Training Neural Networks to Pilot Autonomous Vehicles: Scaled Self-Driving Car

Jason Zisheng Chang
Bard College

Training Neural Networks to Pilot Autonomous Vehicles: Scaled Self-Driving Car

Zishneng Jason Chang,

The Division of Science, Mathematics, and Computing
of
Bard College

May 2018

Contents

1 Introduction **6**

1.1 Purpose . . . 9

1.2 Scope . . . 10

2 Background **11**

2.1 Computer Vision and Autonomous Cars . . . 11

2.1.1 ALVINN . . . 11

2.1.2 NVIDIA end to end learning . . . 12

2.2 Neural Networks . . . 12

2.2.1 Activation Function . . . 13

2.3 Training . . . 14

2.3.1 Error Functions . . . 14

2.3.2 Overfitting . . . 15

2.4 Convolutional Neural Networks . . . 16

2.4.1 Dense Layer . . . 18

2.5 Recurrent Neural Networks . . . 18

2.5.1 Long Short-Term Memory . . . 19

2.6 Models . . . 20

3 Implementation **25**

3.1 System Overview . . . 25

3.2 Donkeycar Platform . . . 27

3.2.1 Datastore and Records . . . 28

4 Datasets and Training **30**

4.1 Tracks 30

4.1.1 8-Track 30

4.2 Circuit 32

5 Methods **34**

5.1 Model Training 34

5.2 Steering Accuracy 34

5.3 Position Tracking 35

5.3.1 Path Averaging and Comparison 37

5.4 Track Trials 38

6 Results and Analysis **39**

6.1 Steering Accuracy 39

6.2 Raw Model Paths 42

6.3 Averaged Model Paths 44

6.3.1 Human Driver Path 45

6.3.2 Linear Model 46

6.3.3 Categorical 47

6.3.4 RNN 48

6.3.5 RNN Categorical 49

6.4 Circuit Path Deviation 50

7 Conclusion **52**

Abstract

This project explores the use of deep convolutional neural networks in autonomous cars. Successful implementation of autonomous vehicles has many societal benefits. One of the main benefits is its potential to significantly reduce traffic accidents. In the United States, the National Highway Traffic Safety Administration states that human error is at fault for 93% of automotive crashes. Robust driverless vehicles can prevent many of these collisions. The main challenge in developing autonomous vehicles today is how to create a system that is able to accurately perceive and process the world around it. In 2016, NVIDIA successfully trained a deep convolutional neural network to map raw images from a single front-facing camera into steering commands. Today, automotive companies such as Google's Waymo, and Tesla's Autopilot, utilize deep convolutional neural networks to control their autonomous vehicles. The goal of this project is to evaluate how well a recurrent neural network and categorical output perform when combined with NVIDIA's platform. These models' performances are then evaluated on a scaled self driving car and compared to a human driver. NVIDIA's model combined with a RNN is able to keep the car within 6.1 cm of a human driver's path.

Acknowledgements

Thank you to Professor Sven Anderson for mentoring me through this senior project and advising me throughout the semester. Another thank you to my friends for getting me through this semester. And to Patrycja Witanowska for joining me through these highs and lows, and pushing me to be my best.

1 Introduction

The goal of developing an autonomous car is to create a vehicle that navigates from point A to B without any human intervention. While the idea of self-driving vehicles have existed since the the 20s [14], the idea of a intelligent vehicle being able to sense and perceive did not yet exist. As computers were not commonplace, solutions to autonomous vehicles involved infrastructure built to guide them. An example of early developments during the 50s was that of GM and RCA's automated highway prototype (Figure 1) [1]. This prototype housed magnets that tracked steel cables embedded in the road. However, the cost of installing these steel cables proved infeasible, and the magnetic system it utilized to steer was unstable at high speeds. During a demonstration, a reporter noted the vehicle's inability to detect vehicles ahead of it, relying on the demonstrator to stop before colliding into a stopped car ahead [8]. The lack of intelligence to guide the vehicle hindered any further development.

1. INTRODUCTION

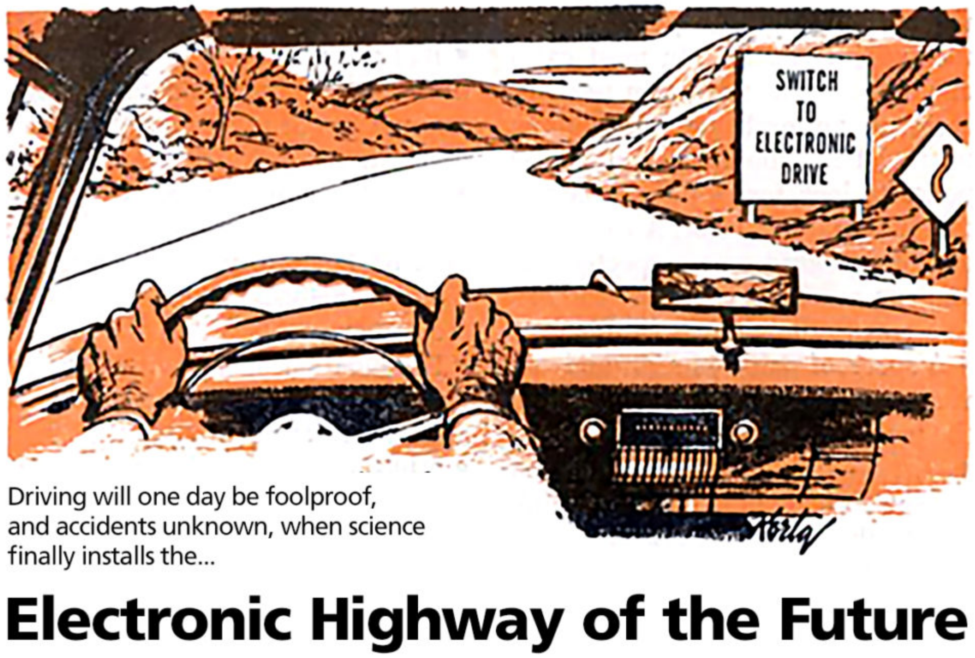

SWITCH TO ELECTRONIC DRIVE
Driving will one day be foolproof, and accidents unknown, when science finally installs the...
Electronic Highway of the Future

Figure 1: Advertisement for GM's "Electronic Highway of the Future"

During the 1970s the digital revolution sparked the development of intelligent autonomous systems [1]. Robotics researchers in the 70s aimed to reverse engineer intelligent systems found in animals: sensing, processing, and reacting. In 1979 the Stanford Cart demonstrated the ability for a robot to complete these tasks. In this project, the cart navigated around obstacles by processing stereoscopic images captured by a swiveling camera [13]. This project laid the foundations for developing cars that could perceive highways, intelligently process the information, and react from their perceptions.

Ernst Dickmann's VaMoRs Mercedes van (Figure 2) successfully demonstrated the application of computer vision and machine intelligence in autonomous vehicles. Notably, the van was able to drive itself without relying on any external guidance infrastructure. This vehicle was capable of travelling on freeways at speeds above 130 km/h. Dickmann's system used Kalman filters to estimate the position of the vehicle through a combination of inertial measurements and images [5]. Feature extractors extracted edges and areas of similar textures to determine vehicle direction. This project pioneered the beginning of intelligent autonomous vehicles.

Figure 2: Dickmann's VaMoRs Autonomous Van

In 2005 The United States Defense Advanced Research Projects Administration (DARPA) sought to spur autonomous vehicle development in the United States. The result was the Grand Challenge, a competition that challenged dozens of teams to to create an autonomous vehicle that could navigate a 150 mile course in California's Mojave desert. During its inaugural competition in 2005, no team finished the course. By the fall of 2005, 5 vehicles out of 195 managed to cross the finish line. In 2007 DARPA shifted their course to a staged city environment. Out of the 11 teams, 6 completed the course [4]. An area of needed improvement was computer vision. Autonomous vehicles require methods of recognizing asphalt, other vehicles, and lane markings. Competitors in DARPA's Grand Challenge solved these patten recognition tasks with custom made feature extractors and classifiers.

The breakthrough of convolutional neural networks(CNNs) is its unique ability to learn and develop feature extractors from training examples. This enables the neural network to find the most effective feature extraction filters [17]. The first demonstration of its application in autonomous vehicles was Pomerleaus Autonomous Land Vehicle in Neural Networks (ALVINN) in 1989 [15]. ALVINN demonstrated a convolutional neural network that learned to steer a car on public roads, capable of speeds up to 3.5 mph. However, during the 20th century the true potential of CNNs were limited by the processing power required. Today, technological advances have enabled researchers to access resources capable of applying far more data and computational power to the task. For reference, ALVINN's architecture consisted of a single hidden layer back-propagating network. Neural networks today can be composed of dozens to hundreds of hidden layers. These multi-layered neural networks are referred to as deep neural networks.

In recent years deep convolutional neural network models have become commonplace in autonomous vehicles. The Tesla Model S is known to use deep neural networks for vision based obstacle detection and avoidance in their electric vehicles [2]. NVIDIA's self driving car, DAVE-2 is capable of driving on public roads using deep neural networks. This project focuses on expanding upon NVIDIA's network detailed in their paper [3]. This project compares a categorical model vs. single output model, and a single state convolutional neural network to a recurrent neural network.

Variations of these networks are evaluated using a scaled self-driving car platform. The performance of the these neural network models is assessed by the ability of the neural network to generalize to new tracks. The aim of using this platform is to evaluate performance of these neural networks in a real environment rather than a simulated one.

1.1 Purpose

This senior project explores implementations of deep convolutional neural networks for autonomous vehicles. All implementations are modified versions of NVIDIA's published convolutional neural network [3]. This project explores the following variations:

- Categorical Output vs. Single Output
- Recurrent Neural Network vs. Single State Convolutional Network

NVIDIA's network is a single state convolutional neural network that does not retain any internal information from a sequence of inputs, acting only on current information. The goal of using a recurrent neural network is to address this shortcoming. A recurrent neural network forms connections between units along a sequence, enabling the network to remember a sequence of inputs, making them applicable to the task of self driving cars.

The second variation this project explores is implementation of a model with multiple outputs: steering angles mapped to bins according to their output angles. This system allows the network to assign probabilites to a vector of steering angles. This could possibly enable the vehicles to generalize more effectively. These models are evaluated using a open source scaled self driving car platform, known as Donkeycar.

1.2 Scope

While this project aims to build a autonomous vehicle, the limitations of the project are listed here:

- The car will not be considered to be fully autonomous. The scaled car is only able to predict steering angles.
- The physics of a remote control car by no means simulates the driving dynamics of a real life car.
- Speed will be kept at such a low rate that slippage and other physical factors can be ignored.

2 Background

This section describes the information needed before proceeding into the project.

2.1 Computer Vision and Autonomous Cars

The most critical feature of a autonomous vehicle is its ability to detect and avoid obstacles around it (pedestrians, traffic cones, or buildings). In 1979 the Stanford Cart demonstrated the ability of a robot to find objects in a image and navigate around them [13]. Capable of traversing a 8 meter room in 5 hours, obstacles were identified in images by corners and areas of high contrast.

DARPA's 2007 Grand Challenge exhibited the advancements in technology since the Stanford Cart. Competitors' vehicles relied on a combination of sensors and cameras to navigate the course. The winner of DARPA's 2005 Grand Challenge, the STANLEY Stanford team, detected obstacles by utilizing a laser to steer and a camera sensor to control throttle. Their system did not utilize the camera system to determine the direction the vehicle should travel [19]. Creating robust road surface and obstacle detection algorithms for images required too much effort. The breakthrough of convolutional neural networks is its ability to learn to recognize road surfaces and obstacles independently with minimal human preprocessing.

2.1.1 ALVINN

With convolutional neural networks, obstacles and road surfaces can be detected using a camera. ALVINN first demonstrated the potential for convolutional neural networks to be used in an autonomous car. ALVINN sensed the world around it through a forward facing camera and laser range finder. The neural network model processed this information, and outputted a direction in which the car should travel.

