Imperial College London

Department of Computing

Improving Autonomous Driving Agents using Bio-Inspired Visual Attention

Author:
Alexander Makrigiorgos

Supervisor:
Aldo Faisal

Submitted in partial fulfillment of the requirements for the MSc degree in MSc Computing (Artificial Intelligence) of Imperial College London

September 2019

Abstract

Humans' eye movements contain a wealth of information about their intentions and decision-making processes when performing tasks, rapidly fixating on objects of interest while tuning out unnecessary details. This ability to identify task-specific high-interest regions within an image could be very beneficial to machine learning agents attempting to learn visuomotor tasks which require making sense of a complex visual environment. In this research project, we investigate the effect of combining a visual attention model, trained using human eye movement data, with end-to-end autonomous driving systems. Autonomous driving is a challenging multi-task problem which has grown in popularity in recent years, and requires a deep understanding of the visual environment in which it takes place. To create a system capable of imitating human gaze patterns, we collect 3 hours of eye movement data from human drivers in a virtual reality environment. This data is used to train several deep neural networks to attempt to predict where humans are most likely to look when driving. Our most successful model shows a clear ability to identify key objects in driving sequences and mimic human attention dynamics. We then use the outputs of this trained network to selectively mask driving images using a variety of masking techniques. Finally, autonomous driving agents are trained using these masked images as input. Upon comparison, we found that a dual-branch architecture which processes both raw and attention-masked images substantially outperforms all other models in terms of average prediction error, validating our hypothesis that a visual attention model learned from human data can bolster the performance of machine learning agents in complex settings.

This work builds on an Independent Study Option carried out under Dr. Aldo Faisal's supervision in the winter term of 2019 [21]. The ISO involved a comprehensive review of literature in the fields of autonomous driving and human gaze prediction, portions of which are summarized with attribution in the Background chapter of this report. All data collection experiments, novel architecture designs and implementations presented in this report are entirely independent of the work done in the ISO. Appendix B summarizes the overlaps between the content of the ISO and this report.

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Aldo Faisal, for his guidance and support throughout this project. His feedback and insights were invaluable, and motivated me to challenge myself and push the limits of what I could accomplish.

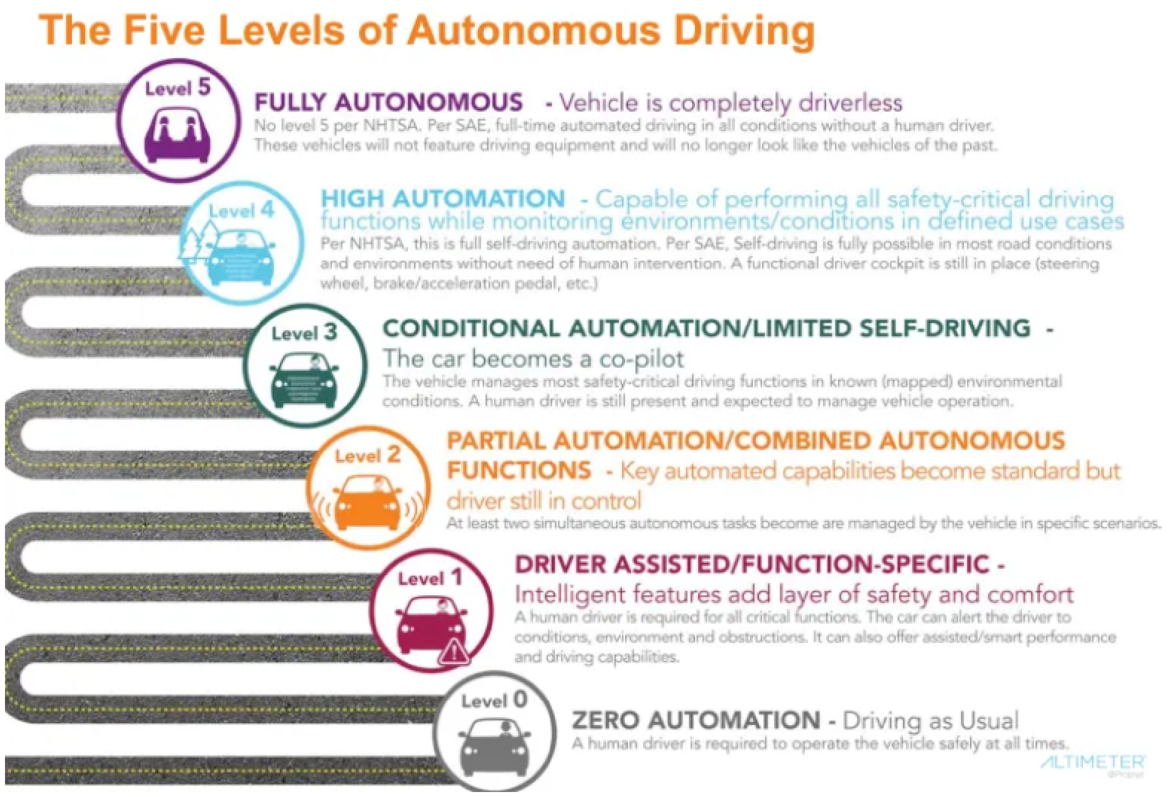
Contents

Chapter 1

Introduction

Autonomous driving is a rapidly expanding field of research with the potential to provide enormous social, economic and environmental benefits to society. Dozens of major car manufacturers and tech companies have begun to invest in the driverless car industry, with new competitors emerging daily and initial testing on public roads already taking place. There are six commonly defined levels of autonomous driving. Level 0 indicates a fully manual system, operated entirely by a human driver. Level 1 involves computer assistance in simple driving scenarios such as cruise control or anti-lock braking. At level 2, the computer is able to partially automate one feature at a time in certain circumstances, such as automatic emergency braking. Level 3 allows the computer to simultaneously control multiple driving functions, with a human driver ready to take over in case anything goes wrong. This is the current state of publicly available autonomous technology, including features such as simultaneous lane keeping and cruise control. Level 4 lets the computer automate all driving functions, but still requires a human driver to be present in the rare case that they are required to take over, while level 5 removes the need for a human altogether and can operate entirely independently. Currently, the leaders in autonomous technology have achieved level 4 autonomy in specific, well-defined environments (college campuses, geo-fenced city blocks, etc.). At present, all of these companies use a ”modular pipeline” approach to carry out the driving task, which involves combining multiple subsystems such as sensory perception, path planning and vehicle control to form a full driving system.

While modular pipeline systems are very effective at driving in well-controlled environments, they are also somewhat restricted by certain requirements, such as their reliance on hand-coded road rules and high-definition environmental maps that must be frequently updated. Because of these restrictions, there is almost no possibility for modular pipeline systems to ever progress to level 5 autonomy. In recent years, end-to-end autonomous driving systems have attracted increasing amounts of interest as a potential solution to this scalability issue, with the latest breakthroughs in deep learning having substantially improved the capabilities of these systems. End-to-end autonomous systems attempt to directly learn the driving task by train-


The Five Levels of Autonomous Driving

Level 5 FULLY AUTONOMOUS - Vehicle is completely driverless
No level 5 per NHTSA. Per SAE, full-time automated driving in all conditions without a human driver. These vehicles will not feature driving equipment and will no longer look like the vehicles of the past.

Level 4 HIGH AUTOMATION - Capable of performing all safety-critical driving functions while monitoring environments/conditions in defined use cases
Per NHTSA, this is full self-driving automation. Per SAE, Self-driving is fully possible in most road conditions and environments without need of human intervention. A functional driver cockpit is still in place (steering wheel, brake/acceleration pedal, etc.)

Level 3 CONDITIONAL AUTOMATION/LIMITED SELF-DRIVING - The car becomes a co-pilot
The vehicle manages most safety-critical driving functions in known (mapped) environmental conditions. A human driver is still present and expected to manage vehicle operation.

Level 2 PARTIAL AUTOMATION/COMBINED AUTONOMOUS FUNCTIONS - Key automated capabilities become standard but driver still in control
At least two simultaneous autonomous tasks become are managed by the vehicle in specific scenarios.

Level 1 DRIVER ASSISTED/FUNCTION-SPECIFIC - Intelligent features add layer of safety and comfort
A human driver is required for all critical functions. The car can alert the driver to conditions, environment and obstructions. It can also offer assisted/smart performance and driving capabilities.

Level 0 ZERO AUTOMATION - Driving as Usual
A human driver is required to operate the vehicle safely at all times.

Figure 1.1: Six Levels of Autonomous Driving [1]

ing deep neural networks on sensory input, with the theoretical ability to drive in previously unseen environments and to continuously learn from experience and optimize their behaviour. In practice, however, these networks can be unpredictable or difficult to interpret, require massive amounts of training data to perform well, and may not be prepared to deal with rarely encountered edge cases on the open road. In this project, we aim to address some of these issues by taking inspiration from the way in which humans approach the driving task.

Humans have a very finely tuned sensory system which allows them to quickly locate objects of visual interest within a scene, ignoring irrelevant details and only processing what is important. Eye movements encode a wealth of information about humans' intentions, understanding of their environment and decision-making processes. The ability to understand and recreate this attention-focusing mechanism has the potential to provide similar benefits to machine learning systems which have to parse complex visual scenes in order to complete their tasks, reducing computational burden and improving performance by quickly identifying key features in images. In this project, we attempt to inject the expertise of human drivers into an end-to-end autonomous driving system by incorporating an attention model which has learned to predict human gaze patterns in the driving context. By directing the focus of the driving network to specific image locations which contain important information for driving, we aim to improve both the training speed and overall performance of these networks.

The main contributions of this project are as follows. First, we collect and manually label a high-quality dataset of driving sequences annotated with the eye movements of the driver in a virtual reality simulator. Second, we implement and compare the performance of several recently published gaze prediction neural networks on our recorded dataset, and make some novel additions to the best-performing network in order to further boost performance. Finally, we propose and test multiple methods of incorporating the attention maps produced by the gaze prediction networks into autonomous driving systems, and compare the overall performance of systems with and without attention.

Using the recorded eye movement data from our driving experiments, we show that we have produced a gaze prediction network which demonstrates a clear understanding of the driving task, with the ability to identify key objects and features in driving images as a human would. A demonstration of a driving sequence blended with our model's attention map predictions can be found at `https://www.youtube.com/watch?v=A2cy70a5xtg`. Furthermore, we show that a dual-branch autonomous driving architecture which learns from both raw and attention-masked driving images provides a noticeable increase in performance compared to a standard architecture trained only on raw images, predicting driving actions with a 25.5% average reduction in error. Finally, we make the interesting observation that an autonomous agent trained on images which contain *only* the image regions selected by the gaze prediction network, with the rest of the image blacked out, sees little to no performance decrease with respect to an agent trained on full images.

The remainder of this report is structured in the following way: Chapter 2 reviews the key papers and technological advancements which have motivated our experiments and choice of deep learning architectures in both gaze prediction and autonomous driving. We also include an overview of the design, functionality and core features of the Car Learning to Act (CARLA) driving simulator, which we use in all of our experiments. Chapter 3 describes the collection, preprocessing and preliminary analysis of the two datasets used in this experiment: one for training a gaze prediction network, the other for training autonomous driving agents. In Chapter 4, we provide detailed descriptions for all of the deep learning models developed and evaluated in the course of this project. For each model, we provide an exact description of the model's architecture, training process, and the intuition behind its design. Chapter 5 compares the performance of all implemented models for both tasks. In addition, we individually evaluate and attempt to understand each model's observed behaviour, discussing the implications and conclusions that can be drawn from our results. Finally in Chapter 6, we conclude by reflecting on the project's goals and the results which arose from our experiments, as well as considering improvements and promising directions for future work.

Chapter 2

Background

In this chapter, we provide an overview of the existing research and literature which inspired this project. We begin by looking at the recent advancements which have made end-to-end autonomous driving a potentially viable alternative to modular pipelines. Afterwards, we introduce the topic of human visual attention and review some of the models which have been most successful at predicting human gaze in different scenarios. Finally, we describe the design and functionality of the CARLA driving simulator, used to conduct our gaze collection experiments and to train and evaluate our autonomous driving agents in this project.

2.1 End-to-End Autonomous Driving

The ability to steer a vehicle using commands generated by a neural network was first demonstrated by David Pomerleau in 1989, with his Autonomous Land Vehicle in a Neural Network (ALVINN) experiment[24]/[21], in which a retrofitted army ambulance successfully drove along public roads at low speeds using a three-layer neural network to convert camera images into steering commands. This process of directly learning driving commands from sensory input is referred to as end-to-end autonomous driving, as distinguished from the modular pipeline approach which deconstructs the driving task into sub-modules such as perception, path planning and vehicle control. In this project we focus on the end-to-end model, which has attracted a large amount of interest in recent years with the ever-growing popularity of convolutional neural networks for image processing and the continuously increasing capabilities of modern GPUs for training deep networks.

Within the end-to-end paradigm, there are two main approaches: (1) Reinforcement Learning, in which an agent receives a reward signal based on its driving behaviour and learns to optimize its actions to maximize this reward signal, and (2) Imitation Learning, in which an agent uses expert data (such as the actions of a human driver) to learn a driving policy by mimicking the actions of the expert. Thus far, attempts at training a self-driving system purely through reinforcement learning have not been successful[25][36][8][21], largely due to the complex nature of the task requiring tens of millions of sample episodes to begin learning a stable policy. In addition, these systems cannot be effectively trained in the real world as the trial-and-error nature of the training process would require constant human intervention to avoid collisions. By contrast, imitation learners can learn an effective policy using just a few hours of recorded expert data[8][6], and can be safely trained off-line such that only fully trained vehicles would actually be allowed to operate in the real world. As such, the bulk of end-to-end autonomous driving research in recent years has been focused on exploring and improving on imitation learning approaches. The first research team to successfully drive a CNN-guided imitation learning agent, nicknamed DAVE-2, on public highways and spark a renewed interest in the field of end-to-end driving was that of Mariusz Bojarski in 2016[3]/[21].

2.1.1 Lane Following with DAVE-2

The original DAVE (DARPA Autonomous Vehicle)[20] project took place in 2004, in which a shallow convolutional network was trained to drive a small robot through alleyways while avoiding obstacles scattered throughout the area. While this experiment demonstrated the potential of CNNs for steering a vehicle, the mean distance between crashes was approximately twenty meters, clearly insufficient for testing on public roads. Over a decade later, NVIDIA introduced DAVE-2[3], a system which leveraged the vastly increased power of GPUs to train a deeper, more powerful network and successfully drive on public highways with little to no human intervention required.

To train the imitation learning network, 72 hours of human driving data was collected using vehicles with three cameras mounted behind the windshield and a Controller Area Network (CAN) bus recording the angle of the steering wheel at all times. Each camera frame is thus associated with a steering angle which is used as the command to emulate when training in a supervised manner. To establish diversity in the dataset, data was recorded in a variety of weather conditions (sun, fog, rain, snow), road types (highways, cities, rural unpaved roads) and different times of day. As the desired behaviour of the vehicle was simple lane following, any frames including other actions such as turning or waiting at traffic lights were removed. In addition, to avoid bias in the dataset which could affect the vehicle's behaviour, the proportion of images showing straight and curved roads was manually balanced.

The block diagram for the training process of the DAVE-2 network is shown in Figure 2.1. As seen, data augmentation was performed to increase the stability of the produced driving policy. This addresses a well-known issue in imitation learning systems, namely that since they are trained on data with no examples of driving errors, they have difficulty recovering from their own mistakes in a testing environment as they have not observed any data showing them how to recover. Specific image shifts and rotations were performed using images from all three cameras to approximate a viewpoint transformation which makes the car appear to be drifting from the center of the lane or badly angled with respect to the road markings. These augmented images were then each manually labeled with a steering angle which would restore the vehicle to the desired position. After augmentation, the desired steering command is compared to the angle predicted by the CNN, and the mean-squared error between the two is calculated and used to train the network for subsequent predictions.

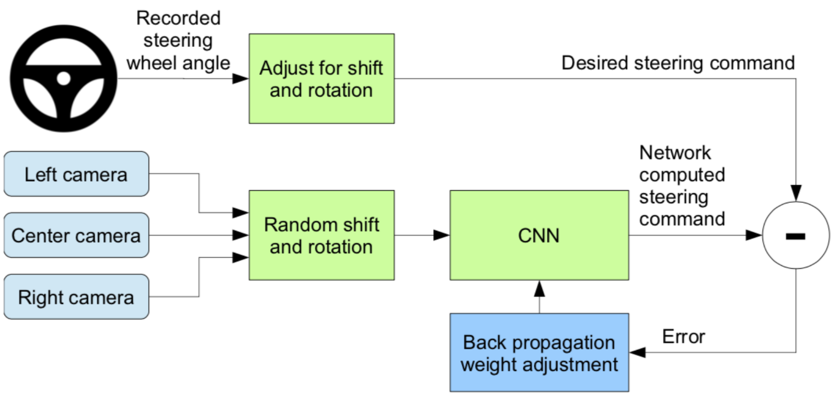

Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
-
Back propagation weight adjustment
Error

Figure 2.1: DAVE-2 Training Block Diagram (Taken from [3]/[21])

The network architecture of DAVE-2 consists of five convolutional layers, each followed by a max-pooling layer, and ending with three fully-connected layers which output a single scalar representing the desired steering angle. Using only this relatively simple architecture, DAVE-2 achieved major improvements over any previous end-to-end systems in a real-world lane following task. The system was tested on New Jersey highways in various weather conditions, and was able to lane follow without human intervention approximately 98% of the time. The success of this research paved the way for many subsequent efforts which have attempted to address the significantly more challenging task of fully autonomous point-to-point navigation.

2.1.2 Navigation with Conditional Imitation Learning

While successful at simple lane following, Bojarski's imitation learning network contains some major limitations which prevent it from scaling to the task of full autonomous navigation. Chief among these is an inability to deal with any sort of intersection scenario, where the car must choose between multiple directions. The neural network has been trained to find a mathematical mapping between sensory inputs and control signal outputs, but in this case there is no mathematical function which can always describe the correct direction to take - it is dependent on the destination of the user. Pomerleau described the problematic behaviour for this scenario in his original ALVINN paper published in 1989: ”Currently upon reaching a fork, the network may output two widely discrepant travel directions, one for each choice. The result is often an oscillation in the dictated travel directions and hence inaccurate road following[24]".

In 2017, Codevilla et al. proposed a solution to this problem known as conditional imitation learning, hereafter referred to as CIL[6]/[21]. This framework introduces the concept of a high-level command as a new input to the imitation learning network, which tells the vehicle what actions it should take when navigating between destinations. These commands take the form of simple actions (e.g. take the next right, continue straight, follow the lane) which remove the requirement for the imitation learning network to perform any sort of high-level route planning and simply attend to the lower-level tasks of providing acceleration, braking and steering commands. These commands can then easily be provided to the network at test time by any sort of route planning or mapping application, e.g. Google Maps.

To implement this method, we must expand our dataset from a collection of observations and their output actions $D = \{(\mathbf{o}_i, \mathbf{a}_i)\}_{i=1}^N$ to a set of observations, commands and actions $D = \{(\mathbf{o}_i, \mathbf{c}_i, \mathbf{a}_i)\}_{i=1}^N$[6]/[21]. In Codevilla's CIL dataset, each observation $\mathbf{o}_i$ consists of an input image paired with the current speed of the vehicle, which is necessary to produce sensible acceleration or braking commands in the presence of dynamic objects. Each command $\mathbf{c}_i$ corresponds to one of a discrete set of high-level commands as discussed above, and the outputs $\mathbf{a}_i$ are the vehicle actions in terms of throttle, brake and steering control signals.

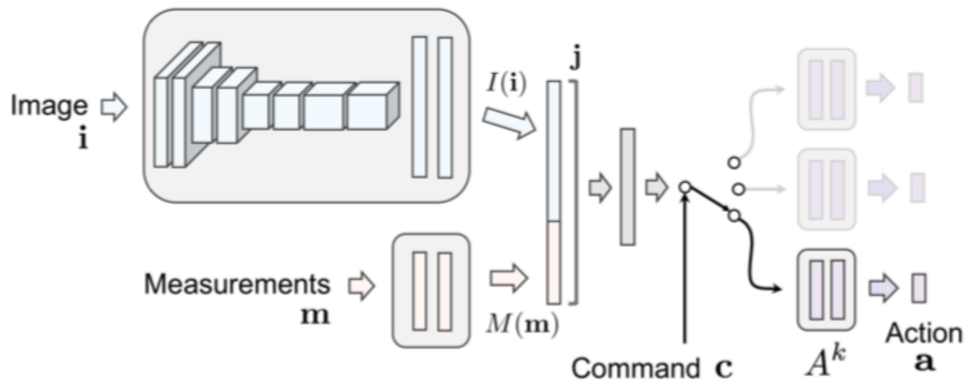


Figure 2.2: Conditional Imitation Learning Model Architecture (taken from [6]/[21])

The architecture used to incorporate these high-level commands into the training process is shown in Figure 2.2. Input images are processed by a 2D CNN consisting of eight stacked convolutional layers followed by two fully connected layers, while the speed input is processed by two fully connected layers, and the two resulting feature vectors are concatenated. This joint sensory vector is then passed to one of several possible branch heads, depending on the high-level command recorded for that frame. Each branch head is trained only on samples for the specific command it is associated with, producing highly specialized mappings from observation to action. The network is trained to reduce the multi-task loss for all predicted actions, for which more details can be found in Section 4.2.1.

2.1.3 Attention Models

Attention models have seen success in a variety of machine learning tasks[34][32][37] by emphasizing important features in input data, leading to improved performance as well as more transparent and explainable models. Thus far, however, there have been very few attempts to integrate attention mechanisms into end-to-end systems for autonomous driving. To our knowledge, only one method has been published for incorporating an attention mechanism into an imitation learning agent in the context of self-driving. The Multi-Task Learning from Demonstration (MT-LfD) framework, proposed by Mehta et al. in 2018, uses a set of handcrafted auxiliary predictions to focus the network's attention on features which were deemed by the authors to be important for the driving task[22]/[21]. These include visual affordances, such as calculating the distance to an upcoming intersection or an oncoming vehicle, and action primitives which decompose larger actions into a series of smaller sub-actions (e.g. "turn left at the upcoming intersection" becomes "slow down", "turn left", "speed up"). A total of 21 auxiliary predictions are defined and used to influence the training of the network.

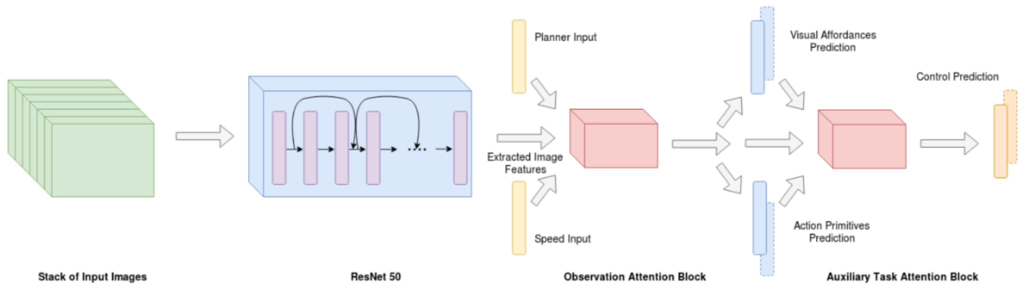

Planner Input
Visual Affordances Prediction
Control Prediction
Extracted Image Features
Speed Input
Action Primitives Prediction
Stack of Input Images
ResNet 50
Observation Attention Block
Auxiliary Task Attention Block

Figure 2.3: MT-LfD Network (taken from [22]/[21])

The architecture used for the MT-LfD framework is pictured in Figure 2.3. After initial processing of the input images, an intermediate prediction stage takes place, in which the network outputs its predictions for each of the 21 auxiliary tasks. These predictions are then reintroduced to the network and further processed by another convolutional block, which outputs the final predictions for actually controlling the vehicle. The training loss includes the loss from both the main and auxiliary predictions, forcing the network to learn all tasks simultaneously with the goal of highlighting the connections between the two. In practice, however, the model did not see any improvement in overall performance as a result of training with these auxiliary predictions. This is not necessarily a reflection on the usefulness of attention models for the driving task in general; we argue that the structure of this network is rather restrictive, as so many strictly defined tasks being included in the network's loss function may limit its ability to learn other important features present in the input data. In Section 2.2, we explore techniques for predicting attention maps from human eye movements, which have been shown to encode large amounts of task-relevant information and which we believe can provide a more natural way of incorporating attention into autonomous driving systems.

2.1.4 Evaluating Autonomous Agents

The ideal way of evaluating an autonomous driving agent's performance is, of course, to test it in the real world and observe its performance. In practice however, this is not a realistic option for most research groups due to logistics such as cost and safety concerns. Another option, then, is to evaluate these agents offline by computing performance metrics on a validation set of driving data. This is a very common practice in the evaluation of machine learning systems, but in an open-loop environment such as driving in the real world, where the agent is subject to the consequences of its actions, it is not necessarily guaranteed that a model with low offline prediction errors will actually perform well. In this section we review the findings of Codevilla et. al's study ”On Offline Evaluation of Autonomous Driving Systems” [5], which suggests the best metrics and validation set features to use to ensure that offline performance correlates well with online performance. In this study, 45 Conditional Imitation Learning[6] models were trained and compared in terms of online and offline performance evaluations. The models differed in three main respects. First, the number of convolutional layers in the feature extraction network is varied from 4 to 12 layers. Second, the amount and distribution of the training data used for different models is varied. Models are trained with 0.2 to 80 hours of recorded data, and this data is distributed in four different ways: central camera images with noise added to the control signals, central images without noise, images from three cameras with noise, and images from three cameras without noise. Third, training parameters such as dropout, regularization and data augmentation likelihood are varied. All models are trained using the weighted sum of mean-squared errors between the predicted and ground-truth control values (see Section 4.2.1 for more details). All models were then evaluated using a number of different offline metrics, and tested online in the CARLA simulator[8], measuring online performance in terms of navigation episode completion rate, number of kilometers driven per driving infraction, and average distance driven towards goal destinations.

The results of the study indicate that the correlation between offline metrics and online performance is generally quite weak - agents with very low prediction errors often still prove to be poor drivers. Many specific insights are made into different training conditions, custom-designed metrics and more - we refer the reader to the cited paper for further details on these findings. For this project, we make use of the two most important factors which were found to increase the correlation between offline and online performance. First, the study found that measuring offline performance using Mean Absolute Error rather than Mean Squared Error increases

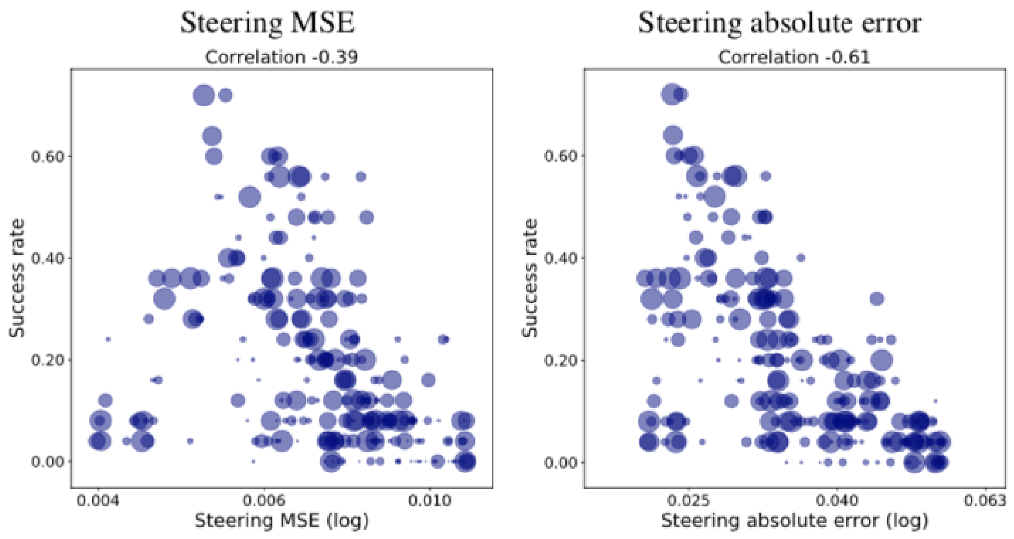


Figure 2.4: Scatter plots of navigation episode success rate vs. MSE and MAE for a large selection of autonomous driving models (taken from [5]).

