Fast & Cautious: Learning Autonomous Driving on a Simulator

April 25, 2019

Abstract

The majority of traditional autonomous driving systems rely extensively on non-visual sensing hardware to perceive the surrounding environment, whilst effective, these solutions are often very expensive and require significant computing power to run. Thus, vision-based systems offer an alternative that utilises common visual hardware to sense the environment which is both low-cost and low-complexity. In this report we aim to develop a system which can learn to predict the steering inputs required to control a car in a racing simulator from visual information extracted from race track images, in real-time. We present a system which attempts to learn the relationship between extracted features from a popular pre-trained image classification network (VGG16) and the car's steering controls supplied by a robot demonstrator. The method is tested on a variety of different tracks and shows good generalisation skills to previously unseen environments, however the system is limited to prediction on geometrically simple tracks with few corners and minimal undulation. Complex track environments such as winding roads and sharp corners present difficulties for the system, thus highlighting some limitations of the feature extraction and machine learning methods.

Keywords: *Autonomous driving, Navigation, Computer Vision, Feature Extraction, Machine Learning*

I certify that all material in this dissertation which is not my own work has been identified.

Contents

1 Introduction

This project designs and implements a system that can learn how to drive a car around a racing track in a simulator (The Open-Source Racing Car Simulator - TORCS [31]) perceiving only the visual information captured from a forward-facing camera. A transfer learning approach is utilised to apply the spatial knowledge of popular object classification models to predict steering outputs for race circuit images. The complete system demonstrates the viability of computer vision-based models for tackling the problem of autonomous driving in a manner that is effective and low-cost.

Autonomous driving is a well-established research field which has risen in exposure during recent years. Early work started during the 80's & 90's which evolved from the rudimentary models of vehicle control (such as following magnetic strips embedded in roads [8]) and demonstrated a step-up in technological ability, with systems showing the feasibility of motion control in complex urban situations and off-road environments [18, 25]. Reviews of this work credited the success of these complex systems primarily to the advances in machine sensing and vision, more specifically the increased availability of capable hardware due to increased microprocessor performance. This has facilitated more computationally expensive and hence more accurate detection models to be run on cheaper hardware and thus more viable autonomous driving models and components have emerged - "*simple tasks like lane recognition can now be performed by a standard PC*" [11].

The underlying principles of autonomous driving involve perceiving the observable environment, identifying the important information (in the form of 'features') from the scene, and using some model to map these features to steering and velocity outputs. Traditionally, autonomous driving models rely on extensive hardware systems and active sensors to perceive the environment in real-time, such as those in the 2007 Defence Advanced Research Projects Agency (DARPA) Urban Challenge [6]. Utilising computer vision techniques often traded accuracy and precision for speed, expense and simplicity, however, further advances in both computer vision and machine learning (a common pairing in autonomous driving) have led to camera sensing techniques becoming a practical and popular alternative [3]. Furthermore, influential automotive industry leaders such as Tesla have popularised autonomous driving systems that utilise computer vision to sense the environment, demonstrating its suitability for real-world autonomous driving at a low cost. This recent increase in viability of effective low-cost vision-based autonomous driving solutions constitutes the primary motivation for this project.

In this report a brief introduction to the research area of autonomous driving is presented, followed by a statement of the project goals. The architectural design of the system is described in Section 3.1, detailing the image processing and machine learning techniques used, followed by an account of the development of the system, including challenges encountered during the training process and additional findings. We present and examine the experimental results from offline testing in section 5 and transition into the critical evaluation of the system. The report is concluded with a final evaluation of the project and a brief discussion of the potential future work to build upon the system from the findings ascertained during the development and testing of this project.

2 Literature Review and Specification

As the largest incentivised autonomous driving project, the DARPA Grand Challenges have led to the development of some very capable autonomous driving solutions, through the use of

complex technology and extensive research, systems such as 'Stanley' [27] and 'Boss' [28] have demonstrated their excellent ability to navigate off-road and urban environments. However, utilising state-of-the-art sensing and computing technology as done in these projects often requires significant funding (as was allocated in the 2007 challenge) and can create additional maintenance problems. Furthermore, complex hardware and software integration frequently introduces more opportunities for bugs to occur (see S. Thrun's personal account [26]).

Thus, a research opportunity arises for low-cost autonomous driving solutions that can achieve comparable results to the traditional hardware-dependant systems. For many driving applications a high-cost specialised solution is simply not practical, for example where there is some contingency for error/inaccuracies (e.g. off-road driving), or cases where a vast array of physical hardware is impractical (e.g. discreet applications [30]). Computer vision is the most popular technology used in low-cost solutions, primarily due to extensive development of camera technology from industry pressure (e.g. smart-phones, digital cameras, etc.), but also because it lends itself nicely to machine learning techniques (also a thriving research topic) [12]. In addition to the benefit of being low-cost, vision-based solutions also provide promising advantages over LIDAR/RADAR systems due to the range at which drivable paths can be detected. For example, research has shown that lanes can be detected as far away as 100M with an 800x600 camera [13], whereas LIDAR and RADAR systems are limited to a range of approximately 20M. Stanley demonstrated how even a simple vision analysis system can identify obstacles far beyond the range of laser sensors as shown in Figure 1.

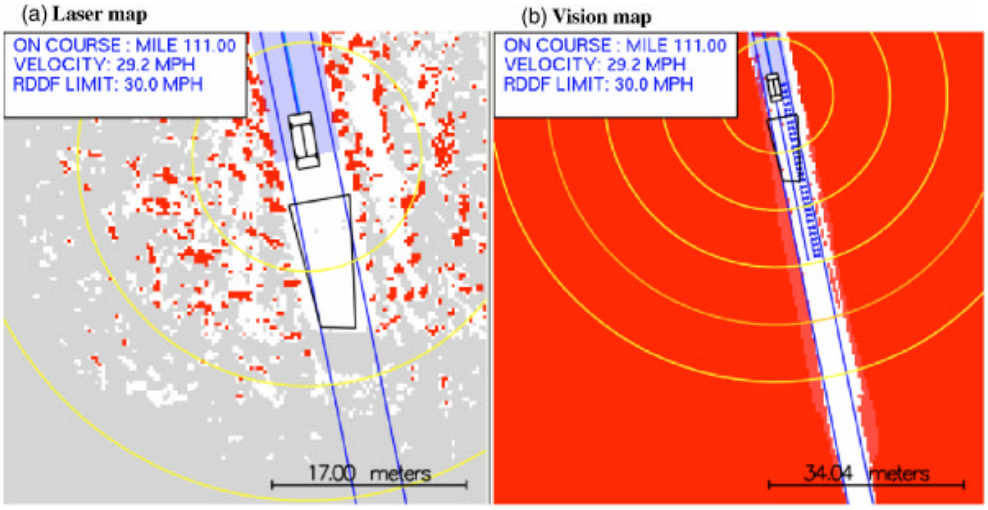


Figure 1: From [27], comparison of the laser-based and image-based mapper (red areas denote obstacles).

Building on previous work [21], this project aims to create a system that can learn (under supervision where necessary) to drive a car around a racing track in a simulator (TORCS) using visual information only. It has been shown that using simple statistical learning techniques a car can teach itself to steer around a track at a fixed speed, we aim to investigate the performance of a system that both utilises modern machine learning technology and contemporary image processing techniques. The enhanced capability of deep learning technology may facilitate the prediction of acceleration and braking; however this is an ambitious step-up and may not be achievable given the relatively limited time and resources available.

2.1 Requirements

- **Vehicle Control Output**
  At any stage of track navigation, the system must output a vehicle control response to

the racing simulator. This requirement summarises the need for the system to be robust, thus given any situation (image from the simulator) presented to it, the system must remain in control of the racing robot, and not 'give up'.

- **Navigation Prediction of TORCS Tracks**
  Whilst how fast the system navigates the TORCS race tracks is a key performance indicator, the system must be able to learn how to navigate around the race tracks in the simulation to begin with. Using only visual information, and learning corrections where necessary, the system must be able to predict navigation outputs for the TORCS race tracks that it is presented with, autonomously.

- **Real-time Performance**
  In order for the autonomous driving robot to race around the tracks in the simulation at speed the prediction system must be able to process the visual information in real-time. The image processing and learning modules must be efficient enough to output vehicle control responses at a rate that is sufficient for the robot to navigate the track smoothly. Previous research [21] has shown that the lower bound for satisfactory results is around 20 images per second, the system must achieve a minimum processing rate of 20fps or more.

2.2 Evaluation

- **Generalisability to New Tracks (Average Steering Error)**
  When exposed to new driving environments in the form of previously unseen racing TORCS racing tracks, the average steering angle output error (deviation from the human/robot demonstrator) is a direct performance indicator of the generalisability of the autonomous driving model. Thus, the average absolute steering angle error when the system is exposed to new track environments is a valid empirical measure of the generalisability and robustness of the system, two very important factors in autonomous driving solutions.

- **Robot Lap Time Hierarchy**
  Devising a track racing robot lends itself to an inherent criterion to evaluate the system's performance - lap time. There are several hard-coded racing robots included in the TORCS racing simulator. The included bots have access to extensive track data and pre-processed features, e.g. track-axis angle, individual wheel velocity, etc. It is therefore safe to assume that the bots are a good benchmark to compare our model against. The lap times (in seconds) achieved by the developed solution are a valid objective measure of the performance of the system.

3 Design

3.1 High-level System Architecture

In high-level terms, for a vision-based prediction system to autonomously navigate around a TORCS race track it must successfully form an association model that can learn the relationship between race track images and a set of corresponding driver controls. The model must be trained so that it can extract the features from an image that are relevant (i.e. those that depict the topology of the race track and where it leads) and correctly associate them with the desired output control (Figure 2 provides a visual overview). Thus, the functionality of the system can be characterised into three main tasks:

1. **Image Acquisition and Pre-processing** - Once the image is captured by the on-board camera (simulated) it undergoes some pre-processing before it is passed onto the machine learning algorithms, techniques such as noise reduction and contrast enhancement significantly help to ensure that the desired information is detected. More complex techniques such as Saliency Sampling can also be used to help improve the spatial sampling of input images for transfer to neural networks [20] (covered in more detail in Section 4.2.4).

2. **Feature Extraction** - Next, important features of the environment that the vehicle can perceive must be identified and extracted, most importantly the road area must be detected so that the vehicle can plan its route. This stage helps to highlight the important features for the machine learning algorithm to use and avoids learning unnecessary feature relationships. Various methods exist for feature extraction such as Histogram of Oriented Gradients (HOG), Canny edge detection and pre-trained image classification networks - these are discussed in more depth later in Section 4.2.

3. **Decision Making/Vehicle Control** - Finally, based on the relevant extracted information, the decision making system will decide on the appropriate vehicle control response so that it may follow the desired path around the track. A wide range of machine learning techniques such as simple k-means clustering, random forest regressors and neural network regression have proven to be viable methods for predicting the appropriate vehicle control responses, the performance of which are analysed and discussed in Section 4.

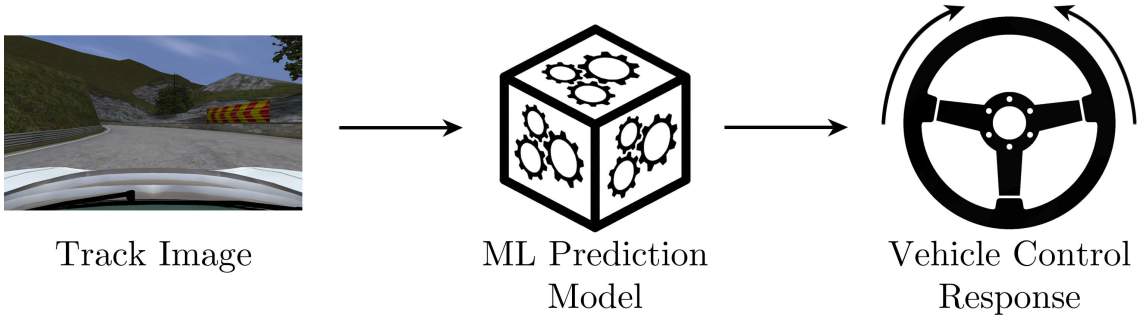


Figure 2: Overview of the vehicle control prediction system.