However, ALVINN's potential at the time was limited due to the processing power needed to train convolutional neural networks.

2.1.2 NVIDIA end to end learning

In 2015 NVIDIA's research team successfully developed a convolutional neural network capable of mapping raw pixels from a front facing camera into steering commands [3]. NVIDIA's neural network is able to learn useful road features in images with only steering inputs as a training signal. With their trained network, NVIDIA's neural network is able to navigate a car through highways and traffic. The breakthrough of NVIDIA's system is the ability for the car to navigate itself utilizing only a camera. NVIDIA's neural network serves as the basis for this research project.

2.2 Neural Networks

Neural networks are a type of machine learning algorithm vaguely inspired by biological neural networks. These networks consist of thousands to millions of densely connected nodes. Through training, neural networks can learn to classify data. A general model of a neural network can be seen in Figure 3. Each node's incoming connection is assigned a number known as a weight. These weights are initialized with random numbers. Each neuron's weight dictates what it will output. This example is a single output neural network with four inputs and one hidden layer.

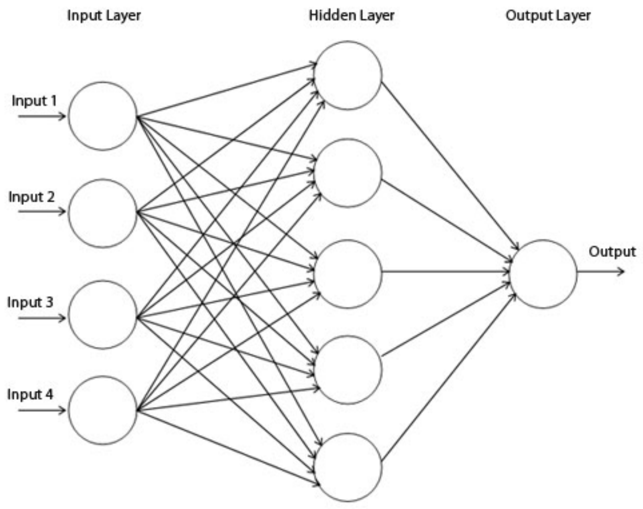


Figure 3: Neural network architecture

Artificial Node Neural networks are made up of many of artificial neurons - known as nodes. These nodes are the basis for how neural networks transform raw inputs into desired outputs. The artificial neuron works by calculating the sum of its weighted inputs, adding a bias, and outputting a value that is a function of a summed input. Each node can be described as a transposed multiplication of a input X.

$$f(W^T * X) \tag{1}$$

2.2.1 Activation Function

The activation layer is responsible for taking the output of the artificial node y, and determining what the neuron fires. This next section will describe two types of activations layers in this project: rectified exponential linear units and softmax.

Rectified Linear Units Rectified linear units are an activation function that outputs x only if x is positive. Otherwise, it outputs 0. The benefit of using rectified linear units is its ability to sparsely fire nodes only with negative weights. These units are used in NVIDIA's model described in Section 2.6.

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

Softmax The softmax activation function is used in the final layer of the categorical output models described in Section 2.6. This activation function is a generalization of the logistic function, squashing arbitrary real values between the range $(0, 1)$. The vector of a softmax output sums to 1. For $j = 1, ..., K$. This equation is used for categorical steering model. When used with cross entropy, it yields class probability.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2}$$

2.3 Training

For the neural network to learn, it must be able to reduce the error between its output and the desired output. This training process is done through a two step process: feed forward and error back propagation. During the feed forward pass the network is given an input and calculates an output determined by the weights mentioned previously. As the network trains, it aims to reduce the error between the output of the network and the actual output by changing the weights of its nodes.

Epochs A full training cycles are measured in epochs. The general definition of an epoch is a pass of the network over the entire dataset.

2.3.1 Error Functions

A loss function calculates the error: the difference between the correct output and model's ouput. This error is propogated backwards through the model to reduce its error. While there are different ways of calculating the error, this project's single output neural networks use the mean squared error loss seen in Equation 3 where X is a correct steering angle, Y is the predicted steering angle, and n is the total number of predictions.

$$MSE = \frac{1}{n}\sum_{n=1}^{n}(Y - f(X))^2 \tag{3}$$

The loss function used in a multi-output model is the cross-entropy loss function. This function returns the cross entropy between a predicted distribution and a true distribution, where every element in the distributions are in the range $[0, 1]$. Equation 4 shows the function, where M equals the number of categories, y is its binary class c (if it is the correct classification for observation o), and the predicted probability p o is of class c

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) \tag{4}$$

2.3.2 Overfitting

This training process is repeated over and over until the network reaches an error minimum. However, it is important to note that a minimum may not be the best solution for the problem. This issue is known as overfitting. Overfitting is a result of networks containing more weights than required of the problem. While the network may accurately classify the data it learned from, it might not be able to generalize the problem. Thus the true goal of a network is not for it to find the local minimum, but for it to generalize a optimal solution for all data. Methods to prevent this include early stopping: interrupting the training process so that the neural network does not reach the local minimum for a training data set.

Dropout Layer Dropout layers purposely select a percentage of nodes to be deactivated. Through deactivation, the network is forced to generalize with different nodes.

Validation and Training Data Another method to prevent overfitting is the use of validation data. While training the network, a subset of the data is set aside from training. After the network trains over the dataset, a validation dataset is used to give a unbiased evaluation of the network's performance during each training interval.

With this general overview of artificial neural networks, the next section details the neural networks used to analyze images: Convolutional Neural Networks(CNN).

2.4 Convolutional Neural Networks

In 1983 Kunihiko Fukushima introduced a neocognitron(a hierarchial multilayered artificial neural network) that could recognize hand written characters and patterns [7] using neural networks. Inspired by the neocognitron, in 1998 LeCun introduced the first convolutional neural network model with back propagation and gradient based learning. The convolutional neural network LeCun developed was able to accurately classify hand written digits in a 32x32 image [12]. While computing power in 1998 limited the resolution and scalability of the network, LeCun's reserach demonstrated that a CNN could be trained to recognize visual patterns with minimal preprocessing.

In the 21st century the ImageNet Project exhibited the full potential of Deep CNNs. Since 2010 the ImageNet Project has hosted an annual competition challenging teams to classify a large visual database with over 14 million annotated images [6]. During 2010 a reasonable error rate was considered to be 25%. Then in 2012 a deep convolutional neural network achieved a 16% error rate [11]. This dramatic improvement attracted both industry and research attention, sparking a deep neural network boom. This boom motivated NVIDIA to research using deep convolutional neural networks in autonomous cars.

Like neural networks, convolutional neural networks are made up of learnable weights. Nodes in a CNN receive inputs, calculate the weighted sum, pass it through a activation function and respond with a output. What separates convolutional neural networks from neural networks is their ability to analyze a volume of data using location invariant feature detectors. In this project's case, a three color depth image. Convolutional layers generate a feature map by iterating filters over an image, detecting features such as straight edges, simple colors, and curves. By stacking filters, the convolutional neural network is able to learn complex features and shapes.

An example of a convolutional filter is seen in Figure 4. Assume that in this example, there are only two color channels; black and white represented by 1 and 0. The convolutional filter is represented by matrix A, and the image represented by matrix B. The black border represents the location of the filter on the image: $(2, 2)$. The filter then multiplies each feature by the corresponding pixel in the image, sums it, and divides by the total number of pixels in the feature.

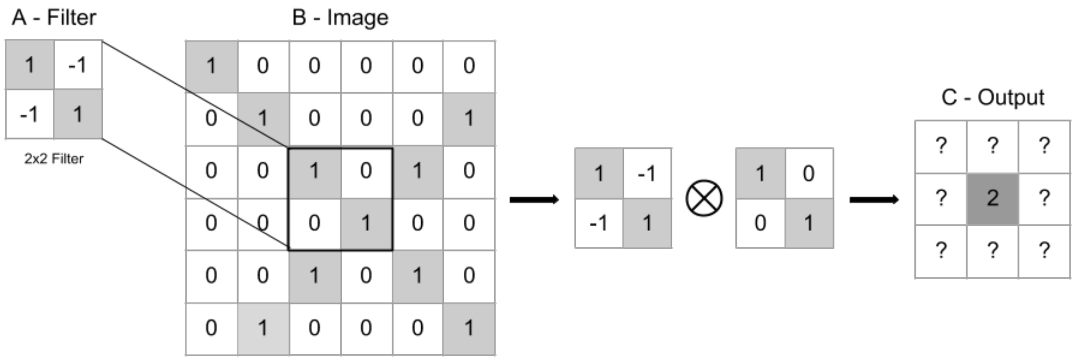


Figure 4: Example of a Convolutional Filter

The feature in Figure 4 is calculated as follows:

$$A \otimes B = \frac{(1 * 1) + (-1 * 0) + (-1 * 0) + (1 * 1)}{4} = \frac{2}{2} = 2 \tag{5}$$

This process is repeated until the feature has iterated over every image patch. The final output of the feature map is shown as C in figure 5:

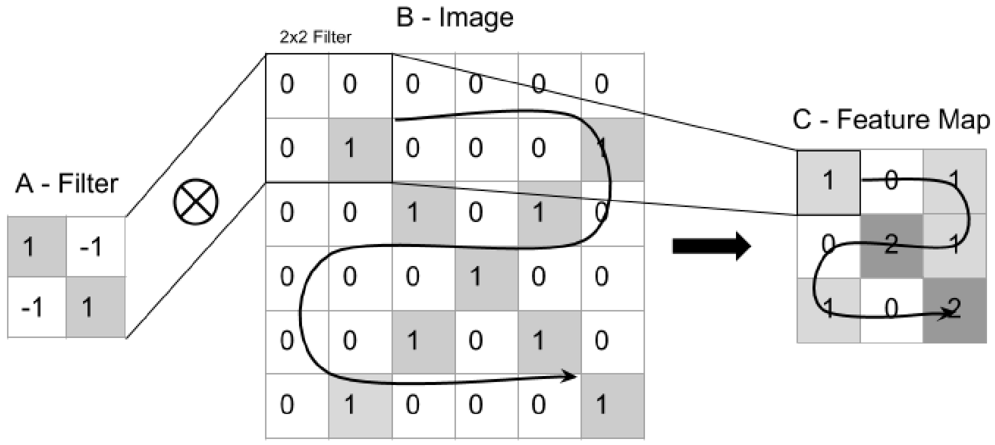


Figure 5: Feature Map

2.4.1 Dense Layer

In this project, dense layers represent a matrix vector multiplication that transforms a convolutional layer's two dimensional input into a one dimensional vector output. Every input is connected to every output by a learned weight.

2.5 Recurrent Neural Networks

Recurrent neural networks are a class of artificial neural networks that forms connections between units over time. The first development in recurrrent neural networks was in the 1980s by John Hopfield [18]. In this network, the state of the nodes activated depending on the input it received from every other node. In 1989 the Elman network introduced the idea of inputs from previous time steps fed forwards. Since then, recurrent neural networks have been demonstrated to be successful for speech recognition, text to speech synthesis, machine translation, language modeling, and image captioning. The basic algoirthm of a RNN is similar to a feedforward neural network layer, but includes a separate set of weights to evaluate results of previous timestep elements.

Figure 6 shows a example of a single unfolded recurrent neural network, where a represents the node, h_t the output and x_t the input. The variable t represents the timestep of the network.

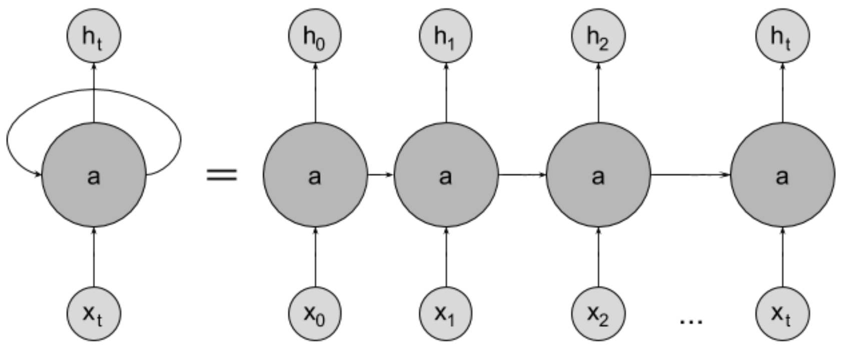


Figure 6: A unfolded recurrent neural network

The network is able to exhibit dynamic temporal behavior by enabling previous network states to influence the output of the current network. However, a flaw with recurrent neural networks is the inability to integrate information from distant timesteps.