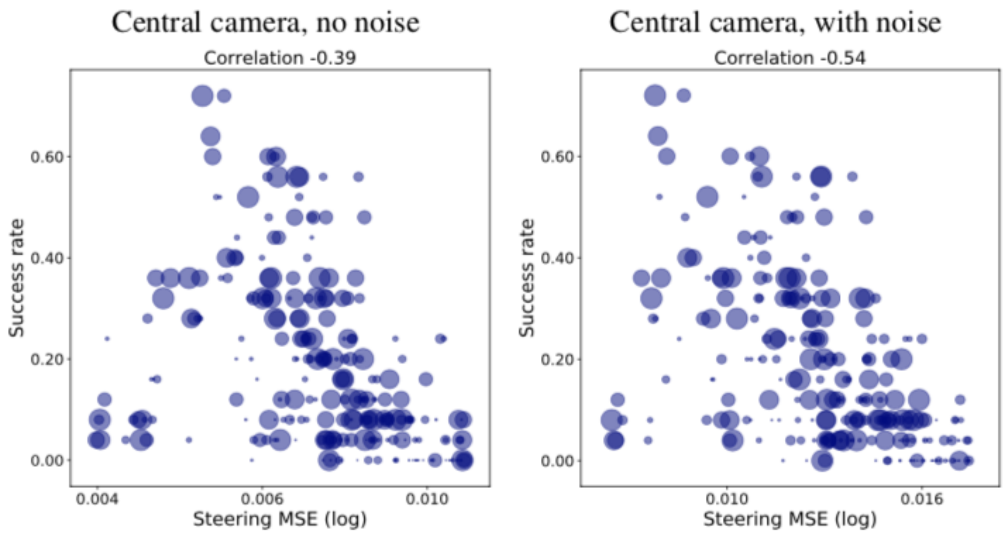


Figure 2.5: Scatter plots of navigation episode success rate vs. MSE for a large selection of autonomous driving models, comparing models trained with and without control signal noise injections (taken from [5]).

performance correlation by 56%, with MSE in particular shown to be an extremely inaccurate method of measuring performance. Second, the addition of noise to the dataset's control signals further increases the correlation by 38%. The results from the study indicating these improvements are shown in Figures 2.4 and 2.5.

2.2 Human Visual Attention

The link between human eye movements and the internal cognitive processes which they reflect is a highly active field of research. Alfred Yarbus first demonstrated this connection in 1967 with a series of experiments in which he recorded subjects' gaze patterns while viewing a painting [35]. By comparing the patterns produced by subjects who were freely viewing the painting against those who were given specific instructions about what to look for, he showed that different internal goals produced consistently different gaze patterns. An example of these results is shown in Figure 2.6, where a clear shift can be seen in the subjects' visual interrogation of the scene, particularly from focusing on faces to things such as clothing or objects in the room which may be indicative of wealth.

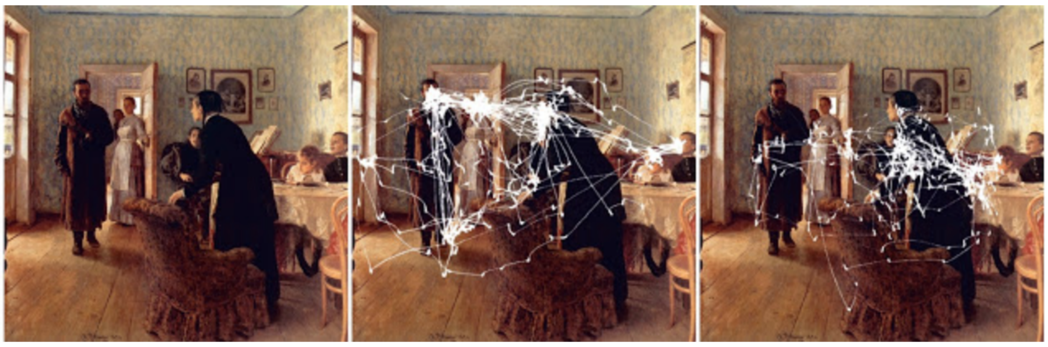


Figure 2.6: Example of differing gaze patterns for subjects viewing Repin's "An Unexpected Visitor". The leftmost image is the original painting, the middle is overlaid with fixations for subjects freely viewing the painting, and the right is overlaid with fixations for subjects who were asked to evaluate the material circumstances of the family in the painting [35].

For this project, we are interested in creating a system which can predict how a human would distribute their gaze in a given scene, in order to produce an attention map which could boost the performance of an autonomous driving agent. There exist two main approaches to predicting gaze patterns: *bottom-up* and *top-down* saliency prediction. In bottom-up saliency prediction, we assume that the subject has no agenda, and directs their gaze to whatever interests them in the scene they are viewing. In top-down saliency prediction, we assume that the subject has a specific goal or a task to complete, and is purposefully directing their gaze to acquire the information necessary for completing that task. As this project concerns gaze patterns produced while a subject is driving, we are mainly interested in predicting gaze in top-down settings, as we have a clear goal (reaching our destination without crashing). Nevertheless, we implement and evaluate both bottom-up and top-down gaze prediction methods in order to establish a comparison between the two and verify that top-down approaches indeed perform better in this setting.

2.2.1 Bottom-Up Gaze Prediction

In recent decades, computational models for predicting gaze in a bottom-up setting have focused on identifying features within an image which are likely to attract a human's attention. In recent years, however, these approaches have largely given way to data-driven approaches which make use of deep convolutional networks and widely available object recognition datasets to automatically identify important features for gaze prediction.

The first model to achieve high performance in the saliency prediction task using hierarchical feature learning was the Ensemble of Deep Networks (eDN) model of Vig et al. [33]/[21] shown in Figure 2.7, which used randomly generated multi-layer feature extractors to create high-quality feature vectors. Hundreds of feature extractors were created by randomly combining different smoothing, activation and filtering operations. These extractors were then each individually evaluated, and the most effective were combined into a single model which blended their predictions into a single feature vector, which was used to train a support vector machine to predict saliency maps. At the time of its publication in 2014, the eDN model outperformed all previous models on the well-known MIT300 saliency prediction benchmark.

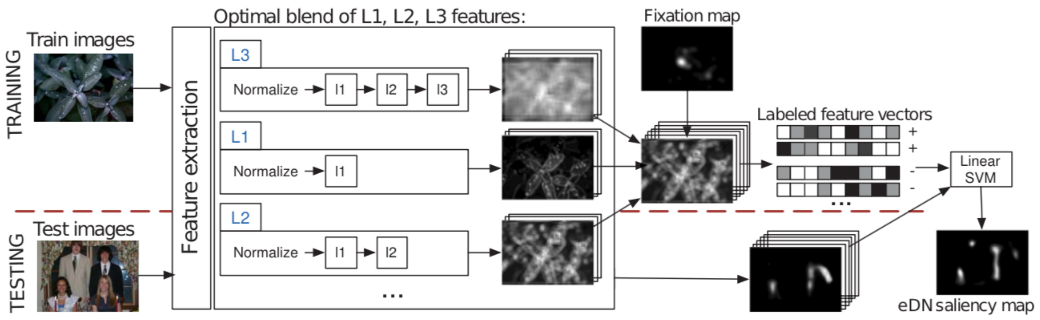


Figure 2.7: Ensemble of Deep Networks (taken from [33]/[21])

In the following years, another major step forward was made with the discovery that pre-trained object recognition networks such as Alexnet[16] and VGG[26] performed exceptionally well at identifying visually salient features. By using these networks as feature extractors, followed by fine-tuning for the saliency task, models such as SALICON[13], DeepFix[17], MLNet[7] and DeepGaze I and II [19][18] continued to see increased scores on the MIT300 and SALICON benchmarks. The MLNet model, which in 2016 achieved the highest results on SALICON, the largest public saliency prediction dataset, is shown in Figure 2.8.

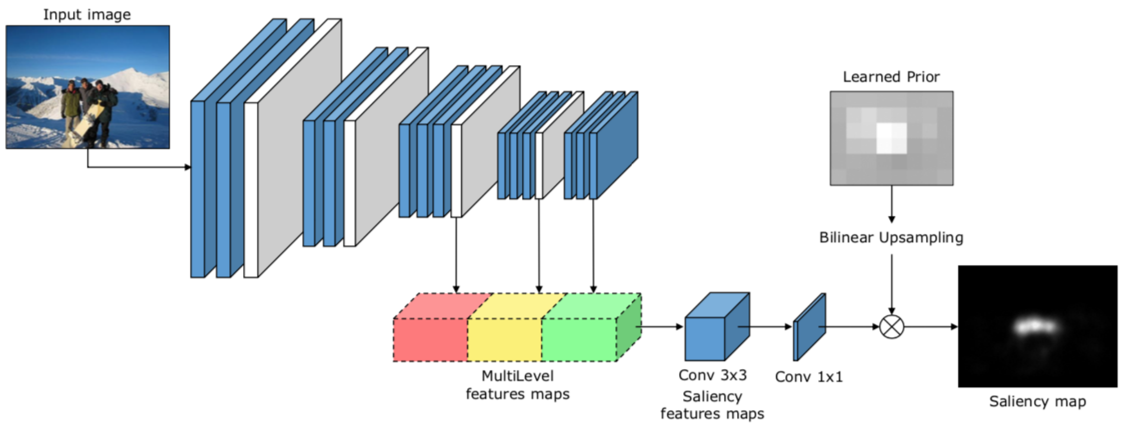


Figure 2.8: Multi-Level Net (MLNet) Architecture (taken from [7])

MLNet uses the VGG-16 network as a feature extractor for input images, with the fully-connected layers used for classification removed. All parameters are frozen to their state after being trained on the ImageNet object recognition database. Activated feature maps are then extracted from different layers of the network to obtain both high and low-level feature representations. These maps are then upsampled to the size of the input image, concatenated and processed by a shallow second network, the Encoding network. This network simply consists of a 3x3 convolutional layer followed by a 1x1 convolutional layer, and is trained using ground-truth saliency maps to produce saliency-specific feature maps from the VGG-16 activations. The final important step used by MLNet and many other networks is the incorporation of a prior. Humans exhibit certain biases when looking at images, in particular a strong tendency to focus on the center of an image rather than the periphery. To prevent the network from simply learning to reproduce this bias, a prior is learned alongside the main training objective and used to mask the predictions of the network in order to remove the bias from the model's final output. More details on the design of MLNet can be found in Section 4.1.2.

2.2.2 Top-Down Gaze Prediction

Far fewer attempts have been made to predict gaze in top-down settings than in bottom-up settings[21]. This is likely due to the inherently more difficult nature of the task. Bottom-up approaches are simply focused on identifying general image features that people find interesting and are likely to look at - as such, they benefit greatly from using pretrained networks for generalized object detection, since these networks generally capture the most semantically interesting high-level features of an image. Each top-down approach, on the other hand, must be tuned and trained specifically for the task it is being used for. Consider a simple example for two tasks: making a cup of tea versus making a sandwich. Both tasks would involve similar scenes and objects (kitchen, pantry, etc.), but the objects focused on by the user would be very different. As such, there is no way to design a 'general' top-down gaze predictor. Confounding variables also have to be considered in top-down settings; a human may be very likely to look at a brightly colored billboard advertisement every time they drive by it, despite its irrelevance to the actual task of safely driving a car. The exact degree to which bottom-up and top-down influences dictate how humans allocate their gaze remains an open question. In order to form an understanding the decision-making process behind the scheduling of eye movements in a top-down setting, we consider the Uncertainty-Reward model proposed by Sprague and Ballard in 2003[27]/[21].

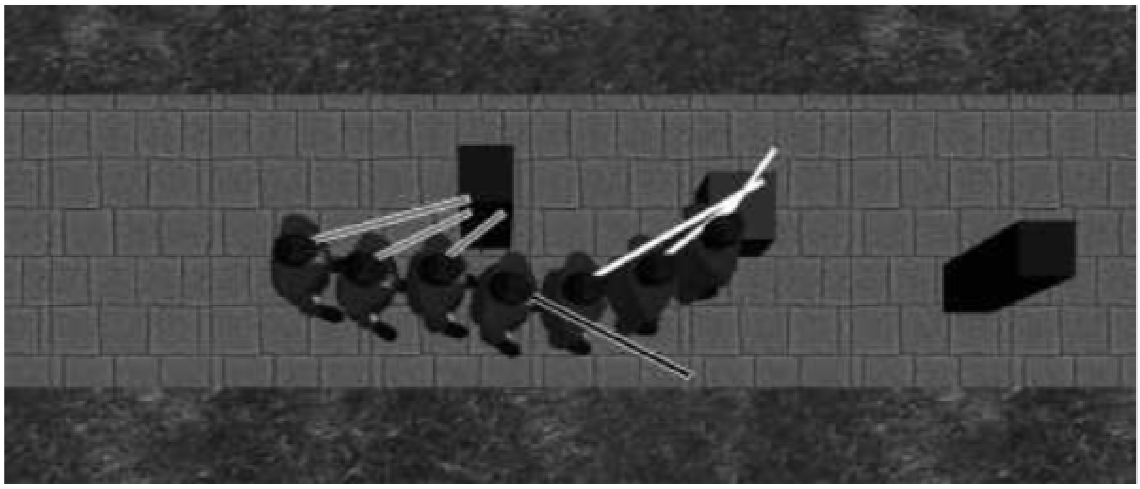


Figure 2.9: Sidewalk Navigation Agent's chosen state updates for seven consecutive time steps. Gray bars indicate obstacle avoidance, black indicate sidewalk following, and white indicate litter collection. The agent focuses on reducing uncertainty from variables which have potential to cause negative reward in the near future such as hitting the obstacle or leaving the sidewalk (taken from [27]).

The Uncertainty-Reward model hypothesizes that eye movements are directed towards certain objects in a scene in order to reduce uncertainty about the state of those objects, with the overall goal of maximizing some internal reward, i.e. completion of the given task. Within this framework, then, specific eye movements to different objects can each be assigned a value based on the amount of uncertainty that is resolved by focusing on that object, weighted by the object's relevance to completing the objective. Sprague & Ballard initially tested this hypothesis with a simple experiment, in which a virtual reinforcement learning agent was trained to simultaneously perform three tasks: sidewalk following, litter collection, and obstacle avoidance. A reward function was designed to give positive rewards for remaining on the sidewalk and collecting litter, and negative rewards for colliding with obstacles. To model human gaze scheduling and introduce uncertainty into the experiment, the model was constrained to only update one variable from the state space every 300ms, while all other variables were updated via Kalmann filters, providing increasingly uncertain estimates as to their true values. The model then learned to maximize its reward in each task by discovering through trial and error which variables were most important to regularly update (e.g. distance from edge of the sidewalk for sidewalk following, distance and direction of the oncoming object for litter collection) in order to pick the best actions for its task - see Figure 2.9 for an example. Results from this experiment showed that the agent using learned gaze scheduling to update state variables earned more average reward per episode than baseline agents that used round-robin scheduling for variable updates. Subsequent experiments over the next decade extended this framework to a virtual driving task[28][29][15], further validating the uncertainty-reward model (see [21] for more details), but the first attempts to go beyond variable updates and predict attention maps at full pixel-level detail came in 2018 with Zhang et al.'s Attention-Guided Imitation Learning (AGIL) network[37]/[21].

The AGIL network trains an imitation learning agent to play Atari games in a two-stage learning process. First, a top-down visual attention network is trained which learns to predict where in the game screen a human is most likely to look. The saliency maps predicted by this network are then used to augment the training process of a policy network, which outputs actions for the current frame. The visual attention network is shown in Figure 2.10. Three separate channels are each trained to make their own individual predictions through a series of convolutional and transposed convolutional layers. Each channel aims to capture different features of the input frames by preprocessing them in different ways. The top channel processes the frames with no modifications, the middle channel processes optical flow maps, and the bottom channel computes bottom-up saliency features such as edges and corners. The predictions from each branch are then averaged and normalized with a softmax function to output a single probability map. A very similar model which uses a semantic segmentation branch in place of the bottom-up saliency branch is the DR(eye)VE model[23] of Palazzi et. al, described in detail in Section 4.1.4, which predicts gaze for the driving task.

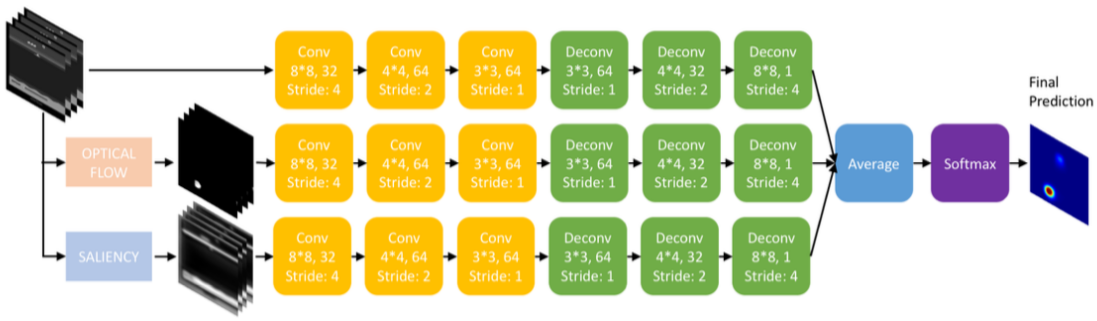


Figure 2.10: AGIL Visual Attention Network (taken from [37]), [21].

The AGIL policy network, shown in Figure 2.11, uses a two-channel architecture to take the probability maps produced by the attention network into account. In order to avoid any loss of overall information, the top channel takes the current image frame without any alteration, and processes it with a convolutional network. The bottom branch, however, performs an elementwise multiplication of the current frame with the attention map produced by the network, masking the image so that only the regions which were chosen by the attention network remain visible. This masked image is then processed by an identical convolutional network, and the output features are averaged and processed by a fully connected layer to predict actions for that frame. By learning in this manner, the AGIL agent outperformed a baseline imitation learning agent in all eight games for which it was trained, strongly validating the use of the visual attention model.

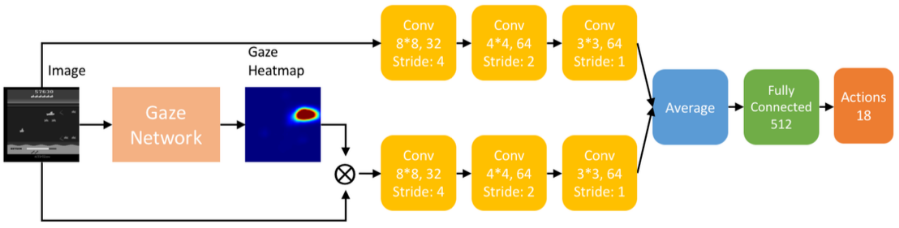


Figure 2.11: AGIL Policy Network (taken from [37]), [21].

2.2.3 Evaluation Metrics for Saliency Prediction

A wide variety of metrics exist for assessing the similarity between predicted and ground truth saliency maps. Different metrics can rank saliency models differently based on a variety of factors such as treatment of false positives and negatives, accounting for central biases, and how saliency maps have been pre-processed. While there is no 'best' metric for evaluating how well a model can predict human gaze, knowledge of the properties of each metric can be very useful when training and scoring models for specific applications. In "What do different evaluation metrics tell us about saliency models"[4], Bylinskii et al. provide a comprehensive description of the strengths and weaknesses of 8 different saliency metrics: Area under ROC curve (AUC), shuffled AUC (sAUC), Normalized Scanpath Saliency (NSS), Correlation Coefficient (CC), Earth Mover's Distance (EMD), Similarity (SIM), Kullback-Liebler Divergence (KL), and Information Gain (IG). In addition, suggestions are made for which metrics should be used based on different dataset characteristics and targeted applications.

The above metrics can be divide into two categories based on how they require ground truth saliency maps to be represented for their calculations. *Location-based* metrics (AUC, sAUC, NSS, IG) represent ground truth maps as a set of discrete fixation locations, and evaluate a model's accuracy in correctly classifying image pixels as either containing a fixation or not. *Distribution-based* metrics (SIM, CC, EMD, KLD) interpret all saliency maps as continuous distributions, and evaluate the similarity between ground-truth and predicted distributions. Of course, since eye trackers record gaze in terms of discrete locations, ground-truth distribution maps cannot be directly observed. Instead, these maps obtained by Gaussian blurring of the fixation locations. In this project we represent saliency maps as normalized probability distributions (for a detailed description of how we generate saliency distributions from fixation locations, refer to Section 3.1.4), and as such we evaluate our models using distribution-based metrics. Of the four metrics presented in the paper, we discard SIM due to the fact that it can be unpredictably affected by Gaussian blurring, which we use for creating our fixation maps, and EMD due to its computational expense and difficulty to optimize. We therefore evaluate our models using KLD and CC.

Kullback-Liebler Divergence

Kullback-Liebler Divergence is a commonly used metric for measuring the difference between two probability distributions. For a predicted saliency map X and a ground-truth map Y, the KLD is defined:

$$D_{KL}(Y||X) = \sum_i Y(i)log\left(\epsilon + \frac{Y(i)}{\epsilon + X(i)}\right) \tag{2.1}$$

where ϵ is a small constant for regularization. This metric is computed on a per-pixel basis for both images, with lower total values indicating greater similarity. KLD is very sensitive to missing ground-truth fixations - for pixels where the model predicts a near-zero probability of fixation while the ground-truth probability is high, an undesirably large value will be added. As such, models trained using KLD as a loss function will be less likely to produce sparse prediction maps, as doing so may cause them to miss image regions containing fixations, for which they would be strongly penalized. This is a useful feature for applications which rely on detecting multiple features in input images.

Correlation Coefficient

Pearson's Correlation Coefficient is a well-known statistical method of measuring the linear correlation between two dependent variables. For predicted and ground-truth saliency maps X and Y, CC is defined:

$$CC(Y||X) = \frac{cov(X,Y)}{\sigma(X) * \sigma(Y)} \tag{2.2}$$

where σ indicates the standard deviation. CC values are high in locations where the predicted and ground-truth maps have similar values. As such, higher total values indicate greater image similarity. False positives and false negatives are penalized equally by this metric, making it an overall fair and accurate metric for assessing the similarity of two saliency maps.

2.3 CARLA Driving Simulator

The driving experiments in this project are performed on the CARLA (Car Learning to Act) simulator[8], version 0.8.2 (stable). CARLA is a high-fidelity, open-source urban town driving simulator. It is built on Unreal Engine 4 and provides state-of-the-art rendering quality, realistic physics and a variety of useful tools which allow users to define and interact with agents that operate in the simulated world. In this section, we describe the design and functionality of the simulator.

2.3.1 Architecture

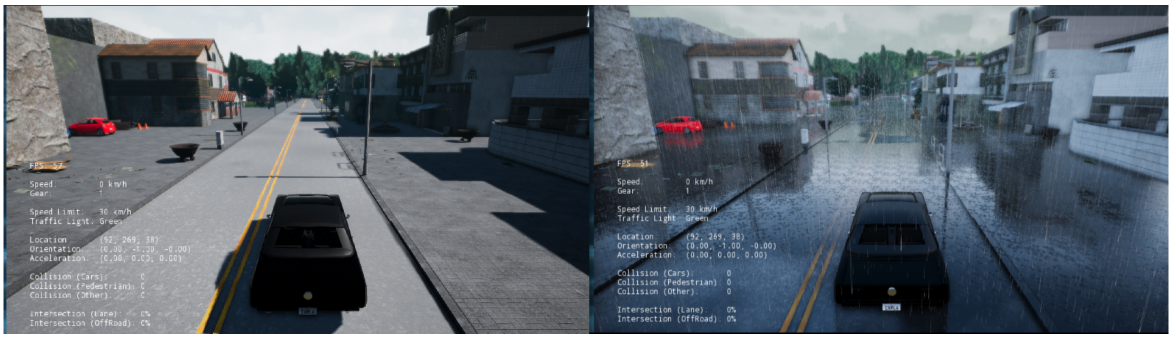
CARLA can be launched in two modes: standalone and server. The standalone mode allows the user to launch CARLA like a videogame, controlling the vehicle with keyboard inputs and driving around the environment. In server mode, the vehicle is controlled by an external Python client application, which sends driving instructions to the server and receives sensor readings and meta-information as output. The server can be run using either a variable time step, which attempts to keep the server running at a realistic speed for interaction with a human, or a fixed time step which allows the simulation to run faster and enhances consistency (highly recommended when recording data for autonomous agents to train on). The server interacts with the client via sockets which are bound to port numbers 2000, 2001, and 2002 by default.

The commands sent by the client fall into two categories: actions and meta-commands. Actions are the control signals which are used to drive the vehicle, such as throttle, brake and acceleration. These actions can be input by a human driver with a keyboard or steering wheel setup, as in standalone mode, or they can be generated by some intelligent agent such as a trained neural network. Meta-commands can be used to configure most aspects of the simulation, such as weather settings, episode timeouts, and more. These configurations can be either pre-defined in a settings file, by default named CarlaSettings.ini, or configured while the simulation is running via the client. A number of example scripts are automatically included in the CARLA repository, showing the user how to launch a client, control a vehicle via keyboard commands or computer-generated commands, read and record sensor data from the server, and run scenarios for assessing a model's performance.

2.3.2 Environment

The CARLA environment contains two manually designed towns, constructed using realistic 3D models of a wide variety of different buildings, terrains and dynamically spawning objects. Town 1 contains 2.9km of drivable roads and is used for both training and evaluating driving agents, while Town 2, containing 1.4km of drivable roads, is typically reserved for testing purposes only. At each server reset, non-player vehicles and pedestrians are randomly spawned within the environment, in numbers which can be configured by the Python client. The non-player vehicles operate using a simple controller which allows them to lane follow, make decisions at intersections, and avoid collisions with other cars and pedestrians. Pedestrians wander around the town with an overall tendency to remain on the sidewalks and marked pedestrian crossings, but with the freedom to cross roads at any point.

