

https://dspace.org/

The Application of Attribution Methods to Explain an End-To-End Model For Self-Driving Cars

Chalom, Jason Max

2024-09
University of the Witwatersrand, Johannesburg
https://hdl.handle.net/10539/46909

Chalom, Jason Max. (2024). The Application of Attribution Methods to Explain an End-To-End Model For Self-Driving Cars. [Master's dissertation, University of the Witwatersrand, Johannesburg]. WIReDSpace. https://hdl.handle.net/10539/46909

Downloaded from a DSpace repository. https://dspace.org/

The Application of Attribution Methods to Explain an End-To-End Model For Self-Driving Cars

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Jason Chalom
711985

Supervised by Professor Richard Klein

September 27, 2024



A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science

Declaration

I, Jason Max Chalom, hereby declare that the contents of this master's dissertation to be my own work. This dissertation is submitted for the degree of Master of Science in Computer Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

Signature: 

Signed on 27 day of September , 2024 in Johannesburg.

Abstract

There has been significant development in producing autonomous vehicles but a growing concern is in understanding how these systems work, and why certain decisions were made. This has a direct impact on the safety of people who may come into contact with these systems.

This research reproduced the experimental setup for an end-to-end system by Bojarski *et al.* [2016b]. End-to-end self-driving AI structures are built on top of black-box machine learning techniques. The source code can be found here: `https://github.com/TRex22/masters-gradsum`.

An allure of end-to-end structures is that they need very little human input once trained on large datasets and therefore have a much lower barrier to entry, but they are also harder to understand and interpret as a result.

Bojarski *et al.* [2016b] defined a reduced problem space setup for a self-driving vehicle. This task only has a forward-facing camera which generates RGB images as input, and only the vehicle's steering angle is the output. This work expanded the setup to include six CNN model architectures over the single model used by Bojarski *et al.* [2016b] to compare the behaviours, outputs and performance of the varying architectures.

There have been recent developments in applying attribution methods to deep neural networks in order to understand the relationship between the features present in the input data and the output. GradCAM is an example of an attribution technique which has been validated by Adebayo *et al.* [2018].

We devised an attribution analysis scheme called GradSUM which is applied to the models throughout their training and evaluation phases in order to explain what features of the input data are being extracted by the models. This scheme uses GradCAM and uses segmentation maps to correlate inputted semantic information using the resultant gradient maps. This produces a model profile for an epoch which can then be used to analyse that epoch.

Six models were trained, and their performance compared using their MSE loss. An autonomy metric (MoA) common in literature was also used. This metric tracks where a human has to take over to stop a dangerous situation. The models produced good loss results. Two model architectures were constructed to be simple in order to compare against the more complex models. These performed well on the loss and MoA metrics for the training data subset but performed poorly on other data. They were used as a control to make sure that the proposed GradSUM scheme would adequately help compare emergent behaviours between architectures.

Using GradSUM on the architectures, we found that three out of the six models were able to learn meaningful contextual information. The other three models did not learn anything meaningful. The two trained simple models' overall activation percentages were also close to zero, indicating these simple model architectures did not learn enough useful information or became over-trained on features not useful to safely driving a vehicle.

Dedication

In loving memory of my father,
Raymond Edward Chalom.

Acknowledgements

I want to express my gratitude to my supervisor, Professor Richard Klein, for guiding me through the process of completing my master's degree.

I would also like to acknowledge my family and my girlfriend, Kezia, for all the support during the ups and downs of completing a master's degree.

My deepest thanks also go to Michael Herd for his help and guidance through this journey.

This work also would not have been possible without the support of my employer, nCino Inc. (Formally DocFox Inc.).

Some computations were performed using High-Performance Computing infrastructure provided by the Mathematical Sciences Support unit at the University of the Witwatersrand.

Contents

Preface
Declaration . . . i
Abstract . . . ii
Dedication . . . iii
Acknowledgements . . . iv
Table of Contents . . . v
List of Figures . . . ix
List of Tables . . . xii
Glossary of Terms . . . xiii

1 Introduction **1**
1.1 The core objectives of this research . . . 4

2 Background **5**
2.1 Introduction . . . 5
2.2 Historical Work . . . 5
2.3 A Scale of Autonomy . . . 6
2.4 Hierarchical Structures . . . 7
2.5 End-to-End Structures . . . 7
2.6 Deep Learning Attribution Methods . . . 8
2.6.1 Perturbation based methods . . . 8
2.6.2 Back-propagation based methods . . . 8
2.6.3 Saliency Maps . . . 9
2.6.4 Gradient-weighted Class Activation Mapping (GradCAM) . . . 9
2.6.5 VisualBackProp . . . 10
2.6.6 Evaluating Attribution Methods . . . 11
2.7 Issues with Gradient-Based Attribution Methods . . . 13
2.8 Analysing Black-box Models using Attribution Methods . . . 13
2.9 Simulators . . . 15
2.9.1 CARLA . . . 15
2.9.2 AirSim . . . 16
2.9.3 MIT VISTA 2.0 . . . 17
2.9.4 Comparison of Simulators . . . 19
2.10 Datasets . . . 20
2.10.1 Microsoft's AirSim Tutorial Dataset . . . 20

2.10.2 Berkeley DeepDrive BDD100k . . . 20
2.10.3 Udacity Self-Driving Car Dataset . . . 21
2.10.4 The Cityscapes Dataset . . . 22
2.10.5 From Games Dataset . . . 23
2.10.6 Oxford's Robotic Car . . . 23
2.10.7 Comma.ai comma2k19 dataset . . . 24
2.10.8 Baidu ApolloScape . . . 24
2.10.9 Comparison of Datasets . . . 26
2.10.10 Manual Generation of Simulated Data . . . 27
2.11 Methods to Improve Training Neural Networks . . . 27
2.11.1 GPU Optimisation . . . 27
2.11.2 Image pre-processing techniques . . . 27
2.11.3 Early Stopping . . . 28
2.11.4 Small Initial Random Weights . . . 28
2.11.5 ReLU Layers . . . 29
2.11.6 Drop-out Layers . . . 29
2.12 Real-world usage of end-to-end systems for autonomous vehicles . . . 30
2.13 Safety Concerns . . . 31
2.14 Conclusion . . . 31

3 An Experimental Setup for an End-to-End System for Self-Driving Cars 33
3.1 Introduction . . . 33
3.1.1 The CNN model used in the End-to-End Learning Experimental Setup . . . 33
3.1.2 Two CNN models used for applying visual back-propagation to autonomous vehicles . . . 33
3.1.3 CNN model used in the Autonomous Driving Cookbook . . . 34
3.1.4 A metric of autonomy for self-driving car model . . . 34
3.2 Experimental Setup . . . 35
3.3 Problems with this setup . . . 36
3.3.1 Usability of the model . . . 36
3.4 Conclusion . . . 37

4 Methodology 38
4.1 Introduction . . . 38
4.2 Aim . . . 38
4.3 Hypothesis . . . 38
4.4 Selected CNN Models . . . 39
4.4.1 Simplified CNN model architectures . . . 39
4.5 Temporal Leakage Mitigation . . . 41
4.6 Pre-processing Steps . . . 41
4.7 Analysis of Useful Datasets . . . 41
4.7.1 Udacity Self-Driving Car Dataset . . . 42
4.7.2 Microsoft's AirSim Tutorial Dataset . . . 43
4.7.3 The Cityscapes Dataset . . . 44
4.7.4 The From Games Dataset . . . 48

4.8 Selection of a Viable Epsilon Value for Steering Angle Based on Datasets 50
4.9 Attribution of decisions made by selected models 51
4.9.1 Methods Applied to Analyse Attribution 51
4.9.2 Definition for applying a threshold to the activation map and counting the resultant pixels 53
4.9.3 Definition of the GradSUM analysis scheme 54
4.10 Pilot Study: Should a threshold value be used with an attribution map? 55
4.10.1 Results from the untrained epoch 55
4.10.2 Results from epoch 296 57
4.10.3 Discussions on the results of the pilot study on epoch 1 and 296 . 57
4.11 Pilot Study: Comparing GradSUM and Traditional GradCAM Analysis . 58
4.11.1 Selecting the labels for classification 59
4.11.2 The experimental setup 59
4.11.3 Results of the pilot study 60
4.11.4 Conclusion of the pilot study 63
4.12 Pilot Study: Comparison of Fine and Coarse Segmentation Maps on GradSUM Results 64
4.12.1 Results from epoch 1 (untrained) 64
4.12.2 Results from epoch 249 64
4.12.3 Discussion on the results of the pilot study 65
4.13 Canny Edge Map Results on the Cityscapes Dataset 65
4.13.1 The Resultant profile generated with Canny Edge Maps across the whole *fine* Cityscapes Segmentation Dataset 66
4.14 Training Details 67
4.14.1 Pre-training steps 67
4.14.2 Model initialisation 67
4.14.3 Early stopping conditions 68
4.15 Technical Details of the Experiments 68
4.16 Research Plan 69
4.17 Conclusion 71

5 Experimental Results **72**
5.1 Introduction 72
5.2 Training Results 72
5.2.1 Training Results per Model 73
5.3 Testing Results 77
5.3.1 Random Test Results 77
5.3.2 Test Results per Model 78
5.4 GradSUM Analysis Results 80
5.4.1 Random GradSUM Results 80
5.4.2 GradSUM Results across trained epochs per model 82
5.4.3 Example GradCAM outputs for each model 85
5.4.4 Time taken to Generate Results 86
5.5 Additional Average Experimental Runs 87
5.6 Conclusion 90

6 Discussion **92**
6.1 Introduction . . . 92
6.2 Overall Performance of the End-To-End Systems Based on Results . . . 92
6.3 Evaluation of Performance Per Model . . . 94
6.3.1 NetSVF Behaviour . . . 94
6.3.2 NetHVF Behaviour . . . 95
6.3.3 End-to-End Behaviour . . . 96
6.3.4 Autonomous Cookbook Behaviour . . . 96
6.3.5 TestModel1 Behaviour . . . 97
6.3.6 TestModel2 Behaviour . . . 97
6.3.7 *TestModel1* and *TestModel2* Fixation on *human* Pixel Groups . . . 98
6.3.8 The Difficulty of Predicting Steering Angle from Sparse Data . . . 98
6.4 Cost-Benefit of using the GradSUM system . . . 98
6.5 Future Work . . . 99
6.5.1 Application of other attribution methods . . . 99
6.5.2 Application of attention based models . . . 99
6.5.3 Improving the ground truth dataset . . . 101
6.5.4 Improve the selection of models and their architectures . . . 101
6.5.5 Evaluate different regression tasks and the performance of GradSUM 101
6.6 Conclusion . . . 101

7 Conclusion **103**

A Cityscapes Segmentation Labels and Groups **105**
A.1 Counts of Cityscapes Dataset Pixels by Label . . . 105
A.2 Examples of Each Cityscapes Segmentation Label . . . 106
A.3 Pixel Counts of Each Cityscapes Segmentation Group . . . 107
A.4 Binary Split for Each Cityscapes Label . . . 108

B From Games Segmentation Labels and Groups **109**
B.1 Counts of From Games Dataset Pixels by Label . . . 109
B.2 Examples of Each From Games Segmentation Label . . . 110
B.3 Pixel Counts of Each From Games Segmentation Group . . . 111

References **117**

List of Figures

2.1 Levels of Autonomy as defined by the SAE J3016 standard. Diagram taken from [NHTSA 2017] . . . 6
2.2 An example of a hierarchical autonomous vehicle system [Paden *et al.* 2016] 7
2.3 A general structure of an end-to-end autonomous vehicle system [Bojarski *et al.* 2016b] . . . 8
2.4 Diagram demonstrating the process of GradCAM [Selvaraju *et al.* 2016] . 10
2.5 Diagram demonstrating the process of VisualBackProp [Bojarski *et al.* 2016a] 11
2.6 Cascading randomization results generated by Adebayo *et al.* [2018] . . . 12
2.7 Data randomisation results generated by Adebayo *et al.* [2018] . . . 13
2.8 Reverse engineering approach for explaining black-box models [Guidotti *et al.* 2018] . . . 14
2.9 A street shown from a third-person view with four weather conditions [Dosovitskiy *et al.* 2017] . . . 16
2.10 A snapshot of an example configuration available in AirSim . . . 17
2.11 Example of augmented data produced by VISTA 2.0 [Gordon 2022] . . . 17
2.12 Sample images pre and post-cropping included in the AirSim Tutorial Dataset [Spryn and Sharma 2018] . . . 20
2.13 Four examples of the types of data available in the Berkeley DeepDrive BDD100k overlaid on top of still images captured from video in the dataset [Yu *et al.* 2018] . . . 21
2.14 Samples taken from challenge #2 of the Udacity Collection [Cameron 2016] 21
2.15 Example data from The Cityscapes Dataset. The top image is a blurred example RGB frame from the Berlin sub-set. The bottom left image is an example of a *fine* segmentation map from Stuttgart, and the bottom right is an example of a *coarse* segmentation map from Nuremberg [Cordts *et al.* 2016] . . . 22
2.16 Example data from the From Games dataset. The left side represents two front facing image samples from the dataset, with the corresponding segmentation label on the right [Richter *et al.* 2016b] . . . 23
2.17 To the left is a 3D map produced by the Lidar data present in the Oxford's Robotic Car dataset, and to the right are some example images from the same dataset in a few different conditions [Maddern *et al.* 2017] . . . 23
2.18 Sample highway image stills are taken from the *comma2k19* dataset [Schafer *et al.* 2018] . . . 24
2.19 Some examples of images from the ApolloScape dataset [Wang *et al.* 2018] 24

2.20 Example Neural Network With and Without Drop-out applied to Successive Layers [Baeldung 2023] . . . 30
2.21 An example of a shared architecture and RGB scene used by Tesla, Inc. for their full self-driving AI models [Karpathy 2019] . . . 31

3.1 A diagrammatic representation of the model used in the article, End-to-End Learning for Self-Driving Cars [Bojarski *et al.* 2016b] . . . 34

4.1 Comparison of a Udacity Dataset Sample Before and After Cropping the ROI [Cameron 2016] . . . 43
4.2 Comparison of a AirSim Tutorial Dataset Sample Before and After Cropping the ROI [Spryn and Sharma 2018] . . . 44
4.3 Comparison of a Cityscapes Dataset Sample Before and After Cropping the ROI [Cordts *et al.* 2016] . . . 45
4.4 Scaled Count of all Segmentation Groups Found in Cropped and Scaled Cityscapes Segmentation Data . . . 46
4.5 Examples of each group from segmentations. Three images are shown: the original cropped RGB image, the cropped segmentation map, and the binary map of the group . . . 47
4.6 Comparison of a Cityscapes Dataset Sample Before and After Processing Segmentation [Cordts *et al.* 2016] . . . 47
4.7 Comparison of a From Games Dataset Sample Before and After Cropping the ROI [Cordts *et al.* 2016] . . . 48
4.8 Scaled Count of all Segmentation Groups Found in Cropped and Scaled From Games Segmentation Data . . . 49
4.9 Examples of each group from segmentations. Three images are shown: the original cropped RGB image, the cropped segmentation map, and the binary map of the group . . . 50
4.10 Example gradient maps produced by GradCAM [Selvaraju *et al.* 2016] . . . 52
4.11 Effect of threshold value on the spread of signal for epoch 1 . . . 56
4.12 Effect of threshold value on the spread of signal for epoch 296 . . . 57
4.13 GradSUM Results (Percentages per Group) for 100 Random Models Solving the Binary Classification Problem (Averaged Results per Group) . . . 61
4.14 Comparison of the GradSUM Model Profiles for the Trained *End-to-End* Models (Using The Cityscapes Dataset) . . . 61
4.15 Comparison of the GradSUM Model Profiles for the Trained *End-to-End* Models (Using From Games Dataset) . . . 63
4.16 Comparison of different segmentation types (model profiles) for Epoch 1 (untrained) using GradSUM . . . 64
4.17 Comparison of different segmentation types (model profiles) for Epoch 249 using GradSUM . . . 65
4.18 Example of a Canny Edge Map Generated on the Cityscapes Dataset . . . 66
4.19 Resultant canny edge group profile . . . 67
4.20 Flow diagram of experimental flow followed in the analysis of end-to-end models . . . 70

5.1 Comparison of training and validation loss per model (over each epoch) for *NetSVF* and *NetHVF* Architectures . . . 73
5.2 Comparison of training and validation loss per model (over each epoch) for the *End-to-End* and *Autonomous Cookbook* Architectures . . . 74
5.3 Comparison of training and validation loss per model (over each epoch) for TestModel1 and TestModel2 Architectures . . . 74
5.4 Comparison of Model Train Loss and Validation Loss . . . 75
5.5 Comparison of Model Train Loss and Validation Loss . . . 77
5.6 Spread of results from running the Udacity Self-Driving Car Dataset on Models with small Random Weights . . . 78
5.7 Comparison of Model Autonomy over Three Datasets . . . 78
5.8 Comparison of randomly initialised versions of each model architecture . 81
5.9 Comparison of model profiles where the models are trained to their best epoch . . . 83
5.10 Examples of a single GradCAM export over each model over a series of epochs, where the models have been trained on the Udacity dataset . . . 85
5.11 Comparison of Time Taken by Each Model Architecture . . . 87
5.12 Best Selected Models Over All 10 Runs and Each Architecture . . . 88
5.13 GradSUM box-and-whisker profile plots for each model architecture . . . 89

6.1 Figures Related to the Overall Performance of End-To-End Systems . . . 93
6.2 Example attention results of input images using a vision transformer [Dosovitskiy *et al.* 2020] . . . 100

A.1 Scaled Counts of Segmentation Labels from Cityscapes Segmentation Data 105
A.2 Examples of Labels from Cityscapes. White represents the label. The original cropped frame, the segmentation map and the label map are shown 106
A.3 Examples of Labels from Cityscapes. White represents the label. The original cropped frame, the segmentation map and the label map are shown 107

B.1 Scaled Counts of Segmentation Labels from From Games Segmentation Data 109
B.2 Examples of Labels from From games. White represents the label. The original cropped frame, the segmentation map and the label map are shown 110
B.3 Examples of Labels from From Games. White represents the label. The original cropped frame, the segmentation map and the label map are shown 111

List of Tables

2.1 Comparison of the different proposed simulators [Microsoft Research 2018; Dosovitskiy *et al.* 2017; Amini *et al.* 2022] 19
2.2 Comparison of the different proposed datasets [Yu *et al.* 2018; Cameron 2016; Spryn and Sharma 2018; Wang *et al.* 2018; Schafer *et al.* 2018; Maddern *et al.* 2017; Cordts *et al.* 2016; Richter *et al.* 2016a] 26

4.1 Comparison of the six different CNN models [Bojarski *et al.* 2016ab; Spryn and Sharma 2018] (From largest structure to smallest structure) 40
4.2 Udacity Dataset Summary 42
4.3 AirSim Tutorial Dataset Summary 43
4.4 Cityscapes Dataset Summary 44
4.5 Table of Cityscapes labels as divided per group [Cordts *et al.* 2016] . . . 45
4.6 From Games Dataset Summary 48
4.7 Steering Ratio per Dataset for Different Accuracy Decimal Places 50
4.8 Comparison of each Cityscapes segmentation group sample counts for the simple binary classification problem 59
4.9 Loss results from the trained models for the binary classification problems 62

5.1 Results of the Best Model Epochs During Training 75
5.2 Comparison of Autonomy Results Per Trained Model 80
5.3 Comparison of Loss (MSE) Results Per Trained Model for Best Epoch . . 80
5.4 Time that was taken to train, evaluate (test) and analyse model architectures 87

A.1 Pixel Counts per Group from Cityscapes 107
A.2 Comparison of each Cityscapes segmentation label sample counts for the simple binary classification problem 108

B.1 Pixel Counts per Group from From Games 111

Glossary of Terms

AI Artificial Intelligence. ii, 1–5, 7, 20, 27, 30–36, 92, 94, 101

CNN Convolutional Neural Network. ii, 2–4, 7–9, 14, 15, 22, 29, 33–36, 38–40, 58, 63, 71, 77, 99, 100, 103

DARPA Defense Advanced Research Projects Agency. 5

GB Gigabyte. 68

GPU Graphics Processing Unit. 27, 68, 69

GradCAM Gradient-weighted Class Activation Mapping. ii, 3, 4, 9, 10, 13, 14, 38, 44, 52–55, 58, 64–66, 69, 71, 85, 86, 98–100, 103

GradSUM Gradient Summation. ii, 3, 4, 38, 41, 54–61, 63–65, 68, 69, 71, 72, 76, 77, 80–84, 87–90, 93–104

IoU Intersection Over Union. 14

MoA Metric of Autonomy. ii, 3, 4, 34, 35, 50, 68, 72, 76, 78, 79, 87, 90, 95

MSE Mean Squared Error. ii, 3, 4, 38, 68, 72, 73, 77, 79, 80, 87, 90, 93–95

NHTSA National Highway Traffic Safety Administration. 15, 31

NLP Natural Language Processing. 99, 100

ReLU Rectified Linear Unit. 9, 10, 29, 39

RGB Red Green Blue. ii, 2, 3, 22, 23, 26, 30, 31, 35, 38, 39, 44, 47, 50, 51, 60, 66, 71, 103

ROI Region of Interest. 27, 43–49, 67

ROS Robotic Operating System. 16

SAE Society of Automotive Engineers. 6

ViT Vision Transformer. 99, 100

Chapter 1

Introduction

Autonomous vehicles, also known as self-driving cars, promise to make modern roads safer and more efficient [Liu *et al.* 2016]. According to Paden *et al.* [2016], implementing autonomous vehicles could meaningfully reduce annual road deaths. This technology could also be used to help those who cannot drive for themselves.

This kind of autonomous system needs to run in a real-world environment with other vehicles on the road controlled by both people and other Artificial Intelligence (AI) systems. This means there is a need to understand how such a system works and, when problems occur, be able to figure out how the AI system made the decisions which led to the wrong outcome. It is also essential to understand such an AI system to improve it over time. This kind of system must have a minimal margin of error because of the danger to the public due to incorrect decisions being made [Fridman *et al.* 2017].

There are also potential legal requirements for understanding and reasoning why an AI system made a specific decision when operated around people or property. When accidents happen, this reasoning will be required to find out who is potentially liable and what needs to be fixed or changed to prevent future accidents from arising.

A recent Forbes article reports that Tesla, Inc. had to recall their autonomous self-driving beta software due to concerns over safety and increasing crash risks for the public [Forbes 2023].

There are two overall taxonomies in building an autonomous vehicle system; a segmented pipeline of interactive modules (hierarchical structures) and a monolithic structure (end-to-end structures) [Paden *et al.* 2016].

The hierarchical structure is an older approach to designing an AI system for driving. Paden *et al.* [2016] describes it as a piecemeal design where the completed system is made up of sub-systems or modules. Each module has a different responsibility and passes information back and forth to other modules as needed. This design allows for various techniques and technologies to interact and work together. It also allows for better logging and monitoring of each part of the system, which helps to understand what decisions are being made at a module level. This allows for the allocation of blame to a single module.

However, these systems are hand-crafted, therefore more time-consuming to create and modify due to having complex and numerous components.

The end-to-end structure is a more modern approach. Rather than having hand-crafted sub-modules, a black-box AI technique such as a deep Convolutional Neural Network (CNN) is used [Nie *et al.* 2018]. This structure uses more powerful computers and readily available datasets to train a numerical model to drive a vehicle. This makes it harder to explain why a particular decision was made. It also makes modifying or improving such a system harder once implemented and trained. This approach does not contain the interconnected complexity of the hierarchical model. This approach does however introduce its own complexities in the form of a deeply connected set of trainable parameters which are trained by an algorithm rather than being hand-crafted. Antipov *et al.* [2015] has shown that in many cases, end-to-end systems can outperform the more hand-crafted approaches in predicting outcomes from extracted features in a large dataset. The human effort in getting this kind of model working in a real environment is lower because of readily available and open datasets in the field [Bojarski *et al.* 2016b].