2.5.1 Long Short-Term Memory

RNNs suffer issues storing memory over long periods of time. Errors flowing backward either vanish or blow up over long sequences. As a solution to this problem, Hochreiter and Schmidhuber introduced the long short term memory cell [9]. The goal of a LSTM is to have cells that make decisions on what to store from previous timesteps by using gates which open and close depending on the information. The gates are determined by weights which are adjusted through gradient-based learning.

Figure 7 shows an LSTM cell and the two time steps before the current state t. The difference between this and a recurrent neural network is how information from previous timesteps are inputted into the current step. There are three main gates in the network shown in Figure 7: forget gate (7a), input gate (7b), and the output gate (7c). The forget gate is responsible for choosing what the neural network will take as inputs. The input gate layer decides which of the gates should be updated. The final output gate pushes the values of the previous state into the neural network.

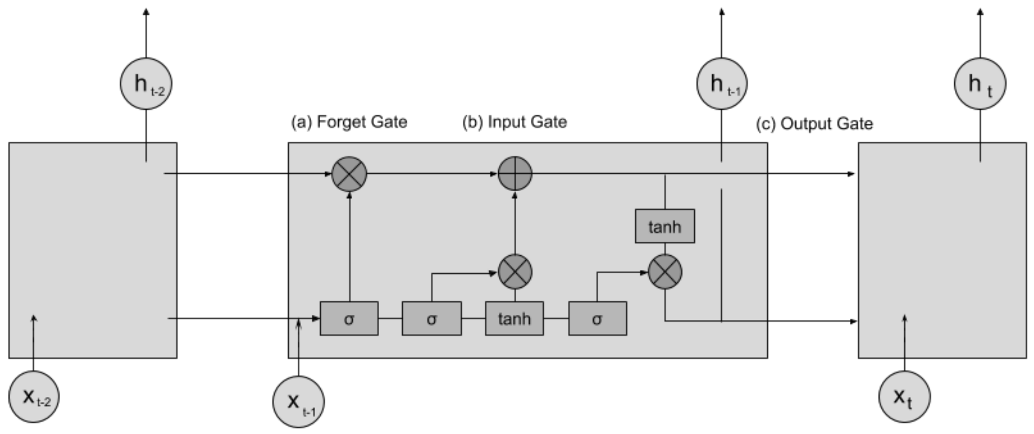


Figure 7: A unfolded LSTM network

2.6 Models

This research project explores the effectiveness of combining LSTMs with NVIDIA's deep convolutional neural network, and comparing single output models to categorical output models. This section discusswa the four models being evaluated in detail.

When designing a neural network, small changes in structure or parameters of the network can drastically alter its performance. The models described here are based on NVIDIA's neural network. The only modifications to NVIDIA's network is the number of outputs and recurrent layers.

A total of four different models are explored. These different modifications can be seen in Table 1. The columns are the number of outputs, and the rows state whether the model is recurrent or single state. The values in the table are the model names for each combination.

Table 1: Names of models being evaluated

	Single Output - Linear	**Multiple Outputs - Categorical**
Single State	linear	categorical
Recurrent	rnn	rnn_cat

Linear The linear model is the same model found in NVIDIA's published paper [3]. This model outputs a floating point number between $[-1.0, 1.0]$ that indicates steering direction.

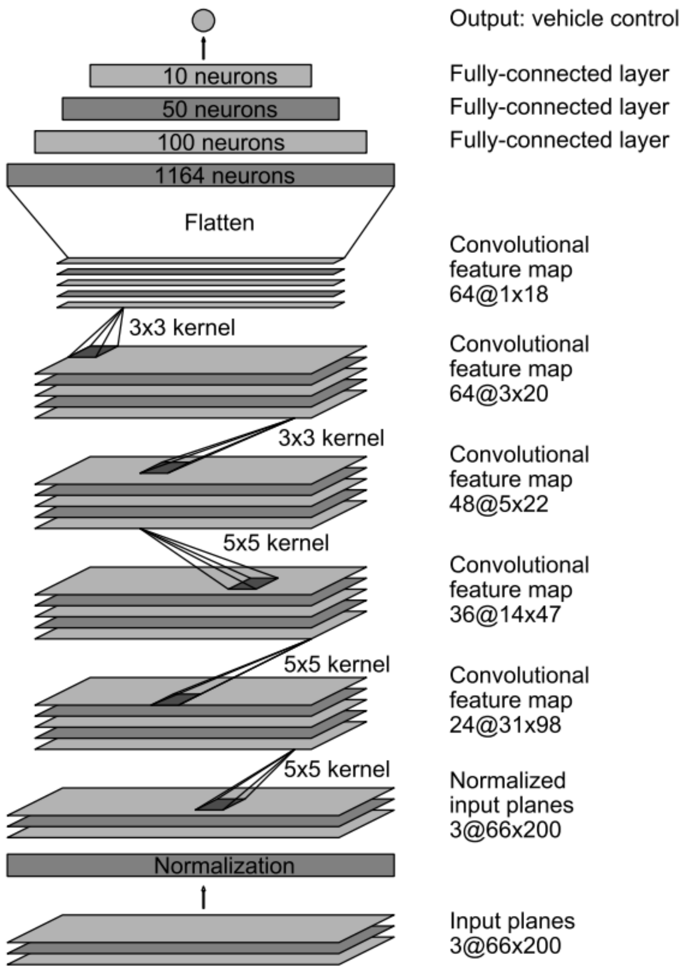


Figure 8: NVIDIA Linear Output network architecture [3]

Categorical This categorical output model returns a one dimensional 15 column vector representing the probability of each steering angle. Figure 9 shows the implementation of categorical outputs.

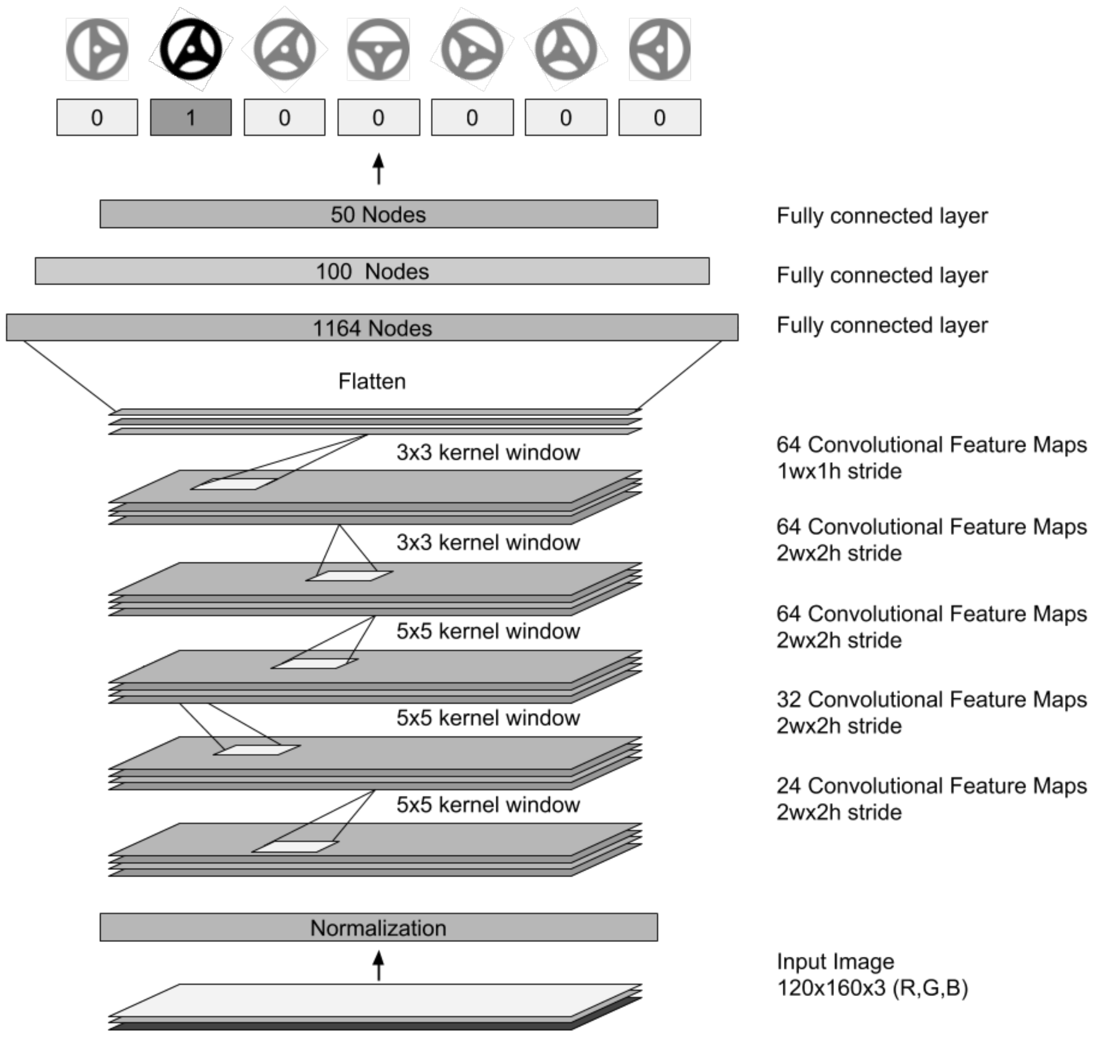

0
1
0
0
0
0
0
50 Nodes
Fully connected layer
100 Nodes
Fully connected layer
1164 Nodes
Fully connected layer
Flatten
3x3 kernel window
64 Convolutional Feature Maps 1wx1h stride
3x3 kernel window
64 Convolutional Feature Maps 2wx2h stride
5x5 kernel window
64 Convolutional Feature Maps 2wx2h stride
5x5 kernel window
32 Convolutional Feature Maps 2wx2h stride
5x5 kernel window
24 Convolutional Feature Maps 2wx2h stride
Normalization
Input Image 120x160x3 (R,G,B)

Figure 9: NVIDIA Categorical output network architecture

RNN The recurrent neural network model uses long short term memory cells for the final dense layers and recurrent sequencing convolutional layers - thus enabling the model to exhibit memory. There are only two differences between this model and the linear model: the implementation of recurrent and LSTM nodes and number of filters for two convolutional layers. In order for the model to run on a Raspberry Pi 3 (Tensorflow throws a generic out-of-memory error), the number of filters for the last two layers have been reduced from 64 to 32 filters. This model was sourced from Tawn Kramer's Github Branch of Donkeycar [10].