The system is defined by its ability to *learn* how to navigate around a race track. A supervised learning approach is implemented to achieve this whereby the 'ML Prediction Model' (a multilayer neural network - covered in more detail in Section 3.4) is presented with data captured from both human and race-robot drivers navigating a series of different tracks from which it can learn to imitate the vehicle control technique.

Race track images are first processed by a pre-trained neural network before being transferred to the ML Prediction Model. The role of the pre-trained NN is to compact the data and extract important spatial features from the scene, thus avoiding the need to learn the features directly - Section 4.2.3 provides a detailed description of this process. This important knowledge building stage of the system is achieved during the 'learning phase' - an iterative process in which the ML Prediction Model is trained (Figure 3 provides a visual overview).

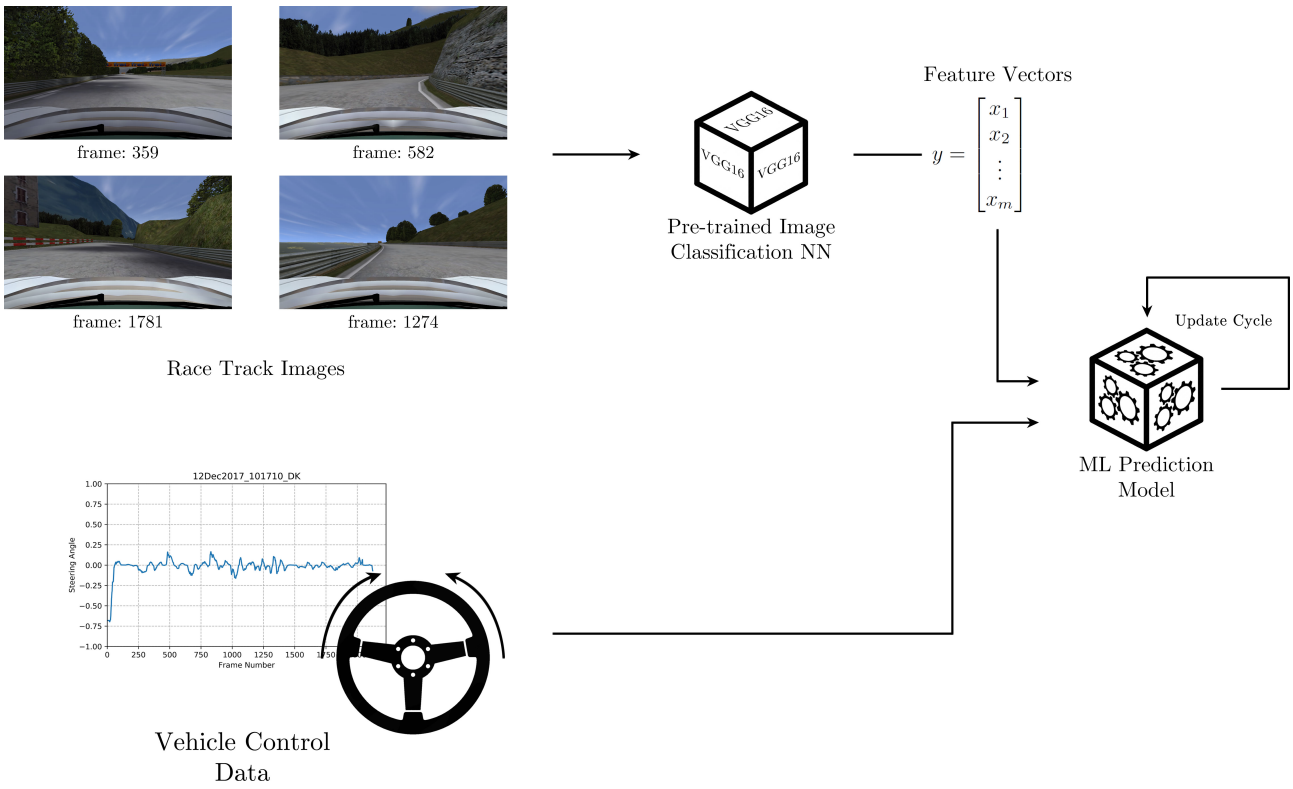


Figure 3: Overview of the system training phase.

After the ML Prediction Model has undergone sufficient training to predict the vehicle control responses to a reasonable level of accuracy it can then be used and tested for autonomous driving in real-time. The TORCS simulator permits the capture of images during the race simulation as they are rendered, these can then be extracted and pipelined into the feature extraction NN and the subsequent feature vectors supplied to the ML Prediction Model to output a live vehicle control response.

3.2 Racing Simulator & Data Collection

TORCS is a well-established open-source racing simulator that is widely used and adapted for scientific research applications primarily due to its readily extensible code and realistic physics engine [15]. An important property of the simulator that makes it suitable for this project is its detailed modelling of environmental properties, the race tracks are visually accurate and include realistic features such as global shadows, accurate surface textures and tyre marks [31]. This environmental accuracy combined with a high factor of diversity (due to the many different racing tracks) is especially pertinent to ensuring that the visual learning model is robust and generalises well.

Due to the open-source nature of TORCS, the codebase can easily be extended to capture and/or manipulate the race data in various ways, however for the purposes of this project only simple modifications were implemented to 'record' the details of a given race. For each race the following details were captured at a rate of 50Hz:

- **Image Data** - One forward facing 'camera' from the point of view of the car was rendered when running the simulation, the pixel values that were output from the simulation were simply captured at each frame.
- **Vehicle Control Data** - Various different parameters are made available by the simulator such as pitch, roll, car gear etc. However, the primary values relevant to the

project are acceleration, braking and steering angle - these were captured for each frame (and thus correspond to a captured image).

By capturing only the information required from the TORCS, the data representation for a race remains very concise, this helps to streamline the learning process and avoid unnecessary data processing. Furthermore, the result is also less overhead for the simulation so it can continue to run in real-time.

3.3 Feature Extraction

As identified when reviewing the research literature, an important barrier facing autonomous driving solutions is the perception problem [25]. Autonomous vision systems must pick up on the same perceptual cues that describe the environment as humans, primarily lane markings and road boundaries are required to determine the path of the vehicle. Whilst this may seem a trivial problem, there are many factors (often environmental) that make this a very difficult problem to solve, such as changes in road illumination and varying weather conditions. There are a multitude of different methods to 'perceive' the track environment, each of varying complexity. Several different techniques were tested during the development of this project (discussed in Section 4.2), the results of which concluded that a pre-trained image classification NN was most suitable for this application.

A major advantage of using a pre-trained network (such as VGG16 - the suitability of different networks is covered later in Section 4.2.3) is that it facilitates the implementation of a transfer learning approach whereby accurate prediction models can be constructed in an efficient and time-saving manner [19]. The visual knowledge (e.g. patterns) that the pre-trained model has learnt during its training for image classification provides a solid baseline to start learning different problems, such as extracting road boundaries, for example. It has been shown in previous research (as covered in Section 2) that relatively simple feature descriptors such as HOG can provide satisfactory results, however we opted to investigate the effects of using more complex and modern methods of feature extraction that has the potential to describe the images in a manner that can't be achieved with traditional methods.

The VGG16 Deep Convolutional Neural Network (DCNN) is a well-established image classifier model that has achieved excellent (92.7%) top-5 test accuracy on the ImageNet dataset and famously won the ImageNet Challenge in 2014 (see [24] for a detailed review of the network and its architecture from the authors). A desirable property of the VGG16 network that makes it extremely popular in computer vision research, and suitable for use in this project, is its discrete architecture (as depicted in Figure 4). In a simplified view, the network consists of two parts:

1. **Convolutional Base** - A series of convolutional and pooling layers which act as a fixed feature extractor for the network.
2. **Classifier** - A fully-connected layer which aims to classify the image from the activated features in the convolutional base.

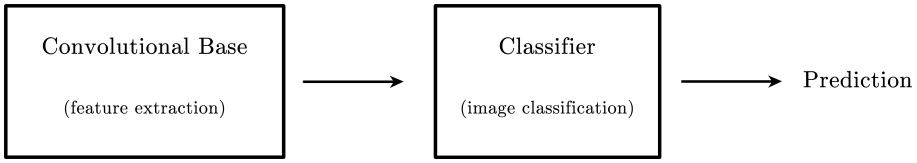


Figure 4: Simplified outline of the VGG16 network architecture.

As a result of the modular-like network design the fully connected classifier layers (also referred to as the 'top') can be easily omitted when using the network, this 'exposes' the convolutional base and results in the ability to output raw feature vectors from the network. For the purposes of this project the VGG16 feature vectors that describe an image can be left unmodified and supplied directly into the ML Prediction Model (see Figure 5) which acts in place of the classifier and uses the feature vectors to predict the vehicle response outputs. Due to the inherent 'deep' nature of the VGG16 network, the output feature vectors are significantly larger ($|512 \times 7 \times 7|$) and thus more complex than most feature descriptors (e.g. HOG).

The high data dimensionality of feature vectors can present a problem when input into the ML Prediction Model if it is to remain efficient enough to run in real-time. To help combat this optional average pooling layers in the VGG16 network are used to down-sample the output into a more manageable feature vector size ($|512|$). In addition to compacting the output of the network, average pooling can also help to identify and extract large, smooth features from images [4], which can prove effective for discerning the curvature of the race track. The effect of applying different pooling methods is investigated and discussed further in Section 4.2.3.

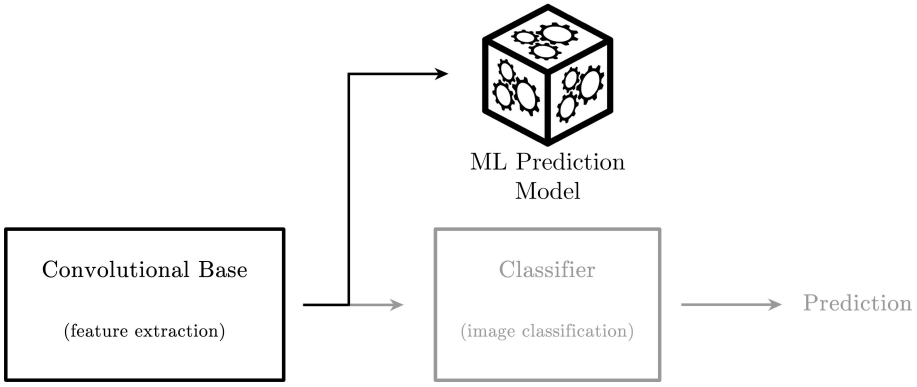


Figure 5: Outline of the linked VGG16 and ML Prediction Model architecture.

The VGG16 network is designed to be efficient so that its applications are extended to real-time problems and thus imposes a limit on the image size that is input to the network (224x224) to improve processing speed - smaller images reduce the computation time of feature vectors exponentially [19]. The images captured from the TORCS simulator therefore undergo a downscaling process from the 1920x1080 resolution that they are captured at, this process is computationally inexpensive, and the resulting image still contains enough information to extract large geographical features, such as roads. Figure 6 provides an overview of the complete feature extraction process.