A problem with end-to-end systems is the black-box nature of the decision-making process. In a modular system, blame can be apportioned to a particular module or intersection of modules. This is not possible in an end-to-end system. Attribution methods are required to explain decisions made in these kinds of models. There are state-of-the-art attribution methods which promise to make these models more understandable in the future [Ancona *et al.* 2017].

A reduced problem domain in the field of autonomous vehicles is a task which has sparse input data and a sparse training signal. This setup is comprised of input data from a single forward-facing camera and is trained to predict the vehicle's steering angle. It is a useful minimum problem domain to test out techniques that can then be scaled up to a bigger self-driving setup [Bojarski *et al.* 2016b].

Bojarski *et al.* [2016b] using a single attribution method evaluated manually, found that a CNN model trained in this reduced domain could extract meaningful semantic information from the input data (information such as road markings). This research shows that by using attribution techniques incorporated into an automatic evaluation scheme, some CNN deep-learning models are able to extract meaningful information from the input data but other architectures failed to do so. These architectures were far more simple in design than the ones which performed as expected. This showed that the depth of feature extraction internally to the model and the quality of the extraction (as well as the quality of training the model) has a direct impact on what feature extraction the model is able to perform.

Attribution methods can be used to determine what part of the input data has the highest activation or effect on the outcomes of a neural network model. These methods produce an activation map (also known as a heat-map). This result has the same dimensional size as the input data but with values representing the importance or magnitude of a pixel on the model being evaluated. Most of these methods are used in classification problems to identify what part of the input data is being used to identify a particular category [Selvaraju *et al.* 2016].

When using attribution methods with regression problems, such as predicting steering angle from a red green blue (RGB) image, it is more difficult to determine at scale (across many frames extracted from a driving video sequence) what the deep-learning AI model is evaluating in the given input.

This research uses a ground truth dataset of forward facing RGB images from a video sequence of driving with equivalent segmentation maps of those images. The segmentation maps delineate different objects of note in each frame. These maps are then applied to the computed activation maps (attribution heat-maps) to determine which pixels of the input are associated with which category from the ground truth data to show their relative impact on the predictions made by the AI model.

We evaluate multiple AI model architectures and the performance of these models using synthetic metrics such as mean squared error (MSE) from the expected and predicted outputs (steering angle) as well as an autonomy metric (MoA) measuring how much of the time the AI model is autonomous when being evaluated against real data. We perform a further analysis on these models where attribution methods are applied to determine the model behaviours over time by attributing which elements in the ground truth dataset has the largest impact on the predictions made by these models.

This research reproduces the experimental setup of Bojarski *et al.* [2016b] (including other model architectures from literature) and determines that some deep-learning CNN model architectures are able to learn meaningful information from a sparse input signal. However some of the analysed architectures are unable to learn meaningful features from the given input data.

This research shows that the model architecture plays a vital role in what categories each model focuses on when trained and that some categories have more importance in causing model weight activation than other categories. Some model architectures can learn to observe more meaningful parts of the input, whilst others may fixate on less meaningful parts. Simple architectures tend to over-fit whilst training and focus on spurious correlations.

Another noteworthy result of this work is that understanding an end-to-end system requires evaluation of multiple metrics. When looking at only one metric such as MSE, it can lead to incorrect conclusions on the performance of a specific model since it may perform well in a synthetic test over a large dataset but poorly on important edge cases.

This work's main contribution is in devising a statistical scheme we call GradSUM to help identify, with the use of a ground-truth dataset and an attribution method (GradCAM), which components of the forward facing input images of the ground truth datasets each evaluated model architecture focuses on throughout their training and evaluation phases. This helps us deduce and compare, using averaged results, the behaviours of these model architectures.

This work has been open-sourced and can be found on GitHub[1].

[1] https://github.com/TRex22/masters-gradsum

1.1 The core objectives of this research

- To recreate the experimental setup and model architecture used by Bojarski *et al.* [2016b].
- To recreate other model architectures defined by Bojarski *et al.* [2016a]; Spryn and Sharma [2018].
- Introduce two simple CNN architectures based on the same core design of the model architectures found in literature. These models are used as controls to see if the model architecture's complexity has an effect on the outcomes of the experiments and analyses we perform in this work.
- To introduce a performance metric to rate the autonomy of each model based on the work by Bojarski *et al.* [2016b]. This metric is used to compare the theoretical real-world performance of each model architecture after training.
- To introduce an attribution analysis scheme we call GradSUM which produces a model profile based on the statistical analysis of many GradCAM results generated off a ground truth dataset (The Cityscapes Dataset). This ground truth dataset contains high quality segmentation maps we use to identify which pixels belong in which groups of segmentation across all GradCAM results that are generated.
- To compare the results of the trained models in terms of their GradSUM analysis, autonomy metric (MoA), and their MSE loss. This comparison allows us to make observations and conclusions on the behaviour and performance of each model architecture.
- To show that some of the model architectures once trained, are able to learn meaningful road features as Bojarski *et al.* [2016b] observed, but that other model architectures (even once trained) are unable to learn meaningful road features in the same way.
- To open up the possibility of future work looking at applying other analysis techniques, to different kinds of deep learning models, and in different problem domains to improve explainable AI.

Chapter 2

Background

2.1 Introduction

This section introduces the background information, methods, techniques and available resources that will be used to produce an experimental analysis of multiple end-to-end models and their behaviours. There are also sections of relevant background in the field to give proper context to the proposed research and application of available methods.

2.2 Historical Work

The idea of a self-driving car started in the 1920s [Paden *et al.* 2016]. Research began with radio-controlled vehicles and followed radio and magnetic control circuitry developments. From the late 1960s onwards, research began into rudimentary computer control via special roadways embedded with magnetic strips [Gringer 2019]. Developments in computing around algorithms and AI models began in the 1980s and 1990s. This eventually led to the development of the modern autonomous vehicle [Paden *et al.* 2016]. The 1980s saw the introduction of very early vision systems, neural networks, and Defence Advanced Research Projects Agency (DARPA) funding for the application of this research to produce autonomous vehicles. In the early 2000s DARPA began funding a grand challenge which sought to demonstrate complete autonomous control of a vehicle. There were a few similar events sponsored by DARPA, and their outcomes showed that a completely autonomous vehicle was a tough challenge [Paden *et al.* 2016]. By the late 2000s and early 2010s, academic, commercial, and investment interest had grown in autonomous vehicle research and development. This has produced cheaper and more readily available computational power, storage, and interconnectivity of devices [Paden *et al.* 2016]. This increase in readily available AI models has bolstered the speed of autonomous vehicle research. However, there is a growing need to explain how these AI models operate and what information they focus on when making decisions [Bojarski *et al.* 2016a].

2.3 A Scale of Autonomy

The Society of Auto-mobile Engineers (SAE) defined a scale of how autonomous a vehicle is, which is defined by the J3016 standard. This standard defines five levels of autonomy as seen in Figure 2.1 [Paden *et al.* 2016].

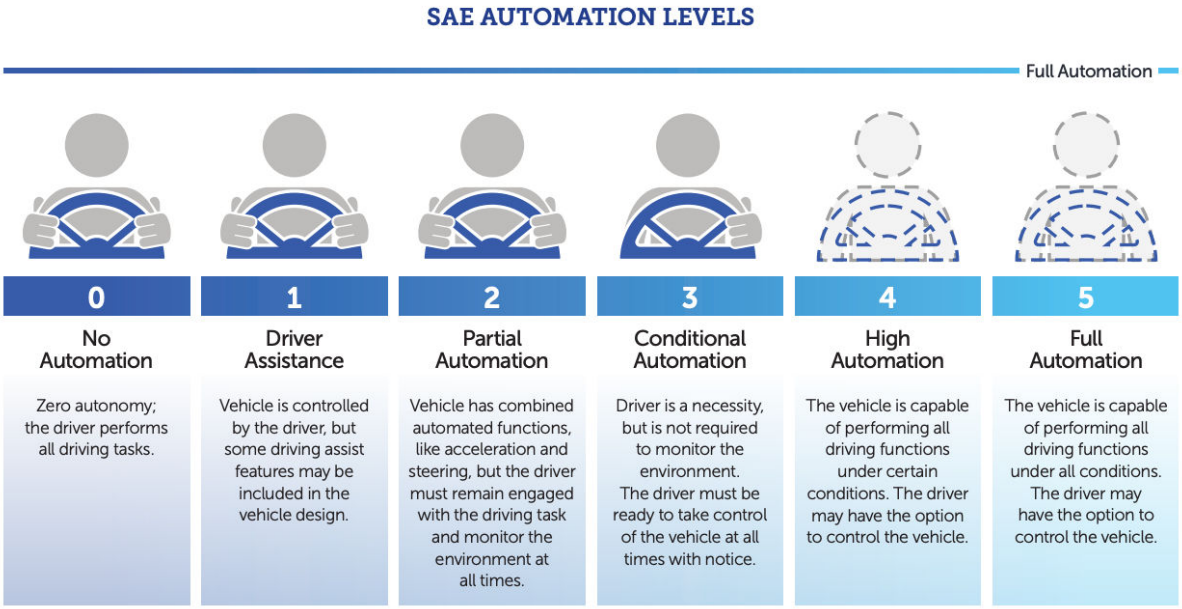


Figure 2.1: Levels of Autonomy as defined by the SAE J3016 standard. Diagram taken from [NHTSA 2017]

Examples of the kind of autonomy a vehicle can exhibit at each level [Paden *et al.* 2016]:

- Level 1: is fully human-controlled (all driving tasks)
  - Fuel ratios are manually controlled in very early motor vehicles
- Level 2: has basic automatic assistance. Examples include:
  - Adaptive cruise control
  - Anti-locking brakes
  - Adaptive fuel and air control
- Level 3: Is a partial form of autonomy where under certain conditions, the system can drive but a human is still the operator
  - Automatic parking
- Level 4: Full autonomy, where the operator can still accept control and take over (still has conditions)
- Level 5: The system is fully autonomous under all conditions

2.4 Hierarchical Structures

A hierarchical structure can be seen as a pipeline of modules with inputs and outputs that interface to produce an overall autonomous vehicle AI system [Paden *et al.* 2016]. Figure 2.2 is an example of such a system. The decisions made by this AI may be encoded into the system on a per module basis. This methodology also allows multiple competing methods to be interfaced with the same final decision mechanism for potentially better results. Such a system can be implemented to log and attribute actions to specific inputs and modules for review in real time or in the future. Many examples of such architectures involve considerable manual hand-crafted work to optimise the independent modules and the overall structure of the intelligent system [Paden *et al.* 2016]. According to Gómez *et al.* [2010] this was a popular structure to employ when there was a shortage of available open datasets. Paden *et al.* [2016] adds that a lack of access to cheap, efficient, and easily accessible computation and data also led to the use of hierarchical structures.

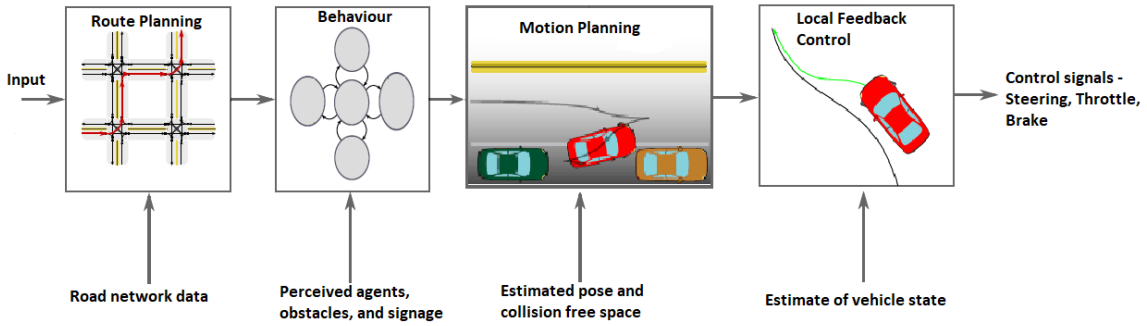


Figure 2.2: An example of a hierarchical autonomous vehicle system [Paden *et al.* 2016]

2.5 End-to-End Structures

Bojarski *et al.* [2016b] defines an end-to-end structure as a black-box in how it learns and operates. It is a deep neural network of some design - usually a CNN model. The system is trained off a large dataset using high-performance computing. Figure 2.3 is an example of such a structure in the context of a self-driving vehicle. Weng [2019] states that combining this kind of system as a sub-module in a hierarchical structure is possible. From a high-level overview, this kind of system takes in some kind of input (usually pre-processed for optimal performance) and returns control signals to the autonomous vehicle.

A limitation of this approach, according to Li [2017] is that it is hard to understand what feature extraction from the input data is occurring inside the network. Reasoning about specific emergent behaviours the system may demonstrate is also tricky.

Desirable behaviour has to exist in the training dataset. Datasets have to be created with care not to introduce over-training and undesirable behaviour into the model network. Desirable outcomes for a specific environment include encoding specific road rules in the actions taken by drivers [Spryn and Sharma 2018].

A benefit of this approach is that it can ingest a large amount of data without much human interaction and produce meaningful results [Bojarski *et al.* 2016b].

However, this is not a reinforcement method where a reward function can be used to automatically influence the network's training. There are attempts to apply reinforcement learning to the training of this class of system architecture. However Weng [2019] states that it has proven difficult to scale to real-world driving scenarios in real-time. A significant limiting factor is the availability of large amounts of naturalistic (real-world) data samples and the ability to process them on the fly [Weng 2019; Fridman *et al.* 2017].

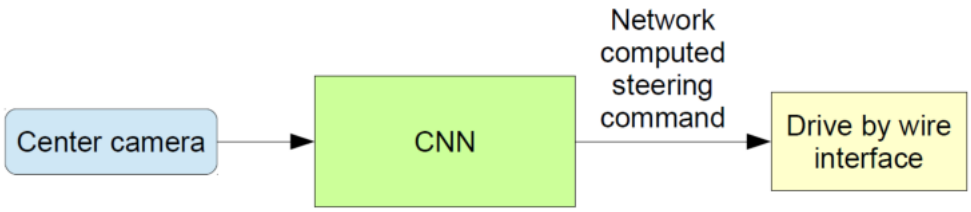


Figure 2.3: A general structure of an end-to-end autonomous vehicle system [Bojarski *et al.* 2016b]

2.6 Deep Learning Attribution Methods

Ancona *et al.* [2017] states that deep neural networks have many applications. By their nature, they are complex and hard to understand. In order to produce predictable results understanding or attributing the feature extraction of these models is required. Attributions are visualised as heat-maps of relevant features from the input - especially with image based input data.

2.6.1 Perturbation based methods

Ancona *et al.* [2017] describe the class of perturbation methods as a type of function which looks at the difference between an original network trained on the input data and a second identical network trained on the same input data but where the relevant features have been altered, such as being masked, obscured, removed or biased. The example that Ancona *et al.* [2017] tested uses a form of occlusion where grey squares occlude portions of each input image.

2.6.2 Back-propagation based methods

Back-propagation based attribution methods involve both forward and backward passes through a network to compute the contribution of input features to the output of the network [Ancona *et al.* 2017; Nie *et al.* 2018].

In CNN models, convolutional layers retain spatial information captured from the input, allowing for later convolutional layers in a CNN model to capture more abstract but

relevant representations over time. This property allows later convolutional layers to represent relevant input features for their corresponding output [Selvaraju *et al.* 2016].

This class of method usually uses gradient-based numerical techniques in the back-propagation step since back-propagation is itself gradient based [Ancona *et al.* 2017].

Gradient-weighted Class Activation Mapping (GradCAM) is an example of a gradient-based activation method which uses back-propagation to generate localisation maps which highlight important regions in the input for a given target class [Selvaraju *et al.* 2016].

2.6.3 Saliency Maps

Saliency maps are also known as attribution maps, are calculated by taking the absolute partial derivative of the output neuron with respect to the input features.

Many methods can be considered as saliency techniques such as Gradient-weighted Class Activation Mapping (GradCAM) [Ancona *et al.* 2017; Simonyan *et al.* 2013].

Saliency maps are defined as:

$$map = \frac{\partial \text{output}}{\partial input} \tag{2.1}$$

According to Ancona *et al.* [2017], saliency maps have issues determining input features which have a negative effect on the output due to the absolute value used. Also the maps produced are very sensitive to noise in the network.

2.6.4 Gradient-weighted Class Activation Mapping (GradCAM)

Selvaraju *et al.* [2016] defines GradCAM as a method that looks at the gradient information in the last layer of the CNN model to determine the importance of the neurons in that layer of the network to the final decisions or output of the network. The full process of this method is shown in Figure 2.4.

This method applies a global average pooling on the gradients calculated in the backward step and obtains a metric of each neuron's importance in the layer. Global average pooling is the average output of each feature map from the previous step. This can be defined as:

$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A_{ij}^k}}_{\text{gradients via backprop}} \tag{2.2}$$

where c is the class of the localisation map $\mathbb{R}^{u \times v}$ with width u and height v, y^c is the score of the class c, k is a feature map, A^k are the feature maps of a convolutional layer, and α_k^c are the neuron importance weights. α_k^c represents the importance of the feature map k for a target class c.

The final heat-map is produced by using the weights calculated in the previous step to generate a linear combination with the forward activation maps, which is then passed

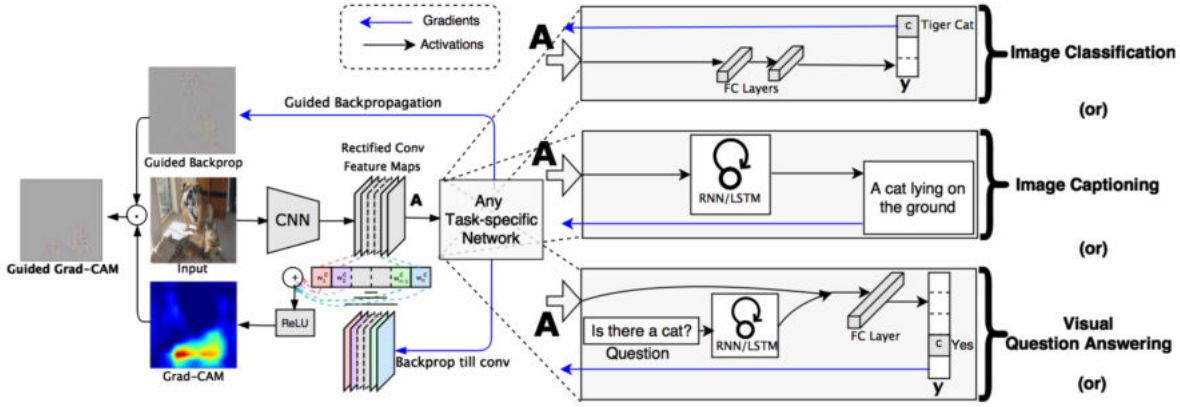


Figure 2.4: Diagram demonstrating the process of GradCAM [Selvaraju *et al.* 2016]

into a ReLU layer. This function returns only positive and zero values. This is defined mathematically as:

$$L^c_{\text{Grad-CAM}} = \text{ReLU}(\underbrace{\sum_k \alpha^c_k A^k}_{\text{linear combination}}) \tag{2.3}$$

This heat-map is coarse and has the same dimensionality as the convolutional feature map. ReLU is used only to capture positive pixels since the positive pixels directly influence the outcome of the network.

Guided GradCAM seeks to combine Guided Back-propagation and GradCAM using point-wise multiplication to create higher-resolution heat-maps to show the finer-grained importance of pixels [Selvaraju *et al.* 2016].

2.6.5 VisualBackProp

VisualBackProp builds on the basic premise of GradCAM by combining the feature maps from the deeper convolutional layers close to the output with the feature maps of the shallow layers. Figure 2.5 demonstrates this process. The deeper layers tend to be a much lower resolution but higher in activation importance for the specific output layer of this network's forward run. The more shallow layers tend to have much higher resolution but have more degrees of separation from the output layers and therefore have much less activation importance [Bojarski *et al.* 2016a; Selvaraju *et al.* 2016].

Steps of the VisualBackProp Algorithm [Bojarski *et al.* 2016a]:

- A forward propagation pass is initially completed on the network
- A backward pass is conducted
- The feature maps produced after each ReLU step for each layer are collected and averaged (per layer)

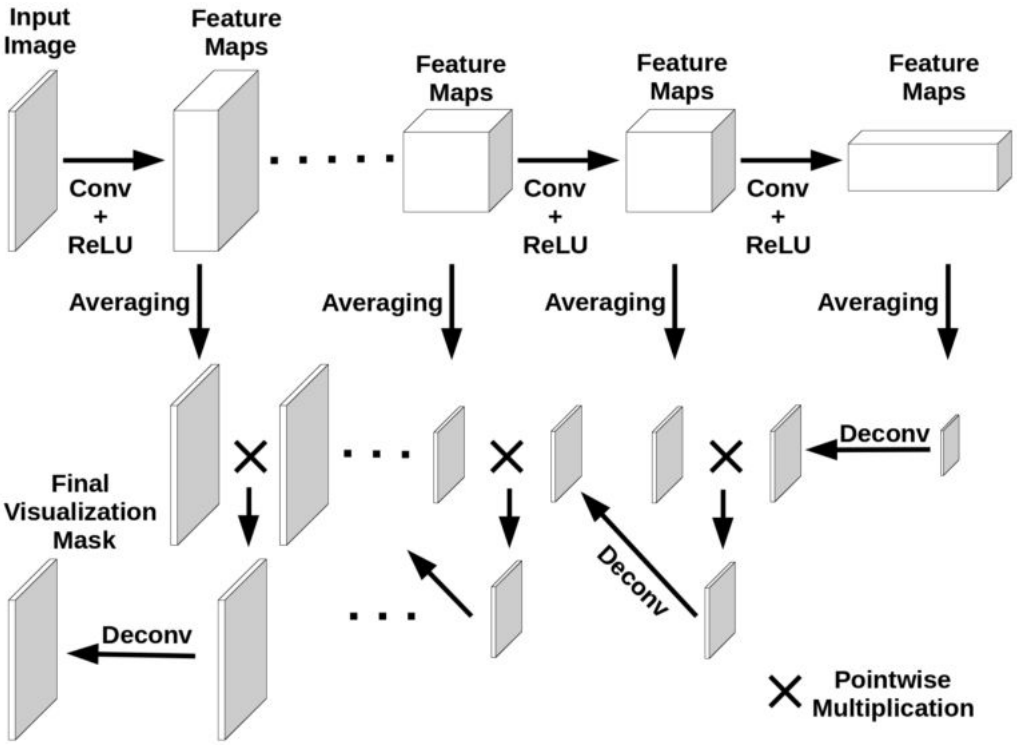


Figure 2.5: Diagram demonstrating the process of VisualBackProp [Bojarski *et al.* 2016a]

- The deepest layer is used
- The scaled-up averaged feature map is point-wise multiplied by the average feature map from the previous layer
- This process continues until the network's input is reached
- The final result is a mask which has the same dimensionality as the input, which is then binarised

2.6.6 Evaluating Attribution Methods

Bojarski *et al.* [2018] proposes the use of a visualisation method of analysis. The method produces an image map against the input image, which can be used to generate a step-wise (per epoch) visualisation of the network. Adebayo *et al.* [2018] describe two ways of generating the final visualisation. The first is taking the absolute value of a normalised map, and the second is showing the map's positive and negative values in different colours.

Adebayo *et al.* [2018] describe current assessment methods of gradient-based attribution techniques where perturbation techniques are applied to assess the quality of the other techniques. A metric based on the entropy of the results can be used. Another approach is to compute the similarity of the gradient maps produced between the methods.

Adebayo *et al.* [2018] continue by proposing two tests to determine the independence of attribution methods to model parameters and data. These tests are based on the work of Nie *et al.* [2018].