To increase the visual diversity of the simulation, the appearance of vehicles and pedestrians is randomized from a large selection of possible colors, clothing items and miscellaneous objects such as guitar cases, cell phones or umbrellas. At the time of the stable release, CARLA also contains two lighting conditions (Noon, Sunset), and seven environmental conditions (Clear, Cloudy, Wet, WetCloudy, MidRain, HardRain, SoftRain), for a total of fourteen possible environmental settings. An example of two different weather settings can be seen in Figure 2.12.


```
FPS: 57

Speed:            0 km/h
Gear:             1

Speed Limit:      30 km/h
Traffic Light:    Green

Location          (92, 269, 38)
Orientation:      (0.00, -1.00, -0.00)
Acceleration:     (0.00, 0.00, 0.00)

Collision (Cars):          0
Collision (Pedestrian):    0
Collision (Other):         0

Intersection (Lane):       0%
Intersection (OffRoad):    0%
```

```
FPS: 51

Speed:            0 km/h
Gear:             1

Speed Limit:      30 km/h
Traffic Light:    Green

Location:         (92, 269, 38)
Orientation:      (0.00, -1.00, -0.00)
Acceleration:     (0.00, 0.00, 0.00)

Collision (Cars):          0
Collision (Pedestrian):    0
Collision (Other):         0

Intersection (Lane):       0%
Intersection (OffRoad):    0%
```

Figure 2.12: The same scene with two different weather settings in the CARLA simulator

2.3.3 Measurements

CARLA provides users with a wealth of information about the simulation state via a configurable sensor suite and large amounts of meta-information which the user can access and record. Four types of configurable vehicle sensors are provided:

- A standard RGB camera which applies some post-processing effects (e.g. Bloom, Grain Jitter, Depth of Field) to the recorded images to increase the realism of the scene.
- A depth map camera with a max range of 1km
- A semantic segmentation camera which provides ground-truth semantic segmentation maps of the scene, divided into 12 categories: None, Buildings, Fences, Other, Pedestrians, Poles, RoadLines, Roads, Sidewalks, Vegetation, Vehicles, Walls, and TrafficSigns.

- A rotating Lidar implemented with ray-casting which returns Point Cloud maps of the scene.

Each of these sensors can be manually configured to alter the field of view of the camera, as well as the position and orientation with respect to the vehicle. The Python API provides a command-line option for users to write the camera recordings to disk while the simulation is running, with each consecutive frame of the simulation producing a PNG image which is written to an output directory.

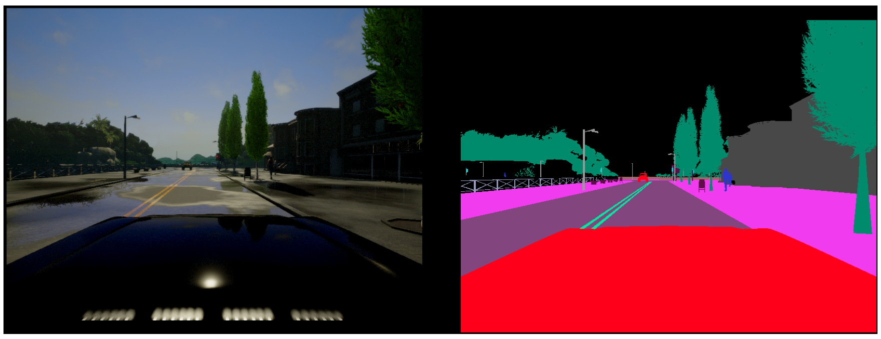


Figure 2.13: Example outputs for the RGB and SemSeg cameras

The user can also record a range of measurements about the state of both their player agent and other non-player agents. The player measurements include location coordinates, vehicle orientation, speed, acceleration, collision intensity values, and percentage of vehicle overlap with opposite lanes or sidewalks. Non-player agents include pedestrians, vehicles, traffic lights and road signs, and the measurements provided for each include location, speed (where applicable), and the locations and dimensions for a bounding box surrounding the agents. These sensors and measurements provide very useful feedback in the training and evaluation of autonomous driving agents.

Chapter 3

Data Collection & Preprocessing

A significant component of this project was the collection and pre-processing of two datasets. The first, which we will refer to as the Gaze dataset, consists of approximately 3 hours of eye movement data collected from human subjects while driving in the CARLA simulator. The second, referred to as the Driving dataset, is comprised of 11 hours of driving data from an expert autopilot agent provided by the CARLA simulator.

3.1 Gaze Dataset

3.1.1 Experimental Setup and Procedure

The collection of the Gaze dataset took place in the Brain and Behaviour Laboratory in the Royal School of Mines building. The purpose of these experiments was to track and record the eye movements of humans while driving. To make the experiment feel as realistic as possible for the subjects, we use a modified version of the CARLA environment which has been configured for virtual reality, allowing subjects to freely view the environment as they would when driving in the real world. A Ferrari 458 Italia USB steering wheel is used to control the movements of the simulated vehicle, with buttons located on the steering wheel used to control the gas, brake and reverse functions. We record eye movements using an HTC Vive headset equipped with SMI Eye-tracking plugins. The configuration of CARLA for virtual reality and the code for obtaining and recording the location of the subjects' gaze in the CARLA world coordinate system was written by a previous Master's student, Julien Gerard. A picture of our experimental setup is shown in Figure 3.1.

In the experiment, subjects were instructed to drive for six three-minute episodes in CARLA's Town 1. Each episode used a different weather preset to increase diversity

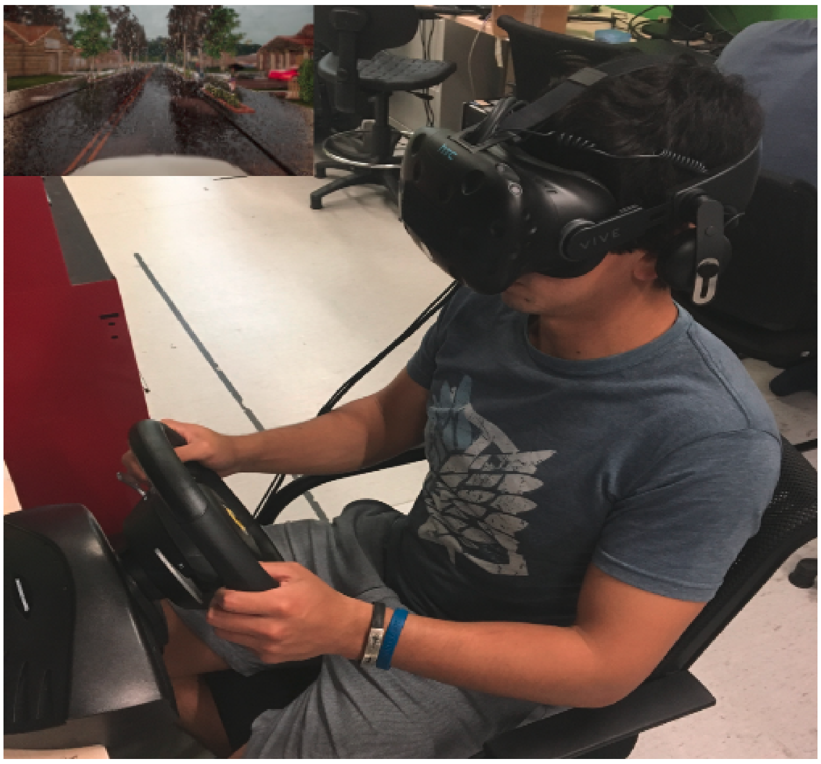


Figure 3.1: Experimental Setup for recording the Gaze Dataset

in the recorded images; the six settings used were ’Clear Noon’, ’Cloudy Noon’, ’Mid Rainy Sunset’, ’Hard Rainy Noon’, ’Wet Noon’, and ’Clear Sunset’. Before the experiment began, subjects were given several minutes to familiarize themselves with driving in the simulation, and a calibration of the eye tracker was performed. Subjects were then asked to drive around in a safe manner, respecting all traffic lights and road rules and giving pedestrians right of way. No specific navigation directions were given, thus subjects simply chose a random direction at every intersection. In order to include data on gaze patterns when recovering from a mistake, subjects’ steering commands were occasionally overridden by external commands, causing their vehicle to drift briefly to the left or right.

3.1.2 Data Collection

The VR CARLA environment is run on a Windows 10 laptop with 16GB of RAM, equipped with an NVIDIA GeForce GTX 1070 graphics card and an i7-7700HQ @ 2.80GHz CPU. In the simulation, a single camera is mounted on the roof of the vehicle, recording images at a rate of 25 frames per second, at a resolution of 400x300 pixels. CARLA’s provided method for saving camera images involves writing single images to disk in PNG format at each simulated time step. However, this disk-intensive process caused a severe slowdown of the simulation, rendering it unplayable for humans. To fix this problem, the recording method was rewritten to store up to 1000 recorded images in a RAM buffer, which when full is saved to disk as a compressed numpy array of size 1000x400x300. This results in about a half-second freezing of the simulation when the buffer is flushed, but otherwise allows the simulation to run smoothly.

In addition to saving camera outputs, at each time step the absolute position of the user's gaze in the CARLA world coordinate system is recorded. This information is acquired by means of an invisible 'GazeCursor' object provided by the SMI EyeTracking plugin for Unreal Engine and added into CARLA via the Unreal Editor. The cursor object is placed by determining the 3D angle of the user's gaze, projecting a vector in this direction from the user's current location, and placing the GazeCursor object at the coordinates where the vector intersects with an object in CARLA. This information is then written to a binary file, along with the corresponding frame number which is used to sync the gaze information with the frames recorded by the camera.

Each episode was recorded for 4500 frames, plus a small window of additional frames at the beginning and end of each episode which was not included in the dataset, but was used for some of the computations described below, such as fixation map and optical flow calculations. Eleven sets of six episodes were recorded in total, for a total of 297,000 frames, but one subject's data was excluded for poor driving behaviour, and another subject's data was excluded due to an error with the eye tracker during the experiment. The final dataset therefore consists of 243,000 frames, corresponding to 2.7 hours of driving data.

3.1.3 World-to-Camera Gaze Projection

In order to map the 3D position of a subject's gaze to a 2D location in the recorded camera frames, we require a mathematical transformation to map between absolute CARLA coordinates (X,Y,Z) and image pixel coordinates (x,y). This World-to-Camera transformation is known as forward projection, and can be calculated as follows. First, we must transform the 3D gaze position from the World coordinate system (CARLA coordinates) to the Camera coordinate system, in which the camera is always located at the origin, with the lens pointed down the Z axis, and the X and Y axes defining the image plane. This is a rigid 3D transformation which requires knowledge of the rotation and translation of the camera relative to the world; the matrix describing this 3D transformation is referred to as a camera's *extrinsic* matrix. This matrix takes the following form, with the top-left 3x3 matrix describing the world-to-camera rotation, the top-right 3x1 matrix describing the world-to-camera translation, and the bottom row added for convenience to make the coordinates homogeneous for future calculations:

$$[R|t] = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Next, we must transform our coordinates from the 3D Camera coordinate space to a 2D image space. This is known as perspective projection. We accomplish this by making use of the fact that the camera is pointed along the Z axis, and project the object whose 2D coordinates we wish to obtain onto an X-Y plane located f distance from the camera along the Z axis, where f is the focal length of the camera, as shown in Figure 3.2. To scale the returned coordinates to the resolution of the images produced by the camera, we also multiply the results by the ”principal point offset”. The principal point is the intersection of the camera’s Z axis with the image plane, with the principal point offset therefore being the offset between the principal point and the camera’s origin. The focal length information and the principal point offset can then be combined into a single matrix, known as the camera’s *intrinsic* matrix K, and multiplied by the 3D Camera coordinates of the object of interest to retrieve the desired pixel coordinates, as shown below:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} f_x & 0 & W/2 & 0 \\ 0 & f_y & H/2 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \longrightarrow x = \frac{x'}{z'}, y = \frac{y'}{z'} \tag{3.1}$$

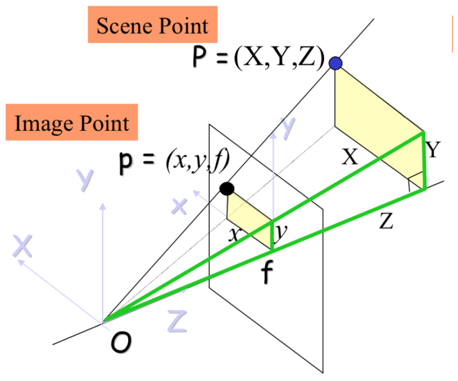


Figure 3.2: Projection from 3D Camera coordinates to 2D Image coordinates

The forward projection process is summarized in Algorithm 1. In the rare case that the human is looking at something outside the field of vision of the camera, the pixels returned do not fall within the correct range of the image borders, and are therefore excluded from any future computations using these values, with this frame simply being marked empty of any fixations.

Both the extrinsic and intrinsic matrices of the vehicle-mounted camera are provided by the Unreal Engine at each time step, thus using the above process we are able to

Algorithm 1 Retrieve X,Y Pixel Coordinates

```
1: function GetXY($eyeX, eyeY, eyeZ, M\_intrinsic, M\_extrinsic$)
2:     $World\_pos \leftarrow [eyeX, eyeY, eyeZ, 1]$                ▷ Homogenized
3:     $Camera\_pos\_3d \leftarrow M\_extrinsic \cdot World\_pos$
4:     $Pos\_2d \leftarrow M\_intrinsic \cdot Camera\_pos\_3d[:3]$
5:     $Norm\_Pos\_2d \leftarrow [Pos\_2d[0]/Pos\_2d[2], [Pos\_2d[1]/Pos\_2d[2], Pos\_2d[2]]$
6:     if $Norm\_Pos\_2d[2] > 0$ then          ▷ Check object is not behind camera
7:         $X\_2D \leftarrow 2d\_pos[0]$
8:         $Y\_2D \leftarrow 2d\_pos[1]$
       return $X\_2D, Y\_2D$
```

obtain the X,Y location of the driver's gaze within each frame. To verify the correctness of the returned locations, a test was performed in which the GazeCursor object was made visible within the CARLA environment, and a short driving sequence was recorded with the cursor appearing as a red globe in the captured frames. Forward projection was then used to annotate the images with a green circle at the calculated locations, and it was verified that the rectangles and the GazeCursor appeared at the same locations.

3.1.4 Fixation Map Generation

Using the method described above, each frame is associated with a single X,Y position indicating the driver's current gaze fixation within the frame. Rather than only considering a single point, however, it is more appropriate to consider all fixations made within a small temporal window of the current frame when constructing a fixation map. This includes fixations that have yet to occur at the current time step, which may seem counterintuitive but in fact provides a valid representation of a human's current attention distribution, given that the upcoming movements of the eye are pre-planned, as indicated by studies such as those of Hoppe et al. in 2019[12].

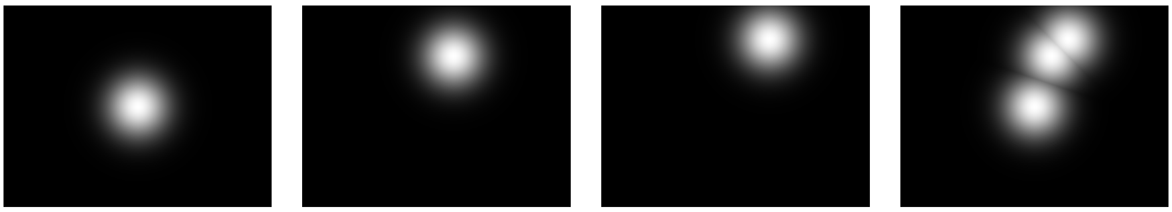


Figure 3.3: Visualization of how Gaussians are generated for three individual fixations and combined via the max operation into a single attention map

As described in Section 2.2.3, our models and evaluation metrics require us to treat saliency maps as continuous probability distributions. Unlike in bottom-up saliency approaches, where benchmarks such as MIT300 set a universal standard for fixation map computation, there is no well-established method for creating continuous maps

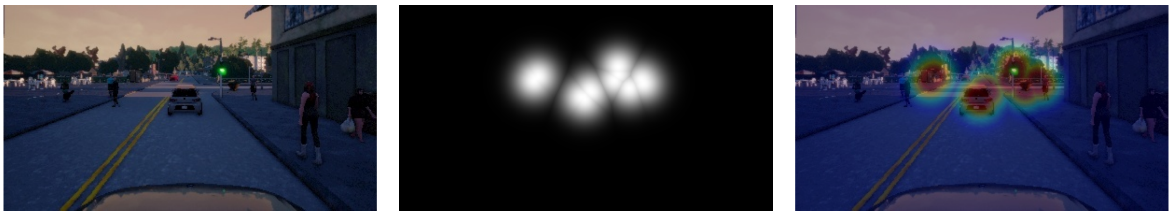


Figure 3.4: Example of the fixation map generated for a single camera frame, and the corresponding attention heatmap produced by blending the images

from fixation data in a top-down setting. In this project, we use the following approach, proposed by Palazzi et al. [23] in 2018. To construct a fixation map F_t for a frame at time t, we first retrieve the absolute gaze locations from time steps $t - 12$ to $t + 12$, for a total of 25 fixations (including the current frame), representing a 1-second window of time centered on t. These absolute locations are then converted to X,Y coordinates within the current frame using the same process described in Section 3.1.3, but with all (X,Y,Z) gaze locations transformed using only the extrinsic matrix for the current frame. Starting with 25 400x300 arrays of zeros, a multivariate Gaussian is generated, centered on each of these projected X,Y coordinates, with a variance of $\sigma^2 = 50$ pixels to account for any spatial acquisition errors in the eye tracking system and the fact that humans see in greater detail in a small window surrounding the location of their gaze. The 25 resulting images are then combined via a max operation, and the final image is normalized, resulting in a continuous probability distribution modeling the subject's visual attention for the current frame. This process is summarized in Algorithm 2, and an example of its creation and outputs are visualized in Figures 3.3 and 3.4.

Algorithm 2 Compute Fixation Map from Individual Fixations

```
1: function COMPUTE_FMAP($GazeXYList, Sigma$)
2:     $F_t \leftarrow$ Empty[Width, Height]
3:     for $X, Y$ in $GazeXYList$ do
4:         $G \leftarrow$ multivariate_normal($X, Y, Sigma$)
5:         $PDF \leftarrow$ probability_density_function($G$)
6:         $F_t \leftarrow$ elemwise_max($F_t, PDF$)
7:     $F_t \leftarrow F_t/\text{sum}(F_t)$
8:     return $F_t$
```

3.1.5 Optical Flow Map Generation

For each frame in each episode, a dense optical flow map was generated using Gunnar Farneback's optical flow algorithm [9]. This algorithm generates an image pyramid, with each pyramid level having a lower resolution than the previous image. Optical flow tracking begins at the lowest (most coarse resolution) level of the pyramid, by approximating the two image frames to be compared with quadratic polynomials, and observing how the polynomials are transformed under the translation from one frame to the next. Key points identified at the lower resolutions are then passed on to the higher-resolution levels, refining the tracking at each level. This method is more computationally expensive than other methods such as Lucas-Kanade, but typically produces more accurate results. An example of two camera frames and their resulting optical flow map is shown in Figure 3.5 While testing the creation of these dense flow maps in the driving sequences, it was observed that pedestrians did not tend to walk fast enough between two consecutive frames to produce a noticeable output in the flow map. As pedestrians are highly likely to be a focus of attention for someone driving a car, it was decided to calculate the optical flow between the current frame and the frame captured 3 time steps previously, therefore allowing pedestrian movements to be more accurately picked up and represented.

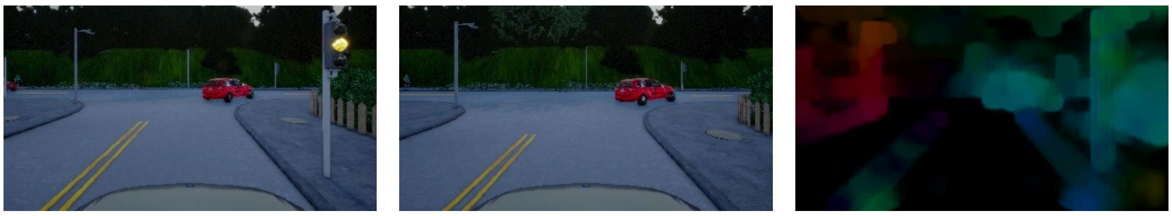


Figure 3.5: Farneback dense optical flow map produced by two camera frames separated by 3 time steps

3.1.6 Data Labeling

After collection, the data was manually annotated in two ways. First, each frame was labeled with the current high-level intention of the driver, corresponding to the high-level commands issued by CARLA's route planner for autonomous driving vehicles. These intentions are categorized simply into ”Lane Following”, ”Left Turn”, ”Right Turn”, and ”Straight”, with the latter three commands signaling the driver's intention when approaching an intersection. These labels are used to train the Intention-Branched DR(eye)VE Model described in Section 4.1.4. As with CARLA's route planner, the intention for a turn is defined when the vehicle comes within a short distance of the intersection, and returns to ”Lane Following” when the turn is completed. Of the 243,000 frames in the dataset, 211,651 (87.1%) are labeled as ”Lane Following”, 10,684 (4.4%) as ”Straight”, 10385 (4.3%) as ”Right Turn”, and the remaining 10,280 (4.2%) as ”Left Turn”.

As CARLA is an accurate representation of a small urban town, a significant portion of the collected data consists simply of periods where the driver is waiting at a traffic light. To distinguish between active driving sequences, where the driver must pay constant attention to avoid collisions or drifting from their lane, and traffic lights where minimal attention must be paid (and thus eye movements may be less predictable), a second manual annotation of the dataset was done, simply labeling all frames as either ’active’ or ’traffic’. Of the 243,000 recorded frames, 79,437 (32.7%)

are labeled as ’traffic’, with the remaining 163,563 (67.3%) labeled as ’active’. In Section 5.1.4, we see that our gaze prediction network performs substantially better on ’active’ frames.

3.1.7 Analysis, Curation

Upon collecting data from all subjects, some preliminary analysis was performed to inspect the overall gaze behaviour across subjects and to determine if any curation could be performed to reduce the size and improve the quality of the dataset. First, we calculated the mean fixation map for all frames, visualized in the left of Figure 3.6 (blended with the mean training frame). As is very common in gaze tracking experiments, there is a strong bias to look towards the center of the image. Naturally, this is also due to the fact that humans are constantly watching the road while driving. We also calculated the mean fixation map for traffic sequences versus active sequences, obtaining the results in the middle and right of Figure 3.6. In the traffic frames, there is a clearly visible shift in attention slightly above and to the right of the center of the image, where the traffic light would be located.

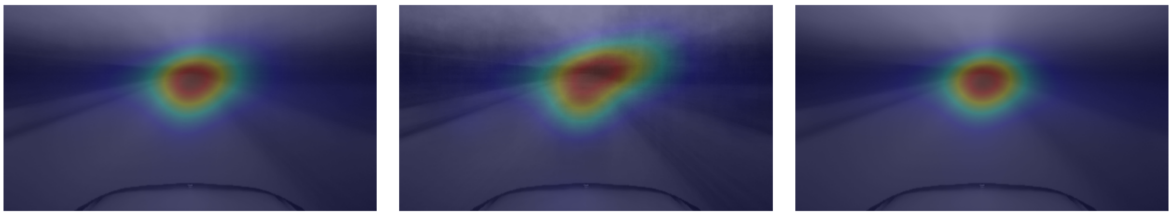


Figure 3.6: Blended mean fixation maps and driving frames for the entire dataset (left), traffic frames (middle) and active frames (right)

Next, we looked at the distribution of subjects’ gaze within the camera pixel space to determine if any regions could be safely cropped. We plotted the fixation distribution along X and Y axes, obtaining the results displayed in Figure 3.7. As already noted from the mean maps, fixations are most likely to occur in the center of the image. Along the X axis, the distribution is fairly symmetric, with fixations less likely to occur the farther one moves from the center of the image. A fair amount of fixations still occur even at the extreme edges of the image, almost certainly occurring when humans look down the road which they are planning to make a turn on. The Y axis’ distribution is not so balanced, with very few fixations occurring above the center of the image, and almost no occurrences at the borders of the image. This makes sense, as there is typically no need for humans to look above the vanishing point of the road while driving. Using this information, we decided to crop the top 80 pixel rows from each image, as in the 243,000 frames of the dataset, only 231 (<0.1%) of frames contained fixations in this area.

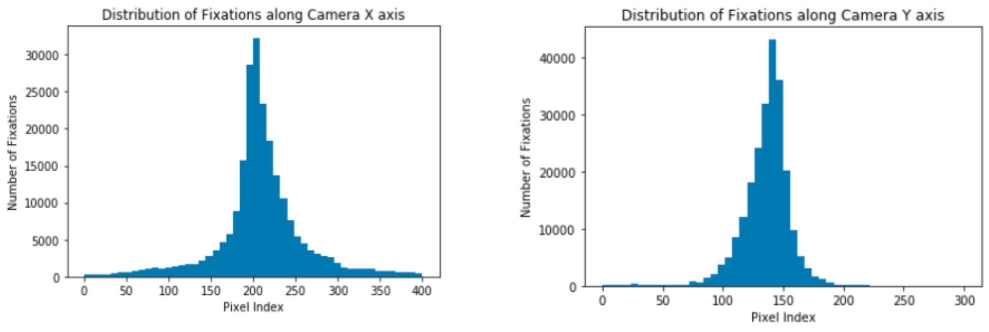


Figure 3.7: Distribution of human gaze fixations along X and Y camera axes for all recorded episodes

3.2 Driving Dataset

3.2.1 Expert Demonstrator

The data for the Driving dataset was generated by an 'expert' demonstrator agent provided by the CARLA developers. This agent has access to privileged information about the state of the server at all times, including a map of its environment and exact positions of all other agents in the simulation. For each episode, the agent is randomly spawned within the town, and provided with a destination. The weather settings, number of vehicles and pedestrians are all randomly chosen within the ranges defined by the the user. A route planner uses the A* algorithm to determine a path to the destination, and this path is then split into waypoints for the agent to navigate between. The throttle, gas and steering values for the agent to use are generated by a PID controller based on the current waypoints. The agent respects all speed limits and traffic lights, and attempts to avoid pedestrians by slowing down if any pedestrians are within 15 meters, and coming to a complete stop if they are within 5 meters. Upon successfully reaching the destination, the episode terminates and a new episode begins.

Each frame, along with the outputs from any cameras attached to the vehicle, the client generates a metadata file in JSON format with the following information:

Name	**Data Type**	**Physical Unit**	**Range**
Frame Number	Integer	-	-
Elapsed Time	Integer	Milliseconds	-
Vehicle Location X	Integer	-	-
Vehicle Location Y	Integer	-	-
Vehicle Location Z	Integer	-	-
Vehicle Orientation Vector	Float tuple	-	[0,1]
Vehicle Acceleration	Float	m/s^2	-
Vehicle Speed	Float	km/hr	-

Stopped (Traffic Light)	Boolean	-	{0,1}
Stopped (Vehicle)	Boolean	-	{0,1}
Stopped (Pedestrian)	Boolean	-	{0,1}
Vehicle Location X	Integer	-	-
High Level Command	Integer	-	{2,3,4,5}
Throttle	Float	-	[0,1]
Brake	Float	-	[0,1]
Steer Angle	Float	-	[-1,1]
Hand Brake Toggled	Boolean	-	{0,1}
Reverse Toggled	Boolean	-	{0,1}
Throttle Noise	Float	-	-
Brake Noise	Float	-	-
Steer Angle Noise	Float	-	-

Table 3.1: Driving metadata collected by CARLA's expert demonstrator for each frame

Another JSON file is generated for each completed episode, containing the number of pedestrians and vehicles present in the episode, the random seed used to spawn these objects, and the weather setting used.

To improve the diversity of the dataset and increase the number of state-action pairs visited, noise is periodically injected into the steering, brake and throttle commands generated by the PID controller. The inclusion of this noise in training datasets for autonomous vehicles has been shown to significantly improve the correlation between offline evaluation metrics and true driving performance[5]. The noise signals last from 2 to 4 seconds, and gradually increase in intensity over time to compensate for the controller's attempts to restore the vehicle to the desired state. When recording the metadata for the episode, the values for the controls generated by the PID controller and the values truly being used to control the vehicle as a result of the noise are recorded separately. This is to ensure that when training an agent for autonomous driving, the noisy signals are not used as a signal to emulate. At each time step, the controller will always be producing the ideal commands to recover from any drift caused by the noisy signals, thus providing any trained agents with data on how to recover from potential mistakes. If, however, the noisy commands result in the vehicle crashing, the episode is terminated and erased.