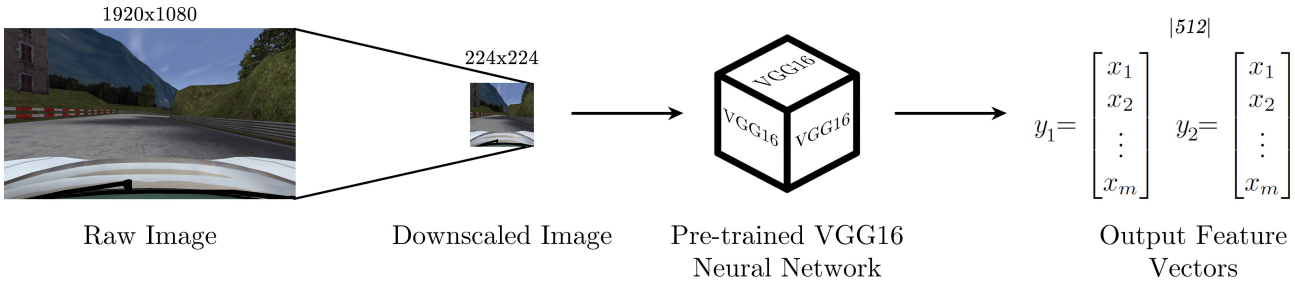


Figure 6: Feature extraction process overview.

3.4 ML Prediction Model

The ML Prediction Model's architecture consists of a single hidden layer Neural Network (NN) (See Figure 7).

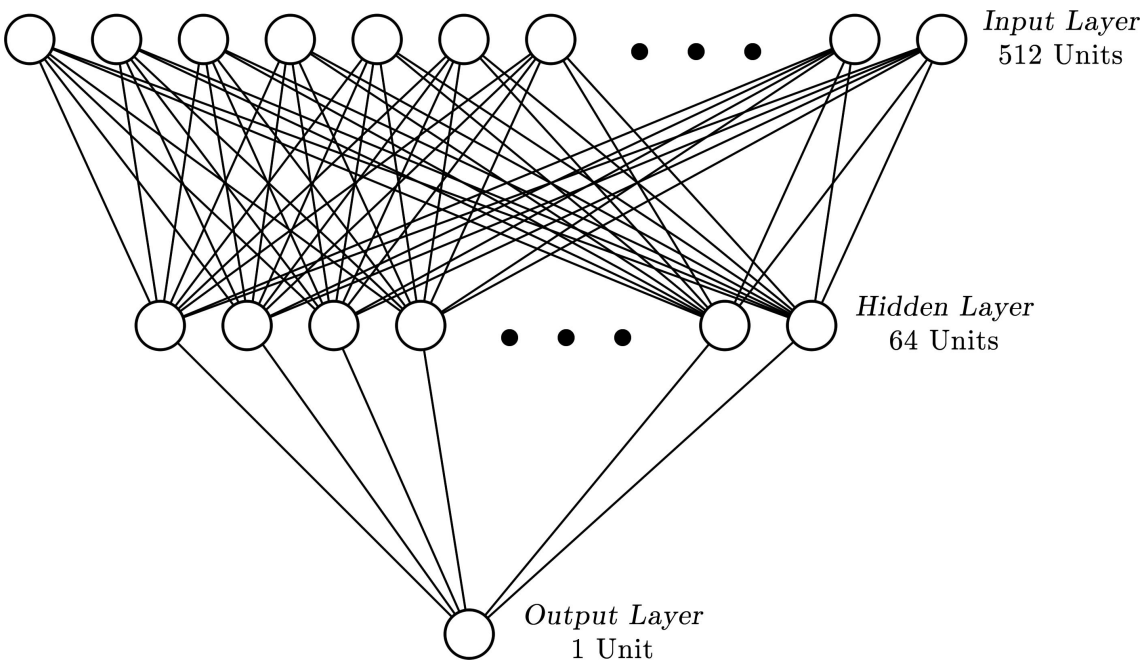


Figure 7: Outline of the ML Prediction Model network architecture.

The input consists of 512 neurons which receive data in the form of VGG16 feature vectors, a Rectified Linear Unit (ReLU) activation function is employed for each unit in the layer. Each of these 512 input units is fully connected to the hidden layer of 64 units (also activated using a ReLU function), which is in turn fully connected to the linearly activated single-unit output layer that returns the predicted steering angle.

The development of the prediction network was designed around two key requirements:

- **Time-performance** - Given the complex (and relatively time consuming) VGG16 feature extraction process, it is important that the prediction network processes the feature vectors in a timely manner so that the overall time-complexity of the system is kept to a minimum. The depth of the network is therefore kept to a modest 3 layers, resulting in only 577 neurons in total.
- **Feature Vector Input** - As the prediction network is essentially replacing the classifier section of the VGG16 network it must integrate well with the output feature vectors from the convolutional base (See Figure 4). The dimensions of the feature vectors match the number of neurons in the input layer to give the network an optimum chance to learn the relevance of each feature identified in the image.

4 Development

The development required for this project is divided into 3 logical parts:

1. An initial analysis of the data must be performed to identify potential import trends and/or sub-problems.
2. Different image processing techniques and feature extraction methods must be investigated and implemented.
3. Different machine learning techniques which process the extracted features must be explored and implemented.

In practice, however, the development process has been less structured and chronological than the steps detailed above and more of a dynamic process. For example, the suitability of the different feature extraction methods cannot be validly assessed without integration with a machine learning model to test the effectiveness and performance of the technique, thus, the development process often 'jumped' between the different stages of the development plan throughout the project. Nevertheless, the development and implementation procedures are presented in this section as distinct entities for ease of understanding for the reader.

4.1 Initial Data Analysis

As is the case with most research problems, it is often wise to 'take a step back' and analyse the situation (such as the data available, available hardware etc.) before commencing development. Prior to tackling the machine learning aspect of the project we investigated simplifying the problem by analysing the vehicle control data (collected from human and robot drivers) in an attempt to identify any patterns in the data from which a simpler sub-problem may emerge.

We hypothesised that the 3 vehicle controls (acceleration, braking, steering) are most often used in velocity-direction pairs (e.g. accelerate left, brake right, etc.), thus there may exist a relationship in the data whereby it can be clustered into velocity-direction pairs. The existence of clear clusters in the data would create a less complex classification problem that would provide a good starting point for tackling a larger regression problem. A simple k-means clustering algorithm was implemented and tested on a set of vehicle control data from a human demonstrator with differing numbers of clusters (see Figure 8).

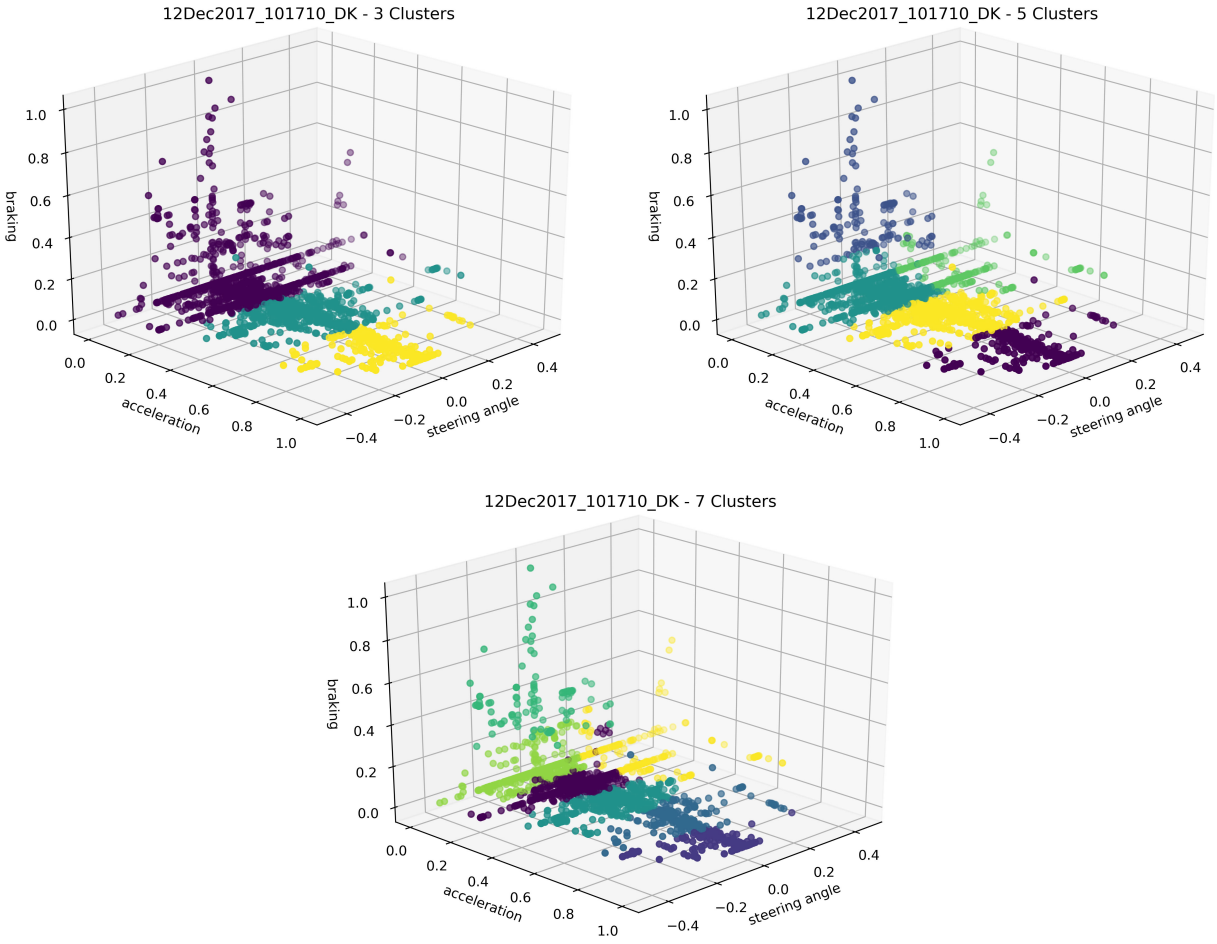


Figure 8: Results of k-means clustering with 3, 5 and 7 clusters.

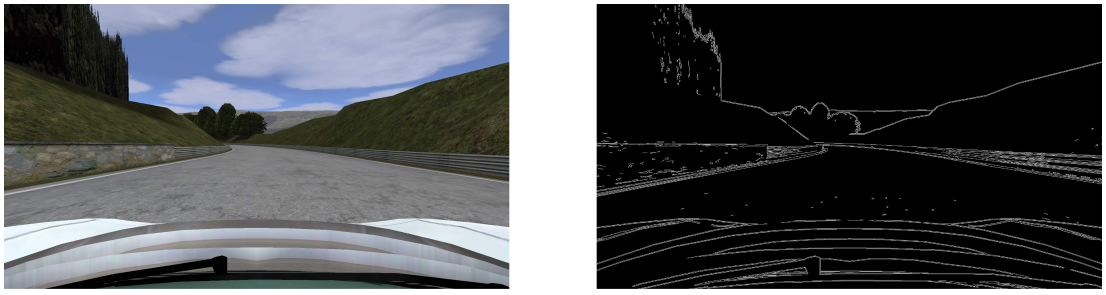
The k-means clustering algorithm has clearly identified groups in the data set, however there are few signs of the velocity-direction pairs as predicted. The graph showing k-means with 3 clusters best illustrates the patterns identified by the algorithm, the data is categorised primarily by the acceleration parameter and demonstrates a poor ability to be separated by steering angle. Even with a higher number of clusters, the data is still parameterised heavily by acceleration, few groups emerge that identify patterns in steering angle. The root of this issue appears to be that there are only small variations in the steering angle, the majority of values are often within a -0.2 to 0.2 range for both the human and robot drivers, thus it is difficult to identify clear clusters in the compact range of data points. However, a reasonable explanation for this is that the road-style tracks in TORCS used for this project (more detail on the TORCS tracks is covered in Section 5) contain predominantly wide sweeping turns that require relatively low steering input compared to the tight turns of a race track, for example.

4.2 Image Processing/Feature Extraction

As detailed in Section 2, the scope for this project is to create a system that can learn to drive around a race track using visual information only. Several different feature extraction techniques were developed and tested before arriving on the final design of using a pre-trained image classification network (VGG16). The different methods and their effectiveness will be discussed in this section.