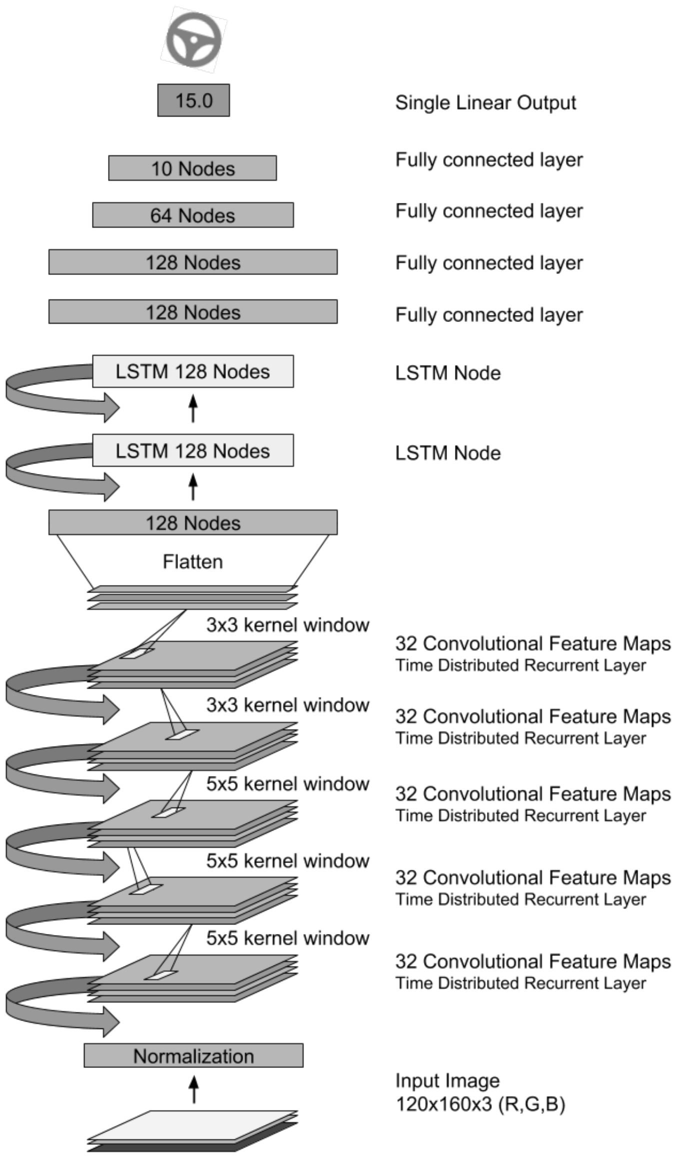


Figure 10: RNN Linear output architecture

RNN Categorical This is the same recurrent neural network model as described above, with the difference being the output. This model returns a one dimensional 15 column vector representing the probability of each steering angle.

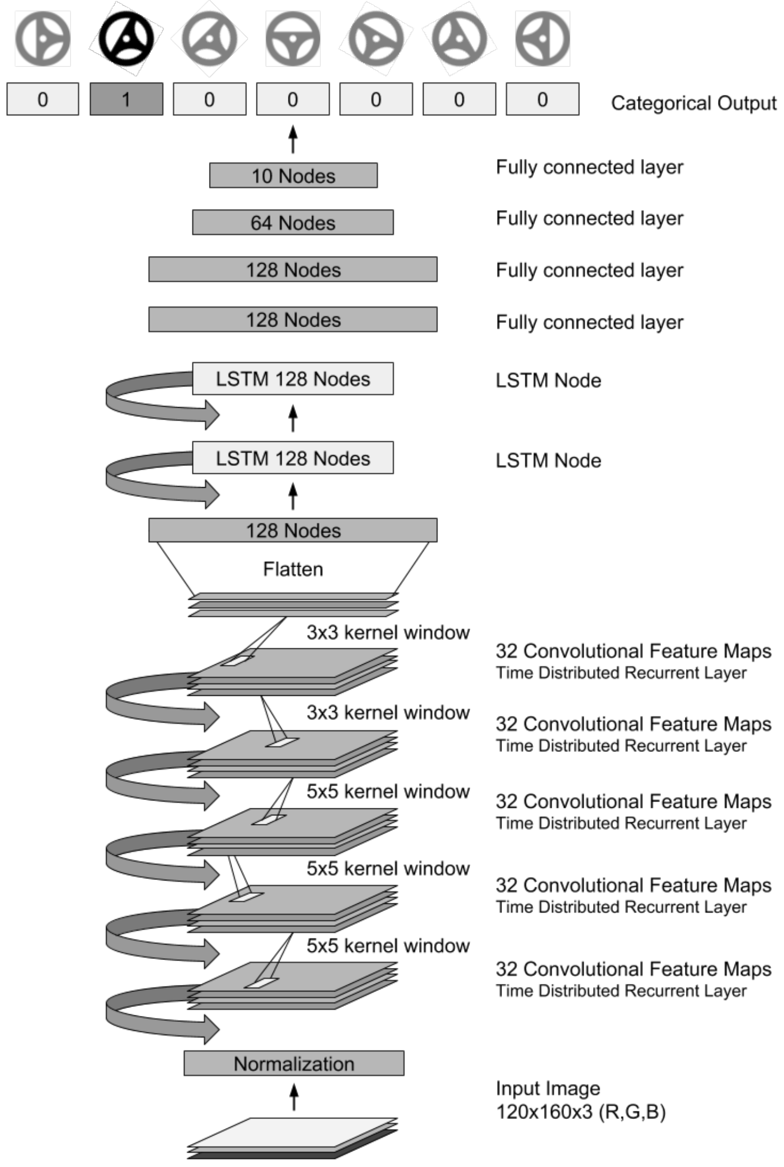


Figure 11: RNN Categorical output network architecture

3 Implementation

This section explains how these models are implemented on the car hardware.

3.1 System Overview

The scaled self driving car platform is built on the Donkeycar open source platform. This platform combines a RC Car, Raspberry Pi, Python, and various Python packages(Tornado, Keras, Tensorflow, OpenCV) to create a scaled autonomous vehicle. This section details the components used to build the platform. Figure 12 shows a image of the vehicle, and its various components.

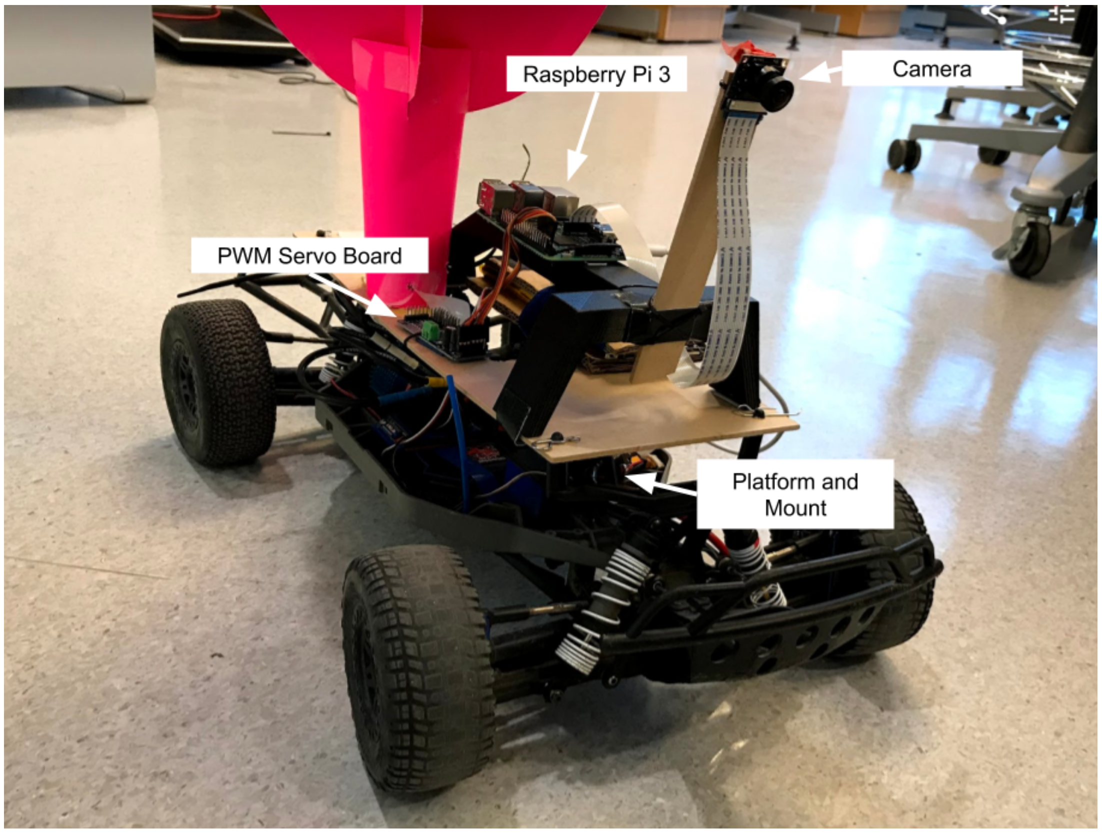


Figure 12: The scaled self driving car. Modified 1/10 Traxxas Slash 4x4.

The platform's foundation is built using modules called parts. Each part wraps a functional component of a vehicle. The parts used in this research project are cameras, acutuators, motor controllers, pilots, web console, and tubs(for data). Figure 13 shows the main components used for driving the vehicle with a neural network model.

Raspberry Pi 3 The Raspberry Pi ARMv8 64 bit processor is responsible for neural network computations, communications with the user, and controlling the remote control car through a pulse width modulation servo board. Two channels on the servo board are connected to the RC Car: channel one controls the speed of the motor, and channel two controls the steering.

Traxxas Slash 4x4 The RC Car chosen for this project is a Traxxas Slash 4x4. The Traxxas Slash is a consumer grade remote control car modeled at 1/10th scale. At 1/10th scale there is substantial space for a Raspberry Pi, servo board, and battery to be mounted.

Platform and Camera Mount The mounting platform is built using a piece of flat plywood and a 3D printed camera mount. The 3D Printed Camera mount is uploaded on this project's github repository page [**ssdc**]

Keras In this research project all neural networks are implemented using Keras. Keras is an open source neural network library written in Python. A tensorflow backend is used for building the networks. The core data structure of Keras are models made up of a linear stack of layers. For example, to construct a simple Sequential model with 100 inputs and 10 categorical outputs:

```python
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, batch_size=32)
```

3.2 Donkeycar Platform

This section describes the integration of these components. There are two modes during initialization of the donkeycar: drive or train. Training is covered in Section 5.1.

In drive mode, the car launches an online web controller on the local network through WiFi. Within drive mode there are two options: human pilot or autonomous pilot. Figure 13 shows the components set up in autonomous drive mode. The web controller in Figure 13 is the main interface for controlling the car.

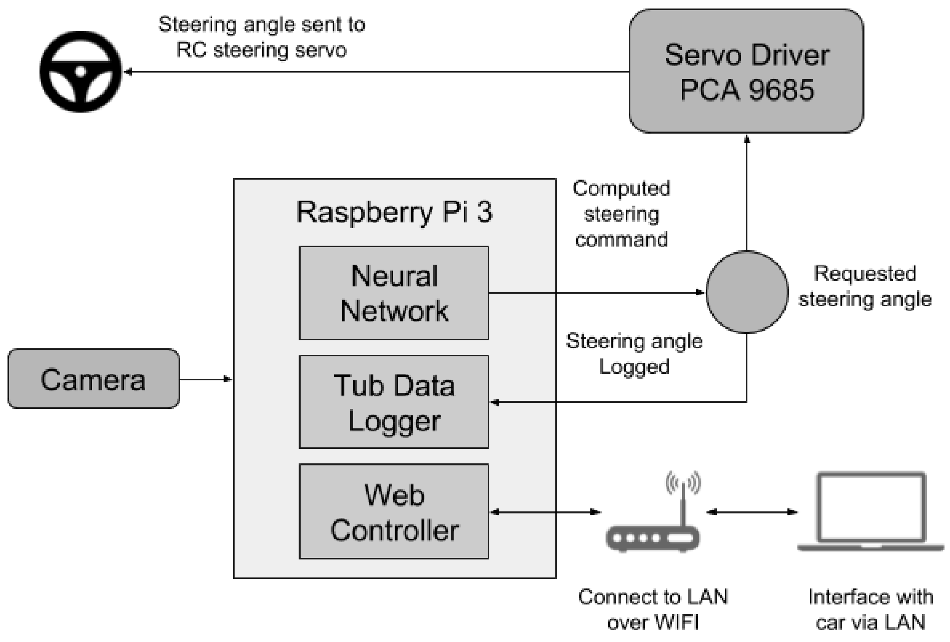


Figure 13: Donkeycar platform components

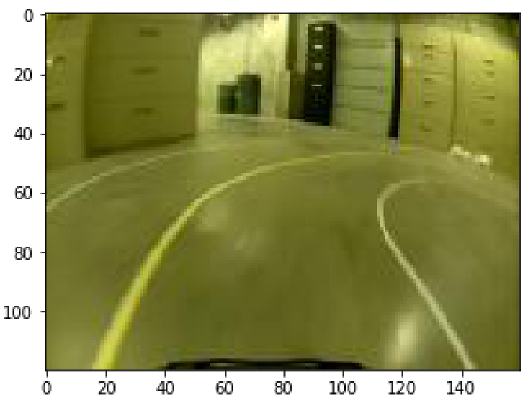
After connecting to the interface on the same local network, the user can toggle the car between autonomous and human pilot mode. In human pilot mode, the user controls the steering and throttle of the car. This mode is primarily used when collecting training samples.