3.2.2 Data Collection

The data for the Driving dataset was generated using the same environmental parameters as the Gaze dataset. The data was collected in both CARLA towns, with the training set composed only of Town 1 images and the validation set containing images from both towns. Weather presets were randomly chosen from the 'Clear Noon', 'Cloudy Noon', 'Mid Rainy Sunset', 'Hard Rainy Noon', 'Wet Noon', and 'Clear

Sunset' presets. The number of pedestrians spawned ranged from 120 to 200, and the number of vehicles spawned ranged from 20 to 40. Episodes typically ranged between 1000 and 8000 frames, with 229 total episodes resulting in 657,456 frames, equivalent to 7.3 hours of driving at 25 frames per second.

3.2.3 Preprocessing

Rather than compute fixation maps on the fly while training an imitation learning network, which would drastically slow down the training process and require computing the same attention maps many times, we pre-compute the fixation maps for all frames in the training and test sequence using our best-performing fully trained Gaze Prediction network, the Intention-Branched DR(eye)VE model. The branch of the Gaze network used to predict the attention map for each frame was chosen based on the high-level command recorded in that frame's metadata file. All camera frames were cropped to match the dimensions of the Gaze dataset frames before computing fixation maps. The generated attention maps were then used to mask the images from which they were sampled, as depicted in Figure 3.8, for the same purpose of avoiding repeatedly performing the masking operation while training the network. Three different types of masks are generated: hard attention, soft attention, and baseline masks. Hard attention masks are simply computed via element-wise multiplication (denoted by the $\odot$ symbol in our equations) of the attention map F_t and the input camera image I_t. Soft attention masks are computed by reducing the brightness of the images by a factor λ in areas where the attention map does not overlap with the image. Finally, the baseline masks are computed via element-wise multiplication of the camera images with the mean fixation map of the dataset G. Figure 3.8 shows an example driving image with each different type of masking applied to it.

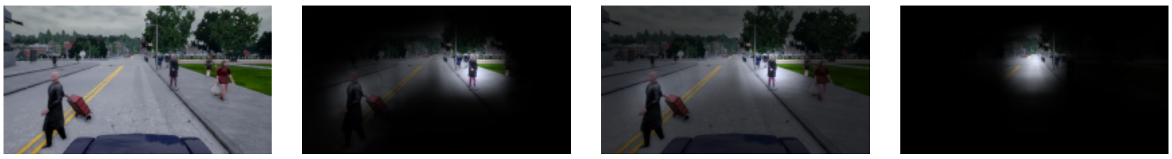


Figure 3.8: Example driving image with different types of attention masking applied. From left to right: Raw image, Hard masking, Soft masking, Baseline masking. Hard and soft masks use the attention map generated by our DR(eye)VE gaze prediction model, while the baseline mask uses the mean fixation map for the training dataset.

$$M_{hard} = I_t \odot F_t \tag{3.2}$$

$$M_{soft} = \lambda * I_t + (1 - \lambda) * I_t \odot F_t \tag{3.3}$$

$$M_{base} = I_t \odot G \tag{3.4}$$

Once all frames had been processed in this manner, all images and attention maps were downsampled by a factor of 2, and the entire dataset was converted into

TFRecord format for training with the Imitation Learning networks. To avoid running into memory issues when loading these files, the dataset was split into TFRecords containing 2000 labeled training samples each. Each sample was stored as a Tensorflow Example dictionary in the following format:

Key	**Feature**
’image’	200x110 RGB Camera Image
’hard’	200x110 Hard-Masked Camera Image
’soft’	200x110 Soft-Masked Camera Image
’base’	200x110 Baseline-Masked Camera Image
’speed’	Target prediction value for vehicle speed
’gas’	Target action value for vehicle throttle
’brake’	Target action value for vehicle brake
’steer’	Target action value for vehicle steering angle
’command’	High-level command for network branching

Table 3.2: Description of Tensorflow Example Dictionaries used for training autonomous driving agents

Chapter 4

Deep Learning

In this chapter, we give an overview of the architectures and training procedures for the various deep learning models used in the gaze prediction and autonomous driving tasks. Details on the implementations and results of each model can be found in Chapters 5 and 6.

4.1 Gaze Prediction Models

4.1.1 Deep Gaze II

Deep Gaze II [18] is currently the top-ranked model on the live MIT300 benchmark for predicting bottom-up visual saliency in images. This network trains in two stages, first making use of powerful pre-trained networks for object recognition to obtain an estimate of important image features, and then using a second, trainable network to fine-tune for the saliency prediction task. The overall architecture is shown in Figure 4.1. In the first stage, input images are fed into a frozen VGG-19 [26] network which has been pre-trained on the ImageNet object recognition dataset, with the final fully-connected layers used for classification removed. A selection of the activated feature maps from specific layers, namely the conv5_1, relu5_1, relu5_2, conv5_3, and relu5_4 layers, are then extracted and resized to match the dimensions of the conv2_1 layer. The extracted layers were chosen by comparing results from a random search of different sets of layers. The resized layers are then combined into a single tensor (of size 5x512), which forms the input to the model's secondary network, the "readout" network. The readout network is simply formed of four layers of 1x1 convolutions, typically used to reduce the filter dimensionality of deep networks, each followed by a ReLu activation layer. The number of features in each layer, in order, is 16, 32, 2, and finally 1 for a single output channel, which is then smoothed by convolution with a Gaussian filter. Note that since 1x1 convolutions preserve the size of their inputs, the height and width of the features remains the same throughout the readout network.

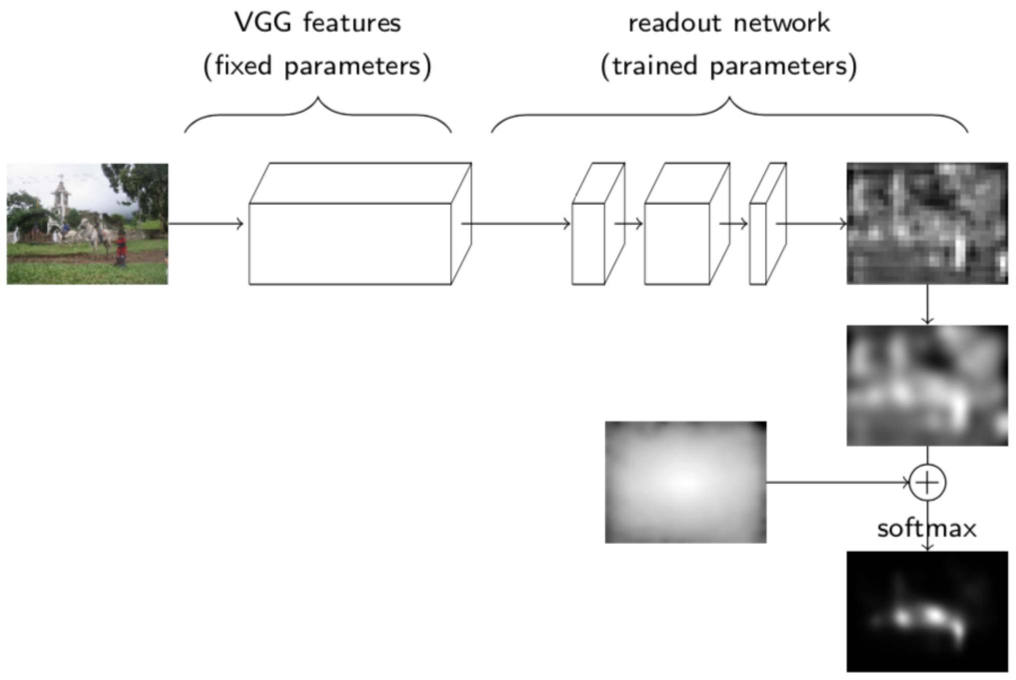


Figure 4.1: DeepGaze II Model Architecture (taken from [18])

Humans exhibit a strong tendency to look at the center of images, known as a central bias [30]. To model this bias, a baseline bias term is calculated using the average of all fixations across the training dataset, and this baseline is combined with all predictions made by the readout network. Finally, a softmax function is used to convert the output into a probability distribution. Pre-trained versions of the DeepGaze II network are available for download at `https://deepgaze.bethgelab.org/`, along with Jupyter Notebooks demonstrating their use and allowing for the substitution of center biases from custom datasets to yield better results. We use this pretrained model, with a substituted center bias from our dataset, as a baseline comparison for the remainder of the gaze models discussed in this chapter, all of which are retrained from scratch on our Gaze dataset.

4.1.2 MLNet

Multi-Level Net (MLNet) [7] is another bottom-up saliency model, which in 2016 achieved the highest score in all metrics on the SALICON (Saliency in Context) dataset [14]. Unlike the MIT300 dataset, SALICON does not actually measure ground truth eye movements from subjects when viewing an image, but instead approximates visual attention by having subjects move a mouse to the location they are currently paying attention to, allowing for much larger-scale data collection.

As the name suggests, MLNet extracts features from different levels of its feature extraction network, using a combination of low, mid and high-level features to form its predictions rather than using only high-level features from the final convolutional stage. The feature extraction network used is a pre-trained model of VGG-16, with two minor modifications: the final max-pooling layer is removed, and the second-to-last max-pooling layer has its stride reduced from 2 to 1. These changes are made in an effort to reduce the amount of total downsampling done by the network; while a standard VGG-16 network with 5 max-pooling layers downsamples images by a factor of 32, this modified architecture downsamples only by a factor of 8. This reduces the amount of upsampling required to scale saliency maps back to the original input space, allowing for more finely detailed outputs after rescaling.

Layer	**Output Height**	**Output Width**	**Output Channels**	**Number of Parameters**
Feature Extraction Network (VGG-16)				
Input	480	640	3	**0**
Conv2D (3x3)	480	640	64	64 x ((3x3) x 3)) + 64 = **1,792**
Conv2D (3x3)	480	640	64	64 x ((3x3) x 64)) + 64 = **36,928**
Pooling (2x2)	240	320	64	**0**
Conv2D (3x3)	240	320	128	128 x ((3x3) x 64)) + 128 = **73,856**
Conv2D (3x3)	240	320	128	128 x ((3x3) x 128)) + 128 = **147,584**
Pooling (2x2)	120	160	128	**0**
Conv2D (3x3)	120	160	256	256 x ((3x3) x 128)) + 256 = **295,168**
Conv2D (3x3)	120	160	256	256 x ((3x3) x 256)) + 256 = **590,080**
Conv2D (3x3)	120	160	256	256 x ((3x3) x 256)) + 256 = **590,080**
Pooling (2x2)	60	80	256	**0**
Conv2D (3x3)	60	80	512	512 x ((3x3) x 256)) + 512 = **1,180,160**
Conv2D (3x3)	60	80	512	512 x ((3x3) x 512)) + 512 = **2,359,808**
Conv2D (3x3)	60	80	512	512 x ((3x3) x 512)) + 512 = **2,359,808**
Encoder Network				
Concatenate	60	80	1280	**0**
Dropout (0.5)	60	80	1280	**0**
Conv2D (3x3)	60	80	64	64 x ((3x3) x 1280)) + 64 = **737,344**
Conv2D (1x1)	60	80	1	1 x ((1x1) x 64)) + 1 = **65**
EltWise_Product	60	80	1	(60 / 10) x (80 / 10) = **48**
Activation(ReLu)	60	80	1	**0**
Total				**14,452,154**

Table 4.1: Multi-Level Net

Feature maps are extracted from three different convolutional stages of the network (highlighted in blue in Table 4.1), and concatenated into a single tensor. This tensor is then input into an encoding network, which consists of a 64x3x3 convolutional layer followed by a 1x1 convolutional layer to collapse the feature maps into a single output prediction. As in the Deep Gaze II model, these predictions are then combined with a bias term via element-wise multiplication before being converted to a probability function via the softmax function. Rather than pre-computing the bias, however, the MLNet model learns a custom bias U. This bias is initialized as a coarse (size $w'xh'$) mask of ones, which is upsampled to the size of the predictions (h, w) before being combined with the saliency predictions. The bias is then trained alongside the main network to adjust its coarse values based on the total loss.

The loss function used for training MLNet is shown below:

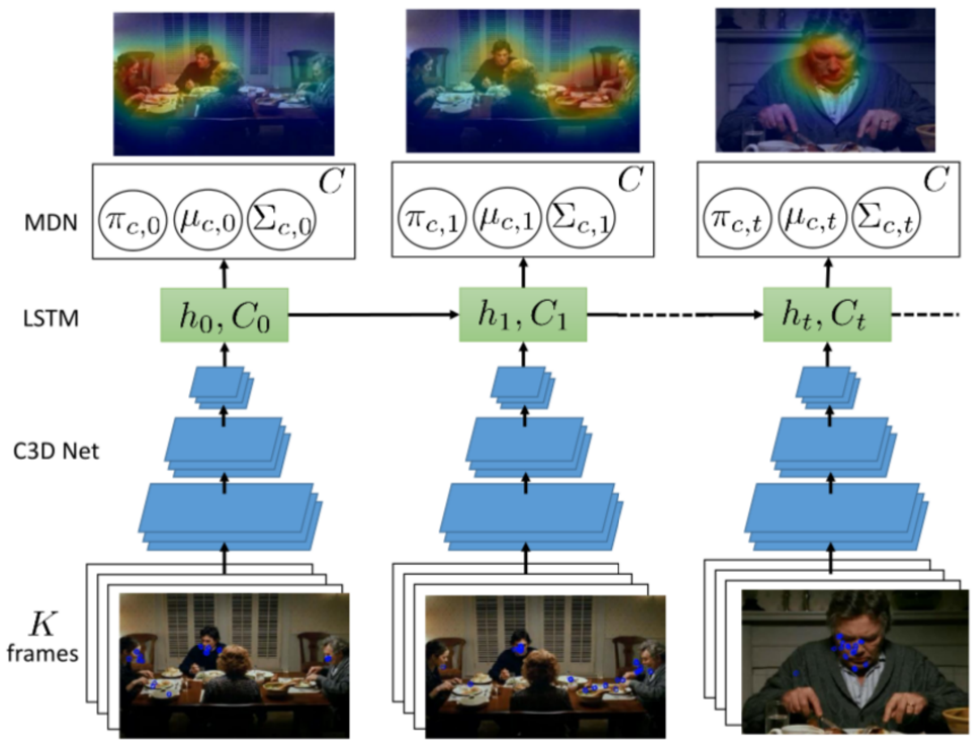
$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\frac{\phi(\mathbf{x}_i)}{max(\phi(\mathbf{x}_i))} - \mathbf{y}_i}{\alpha - \mathbf{y}_i}\right)^2 - +\lambda|\mathbf{1} - U|^2 \tag{4.1}$$

The first term in the summation measures the mean-squared error between the normalized network predictions, $\phi(x_i)$, and the ground-truth labels y_i, weighted by a linear function $\alpha - y_i$. This weighting function is added to increase the importance of high-probability pixels from the ground truth map, as the large majority of pixels have values at or near zero. With $\alpha = 1.1$, ground truth samples y_i with a value of zero have their loss weight decreased slightly, while higher valued samples (e.g. $y_i = 0.8 \rightarrow \alpha - y_i = 0.3$) have their loss weights boosted. The second term in the overall loss function is an L_2 penalty on the custom bias U, with $\lambda = 1$ / $(w' * h')$. Since U is initialized as a mask of ones, this $|1 - U|^2$ term penalizes the model for deviating too much from this original initialization, and encourages the overall network to decrease loss by adjusting convolutional weights rather than by adjusting the bias.

4.1.3 Recurrent Mixture Density Network

The Recurrent Mixture Density Network [2] (RMDN) proposed by Bazzani et al. in 2017 was the first deep network to use sequences of consecutive frames to predict saliency in video sequences in a top-down manner, outperforming all existing video saliency prediction methods on the Hollywood2 dataset, as well as achieving state-of-the-art results in action recognition on both the Hollywood2 and UCF101 datasets. An visual overview of the network is shown in Figure 4.2.

The input to the RMDN model is a 'clip' of K frames beginning at time t - k + 1 and ending at time t. This clip is first processed by a 3D Convolution Network (C3D) encoder based on the architecture proposed by Tran et al. in 2014 [31]. The full details of this C3D network can be seen in Table 4.2. This 3D architecture allows the network to capture short term spatio-temporal features such as motion, which can be critical in predicting human attention patterns. The C3D-processed clips are



MDN
LSTM
C3D Net
K frames
C
$\pi_{c,0}$ $\mu_{c,0}$ $\Sigma_{c,0}$
$\pi_{c,1}$ $\mu_{c,1}$ $\Sigma_{c,1}$
$\pi_{c,t}$ $\mu_{c,t}$ $\Sigma_{c,t}$
h_0, C_0
h_1, C_1
h_t, C_t

Figure 4.2: Recurrent Mixture Density Network (taken from [2])

then sequentially fed into a recurrent neural network using Long Short Term Memory (LSTM) [11] memory blocks. This allows the network to capture longer-term dependencies in the input video sequences. Finally, a saliency map is constructed in the form of a Gaussian Mixture Model with C mixture components. The GMM is defined by a set of parameters $\mu_c, \pi_c, \sigma_c, \rho_c$, representing the mean, mixture, variance and correlation coefficients for each of the C Gaussians. Each mixture parameter is obtained simply by processing the LSTM's 128-dimensional hidden state with a single fully-connected layer, referred to as the Mixture Density Network. The parameters are then each normalized with the appropriate functions (details can be found in Table 4.3) to output a single normalized probability distribution.

The C3D network parameters are obtained from the pre-trained action recognition model of Tran et al. [31], and remain frozen throughout the training of the model. The LSTM and Mixture Density Networks are jointly trained by optimizing the negative log-likelihood of the ground truth saliency maps on the predicted Gaussian Mixture Model, calculated as below:

$$L(v^i, a^i) = \sum_{t=0}^{T_i-1} \sum_{j=1}^{A} -log \left(\sum_{c=1}^{C} \pi_t^c \mathcal{N}_{v_i}(a_{t,j}^i; \mu_t^c, \sigma_t^c, \rho_t^t) \right) \tag{4.2}$$

Here v^i denotes the i-th video clip, a^i is the network's prediction, and the parameters for each of the C Gaussians $\mathcal{N}_{v_i}$ which make up the GMM are produced by the LSTM at each time step c, dependent upon the input clip v_i.

Layer	Output Depth	Output Height	Output Width	Output Channels	Number of Parameters
Input	16	128	171	3	**0**
Conv3D (3x3x3)	16	128	171	64	64x((3x3x3)x3)+64 = **5,248**
Pool3D (1x2x2)	16	64	85	64	**0**
Conv3D (3x3x3)	16	84	85	128	128x((3x3x3)x64)+128 = **221,312**
Pool3D (2x2x2)	8	32	42	128	**0**
Conv3D (3x3x3)	8	32	42	256	256x((3x3x3)x128)+256 = **884,992**
Conv3D (3x3x3)	8	32	42	256	256x((3x3x3)x256)+256 = **1,769,728**
Pool3D (2x2x2)	4	16	21	256	**0**
Conv3D (3x3x3)	4	16	21	512	512x((3x3x3)x256)+512 = **3,539,456**
Conv3D (3x3x3)	4	16	21	512	512x((3x3x3)x512)+512 = **7,078,400**
Pool3D (2x2x2)	2	8	10	512	**0**
Conv3D (3x3x3)	2	8	10	512	512x((3x3x3)x512)+512 = **7,078,400**
Conv3D (3x3x3)	2	8	10	512	512x((3x3x3)x512)+512 = **7,078,400**
Flatten	1	1	1	81920	**0**
Total					**27,655,936**

Table 4.2: C3D Encoder Network Architecture for RMDN

Layer	Output Shape	Number of Parameters
Input	(50, 81920)	**0**
LSTM	(50, 128)	4 x (81,920 x 128 + 128^2 + 128) = **42,009,088**
Dropout (0.5)	(50, 128)	**0**
Fully-Connected (Weight)	(50, 20)	(128 x 20) + 20 = **2,580**
Activation (Softmax)	(50, 20)	**0**
Reshape	(50, 20, 1)	**0**
Fully-Connected (Mean)	(50, 40)	(128 x 40) + 40 = **5160**
Reshape	(50, 20, 2)	**0**
Activation (ReLu)	(50, 20, 2)	**0**
Fully-Connected (Variance)	(50, 40)	(128 x 40) + 40 = **5160**
Reshape	(50, 20, 2)	**0**
ElemWise: $e^x + 1$	(50, 20, 2)	**0**
Fully-Connected (Correlation)	(50, 20)	(128 x 20) + 20 = **2580**
Reshape	(50, 20, 1)	**0**
Activation (tanh)	(50, 20, 1)	**0**
Concatenate	(50, 20, 6)	**0**
Total		**42,024,568**

Table 4.3: Mixture Density Network Architecture for RMDN, with 50 LSTM time steps

4.1.4 DR(eye)VE

The final gaze prediction model used is based on the DR(eye)VE model [23], proposed by Palazzi et al. in 2017 for top-down saliency prediction in the task of real-world driving. As with RMDN, the DR(eye)VE network uses clips of consecutive frames within a short time window as input. Rather than look for longer-term dependencies, however, the DR(eye)VE network simply makes predictions based on the input clip, making use of a deep multi-branch architecture to process the inputs in several different ways, very similar to the AGIL Gaze Prediction Network discussed in the Background chapter [37]. The original DR(eye)VE architecture contained three branches: one for processing RGB images, one for optical flow images, and one for semantically segmented images. In this project, we omit the semantic segmentation branch, as we were not able to attach both RGB and Semantic Segmentation cameras to the vehicle while maintaining an acceptable frame rate to allow humans to drive naturally, as explained in Chapter 3. An ablation study performed by the authors of the DR(eye)VE model in order to inspect the effect of each branch on the model's predictions showed that the semantic segmentation branch contributed very little to the final predictions of the model in comparison to the optical flow and RGB image branches, thus we do not expect that omitting this branch will significantly impact the performance of the network.

The overall model is therefore composed of two identical saliency branches, the first of which processes clips of raw RGB images, and the second of which processes clips of corresponding dense optical flow maps, pre-computed as described in Chapter 3. The two branches are first trained independently, and subsequently fine-tuned by training both branches simultaneously, such that the network learns to weight the contributions of each branch and optimize its performance. The overall architecture is depicted in Figure 4.3. The final prediction of the network is the normalized average of the individual branch predictions.

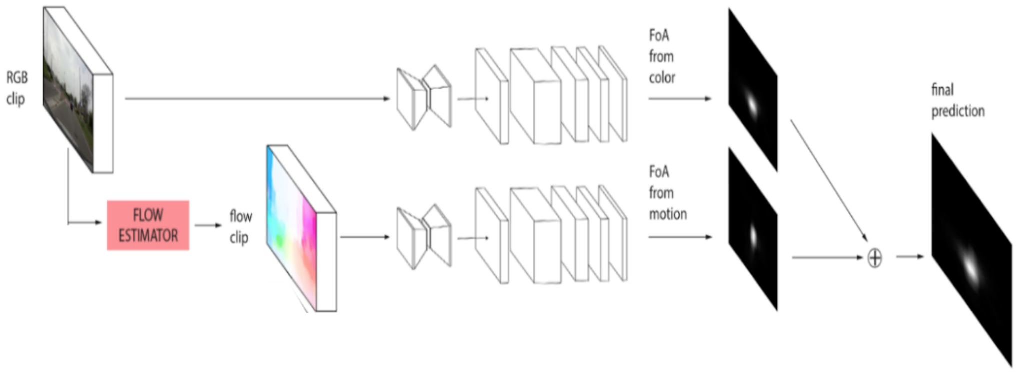


Figure 4.3: Multi-Branch DR(eye)VE Network Architecture (taken from [23])

The individual architecture of each saliency branch consists of two training streams, each taking a different input and producing their own outputs. Given an input tensor with N frames of size X_{orig}, Y_{orig}, the first stream simply takes this tensor, resizes the images to X_{res}, Y_{res}, and processes it with a 3D Convolutional network which produces a fixation map of size X_{res}, Y_{res}. This network is referred to as the COARSE module, and is strongly based on the C3D architecture of Tran et al. [31] - the exact details of this network are given in Table 4.4. The resulting fixation map is then upsampled to the original size X_{orig}, Y_{orig}, concatenated with the final frame of the input clip (the frame for which the prediction is being made), and further refined by a series of 2D convolutional layers (the REFINE module), producing a full-frame saliency map prediction for the input clip. The second stream, rather than resizing the input images, takes as input a randomly cropped section of the images, also of size X_{res}, Y_{res}. It then processes this random crop using the same COARSE network as the first stream, with shared weights, and directly outputs a saliency map prediction for the area which was chosen by the random crop. The full-frame and cropped losses are computed separately for the two streams, and summed to calculate the final loss for the branch.

The reasoning behind this unusual architecture, shown in Figure 4.4, is that drivers tend to exhibit a strong central bias when driving - that is, the majority of the time, they are looking directly in front of the car. If the network is trained only on full images, it learns to minimize loss by reproducing this central bias and often misses important features that would have attracted the attention of a human. This can stall the training process early on and prevent the network from being able to accurately

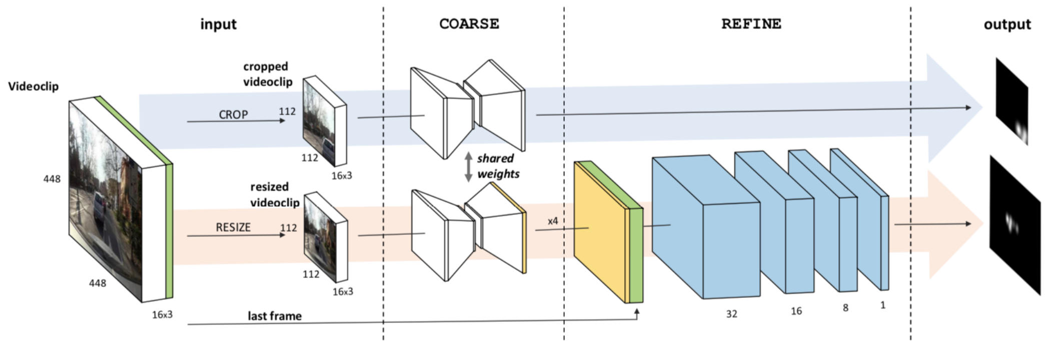


Figure 4.4: Individual Saliency Branch Architecture (taken from [23])

learn to predict gaze [23]. The random cropping strategy can be thought of as a form of data augmentation which forces the network to pay attention to features within the image rather than simply the spatial location of the labels; the majority of the cropped areas are outside of the center of the image, and by including the loss from these predictions in the branch's overall loss, the network is penalized for attempting to only predict in central regions. At test time, the cropped output is discarded, and only the full-frame prediction is used to ouput a predicted saliency map.

The loss function used by the DR(eye)VE network is the Kullback-Liebler Divergence between the ground truth (Y) and predicted ($\hat{Y}$) attention maps for all pixels i, defined as below (note that ϵ is simply a small-valued constant added for numerical stability):

$$D_{KL}(Y||\hat{Y}) = \sum_i Y(i) log \left(\epsilon + \frac{Y(i)}{\epsilon + \hat{Y}(i)} \right) \tag{4.3}$$

The individual branch loss for a single sample is therefore calculated as follows:

$$L_{branch}(Y, \hat{Y}) = D_{KL}(\phi(Y)||C(\phi(\hat{Y})) + D_{KL}((Y)||R(C(\psi(\hat{Y})), \hat{Y}) \tag{4.4}$$

In this calculation, C and R refer respectively to the COARSE and REFINE modules defined above, while ϕ and ψ represent the random cropping operation and resizing operations. The final loss of the network is simply the average of the losses from the two individual branches.