4.2.1 Canny Edge Detection

Canny edge detection is a popular technique in object detection and image classification research [16], it is often used to extract structural information from an image and can help to reduce the amount of information in an input to a post-processing algorithm (the reader is pointed to [7] for a detailed explanation of the Canny edge detector). Given its effectiveness, a simple canny edge detector was implemented to test its potential usefulness in feature extraction for the application of this project.



Raw Image

Canny edge-detected Image (σ=0.9)

Figure 9: Testing Canny edge detection on a sample race track image.

Various iterations of tweaking the sensitivity parameter (σ) were necessary to achieve a satisfactory trade-off between correctly identifying the shapes in the image and reducing the amount of noise (e.g. surface textures) from the edge detector. Figure 9 shows the results of running the Canny edge detector on a race track image from the simulation with $\sigma = 0.9$, this sensitivity value proved to yield the best and most consistent results from the sample dataset. As can been seen in the resultant image, the edge detector has correctly identified the main structural features from the image including the road ahead and some unimportant features such as clouds in the sky have been removed.

Initially the Canny edge detector appeared to be a promising technique to incorporate into the system, however upon further investigation there are some flaws with the Canny edge detector (and edge detection as a whole) that limit its applicability to this project. Firstly, the algorithm required extensive tweaking to achieve satisfactory results on the sample image, however this is only one section of a particular track thus raising concerns as to how well the technique will generalise to different environments (e.g. shadows from hills & trees, different road types etc.) where the fixed σ value would not be appropriate. Furthermore, as corroborated by lane detection research, edge detection methods are particularly susceptible to road artifacts such as cracks, car shadows and skid marks [1]. The Canny edge detector often seemed to pick up many illusory edges on the road directly ahead of the car which would have a negative impact on performance when paired with a ML system as it would impede the identification and learning of the important geographical features, like road curves.

The Canny edge detector was therefore found to have limited applications to the project. Despite being effective at finding sharp edges in an image, the inherent nature of the method is somewhat crude and removes a significant amount of information from the image that may be important to a machine learning algorithm for determining which direction to steer. Furthermore, as mentioned in Section 3.3, autonomous driving systems rely on perceiving the same visual cues as humans to describe the environment, thus relying simply on a limited set of edges extracted from an image was determined to be an ill-advised approach to tackling the problem - a less binary feature extraction technique would be more suitable.

4.2.2 HOG Features

The HOG descriptor is a feature extraction technique which uses the occurrence distribution of intensity gradient orientations in localised portions of an image to describe features and shapes in an image. By dividing an image into a number of sub-regions called *cells*, the histogram of gradient directions or edge orientations for the pixels within each cell can be computed. The resultant feature vector is simply a concatenation of these histograms. In practice, overlapping descriptor blocks are required to achieve good results, for more information on the technique we refer the reader to the original publication [10].

The HOG descriptor should be effective at describing the geometry of the scene for our application.The clear (and more importantly high-contrast) boundaries separating the road/track from the surrounding environment should allow the feature descriptor to easily identify and extract the track for use by further machine learning algorithms. Using the HOG functionality obtained from the `scikit-image` library [29], a simple HOG feature descriptor was implemented to test its effectiveness at extracting the important information from the track images when combined with a preliminary ML Prediction Model.

Before feature extraction occurs, the images are first converted to grayscale and down-scaled for increased processing performance. They are then passed to the HOG feature descriptor which is initialised with a set of parameters, including the number of cells per block (CPB) and number of pixels per cell (PPC). Figure 10 provides an overview of the HOG feature extraction process (where the resultant feature vector is visualised for demonstration).

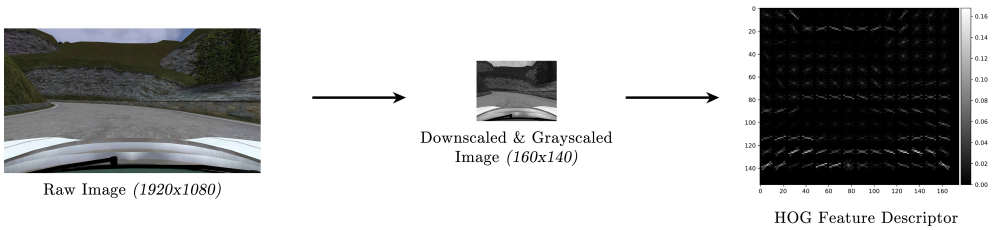


Figure 10: HOG feature extraction procedure.

As with the implementation of the Canny edge detector, the various different initialisation parameters for the HOG process have a large effect on the output feature vectors, and hence the performance of the ML Prediction Model. The number of PPC is an effective resolution of the HOG algorithm, a low PPC value results in a large output feature vector which will describe the image in great detail, however it will take longer to compute (and will consequently increase further processing time in the ML Prediction Model). As with most computational problems, a trade-off must be made between the depth/richness of the feature vector data (too many PPC will describe the image too generally) and compute time. A satisfactory balance between these two variables was achieved with a cell size of 10×10 pixels.

The HOG algorithm also provides functionality to contrast-normalize the histograms for each cell by computing an intensity value across a larger portion of the image, called a *block*, from which the cells within the block can be normalised. This results in better invariance to illumination, shadowing, etc. [10]. A CPB value of 8×8 was found to be optimal for our application when tested with the ML Prediction Model, providing sufficient robustness against the environmental artifacts from the simulation.

Figure 11 shows the results of testing the model using HOG features on one of the simpler tracks in the dataset, 'E-track 5' (a track previously unseen by the model), which is a small course with only 6 corners. The plot shows some correlation between the predicted steering values and those of the robot demonstrator and an average absolute error value of 0.033. The curves (represented by 'crests' and 'troughs' on the plot) are mostly followed well, however there are some sections of the track where the predicted values fluctuate significantly, representing the system picking up noise in the HOG defined scene. Furthermore, the change in predicted steering responses is noticeably sharp in places and is not indicative of the smooth inputs that are required to navigate the track, like those of the demonstrator.

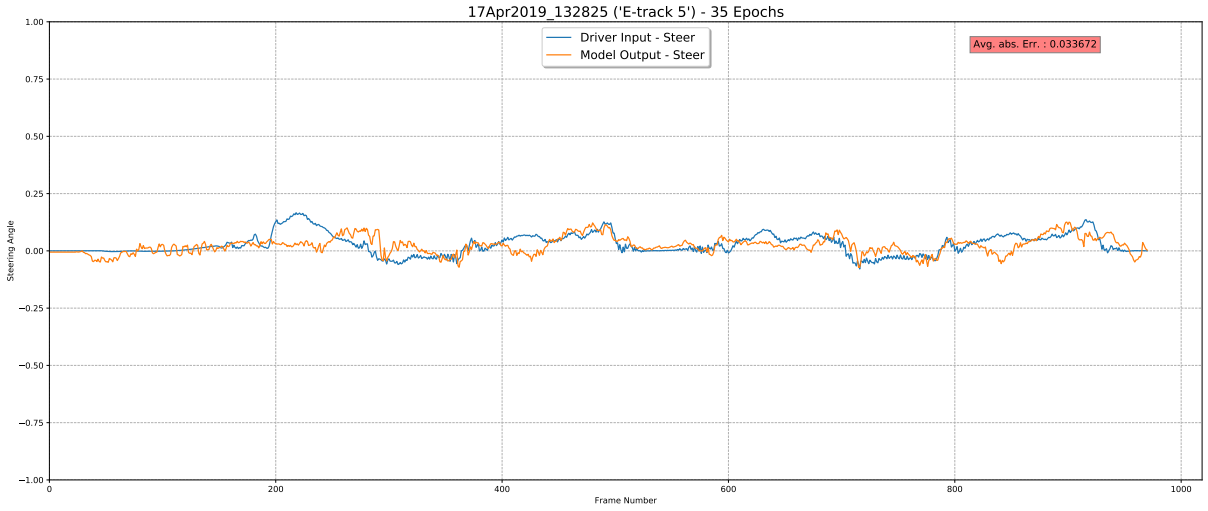


Figure 11: Performance of the HOG features on a previously unseen track.

The implementation and use of the HOG feature descriptor was an effective starting point to the feature extraction phase of the project. The time required to process the images from the simulation was low which is a significant advantage of this method. The previously discussed results show that it can characterise the geometry of the race track images well enough for the ML Prediction Model to learn basic driving actions on, however the prediction network struggled in places to maintain a smooth change in steering output, indicating some limitations of the method. This helped to set an objective baseline for the ML Prediction Model, however a key motivation for this project is investigating the effectiveness of more contemporary computer vision techniques, such as pre-trained image classification networks (e.g. VGG16).

4.2.3 Convolutional Neural Networks

CNN's have been used for image classification and object detection tasks for several decades, however it is only in recent years that the power and versatility of the descriptors extracted from CNN's have been exploited. The image representations/features extracted by publicly available networks such as VGG16 [24] and Xception [9] can be used to tackle a diverse range of tasks in computer vision and are quickly becoming the "*primary candidate in most visual recognition tasks*" [23].

Due to the ease and simplicity of the available implementations we use Keras (with Tensorflow backend) to load and manipulate the pre-trained CNN's for feature extraction. Keras provides a vast array of deep learning models (also referred to as *applications*) alongside pre-trained weights. The different models are implemented with a flexible design, thus they can be easily manipulated for the needs of the user, for the purposes of this project they are used as feature extractors. As detailed in Section 3.3, the convolutional base of the model is retained as the feature descriptor, and the classifier layer is omitted, this is easily achieved by setting the `include_top` argument at network initialisation to `False`.

Two of the most popular and highest performing networks were chosen to test the performance of CNN feature extractors:

- **VGG16** - A 23-layer network with 23 thousand parameters, developed by the Visual Geometry Group (VGG), University of Oxford (92.7% top-5 accuracy on ImageNet validation dataset). Output feature vector size: $|512 \times 7 \times 7|$
- **Xception** - A 126-layer network with 138 thousand parameters, developed at Google Inc. (94.5% top-5 accuracy on ImageNet validation dataset). Output feature vector size: $|2048 \times 10 \times 10|$

Before integration with the ML Prediction Model, the image classification network weights were first verified against a set of sample images from the ImageNet database to ensure that the weights resulted in the appropriate classifications. An accuracy rate of 96% was achieved over 100 images, thus confirming the correct functionality of the network. Whilst time-consuming, it is important to verify each component of the prediction model individually as debugging the fully integrated system is time-consuming given the 'black-box' nature of neural networks, especially when using pre-trained weights [14].

The feature vector outputs of the two networks were fed into a simple 3-layer NN (an early evolution of the final ML Prediction Model detailed in 3.4) to ascertain an early indication of the suitability of the different networks for this project. Each network was trained on robot driving data from half of the available training data/tracks and then tested on a previously

unseen road track of moderate complexity, 'Allondaz'. The results of this initial testing are shown below in Figure 12.