In autonomous mode, images are fed into the neural network model, and the output (steering angle) is sent to the servo driver.

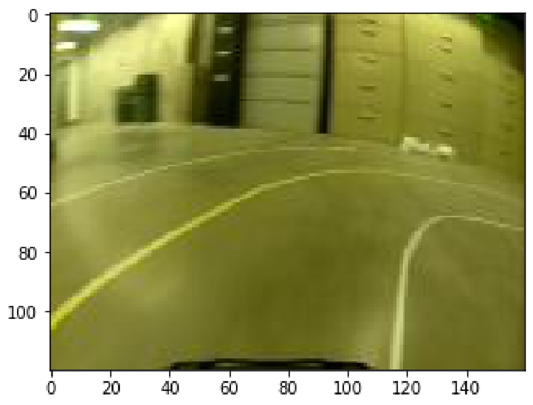
In both autonomous and user pilot modes, images and steering information are saved in a datastore.

3.2.1 Datastore and Records

All images and steering angles during each session are stored in a datastore. At 20hz, the steering angle and image is captured and written into a record. The record is written as a JSON file containing the location of the image file, steering angle, time, and throttle.



(a) Sample image 730 from 8-Track dataset



(b) Sample image 740 from 8-Track dataset

Figure 14: Dataset image samples

```
{ "record_740": {
    "cam/image_array": "740_cam-image_array_.jpg",
    "user/angle": 0.5569997712185508,
    "user/throttle": 0.6010691979352165,
    "user/mode": "user" },
"record_750": {
    "cam/image_array": "750_cam-image_array_.jpg",
    "user/angle": 0.7670956034409372,
    "user/throttle": 0.6042288050932043,
    "user/mode": "user"} }
```

Figure 15: JSON file referencing images in Figure 14

4 Datasets and Training

The models are evaluated by the ability to determine the correct steering angle compared to a human driver and the path of the vehicle compared to one piloted by a human driver. The performance of these models are evaluated using two types of tracks detailed in this section.

4.1 Tracks

The tracks are composed of two white lanes taped one meter apart, and a yellow center divider lane. The tracks are made using white ribbon and yellow masking tape.

4.1.1 8-Track

To test generalization, the models are trained with only the 8-Track dataset. The training dataset contains 59,190 images. The dimensions of the track are shown in Figure 16. The red arrow points to the direction the vehicle travels in and the initial position of the car.

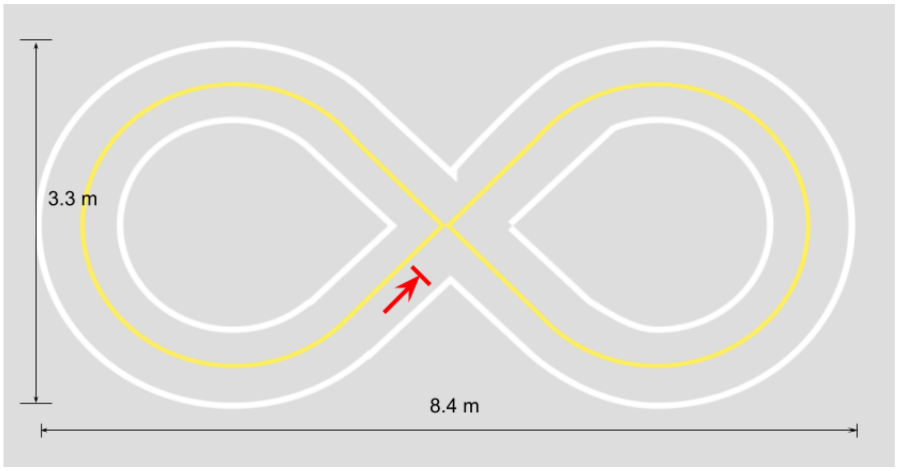


Figure 16: 8-Track - The red arrow indicates direction and initial position

The 8-Track is used to build the training dataset as the steering angles required to navigate it encompass the remote control car's steering range. Figure 17 below shows the histogram of steering angles requested by a human driver over 20 laps. Note that the steering angles are not an even distribution due to the physical behavior of hardware and steering servo.

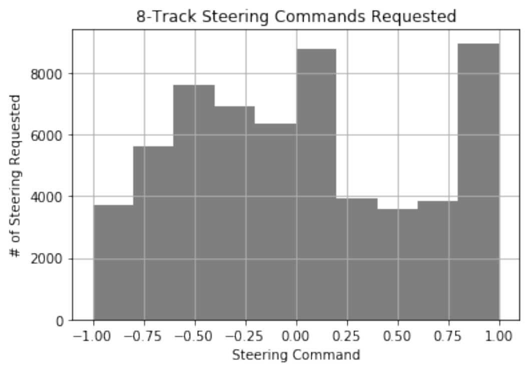


Figure 17: 8-Track Steering Histogram

Figure 18 is a graph of the car's turning radius path for each steering angle request. The paths are all labeled. A explanation for the uneven distribution seen in 17 is the car's bias towards right turns. Steering requests between $-0.4, 0.4$ are negligible.

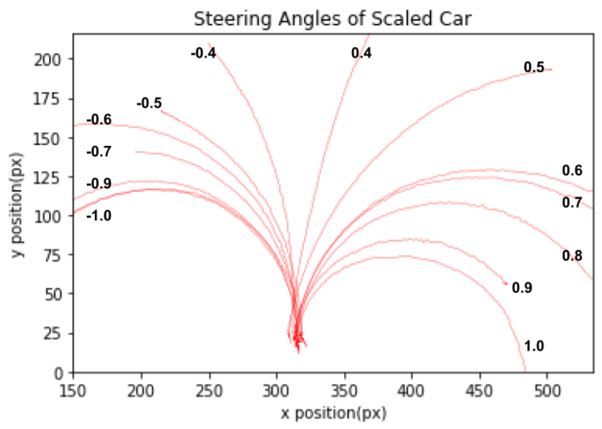


Figure 18: Car steering angle requests vs. path graphed

4.2 Circuit

The dimensions of the network are shown in Figure 19. This track consists of 2 left turns and 1 right turn. The purpose of this track is to evaluate the model's ability to generalize onto a track different than the 8-Track. The car was driven counter clockwise. The red arrow shows where the vehicle starts from, and which direction it is traveling in.

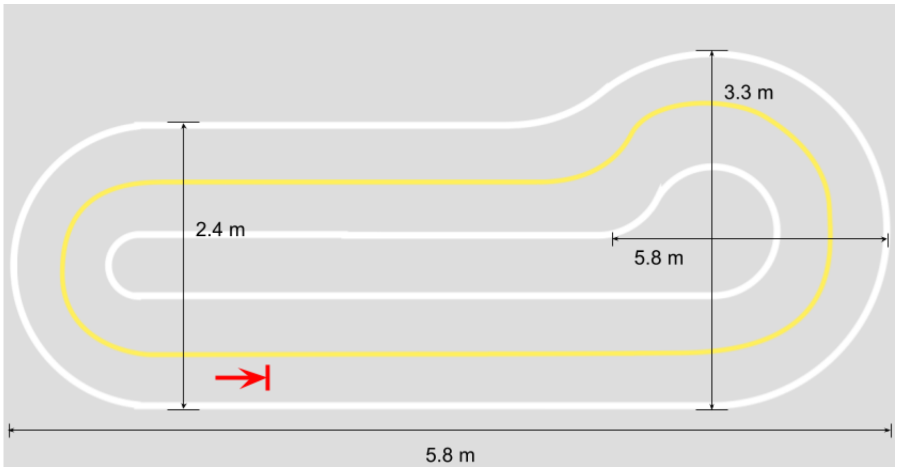


Figure 19: Circuit - The red arrow indicates direction and initial position

Figure 20 below shows the histogram of steering angles requested by a human driver over 20 laps on the circuit

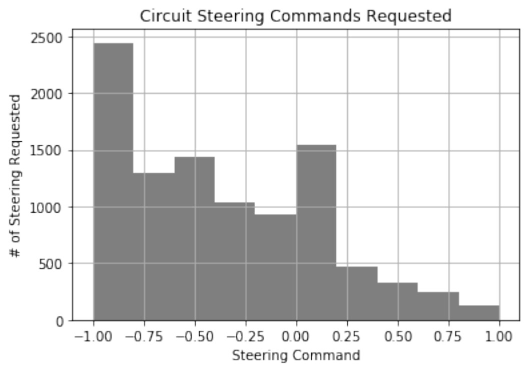


Figure 20: Circuit Steering Histogram

5 Methods

This section details how the models are trained and evaluated when navigating each track.

5.1 Model Training

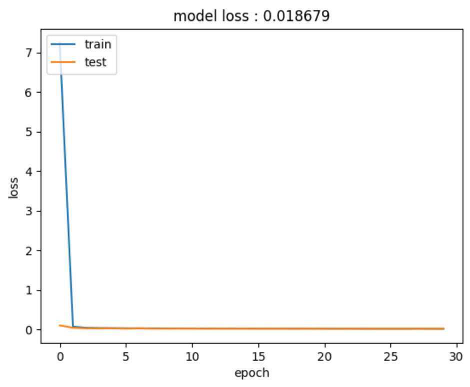
All models were trained using the same dataset: 59,190 images of a human driver navigating the 8-Track. In these recordings, the car is limited to 50% throttle. All models are given three training sessions. After three training sessions, the model with the lowest error loss is selected for piloting the car. Models are trained using a NVIDIA GTX 1070 and took an average of 25.3 minutes each.

Each training session continued until the validation loss increased for 5 epochs. Validation losses must improve by a minimum of 0.0005 to be considered as progress. Figure 21 shows the training loss values for each model over time.

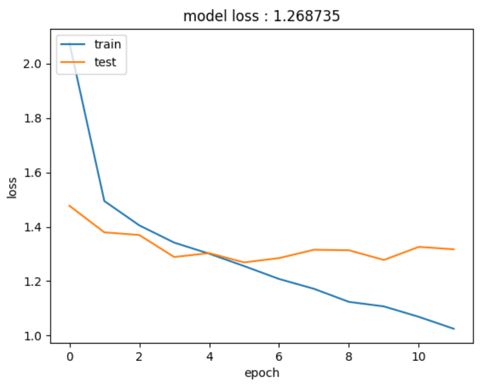
5.2 Steering Accuracy

To evaluate each models' steering accuracy, the models are tested on a dataset excluded from the training and validation dataset. For each record in the dataset, the model predicted a steering angle. This predicted steering angle was compared to the actual steering angle for the record. The mean squared error was taken between The difference of the predicted angle and human angle (Equation 3).

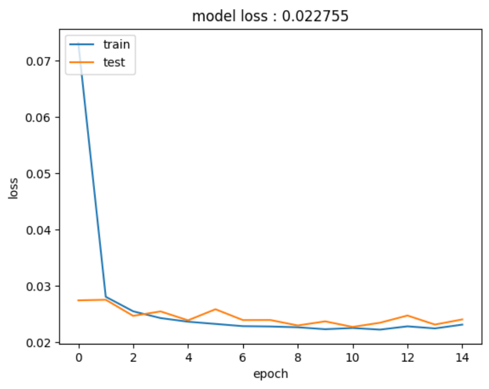
For the 8-Track, the models were evaluated on a recorded dataset of a human driver looping twice around it. There are a total of 725 images in this dataset. The circuit testing dataset contains 4820 records of a human driver driving 5 loops around it. A higher mean squared error is expected for the circuit dataset as none of the models have been trained on it.