Intention-Branched DR(eye)VE Model

In this project, we propose an addition to the above model, which we refer to as the Intention-Branched DR(eye)VE Model. Drawing inspiration from the Conditional

Imitation Learning (CIL) model of Codevilla et al.[6], we hypothesize that giving the Gaze network access to a high-level signal that indicates the driver's intentions for a given input clip will lead to more accurate predictions about where the driver will look. This is a logical assumption given the top-down nature of the task; a driver intending to make a left turn at the upcoming intersection, for example, will likely spend more time inspecting the left lane for oncoming traffic or pedestrians. As this intention signal is already being provided for the autonomous driving agent which the Gaze network is intended to augment, it can be easily provided to the Gaze network as well.

The intention-branched model is trained in a similar manner to the CIL model, with each output branch trained only on samples labeled with the corresponding high-level intention. Given the lower number of samples labeled with Left, Right and Straight intentions in comparison to the Follow intention, we choose to bootstrap the individual branches as normal, using all training samples, and only begin branching the model during the fine-tuning process which trains both branches simultaneously. This decision is made to avoid overfitting to the small number of samples for these actions, allowing the convolutional networks to learn general features before specializing for individual high-level intentions.

Layer	Output Depth	Output Height	Output Width	Output Channels	Number of Parameters
Input	16	56	56	3	**0**
Conv3D (3x3x3)	16	56	56	64	64x((3x3x3)x3)+64 = **5,248**
Pool3D (1x2x2)	16	28	28	64	**0**
Conv3D (3x3x3)	16	28	28	128	128x((3x3x3)x64)+128 = **221,312**
Pool3D (2x2x2)	8	14	14	128	**0**
Conv3D (3x3x3)	8	14	14	256	256x((3x3x3)x128)+256 = **884,992**
Conv3D (3x3x3)	8	14	14	256	256x((3x3x3)x256)+256 = **1,769,728**
Pool3D (2x2x2)	4	7	7	256	**0**
Conv3D (3x3x3)	4	7	7	512	512x((3x3x3)x256)+512 = **3,539,456**
Conv3D (3x3x3)	4	7	7	512	512x((3x3x3)x512)+512 = **7,078,400**
Pool3D (4x1x1)	1	7	7	512	**0**
UpSample x 8	1	56	56	512	**0**
Total					**13,499,136**

Table 4.4: DR(eye)VE COARSE Network Architecture

Layer	Output Height	Output Width	Output Channels	Number of Parameters
Full Frame Prediction				
COARSE Model Output (Small)	56	56	512	**13,499,136**
Conv2D (3x3)	56	56	1	1 x ((3x3) x 512)) + 1 = **4,609**
UpSample x 8	224	224	1	**0**
Input (Full)	224	224	3	**0**
Concatenate	224	224	4	**0**
Conv2D (3x3)	224	224	32	32 x ((3x3) x 4)) + 32 = **1,184**
Conv2D (3x3)	224	224	16	16 x ((3x3) x 32)) + 16 = **4,624**
Conv2D (3x3)	224	224	8	8 x ((3x3) x 16)) + 8 = **1,160**
Conv2D (3x3)	224	224	1	1 x ((3x3) x 8)) + 1 = **73**
Crop Prediction				
COARSE Model Output (Cropped)	56	56	512	**13,499,136**
Conv2D (3x3)	56	56	1	1 x ((3x3) x 512)) + 1 = **4,609**
Total				**13,515,395**

Table 4.5: Individual Saliency Branch Architecture for DR(eye)VE Network

4.2 Driving Models

4.2.1 Conditional Imitation Learning

To train our imitation learning agents for autonomous driving, we start with the Conditional Imitation Learning (CIL) framework of Codevilla et al. [6]. The architecture of this model is quite straightforward in comparison to the Gaze models previously discussed. The network receives two inputs: an RGB camera image, and a float indicating the current speed of the vehicle. The RGB image is processed by a 2D Convolutional Network, while the speed input is processed by two fully-connected layers. These outputs are then concatenated and fed into one of five branch heads, selected based on the high-level command associated with the input frame, as shown in Figure 4.5. Each branch head is associated with one of four commands: Follow, Left, Right, and Straight, with the fifth branch used in the rare case that no high-level command is provided (this sometimes occurs for a small number of frames when the expert demonstrator has reached its target destination but the episode has not yet terminated). Each branch consists of two fully-connected layers, and outputs four floats representing the actions for the Steer, Throttle and Brake commands, as well as the predicted current speed of the vehicle.

The inclusion of the ’Speed’ output signal is not immediately clear, as simply predicting speed without any associated action does not actually affect the behaviour of the vehicle. While this information was not published in the official CIL paper, the authors, who are very active on GitHub and Discord, have explained that this output was included to help solve a particular issue which was observed in fully trained models; occasionally, after the vehicle has come to a stop for any reason, it remains still and will not start again. The speed prediction was introduced as a workaround to this issue - when actually driving a vehicle in CARLA, if the network outputs 0.0 for the throttle action but has a predicted speed above a certain threshold, the throttle is manually activated with a small value, giving the vehicle a small push forward which resolves the stalling problem and gets the agent to drive normally again.

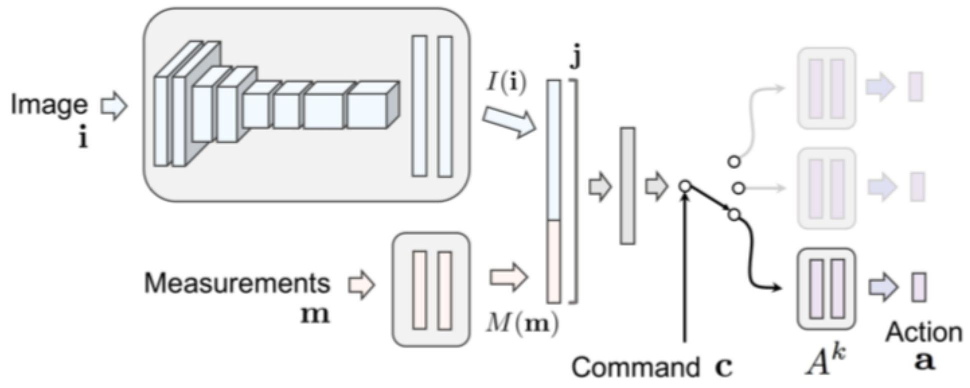


Figure 4.5: Conditional Imitation Learning Model Architecture (taken from [6])

The loss function used for training the network is a weighted combination of the mean-squared errors between each predicted control value (gas, brake, steer and speed) and its ground-truth label as given by the expert demonstrator. This can be calculated simply as follows:

$$\begin{aligned} L_{tot}(y, \hat{y}) = &\lambda_{gas}(y_{gas} - \hat{y}_{gas})^2 + \lambda_{brake}(y_{brake} - \hat{y}_{brake})^2 \\ &+ \lambda_{steer}(y_{steer} - \hat{y}_{steer})^2 + \lambda_{speed}(y_{speed} - \hat{y}_{speed})^2 \end{aligned} \tag{4.5}$$

The lambda values allow the user to configure the importance of each task, based on observation of the vehicle's driving behaviour, e.g. if the vehicle tends to drift off the road, the steering lambda may be increased to force the network to pay more attention to optimizing this control signal. The lambdas used in this implementation, as suggested by the authors of the network, are $\lambda_{gas} = 1.66, \lambda_{brake} = 1.66, \lambda_{steer} = 14.29, \lambda_{speed} = 0.00001$. The speed lambda is set close to zero as we do not actually use it to control the car's actions, only to restart the vehicle in the event of stalling as explained above.

4.2.2 Attention-Guided CIL

We experiment with two main methods of integrating the attention maps produced by the best-performing gaze prediction model, the Intention-Branched DR(eye)VE network, into this CIL architecture. The first method, visualized in Figure 4.6, simply replaces the input to the CIL network with one that has been masked by the output of the Gaze network. We test this model with hard, soft and baseline attention masking as defined in Section 3.2.3.

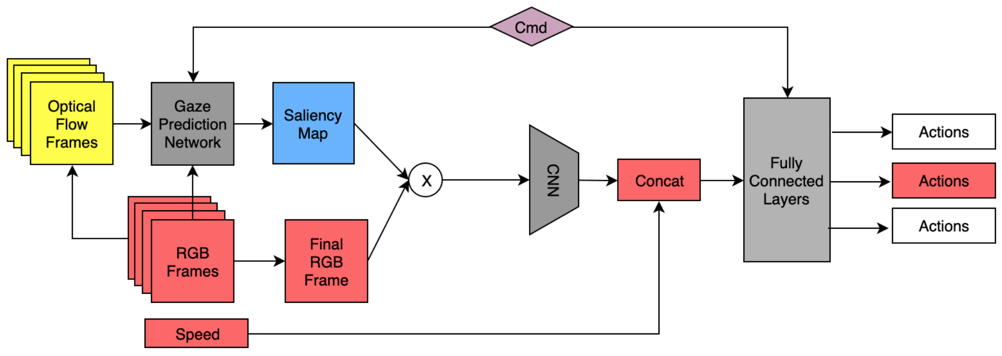


Figure 4.6: Single Branch Attention-Masked CIL Model

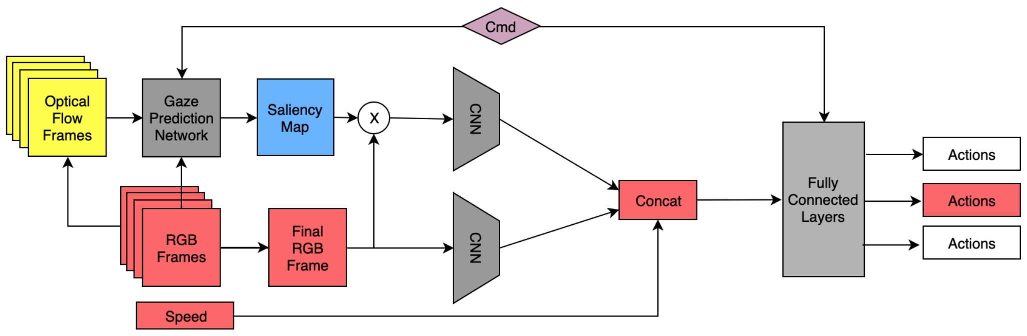


Figure 4.7: Dual Branch Attention-Masked CIL Model

The second method, visualized in Figure 4.7, uses a dual-branch architecture to process both the original images and the attention-masked images in parallel. The features of each network are then concatenated and processed by two fully-connected layers before being input to the separate branch heads as usual. This architecture was inspired by the AGIL network described in the Background chapter [37], which achieved improved performance on Atari games compared to a single-branch version without attention. Two main differences exist between our model and the AGIL model: first, we concatenate our branch outputs rather than averaging them, as we do not wish to lose any information in the averaging process. Second, we continue to branch our output heads as with the previously discussed CIL models, rather than directly predicting actions from the fully connected layers. This model is tested only with hard attention masks as the input to the second branch.

Chapter 5

Results and Evaluation

5.1 Gaze Networks

In this section, we provide details on the training, evaluation and observed behaviours of each gaze network, followed by a side-by-side comparison of all networks. Due to the large number of deep models which we train from scratch on our dataset and the amount of training time required for each, we are not able to perform an extensive hyperparameter search for every network. In general, we begin with the parameters suggested by the authors of the papers from which these networks were taken, and experiment with a few alternative values to optimize performance on our data. All networks are trained on episodes 1 to 48 from the Gaze dataset and tested on episodes 49 to 54. Our validation set is composed of 500 frames from the middle of each episode, from frame 2500 to 3000. This ensures a fair balance of different drivers' behaviours and different environmental conditions between the training and validation sets. Please refer to Chapter 5 for the specifics of each network's individual architecture and loss function.

5.1.1 DeepGaze II

As noted in Section 4.1.1, we use a fully pre-trained version of DeepGaze II to provide a baseline for bottom-up prediction performance on our test set. As such, we did not need to do any training or hyperparameter tuning for this network, and simply used the pre-trained version to make predictions for the test set episodes. Per our discussion on saliency model evaluation in Section 2.2.3, we evaluate model predictions based on two metrics: Kullback-Liebler Divergence (KLD) and Pearson's Correlation Coefficient (CC). The predicted attention maps from Deep Gaze II had an average KLD of **1.287** and a CC of **0.511** when compared to the ground truth attention maps for the six test set episodes.

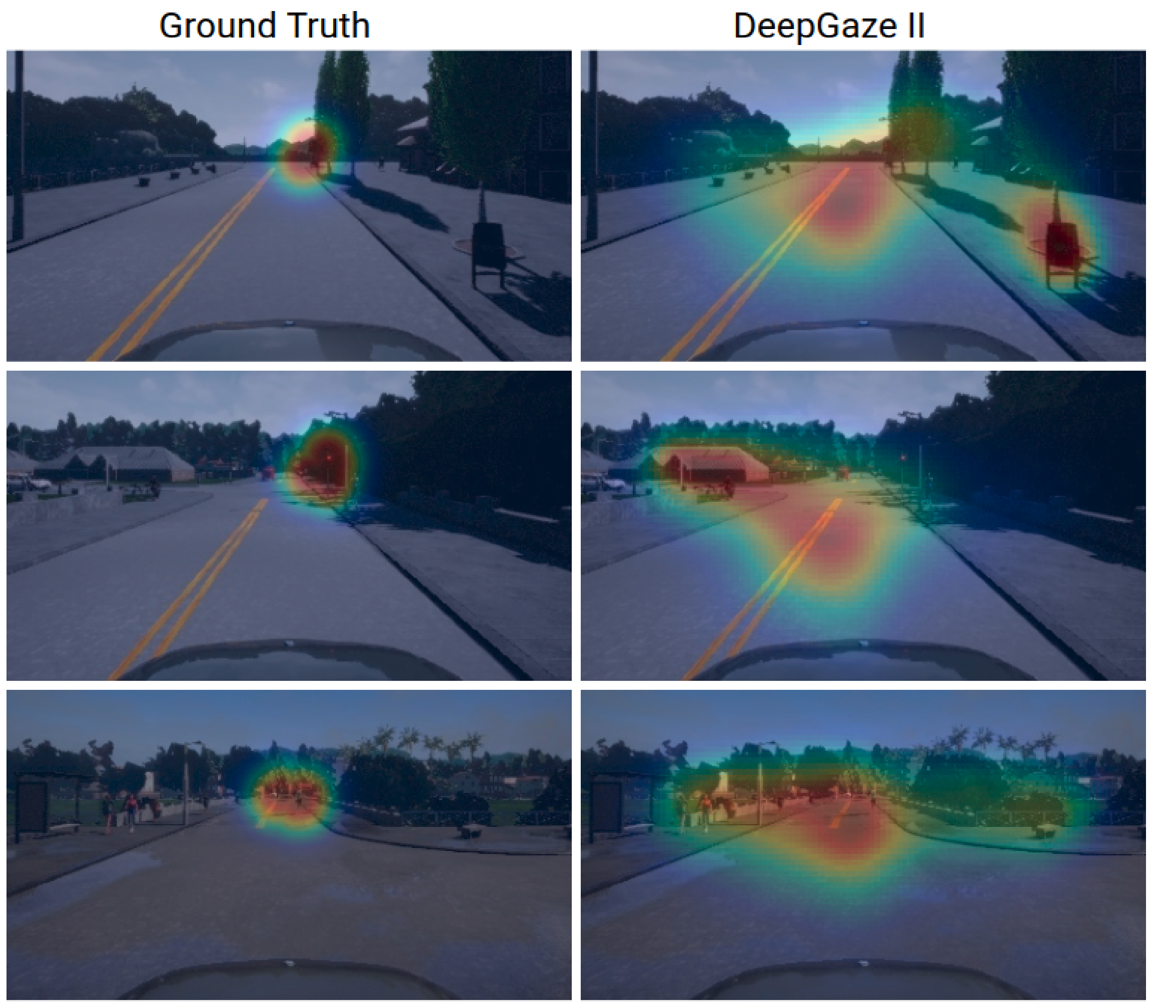


Figure 5.1: Sample test set predictions from the pre-trained Deep Gaze II Model. Note how the model predicts fixations for unimportant objects such as the mailbox in the top figure and the house in the middle figure, showing its lack of task-specific knowledge.

The influence of the pre-trained VGG-16 model which performs feature extraction for DeepGaze II can be clearly seen in the outputs for this network, several of which are shown in Figure 5.1. The model does a very good job of identifying high-level objects in the image such as pedestrians, other vehicles and traffic lights. However, since it was not trained specifically for the driving task, it pays equal attention to objects which are less relevant to driving, such as mailboxes, fences, or house windows. These results highlight the limited use of generalized bottom-up saliency networks for specific task-oriented settings.

5.1.2 MLNet

Training

The MLNet model is trained using the stochastic gradient descent algorithm with a Nesterov momentum of 0.9 and a Dropout value of 0.5 for the single dropout layer present in the network. The initial learning rate was set to $\eta = 10^{-3}$, as in the official code for the MLNet model at `https://github.com/marcellacornia/mlnet`[7]. We then varied the learning rate, testing values of $\eta = \{10^{-3}, 10^{-2}, 10^{-1}\}$ obtaining the results displayed in Figure 5.2. We found that a significantly higher learning rate of $\eta = 10^{-1}$ produced the best validation set results for this network - any further increase in the learning rate led to divergences in the loss while training. As the network is quite large, we were limited to a maximum batch size of 4; larger batches resulted in crashes due to out of memory errors. To minimize the amount of time spent training, we use an Early Stopping callback which halts the training process if 5 consecutive epochs show no improvement in the validation set accuracy. This typically occurs after 20 to 25 epochs, which corresponds to about 3 hours of training on an NVIDIA GTX 1070 graphics card.

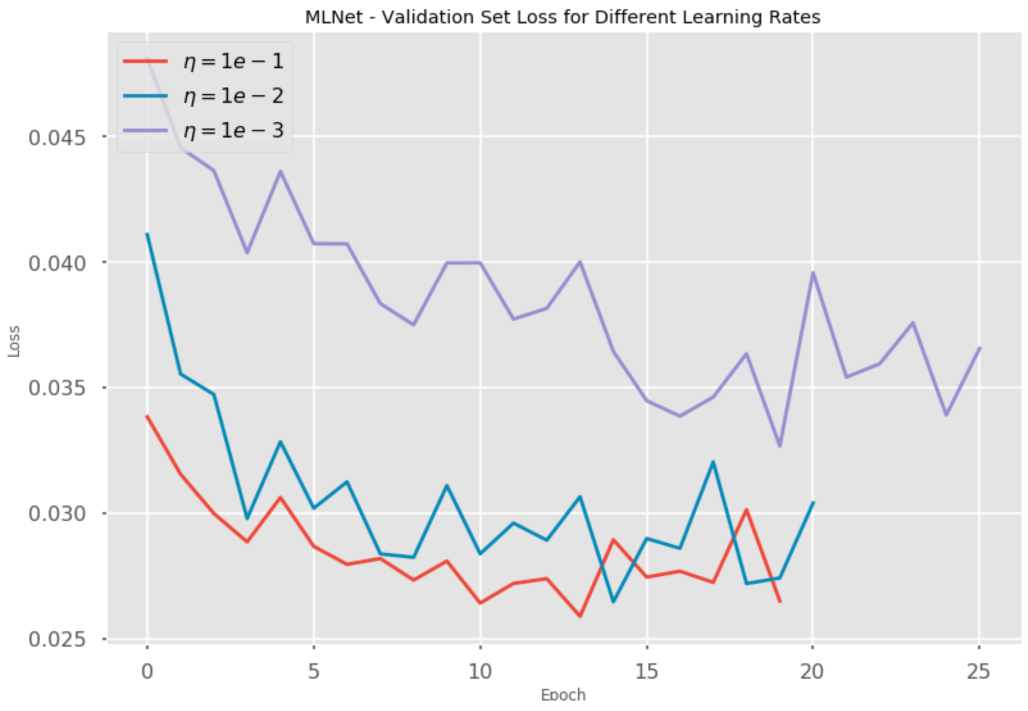


Figure 5.2: Comparison of 3 tested learning rates for the Multi-Level Net, with $\eta = 10^{-1}$ providing the best results.

Results and Behaviour

Using the above parameters, the fully trained model which achieved the lowest validation error was used to make predictions for all test set episodes, achieving an average KLD of **1.222** and an average CC of **0.553**, only slightly outperforming the DeepGaze II model despite being fully retrained on our data. This is likely due to the fact that, as with DeepGaze II, the feature extraction in MLNet is performed entirely by a frozen VGG-16 model, with only a small number of convolutional layers actually being trained by the new dataset. This does not appear to be sufficient for adapting to the task of top-down prediction, as the network displays the same behaviour of focusing on objects and image regions which have no relevance to driving (e.g. the Coca-Cola machine in the center prediction of Figure 5.3). One notable difference between the outputs of these two bottom-up models is that MLNet tends to spread out its predicted areas more, with a few areas of high fixation probability and a large number of regions with low fixation probability, as is clearly visible in the top and bottom predicted maps in Figure 5.3. DeepGaze II, on the other hand, typically predicts fixations with high probability in one or two focused regions and zero probability in all other areas. This can likely be attributed to the different feature maps which were extracted from VGG-16 between the two architectures. As described in Section 4.1.2, MLNet extracts features from 3 different convolutional blocks, mixing high and low-level features, which may explain why more attention is paid to low-level features such as the curving of the road while DeepGaze II tends to focus exclusively on high-level recognizable objects.

5.1.3 RMDN

Training

As suggested by the authors in [2], we train RMDN using the RMSProp algorithm and a value of 0.5 for the network's single Dropout layer following the LSTM module. Each model was trained for 30 epochs, taking roughly 4 hours to train fully on our setup. Once again, due to the very large size of the network, our batch size was limited to a maximum of 4 to avoid encountering out-of-memory issues. We experimented with learning rates of $\eta = \{3 * 10^{-2}, 3 * 10^{-3}, 3 * 10^{-4}\}$, with the best validation set results obtained by $\eta = 3 * 10^{-3}$. We then fixed the learning rate to this value and varied the number of Gaussian mixture components produced by the LSTM, testing values of C = {5, 20, 40}. The number of components used did not seem to affect performance significantly after the first 10 epochs of training, with all models having equal loss on the validation set. Figure 5.4 shows the training history for these hyperparameter variations.

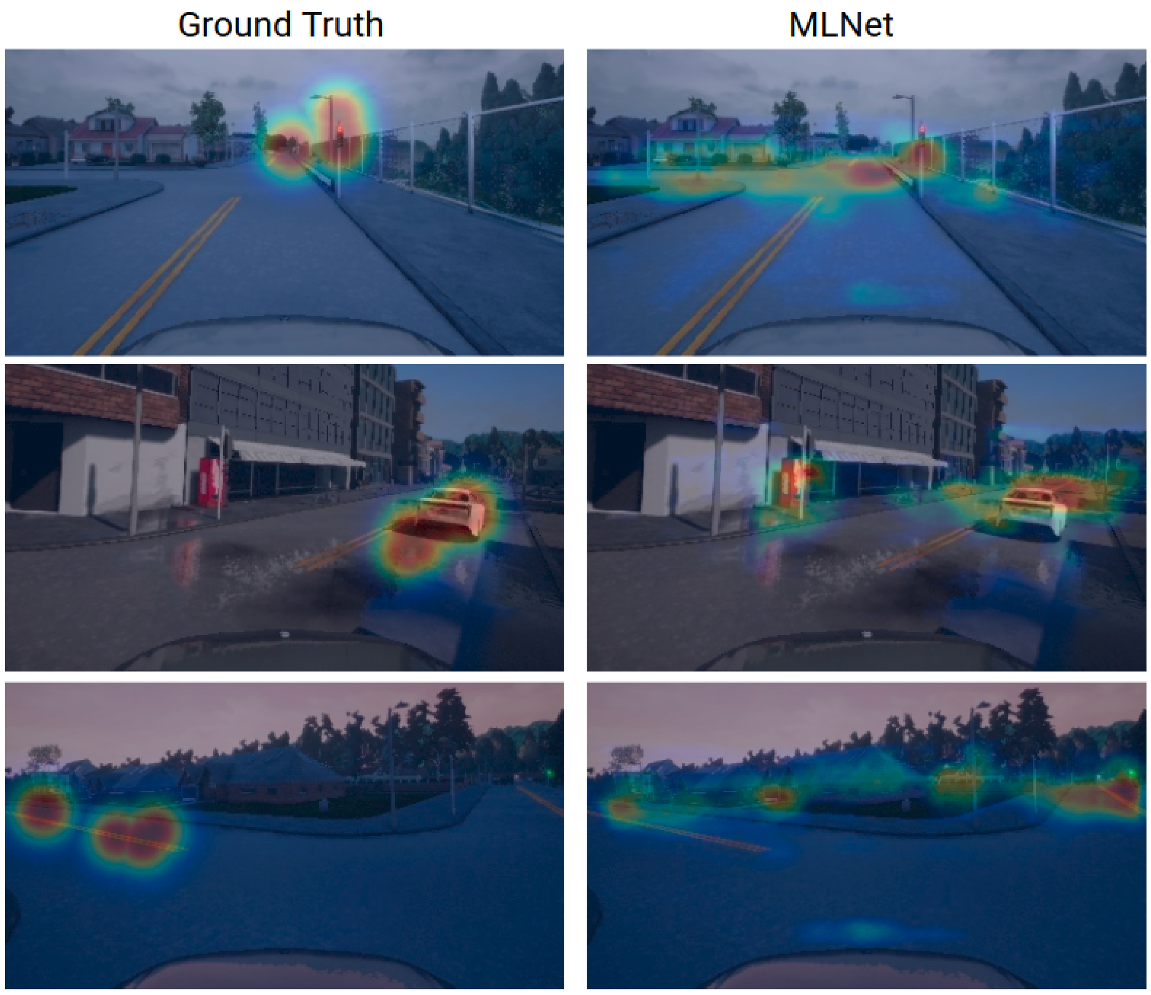


Figure 5.3: Sample test set predictions from the fully trained MLNet Model. Similar to DeepGaze II, MLNet is distracted by objects such as the Coke machine in the middle image, lacking the ability to identify which objects are important for the driving task or predict where a human would look while executing certain actions, such as the left turn in the bottom image where the human uses lane markings to guide their movement.