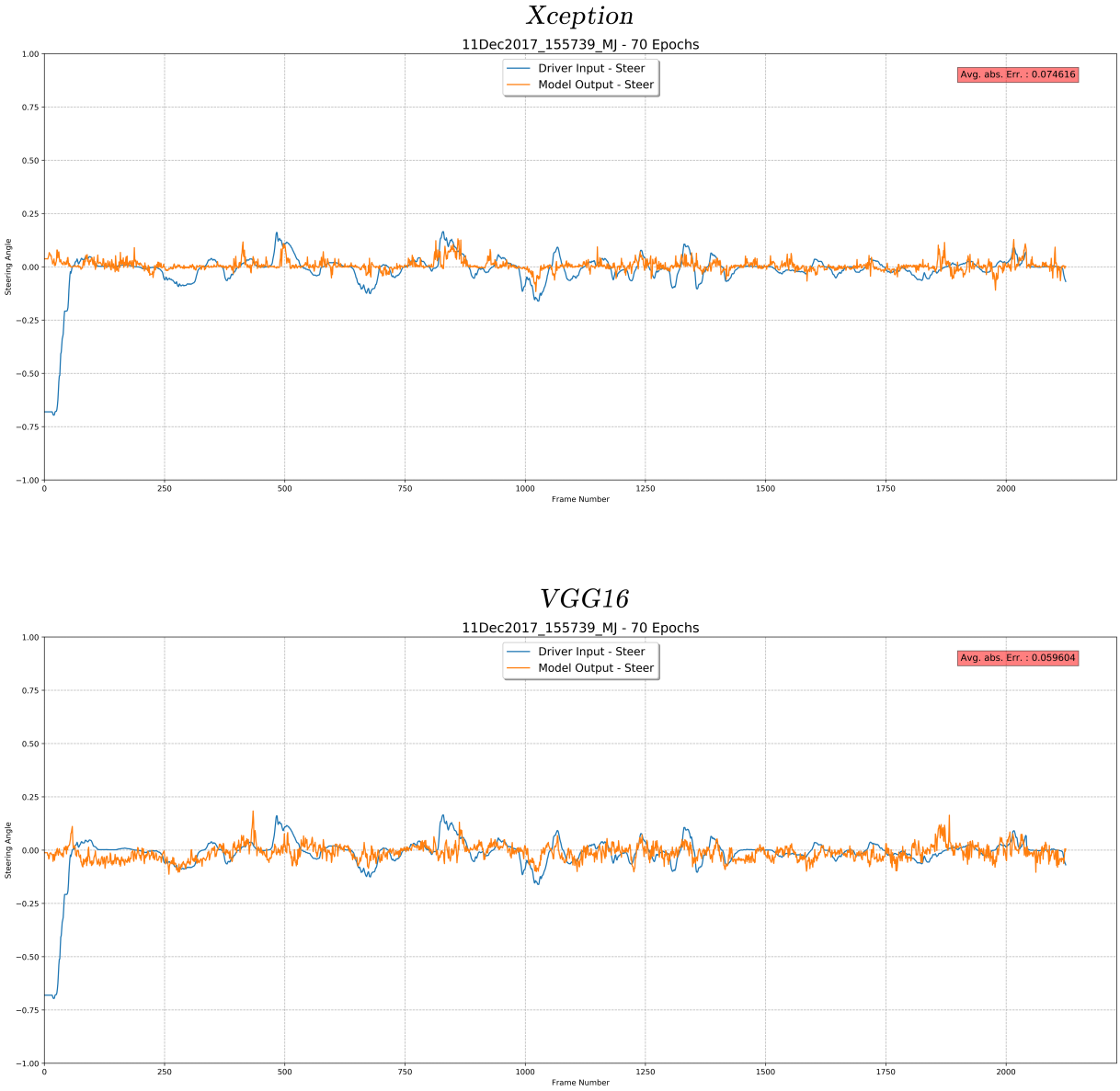


Figure 12: Performance of the Xception (**top**) and VGG16 (**bottom**) feature extractor networks.

The two graphs demonstrate quite different performances of the two networks, the prediction from the Xception network features shows little deviation from the centre steering angle where the model is perhaps 'missing' shallow curves and turns, only the large (requiring a steering angle of $\sim |0.125|$ or more) turns appear the elicit a response from the model, as a result the mean average error ($|actual - predicted|$) is 0.075. The VGG16 feature vectors result in more accurate outputs from the prediction network where an average mean error of 0.060 is achieved. As the plot shows, the predicted steering outputs match the actual values more closely and follow the fluctuations in steering inputs much more readily than the Xception network.

The performance difference between the two models can most likely be attributed to the difference in structure, primarily the network depth. The Xception network has significantly more layers and parameters than the VGG16 network and the resultant feature vector is also significantly ($\sim 8\times$) larger. The higher complexity of the Xception network may cause more

detailed features to be learned that are useful for image classification but don't necessarily transfer to the application of this project where a desired feature descriptor delineates the shape and direction of the road. Furthermore, the large feature vector output may negatively impact the performance of the prediction model by causing it to learn unnecessary features of the image, whereas the smaller output of the VGG16 network may give the prediction network a better chance of learning characteristics of an image that better describe its topology in relation to the input steering values. The promising and superior performance of the VGG16 network demonstrated that it was effective at extracting useful features from the race track images for the prediction model to output sufficiently accurate steering values, thus this method was carried forward into further development.

A problem that we encountered when integrating the output of the VGG16 network with the ML Prediction Model was that the large size of the feature vectors has a significant effect on the compute time required to generate and process them. As stated in Section 2.1, it is important that the system achieves real-time performance, thus processing $|512 \times 7 \times 7|$ dimensional feature vectors was not desirable. One method that we found to be highly effective at reducing the descriptor dimensionality was *pooling*, this involves downsampling the output from the convolutional layers in the network so that the final output is smaller in size. The Keras implementation of the VGG16 network supports both average and max pooling, and when activated, reduce the size of the output feature vector down to $|512|$ which is much more suitable for the applications of this project.

Pooling Method	Average Absolute Error	Feature Vector Dimensionality
None	0.062	$\|512 \times 7 \times 7\|$
Max	0.058	$\|512\|$
Average	0.052	$\|512\|$

Table 1: Performance of different pooling methods for the VGG16 network (averaged over 3 runs).

Table 1 shows the results of testing the different pooling methods for VGG16 when combined with an early ML Prediction Model. For each pooling method, the network was trained on robot driving data from half of the available training data/tracks and then tested on a previously unseen road track of moderate complexity, 'Allondaz'. Somewhat surprisingly, the addition of optional pooling for both methods showed a decrease in the average error over the runs with no pooling applied, despite the loss of information through each pooling layer. The accuracy increases for both pooling methods could be attributed to the increased robustness to spatial variations that results from generalising the output from the convolutional filters [32].

Furthermore, it can be seen that average pooling yielded the largest decrease in average error for the predicted steering values, a possible explanation for this is the smoothing effect that average pooling layers have on the network. Dissimilar to max pooling, all information from the previous layer is retained (as a global average) which helps to extract smooth features from the race track images that the ML Prediction network appears to learn more effectively from. It could be conjectured that large flowing features, such as road curves, are better characterised by 'soft' features that result from average pooling than the 'sharp' features that max pooling generates.

4.2.4 Saliency Sampler

A large constraint of this project is the time and computation required when processing the images from the race simulation. The high computational complexity that is associated with extracting features from high resolution images means that they must be downscaled before being processed to achieve reasonable time-performance from the system. For use in the VGG16 network implemented in this project, the images are uniformly downsampled to 224×224 resolution, this is an effective method to reduce the input size to the network, however it results in an indiscriminate loss of information from the original image used to extract features from.

To help reduce the negative effects of information loss from downsampling, we implement a novel saliency-based sampling method proposed in [20], which learns to allocate pixels in an input image to regions that are particularly important to the given task. The saliency sampling layer acts to emphasise task-relevant portions of the image and suppress irrelevant parts, and by doing so more of the important information in the image is retained during downsampling for input to the VGG16 network.

The code for the saliency sampler is provided in a format so that it can be 'plugged' into an existing CNN, in our case the VGG16 & prediction network, and thus works as follows:

1. Downsample the original image I to generate a low-resolution image I_d.
2. The saliency network f_s uses I_d to compute a saliency map $S = f_s(I_d)$, where the different portions of the image are assigned weights based on their task-relevance.
3. The deterministic grid sampler g is used to sample the high-resolution image I according to the saliency map, resulting in a resampled image $I_s = g(I, S)$ with the same resolution as I_d.
4. The original task network (VGG16) f_t is used to compute the final output $y = f_t(I_s)$.

The saliency sampler can then be trained alongside the Prediction network (the VGG16 network is left untouched) and results in the race track images being sampled in a manner that accentuates the features that are important for steering value prediction. As shown in Figure 13, the sampled images often decreased the size of the car bonnet (a by-product of the TORCS system) as this had little/no impact on the direction of the vehicle and the defining road features such as the barriers and corners are bloated.

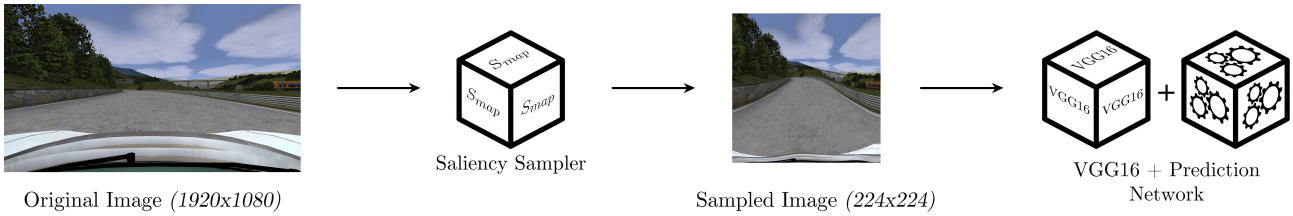


Figure 13: Outline of the data pipeline with the additional saliency sampler.

Table 2 presents the prediction accuracy results of the system with and without the saliency sampler. For each variation of the system, the network was trained on robot driving data from half of the available training data/tracks and then tested on a previously unseen road track of moderate complexity, 'Allondaz'. The results show a small ($\sim 6\%$) decrease in absolute error reduction after implementation of the saliency sampler. Whilst only a minor improvement,

the accuracy increase demonstrates that task-relevant downsampling has a positive effect when applied to the task of extracting features from race track images for autonomous driving.

Saliency Sampler Layer	Average Absolute Error
No	0.052
Yes	0.049

Table 2: Performance of the prediction network with and without the additional saliency sampler layer (averaged over 3 runs).

4.3 Machine Learning Techniques

As stated in Section 3.1, an integral part of the autonomous driving system is the association model that must learn the relationship between the race track images (in the form of feature vectors) and the corresponding driver input controls. There are numerous different machine learning methods and algorithms of varying complexity that could be used to form such an association model, in this section we detail the different machine learning techniques that were tested and report on their effectiveness.

4.3.1 Statistical Techniques

To achieve a baseline upon which to gauge the effectiveness of modern machine learning methods we first implemented two well-known statistical machine learning techniques: k-nearest neighbours (KNN) and the more sophisticated random decision forests (RDF). We do not cover the workings of each method in this report, for in-depth explanations of KNN and RDF methods we refer the reader to [22] and [5] respectively. For each method the model is trained on a small subset of the available tracks and then tested on 'E-track 5', one of the simpler tracks with few corners and few environmental distractions.

KNN

We use the `scikit-learn` library [17] to implement a simple KNN regressor to predict steering output values from the VGG16 feature vectors. Table 3 shows the prediction accuracy results from the KNN regressor for different numbers of neighbours.

Number of Neighbours	Average Absolute Error
2	0.080
4	0.075
6	0.072
8	0.070
10	0.068
50	0.056

Table 3: Performance of the KNN regressor with varying number of neighbours.

The results show that the KNN algorithm is somewhat effective at predicting the correct steering output and becomes more accurate for higher values of k (number of neighbours). However, k cannot be infinitely increased as this would result in underfitting of the data, thus there are limitations to the applicability of the KNN model to the project if a large value of k must be used. Furthermore, as each prediction requires a full search on the dataset for the k-nearest neighbours, the time performance of the algorithm becomes a significant issue as k

increases, especially given the high dimensionality of the VGG16 feature vectors. The mediocre performance of the KNN algorithm could be attributed to its poor generalisability; all the data input to the KNN algorithm is treated with equal weighting and thus there is no ability to discern important features from an image, instead similar features that do not pertain to the geometry of the road may adversely affect the predicted steering output, e.g. clouds, trees, etc.

RDF

We use the provided RDF regressor from the `scikit-learn` library [17] to predict steering output values from the VGG16 feature vectors. Table 4 shows the prediction accuracy results from the RDF regressor for different numbers of trees in the forest (other parameters such as the maximum tree depth and the maximum number of features were left unchanged/at default values).

Number of Trees	Average Absolute Error
10	0.065
20	0.060
30	0.059
40	0.055
50	0.052
200	0.044

Table 4: Performance of the RDF regressor with varying number of trees (estimators).

