

Figure A.5: *Initial trained robot with 1500 images and no data repetition*

A.6 Lane-Detection Failure: 3168 images

Figure A.6: *Path taken by CNN trained Autonomous robot with 3000 images and no data repetition*

A.7 Lane Detection CNN with 38.1% ROI proportion

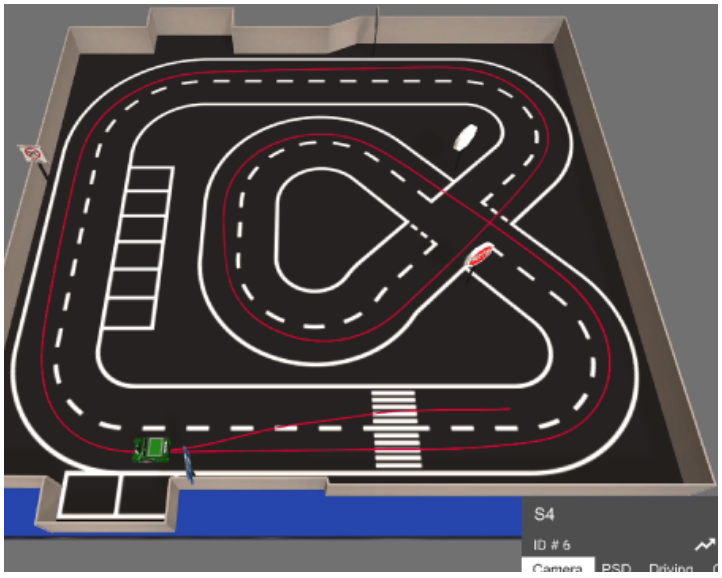

Figure A.7: *Successful lane detection with ROF*

A.8 Track use to Test Adaptability of CNN Model for Lane correction

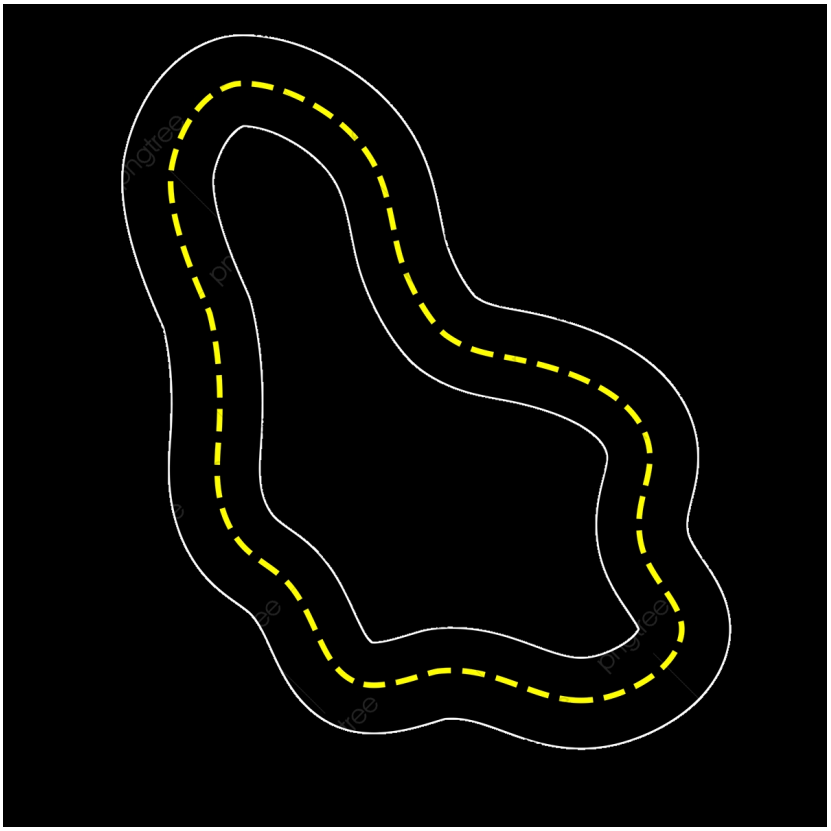

Figure A.8: *Test track adapted from [8]*

A.9 Inception Traffic Sign Training Results

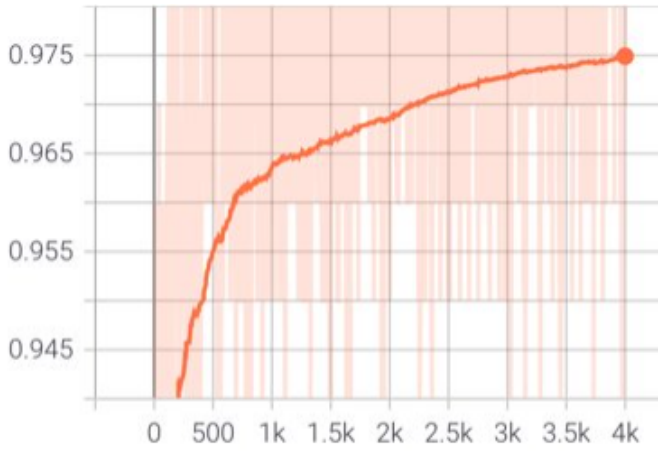

Figure A.9: *Tensorflow training summary for 5 traffic signs*

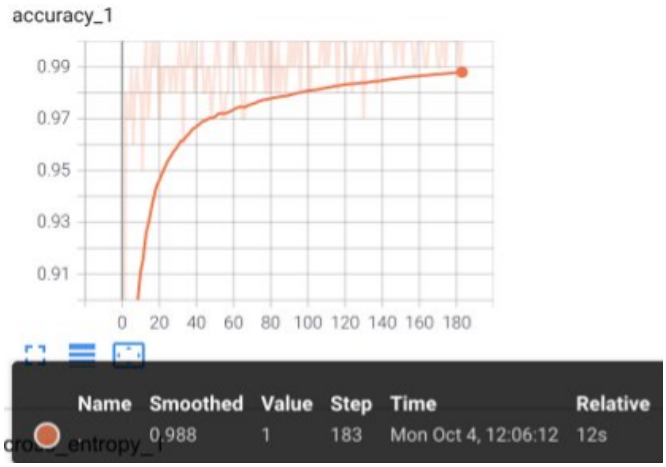

Figure A.10: *Tensorflow training summary for 2 traffic signs*

A.10 Automation of EyeSim Robot- Traffic Sign and Lane Detection

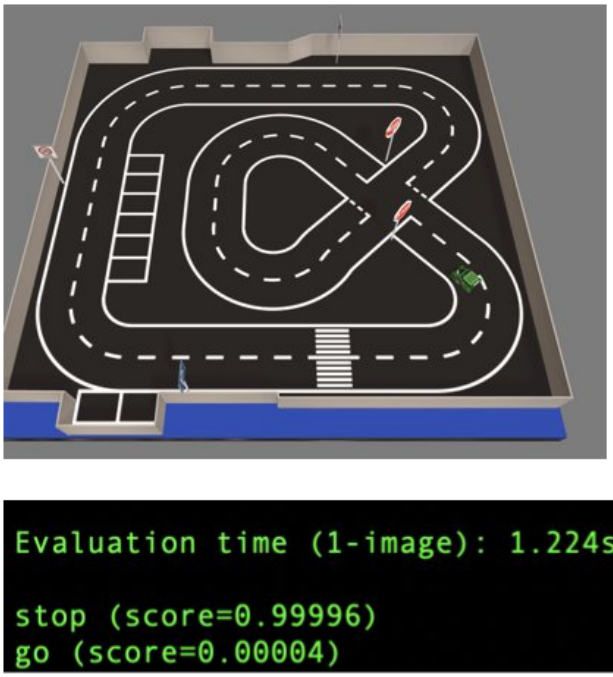

Figure A.11: *Automated mobile robot*

References

[1] Chunmian Lin, Lin li, Wenting Luo, Kelvin Wang, and Jiangang Guo. Transfer learning based traffic sign recognition using inception-v3 model. *Periodica Polytechnica Transportation Engineering*, 47, 08 2018.

[2] Apple, mac m1 spec. `https://support.apple.com/kb/SP824?locale=en_AU`. [Online] Accessed: 2021-10-06.

[3] Sun-Chong Wang. *Artificial Neural Network*, pages 81–100. Springer US, Boston, MA, 2003.

[4] How convolutional neural networks accomplish image recognition? `https://www.kdnuggets.com/2017/08/convolutional-neural-networks-image-recognition.html`. Accessed: 2021-04-13.

[5] Simon S. Haykin. *Neural networks and learning machines*. Prentice Hall, New York, 3rd ed. edition, 2009.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs

Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[7] Yunus Bicer, Ali Alizadeh, Nazim Kemal Ure, Ahmetcan Erdogan, and Orkun Kizilirmak. Sample efficient interactive end-to-end deep learning for self-driving cars with selective multi-class safe dataset aggregation. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov 2019.

[8] pngree. Racetrack image png. `https://pngtree.com/so/race-track`. [Online] Accessed: 2021-10-06.

[9] Piotr Szymak, Pawel Piskur, and Krzysztof Naus. remote sensing the effectiveness of using a pretrained deep learning neural networks for object classification in underwater video. *Remote Sensing*, 12, 09 2020.

[10] Synopsys INC. The 6 levels of vehicle autonomy explained.