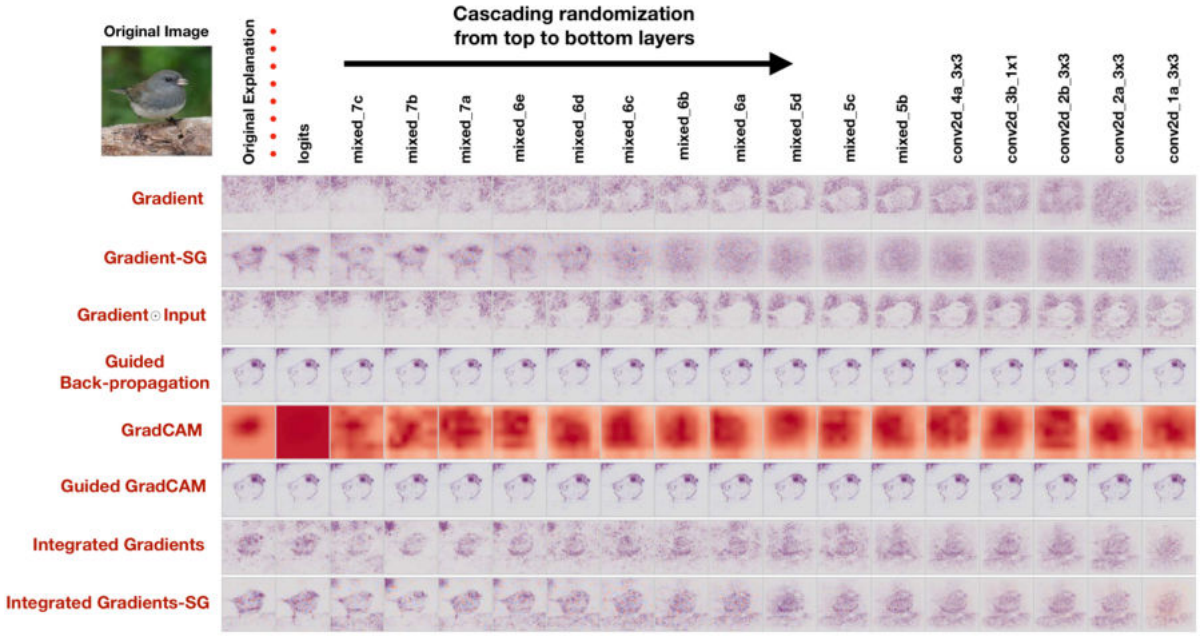


Figure 2.6: Cascading randomization results generated by Adebayo *et al.* [2018]

Model Parameter Randomization Test

The first test is called the *Model Parameter Randomization Test*. The test contains two steps. The first step is called *cascading randomization* where the weights of a model are randomised over time starting from the top layers and moving down to the bottom ones. Figure 2.6 shows the results of *cascading randomization*. This test shows the sensitivity of an attribution method to the model's parameters.

The second step is called *independent randomization*. This is done by performing randomisation layer-by-layer rather than by weight. This gives a more granular indication of dependency for an attribution method by the order of the layer.

Data Randomization Test

The second test is called the *Data Randomization Test*. This test seeks to determine the relationship between the attribution method and the correlation between the labels and the input data (usually images). This test is done by randomising the labels so that the model is randomly guessing its predictions.

The results of this test, as seen in Figure 2.7, show that visual inspection alone may mislead. Guided BackProp appears to have a visual change and may seem reasonable to the input data, but the network has been randomly guessing - so no such correlation should be present [Adebayo *et al.* 2018].

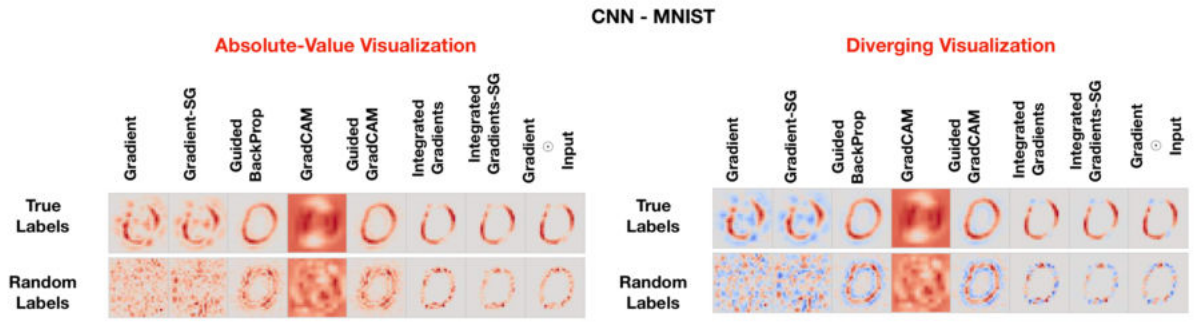


Figure 2.7: Data randomisation results generated by Adebayo *et al.* [2018]

2.7 Issues with Gradient-Based Attribution Methods

Nie *et al.* [2018] show that certain attribution methods, particularly ones in the Back-propagation class of methods, partially reconstructed the input data rather than attribute trained weights and decisions of the network to features in the input. Theoretical proofs using linear models are proven to be too simplistic for these kinds of attribution techniques.

Adebayo *et al.* [2018] describes issues with specific saliency map methods where they are independent of the input data and the model parameters. These specific maps, therefore cannot be used to find a relationship between the input data and output nodes of a deep neural network. They also found that guided back-propagation and guided GradCAM fail to have a relationship between the input data and output nodes of a network and therefore are inadequate to be used to explain the feature extraction process of a neural network. However, GradCAM (not guided) passed their analysis checks.

Ancona *et al.* [2017]; Adebayo *et al.* [2018] describe visualisation methods as being brittle to noise, interference, and easy to manipulate. Manual visualisation analysis is shown to be inadequate without the application of other metrics and can by themselves lead to misleading results.

VisualBackProp, as described by Selvaraju *et al.* [2016], was not analysed to determine if it was accidentally reconstructing input data or if it is independent of input data or the model parameters.

2.8 Analysing Black-box Models using Attribution Methods

Guidotti *et al.* [2018] define the ability to interpret a black-box model as a process to clarify to a human the meaning and explanation of what and how the model produces its predictions. Certain models such as decision trees, conditional logic and linear models, are considered interpretable by humans. The more complex a model is, the more difficult interpretability will be. There are two types of interpretability - global and local. Global interpretability means that the model can be interpreted as a whole, whilst with local

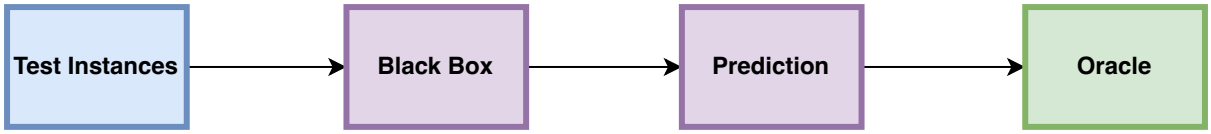


Figure 2.8: Reverse engineering approach for explaining black-box models [Guidotti *et al.* 2018]

interpretability, only a portion of a model is interpretable, or the model is interpretable within a specific set of input or output data within a confined set of parameters.

Guidotti *et al.* [2018] mention several types of explanation that are possible with black-box models. The first is *model explanation*, where an existing interpretable model is trained to mimic the black-box and its interpretations are used. This does not help with explaining and comparing black-box models. Another approach is called *outcome explanation*, where the outcome of a specific input sample is explained. This is a form of localised interpretability. A third method of explanation is called *model inspection*, where a specific property or behaviour of a model is observed and used as a basis for interpreting the model's behaviour. An example would be the perturbation study performed by Adebayo *et al.* [2018] in their work on sanity checks for saliency maps.

There are many different methods for interpreting different kinds of models and different problem spaces. This research focuses on a deep learning CNN model using activation masks and sensitivity analysis of the model's behaviour as described by Guidotti *et al.* [2018]. The selected method to be used is GradCAM as defined by Section 2.6.4. This would be considered the reverse engineering approach, according to Figure 2.8.

Saliency and activation maps are types of masks (or maps) which identify the points in the input data (usually pixels in an image) of what the model is "looking" at, according to Guidotti *et al.* [2018]. These methods are localised explainer methods which generate a second-order result (the map) from the evaluation of a model. These results can be used to attribute what the model focuses on in the input. Some maps define features in the input, and others define the regions causing activations in the model.

When performing analysis of a deep learning model using a heat-map, areas of importance are usually highlighted by the maps used in the analysis, such as in the work conducted by Barkan *et al.* [2021].

The classes or features present in the input data are usually used for the analysis. Barkan *et al.* [2021] produced their own gradient-based algorithm to explain predictions and classifications made by models. They analyse the performance of their method by subjectively evaluating the generated explanation maps. Barkan *et al.* [2021] also perform an objective evaluation by computing confidence scores on the model's performance using the original image and the explanation map. Barkan *et al.* [2021] generates a similarity dataset, where pairs of similar images are selected and given the same ground truth classes. This is then used to quantitatively determine if the confidence scores matched over similar images from the generated dataset. Other metrics, such as intersection over union (IoU) are also used to compare the results of different algorithms in Barkan *et al.* [2021]'s work.

2.9 Simulators

There are many different kinds of simulators in the field of autonomous vehicles. Some focus on specific sub-domains, while others attempt to create a general simulated environment to test end-to-end autonomous driving. In order to test a CNN model where the input data is relatively sparse, more photo-realistic data is desired because it will better reflect a real-world environment [Weng 2019].

Simulators are also useful in generating specific, targeted, and biased data encoded with specific rules [Shah *et al.* 2018]. This data can then be used to train a model, and the same simulator can test, observe, and analyse that model. Simulated data can also be used alongside real-world data to provide better results when combined effectively [Weng 2019].

Table 2.1 compares the features of the simulators discussed in this section.

2.9.1 CARLA

CARLA is a simulator that focuses on building custom environments. It also freely provides assets that can be used to build these kinds of environments [Dosovitskiy *et al.* 2017]. Recently, traffic scenarios were also added to the simulation. It is built on top of *Unreal Engine 4* and uses the engine's photorealism effects. This allows the weather and other environmental attributes to be adjusted and controlled. Figure 2.9 shows some examples of different weather conditions which can be simulated.

As of 2024, CARLA is actively being maintained and developed [CARLA Simulator 2023].

CARLA also contains a driving benchmark and leader-board based on the topology of driving situations defined by the NHTSA [CARLA Autonomous Driving Leaderboard 2023]. These situations include higher-level tasks such as lane merging, obstacle avoidance, and vehicle parking. They do not include low-level tasks such as controlling steering angle when driving on a curved road.



Figure 2.9: A street shown from a third-person view with four weather conditions [Dosovitskiy *et al.* 2017]

2.9.2 AirSim

AirSim is intended to be a platform for generating data for both vehicles and drones. It is a simulator that provides many different environments with their own specific focus and sensor configurations for different kinds of data. Many of the environments focus on photorealism [Shah *et al.* 2018]. There are also environments for factories, tunnels, and faces [Microsoft Research 2018].

The simulator is made for compatibility and works with many robotics frameworks such as the Robotic Operating System (ROS), direct control from a terminal (i.e. using Python) and also both the Unity and Unreal Game Engines [Microsoft Research 2018].

The simulator has a built-in recording function which will record vehicle's parameters and any configured external sensors over time. Figure 2.10 shows an example configuration with three different sensors. The simulator can also interface with input devices, and humans can control and record data in the environment. One of the available environments has a traffic simulation. A few of the environments have extra environment variables, such as the time of day that can be changed. This simulator also comes with a headless mode intended for automated testing [Microsoft Research 2018].

Microsoft Research [2018] mentions that as of 2023 AirSim has been deprecated, and all resources will be archived within the coming year. This makes using this simulator difficult as there is now a lack of official support for the simulator.

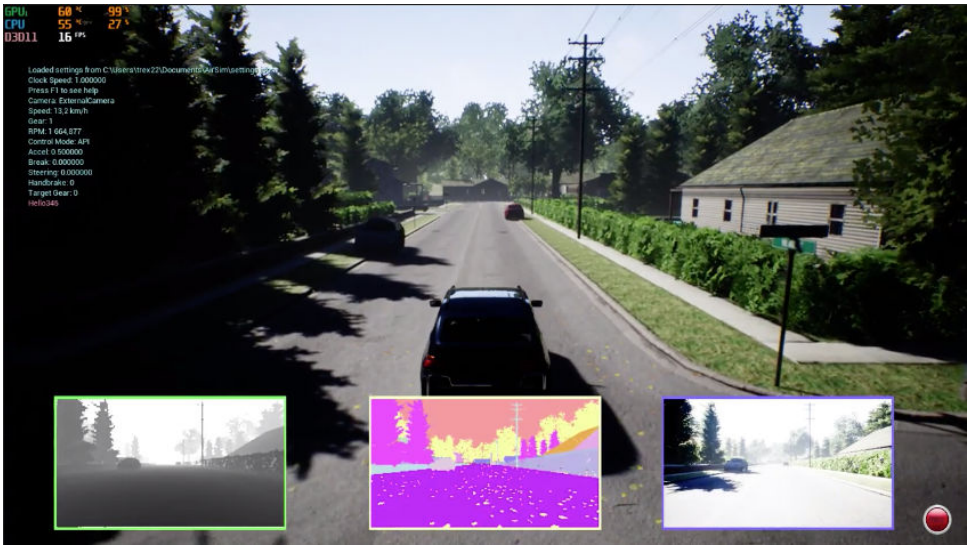


Figure 2.10: A snapshot of an example configuration available in AirSim

2.9.3 MIT VISTA 2.0

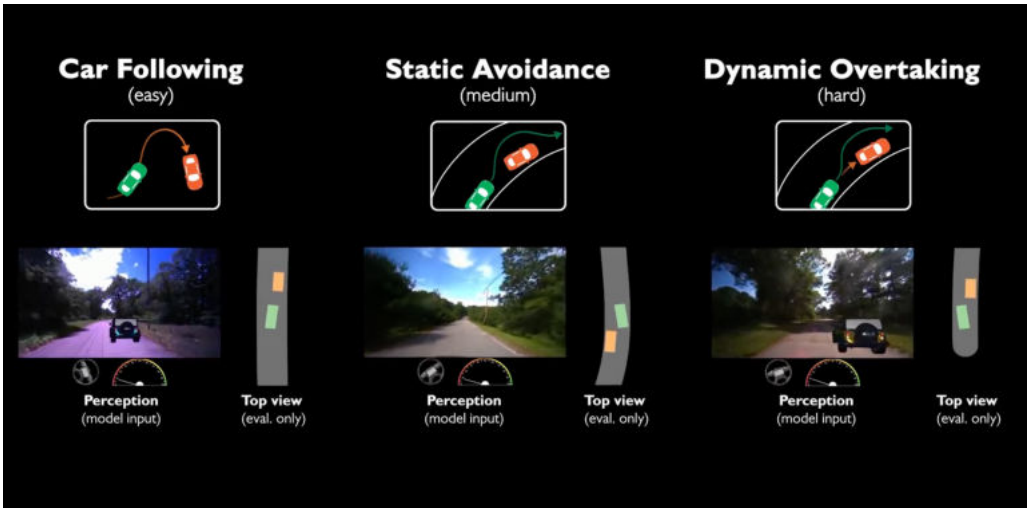


Figure 2.11: Example of augmented data produced by VISTA 2.0 [Gordon 2022]

The MIT VISTA 2.0 simulator focuses on a data-driven approach. It is designed to augment real-world data with hard to produce edge cases and with extra information not initially collected for that dataset [Amini *et al.* 2022]. This simulator allows a much smaller or more uniform dataset to become far more valuable when training models to perform more complex and abstract tasks.

The simulator takes in real world data and applies simulations using agents to generate more data to fill in gaps in the dataset such as adding in virtual cars, changing the weather, or time-of-day. Figure 2.11 shows and example where the lighting conditions are augmented and a simulated vehicle is inserted into a given input sample. This helps

bulk up existing real-world datasets to help improve training and validation results [Amini *et al.* 2022].

The simulator is aimed at working with reinforcement learning, where enough data of edge case scenarios is needed to encode the correct behaviour into those self-driving models [Amini *et al.* 2022]. Its usefulness when dealing with a reduced task space such as steering angle control is limited because it does not produce new travel paths to traverse but augments them.

2.9.4 Comparison of Simulators

Table 2.1: Comparison of the different proposed simulators [Microsoft Research 2018; Dosovitskiy *et al.* 2017; Amini *et al.* 2022]

Simulator	Photorealistic?	Can generate data?	Easily interfaceable?	Customisable?	Has a benchmark?	Being Maintained?	Has Dataset?
CARLA	✓	✓	✓	✓	✓	✓	-
AirSim	✓	✓	-	✓	-	-	✓
MIT VISTA 2.0	-	✓	✓	✓	-	✓	-

All three simulators are able to generate data, and have customisation features. AirSim is no longer maintained and CARLA has a usable benchmark. Only AirSim has an existing dataset which is readily available. Having this existing dataset makes it easier to train a model before integrating its input and output to the simulator.

2.10 Datasets

There are many publicly available datasets targeted towards use in research with a focus on autonomous vehicles. At an earlier point in the field's existence, access to large amounts of generalisable and open data was a problem hindering research [Gómez *et al.* 2010]. There is still a problem of insufficient expressive data available to help cater for the enormous complexities inherent in the problem domain [Spryn and Sharma 2018].

There is a large and ever-growing list of open datasets with different types of data needed for autonomous vehicles from different cameras, angles, sensors, and even different kinds of encoded and labelled data.

Table 2.2 compares all the datasets discussed in this section.

2.10.1 Microsoft's AirSim Tutorial Dataset



Figure 2.12: Sample images pre and post-cropping included in the AirSim Tutorial Dataset [Spryn and Sharma 2018]

This relatively small dataset was generated using AirSim for use with the Autonomous Driving Cookbook [Spryn and Sharma 2018]. It is about an hour's worth of driving data in the *LandscapeMountains* environment. The total size of the dataset is about 3GB but can be used to get an AI model with only a forward-facing camera and steering angle to drive a car in the simulated environment. Figure 2.12 shows an example of input images and an overlay of steering angles. The dataset is made up of sequential images from a video stream captured in AirSim. This dataset does contain more vehicle parameters, such as its current velocity.

2.10.2 Berkeley DeepDrive BDD100k

This is a vast dataset primarily made up of short video clips of 40 seconds each with meta-data associated with those clips, such as segmentation, labelling and path map data. There are around 1.6 terabytes of available data. This dataset contains around a million videos which is approximately ±1100 hours [Yu *et al.* 2020].

The dataset contains different types of classes, such as drivable areas in a scene, and is diverse in weather conditions and geographic locations as shown in Figure 2.13.

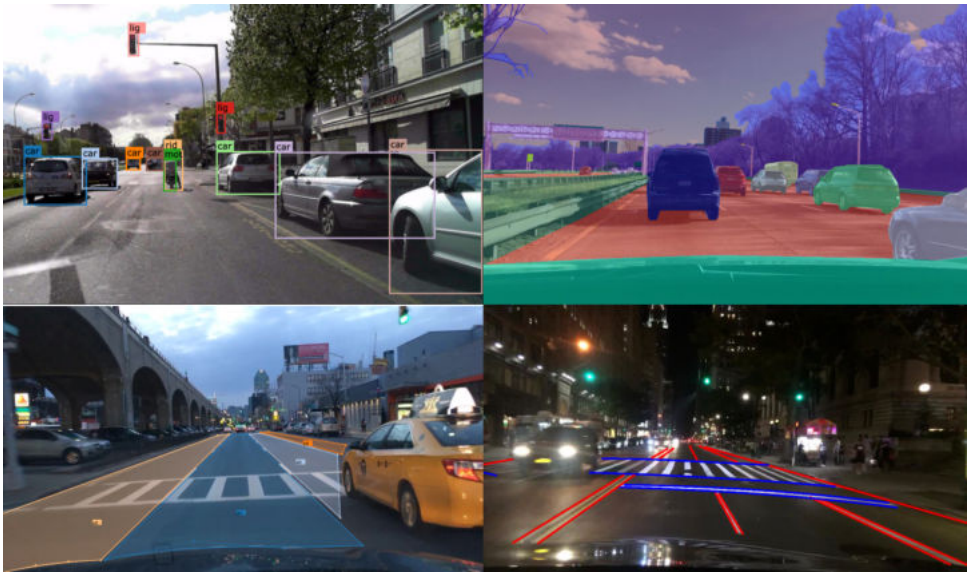


Figure 2.13: Four examples of the types of data available in the Berkeley DeepDrive BDD100k overlaid on top of still images captured from video in the dataset [Yu *et al.* 2018]

2.10.3 Udacity Self-Driving Car Dataset



Figure 2.14: Samples taken from challenge #2 of the Udacity Collection [Cameron 2016]

This dataset started as a few sequential drives recorded and split into images [Gonzalez *et al.* 2017]. Car attributes such as GPS coordinates, gear, brake, throttle, steering angle,

and speed are tracked and saved. This dataset represents multiple days, conditions and geographical locations. Part of the dataset repeats the same path over multiple days and conditions. There are around 290 GBs of data but it has continued to be updated [Cameron 2016].

Bojarski *et al.* [2016a] used this dataset as training, testing and validation data for their two CNN models in the follow-up study to the one which defined an end-to-end setup for self-driving cars. Examples of input RGB images are shown in Figure 2.14.

2.10.4 The Cityscapes Dataset

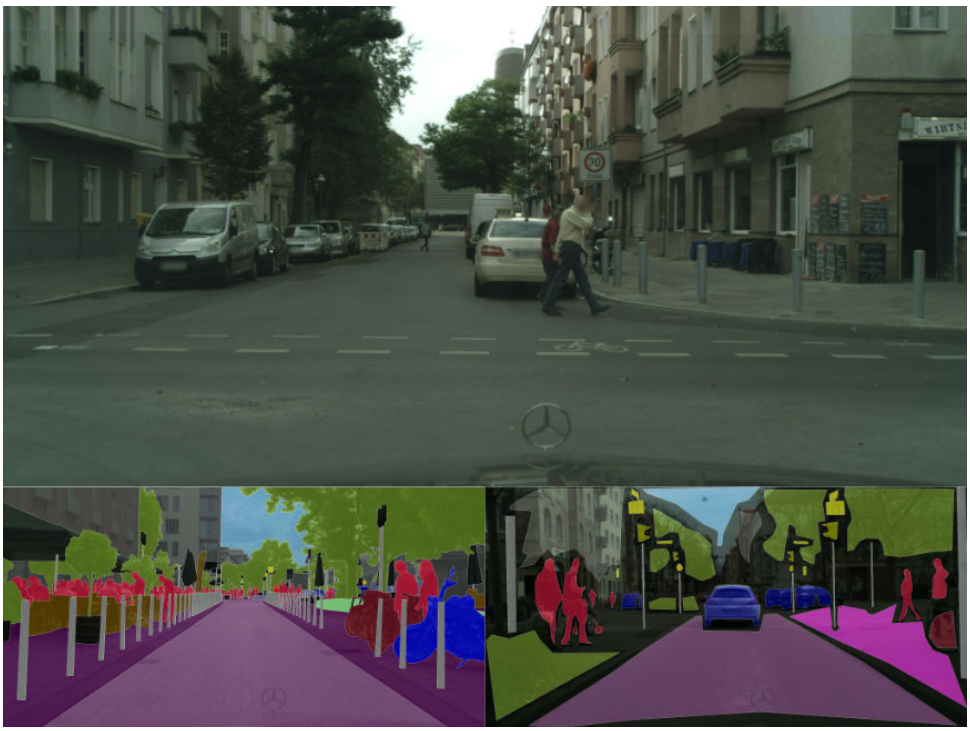


Figure 2.15: Example data from The Cityscapes Dataset. The top image is a blurred example RGB frame from the Berlin sub-set. The bottom left image is an example of a *fine* segmentation map from Stuttgart, and the bottom right is an example of a *coarse* segmentation map from Nuremberg [Cordts *et al.* 2016]

This dataset contains video sequences from 50 cities around central Europe (mainly Germany). The dataset includes *coarse* and *fine* segmentation information, RGB image frames of the video sequences and vehicle parameters such as the steering angle. There are 5000 *fine* segmentation maps of video frames and 20000 *coarse* segmentation frames [Cordts *et al.* 2016]. Examples of input data and the two types of segmentation frames can be seen in Figure 2.15.