The results show an immediate increase in accuracy performance over the KNN algorithm, and like KNN there is a reduction in average error as the corresponding training parameter (number of trees) increases. The returns from increasing the number of trees does diminish over time, however, so there is a compromise between performance and accuracy to be made. Despite the linear time-complexity of scaling the RDF algorithm to a large number of trees, the real-time performance applications of this project limit the number of decision trees that can be used.

Unlike the relatively simple nature of KNN, the random selection of inputs (or combinations of) used to grow the decision trees in RDF makes the algorithm more robust to noise and outliers in the input [5]. This robustness of the algorithm helps it to generalise better between environments as the important features that define the road geometry and therefore more significantly affect the steering response will be identified by the algorithm, and noisy features will be ignored, such as environmental terrain/texture differences, weather, etc.

4.3.2 Neural Networks

In an attempt to improve upon the results of the more traditional statistical machine learning methods, the next logical technology to utilise is neural networks as they excel at learning complex relationships between datasets (as is the case with autonomous driving). Neural networks have also matured enough as a technology that there exists a large support base facilitating their implementation, which significantly eases their development. We use the well-known Keras library to implement our prediction network (also referred to as the ML Prediction Model) which is tasked with learning the relationship between the race track images in the form of VGG16 feature vectors and the corresponding vehicle controls i.e. steering angle input.

Various different network designs were experimented with during development, such as deep networks with 5-15 layers, more/less neurons at each hidden layer, dropout regularisation

between layers, etc. However, the changes to the network often had vastly different impacts on prediction performance and little clear patterns and/or relationships could be ascertained that would result in a consistent increase in performance. Thus, a relatively simple network design was arrived at (as detailed in Section 3.4) relatively early in development so that time was not wasted endlessly testing different infinite network architectures and has only been subject to minor tweaking.

The model demonstrated good convergence during training and generally stabilised on a given loss value at around 35+ epochs (See Figure 14), showing that the network is relatively quick to learn, which is a key benefit given the large size of the datasets (often 3000+ images per track). A mean absolute error loss function was found to elicit consistently good performance from the prediction model when compared to other common loss functions, such as mean squared error. This could be mostly attributed to its robustness against outliers in the data, i.e. large steering deviations when turning corners, where more punishing loss functions would dissuade the prediction of the fluctuating steering values that are required to navigate the tracks with multiple sharp corners.

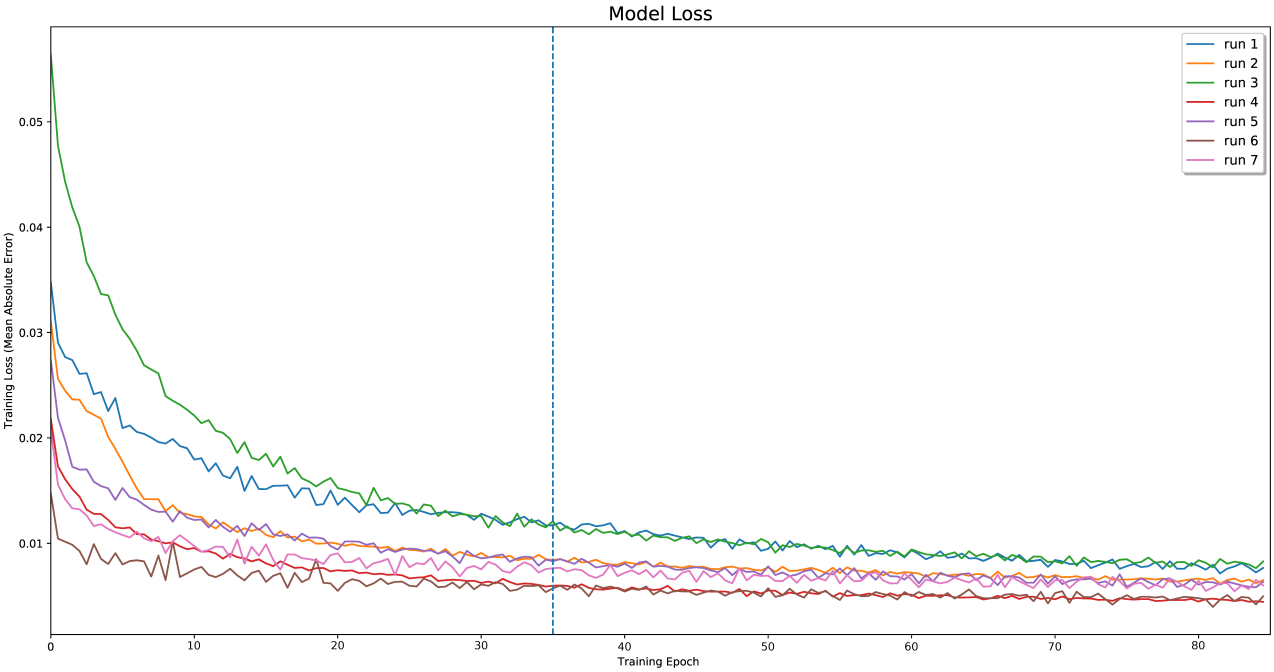


Figure 14: Example training loss history plot (the vertical line indicates 35 epochs).

5 Testing

Available in the TORCS simulator are 25 different tracks (See Figure 15) comprised predominantly of flat formula-1 style courses, however a variety of road and speedway tracks are also offered. The high environmental diversity between the tracks facilitates testing of how well the system generalises to new environments and unfamiliar road types; the varying complexity (i.e. difficulty) of the tracks provides a good scale to incrementally test and judge the performance of the system. We expect that the large number of tracks/environments will help to create a robust prediction model as the changing scenery should be identified as irrelevant noise by the system and the track isolated as the important feature that is consistent across the data.

Figure 15: The available tracks in the TORCS simulator.

For each of the tracks, image and vehicle control data is collected from a robot demonstrator, "Simplix", a state-of-the-art bot which won the 2009 TORCS Endurance World Championship [2], to train the model with. Each dataset contains one lap of the given circuit in the anti-clockwise direction, images and the corresponding vehicle control inputs are recorded at a rate of 50 images/control inputs per second of driving.

5.1 Experiments/Early Validation

Due to the time constraints encountered during this project we are unable to carry out any 'online' testing of our prediction system on the live TORCS simulator, instead we focussed our efforts into exploring the offline performance of our model. We analyse the predicted steering responses output from our model compared to corresponding robot and human data as a metric of how well the system performs. The correlation between the actual and predicted vehicle controls are a direct indicator of how well the system has learned to imitate the human and robot drivers, and thus how to drive around a race track. The vehicle control data is best represented by plotting the steering angle (-1 = full right, 1 = full left) over time (frame number), an annotated example where images from different sections of the track are mapped to the corresponding time frame on the plot is provided in Figure 16 for the readers benefit.

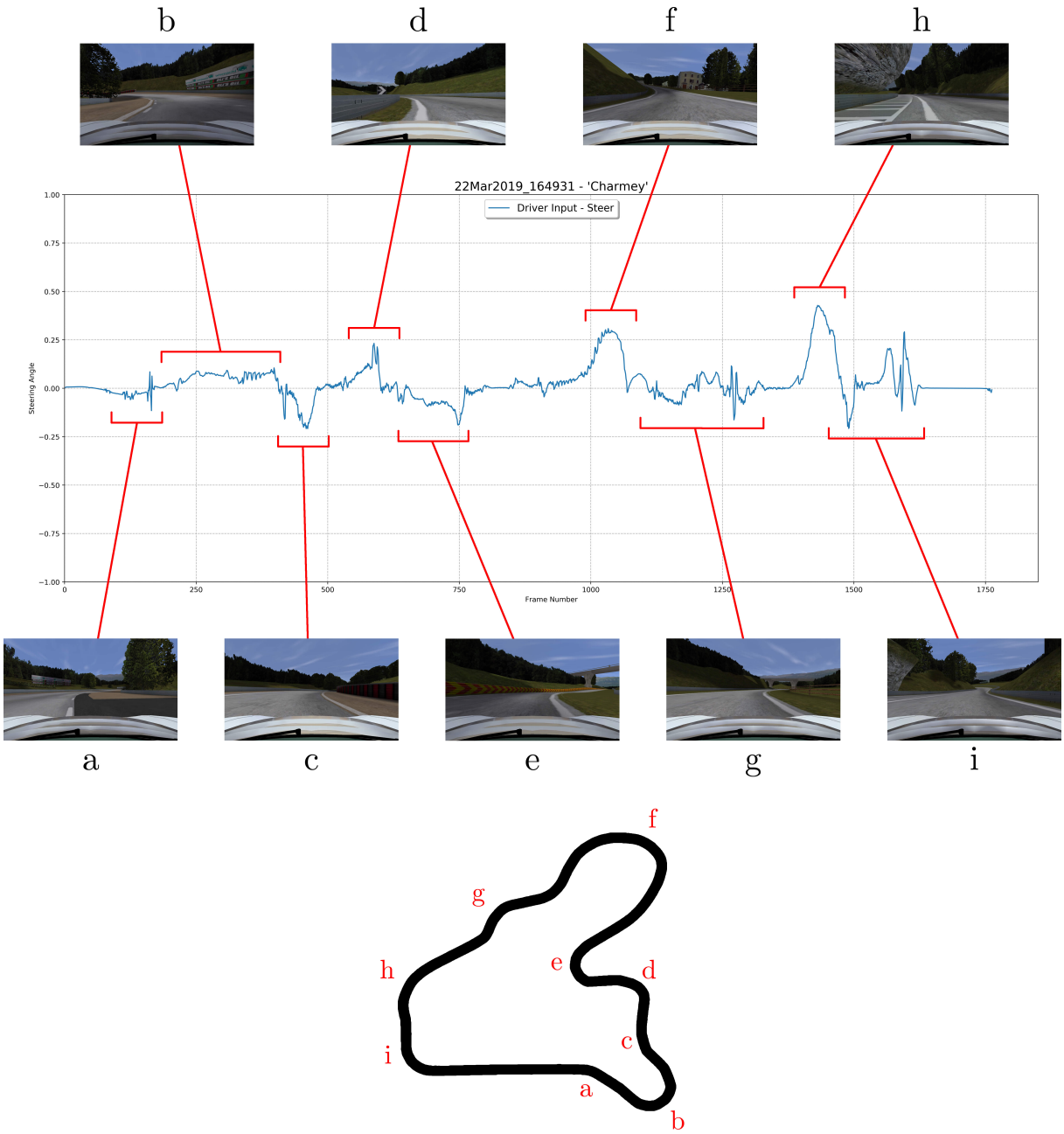


Figure 16: Annotated graph plotting an example dataset from the 'Charmey' road track.

Before commencing testing and experimentation between different combinations of tracks and environments, we first set out to verify the learning functionality of our prediction network. To test that the system is correctly learning to imitate the driver's steering around the track we simply run the model on data from a track that it has been trained on. Figure 17 provides qualitative verification that the system can learn to navigate around a previously encountered race track as the plot shows very little deviation between the actual and predicted steering angles, however it should be noted that this test only verifies the correctness of the learning algorithm and the relationship learned between the data may not necessarily be generalisable to other tracks.