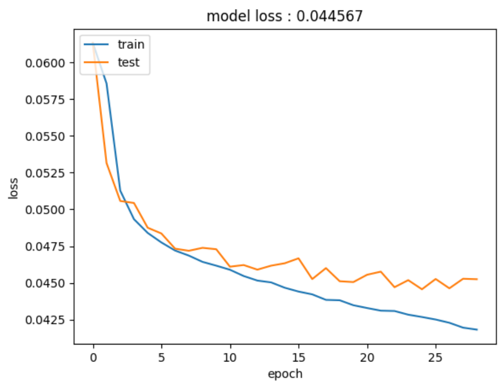
(a) Linear Model Training Loss



(b) Categorical Model Training Loss



(c) RNN Model Training Loss



(d) RNN Categorical Model Training Loss

Figure 21: Model Training Loss

5.3 Position Tracking

While MSE evaluates the model's steering accuracy, it does not evaluate the model's ability to keep the car centered in its lane. Thus, the global position of the vehicle must be known. Automated tracking of the car was implemented by using a ball-tracker in OpenCV and Python. A flourescent pink circle is mounted on the car for the program to track (Seen in Figure 13). The position tracker program is directly sourced from Adrian Rosebrock's blog "Ball Tracking with Open CV" [16]. The only modifications to this program was a different color configuration and addition of a method to export a CSV data file (containing X Y coordinates). A screenshot of the program running is shown in Figures 22 and 23.

For the 8-Track, the camera was placed on a stand 2 meters high at an downward acute angle from the horizon due to a low ceiling. Because of this, the 8-Track dataset is too distorted for positional comparisons. Thus, the dataset is only used as a reference for visualizing the car's position. Figure 22 shows a screenshot of the program tracking the vehicle.

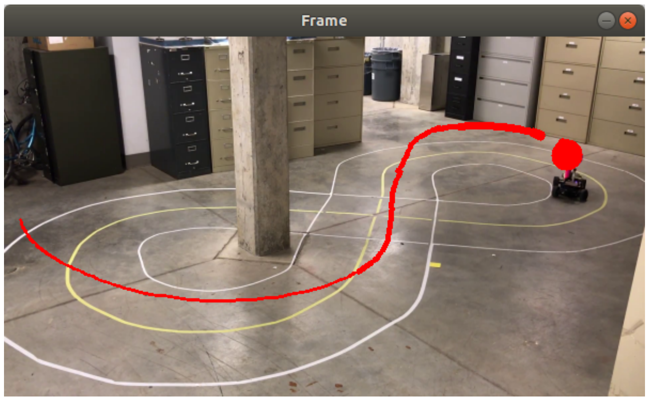


Figure 22: Screenshot of program used to track car on 8-Track.

The Circuit Track's overhead camera was setup 6 meters above the track at a obtuse angle downwards from the horizon. However, calculations in this project assumed it is directly overhead the track at a right angle. In this image, the scale is $1px = 3.27cm$.

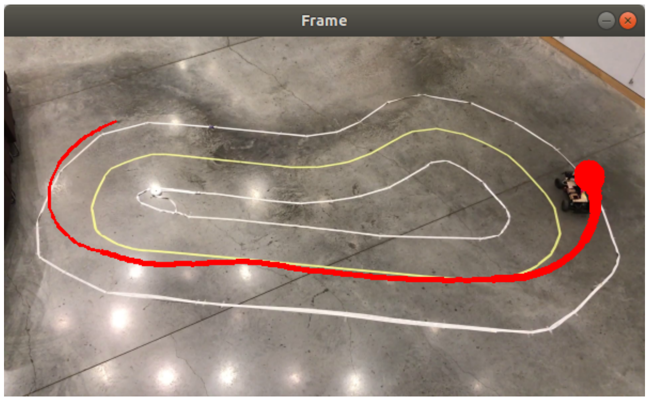


Figure 23: Screenshot of program used to track car on circuit.

Another variable with this track was the reflection from the sun and light fixtures on the floor. The effects of these lights on each model are explained in Section 6

5.3.1 Path Averaging and Comparison

The average of the paths were calculated by taking the mean of all points within 20 pixels of each point. Paths are compared by using vector projections to calculate the distance between the point closest to another path's point. Figure 24 represents the comparison of two paths. Figure 24 (a) shows the output of this function.

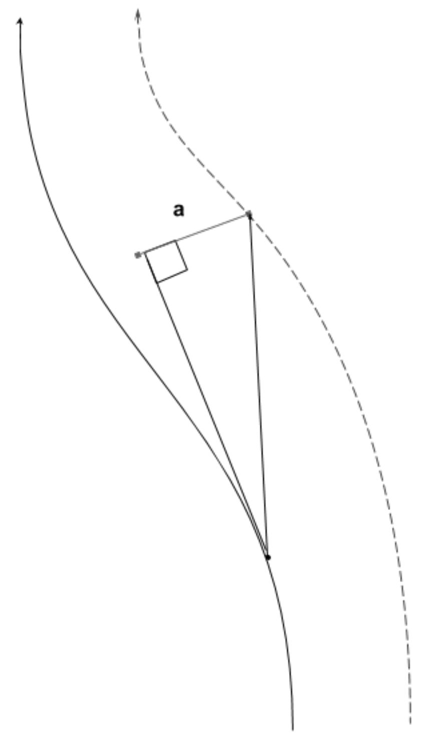


Figure 24: Vector projection to calculate distance between path points

5.4 Track Trials

After setup of the overhead camera, the models are tested on each circuit as follows.

8-Track Each model is given three trials. The trial ends after the car laps the track 20 times, or veers off course. The cars are all started from the same position. As this track was set up in a basement, environment conditions stayed the same.

Circuit Each model is given five trials. The trial ends after the car laps the track ten times, or veers off course. The cars are all started from the same position.

6 Results and Analysis

After training and validating the models, the recorded results of the methods are described in this Chapter.

6.1 Steering Accuracy

Tables 2 and 2 shows the mean squared error for each models' predicted steering angles on the 8-Track dataset. The variance of each model's steering error on the 8-Track is shown in the boxplot in Figure 25. The variance is too high to make a significant comparison about these models from Table 2.

Table 2: Model MSE on 8-Track

8-Track	
Model	MSE
Linear	0.21
Categorical	0.23
RNN	0.18
RNN Categorical	0.28

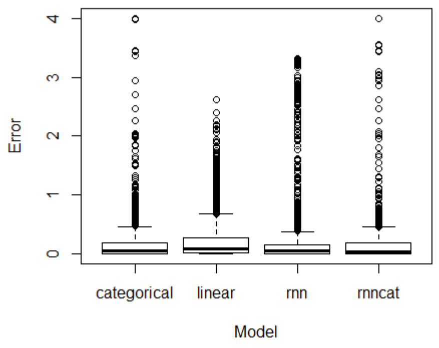


Figure 25: Model and Steering Error Variance

For the circuit dataset, the recurrent neural network has the lowest mean squared error compared to the other models. The categorical output models also have higher errors than single output models. The variance of each model's steering error on the Circuit is shown in the boxplot in Figure 26. From the ANOVA calculation, the F-statistic is 4.21 with a p-value equal to 0.00558 rejecting the null hypothesis of equal mean squared errors. Thus it can be concluded the recurrent neural network has the lowest mean squared error. Notably, the categorical output models have higher mean squared errors than its linear models.

Table 3: Model MSE on Circuit

Circuit	
Model	MSE
Linear	0.30
Categorical	0.33
RNN	0.28
RNN Categorical	0.34

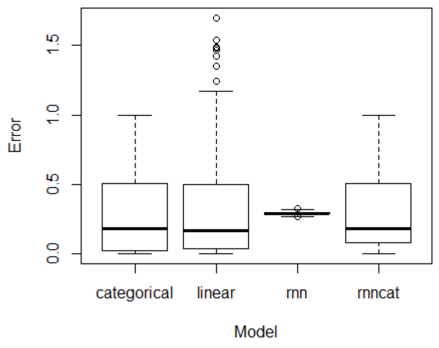


Figure 26: Model and Steering Error Variance

6.2 Raw Model Paths

While the mean squared error assesses the network's ability to predict steering angles, it does not account for any lateral shift from a model's predicted angle. In this project a overhead camera is used to track the global position of the car's when piloted by a human or neural network. Evaluating the ability of the models to generalize is important as a robust autonomous vehicle needs to be able to identify road surfaces it has never been trained on. This section contains figures of the path of the car when piloted by human and neural network models on the 8-Track and Circuit courses. The recorded human path for each track is used as the baseline for comparing the models.

Figures 27 and 28 are graphs of driving paths for both human and neural network models. Every model was able to navigate the 8-Track after training. The paths shown in the Figure 28 is the result of each model piloting the car 20 laps on the 8-Track course.

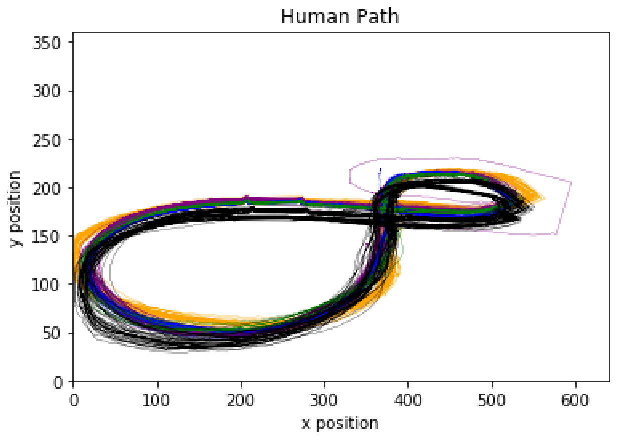


Figure 27: Path of every human and neural network model on 8-Track (px)

Figure 28 is a graph of driving paths for both human and neural network models. On the Circuit, all models failed to complete all five trials. However, the RNN was able to complete the most laps.

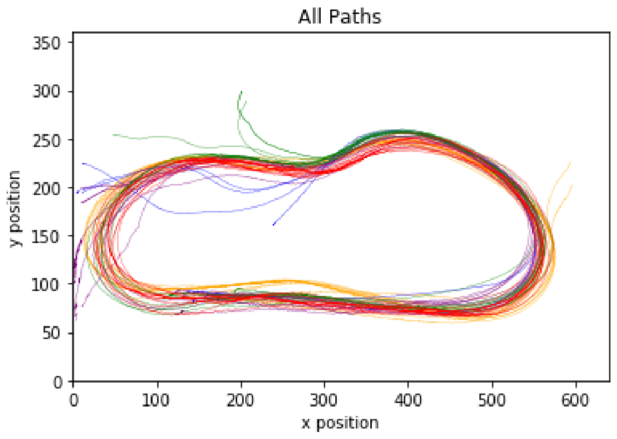


Figure 28: Path of every human and neural network model on circuit (cm)

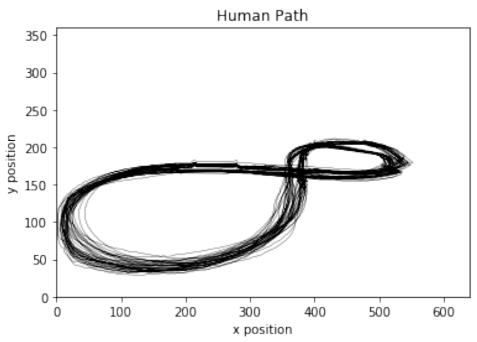
6.3 Averaged Model Paths

The figures below are the averaged paths of the vehicle when piloted by each neural network model. Each model's path is overlaid the averaged path of a human for reference. Note that only the circuit track was used to calculate path deviation.