[11] Y. n. Dong and G. s. Liang. Research and discussion on image recognition and classification algorithm based on deep learning. In *2019 International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)*, pages 274–278, 2019.

[12] Simon S. Haykin. *Neural networks and learning machines*. Prentice Hall, New York, 3rd ed. edition, 2009.

[13] D. Goularas and S. Kamis. Evaluation of deep learning techniques in sentiment analysis from twitter data. In *2019 International Conference on Deep*

Learning and Machine Learning in Emerging Applications (Deep-ML), pages 12–17, 2019.

[14] Michael P Ekstrom. *Digital image processing techniques*, volume 2. Academic Press, 2012.

[15] T. Gayathri Devi, P. Neelamegam, and S. Sudha. Image processing system for automatic segmentation and yield prediction of fruits using open cv. In *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, pages 758–762, 2017.

[16] Eyebot7-userguide. `https://robotics.ee.uwa.edu.au/eyebot/EyeBot7-UserGuide.pdf`. Accessed: 2021-04-06.

[17] Pariwat Ongsulee. Artificial intelligence, machine learning and deep learning. In *2017 15th International Conference on ICT and Knowledge Engineering (ICT KE)*, pages 1–6, 2017.

[18] John Slavio. *Deep learning and artificial intelligence : a beginners' guide to neural networks and deep learning*. [CreateSpace Independent Publishing Platform], San Bernadino, CA, 2017.

[19] Jan. Drugowitsch. *Design and analysis of learning classifier systems : a probabilistic approach*. Studies in computational intelligence ; v. 139. Springer, Berlin, 2008.

[20] *Digital image processing techniques*. Computational techniques ; v. 2. Academic Press, New York, 1984.

[21] T. Gayathri Devi, P. Neelamegam, and S. Sudha. Image processing system for automatic segmentation and yield prediction of fruits using open cv. In *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, pages 758–762, 2017.

[22] T. Gayathri Devi, P. Neelamegam, and S. Sudha. Image processing system for automatic segmentation and yield prediction of fruits using open cv. In *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, pages 758–762, 2017.

[23] Zhilu Chen and Xinming Huang. End-to-end learning for lane keeping of self-driving cars. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1856–1860, 2017.

[24] Sully Chen. How a high school junior made a self-driving car, Dec 2018.

[25] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: A multi-class classification competition. In *The 2011 International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

[26] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pages 117–122, 2018.

[27] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pages 99–104, 2016.

[28] Apple. If you need to install rosetta on your mac. https://support.apple.com/en-us/HT211861. [Online] Accessed: 2021-10-06.

[29] I. Kilic and G. Aydin. Traffic sign detection and recognition using tensorflow' s object detection api with a new benchmark dataset. In *2020 International Conference on Electrical Engineering (ICEE)*, pages 1–5, 2020.

[30] Saahil Afaq and Smitha Rao. Significance of epochs on training a neural network. *International Journal of Scientific & Technology Research*, 9:485–488, 2020.

[31] SAE. Sae standards news: J3016 automated-driving graphic update. https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic. [Online] Accessed: 2021-10-06.

[32] Sully Chen. Sullychen/autopilot-tensorflow. https://github.com/SullyChen/Autopilot-TensorFlow. [Online] Accessed: 2021-10-06.

[33] Mark Daoust. googlecodelabs/tensorflow-for-poets-2. https://github.com/googlecodelabs/tensorflow-for-poets-2. [Online] Accessed: 2021-10-06.

[34] Xiaoling Xia, Cui Xu, and Bing Nan. Inception-v3 for flower classification. In *2017 2nd International Conference on Image, Vision and Computing (ICIVC)*, pages 783–787. IEEE, 2017.

[35] Feng Zhao, Qingming Huang, and Wen Gao. Image matching by normalized cross-correlation. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 2, pages II–II, 2006.

[36] Li Bing, Xu De, Feng Songhe, Wu Aimin, and Yang Xu. Perceptual depth estimation from a single 2d image based on visual perception theory. In Yueting Zhuang, Shi-Qiang Yang, Yong Rui, and Qinming He, editors, *Advances in Multimedia Information Processing - PCM 2006*, pages 88–95, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[37] Ashfaqur Rahman, Abdus Salam, Mahfuzul Islam, and Partha Sarker. An image based approach to compute object distance. *International Journal of Computational Intelligence Systems*, 1:304–312, 03 2012.

[38] Zhihao Chen, Redouane Khemmar, Benoit Decoux, Amphani Atahouet, and Jean-Yves Ertaud. Real time object detection, tracking, and distance and motion estimation based on deep learning: Application to smart mobility. In *2019 Eighth International Conference on Emerging Security Technologies (EST)*, pages 1–6, 2019.