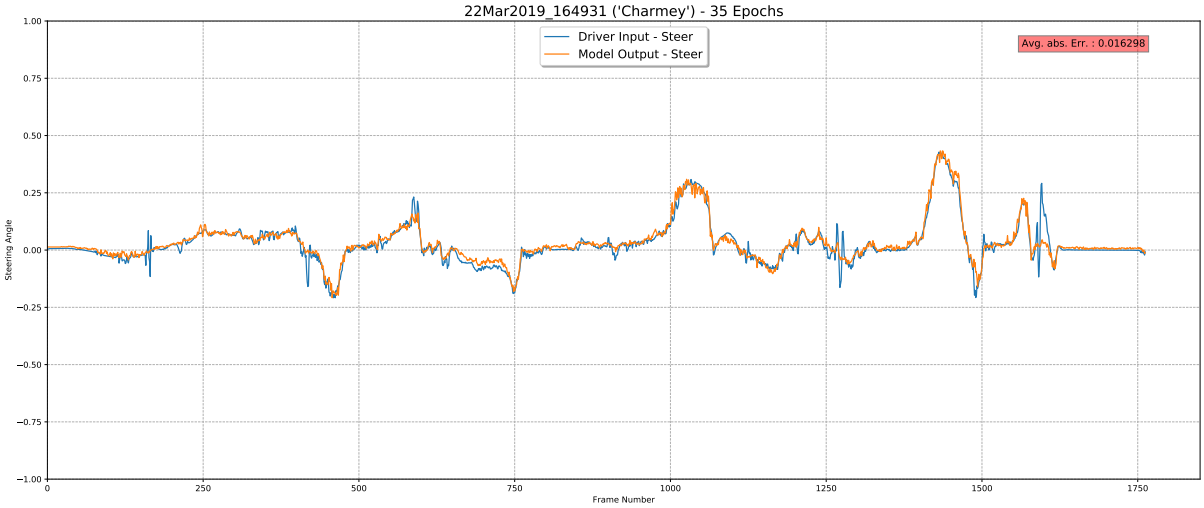


Figure 17: Result of testing the prediction model on a previously encountered track.

To begin testing the performance of our model in new environments we start by testing the prediction model on a track that is simpler than the complex road tracks, both in terms of geometrical complexity and environmental noise (e.g. tracks with forests, bridges, etc.). Figure 18 shows the results of testing the model on 'E-track 5' (previously unseen), a flat track with 6 simple shallow corners and relatively few visual distractions. An average absolute error of 0.024 is low, however this could be attributed to relatively low steering angle required to navigate the long corners, the qualitative results from the graph are a more robust indicator of performance.

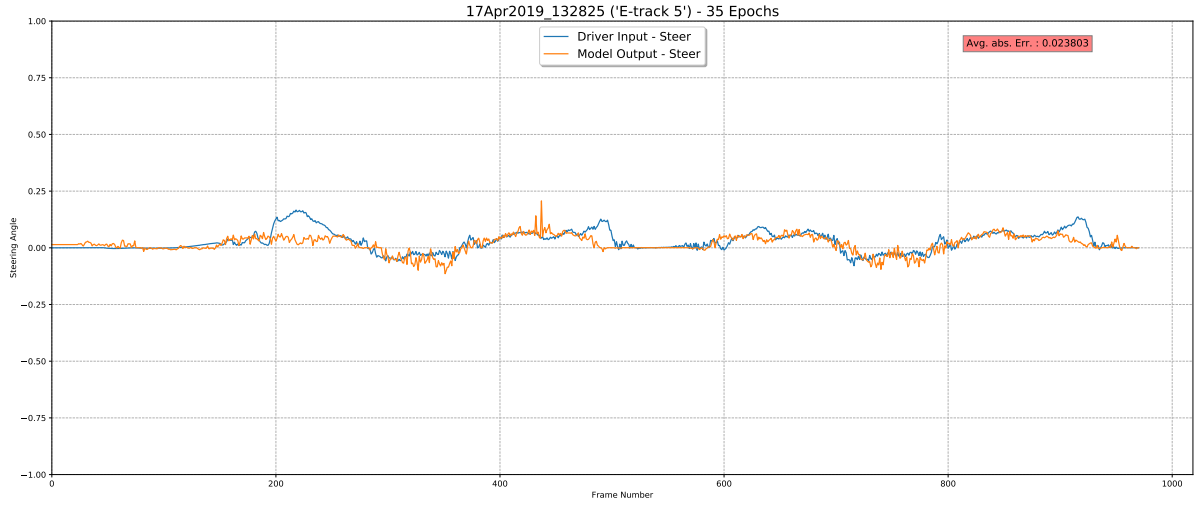


Figure 18: Result of testing the prediction model on a previously unencountered track.

It can be seen from the plot that there is an overall strong correlation between the actual and predicted values. With the exception of the first, all the corners of the track are responded to with an appropriate change in steering angle to match the curves of the track. The system appears to have learned particularly well that the straight sections of the track require no/very little steering input to be navigated as these sections are very closely matched by the model. A possible reason that the first corner elicits only a small response from the system could be that TORCS places the car on the left side of the track to start, so the corner is approached from the inside, whereas the robot driver usually approaches corners from the outside (i.e. the racing line). This uncommon approach to the corner may have 'confused' the model as the curve in the track is viewed from an alternative angle that the model isn't usually exposed to. Overall however, these results demonstrate an important validation that the model can predict

the appropriate steering responses for a previously unseen (albeit simple) track. From here the generalisability of the model can be tested further by analysing its performance on the full set of more complex tracks, as covered in the next section.

5.2 Final Results

Following the initial testing, the model can be tested on the full set of TORCS tracks that are increasingly complex and more comparable with environments that may be encountered by real-world autonomous driving systems - this will yield an insight into the overall generalisability of the prediction model. For each of the 25 tracks on which the system is tested, the model is trained on the remaining 24 and the average absolute error is averaged over 3 runs to accommodate for the non-deterministic nature of neural networks. An epoch training value of 35 for each track was found to be an optimal balance between under and over-fitting of the track data. Both the quantitative (average absolute error) and qualitative (likeness to the demonstrator) results are important indicators of the performance of the system and will be covered in this section.

Track	**Average Absolute Error**
A-speedway	0.016
Arizona Mile Speedway	0.023
E-track 5 (Speedway)	0.026
Chemisay	0.028
Olethros	0.031
Forza	0.031
Karwada	0.036
Corkscrew	0.038
Espie	0.039
Pinabashi Park	0.039
E-track 4	0.039
E-track 3	0.042
Wheel 1	0.044
E-track 2	0.044
Autodromo Lombaro	0.048
Ruudskogen	0.049
Brondehach	0.050
Street 1	0.051
Allondaz	0.052
Goldstone-Sand	0.057
Charmey	0.057
Migrants	0.060
Prenzlauer Dreieck	0.060
Inuksuk	0.068
Aalborg	0.078

Table 5: Performance of the system on the 26 available TORCS racing tracks.

Table 5 presents the numerical results from testing the system across the different tracks, showing mixed performance across the various environments. The low error values for the simpler tracks (i.e. tracks with few corners and visual distractions, such as speedways) shows that the system can closely predict the correct steering responses that are needed to navigate

the simpler tracks. For example, the highest performing track 'A-speedway' consists of two long corners and two long straights, this appears to suit the prediction model well as the steering angle is accurately matched to that of the demonstrator on all three runs (See Figure 19). The similarity between the predicted values for the three runs shows that the system is consistently learning the correct features that correspond to the appropriate steering angles. Furthermore, the smooth changes in the predicted steering angle as the corner is traversed indicates that the model is stable and isn't outputting a 'noisy' signal with anomalous peaks and troughs, this is especially important as rapid changes in steering input often lead to a loss of control of the vehicle.

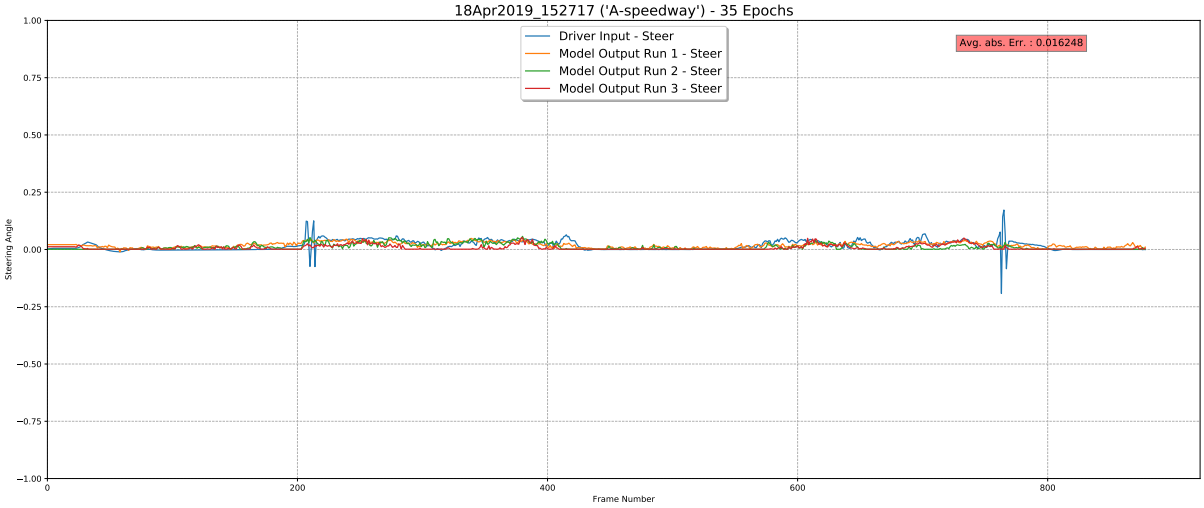


Figure 19: Performance of the prediction model when tested on 'A-speedway', a simple oval speedway.

Tracks that are more environmentally complex such as those that have large elevation changes and/or large geometric features in the surrounding environment (e.g. cliffs, bridges, etc.) appear to present difficulties for the system. Figure 20 shows the prediction plot for 'Inuksuk', a snowy winter mountain track, which is one of the worst performing tracks. It is observable that the prediction model struggles to 'follow' the changes in steering angle across the track and does not exhibit the large steering responses required to navigate the numerous sharp corners of the track. The environment of this race track is very different to those that the system is trained on and thus the model may not be able to identify and ignore the features of the landscape as environmental noise which are not pertinent to describing the geometry of the road. Furthermore, the elevation changes found in these mountain tracks (also 'Allondaz' and 'Charmey') could be an indicator that the system is susceptible to changes in the plane of the road and surrounding environment as it moves latitudinally around the image.

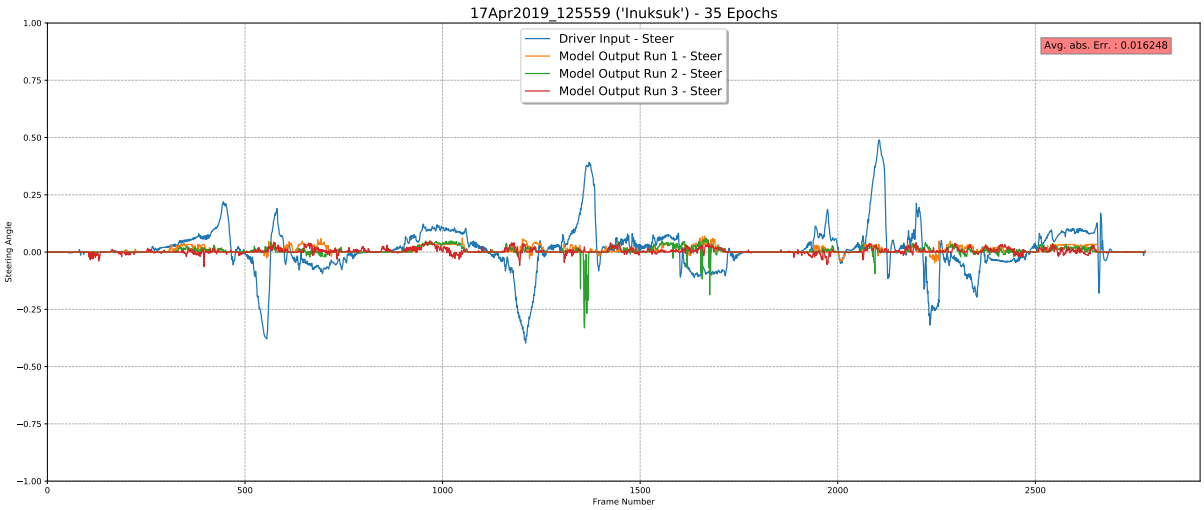


Figure 20: Performance of the prediction model when tested on 'Inuksuk', a snowy mountain track in Canada.