Results and Behaviour

The fully trained RMDN model received an average KLD of **1.029** and an average CC of **0.600** on the test set, outperforming both MLNet and DeepGaze II. Upon visually inspecting the predictions, however, we observed that the model appears to have strongly overfit to the mean of the dataset, and overall does a very poor job of tracking important objects in a scene. Clear examples of this behaviour can be seen in Figure 5.5, where the model twice ignores pedestrians on the road very near the vehicle. Unlike the previously evaluated networks, RMDN has no built-in mechanism for avoiding bias in its predictions, and thus is not penalized for reducing its overall loss by predicting one or two overlapping, highly-weighted central Gaussians for every frame, with all other mixture components having near-zero weights in the

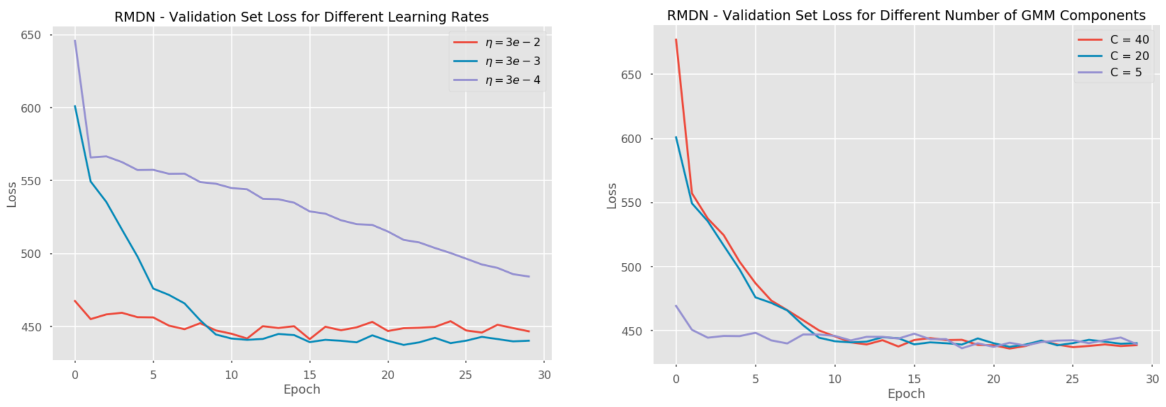


Figure 5.4: Comparison of validation loss for (left) 3 different learning rates and (right) 3 different numbers of GMM components for the Recurrent Mixture Density Network

mixture. The recurrent nature of the network likely worsens the problem, with short-term objects such as pedestrians being ignored in favor of the long-term near-certainty that the driver will return their gaze to the center of the road. To confirm that the model is indeed overfitting to the mean, we computed the average KLD between all RMDN predictions and the mean fixation map of the training dataset, obtaining a value of **0.179**, significantly better than the average performance on the test set. We conclude that RMDN's capabilities are severely limited for this task, despite its reasonably good test set performance.

5.1.4 DR(eye)VE

Training

The DR(eye)VE model is trained using the Adam optimization algorithm, with $\beta_1 = 0.9, \beta_2 = 0.99, \epsilon = 10^{-8}$. We use a batch size of 8 when training individual branches, reducing to 4 during the fine-tuning stage as the size of the network increases. Each branch is trained for 40 epochs, and the checkpoints with the lowest validation set loss are then used to train the combined model for an additional 20 epochs. Training this model in its entirety takes significantly longer than the previously discussed models, as each branch must be individually trained before the fine-tuning phase which trains both branches simultaneously. The total training time comes to about 18 hours on our setup, with each individual branch requiring 6 hours to train, and the fine-tuning stage taking an additional 6 hours. As such, to avoid spending unreasonable amounts of time testing hyperparameter configurations for all training stages, we tune our learning rate only on the Image branch, under the assumption that these parameters will provide good performance for the Flow branch and fine-tuning stage given the identical network structures and training data.

We experiment with learning rates of $\eta = \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. As discussed in

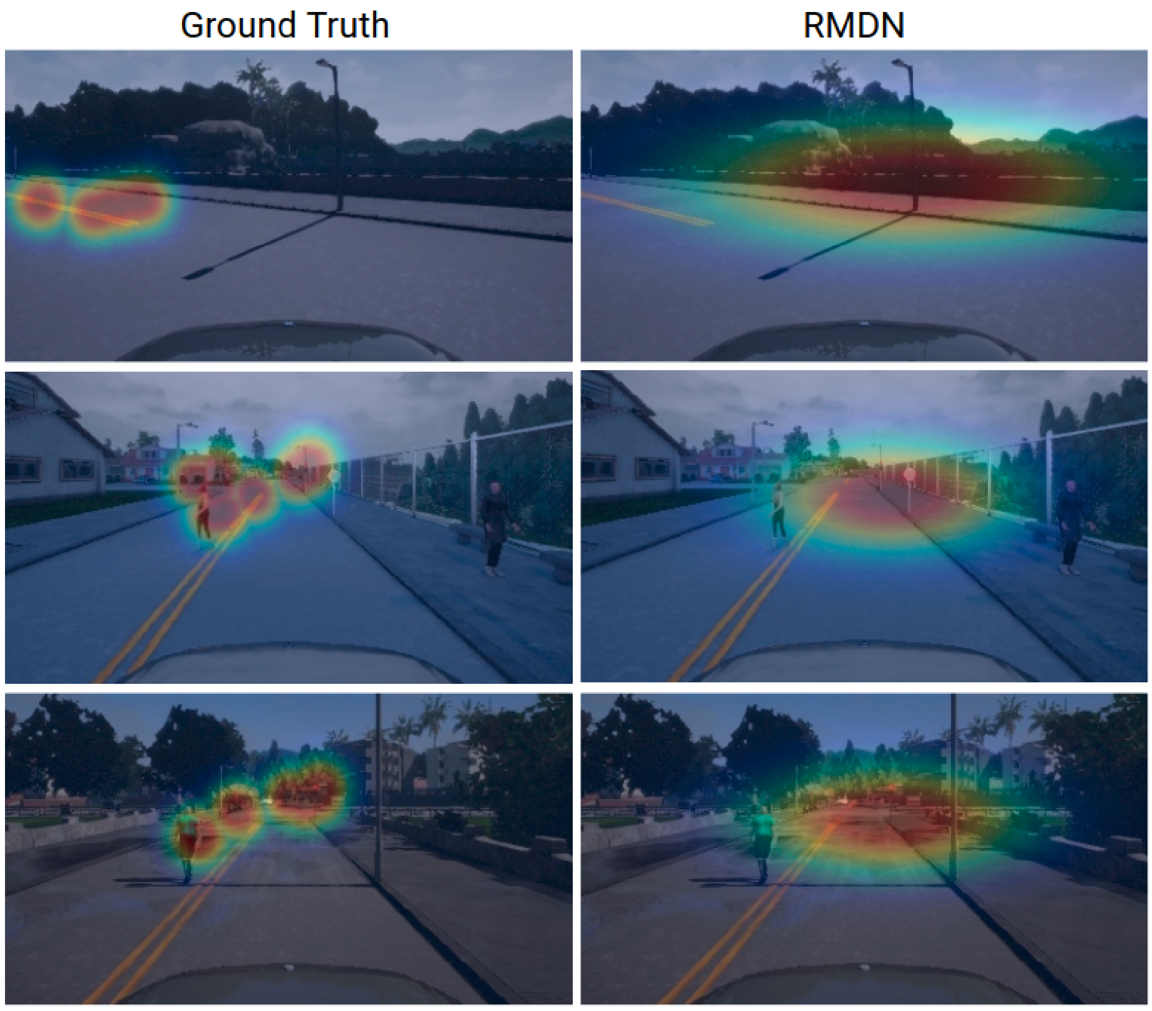
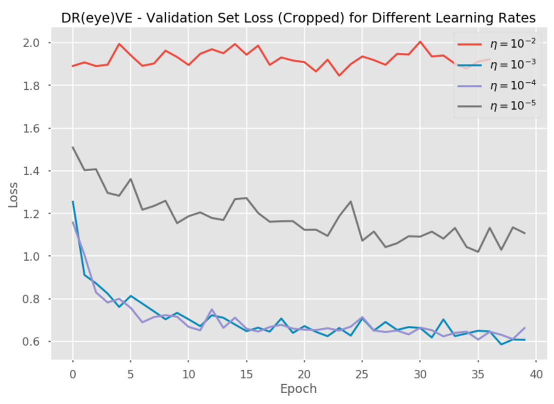


Figure 5.5: Sample test set predictions from the fully trained RMDN Model. Clearly having overfit to the mean of the dataset, RMDN gives very inflexible and uninformative predictions, missing key features such as the pedestrians in the middle and bottom images.

Section 4.1.4, the loss function used to train individual saliency branches combines the Kullback-Liebler Divergence for both the cropped and full-frame (fine) predictions of the branch. While training, these individual losses are recorded in addition to the total loss. The results for each of these losses under different learning rates are displayed in Figure 5.6, while the combined loss is depicted in Figure 5.7. The optimal learning rate was found to be $\eta = 10^{-3}$.

We also examined the effect of random cropping, the strategy described in Section 4.1.4 to avoid overfitting to the central bias in the attention maps. Two networks were trained with identical hyperparameters, with one using the random cropping strategy while the other took a fixed crop from the center of each training image. We compare the full-frame prediction loss of the two networks in Figure 5.8, observing that the random cropping strategy indeed results in better validation set



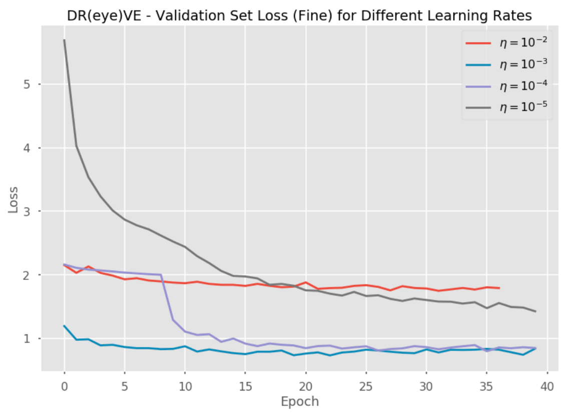


Figure 5.6: Comparison for different learning rates of (left) cropped prediction loss and (right) full-frame prediction loss for the DR(eye)VE network's Image branch

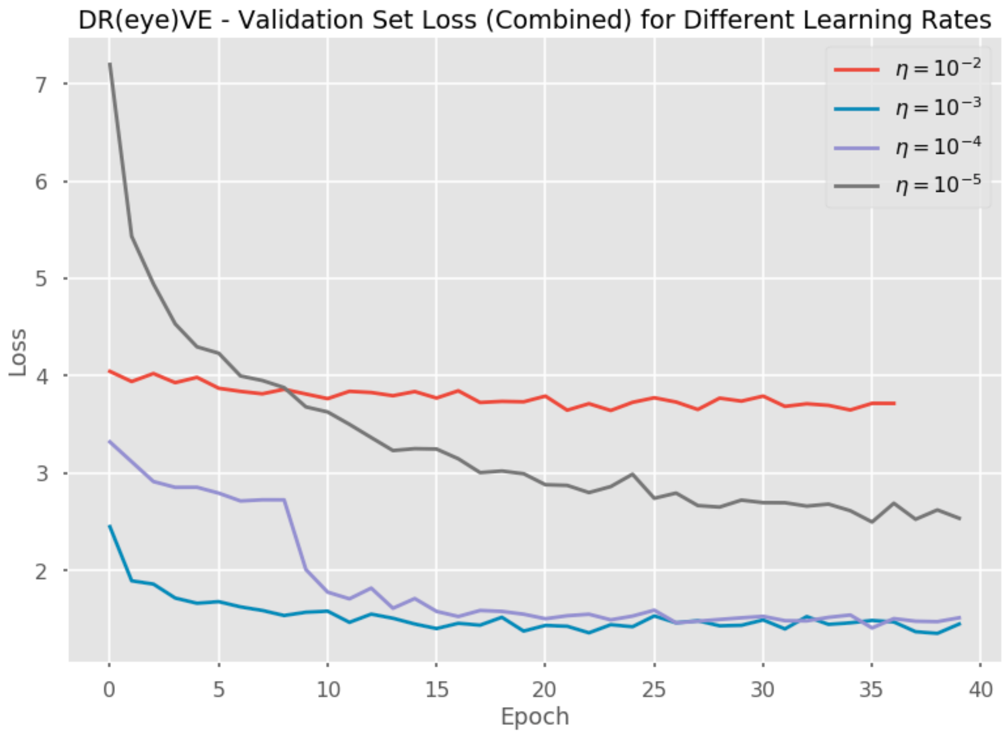


Figure 5.7: Comparison of the total validation set loss of the DR(eye)VE Network's Image Branch for 4 different learning rates

performance. Note that we only consider the full-frame loss, as the cropped loss is no longer a fair measurement when the two networks take different crops, and this change is also reflected in the total loss, which combines the two terms. Indeed, since the central crop model has an easier task, its combined loss values are actually lower than those of the random crop model. This does not accurately reflect its ability to predict gaze, as is made clear by evaluation on the test set: the random crop model has an average KLD of **0.654**, while the central crop model has an average KLD of **0.740**.

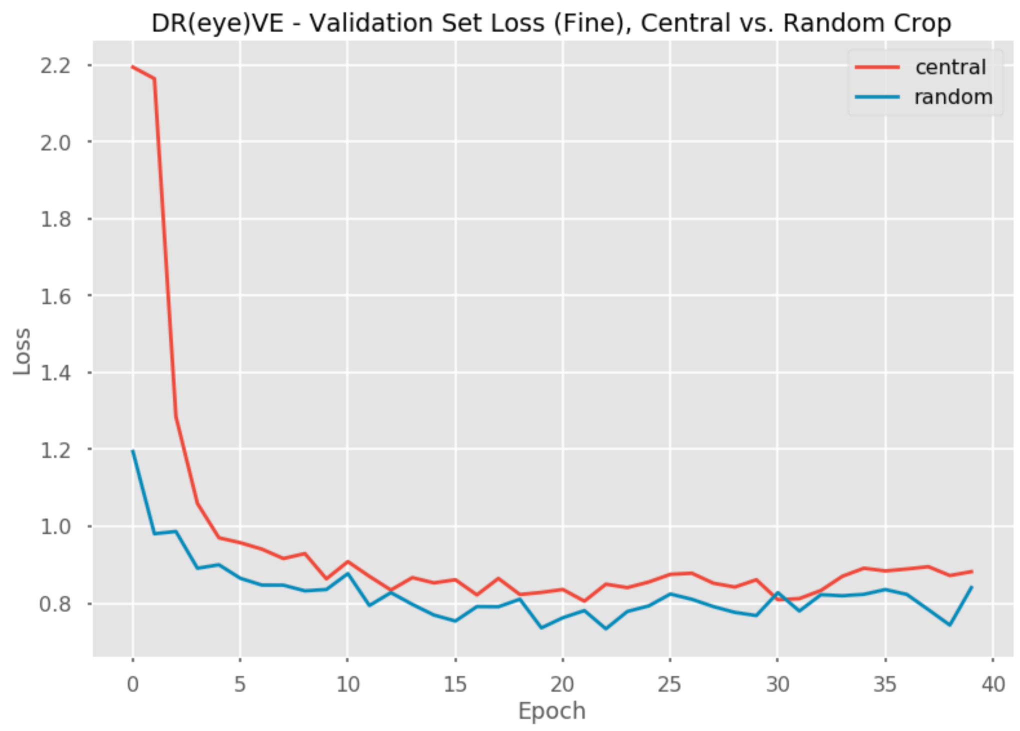


Figure 5.8: Comparison of full-frame validation set loss for the Random and Central cropping strategies

Results and Behaviour

The DR(eye)VE model significantly outperformed all other models on the test set, achieving an average KLD of **0.654** and a CC of **0.761**. The model appears to have gained a solid understanding of human attention dynamics, consistently paying attention to other vehicles, traffic lights and nearby pedestrians while ignoring objects which distracted the bottom-up networks such as Coca-Cola machines, trees or buildings which were irrelevant to the driving task. Sample outputs are displayed in Figure 5.9. The top image shows the very commonly observed human behaviour of using lane markings to guide their turns, with the DR(eye)VE network clearly having learned to predict this behaviour. The central image shows the network splitting its attention between the traffic light and the pedestrian crossing the road in front of it, while in the bottom image attention is split between the vehicle directly in front of the car and the upcoming road sign. As with RMDN, we also compared the model's predictions to the mean fixation map of the training dataset, obtaining a KLD of **0.644**, roughly equal to the test set score. This indicates that the model has not overfit to the mean in the same manner as RMDN, and its high scores on the test set therefore indicate an actual understanding of the nature and dynamics of the task. A driving sequence blended with this model's predictions can be found at `https://www.youtube.com/watch?v=A2cy70a5xtg`. The same sequence is also shown using hard attention masking rather than blending at `https://www.youtube.com/watch?v=n4PdE_oEqgE`, in order to easily see what information is being kept and removed by the hard masking process.

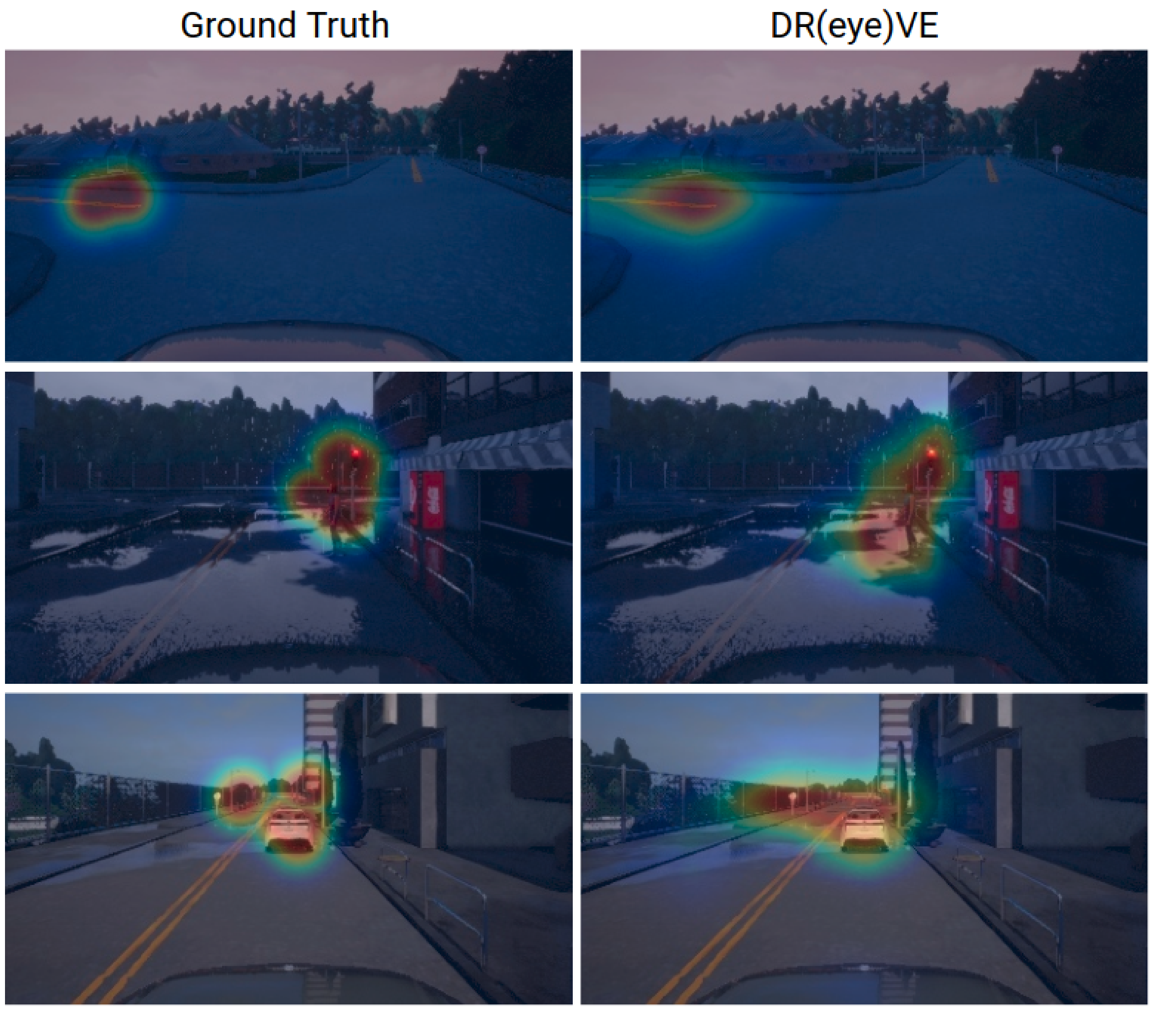


Figure 5.9: Sample test set predictions from the fully trained DR(eye)VE Model. This model shows a much greater understanding of the driving task, correctly fixating on lane markings when turns are being executed and showing the ability to split its attention between multiple important objects in a scene, such as the traffic light and pedestrian in the center image.

Individual Branch Contribution

To see the effect of the DR(eye)VE model's multi-branch architecture, we compare the performance of the overall network with the individual performances of its two branches. This is easily done by extracting the individual predictions of the two branches from the model before they are combined into the final prediction. The results are summarized in Table 5.1. We observe that the Image branch, on average, outperforms the Flow branch in both the KLD and CC metrics, but their combined predictions result in significantly improved performance for the full

DR(eye)VE model. This strongly indicates that the two branches are capturing different sets of important information, and each providing a unique benefit to the overall model, validating the multi-branch design.

Configuration	**Kullback-Liebler Divergence (↓)**	**Correlation Coefficient (↑)**
Image Branch	0.785	0.725
Flow Branch	0.839	0.705
Image + Flow	**0.654**	**0.761**

Table 5.1: Comparison of individual and combined branch performance for the fully trained DR(eye)VE model

Intention-Branched DR(eye)VE

As described in Section 4.1.4, we have also implemented the Intention-Branched DR(eye)VE model, which takes into account the high-level command currently being executed by the driver when making saliency predictions. We train the Intention-Branched model using the same parameters as the non-branched model, and compare the performance of the two on the test set. Ultimately, we found that the Intention-Branched model outperformed the non-branched model, but only by a very small margin, obtaining a KLD of **0.642** in comparison to 0.654, and a CC of **0.763** compared to 0.761. This slight increase is in line with our expectations, as the non-branched model already does quite a good job of predicting gaze without the high-level command, and it was unlikely that this extension would drastically improve performance. To see where intention branching provided the most benefit, we also compared the average KLD between the two models for the ”Lane Follow” high level command, which comprises the large majority (88.5%) of the dataset, and the ”Left”, ”Right”, and ”Straight” high level commands, which are significantly less represented. As expected, we found that the underrepresented commands benefit more from the branching process, with an average KLD of **0.660** and CC of **0.758** for the Intention-Branched model, compared to a KLD of **0.684** and a CC of **0.743** for the non-branched model. These improvements are still very small, but account for the majority of the overall performance increase between the two models, with the KLD and CC of the ”Lane Follow” command for the two models being almost exactly the same. One example of an improved prediction from the Intention-Branched model is shown in Figure 5.10.

Effect of Environment

We compare the performance of the DR(eye)VE model for the six individual test episodes, each of which has a different environmental setting in the CARLA environment. The results are displayed in Figure 5.11. The model performs best in the Clou-

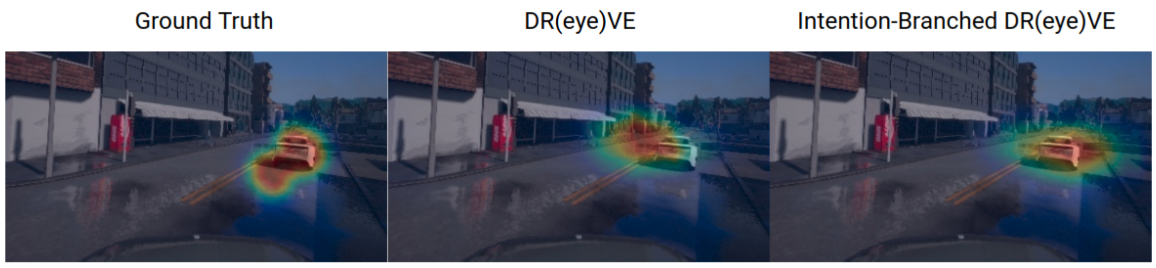


Figure 5.10: Example of an improvement in saliency prediction by the Intention-Branched DR(eye)VE model

dyNoon and WetNoon settings, approximately equal in the ClearNoon, ClearSunset and HardRainyNoon settings, and worst in the RainySunset setting. In general, however, the difference in performance between all settings is quite small, and it does not appear that the model is strongly affected by these environmental conditions.

Active vs. Traffic Sequences

As described in Section 3.1.6, a significant portion of the collected driving data consists of periods where the driver is waiting at a traffic light. We hand-labeled each frame to distinguish between ”active” frames, where the driver must constantly pay attention to avoid a collision, and ”traffic” frames, where minimal attention needs to be paid. Our assumption was that Gaze Prediction models would perform better for active sequences, as humans would be far less likely to let their attention wander in an unpredictable manner for these frames. We confirm this assumption by comparing the KLD and CC of the DR(eye)VE model on the two groups of frames. For active sequences, the model scored an average KLD of **0.600** and CC of **0.787**, while for traffic sequences the results are noticeably worse, with an average KLD of **0.749** and CC of **0.723**. Since we are not overly concerned with the model's predictions while sitting still in traffic (as no driving actions are executed in these scenarios), this is quite an encouraging result, showing that the model performs even better than its average test set metrics indicate for the sequences which are most important to us.

Failure Cases

While the DR(eye)VE model outperforms all other tested models, there is still room for improvement. Figure 5.12 displays some failure cases in which the DR(eye)VE model fails to predict the correct image regions or misses important objects which the human paid attention to (i.e. pedestrians). These images were selected from the set of 'active' driving frames with the highest individual KLDs in comparison to the ground truth fixation maps.

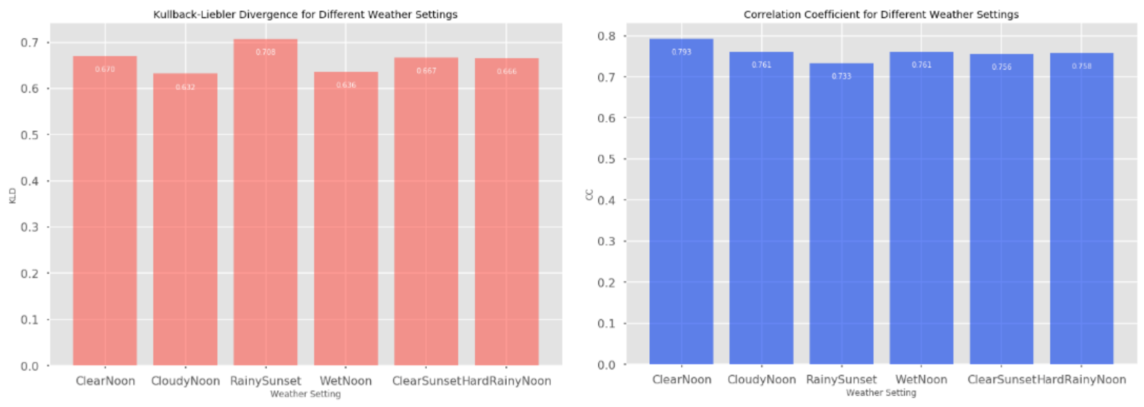


Figure 5.11: KLD (left) and CC (right) on different weather settings for the DR(eye)VE model

5.1.5 Side-by-Side Comparison

Here we provide a summary of the test set performance of all Gaze Prediction models, side by side for easy comparison. Table 5.2 shows the average KLD and CC of each model, while Figure 5.13 shows a set of 5 driving images and the predicted attention map from each model. These images have been selected to showcase the major differences in behaviour between the models; a longer driving sequence showing the behaviour of each model side by side is made available at `https://www.youtube.com/watch?v=CU-K_NVssC0&`. We do not show predictions for both the DR(eye)VE and Intention-Branched DR(eye)VE models because, as previously discussed, they are extremely similar.