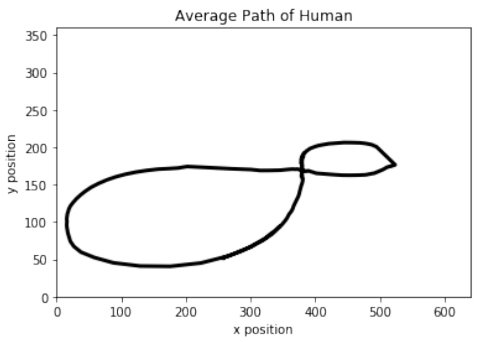
6.3.1 Human Driver Path

The 8-Track and Circuit was recorded from a human pilot.

8-Track The paths shown here were also included in the training set for the models.



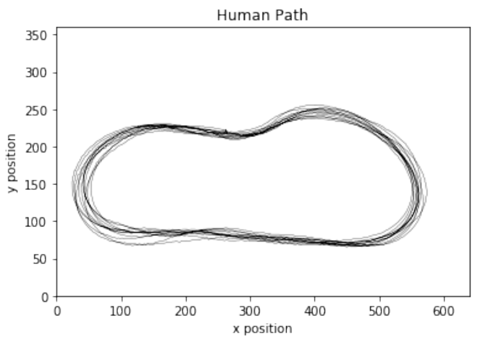
(a) Human Path (px)



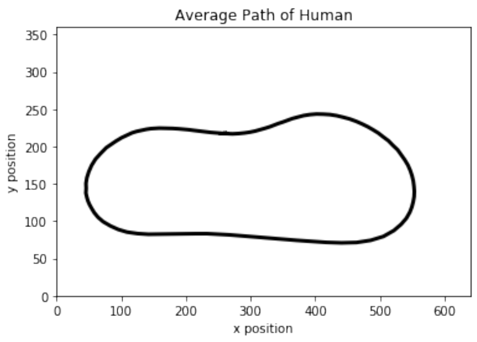
(b) Average Human Path (px)

Figure 29: Human Driver Paths

Circuit A subsection of this path is used to assess the steering accuracy of the models.



(a) Human Path (cm)

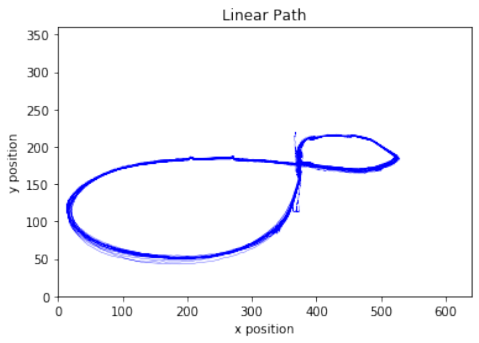


(b) Average Human Path (cm)

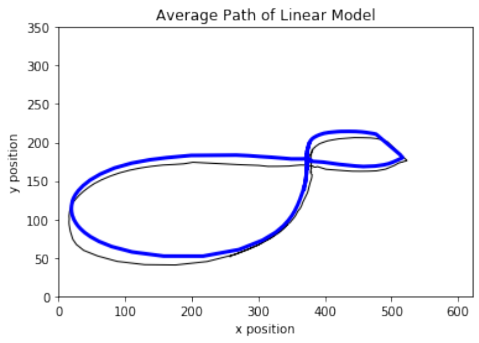
Figure 30: Human Driver Paths

6.3.2 Linear Model

8-Track The linear was able to complete 20 laps of the 8-Track. The black path is the average human pilot path.



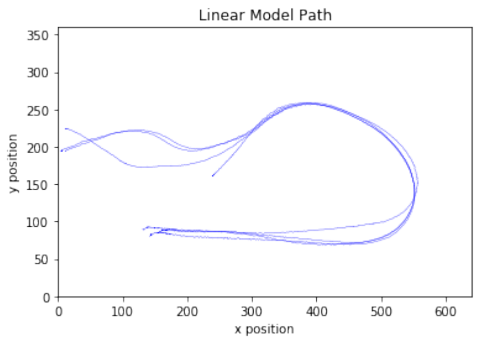
(a) Linear Path (px)



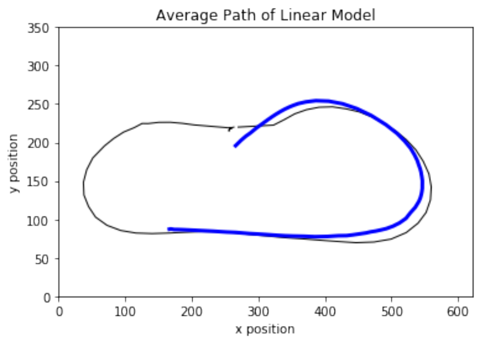
(b) Average Linear Path (px)

Figure 31: Linear Model Paths

Circuit On the circuit, the linear model failed all five trials. The black path is the average human pilot path.



(a) Linear Path (cm)

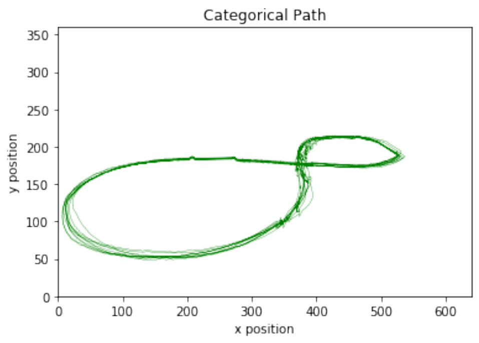


(b) Average Linear Path (cm)

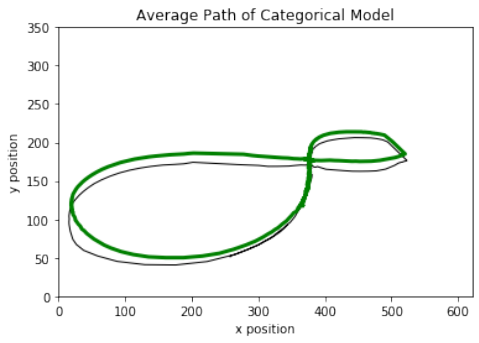
Figure 32: Linear Model Paths

6.3.3 Categorical

8-Track The categorical model was able to navigate 20 laps of the 8-Track. The black path is the human average path.



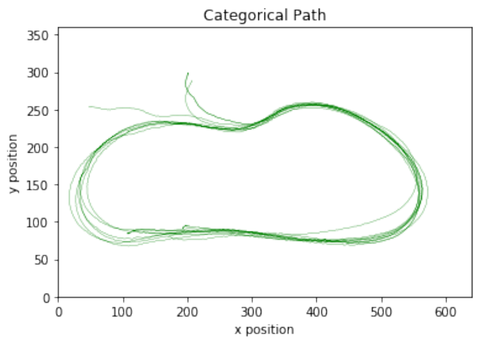
(a) Categorical Path (px)



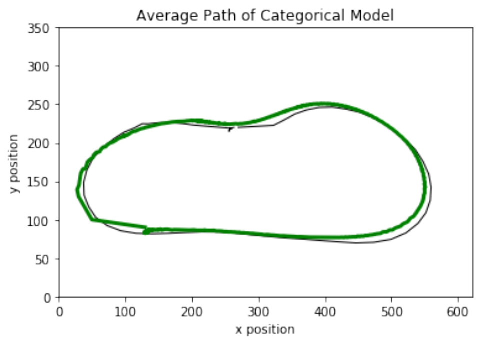
(b) Average Categorical Path (px)

Figure 33: Categorical Model Paths

Circuit The categorical model was only able to complete the trial 2 out of 5 times.



(a) Categorical Path (cm)

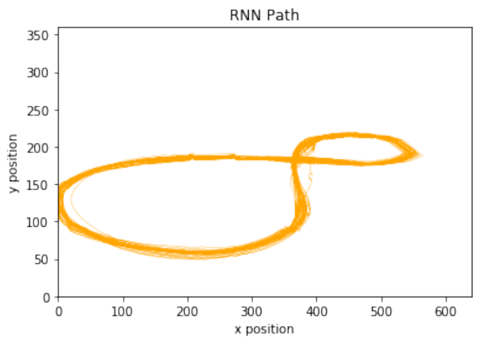


(b) Average Categorical Path (cm)

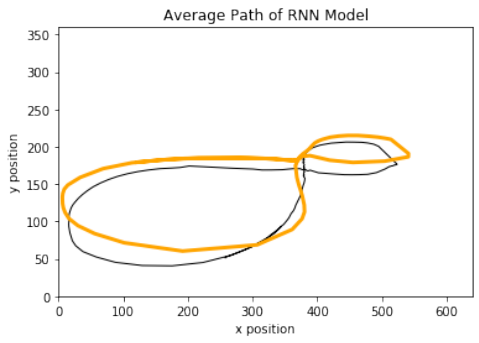
Figure 34: Categorical Model Paths

6.3.4 RNN

8-Track The RNN model was able to navigate 20 laps of the 8-Track. The black path is the average human pilot path.



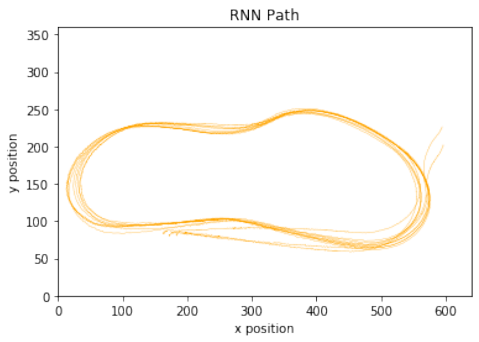
(a) RNN Path (px)



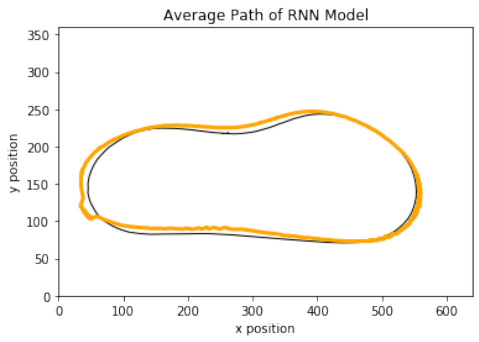
(b) Average RNN Path (px)

Figure 35: RNN Paths

Circuit The RNN model was the only model able to complete all 3/5 trials of the circuit. An interesting characteristic to note is on the bottom straightaway, the RNN model tended to drift towards the inside of the track. This may be a result of the LSTM influencing the RNN to prepare for a right turn that was learned in the 8-Track dataset.



(a) RNN Path (cm)

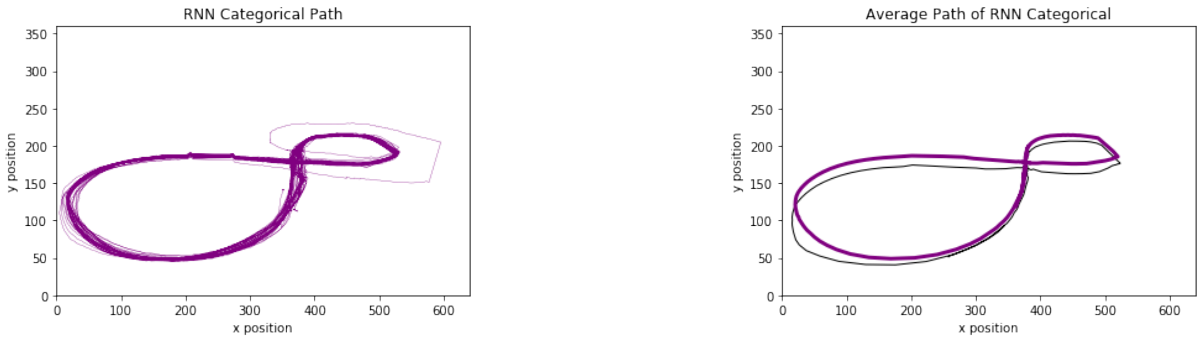


(b) Average RNN Path (cm)

Figure 36: RNN Paths

6.3.5 RNN Categorical

8-Track The RNN Categorical model was able to complete 20 laps without issue.