2.10.5 From Games Dataset

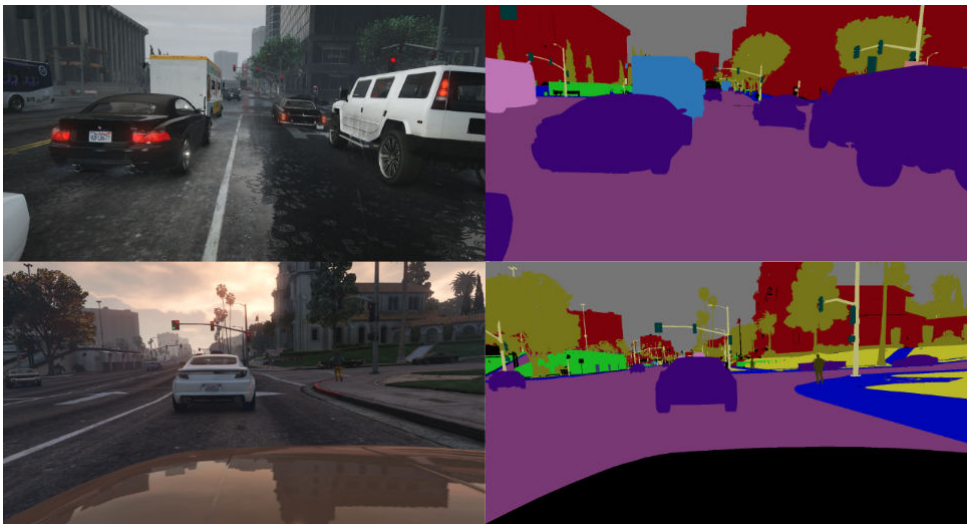


Figure 2.16: Example data from the From Games dataset. The left side represents two front facing image samples from the dataset, with the corresponding segmentation label on the right [Richter *et al.* 2016b]

This dataset contains only segmentations and forward facing RGB image frames generated from video games. The aim was to produce vast amounts of high quality segmentation data as quickly as possible, by reducing how long it takes to label a single image. Annotation speed was improved by using existing annotation data from previous frames, and human input in the difference which was left due to the change between the two frames. This dataset contains over 14 thousand frames all with corresponding high fidelity segmentation maps. These segmentation maps have their own label values but a conversion matrix to Cityscapes labels is provided in the available source code [Richter *et al.* 2016ab]. Two examples of the input data and corresponding segmentation maps can be seen in Figure 2.16.

2.10.6 Oxford's Robotic Car



Figure 2.17: To the left is a 3D map produced by the Lidar data present in the Oxford's Robotic Car dataset, and to the right are some example images from the same dataset in a few different conditions [Maddern *et al.* 2017]

This dataset focuses on the same road route over a year of travel. This captured the same stretch of road in many different scenes such as changing weather conditions, pedestrians, traffic, changing landscape, and even road maintenance. There are over 100 repetitions of this route. The dataset contains stereo images, Lidar, GPS, and six camera angles [Maddern *et al.* 2017]. Figure 2.17 shows some examples from the dataset.

2.10.7 Comma.ai comma2k19 dataset

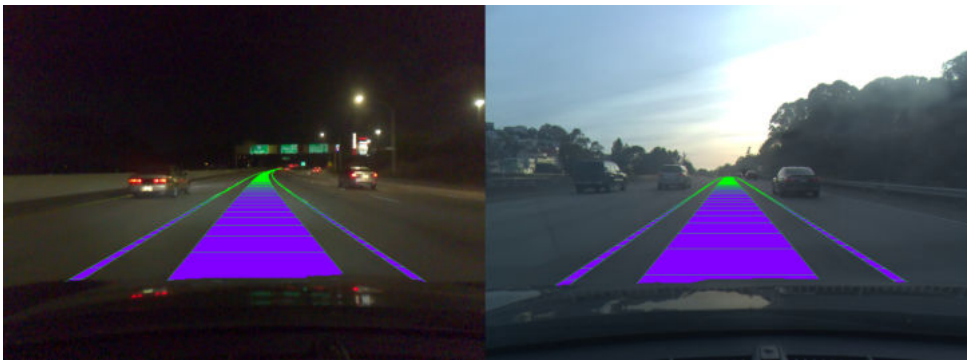


Figure 2.18: Sample highway image stills are taken from the *comma2k19* dataset [Schafer *et al.* 2018]

This dataset contains 33 hours of highway driving on the same highway. It contains data on the vehicle's speed, acceleration, steering angle, forward-facing video, and GPS coordinates. Each video clip is 1 minute in length [Schafer *et al.* 2018]. Figure 2.18 shows example input images from the dataset.

2.10.8 Baidu ApolloScape



Figure 2.19: Some examples of images from the ApolloScape dataset [Wang *et al.* 2018]

This is a collection of distinct datasets which focus on different topics. It is a relatively large combined dataset. There is overlap between the sub-datasets as the same or similar data may be used in conjunction with meta-data such as segmentation maps [Wang *et al.* 2018]. Figure 2.19 shows some examples of input images from the dataset.

Included datasets:

- *Scene Parsing* is a dataset of RGB videos, 3D point maps and semantic segmentation
- *3D Car Instance* is a dataset of stereo video sequences recorded across different cities
- *Lane Segmentation* is a dataset which contains stereo images with an accompanying segmentation map of the road lanes and markings
- *Trajectory* is a dataset of trajectories of all vehicles in the scene (urban environment) meant for planning problems
- *Tracking* is a dataset of captured 3D Lidar data

2.10.9 Comparison of Datasets

Table 2.2: Comparison of the different proposed datasets [Yu *et al.* 2018; Cameron 2016; Spryn and Sharma 2018; Wang *et al.* 2018; Schafer *et al.* 2018; Maddern *et al.* 2017; Cordts *et al.* 2016; Richter *et al.* 2016a]

Dataset	Size	Used in background literature?	Vehicle Control Signals (Speed, steering angle)	Varying weather?	Varying light?	Easily usable?	Varying paths?	Forward RGB Images?	Stereo images?	Segmentation?	Lidar?	Labels?	GPS?
Berkeley DeepDrive BDD100k	1600 GB	-	✓	✓	✓	-	✓	✓	-	✓	-	✓	✓
Udacity Self-Driving Car Dataset	209 GB	✓	✓	✓	✓	✓	-	✓	-	-	-	✓	-
Microsoft's AirSim Tutorial Dataset	3 GB	✓	✓	-	-	✓	✓	✓	-	-	-	-	-
Baidu ApolloScape	1220 GB	-	✓	✓	✓	-	✓	✓	✓	✓	✓	✓	✓
Comma.ai comma2k19 dataset	100 GB	-	✓	-	✓	-	-	✓	✓	✓	-	-	✓
Oxford's Robotic Car	23150 GB	-	✓	✓	✓	-	-	-	✓	-	✓	✓	✓
Cityscapes Dataset	74.9 GB	-	✓	-	-	✓	✓	✓	✓	✓	-	✓	✓
From Games Dataset	34.1 GB	-	-	✓	✓	✓	-	✓	-	✓	-	✓	-

Most of the datasets evaluated have vehicle control signals (except *From Games*), and all datasets except *Oxford's Robotic Car* have forward RGB images. The two datasets that are used in background literature are sized differently, with *Microsoft's AirSim Tutorial Dataset* being the smallest dataset, and *Udacity Self-Driving Car Dataset* being a considerably larger dataset. *Cityscapes Dataset* whilst smaller than *Udacity Self-Driving Car Dataset* has both fine and coarse segmentation maps. *From Games* contains only one group of high accuracy segmentation maps and the corresponding forward facing RGB image frames. *From Games* also includes a mapping matrix to convert its segmentation maps to have compatible labels to the *Cityscapes Dataset*, making it an ideal comparison dataset to the *Cityscapes Dataset*.

2.10.10 Manual Generation of Simulated Data

The datasets described in this chapter contain a vast amount of data which can be used to train a deep-learning network to drive a car. There are also many other open datasets available for use [Cambridge Spark 2018].

The datasets discussed are large but may have issues with too much of the same kind of action or road rule encoded into the data. This bias can negatively affect training performant AI models to drive vehicles. A solution to this could be to sample from multiple datasets. This introduces its own problems with camera angles, artefacts in the data and camera calibrations.

Another solution is constructing a smaller specialised dataset that encodes the desired behaviours. This dataset can then be used to train a network and observe its behaviour. It can also be used with additional datasets to bias specific behaviour over other behaviour.

An alternative use of this constructed dataset is to evaluate the performance of a model trained on other datasets as a benchmark of behaviour.

Some of the simulators discussed previously provide tools to easily record human actions in a simulated environment - which can quickly and effectively produce a dataset. Whilst this requires human interaction, Spryn and Sharma [2018] shows that even an hour of recorded simulated footage, a small dataset, is effective in getting results when using an end-to-end autonomous vehicle learning model.

The MIT VISTA 2.0 simulator is an example of augmenting existing real-world data with simulated edge cases, events and added sensor information for improved training and evaluation of models.

2.11 Methods to Improve Training Neural Networks

2.11.1 GPU Optimisation

According to Goodfellow *et al.* [2016] when the performance requirements of a large enough deep learning model are large, it is possible to produce models with a lower output quality due to the high computational demand. Graphics Processing Units (GPU)s are specialised compute accelerators which aid in processing extensive multivariate calculations. GPUs are well adapted to vision problems because their development was aided by the video game industry. Programming libraries such as PyTorch [PyTorch Machine Learning Framework 2023] are able to efficiently make use of general compute GPUs and abstract away the difficulties of directly managing these devices.

2.11.2 Image pre-processing techniques

Spryn and Sharma [2018] used several image pre-processing techniques to help improve and speed up model training. The first technique was cropping a region of interest (ROI) from the input data, focusing mainly on the section showing the road surface. This is a way to reduce the dimensionality of the input data to speed up training and reduce models fixating on less consequential parts of the input data frames.

Spryn and Sharma [2018] also removed the alpha channel from the input image as it was an additional dimension of the input data with very little value for driving a vehicle.

Rescaling image pixel value ranges so that they lie between $[0, 1]$ is a simple method to help improve the results of training models against images, according to Goodfellow *et al.* [2016], since the distance between the values is smaller, and therefore is easier for a model to train with.

When input image sizes have different dimensions compared to the expected image input, resizing these images using interpolation and extrapolation techniques can be used. The OpenCV documentation mentions a simple method called nearest neighbour interpolation, where the nearest pixel's value is selected by rounding the coordinates of the desired pixel, which will return the nearest pixel [OpenCV: Geometric Image Transformations Documentation].

2.11.3 Early Stopping

Goodfellow *et al.* [2016] defines *early stopping* as an algorithm to select the best-trained weights for a model based on the validation loss achieved during training. The algorithm will train a model for some number of epochs whilst observing the training loss, validation loss and storing the model parameters every time the validation loss improves. The best model parameters are selected which is determined by the validation loss. Training can be stopped based on a distance value which determines how far to train a model without changing the validation performance metric being evaluated before stopping. Goodfellow *et al.* [2016] states that this method is a basic form of model regulariser. There are variations of the early stopping algorithm with many trade-offs such as an increase in the amount of compute required.

2.11.4 Small Initial Random Weights

When initialising deep neural network models, it is possible to start with a model that cannot be optimised. Another possibility is that a non-optimal minimum could be found for the computed weights of the model. The selection of the initial weights is crucial in improving the training results of these models. If the initial weights of the hidden layers are too small (nearing zero), then they will never train. If the weights are too big, training may never converge and find the optimal gradients for the problem [Glorot and Bengio 2010].

A solution to this problem is initialising the model weights with small random values. Uniform distributions appear beneficial because training discriminates and reduces non-useful weights to zero (turns them off), so when training begins, each weight will have an equal opportunity to train. An improved solution is to use the *Xavier Initialization* technique. This technique allows the gradients and activations of the models to train effectively in both the forward and backward passes of the training loop. This is done by selecting random weights whilst considering the number of inputs and outputs of each layer that is being randomised [Glorot and Bengio 2010].

Glorot and Bengio [2010]; PyTorch 2.0 documentation [2023] defines the *Xavier Uniform Initialization* as:

$$[-a, a] \text{ for } a = \text{gain} \times \sqrt{\frac{6}{\text{input size} + \text{output size}}} \tag{2.4}$$

where the *gain* is an optional scaling parameter, and the value *6* is selected from the original article by Glorot and Bengio [2010].

2.11.5 ReLU Layers

A Rectified Linear Unit (ReLU) is an activation function which returns zero when the input is negative, else returns the inputted value. This has the effect of reducing the number of weights with non-zero values. This can help increase the speed and reduce the loss whilst training [Du *et al.* 2017]. According to Goodfellow *et al.* [2016] these units are easy to optimise due to their similarity to linear units.

The layer output when using ReLU is defined as:

$$\text{layer outputs}(x) = \begin{cases} 0 & : \ x \leq 0 \\ x & : \ x > 0 \end{cases} \tag{2.5}$$

ReLU units cause the gradients computed through the units to be large and consistent. This magnifies the gradient direction and improves the learning of the model. A downside to the application of this kind of unit is that inputs which cause activations near or at zero of the model will not be learned [Goodfellow *et al.* 2016].

Cao and Wu [2022] discusses how CNN models are good at identifying and segmenting objects in the input space. Their research aims to understand the inductive bias - the reasoning as to why CNN models are suited to analysing real-world images. They find that even randomly initialised CNN models can identify objects from input images. They speculate that this is because activation functions such as rectified linear units (ReLU) reduce the probability of regions of the image that are not important foreground objects from activating the gradients in these models.

2.11.6 Drop-out Layers

Du *et al.* [2017] refers to drop-out layers as a form of regularising layers. It is a mask that removes some neurons' effect on the next layer in the network. They are used to prevent over-fitting when training neural networks [Baeldung 2023]. Goodfellow *et al.* [2016] mention that this technique helps reduce the need for bagging, where multiple models are trained and evaluated on the same test examples to generate a vote which is used to decide on the final value being predicted.

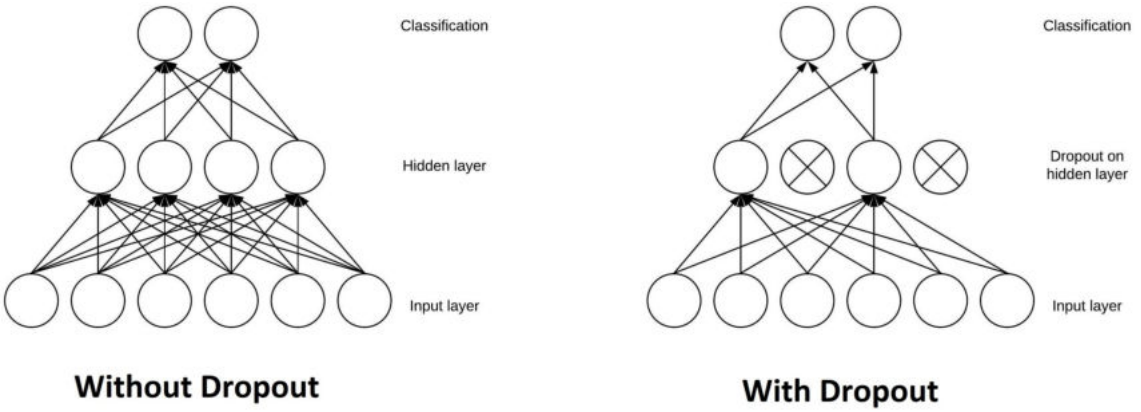


Figure 2.20: Example Neural Network With and Without Drop-out applied to Successive Layers [Baeldung 2023]

Drop-out can be performed by multiplying a neuron's output from a network by zero, effectively nullifying its effect on the overall result of the network's forward pass. The neurons selected to be nullified are determined by random selection. This is similar to training an ensemble of sub-networks in an ensemble approach. Figure 2.20 shows how some nodes get deactivated when using drop-out. The drop-out layers are usually active in the training phase and deactivated in the evaluation phase once the best model parameters have been selected [Goodfellow *et al.* 2016].

2.12 Real-world usage of end-to-end systems for autonomous vehicles

Andrej Karpathy was the director of artificial intelligence at Tesla Inc. He gave a talk for the PyTorch community on the 6th of November 2019 [Karpathy 2019]. He discussed how Tesla, Inc. built their full self-driving architecture in the talk. Figure 2.21 shows two images shared in this talk which show how their architecture is structured. The first diagram shows a hybrid architecture employing multiple deep learning models solving specific problems which are then combined together to solve the overarching problem of autonomous driving. The second image in Figure 2.21 shows how semantic information is extracted from the input scene captured by cameras in the vehicle.

Tesla Inc. employs a complex neural network which analyses images for multiple tasks. These networks in Andrej Karpathy's own words, are "ResNet-50 like". RGB images of a reasonably high resolution (1280x960) are used. A backbone network is shared for 15 different self-driving tasks which are solved by AI "heads" [Karpathy 2019].

According to Andrej Karpathy in his talk, Tesla, Inc. has an architecture which is very similar in style to the reduced end-to-end architecture investigated by this research but on a grander scale. It is an evolution of a theoretically reduced problem set scaled up to tackle the real-world problem of a fully automated self-driving car [Karpathy 2019].



(a) Shared backbone architecture (b) An example RGB scene with extracted labels

Figure 2.21: An example of a shared architecture and RGB scene used by Tesla, Inc. for their full self-driving AI models [Karpathy 2019]

2.13 Safety Concerns

Many experts and critics have raised concerns about the safety and ethics of testing experimental AI systems in the real world - where people may be hurt or injured [Forbes 2023].

Shepardson [2023] reported that in early 2023, Tesla, Inc. recalled their full self-driving car feature via an over-the-air update of their vehicle's internal software. This was due to the NHTSA finding that the software posed an unreasonable risk to safety [Administration 2023].

According to Corso *et al.* [2022] these end-to-end self-driving systems can pose a severe risk to life, damage to property and safety if they are not adequately tested and observed. These systems need concrete requirements for what constitutes safe and reasonable decisions. Then these systems need to be evaluated based on these requirements. Autonomous driving systems should also be tested and evaluated on the safety requirements the NHTSA defined, during its usage. When failures happen, interpretation should also be performed to understand why the event happened and how to prevent such an accident from happening again.

2.14 Conclusion

End-To-End systems for self-driving vehicles show promise in controlling tasks traditionally performed by a human driver. However, they pose safety risks and therefore require testing, evaluation and interpretation of their learned behaviours.

The attribution methods presented have been used as part of a more significant research endeavour to understand an end-to-end model in a reduced problem space. Some of these methods require the model being investigated to be adapted to support them. Attribution and analysis, which have been confirmed not to be recreating the input data and are also

not independent of the model's parameters, have been used to ensure accurate evaluations are made against the differently trained models.

Visual analysis alone was not used on the models as they can be misleading without other metrics and analysis backing up the results.

Resources available for training and testing autonomous vehicle AIs have become plentiful. Many datasets are available focusing on a broad spectrum of sensors, roads and locations. There is also a selection of simulators focusing on photorealism in the field and other simulators focusing on specific sub-problems that have to be solved for autonomous vehicles. Some of the datasets investigated have been used to implement the experiment defined by Chapter 3: An Experimental Setup for an End-to-End System for Self-Driving Cars.

Chapter 3

An Experimental Setup for an End-to-End System for Self-Driving Cars

3.1 Introduction

Bojarski *et al.* [2016b] defines an experimental setup that minimises the number of inputs and outputs required to produce and test AI models for autonomous vehicles. This setup contains only a front-facing camera as input and the steering angle of a vehicle as the predicted output. This framework is limited to help reduce the complexity from more sensor inputs or more detailed outputs required for a fully featured autonomous vehicle. The aim is to test ideas in a limited solution space which can then be expanded out to a multi-task domain. This experimental setup does not look at solving any other tasks required for a full or semi-full self driving car problem.

3.1.1 The CNN model used in the End-to-End Learning Experimental Setup

Bojarski *et al.* [2016b] created a network of relatively large size (250 000 parameters) which is used to produce an end-to-end learning model for autonomous vehicles. Figure 3.1 is a diagrammatic representation of this model. They use visual back-propagation as their attribution method to explain what features the model uses to predict steering angle.

This model is referred to as the *End-To-End* model going forward.

3.1.2 Two CNN models used for applying visual back-propagation to autonomous vehicles

Bojarski *et al.* [2016a] continue their work and apply visual back-propagation on two much larger networks. They named these networks *NetSVF* and *NetHVF*, respectively. The difference between these two networks is their input size, where *NetSVF* has two times the vertical field of view of *NetHVF*.

3.1.3 CNN model used in the Autonomous Driving Cookbook

Spryn and Sharma [2018] has a different model design to Bojarski *et al.* [2016b], as it is considerably smaller with only three convolutional layers. This work makes no attempts at attributing the feature decomposition of this model but focuses on the model's performance in testing using synthetic results such as its validation loss. This model will be referred to as the *Autonomous Cookbook* model.

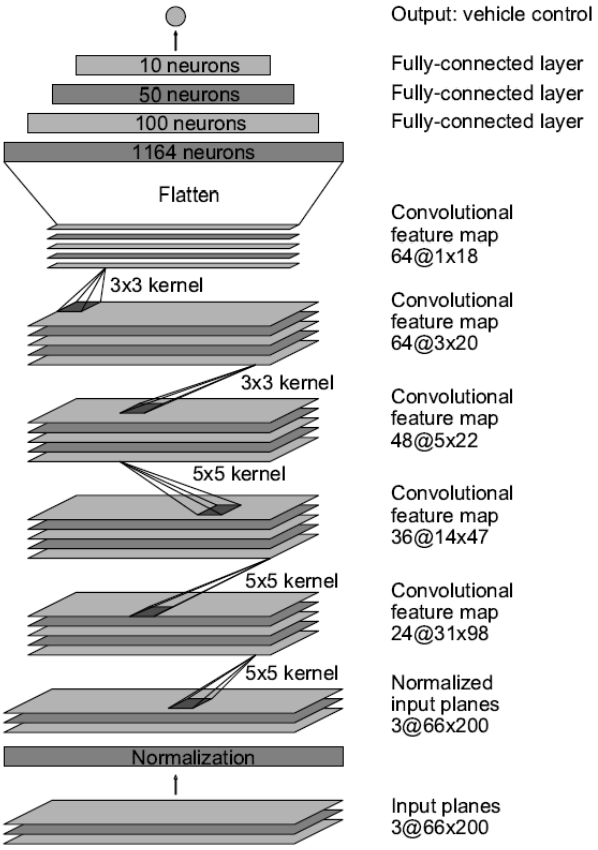


Figure 3.1: A diagrammatic representation of the model used in the article, End-to-End Learning for Self-Driving Cars [Bojarski *et al.* 2016b]

3.1.4 A metric of autonomy for self-driving car model

Bojarski *et al.* [2016b] define a metric called autonomy (MoA) which looks at the level of autonomy that a vehicle AI model has. This is defined as the time percentage in which the vehicle was autonomous, i.e. not driven by a human. An assumption made here is that it requires *6* seconds for a human to intervene in the operation of an autonomous vehicle.

MoA is computed for level 3 and level 4 AI models where a human is expected to intervene when the AI model deviates from expected behaviour.