Network	Kullback-Liebler Divergence (↓)	Correlation Coefficient (↑)
Deep Gaze II[18]	1.287	0.511
MLNet[7]	1.222	0.553
RMDN[2]	1.029	0.600
DR(eye)VE[23]	0.654	0.761
Intention-Branched DR(eye)VE (ours)	**0.642**	**0.763**

Table 5.2: Test Set Performance Comparison for all Gaze Prediction Networks

5.2 Driving Networks

Using the outputs of the Intention-Branched DR(eye)VE model, we applied three types of attention masking to our Driving Dataset frames, as described and visualized in Section 3.2.3: hard masking, soft masking, and baseline masking. We then trained

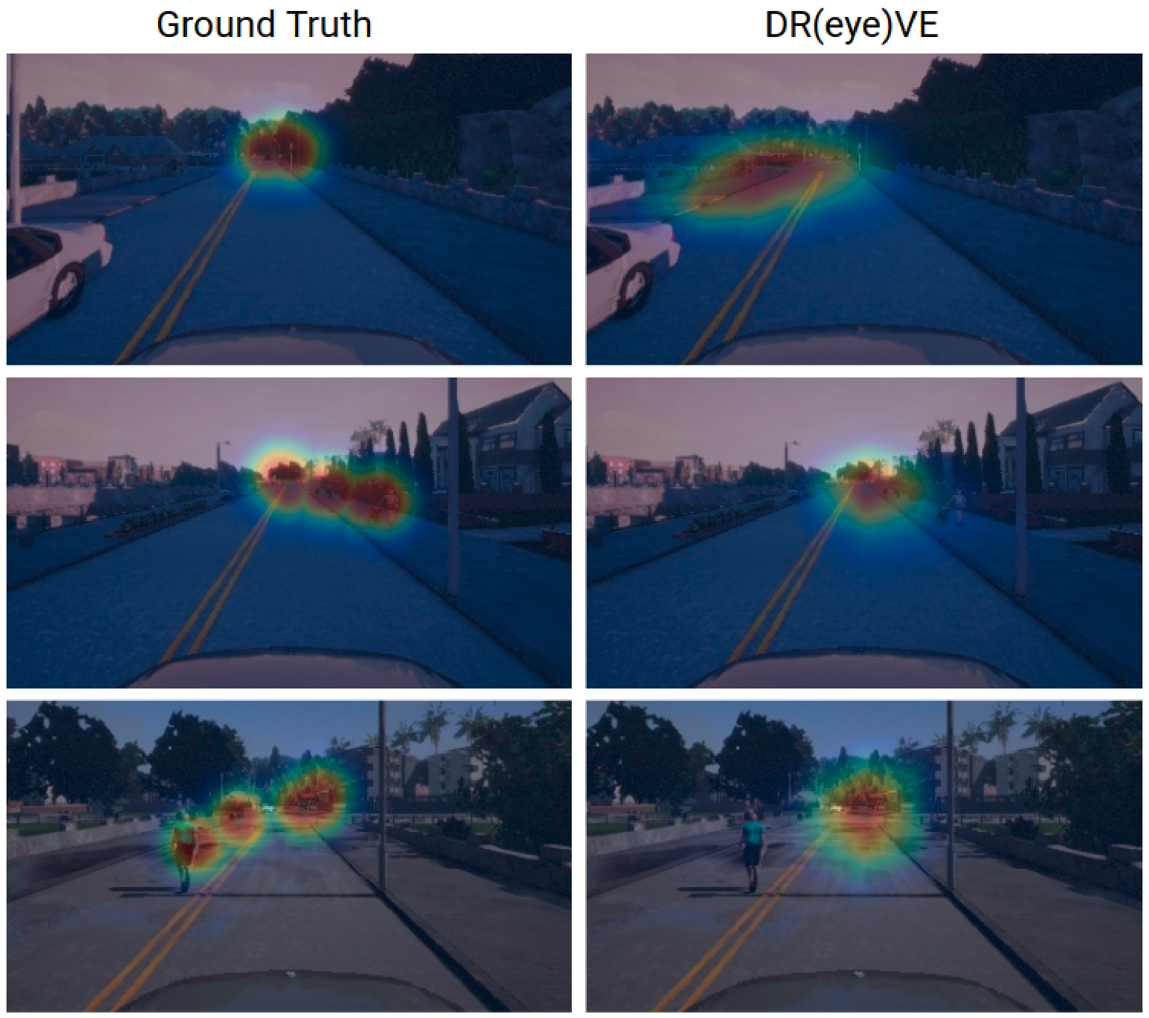


Figure 5.12: Some failure cases for the DR(eye)VE model. In the top image, the model is looking behind the oncoming car rather than at the road ahead of it, while in the center and bottom images it ignores pedestrians which the human driver chose to look at.

and evaluated a total of five different autonomous driving agents on these masked images. Four of these agents use the Conditional Imitation Learning architecture of Codevilla et. al[6]; one of these agents is trained using raw, unaltered images, one on hard-masked images, one on soft-masked images, and one on baseline-masked images. We refer to these agents henceforth as the raw, hard, soft, and baseline agents respectively. Our fifth agent uses a dual-branch architecture reminiscent of the AGIL network of Zhang et al.[37], with the top branch receiving raw images as input and the bottom branch receiving hard-masked images. This agent is referred to as the dual-branch agent. Figure 5.14 shows an image from the training set and its appearance with each type of masking applied, and describes which masks are used to train each agent. To ensure a fair comparison between models, all agents are trained with the same parameters, for an equal number of steps and using the same loss function.

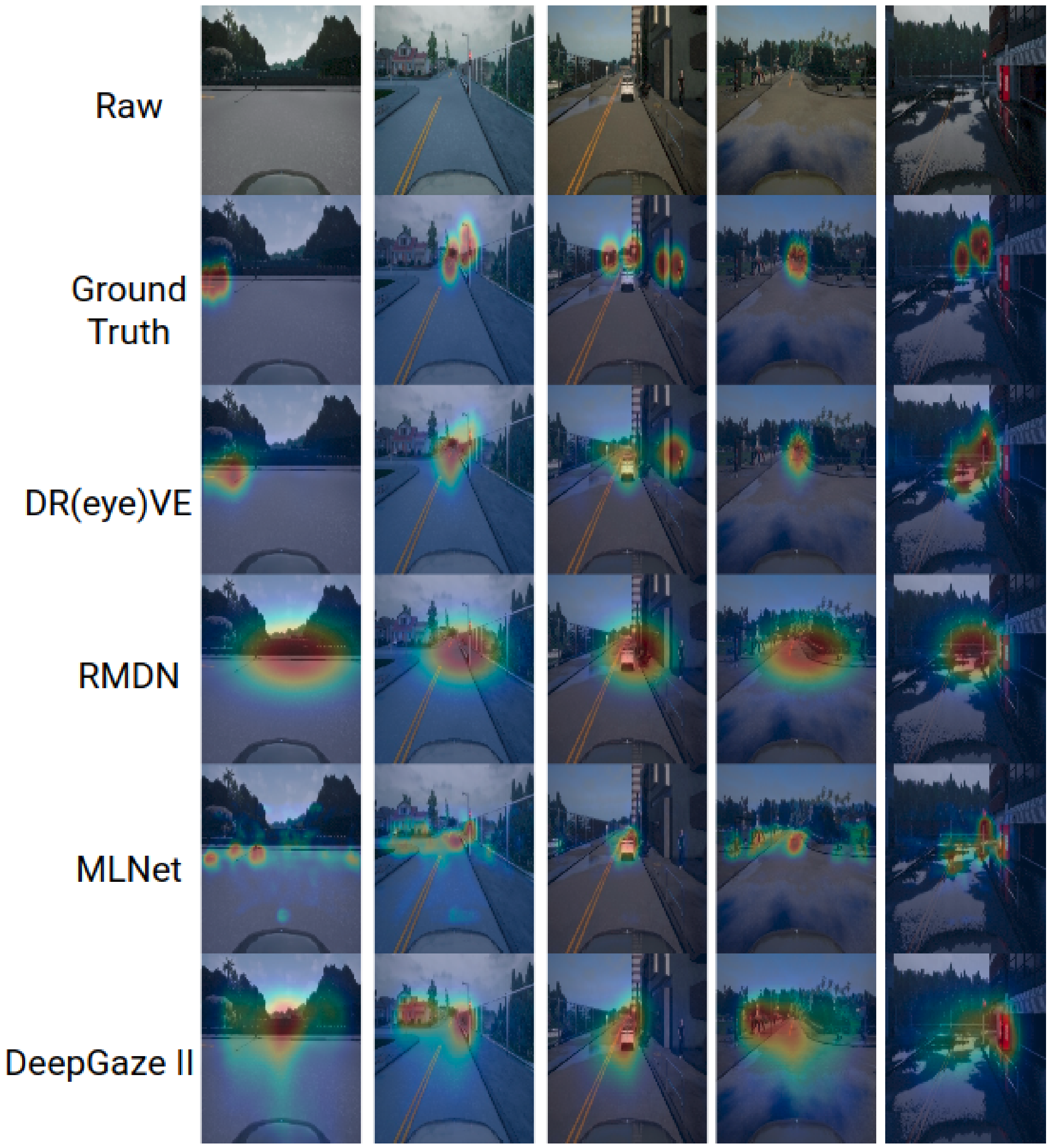


Figure 5.13: Side-by-side comparison of all Gaze Model predictions for 5 driving frames. We can see that the bottom-up DeepGaze II and MLNet models are easily distracted by objects which are not relevant to the driving task, such as the Coke machine in the fifth column and the house in the second column. The RMDN model is very inflexible, always making predictions in the central region of the image. The DR(eye)VE model is clearly superior, closely matching the ground truth attention maps and focusing only on image regions which are relevant to the driving task, such as the lane markings in the first column, the pedestrian and leading vehicle in the third column, and the traffic light in the fifth column.

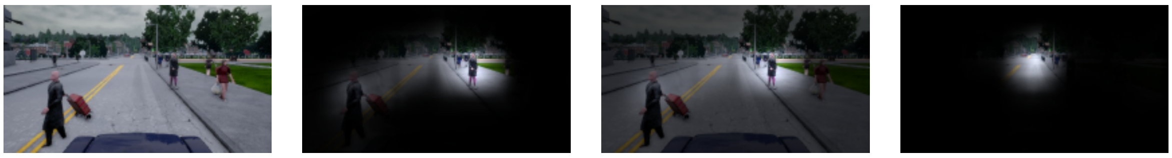


Figure 5.14: We train autonomous agents on driving images masked by the attention maps produced by our gaze prediction network. The raw agent is trained on unmasked images (left), the hard agent on hard-masked images (second from left), the soft agent on soft-masked images, (second from right), and the baseline agent on baseline-masked images (right). The dual-branch agent is trained on both the unmasked and hard-masked images.

The imitation learning agents are trained using the Adam optimization algorithm with $\beta_1 = 0.7, \beta_2 = 0.85$. We use the following hyperparameters recommended by the authors for the same task in [6]: batch size of 120, learning rate of 0.0001, and the λ values specified in Section 4.2.1 for the multi-task training loss (Equation 4.5). During training, we randomly augment images in the following ways and with the given probabilities: Gaussian blurring (9%), Gaussian noise (9%), pixelwise dropout (30%), brightness adjustment (30%), saturation adjustment (30%), and contrast adjustment (30%). Each agent is trained for 300,000 training steps, taking approximately 24 hours to train fully on our setup. We also encountered a bug in which model predictions would occasionally skyrocket, predicting values in the range of millions instead of the 0-1 range, despite individual batch losses appearing to be unaffected. Whenever this occurred, we were forced to abort training and restart the process from the most recent uncorrupted model checkpoint.

Due to time constraints and the extensive training time taken by these networks, we were not able to perform hyperparameter tuning for our autonomous driving agents. We do not believe that this significantly affected the outcome of our experiment for the following two reasons: First, we can assume that the parameters used by [6] will be close or identical to the ideal parameters for our network due to the high similarity in terms of model architecture and training/testing environment. Second, our primary interest is to see how these agents perform in comparison to each other, rather than to maximize their overall performance as much as possible. Since they are all trained using the same parameters, we are still able to accurately judge their relative performance and thus see the effect of our applied attention masking, which is the ultimate goal of this project. Because we do not perform tuning based on validation set performance, we do not require a validation set, and instead split our dataset simply into 80% training frames and 20% testing frames, with the training set containing only frames from Town 1 and the testing frames containing a mixture of frames from Town 1 and Town 2.

We evaluate each agent's performance on the full test set every ten thousand training steps in order to track both training speed and overall performance. Figure 5.15 shows the multi-task test set loss (Equation 4.5) throughout each agent's training process, averaged across three runs for each agent. As expected, the baseline agent

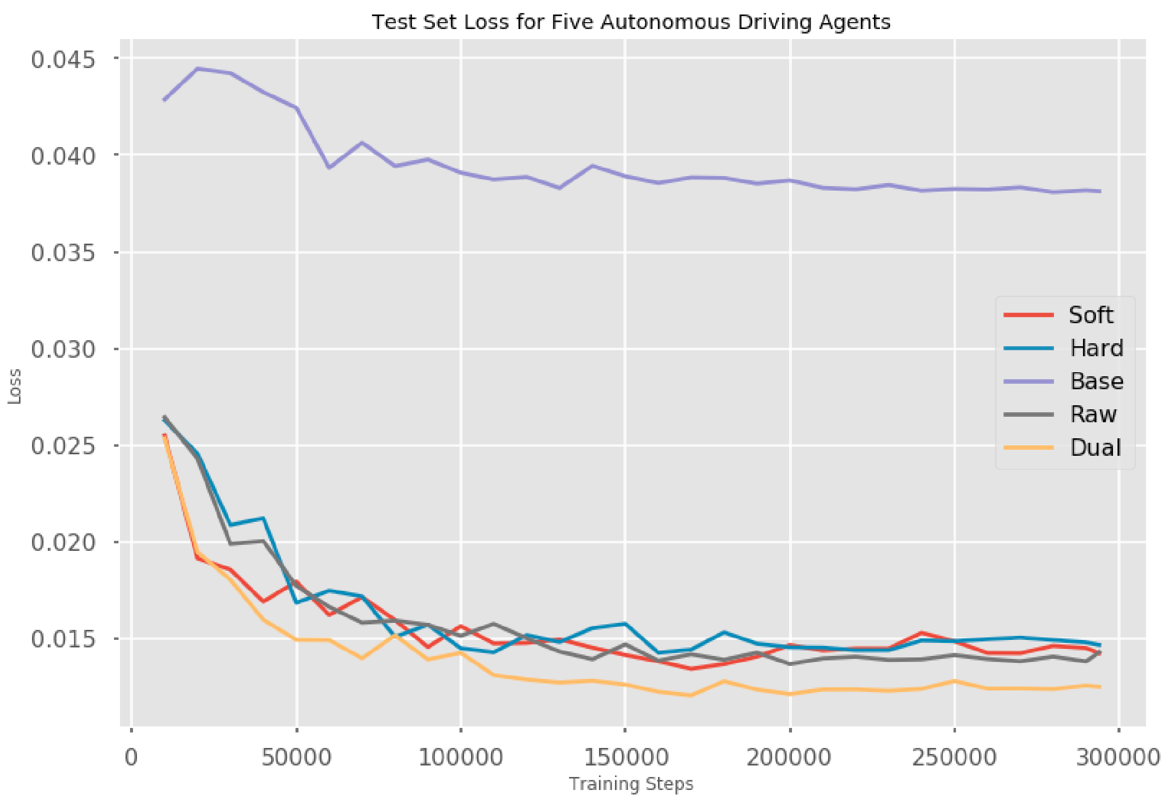


Figure 5.15: Test set loss for each agent, evaluated throughout the training process. Early training stages (10000-50000 steps) show an improved learning speed for the dual-branch and soft agents. After this point, however, the dual-branch agent continues to quickly learn and reduce its error, while the soft agent's performance slows to match that of the raw and hard agents.

has a very high loss compared to all other models, as the static masking process obscures crucial information from the agent. We observe that in the early stages of training, the soft agent appears to learn more quickly on average than the raw and hard agents, with loss values close to those of the dual-branch agent. After 40k training steps, however, the loss for the soft, hard and raw networks evens out, remaining approximately equal for the remainder of the training. Finally, our dual-branched agent was able to both reduce its loss more quickly and achieve a lower overall loss than all other agents, seemingly verifying the design of the dual-branch architecture for incorporating attention maps into an agent's learning process. We report the final multi-task test set loss for all agents in Table 5.3, as well as each of the individual task losses. The dual-branch agent achieves the minimum loss in all tasks, showing that the benefits of learning with attention are pervasive, improving all aspects of the agent's performance.

As discussed in Section 2.1.4, the weighted MSE loss function which we use to train these agents is often a poor indicator of actual driving ability. We therefore perform our final evaluation of these models using the mean average error of the predictions, which has proven to be significantly more correlated with driving performance[5].

Agent	Steer MSE	Brake MSE	Throttle MSE	Speed MSE	Weighted Multi-Task Loss
Raw Agent[6]	$3.366 * 10^{-4}$	$8.229 * 10^{-3}$	0.0231	5.878	0.0143
Baseline Agent (ours)	$5.003 * 10^{-3}$	0.0121	0.0358	13.73	0.0381
Hard Agent (ours)	$4.594 * 10^{-4}$	$9.690 * 10^{-3}$	0.0214	6.186	0.0146
Soft Agent (ours)	$3.878 * 10^{-4}$	$9.777 * 10^{-3}$	0.0210	6.773	0.0142
Dual-Branch Agent (ours)	**$1.596*10^{-4}$**	**$8.063*10^{-3}$**	**0.0205**	**5.848**	**0.0125**

Table 5.3: Test set loss values (lower is better in all categories) for each fully trained autonomous driving agent on all driving actions. Results highlight the superiority of the dual-branch agent, which achieves the lowest prediction loss for all four tasks. Individual task performance between the raw, soft, and hard agents varies by task, with the overall performance of these three models being very similar. The baseline agent, as expected, is the worst by a large degree for all tasks.

The results, depicted in Figure 5.16, further verify the superiority of the dual-branch agent, which learns faster and achieves a significantly lower MAE than all other models, achieving a 25.5% decrease in error with respect to the raw agent. When evaluated using this metric, the raw image agent also begins to slightly outperform the hard-masked and soft-masked agents when evaluated using MAE, although the gap in performance between these agents is still much smaller than the gap between the single and dual-branch agents. The results for the fully trained agents are given in Table 5.4.

Overall, the improved performance of the dual-branch model verifies our hypothesis that integrating a human-inspired attention mechanism into an autonomous driving agent's learning process can improve both performance and learning speed. In addition to this result, some other interesting observations can be made from the relative performance of the hard-masked, soft-masked and raw image agents. Prior to running the experiment, we had expected the soft-masked agent to outperform the raw image agent, as it contains the attention information produced by the Intention-Branched DR(eye)VE model while also retaining all of the original image data. However, the agent does not appear to take advantage of this extra information, and is in fact marginally outperformed by the raw image agent. The difference between the soft-masked agent and the dual-branch agent, both of which have access to the same total information, highlights the importance of model architectures when incorporating attention to machine learning systems. Finally, we were surprised to discover that the hard-masked agent performed approximately equal to the soft-masked and raw image agents, given the extent of the information which is removed during the image masking process. The baseline and hard-masked agents had approximately the same amount of overall information removed, but while the baseline agent was

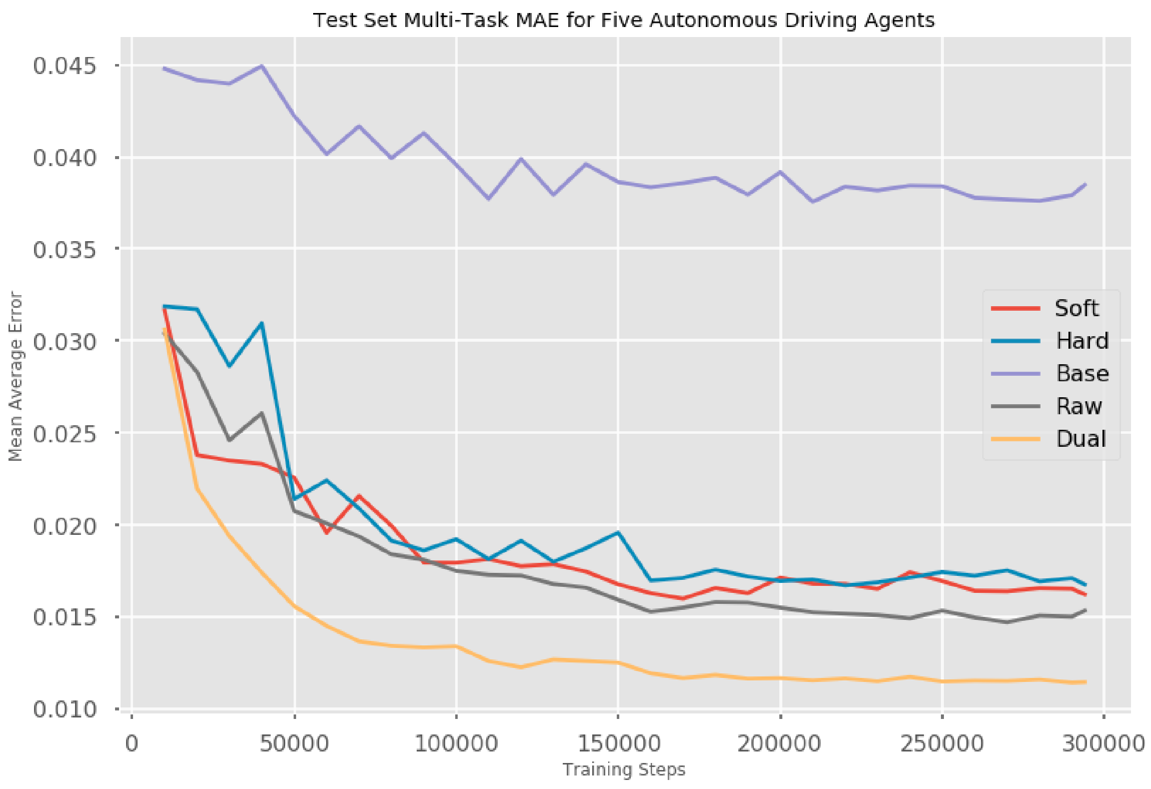


Figure 5.16: Test set MAE for each agent, evaluated throughout the training process. There is an even more pronounced improvement shown by the dual-branch agent over all other agents for this metric, which has been shown to correlate more strongly with online driving performance than MSE[5]. The raw agent can now be seen to have a slight performance gain over the soft and hard agents.

Agent	Steer MAE	Brake MAE	Throttle MAE	Speed MAE	Weighted Multi-Task MAE
Raw Agent[6]	$7.161 * 10^{-3}$	0.0250	0.0757	1.693	0.0153
Baseline Agent (ours)	0.0288	0.0393	0.121	3.096	0.0385
Hard Agent (ours)	$8.212 * 10^{-3}$	0.0303	0.0764	1.753	0.0168
Soft Agent (ours)	$7.955 * 10^{-3}$	0.0299	0.0733	1.877	0.0162
Dual-Branch Agent (ours)	**4.576$*$10^{-3}**	**0.0192**	**0.0626**	**1.620**	**0.0114**

Table 5.4: Test set MAE (lower is better) for each fully trained autonomous driving agent on all driving actions. As with the test set losses, these results show that the dual-branch agent performs best in all four tasks. A slightly more pronounced difference can now be seen between the performance of the raw agent and the soft/hard agents, with the raw agent outperforming the others by a slightly larger margin than before.

severely affected by this removal, the hard-masked agent continued to perform well due to the selective nature of the masking. This is a compelling affirmation of the Intention-Branched DR(eye)VE model's ability to correctly identify the regions of an image which are important to the driving task.

Chapter 6

Conclusion

6.1 Summary

Autonomous driving promises to be a beneficial and highly transformative technology in the near future. While modern approaches to high levels of autonomy use a modular approach to perform well in highly controlled, well-defined environments, end-to-end autonomous driving has recently emerged as a possible alternative which may have the potential to scale beyond what modular approaches are capable of. This approach relies on the ability to understand and react to highly complex real-world scenes captured by the vehicle's sensory systems, with no explicit instructions or behaviours defined beforehand - an ability that humans excel at and perform on a daily basis. By quickly and purposefully directing eye movements to objects of interest, humans can quickly acquire the information needed to perform a task while filtering out unnecessary details. This skill would provide clear benefits to any system which has to parse complex scenes as part of a task.

To create a system capable of replicating human visual attention, we collected several hours of simulated driving data from human drivers while recording their eye movements. We then evaluated four recently published approaches to predicting human gaze on our dataset[18][7][2][23], finding that the most successful architecture used a combination of raw images and preprocessed optical flow maps to provide multiple source of information to the model. We further extended this model by providing the high-level intentions of the driver for each frame, boosting performance even further. Our fully-trained model is able to identify key regions in images which are relevant to the driving task, as measured by its high performance in predicting human gaze for an unseen set of test episodes.

Motivated by other works which have improved the performance of machine learning systems by incorporating attention-focusing mechanisms into their learning processes [34][37], we tested several methods of introducing this information to an autonomous driving agent. Five conditional imitation learning agents[6] were evaluated, with substantial performance increases being attained by a dual-branch model which trained using both raw driving images and masked images produced by elementwise multiplication of these images with the gaze network's produced attention maps. We note the importance of architecture choice when incorporating attention maps into these systems, as our soft-masked agent which in theory had access to the same information as the dual-branch agent did not see any performance increase as a result. Finally, we found that an agent trained solely on the information retained by a hard masking of the original images with the gaze network's attention maps suffered almost no decrease in performance in comparison to a network trained with full driving images, validating the effectiveness of the gaze network in extracting all of the necessary information from these images for the driving task.

6.2 Future Work and Improvements

Several extensions and improvements could be made to our experiments to further increase performance in both the gaze prediction and driving tasks. One factor which may have impacted our ability to faithfully record all eye movements was the fact that since we record gaze location by tagging the position of a GazeCursor object within CARLA, we are limited to sampling gaze locations at the refresh rate of the server (25 frames per second), while our gaze tracking technology is capable of recording at a significantly higher rate. While it is unlikely that we miss any actual gaze fixations in the 40 milliseconds between frames, we may be missing some information encoded in the saccades (rapid movement of the eyes between fixations) of the driver. Due to the lag present in the simulation when recording with more than one camera, we were also unable to take advantage of CARLA's ability to provide ground-truth segmentation maps of driving scenes. While Palazzi et al. found that semantic segmentation added very little to their gaze prediction model[23], this may be more reflective of the pre-trained segmentation algorithm they used to process their images, rather than the somewhat non-intuitive conclusion that semantic labels would not provide important information to a driving system. It would be interesting to see if having access to this information could resolve some of the failure cases for our gaze prediction model. Finally, due to time constraints, we were not able to perform online evaluation of our driving agents, which could provide more specific insights into the effect of incorporating attention into driving systems such as collision rates, navigation episode completion statistics and more.

One important strategy for end-to-end learning which we did not investigate in this project is the reinforcement learning approach. Reinforcement learning provides a framework for models to continuously improve their performance with experience, but tends to suffer from extreme sample inefficiency in complicated environments with continuous action spaces. For this reason, pure reinforcement learning attempts to train self-driving agents have not met with much success. We believe that incorporating visual attention masking into a reinforcement learning agent could potentially have great benefits in terms of sample efficiency, as the agent should be able to much more rapidly associate highlighted image regions with the reward signals it receives as a result of its actions. We believe this to be a promising direction for future work. Beyond autonomous driving, the ability to infuse machine learning systems with human expertise in a natural, data-driven way holds enormous potential. The same principles and strategies implemented in this project could be applied to any vision-based task that we may wish to automate such as street cleaning or construction. While eye movement data is not trivial to collect, portable, battery-powered eye tracking systems have made the collection of high-quality gaze data increasingly viable over the last decade, and we hope to see future works benefiting from visual attention systems such as ours.