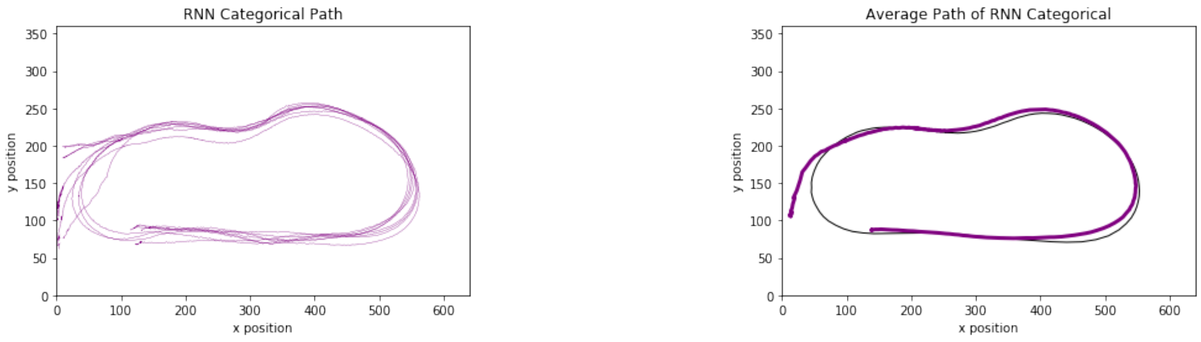


(a) RNN Categorical Path (px)

(b) Average RNN Categorical Path (px)

Figure 37: RNN Categorical Path

Circuit The RNN Categorical only completed 2 out of 5 trials. A possible explanation for 3 trials failing was the lighting reflecting from the floor had impacted the model's performance. On the trials it did complete, a notable difference between this and a RNN model with a single output was its path on the circuit's straightway.



(a) RNN Categorical Path (cm)

(b) Average RNN Categorical Path (cm)

Figure 38: RNN Categorical Path

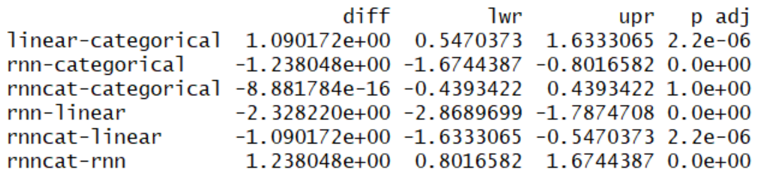
6.4 Circuit Path Deviation

After averaging the positional data for each of the models, the model's deviation is calculated by comparing it to the averaged human pilot path. Table 4 shows the number of points compared and the models' mean deviation from the human path. On average a complete lap around a Circuit is 111 points. As the RNN and Categorical models were able to complete the trials, the number of points for these two models are higher than the Linear and RNN Categorical models.

Table 4: Circuit Models and Average Deviation

Circuit		
	Points	**Mean Deviation**
Linear	54	3.36 cm
Categorical	109	2.22 cm
RNN	112	0.98 cm
RNN Cat	64	4.75 cm

The statistical significance of difference in model mean deviation was determined by performing an analysis of variance using R. The ANOVA test showed a $p < 0.05$ adjusted value between each model's error deviation (Figure 39).



```
                          diff        lwr        upr   p adj
linear-categorical  1.090172e+00  0.5470373  1.6333065 2.2e-06
rnn-categorical    -1.238048e+00 -1.6744387 -0.8016582 0.0e+00
rnncat-categorical -8.881784e-16 -0.4393422  0.4393422 1.0e+00
rnn-linear         -2.328220e+00 -2.8689699 -1.7874708 0.0e+00
rnncat-linear      -1.090172e+00 -1.6333065 -0.5470373 2.2e-06
rnncat-rnn          1.238048e+00  0.8016582  1.6744387 0.0e+00
```

Figure 39: ANOVA Table of Models

The boxplot in 40 shows that the mean of the RNN model is lower than all other models on the Circuit.

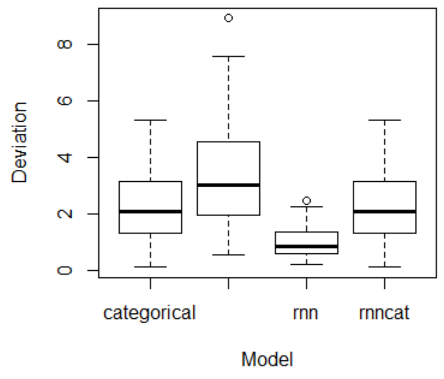


Figure 40: Box Plot of Model Deviation Variances

The histograms in Figure 41 shows the variance for each of the models. The RNN has a significantly tighter variance than compared to other models.

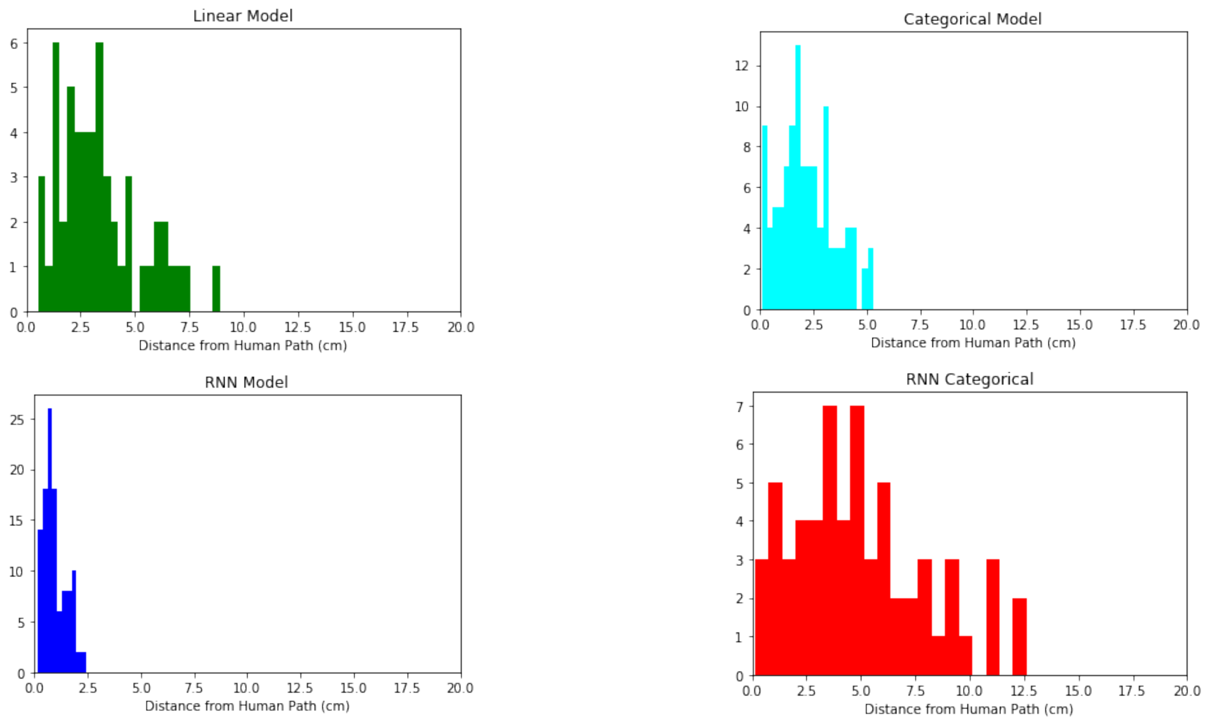


Figure 41: Path deviation of models on Circuit course

7 Conclusion

This project shows that a recurrent neural network in combination with NVIDIA's model outperforms NVIDIA's single state linear output model in generalization. After training the models on the same 8-Track dataset, the RNN model was able to generalize and navigate the Circuit track with a average deviation of 0.98 cm from a human piloted path. NVIDIA's single output network model failed to generalize on the Circuit track, and had an average deviation of 3.36 cm. The RNN model had the smallest variance of 2.25 cm on the Circuit (compared to NVIDIA's linear model variance of 8.36).

Interestingly, the Categorical model had the second smallest variance with 5.22 cm, but was the second worst model in steering accuracy on the Circuit dataset with a MSE of 0.33 (Table 3). Future research could include reconsidering how autonomous vehicle models are evaluated. The categorical model could have been considered a bad model if only evaluated with MSE.

This project also demonstrated that one track is not substantial for models to generalize to any track. Another issue that must be considered is lighting conditions. Reflections of light can confuse the network into outputting incorrect angles. Additional research could increase the diversity and amount of training data. A possible direction for increasing the dataset diversity is training the model to generalize on lighting reflections.

Improvements can also be made to the scaled self driving car platform. Increased processing power on the Traxxas Slash would enable longer RNN sequence lengths. Hardware can be improved with a more accurate steering rack. Additional modifications to the model could include increasing the number of outputs.

References

[1] Keshav Bimbraw. “Autonomous Cars: Past, Present and Future - A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology”. In: 1 (Jan. 2015), pp. 191–198.

[2] Official NVIDIA Blog. *Tesla Self-Driving Car Built on NVIDIA DRIVE PX 2.* URL: `https://blogs.nvidia.com/blog/2016/10/20/tesla-motors-self-driving/`.

[3] Mariusz Bojarski et al. “End to End Learning for Self-Driving Cars”. In: *CoRR* abs/1604.07316 (2016). arXiv: `1604.07316`. URL: `http://arxiv.org/abs/1604.07316`.

[4] DARPA. *Grand Challenge.* URL: `http://archive.darpa.mil/grandchallenge/`.

[5] Ernst D.Dickmanns. “Vehicles capable of dynamic vision: a new breed of technical beings?” In: *ELSEVIER* 103 (1998), pp. 49–76. DOI: `https://www.sciencedirect.com/science/article/pii/S000437029800071X`.

[6] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: (2009).

[7] Kunihiko Fukushima. “Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position”. In: *Biological Cybernetics* 36 (1980), pp. 193–202.

[8] “Highway of the Future”. In: *Electronic Age* (1958). DOI: `http://www.americanradiohistory.com/Archive-Radio-Age/Electronic-Age-1958-Winter.pdf#14`.

[9] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: 9 (Dec. 1997), pp. 1735–80.

[10] year = 2018 publisher = GitHub journal = GitHub repository howpublished = `https://github.com/tawnkramer/donkey/` commit = 2c5ae708b47ecac73435d8ffa5490a54e1fc55ea Kramer Tawn title = Donkey.

[11] *Large Scale Visual Recognition Challenge (ILSVRC).* URL: `http://www.image-net.org/challenges/LSVRC/`.

[12] Yann Lecun et al. “Gradient-based learning applied to document recognition”. In: (1998), pp. 2278–2324.

[13] Hans P Moravec. In: *The Robotics Institute Carnegie Mellon University* (24 February 1983). DOI: `http://www.dtic.mil/dtic/tr/fulltext/u2/a133207.pdf`.

[14] “’Phantom Auto’ will tour city”. In: *The Milwaukee Sentinel* (1926). DOI: `https://news.google.com/newspapers?id=unBQAAAAIBAJ&sjid=QQ8EAAAAIBAJ&pg=7304,3766749`.

[15] Dean A. Pomerleau. “ALVINN: An Autonomous Land Vehicle in a Neural Network”. In: (1989). Ed. by D. S. Touretzky, pp. 305–313. URL: `http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf`.

[16] Adrian Rosebrock. *Ball Tracking with OpenCV.* 2016. URL: `https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/`.

[17] Dominik Scherer, Andreas Müller, and Sven Behnke. *Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition.* Ed. by Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–101. ISBN: 978-3-642-15825-4.

[18] M. Schuster and K.K. Paliwal. “Bidirectional Recurrent Neural Networks”. In: *Trans. Sig. Proc.* 45.11 (Nov. 1997), pp. 2673–2681. ISSN: 1053-587X. DOI: `10.1109/78.650093`. URL: `http://dx.doi.org/10.1109/78.650093`.

[19] Sebastian Thrun. In: *Wiley Online Library* (2006). DOI: `https://onlinelibrary.wiley.com/doi/full/10.1002/rob.20147`.