This metric is defined by:

$$autonomy = (1 - \frac{(I * I_c)}{E}) \cdot 100 \tag{3.1}$$

where I is the number of interventions, I_c is the constant of intervention (6 seconds for a human), and E is the elapsed time in seconds.

Realistically in this experimental setup, where such an AI model is evaluated in simulation or synthetically, the number of human interventions needs to be approximated. Spryn and Sharma [2018] show a technical implementation of the same End-to-End setup, where the approximation of is defined as the deviation (within a margin of error) of the steering angle predicted by the model compared to what the steering angle should have been.

This makes sense as an approximation for the number of human interventions however there is a limit to the amount of time it will take a human to intervene against an AI model when it is predicting dangerous or bad outputs. Bojarski *et al.* [2016b] defines 6 seconds as the time required for a human to retake control of the vehicle.

This is denoted by:

$$\mathbb{1}(x, \varepsilon) = \begin{cases} 0 & x \leq \varepsilon \\ 1 & otherwise, \end{cases}$$

then

$$\text{number of interventions } = \sum_{n=1}^{N} \mathbb{1}(|\theta_n - \phi_n|, \varepsilon) \tag{3.2}$$

where N is the total predictions made, ε is the error margin, θ is the predicted angle, and ϕ is the expected angle.

I_c is also assumed to be 6 seconds when a human is involved as a conservative measure for how quickly a human can intervene. When looking at a synthetic constant of intervention, we can assume it to be 1 second (or frame), as no human is directly in the loop thus, the calculated number of interventions is per frame of data rather than recorded over an elapsed sequence of time. Each data frame in a dataset can have the exact error or deviation from an acceptable predicated steering angle computed to a given accuracy of response (number of decimal places of error that is acceptable for the experimental setup, i.e. ε). The MoA is computed using the absolute sum of the number of interventions (error).

3.2 Experimental Setup

- Apply pre-processing to image data coming from a forward-facing camera in a vehicle (real or simulated) [Spryn and Sharma 2018]
- Train a CNN model using an existing dataset [Spryn and Sharma 2018]
- The model takes in RGB images from the vehicle (forward-facing) as input

- The model returns the steering angle as the output
- Apply an automatic metric to the AI driving to determine how much autonomy the AI has [Bojarski *et al.* 2016b]
- Visualise the internal model State
- Use information gathered to infer the performance of the model and explain its decisions

3.3 Problems with this setup

This configuration of an autonomous vehicle is not a realistic representation of the necessary input for real-world operation. It is considered a simplistic representation of the overall problem domain [Spryn and Sharma 2018]. It does not have the many viewing angles of the vehicle, and so cannot model the complete world around the car. It also does not consider that vehicle control is more than just steering angle. A more realistic model may combine multiple tasks to produce a higher level of autonomy for the vehicle (from the perspective of a real-world implementation).

This configuration requires that any rules to drive correctly and safely are encoded into the input dataset. This makes biases in the input dataset more prevalent in the model's outcomes. This also makes a model's decisions specific to the input data given - affecting the ability of such a model to scale into different real-world environments and scenarios where different rules might be required. This adds up to needing a vast amount of data to overcome model complexity which exceeds the currently available amount of data [Spryn and Sharma 2018].

Another issue is that the CNN models created by Bojarski *et al.* [2016b] are tested with Visual Back-Propagation, which was then processed by human evaluation to determine what feature extractions are occurring. Visual back-propagation has also not had any sanity checks run on the method to confirm how it was able to produce its heat-maps.

Since this setup only predicts steering angle, a bias between the edges of the road and the steering angle may occur. It is hard to determine if the results are due to the edges of the road, other meaningful semantic features in the input dataset, or some other unknown correlation. A analysis of the CNN models require a more quantitative approach.

3.3.1 Usability of the model

Whilst this configuration is considered by Spryn and Sharma [2018] to be simplistic and an introductory model within the field of autonomous vehicles, Bojarski *et al.* [2018] states that it is a helpful setup for testing complex solutions in a reduced problem space where formal proofs and empirical evidence are easier to produce.

3.4 Conclusion

This experimental setup for autonomous vehicles is an informative reduced problem space to test AI models. However, it may be too limited in its output variables to be effective enough in making broad conclusions on the behaviour of some of these AI systems beyond the scope of their solution space. The reduction in complexity is possibly coupled with more sensitivity to undesirable patterns in the input dataset. It is still a valuable indicator of observed behaviour which may scale into larger problem spaces with many more complex tasks. It is a useful setup to produce proof of concept work before scaling up to much larger and more complex domains.

Chapter 4

Methodology

4.1 Introduction

This study aimed to recreate the setup discussed in Chapter 3. The four model architectures discussed in the background and two additional simple model architectures were used. The two new architectures were designated as control models meant to be as simple as possible. This study was composed of training the models on the same selected dataset as Bojarski *et al.* [2016a], the Udacity Self-Driving Car dataset. Other datasets were then used to test the models performance, using metrics such as MSE, which were used by Bojarski *et al.* [2016b].

We also analysed the trained models using the GradCAM attribution method, which was performed in a scheme named GradSUM to perform attribution over a ground truth dataset, The Cityscapes Dataset.

An analysis of the performance of the models with random weights was also conducted to determine how much of the behaviour of the models is sensitive to randomness in these experiments and the effect of model architecture on the performance of the models.

4.2 Aim

The goal of this work was to demonstrate that the conclusion found by Bojarski *et al.* [2016b], *"The CNN is able to learn meaningful road features from a very sparse training signal (steering alone)"* holds true when using a validated attribution method such as GradCAM against multiple CNN model architectures.

4.3 Hypothesis

A CNN trained to steer a vehicle given data from a forward-facing RGB camera inside of that vehicle as input and steering angle as output can extract meaningful semantic information from the input data.

4.4 Selected CNN Models

Four viable CNN models are used in the literature to produce end-to-end models for autonomous vehicles. These models are *NetSVF* [Bojarski *et al.* 2016a], *NetHVF* [Bojarski *et al.* 2016a], *End-To-End Learning Model* [Bojarski *et al.* 2016b], and *Autonomous Driving Cookbook Model* [Spryn and Sharma 2018]. A comparison of these model architectures can be found in Table 4.1.

Bojarski *et al.* [2016b] used *End-To-End Learning Model* in their original work in defining an end-to-end setup for self-driving cars. Bojarski *et al.* [2016a] expanded their work by defining two larger model architectures based on *End-To-End Learning Model*. These architectures are *NetSVF* and *NetHVF*. The last model was a separate implementation of an end-to-end model to predict steering angle by Spryn and Sharma [2018], which serves as a useful independent model architecture.

These architectures range in size, complexity and depth. Two far simpler models were also introduced. These are intended to compare model architectures created to learn how to predict steering angle and simplistic architectures not expected to perform exceptionally well on the task.

They all take in the same input (an RGB image) and output the same single variable (steering angle). The input layers for all models adapt to the dimensionality of the input image data and then go through a pooling layer. Each model has a linear output node which returns the steering angle.

4.4.1 Simplified CNN model architectures

The two simple model architectures that have been introduced in this study are referred to as *TestModel1* and *TestModel2*. They are intentionally made to be simple so that their performance can be compared to the more advanced model architectures.

The aim is to identify if the most simple versions of these types of models can solve the task of predicting steering angles, and if so does the further analysis of the model architectures appear the same or different to the more purpose built model architectures found in literature?

TestModel1

TestModel1 has a single convolutional layer and a linear output layer with ReLU and pooling layers in between the layers. This is designed as the most simplified architecture (with optimisations) possible. The filter size and stride also aim to be as small as possible.

TestModel2

TestModel2 has two convolutional layers and a linear output layer, also with ReLU and pooling layers in-between the convolutional layers. This is meant to be one more convolutional layer than *TestModel1* to see the difference in training and evaluation between having one convolutional layer and two.

Table 4.1: Comparison of the six different CNN models [Bojarski *et al.* 2016ab; Spryn and Sharma 2018] (From largest structure to smallest structure)

Model Name	NetSVF	NetHVF		
Layer Type	Layer Output Size	Layer Output Size	Filter	Stride
Convolution	32 x 123 x 638	32 x 123 x 349	3 x 3	1 x 1
Convolution	32 x 61 x 318	32 x 61 x 173	3 x 3	2 x 2
ReLU	-	-	-	-
MaxPooling2D	-	-	-	-
Convolution	48 x 59 x 316	48 x 59 x 171	3 x 3	1 x 1
ReLU	-	-	-	-
MaxPooling2D	-	-	-	-
Convolution	48 x 29 x 157	48 x 29 x 85	3 x 3	2 x 2
ReLU	-	-	-	-
MaxPooling2D	-	-	-	-
Convolution	64 x 27 x 155	64 x 27 x 83	3 x 3	1 x 1
ReLU	-	-	-	-
MaxPooling2D	-	-	-	-
Convolution	64 x 13 x 77	64 x 13 x 41	3 x 3	2 x 2
ReLU	-	-	-	-
MaxPooling2D	-	-	-	-
Convolution	96 x 11 x 75	96 x 11 x 39	3 x 3	1 x 1
ReLU	-	-	-	-
MaxPooling2D	-	-	-	-
Convolution	96 x 5 x 37	96 x 5 x 19	3 x 3	2 x 2
ReLU	-	-	-	-
MaxPooling2D	-	-	-	-
Convolution	128 x 3 x 35	128 x 3 x 17	3 x 3	1 x 1
ReLU	-	-	-	-
MaxPooling2D	-	-	-	-
Convolution	128 x 1 x 17	128 x 1 x 8	3 x 3	2 x 2
ReLU	-	-	-	-
MaxPooling2D	-	-	-	-
Fully-connected (Flatten)	1024	1024	-	-
Fully-connected	512	512	-	-
Fully-connected	1	1	-	-

	End-to-End Learning Model		
Layer Type	Layer Output Size	Filter	Stride
Convolution	24 x 31 x 98	5 x 5	2 x 2
Convolution	36 x 14 x 47	5 x 5	2 x 2
ReLU	-	-	-
MaxPooling2D	-	-	-
Convolution	48 x 5 x 22	5 x 5	2 x 2
ReLU	-	-	-
MaxPooling2D	-	-	-
Convolution	64 x 3 x 20	3 x 3	5 x 5
ReLU	-	-	-
MaxPooling2D	-	-	-
Fully-connected (Flatten)	1164	-	-
Dropout (p=0.5)	-	-	-
Fully-connected	480	-	-
Fully-connected	18	-	-
Fully-connected	1	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-

	Autonomous Driving Cookbook Model		
Layer Type	Layer Output Size	Filter	Stride
Convolution	59 x 255 x 16	3 x 3	2 x 2
Convolution	29 x 127 x 32	3 x 3	2 x 2
ReLU	-	-	-
MaxPooling2D	-	-	-
Convolution	14 x 63 x 32	3 x 3	2 x 2
ReLU	-	-	-
MaxPooling2D	-	-	-
Fully-connected (Flatten)	6948	-	-
Fully-connected	64	-	-
-	10	-	-
-	1	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-

	Test Model 2 (Control)		
Layer Type	Layer Output Size	Filter	Stride
Convolution	60 x 256 x 32	3x3	1x1
Convolution	60 x 256 x 32	3x3	1x1
ReLU	-	-	-
Convolution	60 x 256 x 32	3x3	1x1
ReLU	-	-	-
MaxPooling2D	-	-	-
Fully-connected (Flatten)	122880	-	-
Fully-connected	10	-	-
Fully-connected	1	-	-

	Test Model 1 (Control)		
Layer Type	Layer Output Size	Filter	Stride
Convolution	60 x 256 x 32	3x3	1x1
ReLU	-	-	-
Convolution	60 x 256 x 32	3x3	1x1
ReLU	-	-	-
MaxPooling2D	-	-	-
Fully-connected (Flatten)	122880	-	-
Fully-connected	10	-	-
Fully-connected	1	-	-
-	-		-

4.5 Temporal Leakage Mitigation

Temporal leakage can be a factor in the datasets being used because the data-points are extracted frames from video sequences. This kind of data leakage can then occur if the frames are divided between training, testing, and validation sets by random selection. This is because video frames which appear near to each other have very closely correlated data. This leads to leaking data between the training and testing subsets, which are intended to be independent of each other. If frames which are consecutive or temporally close are split into these separate training and test subsets, it may cause an artificially boost performance metrics.

The solution we applied for our work is to keep the specific groupings of data per dataset together. The Udacity dataset has two separate driving runs, one *Northbound*, and one *Southbound*. By splitting out the training and testing data between the two distinct groups of data this ensures that there is no temporal leakage. Other datasets provide train, test, and other groups of data within the folder structure of the datasets. These were also utilised where available to reduce the chance of temporal leakage during training.

Later in this work, when we used the GradSUM analysis scheme on the Cityscapes Dataset temporal leakage was not a concern because all of the *fine* segmentation maps were used and the models were not trained against any of that data beforehand.

4.6 Pre-processing Steps

Du *et al.* [2017]; Spryn and Sharma [2018] have defined a set of image pre-processing steps used to train a model on a self-driving dataset. For this study, the following pre-processing steps were performed:

1. Split the dataset into training, testing and validation datasets
2. Randomly drop data-points to remove steering-angle bias
3. Randomise the sample order of the input dataset
4. Crop the image and only keep a region of interest to improve training time and performance
5. Remove the alpha channel from the image to reduce the dimensionality
6. Rescale pixel dimension using nearest neighbour interpolation (if the dimensional size does not match the expected model size)
7. Rescale the pixel values from a scale of [0, 255] to [0, 1] to help improve training and evaluation time

4.7 Analysis of Useful Datasets

There are many datasets that can be chosen for this task. Table 2.2 outlines the comparison of the datasets discussed in Section 2.10. Three datasets were selected for testing the trained models.

The first criterion chosen was to use the same datasets as the literature (where possible). Spryn and Sharma [2018] makes use of Microsoft's AirSim Tutorial Dataset, and Bojarski *et al.* [2016a]; Du *et al.* [2017] makes use of the Udacity Self-Driving Car Dataset.

The Udacity Dataset for Self-Driving Cars is far more extensive than The Cityscapes Dataset and is based on real-world data. Therefore it was selected to be used as the training dataset. The Cityscapes Dataset was selected as the ground truth segmentation dataset because of its relative ease of use, smaller data size (compared to other evaluated datasets containing segmentation data), and prevalent use in other domains of research [Papers With Code 2023].

Looking at the bias of steering-angle is an important metric to observe when selecting a dataset for this task. Most datasets will have a high bias towards a zero steering angle as the majority of the time when driving a vehicle the steering angle is close to zero (with the front wheels facing forward). This can cause over-fitting issues when training models. The simple fix for this problem is to drop steering angle samples from the training dataset so that there is no bias between straight and turning [Spryn and Sharma 2018]. However the number of data-points left could be a concern when training very large and complex models.

4.7.1 Udacity Self-Driving Car Dataset

Reasons for selecting the Udacity Self-Driving Car Dataset

The Udacity Self-Driving Car Dataset described in Section 2.10.3 was used by multiple sources implementing the experimental setup for an end-to-end system as described in Chapter 3 [Bojarski *et al.* 2016a; Du *et al.* 2017]. Therefore it is a natural fit for this study. It is also a medium-sized dataset, which is helpful because the larger datasets are much more difficult to work with and the small datasets may not have enough data points to train and evaluate the models properly.

Table 4.2: Udacity Dataset Summary

Data Group	Total Count	Straight Line Image Count	Swerve Image Count	Swerve Ratio
Dataset Before Drop	26720	19102	7618	0.3988
Dataset After Drop	15236	7618	7618	1.0
Train	10780	5390	5390	1.0
Validation	2938	1469	1469	1.0
Test	1518	759	759	1.0

The dataset is divided into *Northbound* and *Southbound*, which refer to the sequence of frames generated going one direction on the road and then driving the other direction [Cameron 2016]. The dataset is tiny in terms of lighting and road situations. The centre images were selected from this dataset since stereo images are not required for the experimental setup for an end-to-end system as described in Chapter 3.

The dataset is biased towards going straight versus turning the vehicle. According to Spryn and Sharma [2018] this is a typical bias for a driving dataset as most of the driving of a vehicle in the real world will have a steering angle close to zero, i.e. going



(a) Uncropped example frame



(b) Cropped frame

Figure 4.1: Comparison of a Udacity Dataset Sample Before and After Cropping the ROI [Cameron 2016]

forward (with some minor error in sensing). In order to ensure that the bias does not get transferred to any trained models, straight line data points were randomly dropped to match the number of data points representing a turning steering angle. This batch of data points was then split between train, validation and test sets which were used for the training and evaluation of each model. Table 4.2 shows the data spread between samples where the steering angle represents driving straight and turning the vehicle.

ROI Cropping Example

The image dimensions for this dataset are 640x480 pixels. The ROI selected, cropped the images to 640x310 pixels (as seen in Figure 4.1), a closer aspect ratio to what Spryn and Sharma [2018] used when training with Microsoft's AirSim Tutorial Dataset.

4.7.2 Microsoft's AirSim Tutorial Dataset

Reasons for selecting the AirSim Tutorial Dataset

Microsoft's AirSim Tutorial Dataset, as described in Section 2.10.1 was used by Spryn and Sharma [2018] when training the Autonomous Cookbook model. This is a tiny dataset generated in the AirSim simulator. It is a good dataset for evaluating trained models and specialises in steering angle prediction.

Table 4.3: AirSim Tutorial Dataset Summary

Data Group	Total Count	Straight Line Image Count	Swerve Image Count	Swerve Ratio
Dataset Before Drop	46738	34813	11925	0.3425

The dataset is divided between six groups of data points driving straight and three groups of turning. The bias is similar to the Udacity Self-Driving Car Dataset. In this case, no dropping of data points is required as the whole dataset is used to run tests against



(a) Uncropped example frame



(b) Cropped frame

Figure 4.2: Comparison of a AirSim Tutorial Dataset Sample Before and After Cropping the ROI [Spryn and Sharma 2018]

the trained models and not for the training of models. Table 4.3 shows the data spread between samples where the steering angle represents driving straight and turning the vehicle.

ROI Cropping Example

The image dimensions for this dataset are 256x144 pixels. The ROI selected cropped the images to 255x59 pixels. This was adjusted slightly from the original tutorial to have a slightly larger field of view. An example frame from the dataset can be seen in Figure 4.2.

4.7.3 The Cityscapes Dataset

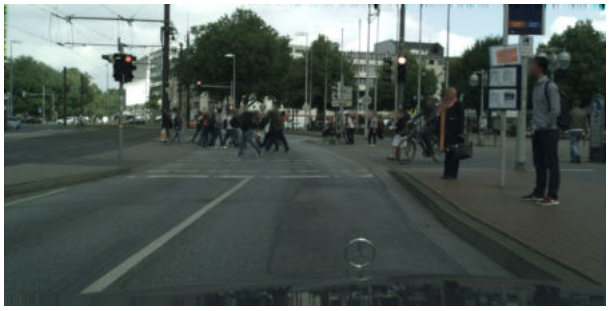
Reasons for selecting the Cityscapes Dataset

The Cityscapes Dataset is used in other research domains such as work using semantic segmentation [Papers With Code 2023]. This makes the dataset an excellent choice to use as a ground truth segmentation dataset for the GradCAM analysis phase. Since this dataset is also a driving dataset with vehicle vectors and RGB frames, it is a good fit for the setup for an end-to-end system as described in Chapter 3.

Table 4.4: Cityscapes Dataset Summary

Data Group	Total Count	Straight Line Image Count	Swerve Image Count	Swerve Ratio
Data Set Before Drop	24997	20053	4944	0.2465
Train + Train Extra	22972	18308	4664	0.2547
Validation	500	408	92	0.2254
Test	1525	1337	188	0.1406

The dataset is divided into four groups: *train*, *train-extra*, *validation* and *test*. To add more training samples for the study, *train* and *train-extra* were combined into a single training subset. Table 4.4 shows that the dataset has a far worse steering bias than the other two datasets. Validation and test data groups have 92 and 188 swerve data points,



(a) Uncropped example frame



(b) Cropped frame

Figure 4.3: Comparison of a Cityscapes Dataset Sample Before and After Cropping the ROI [Cordts *et al.* 2016]

respectively. This is a small amount of data with a harsh bias which makes this dataset less desirable to train models in predicting steering angles.

ROI Cropping Example

The image dimensions for this dataset are 2048x1024 pixels. The ROI selected cropped the images to 1792x530 pixels (as seen in Figure 4.3).

Summary of Segmentation Data in The Cityscapes Dataset

The Cityscapes Dataset contains 30 classes, split into 8 groups in its segmentation data. The segmentation data is also split into two sections: 5000 *fine*, higher quality segmentation maps, and 20000 *coarse* lower quality segmentation maps. Labelled segmentations do not have holes, and the foreground label takes precedence over all other overlapping labels [Cordts *et al.* 2016].

Table 4.5: Table of Cityscapes labels as divided per group [Cordts *et al.* 2016]

Group Name	Classes
flat	road, sidewalk, parking, rail track
human	person, rider
vehicle	car, truck, bus, on rails, motorcycle, bicycle, caravan, trailer
construction	building, wall, fence, guard rail, bridge, tunnel
object	pole, pole group, traffic sign, traffic light
nature	vegetation, terrain
sky	sky
void	ground, dynamic, static

The top three groups in Table 4.5, which are meaningful to driving a vehicle, would be the *flat*, *object*, and *human* groups. These contain most of the meaningful information from the scene to be considered when driving. The *construction* group may also be significant due to the effect buildings and barriers can have on driving. The *sky* group is a far less

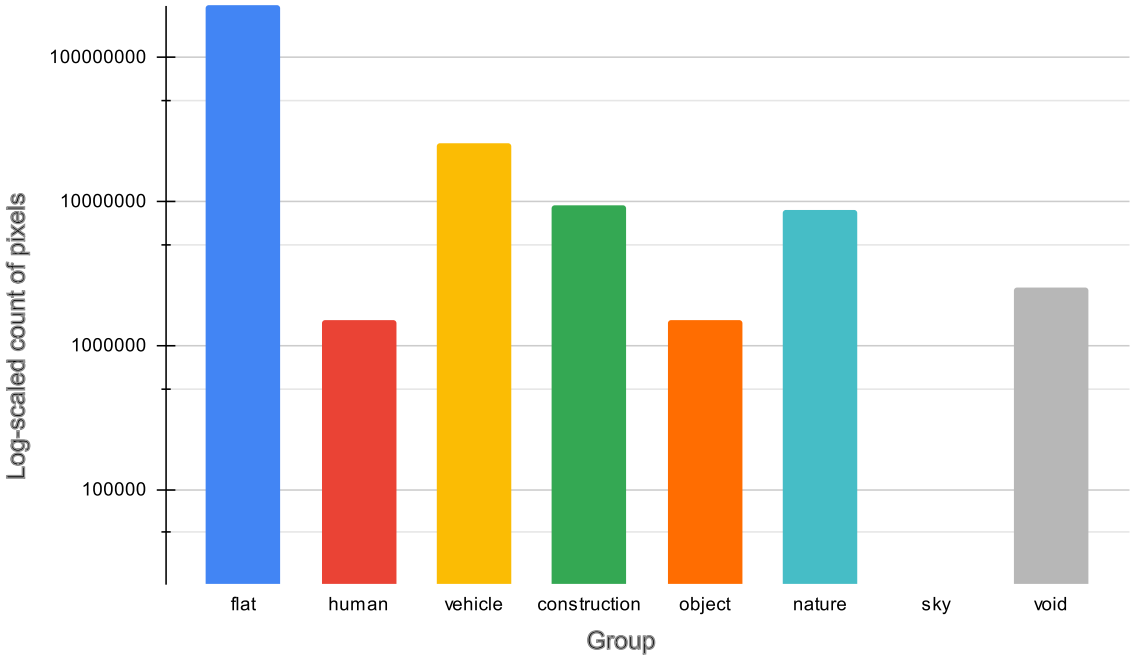


Figure 4.4: Scaled Count of all Segmentation Groups Found in Cropped and Scaled Cityscapes Segmentation Data

significant group with the possibility that *sky* pixels may stand out but may not have as much relevance to driving decisions on the road.