A common pattern that emerges from the plots, such as Figure 20, is the general flatness of the predicted steering values, especially on tracks with sharp corners. It has been shown that the system is capable of following very long, shallow corners, such as those present on the speedway tracks (see Figure 18 and 19), however the model is mostly reluctant to predict large steering responses that are required to navigate the shorter, sharp corners present on most of the tracks TORCS provides. This behaviour could be attributed to how the system can perceive the upcoming changes in track geometry, sharp corners are less visible and thus harder to detect in an image as the track quickly turns out of view, whereas long shallow corners take up a larger portion of the image and are perhaps more likely to be detected by the feature extractor. More discussion on the performance of the model is provided in Section 6.

6 Final Product Evaluation

The results from testing the system presented in Section 5.2 demonstrate equivocal attainment of the goals for the product stated in Section 2. Firstly, irrespective of prediction accuracy, the model was able to process and predict a steering value from track images at an average rate of 47.8 images per second, which is extremely close to the rate at which they can be collected from the simulator (50fps), thus demonstrating the real-time performance of the system.

The system has shown an ability to correctly predict the appropriate steering values for the 'simple' tracks in the dataset, such as the speedways, where the gradual changes in steering angle are closely matched by the system as it perceives the corner in a similar manner to the demonstrator. However, when exposed to more complex tracks with difficult combinations of corners, the system struggled to predict the correct steering values and exhibited generally poor performance. We hypothesise that one (or a combination) of the following factors can be attributed to the deficient performance of the model:

VGG16 Feature Vectors

The prediction network relies on the feature vectors that it receives as an input to accurately describe the geometry of the image, in particular the direction of the track, so that it can associate the appropriate features with the corresponding steering action. The features extracted by the VGG16 network (which is trained for image classification) may not be

describing the scene sufficiently for the prediction network to be able to identify the track and its boundaries from the other visual features in the image. In order for the prediction network to successfully learn which features in the image pertain to required changes in steering angle, it must be able to extract the location and direction of the race track and interpret the surrounding environment as noise. The way that VGG16 represents an image (through the feature vectors) may not be consistent enough to identify the changing structure of the road amongst the also-changing environment.

Another possible limitation of the VGG16 feature vectors may be that they are too specific in their representation of the image and do not describe the features of the image in a sufficiently general manner. This would present difficulties for the prediction network as it attempts to learn the similarities between the images and the corresponding steering inputs. Different track environments, road surfaces and side-walls may result in completely different representations of the image that hinder the ability of the neural network to generalise between the different track environments.

Complexity of NN Prediction Model

The prediction network is tasked with learning the relationship between the VGG16 feature vectors and the supplied vehicle steering input. Whilst the 3-layer neural network architecture is very time-efficient and provided satisfactory results during development, it could be postulated that the network depth should be significantly larger as the aforementioned relationship may be more complex than we anticipated. A deeper network would have a negative impact on the real-time performance of the model; however, it could allow the system to learn a broader set of more complex relationships that are perhaps required to navigate the more elaborate tracks.

7 Project Critical assessment

The primary goal of the project as defined in Section 2 was to "*create a system that can learn to drive a car around a racing track in a simulator using visual information only*", this has been partially achieved. The developed system can learn to predict the correct (within a small error margin) steering responses required to navigate simple race tracks using only the visual information extracted from the TORCS simulator, however the system shows poor performance on the more complex race tracks. Furthermore, integration of the prediction model back into the TORCS simulator to perform 'online' testing was not achieved.

The performance of contemporary image processing techniques has been investigated and assessed as an outcome of the development during the project, traditional feature descriptors such as HOG have been compared to modern, more complex CNN feature extractors such as VGG16. Using a pre-trained image classification network significantly eased the development process and omitted the need to develop a bespoke image processing CNN, and thus permitted more allocation of time and resources to other sections of the project, such as data collection. However, as discussed in Section 6, the aptness of an image classification network with pre-trained weights for characterising the geometry of race track images could be questioned as the output feature vectors resulted in poor performance when generalising to environments with visual distractions and complex road topology. A possible compromise may have been to allocate time to fine-tuning the VGG16 network alongside the prediction network so that the extracted features were more relevant to the given task as this process can help to boost the generalisability of the transferred features [33].

A technical aspect of the project which was not achieved is the functionality to input vehicle controls back into the TORCS simulator. It was determined that the time and resources required to integrate the prediction system with the simulator program would be too costly and would not outweigh the benefits that online testing would return. The comparison of predicted and actual steering values required to navigate the tracks was thought to be a robust enough indicator of system performance to justify the compromise. A negative result of the lack of online testing, however, is that the lap times from the predicted vehicle controls cannot be measured and compared against the racing robots and/or human demonstrators. Whilst the process of online testing would have been very time-consuming, the lack of an additional empirical measurement increases the dependency on qualitative measures of performance (i.e. visually comparing the plot of actual vs predicted steering values) which are, whilst robust, more difficult to compare.

A key requirement for the project was to develop a system that can achieve real-time performance whilst running on standard hardware (i.e. a modern desktop PC), to demonstrate the viability of low-cost vision-based autonomous driving solutions. The complete system presented in this report is capable of processing high-resolution images and predicting the corresponding steering values at a rate of 47.8 frames per second (running on a NVIDIA GTX 1080Ti GPU with 11GB VRAM), which well exceeds the rough minimum requirement of 20 fps. This fast data processing rate means that the system can scale well to tracks with high-speed sections where slower systems may not be able to react quick enough to upcoming changes in the track shape such as obstacles or corners. Furthermore, a prediction interval of 47.8 fps leaves ample opportunity for the extension and/or expansion of the system with a more complex learning procedure so that it can perform well on more advanced tracks and still maintain real-time performance.

On reflection, a development stage of this project which lacked efficiency and transpired to be disproportionately time-consuming was the iterative design process and tweaking of the prediction neural network. The structure of a neural network can have a profound impact on what and how the network learns [33], thus arriving on a final design of the prediction network was difficult as each modification to the model would often produce completely different results and was more reminiscent of a trial-and-error approach, rather than a convergent iterative process. Despite conducting repeatable experiments to test the effect of changes to the network, such as different activation functions, significant difficulty was encountered when attempting to isolate design choices which consistently improved the performance of the network, especially given the numerous interactions between the different parameters and the 'opaque' nature of neural networks, as previously mentioned [14].

8 Conclusion

The aim of this project has been to investigate the viability of a low-cost vision-based autonomous driving system using contemporary machine learning techniques as an alternative to traditional non-visual methods which rely on substantial amounts of expensive hardware. A system with real-time performance has been developed which analyses visual information from a forward-facing camera in a racing car simulator and uses the extracted information to predict a steering angle output for the vehicle.

A pre-trained image classification network (VGG16 [24]) has been used as a method to extract features from images captured in a simulator (TORCS [31]) which are processed by a simple 3-layer neural network to learn the relationship between the characteristics of the race track

images and the corresponding steering responses. We also demonstrate the use of a Saliency Sampler as a method to improve prediction accuracy by downsampling images in a task-relevant manner before feature extraction. The combined system demonstrated good performance on previously unseen tracks that are geometrically simple (i.e. with few corners and/or track undulations) where the predicted steering actions closely match those of a 'robot' demonstrator, however the machine learning model struggled to generalise to the complex tracks in the simulator that contained numerous visual distractions and elaborate road structures, and poorly predicted the appropriate steering responses that would be required to navigate these courses.

Given the 'black-box' nature of both the feature extraction and the prediction networks, it is difficult to deduce which parts of the system limit the overall prediction performance; whether the VGG16 feature vectors poorly describe the geometry of the scene or the prediction network does not effectively process the feature vectors (or a combination of the two) is uncertain, however some discussion on this topic is provided in Section 6.

Future work on this project and other research aiming to integrate CNN feature vectors and prediction NN's would benefit from visualising the activation patterns of the neurons in the feature extraction network, this would aid in understanding how the network responds to the input images e.g. a strong correlation between the activation patterns and the track ahead indicates good task-specific features. Furthermore, analysing changes in the visualised activation patterns as the network undergoes fine-tuning (a potential improvement discussed in Section 7) would be a robust qualitative measure of the impact that fine-tuning has on the network.

References

[1] A. A. Assidiq, O. O. Khalifa, M. R. Islam, and S. Khan. Real time lane detection for autonomous vehicles. In *International Conference on Computer and Communication Engineering*, pages 82–88, May 2008.

[2] W. Beelitz. The SIMPLy mIXed best practice TORCS robot. `http://www.simplix.wdbee-aorp.de/SIMPLIX/SimplixDefault.aspx`. Accessed: 22-03-2019.

[3] J. Borenstein, H. R. Everett, L. Feng, and D. Wehe. Mobile robot positioning: Sensors and techniques. *Journal of robotic systems*, 14(4):231–249, 1997.

[4] Y. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning*, pages 111–118, 2010.

[5] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[6] M. Buehler, K. Iagnemma, and S. Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer, 2009.

[7] J. Canny. A computational approach to edge detection. In *Readings in computer vision*, pages 184–203. Elsevier, 1987.

[8] K. H. F. Cardew. The Automatic Steering of Vehicles: An Experimental System Fitted to a DS 19 Citroen Car. *RRL report; LR 340*, 1970.

[9] F. Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. *CoRR*, abs/1610.02357, 2016.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition*, volume 1, pages 886–893. IEEE Computer Society, 2005.

[11] E. D. Dickmanns. The development of machine vision for road vehicles in the last decade. In *Intelligent Vehicle Symposium*, volume 1, pages 268–281. IEEE, 2002.

[12] F. Esposito and D. Malerba. Machine learning in computer vision. *Applied Artificial Intelligence*, 15(8):693–705, 2001.

[13] A. B. Hillel, R. Lerner, D. Levi, and G. Raz. Recent progress in road and lane detection: a survey. *Machine Vision and Applications*, 25(3):727–745, 2014.

[14] M. Liu et al. Towards better analysis of deep convolutional neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):91–100, 2017.

[15] D. Loiacono. Learning, evolution and adaptation in racing games. In *Proceedings of the 9th Conference on Computing Frontiers*, pages 277–284. ACM, 2012.

[16] E. Nadernejad, S. Sharifzadeh, and H. Hassanpour. Edge detection techniques: Evaluations and comparisons. *Applied Mathematical Sciences*, 2(31):1507–1520, 2008.

[17] F. Pedregosa et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[18] D. A. Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, pages 305–313. MIT Press, 1988.

[19] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[20] A. Recasens et al. Learning to Zoom: a Saliency-Based Sampling Layer for Neural Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 51–66, 2018.

[21] R Rudzits and N Pugeault. Efficient Learning of Pre-attentive Steering in a Driving School Framework. *KI - Künstliche Intelligenz*, 29(1):51 – 57, February 2015.

[22] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT Press, 2006.

[23] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[24] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*, 2015.

[25] C. Thorpe, M. Herbert, T. Kanade, and S. Shafer. Toward autonomous driving: the CMU Navlab. I. Perception. *IEEE Expert*, 6(4):31–42, Aug 1991.

[26] S. Thrun. A personal account of the development of Stanley, the robot that won the DARPA Grand Challenge. *AI Magazine*, 27(4):69, 2006.

[27] S. Thrun et al. Stanley: The robot that won the DARPA Grand Challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[28] C. Urmson et al. Autonomous driving in traffic: Boss and the urban challenge. *AI magazine*, 30(2):17, 2009.

[29] S. van der Walt et al. Scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.

[30] H. Winner and W. Wachenfeld. *Effects of Autonomous Driving on the Vehicle Concept*, pages 255–275. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[31] B. Wymann, C. Dimitrakakisy, A. Sumnery, and C. Guionneauz. TORCS: The open racing car simulator, 2015.

[32] J. Yang et al. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, volume 1, page 6, 2009.

[33] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.