Bibliography

[1] The six official levels of autonomous vehicles explained. `https://boingboing.net/2017/03/03/the-six-official-levels-of-au.html`. Accessed: 2019-09-01. pages 2

[2] BAZZANI, L., LAROCHELLE, H., AND TORRESANI, L. Recurrent mixture density network for spatiotemporal visual attention. *CoRR abs/1603.08199* (2016). pages 36, 37, 49, 58, 66

[3] BOJARSKI, M., TESTA, D. D., DWORAKOWSKI, D., FIRNER, B., FLEPP, B., GOYAL, P., JACKEL, L. D., MONFORT, M., MULLER, U., ZHANG, J., ZHANG, X., ZHAO, J., AND ZIEBA, K. End to end learning for self-driving cars. *CoRR abs/1604.07316* (2016). pages 5, 6

[4] BYLINSKII, Z., JUDD, T., OLIVA, A., TORRALBA, A., AND DURAND, F. What do different evaluation metrics tell us about saliency models? *CoRR abs/1604.03605* (2016). pages 16

[5] CODEVILLA, F., LÓPEZ, A., KOLTUN, V., AND DOSOVITSKIY, A. On offline evaluation of vision-based driving models. *CoRR abs/1809.04843* (2018). pages 9, 10, 30, 62, 64

[6] CODEVILLA, F., MÜLLER, M., DOSOVITSKIY, A., LÓPEZ, A., AND KOLTUN, V. End-to-end driving via conditional imitation learning. *CoRR abs/1710.02410* (2017). pages 5, 7, 9, 42, 43, 44, 59, 61, 63, 64, 67, 80, 81, 82

[7] CORNIA, M., BARALDI, L., SERRA, G., AND CUCCHIARA, R. A deep multi-level network for saliency prediction. *CoRR abs/1609.01064* (2016). pages 12, 13, 34, 48, 58, 66

[8] DOSOVITSKIY, A., ROS, G., CODEVILLA, F., LÓPEZ, A., AND KOLTUN, V. CARLA: an open urban driving simulator. *CoRR abs/1711.03938* (2017). pages 5, 9, 18

[9] FARNEBÄCK, G. Two-frame motion estimation based on polynomial expansion. In *Image Analysis* (Berlin, Heidelberg, 2003), J. Bigun and T. Gustavsson, Eds., Springer Berlin Heidelberg, pp. 363–370. pages 26

[10] GATYS, L. A., KÜMMERER, M., WALLIS, T. S. A., AND BETHGE, M. Guiding human gaze with convolutional neural networks. *CoRR abs/1712.06492* (2017). pages 82

[11] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural computation 9* (12 1997), 1735–80. pages 37

[12] HOPPE, D., AND ROTHKOPF, C. A. Multi-step planning of eye movements in visual search. *Scientific Reports 9*, 1 (2019), 144. pages 25

[13] HUANG, X., SHEN, C., BOIX, X., AND ZHAO, Q. Salicon: Reducing the semantic gap in saliency prediction by adapting deep neural networks. In *2015 IEEE International Conference on Computer Vision (ICCV)* (Dec 2015), pp. 262–270. pages 12

[14] JIANG, M., HUANG, S., DUAN, J., AND ZHAO, Q. Salicon: Saliency in context. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 1072–1080. pages 34

[15] JOHNSON, L., SULLIVAN, B. T., HAYHOE, M. M., AND BALLARD, D. H. Predicting human visuomotor behaviour in a driving task. In *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* (2014). pages 15

[16] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. pages 12

[17] KRUTHIVENTI, S. S., AYUSH, K., AND BABU, R. V. Deepfix: A fully convolutional neural network for predicting human eye fixations. *CoRR abs/1510.02927* (2015). pages 12

[18] KÜMMERER, M., WALLIS, T. S. A., AND BETHGE, M. Deepgaze II: reading fixations from deep features trained on object recognition. *CoRR abs/1610.01563* (2016). pages 12, 33, 34, 58, 66

[19] KMMERER, M., THEIS, L., AND BETHGE, M. Deep gaze i: Boosting saliency prediction with feature maps trained on imagenet. pages 12

[20] LECUN, Y., COSATTO, E., BEN, J., MULLER, U., AND FLEPP, B. Dave: Autonomous off-road vehicle control using end-to-end learning. Tech. Rep. DARPA-IPTO Final Report, Courant Institute/CBLL, http://www.cs.nyu.edu/˜yann/research/dave/index.html, 2004. pages 5

[21] MAKRIGIORGOS, A. Independent study option: Autonomous driving systems and human visual attention, 2019. pages i, 4, 5, 6, 7, 8, 12, 13, 14, 15, 16, 81

[22] MEHTA, A., SUBRAMANIAN, A., AND SUBRAMANIAN, A. Learning end-to-end autonomous driving using guided auxiliary supervision. *CoRR abs/1808.10393* (2018). pages 8, 82

[23] PALAZZI, A., ABATI, D., CALDERARA, S., SOLERA, F., AND CUCCHIARA, R. Predicting the driver's focus of attention: the dr(eye)ve project. *CoRR abs/1705.03854* (2017). pages 15, 26, 39, 40, 41, 58, 66, 67

[24] POMERLEAU, D. A. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1989, pp. 305–313. pages 4, 7

[25] SHALEV-SHWARTZ, S., SHAMMAH, S., AND SHASHUA, A. Safe, multi-agent, reinforcement learning for autonomous driving. *CoRR abs/1610.03295* (2016). pages 5

[26] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations* (2015). pages 12, 33

[27] SPRAGUE, N., AND BALLARD, D. Eye movements for reward maximization. In *Advances in Neural Information Processing Systems 16*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds. MIT Press, 2004, pp. 1467–1474. pages 14

[28] SULLIVAN, B., JOHNSON, L., BALLARD, D., AND HAYHOE, M. A modular reinforcement learning model for human visuomotor behavior in a driving task. pp. 33–40. pages 15

[29] SULLIVAN, B., JOHNSON, L., ROTHKOPF, C., BALLARD, D., AND HAYHOE, M. The effect of uncertainty and reward on fixation behavior in a driving task. *Journal of Vision 12* (08 2012), 1259–1259. pages 15

[30] TATLER, B. The central fixation bias in scene viewing: Selecting an optimal viewing position independently of motor bases and image feature distributions. *Journal of vision 7* (02 2007), 4.1–17. pages 34

[31] TRAN, D., BOURDEV, L. D., FERGUS, R., TORRESANI, L., AND PALURI, M. C3D: generic features for video analysis. *CoRR abs/1412.0767* (2014). pages 36, 37, 40

[32] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, L. U., AND POLOSUKHIN, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008. pages 8

[33] VIG, E., DORR, M., AND COX, D. Large-scale optimization of hierarchical features for saliency prediction in natural images. In *2014 IEEE Conference on Computer Vision and Pattern Recognition* (June 2014), pp. 2798–2805. pages 12

[34] XU, K., BA, J., KIROS, R., CHO, K., COURVILLE, A. C., SALAKHUTDINOV, R., ZEMEL, R. S., AND BENGIO, Y. Show, attend and tell: Neural image caption generation with visual attention. *CoRR abs/1502.03044* (2015). pages 8, 66

[35] YARBUS, A. L. *Eye Movements and Vision*. Plenum. New York., 1967. pages 11

[36] YOU, Y., PAN, X., WANG, Z., AND LU, C. Virtual to real reinforcement learning for autonomous driving. *CoRR abs/1704.03952* (2017). pages 5

[37] ZHANG, R., LIU, Z., ZHANG, L., WHRITNER, J. A., MULLER, K. S., HAYHOE, M. M., AND BALLARD, D. H. AGIL: learning attention from human for visuomotor tasks. *CoRR abs/1806.03960* (2018). pages 8, 15, 16, 39, 45, 59, 66, 81, 82

Appendix A

Implementation

In this chapter, we provide details on the programming implementation process for the various parts of the project. The code for this project was developed on a personal ASUS laptop with an Ubuntu 18.04 operating system, an NVIDIA GeForce GTX 1070 graphics card and an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz. All code has also been tested and verified to run on the Brain and Behaviour Lab's remote machines. All of the code for this project is made available in the submitted code archive, and is also currently uploaded to the Brain and Behaviour Lab's computing servers.

A.1 Project Setup & Development

This code for this project was written in Python, in the form of Jupyter Notebooks. The libraries and packages most frequently used throughout the development process were:

- NumPy for fast computations and creating, saving and loading large data arrays
- OpenCV for image reading, writing and manipulation, as well as the creation of video files from image sequences
- Matplotlib for visualization of dataset features
- Keras, Tensorflow and Theano for deep learning
- os and glob for reading directory contents and generating/writing new file names

Virtual Environments

Much of the deep learning code used to train and compare different models for this project was taken from published code repositories provided by the authors of those models. The programs in these repositories are written for specific versions of deep learning libraries such as Tensorflow, Theano and Keras, which are incompatible with newer versions. To avoid compatibility issues and keep the models working exactly as they worked when published, we create virtual environments using the Anaconda platform, allowing us to run the networks with the packages and versions in which they were originally run. Any Anaconda virtual environment can be described by a .yaml configuration file containing the exact packages and their version numbers which are used in that environment. All of the configuration files used for this project are available in the submitted code archive or on the Brain and Behaviour Lab's remote servers, and using these files the virtual environments can be automatically recreated by Anaconda. The ipykernel package was used to allow Jupyter to easily create Notebook servers which execute in Anaconda virtual environments.

Here we describe the contents of the python modules used to collect data for both the Gaze and Driving datasets.

A.2 Dataset Creation & Preprocessing

In this section we describe the contents of the notebooks used to create and preprocess the Gaze and Driving datasets.

- **run_gaze_collection.py**

  This file is used to collect data for the Gaze experiments. It allows users to connect to a running CARLA server using the Python Client and drive a vehicle using a Ferrari 458 Italia USB steering wheel. A pygame controller was written to read the values from the steering wheel and send them to the server via the Python Client. Weather settings can be changed by pressing 'C' button on the laptop's keyboard. The script can be run with or without data recording using the command line argument `--images-to-disk`. If enabled, it records the camera images, extrinsic and intrinsic matrices for each frame, storing them in a buffer of size 1000 which is flushed to disk when full to avoid lag in the simulation, as described in Section 3.1.2.

- Gaze.ipynb

  This notebook contains the code for converting the collected subject data into a preprocessed, organized dataset which can be directly used by the gaze prediction models. It contains functions to perform the following tasks:

- Convert consecutive 1000x400x300 NumPy arrays into into image sequences of 4500 images in PNG format, given an absolute starting frame number. Store the images in a directory in the format ”Data/episode_number/frames/frame_number.png”.
- Store the recorded extrinsic and intrinsic camera matrices for each frame as a single numpy array in the format ”Data/episode_number/extrinsic.npy”.
- Parse and split the binary files containing subjects’ eye movement data into separate episodes (given starting frame numbers for each episode), convert them into NumPy arrays, and save them in the format ”Data/episode_number/eyesXYZ.npy”.
- Convert a set of (X,Y,Z) gaze coordinates into (x,y) camera coordinates as described in detail in Section 3.1.3. If a list of (X,Y,Z) coordinates is passed in, returns a list of gaze coordinates corresponding to the given intrinsic and extrinsic matrices (used when computing fixation maps).
- Display an image sequence as a video, overlaying the camera frames with red circles at the locations of the subject’s gaze within a 1 second time window of the current frame. Useful to confirm there are no errors in the eye tracker before writing final fixation maps.
- Compute the fixation maps for each frame of an episode as described in Section 3.1.4 and save them in the format ”Data/episode_number/fixations/frame_number.png”. Requires all previously described functions to have been run for the current episode, such that the gaze information and camera matrices are stored in the episode’s directory in NumPy array format.
- Compute the dense optical flow maps for each frame of an episode as described in Section 3.1.5 and save them in the format ”Data/episode_number/optical_flow/frame_number.png”. Requires the camera frames to have been written for the current episode.
- Display an image sequence as a video with four simultaneous views: Raw camera images, computed fixation maps, computed optical flow maps, and a blending of camera images with fixation maps. Useful for verifying a full episode has been correctly written and all frame numbers are synchronized.
- Calculate the mean camera frame and fixation map for an episode or for the entire dataset. This is used for preliminary analysis of the collected data and in the DR(eye)VE network, which subtracts the mean from all samples when training.
- Accumulate the distribution of subjects’ gaze within the camera frame across all episodes and display a histogram with the results. Used when determining which image regions to crop as described in Section 3.1.7.
- Crop all images, fixation maps and flow maps based on the conclusions drawn from the data visualization. In the case of fixation maps, renormalize such that they remain probability distributions.

  - Convert lists containing hand-labeled frame numbers for high-level commands and traffic light stopping into NumPy arrays and save them in the corresponding episode directories, to be used for analysis and by the Intention-Branched DR(eye)VE model.

- Drive.ipynb

  This notebook contains code for producing the Driving dataset from episode data generated by CARLA's expert demonstrator agent. It contains functions for the following tasks:

  - Convert the contents of an episode recorded by the expert demonstrator into an episode which can be used for prediction by the DR(eye)VE network. For each frame, the dense optical flow map is computed, the three frames (RGB, Semantic Segmentation, Optical Flow) are cropped and resized to the dimensions used by the DR(eye)VE network, and the JSON metadata file is read and the necessary control signals for autonomous driving are read. All outputs are then rewritten to a new directory, in the same format and structure as the DR(eye)VE dataset directories.

  - Read the contents of all episodes generated by the previous function, as well as the attention maps predicted by the DR(eye)VE network, downsize the images by a factor of two for input to the CIL network, and write the outputs to a single combined directory with a subdirectory for each data type (RGB frames, attention maps, metadata files). Requires the fully trained DR(eye)VE network to have recorded predictions for all episodes.

  - Using the master directory created by the previous function, precompute hard, soft and baseline attention masks as described in Section 3.2.3, and write the outputs to new subdirectories.

  - Display an image sequence as a video containing four views: Raw images, soft masks, hard masks, and baseline masks. Useful to ensure that all directories have been written correctly and are synchronized before writing the files into TFRecord format.

  - Convert the contents of the master directory into TFRecord format for use by the CIL network. Each set of frames (RGB, soft, hard, baseline masks) is serialized to string format and added to a dictionary of features. Each target value from the corresponding metadata file is then added to the feature dictionary, and the entire dictionary is converted into a single TF Example. A batch size of 2000 is used, such that each generated TFRecord file contains 2000 TF Examples. The final output is a directory of TFRecords which can be universally used by the CIL networks described in Section 4.2 by accessing any TF Example feature by its dictionary key.

A.3 Gaze Networks

In this section, we provide details on the notebooks used to construct, train and evaluate each of the Gaze Prediction models used for this project. The virtual environment utilized for all of these networks is the one described in the 'gaze.yml' file in the code archive. Of most importance is the fact that these notebooks are written in Python 2.7, and the models are trained and tested in Keras 1 using the Theano backend. First we will describe the functions which are present in every notebook, followed by a short description of each specific notebook with any unique details.

- Common Functions

  Here we list the functions which are common to all implemented Gaze Prediction models.

  - **Image Utils**: A set of common utility functions is used for image manipulation. `read_image` reads an image from an input file path, reshapes the dimensions so that the channel dimension is first as required by Theano, optionally converts between RGB and BGR color schemes, optionally resizes the image to any specified dimensions, and finally returns the image as a Numpy array. `normalize` takes a NumPy array as input and normalizes the values to lie between 0 and 255 and converts them to uint8 format. `write_image` takes NumPy arrays in this normalized format and writes them as PNG images to a specified filepath. Finally, `stitch_together` takes any number of equal-sized images and a tuple indicating a desired layout scheme and combines the images into a grid so that they can be visualized side by side.
  - **Callbacks**: When training networks, Keras makes use of functions called "Callbacks", which are called at regular intervals throughout the training process. The History and BaseLogger callbacks are automatically applied to every Keras model to record metrics about the model performance such as loss and validation accuracy. In addition to these, the following callbacks are used by the Gaze Prediction models: (1) Checkpointer, which saves a checkpoint of the model's weights in an h5 file after each epoch, (2) ReduceLROnPlateau, which decreases the learning rate by a factor of 10 if no improvement is seen in the validation loss after 5 training epochs, and (3) PredictionCallback, a custom-defined callback which writes a small number of predictions from the validation set to an output directory after each epoch. This allows the user to monitor the model's progress over time.
  - **Batch Generators**: Custom functions are defined for all networks which yield batches of data for training and evaluation when called by Keras' fit_generator function. Given a batch size, the generators randomly select that number of frames from the Gaze dataset, along with their fixation map labels. The frames are chosen from a specified range depending on

whether the network is performing training or evaluation. Data augmentation is also performed at this stage in the form of randomly mirroring images and their fixation maps. In the case of RMDN and DR(eye)VE, a clip of 16 frames is returned instead of a single frame. Additionally, DR(eye)VE batches include pre-computed dense optical flow maps for each frame.

- **Training** All models use Keras' `fit_generator` function to train. This function takes as input generator functions (as described above) for training and validation data, number of samples to train and validate on for each epoch, the total number of epochs to train for, and all callbacks to be used when training the model.
- **Inference** Once each model has been fully trained, predictions are made for each image in the six test set episodes of the Gaze dataset. For each predicted image, the Kullback-Liebler Divergence and Correlation Coefficient metrics are calculated between the predictions and the ground truth fixations, and these metrics are recorded in a text file.

- MLNet.ipynb

  This notebook is used to build and train the Multi-Level Network. An h5 file containing a pre-trained VGG-16 model is required to initialize the weights of the Feature Extraction network. These layers are set to non-trainable. A custom layer is used in this notebook, named EltWiseProduct, which defines a coarse, learned bias which is upsampled and multiplied elementwise with the output of the feature encoding network as described in detail in Section 4.1.2.

- RMDN.ipynb

  This notebook implements and train the Recurrent Mixture Density Network. In addition, it contains a function which computes C3D encodings for the Gaze dataset - this function loads a pre-trained C3D model, iterates over all images in the Gaze dataset, and predicts a C3D encoding for each sequence of 16 images which is saved as a compressed NumPy array. A download script for the pretrained C3D network is also included. The RMDN then trains using these C3D encodings as input, and a function is used to convert the Gaussian parameter outputs to probability maps. When training, the Prediction Callback writes an image for each Gaussian predicted by the network, as well as the weighted output of the Mixture Model.

- Dreyeve.ipynb

  This notebook is used to construct and train the DR(eye)VE network. The network must be trained in stages as described in Chapter 5.1.4, first training branches individually before fine-tuning the full model by training the branches together. As such, unique training functions and batch generators are used depending on which branches are currently in use. The Prediction Callback used while training this network includes predictions from both the cropped and full-frame image regions.

- Dreyeve-Branched.ipynb

  This notebook extends the Dreyeve notebook to allow the model to train on specific high-level commands. The main changes are made in the batch generators, which are modified to only select image sequences which correspond to the high-level command for the branch currently being trained. In addition, because this model had the best performance of all the Gaze Prediction networks, it was used to make the predictions for the Imitation Learning episodes produced in the Drive notebook. The code for making those predictions is therefore also included in this notebook.

A.4 Driving Networks

This section provides details for the notebooks used to train the two autonomous driving networks implemented for this project, named CIL_TF.ipynb abd CIL_TF_Branched.ipynb. The virtual environment necessary to run these notebooks can be created with the 'cil.yml' file in the code archive. The model is constructed and trained using Tensorflow Estimators, with Tensorflow version 1.12 and Python 2.7. The structure and content of the two notebooks used is nearly identical, with the only significant differences found in the construction of the network and the processing of the input data, as detailed below. The major functions of the two notebooks are the following:

- **Config**: The configuration settings for the CIL notebooks are written in a configuration file labeled 'config-train-production.yaml'. These include training parameters such as number of training steps, filepaths for reading and writing data, and the lambdas to be used for each output action when calculating loss as described in Section 4.2.1. These values are read and saved as TensorFlow Flags before training begins.
- **Hooks**: The TF Estimator equivalent of Callbacks, Hooks are used to perform certain actions throughout the training process. After every ten thousand training steps, the model first saves a checkpoint file containing its current parameters, and then evaluates its performance on both the training and validation sets, saving the results in a TFEvent file. The mean squared error and mean average error are recorded for each task separately, in addition to their lambda-weighted sum. These metrics can then be easily visualized using Tensorboard, an interactive visualization suite provided by TensorFlow.
- **Image Augmentation**: Random augmentations are applied to input images throughout the training process to make the model more robust and avoid overfitting. These augmentations are applied by the `ProbabilisticImageAugmentor` class. This class is initialized with a probability value p between 0 and 1, and an augmentation to perform, augments the image with probability p, and returns the output. More details on the augmentations used can be found in Section 5.2.

- **Batch Generation**: The batch generator creates a TensorFlow Dataset object from the TFRecord files specified, deserializes the TF Examples contained within, applies a series random augmentation to the image features, and finally yields batches of images and labels when called by the training function. In the case of the CIL_TF_Branched notebook, two sets of images are returned with each set of labels, as this branched architecture requires both the raw RGB camera inputs and the hard-masked images.

- **Loss Function**: For each input batch, all five branch heads make their own separate predictions. However, when calculating the loss to train the network, we only wish to consider the predictions made by the branch head corresponding to the current high-level command. Therefore a custom loss function is written to select only the relevant predictions and compare them to the training labels. The mean-squared error is then calculated for each task, weighted by the lambdas defined in the configuration file and summed for the total loss.

- **Model Definition**: The two CIL models previously visualized in Figures 4.6 and 4.7 are constructed with a convolutional architecture identical to that of the original paper [6] and initialized as Estimators with the given configuration file parameters.

- **Training**: The networks are trained via the TF.Estimator.train function, which simply takes as input a generator, number of train steps, and a set of Hooks to apply during training. Outputs are automatically saved to a new directory each time a new training run begins.

Appendix B

Relationship to ISO

This work done in this project builds upon my Independent Study Option[21], which was carried out in the Winter term of 2019 under Dr. Aldo Faisal's supervision. In this ISO, I conducted a literature survey of the history and current state of the art in both end-to-end autonomous driving technology and human gaze prediction techniques. The ISO concluded by suggesting that a model which integrates learned visual attention to an autonomous driving system could potentially improve training speed, performance or both, and proposed some theoretical ways in which this integration could be performed. No data collection or experimentation was performed during the ISO.

This project follows up on the ideas discussed in the ISO by implementing a system which combines visual attention with training autonomous agents. All of the data collection experiments, model implementations, and evaluations contained in this report were carried out solely during the course of the MSc project and do not overlap with the work done in the ISO. We do not actually implement the integration strategies proposed in the ISO for our final models, opting instead to use single-branch masking strategies and the dual-branch strategy of [37] combined with the CIL driving architecture of [6].

Although some of the ISO's content is beyond the scope of this project (e.g. sensor fusion, domain shift from simulation to reality), the core ideas surrounding end-to-end driving and gaze prediction remain central to this work. As such, much of the content of the ISO report is rewritten and included in the Background chapter of this project to preserve coherence, with citations included where necessary. The report additionally contains background sections which did not appear in the ISO, such as descriptions of the techniques used to evaluate gaze and driving systems and a comprehensive description of the CARLA environment. We now proceed with a brief summary of each section of the ISO report, making a note of which section topics were also included in this thesis.

Section 2 of the ISO reviewed the Actor-Critic reinforcement learning algorithm as well as some techniques for visualizing feature map activations in neural networks. These sections were included as key concepts for the theoretical contributions of the ISO, which in the end were not implemented in this project as they largely dealt with reinforcement learning techniques, while we instead focus on imitation learning. This section is therefore not included in any form in this report.

In Section 3 of the ISO, we reviewed advances in autonomous driving in the areas of imitation learning, reinforcement learning, sensory fusion, simulation-to-reality domain shift techniques, and finally existing attention models. From this section, we retain the topics of imitation learning and attention models. Codevilla's imitation learning model reviewed in the ISO[6] is the model which we use to compare performance between agents with and without attention. The Guided Auxiliary Supervision Network discussed in [22] is the only example of which we are aware that attempts to incorporate attention to driving in some manner, and is discussed to highlight the fact that this is a relatively unexplored area.

Section 4 reviews bottom-up and top-down gaze prediction strategies, as well as some applications for both areas such as exploiting bottom-up saliency features to guide human gaze in[10] and the AGIL framework for playing Atari games[37], which inspired our successful dual-branch attention model. We reproduce nearly everything from these sections in this project's report due to their high relevance and the fact that we end up implementing several of the gaze models discussed.

The report concludes by suggesting methods for integrating a visual attention model with autonomous driving architectures. We propose applying attention models to an Actor-Critic algorithm to improve the sample of reinforcement learning techniques for self-driving, as well as the computation of a similarity score between a model's current feature map activations and the attention map recorded by a human which could be included in a model's loss function. These proposals were not carried out in this project, thus they are not included in this report.

This project contains considerable contributions and efforts which are independent of the work done in the ISO. We collected and labeled a high-quality gaze dataset from human drivers, trained and compared performance for four deep gaze prediction models, and introduced a novel addition to the best-performing model to further boost its prediction scores. We then collected a driving image dataset, computed attention masks for each frame, and trained/evaluated five autonomous driving agents using different attention masking techniques to examine the effects of masking and different architectures on prediction performance. All of this work was done during the summer months of the MSc project.

Appendix C

Ethical and Legal Considerations

Fully autonomous vehicles are rapidly becoming a reality, and are poised to bring many transformative changes to society and infrastructure in the near future; reduced accidents, fewer CO2 emissions, more productive commuting, and reduced traffic congestion to name a few. There are several legal and ethical implications, however, that must be well considered before releasing fleets of driverless cars onto public roads. Easily the most debated ethical question with regards to autonomous vehicles is the issue of determining whose safety to prioritize in the event of an unavoidable collision. Imagine a scenario in which a child runs out in front of an autonomous vehicle driving on a two-lane road: should the vehicle swerve into oncoming traffic in order to avoid hitting the child, risking the safety of the driver and people in other vehicles? If a human were driving in such a scenario, their reaction would be instinctive and panicked, but an AI would be forced to make a deliberate decision about whose safety to prioritize. The question of how to weight human life is extremely delicate, and as of yet there is no clear solution to this problem. Another ethical issue to consider is the unemployment which may occur if driverless taxi services become widely available, as they are likely to be cheaper than taxis or ride hailing services. This is an increasingly prevalent issue as machine learning begins to automate more and more tasks.

The legal issues surrounding autonomous vehicles are chiefly focused on insurance - if an accident occurs as a result of a software malfunction, who is liable - the owner of the car, the manufacturer of the car, the developer of the driving software? What if a driverless vehicle collides with a human-operated vehicle? Factors which currently determine liability in accidents such as risky versus safe driving are far less clear when control has been handed over to the computer, and insurance protocols concerning these types of accidents will likely have to be reevaluated. As with other smart technologies, data protection is another important legal issue which will require some consideration as our cars become more intelligent and learn more about our routines and the places we travel. Outside of these general issues in regards to autonomous driving, we do not believe that this project has any specific legal or ethical implications to further consider.

Appendix D

Ethics Checklist

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		X
Does your project involve the use of human embryos?		X
Does your project involve the use of human foetal tissues / cells?		X
Section 2: HUMANS		
Does your project involve human participants?	X	
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)?		X
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		X
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		X
Does it involve processing of genetic information?		X
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		X
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		X
Section 5: ANIMALS		
Does your project involve animals?		X
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		X
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		X

Could the situation in the country put the individuals taking part in the project at risk?		X
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		X
Does your project deal with endangered fauna and/or flora /protected areas?		X
Does your project involve the use of elements that may cause harm to humans, including project staff?		X
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		X
Section 8: DUAL USE		
Does your project have the potential for military applications?		X
Does your project have an exclusive civilian application focus?		X
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		X
Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		X
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		X
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		X
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		X
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		X
Section 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		X
Will your project use or produce goods or information for which there are data protection, or other legal implications?		X
Section 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?	X	

Table D.1: Ethics Checklist