The *flat* group has the most pixels of any group in the Cityscapes Dataset. This is followed by *vehicle* and then *construction*. This bias may skew training models and results towards favouring the *flat* group. However, the *flat* group is one of the most significant groups as it represents the road surface vehicles drive on.

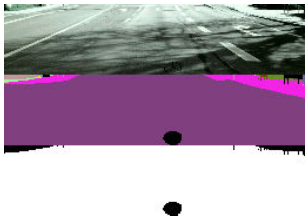
Figure A.1 shows the breakdown of label counts for The Cityscapes Dataset. The *road*, *car*, and *sidewalk* labels have huge individual counts, which map to their corresponding groups and their group counts.

In Figure 4.4, the top two counts per group presented are the *flat* and *vehicle* groups. Conversely, the *sky* group has the smallest pixel count, with a total of 22,068 pixels. This is significantly less than the next smallest group, *human*, which has 1,483,559 pixels. Therefore the *sky* group does not appear to have a pixel value in Figure 4.4 because of the log scaling.

Section A.3, contains a complete break-down of pixel counts per Cityscapes segmentation group.

Examples of Each Cityscapes Group

Figure 4.5 shows an example of each group where that group's pixels were the most of any samples. These examples show how the limited ROI can focus on some groups more



(a) flat



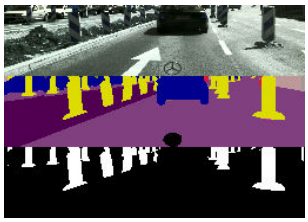
(b) human



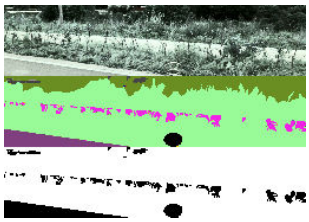
(c) vehicle



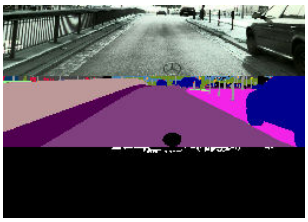
(d) construction



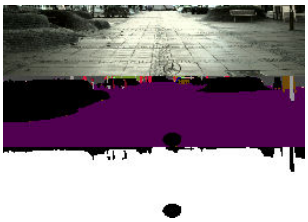
(e) object



(f) nature



(g) sky



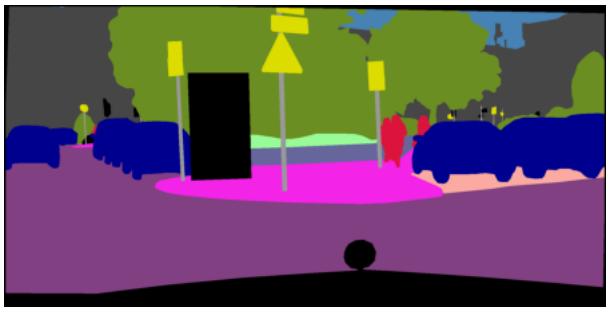
(h) void

Figure 4.5: Examples of each group from segmentations. Three images are shown: the original cropped RGB image, the cropped segmentation map, and the binary map of the group

than others, with the groups with the most prevalence being *flat* and *vehicle*. Examples of each label are shown in Section A.2.

The image dimensions for segmentations are the same as the frames in Cityscapes as seen in Figure 4.6. The segmentation map had the same resolution scaling applied as the input images, but no normalisation of the pixels was performed to ensure that the labels remain positionally the same.

ROI Cropping Example



(a) Original Segmentation



(b) Segmentation After Processing

Figure 4.6: Comparison of a Cityscapes Dataset Sample Before and After Processing Segmentation [Cordts *et al.* 2016]



(a) Uncropped example frame



(b) Cropped frame

Figure 4.7: Comparison of a From Games Dataset Sample Before and After Cropping the ROI [Cordts *et al.* 2016]

4.7.4 The From Games Dataset

Reasons for selecting the From Games Dataset

The From Games dataset was generated to produce a large amount of high quality segmentation data that is label compatible with multiple other datasets including the Cityscapes Dataset. Whilst it does not contain any vehicle control signals, it can be used as a ground truth semantic segmentation dataset as an alternative source of data to the Cityscapes Dataset [Richter *et al.* 2016a].

Table 4.6: From Games Dataset Summary

Data Group	Total Count
Data Set Before Drop	14968
Data Set After Drop	14968

Table 4.6 shows that this dataset is provided in a single group of all image samples and corresponding segmentation labels. Since there are no vehicle control signals, bias of steering angle is not a concern for this dataset.

ROI Cropping Example

The image dimensions for this dataset are 1914x1052 pixels. The ROI selected cropped the images to 1614x420 pixels (as seen in Figure 4.7).

Summary of Segmentation Data in The From Games Dataset

The From Games Dataset uses similar classes and groups to the Cityscapes Dataset, as shown in Table 4.5. The colour codes used in the raw data are different to the Cityscapes Dataset but the dataset's included source code contains a mapping matrix to the Cityscapes values [Richter *et al.* 2016b].

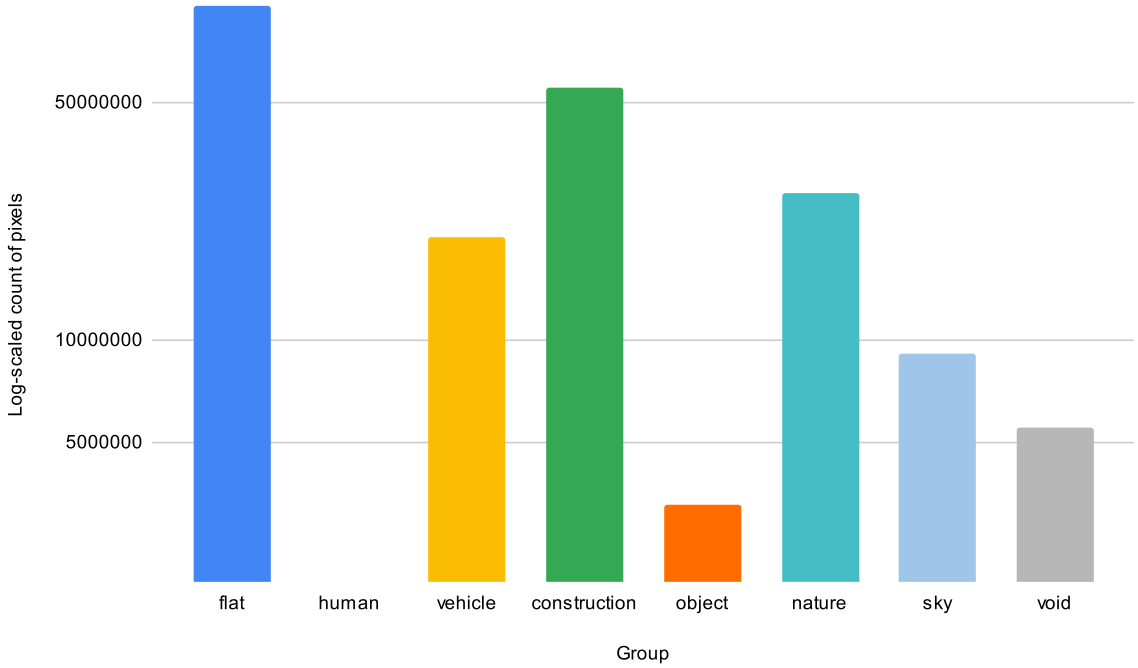


Figure 4.8: Scaled Count of all Segmentation Groups Found in Cropped and Scaled From Games Segmentation Data

Figure 4.8 shows that the From Games Dataset has a different spread of counts per segmentation group as shown in Figure 4.4. The *flat* group is still the largest group in both datasets by orders of magnitude.

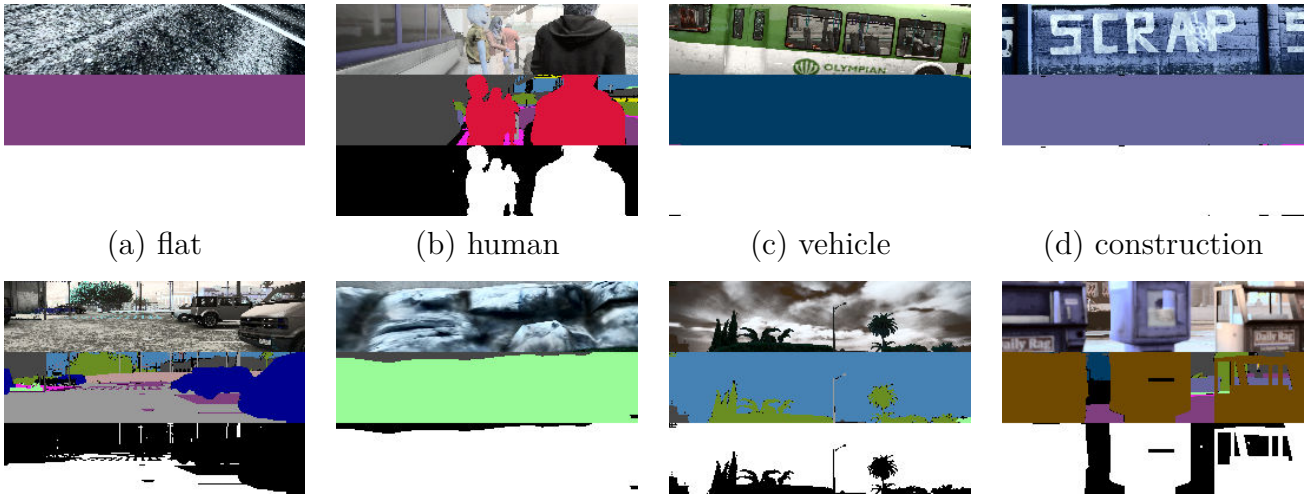
Figure 4.8 shows that the smallest group is the *human* group which is different to the Cityscapes Dataset where it is the *sky* group. The *vehicle*, and *object* groups are smaller in relation to the others in the From Games dataset when compared to the Cityscapes Dataset.

Figure 4.8 finally shows that the *nature*, and *sky* groups are larger in relation to other groups for the From Games dataset when compared to Figure 4.4 of the Cityscapes Dataset.

Examples of Each From Games Group

Figure 4.9 shows an example of each group where that group's pixels were the most of any samples. These examples show how the limited ROI can focus on some groups more than others, with the groups with the most prevalence being *flat* and *vehicle*. Examples of each label are shown in Section B.2.

The image dimensions for segmentations are processed in the same way as the ones from the Cityscapes Dataset, where their dimensions match the image samples, but no normalisation was applied to ensure that the segmentation labels remain positionally the same.



(a) flat (b) human (c) vehicle (d) construction

(e) object (f) nature (g) sky (h) void

Figure 4.9: Examples of each group from segmentations. Three images are shown: the original cropped RGB image, the cropped segmentation map, and the binary map of the group

4.8 Selection of a Viable Epsilon Value for Steering Angle Based on Datasets

Table 4.7: Steering Ratio per Dataset for Different Accuracy Decimal Places

Accuracy	Cityscapes			Udacity			AirSim Dataset		
	Straight	Turning	Ratio	Straight	Turning	Ratio	Straight	Turning	Ratio
1 Decimal Place	20053	4944	0.2465	19102	7618	0.3988	23399	23339	0.9974
2 Decimal Places	6870	18127	0.379	2570	24150	0.1064	8445	38293	4.5344
3 Decimal Places	1248	23749	0.0525	301	26419	0.0114	6954	39784	5.721

All selected datasets use radians to represent steering angle for each data-point used in training, test, and evaluation experiments.

A viable epsilon value needs to be selected to determine which steering angles constitute driving straight and which constitute turning a vehicle. This is also a helpful constant to apply to the MoA metric defined in Chapter 3.

We first define that ‘driving straight‘ is the absolute value of the steering angle being less than the chosen epsilon value, i.e. $|\theta| < \epsilon$.

An analysis was performed comparing an epsilon value at three levels of accuracy (for the radian values). These values were chosen based on the understanding that, in real-world applications, any angular measurement of a steering wheel beyond three decimals places does not create a discernible difference in vehicle direction. The three levels of accuracy were: one decimal place (0.1), two decimal places (0.01) and three decimal places (0.001). Among these options, an epsilon value of 0.1 was selected as it is the smallest value that adequately differentiates between vehicle turning and going straight while avoiding excessive dropping of valid turning data points.

Table 4.7 shows that below one decimal place, there would be more data points of vehicles turning than going straight. The datasets analysed are not random samples but data taken from continuous drives. Spryn and Sharma [2018] affirms that most steering angle data for driving has a straight steering angle. Therefore it would be impossible to have more data-points showing the vehicle turning than going straight so one decimal place (0.1) was selected as the epsilon value.

4.9 Attribution of decisions made by selected models

4.9.1 Methods Applied to Analyse Attribution

The attribution methods discussed in Section 2.6 all produce feature gradient activation maps (colloquially known as heat-maps). An issue with these maps is that they are predisposed to manual visualisation analysis, and it can be challenging to produce a metric which compares different attribution maps against each other [Adebayo *et al.* 2018].

Selvaraju *et al.* [2016] analyses models trained to solve a classification-based problem as depicted in Figure 4.10. In this type of task, each category could be associated with hot spots in the generated gradient map. The input data can help identify activations based on classes. With regression problems, such as predicting steering angle, this is more difficult. The input data does not directly contain class information to extract qualitative reasoning around what classes the models may be activated on, but rather regions of interest in an input image.

We proposed a way to solve this issue using a ground-truth dataset which contains RGB images, a vehicle steering angle set for the same image frame, and an equivalent segmentation map with well-defined categories and boundaries. This segmentation map can then be used to determine which categories activate the model being analysed by comparing the segmentation map to the generated gradient activation map.

In order to properly analyse a model using segmentation and gradient activation maps across a whole dataset such as the Cityscapes Dataset, the analysis needed to be scaled up to count the categories activated by the model across all evaluated data. For a large dataset, this was not feasible to perform via visual analysis, so a statistical regime needed to be devised.

Two methods for combining and counting the categories from the segmentation maps that correspond to the pixel wise indices of the gradient maps were proposed.

The first method involved generating a binary map from the gradient activation map. Here, a threshold is selected, and any value from the gradient map above that threshold is set to 1. Any value below that threshold is set to 0. The segmentation map is then multiplied by the generated binary map. Each category is then counted by pixel in the resultant map.

The second method proposed involves counting the segmentation map per pixel for each category and weighting each pixel by the corresponding value of the gradient activation map. This process requires looping through each available category in the dataset and

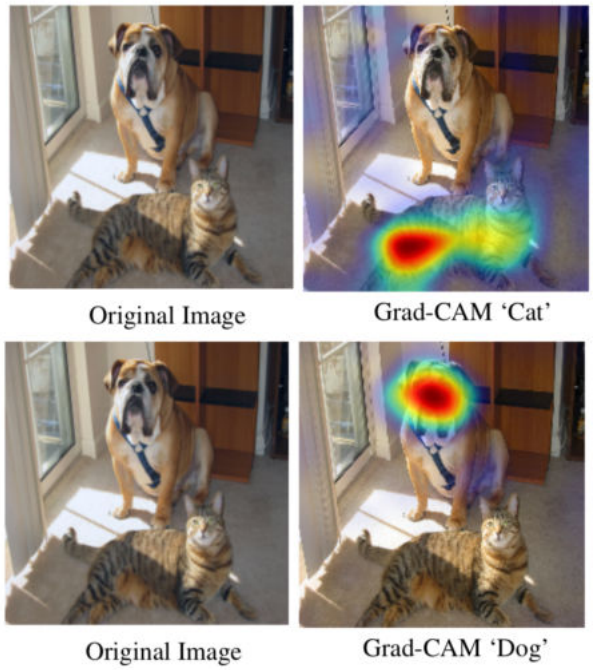


Figure 4.10: Example gradient maps produced by GradCAM [Selvaraju *et al.* 2016]

then generating a sum of all corresponding heat-map values representing each pixel of the corresponding segmentation class. This will create a weighted sum corresponding to the generated heat-map per segmentation class. These generated values can then be further processed.

The first method had the potential to ignore valuable information that was contained in the gradient map but was below the threshold. The second approach produced counts much smaller in value due to a spread out and small series of gradient activation values in the generated maps.

For both methods, a form of scaling was used as Figure 4.4 showed that the categories in the ground truth dataset will not have the same total pixel counts. The pixel counts also had orders of magnitude difference between the useful groups from the Cityscapes Dataset. For each final category count produced from either method, a percentage was generated by dividing the final count for each category by the total count of each category calculated from the original segmentation maps. The final percentage values, therefore, did not represent the percentage across the whole dataset, but rather the percentage of pixels of that class that influenced the model.

For any given model evaluated, each category can be compared to one another. However, the magnitude between the two models should not be compared to each other as the raw value amounts may not correspond to a measure of the importance of a category. The general trends and shape of category percentages between models can be compared and assessments made. Directly comparing values of the categories per epoch does not represent a useful comparison, but comparing values between the categories for each epoch, will show useful information about the change of the behaviour between model

epochs i.e. which categories have more importance to a specific model epoch and that change as the epochs change. The shape of the profile is the useful outcome of this kind of analysis technique. A pilot study was performed to determine whether a threshold value should be selected or the weighted gradient sum method should be used.

4.9.2 Definition for applying a threshold to the activation map and counting the resultant pixels

Let the set of all semantic segmentation groups be defined by:

$$K = \{K^{\text{flat}}, K^{\text{human}}, \ldots\} \tag{4.1}$$

where $k \in K$ are the groups as defined in Table 4.5. For example $K^{\text{flat}} = \{$road, sidewalk, parking, rail track$\}$.

Then let a binary pixel map indicating positions in the image where the semantic segmentation label is within a specific group k be defined as:

$$P^{(k)} \in \{0,1\}^{(w \times h)} \text{ where } k \in K \tag{4.2}$$

For example P^{flat} contains a 1 everywhere where the semantic segmentation class is in K^{flat}. This effectively creates a one-hot encoding for each pixel assigning it to a group, such that:

$$P_{ij}^{k} = \begin{cases} 1 & \text{if the class of pixel } (i,j) \in k \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

Then let $N^{(k)} \in \mathbb{N}$ be the number of pixels in that group such that:

$$N^{(k)} = \sum_{i=1}^{w} \sum_{j=1}^{h} P_{ij}^{k} \tag{4.4}$$

The heat-map from the GradCAM result is then defined as:

$$M \in \mathbb{R}^{(w \times h)} \tag{4.5}$$

Then the result $G^{(k)}$ is defined as:

$$G^{(k)} = \frac{P^{(k)} \odot \mathbb{1}_{[M>\theta]}}{N^{(k)}} \cdot 100 \tag{4.6}$$

where $\odot$ is the element-wise product (Hadamard product), and $\mathbb{1}_{[M>\theta]} \in \{0,1\}^{(w \times h)}$ indicates the positions in M (the GradCAM heat-map) above the threshold θ.

4.9.3 Definition of the GradSUM analysis scheme

Let the set of all semantic segmentation groups be defined by:

$$K = \{K^{\text{flat}}, K^{\text{human}}, \ldots\} \tag{4.7}$$

where $k \in K$ are the groups as defined in Table 4.5. For example $K^{\text{flat}} = \{\text{road, sidewalk, parking, rail track}\}$.

Then let a binary pixel map indicating positions in the image where the semantic segmentation label is within a specific group k be defined as:

$$P^{(k)} \in \{0,1\}^{(w \times h)} \text{ where } k \in K \tag{4.8}$$

For example P^{flat} contains a 1 everywhere where the semantic segmentation class is in K^{flat}. This effectively creates a one-hot encoding for each pixel assigning it to a group, such that:

$$P_{ij}^{k} = \begin{cases} 1 & \text{if the class of pixel } (i,j) \in k \\ 0 & \text{otherwise.} \end{cases} \tag{4.9}$$

Then let $N^{(k)} \in \mathbb{N}$ be the number of pixels in that group such that:

$$N^{(k)} = \sum_{i=1}^{w} \sum_{j=1}^{h} P_{ij}^{k} \tag{4.10}$$

The heat-map from the GradCAM result is then defined as:

$$M \in \mathbb{R}^{(w \times h)} \tag{4.11}$$

Then the result $G^{(k)}$ is defined as:

$$G^{(k)} = \frac{P^{(k)} \odot M}{N^{(k)}} \cdot 100 \tag{4.12}$$

where $\odot$ is the element-wise product (Hadamard product).

Pseudo-code for the GradSUM analysis scheme

Algorithm 1 An algorithm for the GradSUM scheme

$K \leftarrow \{K^{\text{flat}}, K^{\text{human}}, \ldots\}$ ▷ The available segmentation groups in ground truth dataset
for k in K **do**
  for w, h in InputImage **do**
    if InputImage[w,h] is in group k **then**
      $P[k][w][h] \leftarrow 1$
    else
      $P[k][w][h] \leftarrow 0$
    end if
  end for
  $N[k] \leftarrow \text{Sum}(P[k])$ ▷ This is the sum of all pixels present for the given group k
  $M \leftarrow \text{GradCam}(\text{InputImage, model})$
  $G[k] \leftarrow \frac{P[k] \odot M}{N[k]} \cdot 100$ ▷ G is the GradSUM result, and $\odot$ is the element-wise product
end for

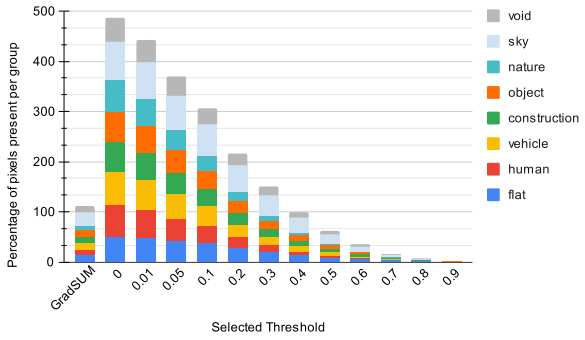
4.10 Pilot Study: Should a threshold value be used with an attribution map?

In this pilot study, the *End-to-End* model used by Bojarski *et al.* [2016b] is trained to 300 epochs and the model epoch with the best validation autonomy is selected. GradCAM is performed on the untrained (first) and best epoch of the training set, and then the model profiles used to analyse the attribution are computed for the untrained and best epochs. GradSUM and a varying scale of threshold values are compared against each other.

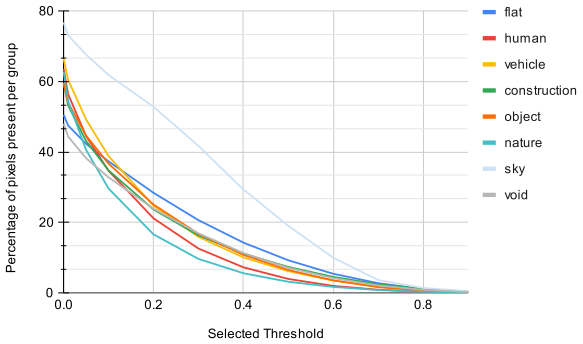
The percentage of each pixel group, per model epoch (excluding GradSUM) is calculated by taking the number of pixels in a specific group that have an activation above the threshold and dividing that by the total number of pixels present for that pixel group. For GradSUM the percentage is calculated by taking the sum of all pixel activation for a specific pixel group divided by the total count of that pixel group. This is because the gradient map's per pixel value is ranged between 0 and 1, determined by how much effect a pixel had on the weights within the evaluated model epoch.

4.10.1 Results from the untrained epoch

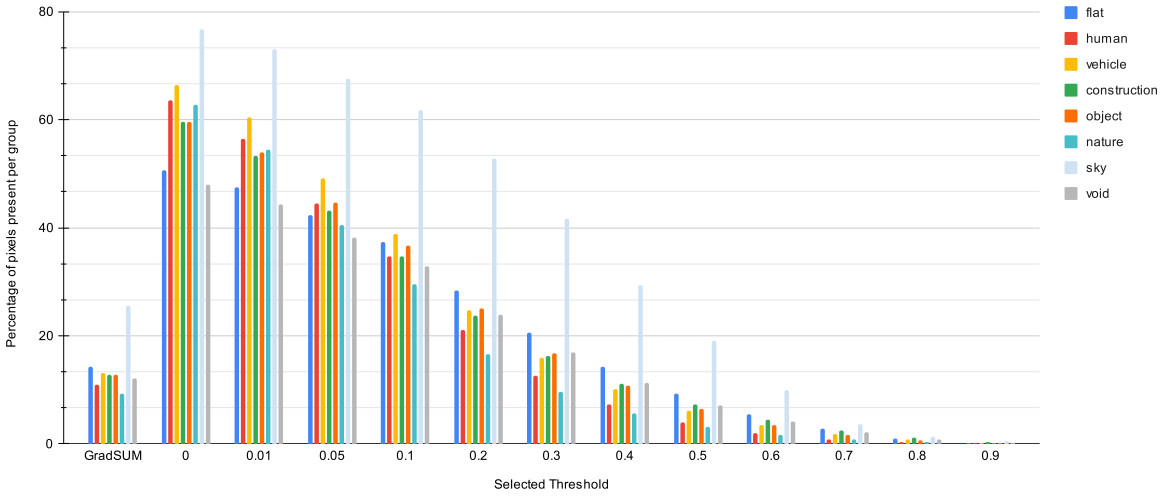
Figure 4.11 (a) shows that GradSUM has considerably smaller values than threshold values up to a threshold of 0.4. However, when looking at the trend lines of the different selected threshold values in Figure 4.11 (b), the relative values of groups compared to the other groups within a profile plot (specific threshold or GradSUM) can differ. This could cause analyses of the output to change. GradSUM does not suffer from this issue as it takes all available activation data into account.



(a) Stacked chart for each threshold (Epoch 1)



(b) Trends for each threshold (Epoch 1)

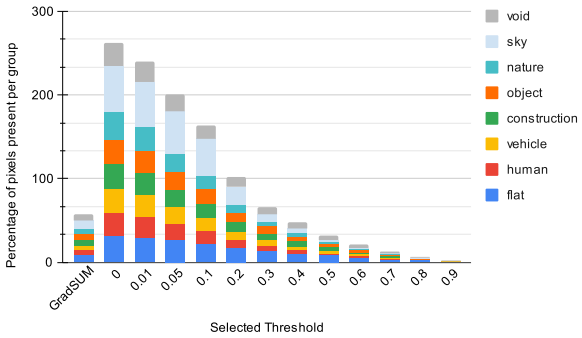


(c) Profiles generated for each threshold group and also GradSUM (Epoch 1)

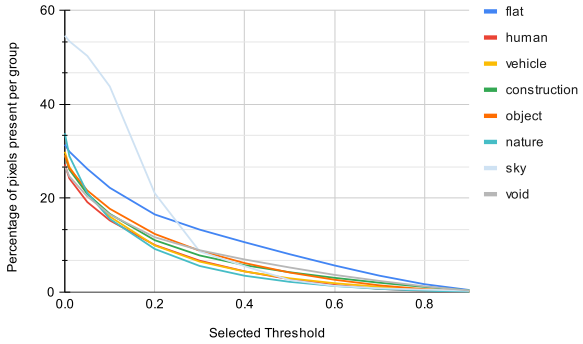
Figure 4.11: Effect of threshold value on the spread of signal for epoch 1

4.10.2 Results from epoch 296

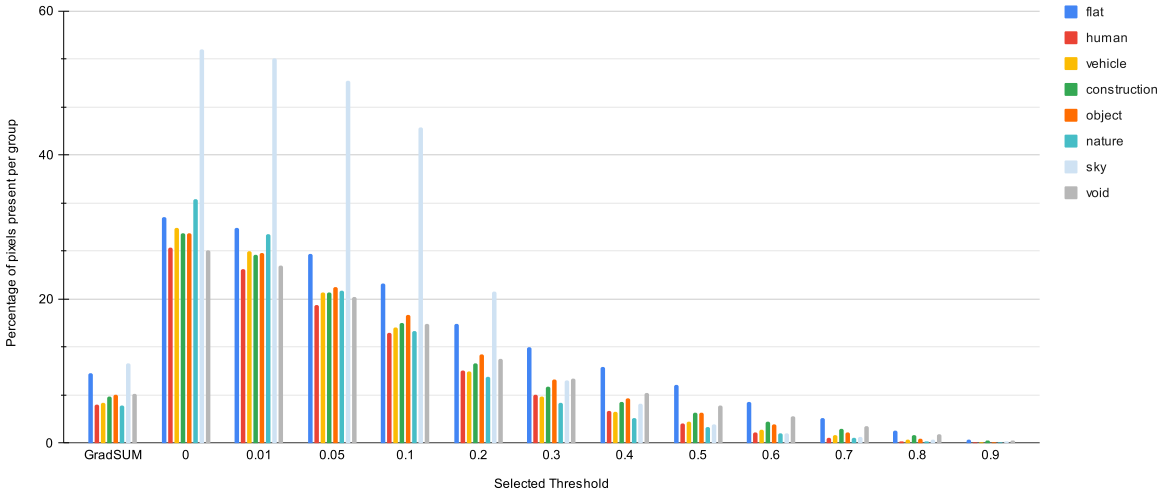
Figure 4.12 shows similar overall general results to the previous figure when comparing different thresholds and GradSUM model profiles. Threshold values appear to change the importance of groups within the plot relative to their peer groups. An added observation is that a far more trained model has smaller values per group and per threshold, but specific patterns in the relative importance of groups can still be observed.



(a) Stacked chart for each threshold (Epoch 296)



(b) Trends for each threshold (Epoch 296)



(c) Profiles generated for each threshold group and also GradSUM (Epoch 296)

Figure 4.12: Effect of threshold value on the spread of signal for epoch 296

4.10.3 Discussions on the results of the pilot study on epoch 1 and 296

When looking at the results of epoch 1 from Figure 4.11 (c), it can be observed that different threshold values change the shape of the grouped bar chart and different groups take up more importance based on the threshold. This is a problem as any observations on the model behaviour based on the relative importance of each group could be obscured by selecting the wrong threshold value. Again GradSUM does not suffer from this problem because it takes all the available gradient data into account. However, it does produce results with smaller values. Having smaller values may make determining the importance

of one Cityscape group over another more difficult, but the relative value of one group over another for a model epoch can be compared.

When evaluating which method should be used to produce model profiles, several factors have to be weighed up. Selecting a threshold value appears to be a critical decision for the first method defined in Section 4.9.2. This value can change the shape of each model profile. It's also a method that eliminates a certain amount of the generated heat-map. There is an implicit assumption that any value below the threshold value is insignificant to the overall behaviour of the model. This assumption may not be true, and models potentially could use correlations of much smaller activations over many more input neurons to produce an output.

The GradSUM method defined in Section 4.9.3 (and algorithm 1) takes into account all the data contained in the generated heat-maps. However the magnitude of the values per each segmentation group can be small when looking at the trained model epoch on Figure 4.12 (c) versus a very small threshold value or the equivalent model profile from the untrained model in Figure 4.11 (c). However the shape of the profile and the comparative values within a single model profile are important in assessing the behaviour of a model rather than comparing model profiles directly against other model profiles in terms of the magnitude of the individual profile group values.

We chose the GradSUM method because there is more confidence that this method takes into account all the generated data from using GradCAM on a specific model, and any further analysis will not look at the magnitudes of the values per segmentation group but rather the overall model profile shapes.

4.11 Pilot Study: Comparing GradSUM and Traditional GradCAM Analysis

Section 2.8 discussed analysis techniques used in conjunction with GradCAM. Some of these techniques involved using saliency and activation maps generated from evaluating an input sample when predicting an output for a binary classification problem. The relationship between the input data and the output heat-maps was evident due to the class information implicit in the input data. The regression task being evaluated in this work does not have classes in the input data, that can be used to determine what regions of interest the heat-maps are identifying. A more classical problem in this domain would be a binary classification problem where a CNN model must determine if a specific group is present in the input image.

The aim of this pilot study was to show that using the GradSUM scheme to identify biases in a fully trained *End-To-End* model performing a binary classification matches up to the expected result which is that the bias found using the scheme reflects the group which the model is trained to classify it's presence in a given image.

Pixel group percentages are calculated the same way as in Section 4.10 for GradSUM.

Table 4.8: Comparison of each Cityscapes segmentation group sample counts for the simple binary classification problem

Dataset Split	train			val		
Group	Yes	No	Total	Yes	No	Total
flat	2975	0	2975	500	0	500
human	2864	111		478	22	
vehicle	2892	83		489	11	
construction	2975	0		500	0	
object	2964	11		498	2	
nature	2906	69		486	14	
sky	1366	1609		222	278	
void	2975	0		500	0	

4.11.1 Selecting the labels for classification

Table 4.8 shows each group found in the fine Cityscapes segmentation dataset. The counts are divided into two groups, *Yes*, and *No*. Each row shows the number of images in the dataset containing that group. The subset of data used for this pilot study was processed using the pre-processing steps outlined in Section 4.6.

Table 4.8 shows that some groups can not be used for this pilot study because the groups appear in every sample. These groups are the *flat*, *construction*, and *void* groups. There are another two groups removed because of extreme data imbalance in the training or validation sets. These groups are the *object*, and *nature* groups.

The three groups left were used for this pilot study, namely *human*, *vehicle*, and *sky*. The labels selected for this study were the *person* label from the *human* group, the *car* label from the *vehicle* group, and the *sky* label from the *sky* group..

For the *End-to-End* model trained on the *person* label binary classification problem, it was expected that the GradSUM analysis would show that the group which has the highest activation on the model would be the *human* group. A similar result was expected for the models trained with the *car*, and *sky* labels respectively.

4.11.2 The experimental setup

The *End-to-End* model was modified to use the Sigmoid function [Goodfellow *et al.* 2016] on the output layer to ensure the output was between $[0, 1]$. According to Goodfellow *et al.* [2016] the Sigmoid function is a basic function used for binary classifications with others such as Soft-Max being expansions on the original Sigmoid function.

The main training and evaluation data was generated from The Cityscapes Dataset. The test set of data was ignored since it does not contain any valid segmentation data as that information is not public so that models can be blindly evaluated with segmentation data those models have never been trained on [Cordts *et al.* 2016].

A second evaluation was performed using another ground truth dataset, the From Games Dataset. This evaluation of each biased model is done to strengthen the results of using

The Cityscapes Dataset. This dataset has not been seen by any of the models evaluated and because the source of the data is so different there should be no correlation between the two datasets. A benefit of using the From Games Dataset is that it has compatible segmentation labels to The Cityscapes Dataset [Richter *et al.* 2016a].

A new output was created for the dataset used. This was done by returning either a 0 or 1, determined by a sample having any pixels related to the selected label. The input was still the pre-processed RGB image from the given sample. Each model was trained and evaluated against this data.

The first part of this study was to investigate if 100 random models per group would have any observable bias towards any groups when using the GradSUM analysis. This was to determine if the observed bias in the classification problem was due to the model architecture or because the training introduced the bias. This was performed on all groups, using a selected label (with the most even split between *Yes* and *No* values) from each group. Figure A.2, lists all the Cityscapes groups, and their label sample count breakdown for the classification problem.

For the second part of this study, enhancements to training the models as described in Section 2.11 and also in Section 4.14 was used. Ten models with randomised small weights, as described in Section 2.11.4 were generated. The model with the best validation loss after ten epochs was fully trained to classify the given label. The aim of these improvements was to limit over-training and attempt to select the initial model weights with the best performance to limit poor results due to poor initialisation.

The GradSUM analysis was performed with the *fine* segmentation maps from The Cityscapes Dataset. This was done to minimise any error or noise being introduced into the GradSUM model profiles due to misclassification of the labels in the segmentation maps, which the *coarse* set of segmentation maps may contain.

4.11.3 Results of the pilot study

Figure 4.13 shows that none of the random results achieved a GradSUM percentage for any group above an average of 1%. This means that none of the models tested had any relevant bias or correlation for any of the selected labels tested against 100 randomly initialised models per label.

Table 4.9 shows that the trained models performed better than randomly initialised models shown in Figure 4.13 at predicting whether the label was present in the given sample.

Figure 4.14 shows that for the model which classified if a *person* was in an image or not, the group with the highest relative percentage of activation was the *human* group. For the model that classified if a *car* was in the scene, the group *vehicle* has the highest classification percentage. The last model which classified if an input image contained pixels from the *sky* label had the *human* group have the highest percentage with the *sky* group coming in second.

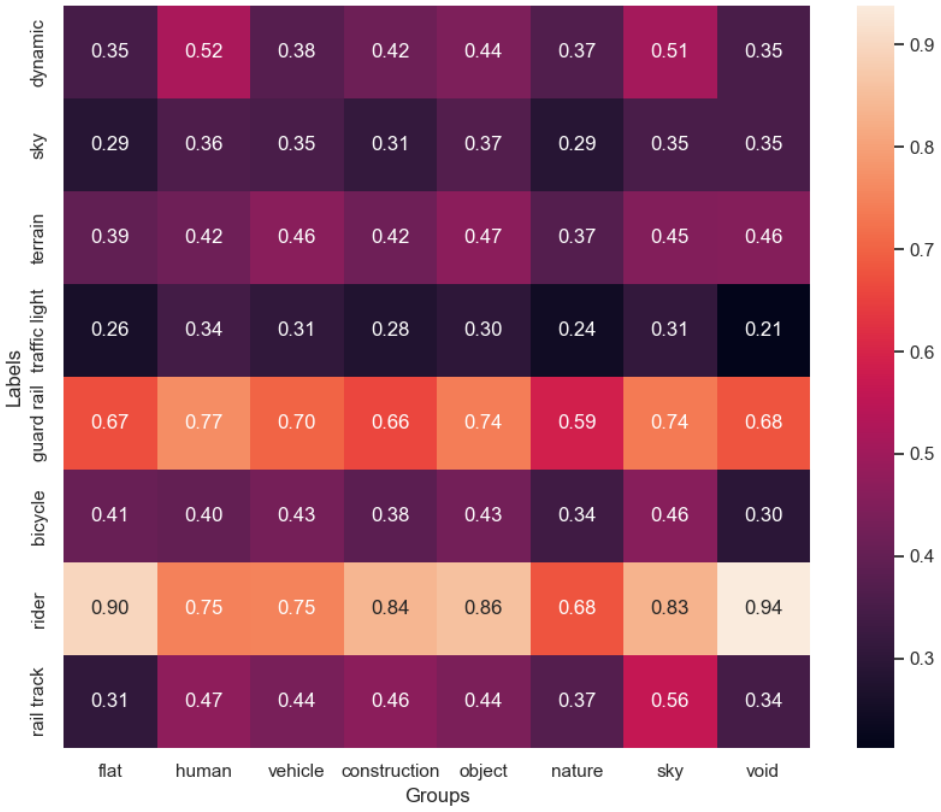


Figure 4.13: GradSUM Results (Percentages per Group) for 100 Random Models Solving the Binary Classification Problem (Averaged Results per Group)

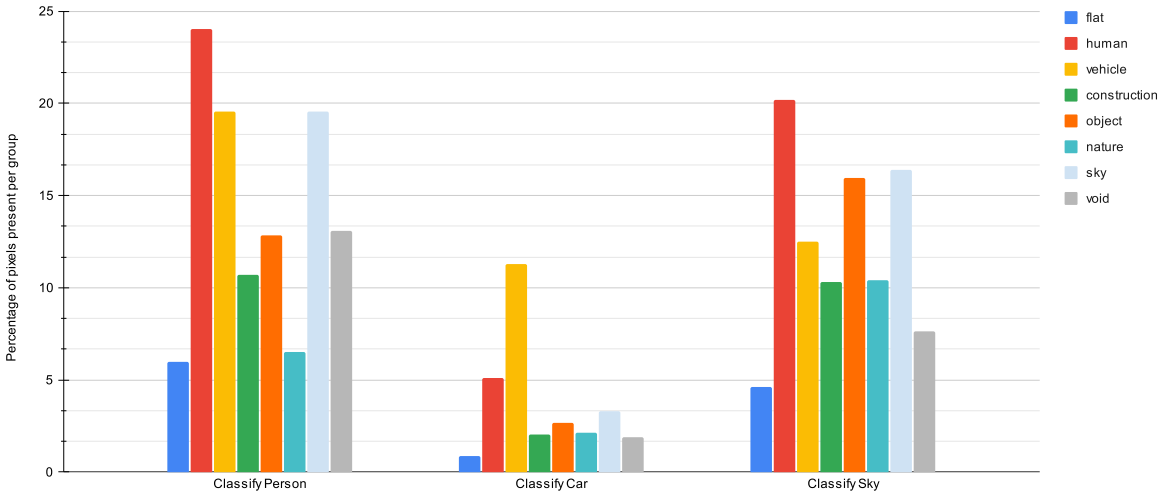


Figure 4.14: Comparison of the GradSUM Model Profiles for the Trained *End-to-End* Models (Using The Cityscapes Dataset)

Table 4.9: Loss results from the trained models for the binary classification problems

Task	Train Loss (MSE)	Validation Loss (MSE)
Classify Person	0.14860	0.1026
Classify Car	0.17786	0.0450
Classify Sky	0.20802	0.2274

These results make sense for the *person* and *car* labels because the models were trained to identify specific labels (which are part of specific groups of labels) and ignore the other labels. So it would be expected that the trained model profiles would show this bias.

The results in Figure 4.14 for the *sky* label classification could indicate problems with the model or the data. The third highest group in the results shown in Figure 4.14 was the *object* group. The fact that *human* group had the highest percentage values, and the *object* group had the third highest percentage results, could indicate that there are spurious correlations that exist with these groups in the dataset used for training and evaluation. This could have affected the results for the *sky* label experiment where those correlations could be more evident. However the fact that the *sky* group had the second highest percentages still shows that the training was able to introduce a bias towards the *sky* group, which was less prevalent in the other results in Figure 4.14 and not at all evident in Figure 4.13 (the random experiment).

An interesting observation can be made that the *car* binary classification problem appears to have smaller relative percentages (in Figure 4.14) for each group and a larger difference between the highest activated group (by percentage) than the other groups. This can be explained by the more considerable amount of data from the dataset where there are *cars* in the scene and the large number of samples where they are not in the scene. The same can be said of the *sky* results, where the *object* and *human* groups would be present in most of the *sky* samples because of how many samples have these groups present as seen in Table 4.8.

We also observe that in Figure 4.14 the *person* binary classification problem has relative percentages between each group which is much closer than the *car* model profiles when comparing the highest value to the lowest values. The dataset for the *person* experiment had fewer samples where people were present than the *vehicle* experiment. This made training more challenging and allowed for more noise to be introduced. Therefore, fewer model weights tended towards zero, allowing for more feature extraction from other groups in the ground truth dataset. Again the same is true for the *sky* experiment where even more feature extraction from some of the more interesting groups happened.

Results of using the From Games Dataset

Figure 4.15 confirms most of the findings found with using the Cityscapes Dataset in Figure 4.14. The task of classifying if the *person* label is present in the input image shows that the most activated group of pixels is the *human* group.

The same findings hold for classifying if the *car* label is present in the input image because the *vehicle* group has the highest activation percentages for the model profile. A notable

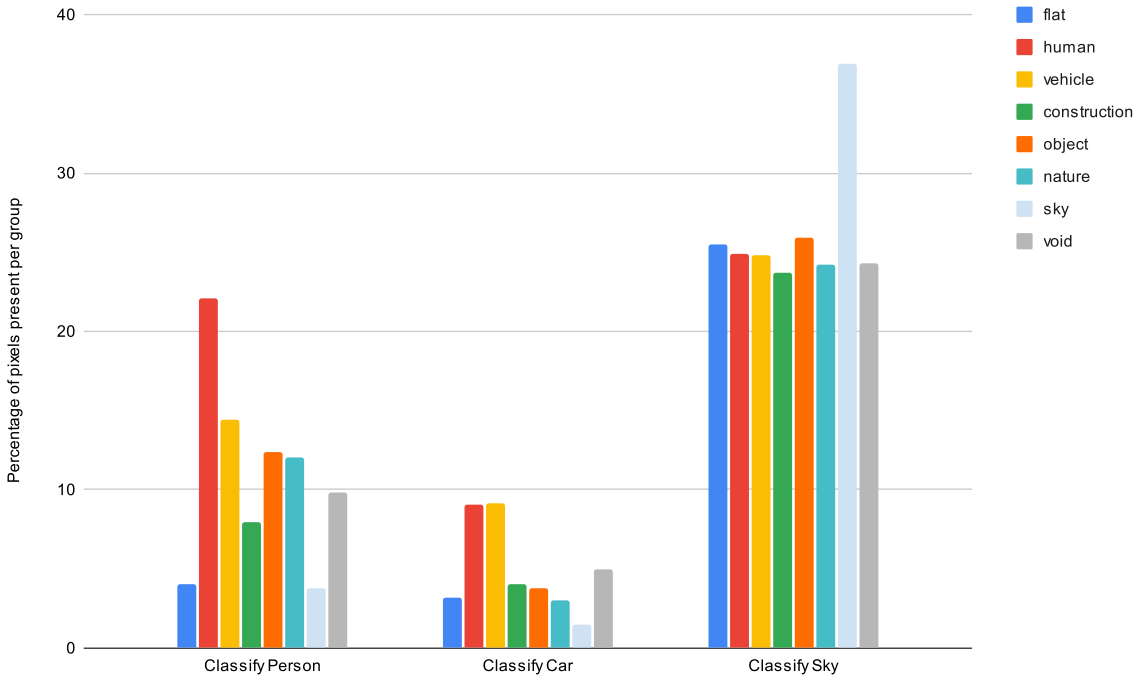


Figure 4.15: Comparison of the GradSUM Model Profiles for the Trained *End-to-End* Models (Using From Games Dataset)

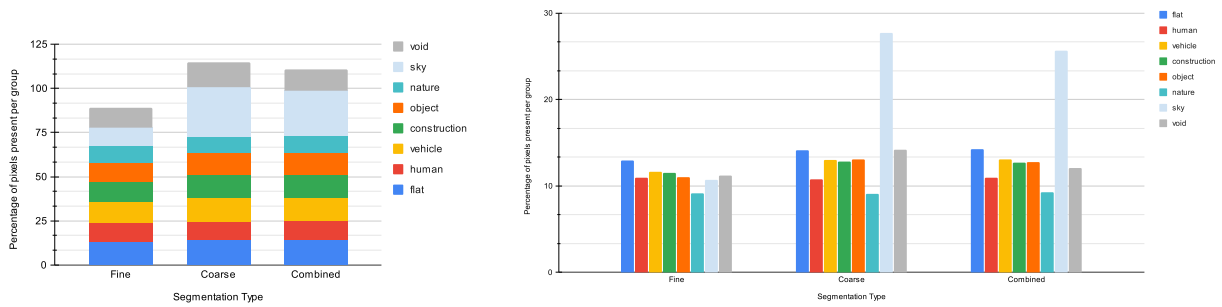
finding in the second test is that the *human* group also has a high level of activation in the model profile, however Table B.1 shows that the *human* group has the least number of pixels in the From Games dataset.

The third test where the model classifies if the *sky* label is present in the input image has the *sky* group as having the most pixel activation as shown in Figure 4.15.

The From Games GradSUM test confirms the result which were found using The Cityscapes Dataset.

4.11.4 Conclusion of the pilot study

It appears that a GradSUM analysis of a CNN model is able to show biases introduced by training that model. However it is sensitive to noise and sample count in the ground-truth dataset. To use this analysis technique would require creating high quality datasets that are large enough for the problem domain. That is an apparent draw-back, however there is still a lot of value in being able to create a model profile whose shape can be compared to other models, and also used to understand a models behaviour against a known ground-truth.



(a) Stacked Chart

(b) Model profiles

Figure 4.16: Comparison of different segmentation types (model profiles) for Epoch 1 (untrained) using GradSUM

4.12 Pilot Study: Comparison of Fine and Coarse Segmentation Maps on GradSUM Results

The Cityscapes Dataset has two different sets of segmentation maps. A lower quality *coarse* set and a higher quality *fine* set [Cordts *et al.* 2016].

A question arose about which set to use for the ground truth used to analyse GradCAM, or perhaps if both sets should be combined. The segmentation maps' accuracy would impact the model profiles generated and if the *coarse* segmentation maps' error is substantial enough it is able to skew an analysis of what groups are activating a model's weights.

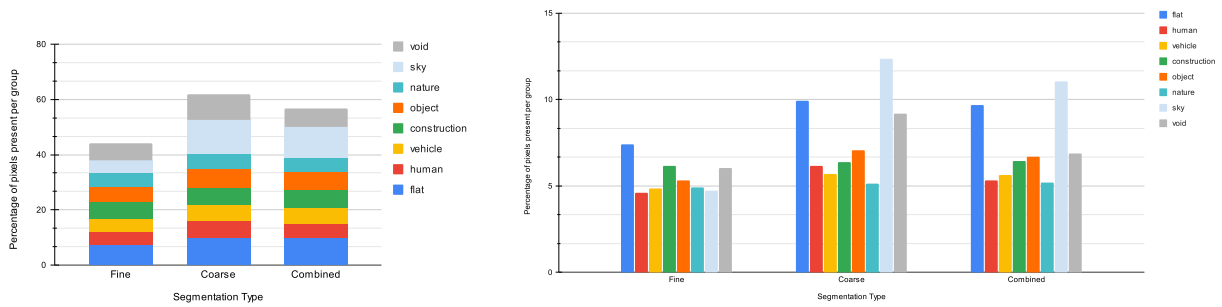
The pilot study was also performed with the *End-to-End* model architecture but using the regression task of predicting steering angle. The model was trained to 250 epochs.

4.12.1 Results from epoch 1 (untrained)

In the first epoch test, shown by Figure 4.16, the relative shapes of the model profiles were different between the *coarse* and *fine* segmentation types. The combined variant was similar to the *coarse* group because of how many more *coarse* samples there were over the number of *fine* samples.

4.12.2 Results from epoch 249

Figure 4.17 shows the same outcome for epoch 249 as with epoch 1 (untrained), where the resultant model profile was skewed by the vastly more extensive and less accurate *coarse* set of segmentation data. Here, the relative effect of the *sky* group over other groups was exaggerated when *coarse* segmentation maps were included in the model profile.



(a) Stacked Chart

(b) Model profiles

Figure 4.17: Comparison of different segmentation types (model profiles) for Epoch 249 using GradSUM

4.12.3 Discussion on the results of the pilot study

The model profile was directly changed by which segmentation data was used. The *coarse* set of segmentation maps has far more data but is far less accurate than the *fine* set of segmentation maps in The Cityscapes Dataset. This could be a problem if the error is large enough to change which groups are more important in the model profiles during the analysis phase. Combining the two sets of segmentation data posed the same problem as just using the *coarse* segmentation maps. For this reason, only the *fine* segmentation maps were used for the analysis phase. This was to limit any potential bias being introduced due to an error in the ground truth data rather than an issue in the training or evaluation of the model in question.

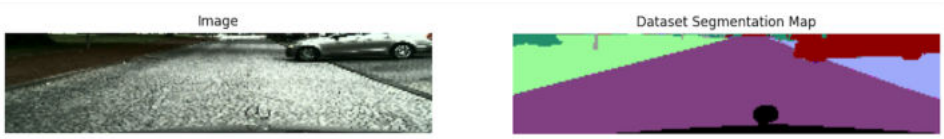
4.13 Canny Edge Map Results on the Cityscapes Dataset

Edge detection methods are a basic image processing technique used to generate a binary map showing only the edges of an image. The Canny Edge Detector is such an algorithm. Edges are the outlines of shapes or regions within an image distinct from other objects or regions in the image [Rosebrock 2021].

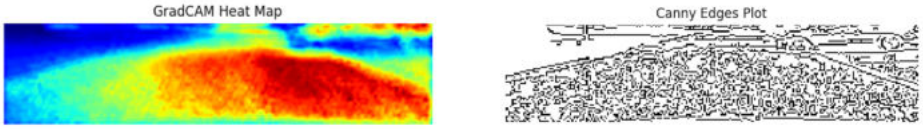
Since edge detection produces maps, we wanted to answer the question, is it possible for the models used for self-driving cars to be performing edge detection? If so, then the resultant model profiles produced may reflect the same shape and pattern as a profile generated using an edge map over a gradient activation map generated from the model.

Since the edge map is a binary map, it is possible to multiply it by the segmentation map, and then the pixels per group can be counted. The same scaled percentage value calculated for GradSUM can also be used, where the count of a group's pixels is divided by the total occurrence of that group's pixels. This is the exact same procedure defined in Section 4.9.3 where the Hadamard product is used, however the GradCAM heat-map M is substituted for the canny edge map.

The Canny Edge detection algorithm was implemented into the OpenCV Python library. The selected parameters are intended to produce a *medium* Canny Edge Map [Rosebrock



(a) Original RGB image and corresponding *fine* segmentation map



(b) Resultant GradCAM and Canny Edge Maps

Figure 4.18: Example of a Canny Edge Map Generated on the Cityscapes Dataset

2021]. The lower hysteresis threshold selected was 30, and the upper threshold selected was 200.

Figure 4.18 (a) shows an example input image from the Cityscapes Dataset and its corresponding segmentation map. Figure 4.18 (b) shows the comparison between a generated GradCAM heat-map and a generated canny edge map. The scene which is shown in the example is fairly typical of the Cityscapes Dataset frame. It shows a forward facing RGB image from a vehicle being driven on a real road. The scene shows the road surface, side-walk, vegetation, and a vehicle on the side of the road. The GradCAM heat-map has a scale from blue to red, with red representing pixels which have a higher impact on activation in the model being observed, and blue representing very low activation. The canny edge map contains found edges in the input frame. It is very noisy due to the road surface being cobbled and not smooth.

Pixel group percentages are calculated in a similar way as in Section 4.10 but instead of the absolute values of each pixel in the map or a threshold against the map being used, the sum of all edges per pixel group are divided by the total number of pixels per group.

4.13.1 The Resultant profile generated with Canny Edge Maps across the whole *fine* Cityscapes Segmentation Dataset

The resultant profile in Figure 4.19, generated with the Canny Edge Detector algorithm applied to the whole *fine* segmentation set, showed some group variation. This variation was approximately ±4 percent. This slight variation can likely be attributed to varying amounts of pixels for each pixel group in the dataset. We can therefore infer that edges are generally consistent between semantic groups (with a ±4 percent margin of difference).

Based on this observation, it is plausible to conclude, that if a model profile demonstrates significant differences between pixel group results within the same model profile, that the natural edges in the dataset (for all pixel groups) did not have any influence on those results or the model behaviours which generated those results.

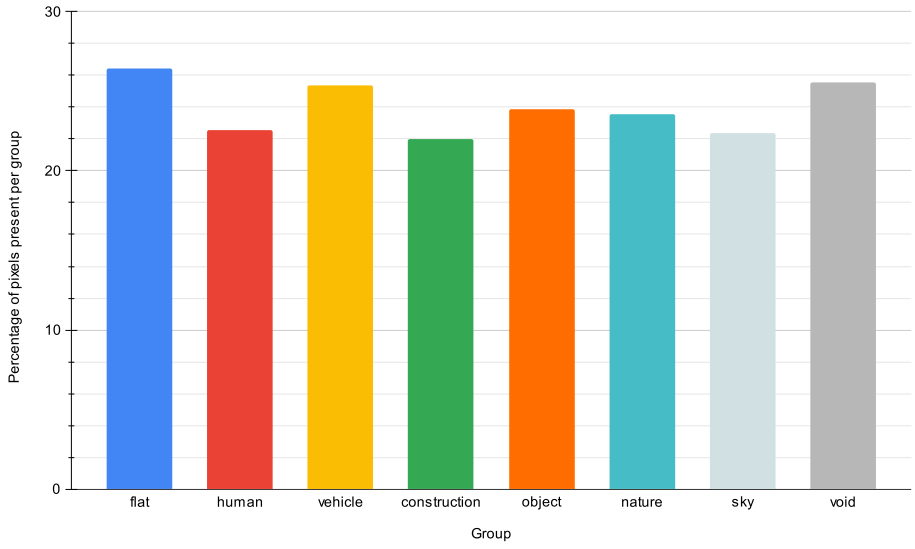


Figure 4.19: Resultant canny edge group profile

4.14 Training Details

The regression task was selected to be the primary task all the models being evaluated would be trained to perform.

4.14.1 Pre-training steps

Most of the pre-training steps applied to the experimental setup replicates the work done by Spryn and Sharma [2018]. ROI cropping is applied to all input data so that the main focus is the road surface and the surroundings. The alpha channels of all images are also removed. For the larger datasets, the cropped images also have been rescaled using nearest neighbour interpolation, and then the pixels of the images have been scaled between $[0, 1]$. Section 4.6 details the steps of pre-training steps.

4.14.2 Model initialisation

As described in Section 2.11.4, random weights were implemented for all models. When the models were initialised, they would have small random weights using the Xavier Uniform random weights implemented in the PyTorch Machine Learning Framework [2023].

Ten versions of each model were initialised and trained to 10 epochs. The best validation error from these batches of initial models was then selected and trained to convergence. This was done to reduce the possibility of a bad random initialisation of a model since there was a spread of possible initial random behaviour. This is discussed further in Chapter 5.

4.14.3 Early stopping conditions

The early stopping condition, as described in Section 2.11.3 was applied to the training of the model architectures. Two conditions were set. One was against the MSE loss where the training would only stop if the validation loss did not change for 5 epochs. The other condition was against the MoA metric, and the distance was set to 20 epochs since autonomy improved at a slower rate than the MSE loss during training.

4.15 Technical Details of the Experiments

The infrastructure provided by the Mathematical Sciences Support unit at the University of the Witwatersrand was used to perform many of the experiments in this research. A personal workstation was also used.

The two systems used were a system called Kraken and another system (the personal workstation) called Fractal.

Kraken has two Nvidia GeForce GTX 1080 Ti graphics accelerator cards which were used to train, evaluate and analyse models. Each graphics processing unit (GPU) has 11GB of memory. Kraken also has 62GB of system memory and 32 logical processors.

Fractal has 96GB of memory, 12 logical processors and two GPUs. One Nvidia RTX 3060 with 12GB of memory and a Nvidia RTX 4070 Super also with 12GB of memory. The two GPUs both have similar architectures with new features, such as tensor and ray tracing cores that deep learning programming libraries can take advantage of for faster compute.

The abundance of memory and available resources was used to run the experiments with increased parallelism in order to decrease the amount of time needed to compute the results of this work.

Multiple systems were employed in these experiments due to local issues with the available electrical supply. By having resources in different electrical zones with varying levels of backup power, it was possible to run portions of the experiments when an electrical supply was available to maximise how much time was available to complete all required computations.

GitHub [2023][1] was used for source code tracking and automated testing via GitHub Workflows. Automated tests were created for essential components and calculations of the experimental workflow to ensure that sane results were returned for these critical parts of the experiments that we ran. The dataset sampling mechanism, the computation of metrics such as autonomy, and the implementation of GradSUM were tested. This added a level of confidence to the results discussed later on.

The Python Programming Language [2023] was the primary programming language used. The Ruby Programming Language [2023] was another programming language used to per-

[1]Repository link for this work: https://github.com/TRex22/masters-gradsum

form some ancillary data processing. The PyTorch Machine Learning Framework [2023] was the main deep-learning library used to implement, train, and evaluate the models. NumPy [2023] and OpenCV [2023] were two additional libraries used for specific image processing and other calculations. PyTorch library for CAM Methods was the programming library used to implement GradCAM operations [Gildenblat and contributors 2021]. Weights & Biases [2023] was used for tracking the training and evaluation results of the models in real-time.

Some custom modifications were made to the PyTorch library for CAM Methods so its performance was improved for GPU computation. This helped reduce the required time to complete the GradCAM computations.

4.16 Research Plan

Outline of the setup
The main components to this research are shown in Figure 4.20 and the high-level steps are: 1. Train the models on the Udacity Dataset
2. Evaluate the models against the test datasets
3. Analyse the models using a ground truth dataset (The Cityscapes Dataset) using GradSUM
4. Re-run main model architectures 10 times using GradSUM and produce statistical results of model architecture behaviour over-time.

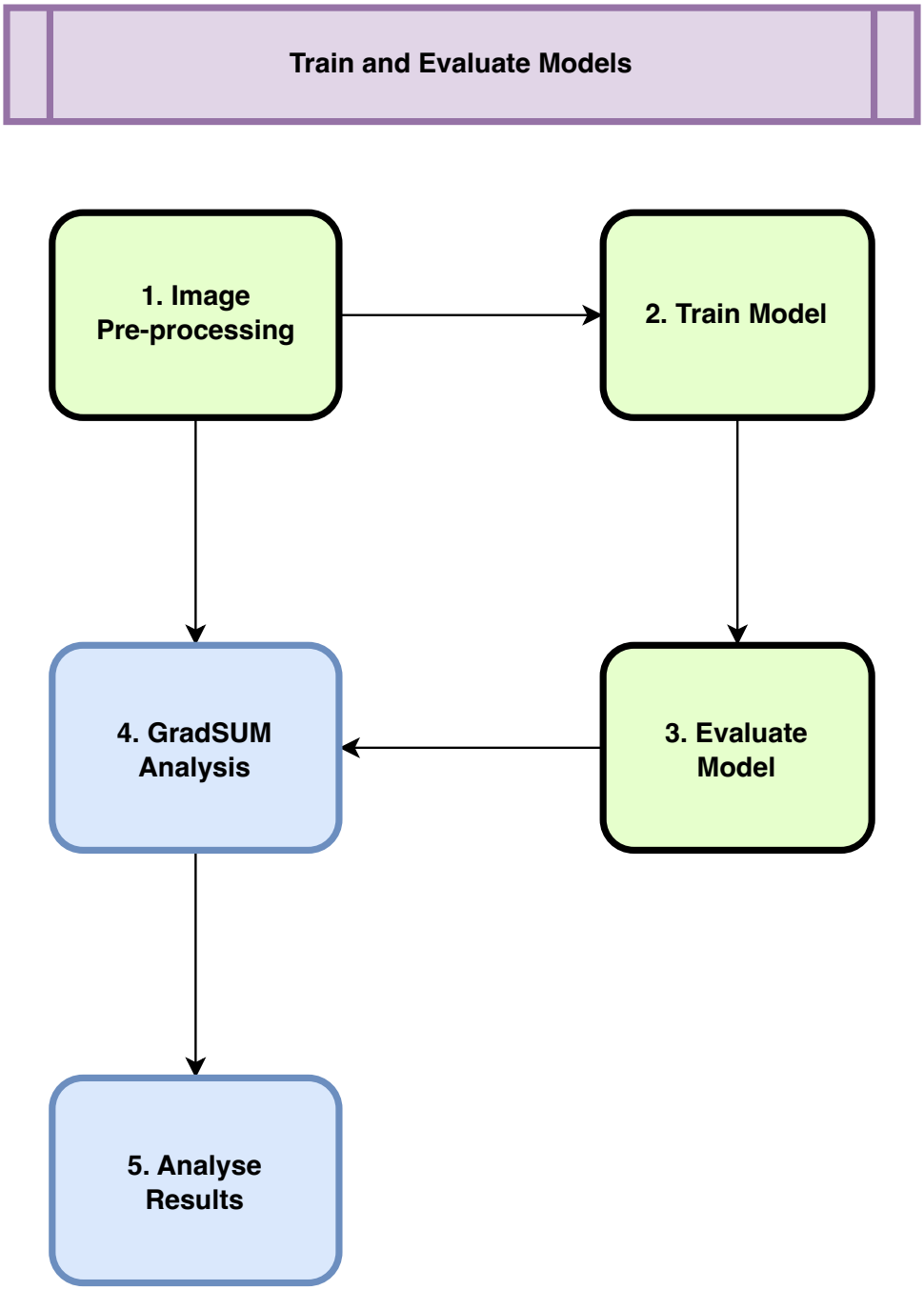


Figure 4.20: Flow diagram of experimental flow followed in the analysis of end-to-end models

4.17 Conclusion

In order to answer if a trained CNN model which can predict steering angle given an RGB image, can learn meaningful features from the input data, GradCAM analysis of the model in question was performed. Four model architectures were selected to be evaluated from the literature, and two additional models were used as controls.

The Cityscapes Dataset was a natural choice for a ground truth dataset due to its *fine* segmentation maps [Papers With Code 2023]. The Udacity Dataset was chosen for training the models due to its number of data points. Microsoft's AirSim Tutorial Dataset was chosen as an extra dataset to evaluate trained models and compare them since the dataset specialises in steering angle prediction without much extra information being present.

It was important to drop data-points from the used datasets to balance the datasets in terms of steering angle during training (see Section 4.7), since any bias between swerve and straight data-points would cause the models to poorly train.

Attribution was made using the GradCAM algorithm. The gradient activation maps produced by this algorithm generate a heat-map which represents which pixels in the inputted image data had the largest effect on the model's output predictions. These maps were analysed against the entire *fine* Cityscapes segmentation maps. The resultant pixels per group were processed using GradSUM and was computed for a range of trained epochs for each model. The resultant model profiles per epoch were analysed to determine model behaviour as the models were being trained.

A test of the effect of edges found in the groups of The Cityscapes Dataset *fine* segmentation maps was performed. It was found that the groups behave similarly when using their edge maps and counting the number of activated pixels per group. This implies that results per model should not have a sizeable observable effect due to the edges of each group.

The following chapters will discuss an analysis of the results produced following this chapter. From these results, a conclusion can be made about what each model can learn from a sparse input signal, being a forward facing image from a vehicle driving on a real-world road.

Chapter 5

Experimental Results

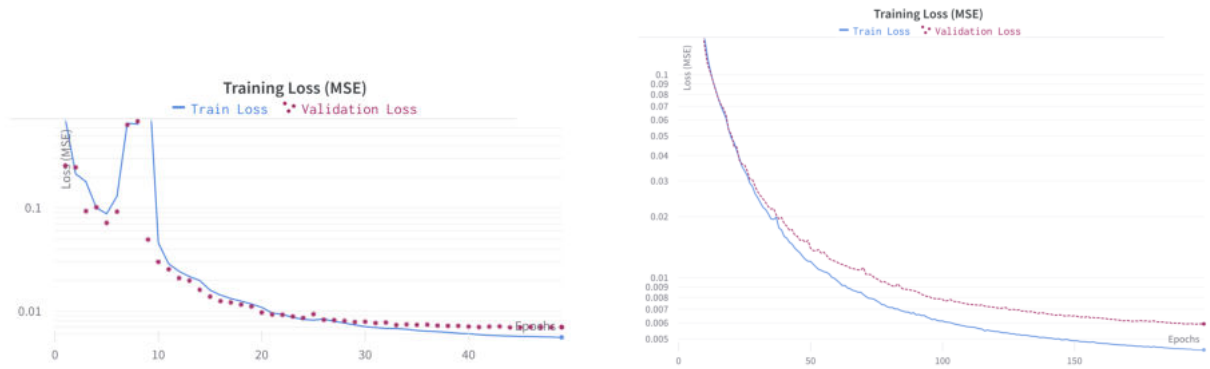
5.1 Introduction

The selected model architectures were trained, evaluated, and had GradSUM performed for the regression task of predicting steering angle. This generated a lot of data to analyse. When determining the behaviour of each model, all the relevant results collected for each architecture must be evaluated together and not in isolation. This section lists the results computed during the training phase, the evaluation phase, and the produced GradSUM results using the ground truth dataset (the Cityscapes Dataset). Training was performed on the Udacity Dataset, and evaluation used the testing subset sampled from the Udacity Dataset, the entire Cityscapes Dataset and Microsoft's AirSim Tutorial Dataset. Randomised experiments were performed on each model architecture, testing 100 randomly initialised models to produce their evaluation metrics of MSE loss and autonomy. The same random experiment was performed to produce GradSUM model profiles so that generalised observations of how these model architectures perform before being trained could be made. Subsequently, a GradSUM analysis was performed on successive epochs for each model architecture to show how the models focus on certain Cityscapes groups and how that changes over training time (epoch).

Section 5.2 considers the training and validation metrics for all six evaluated model architectures. Section 5.3 considers the evaluation (testing) results for the models. The GradSUM analysis results are shown and discussed in Section 5.4. Chapter 6 contains further discussion and analysis of each model based on all generated results.

5.2 Training Results

All the models were trained and evaluated using two early stopping rules. Early stopping was implemented to reduce over-fitting as discussed in Section 2.11.3 and Section 4.14.3. The first rule tracked the model epoch with the best validation MSE result and would stop training if the validation result did not positively change in 5 epochs. The second early stopping rule tracked the MoA of the model over training epochs and would stop training if the metric did not change over 20 epochs. The model epoch with the best



(a) NetSVF (b) NetHVF

Figure 5.1: Comparison of training and validation loss per model (over each epoch) for *NetSVF* and *NetHVF* Architectures

validation MSE result was the selected as the final trained model during other evaluation tests discussed below. Each model architecture used the model initialisation scheme discussed in Section 4.14.2 where the models were initialised 10 times with small random weights, trained for 10 epochs, and the best model by validation MSE was selected to be trained until convergence.

5.2.1 Training Results per Model

NetSVF Training Results

Figure 5.1 (a) presents the training and validation loss graphs over epochs for the *NetSVF* model, from epoch 0 to 44. Epoch 44 is where the *NetSVF* model stopped training due to the early stopping rules during training.

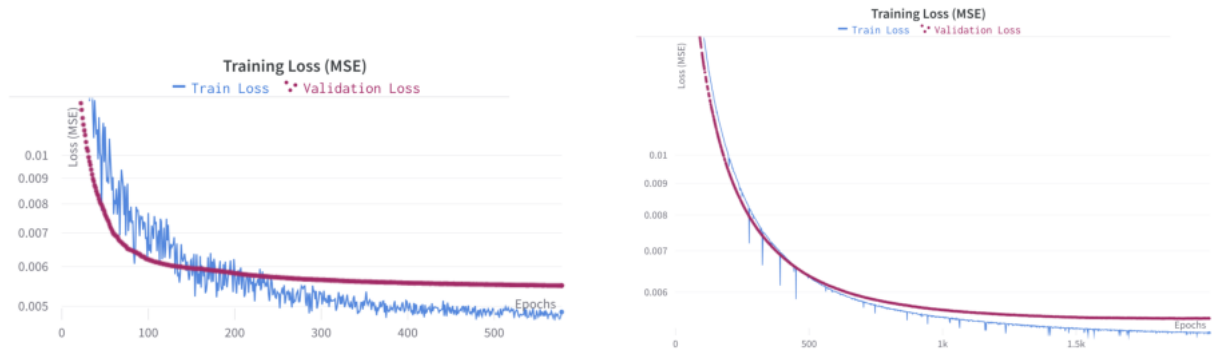
In Figure 5.1 (a) at around epoch 25 an observation can be made where the training loss becomes smaller than the validation loss. Both the training and validation loss continued to decrease but the rate of the training loss decreasing was bigger than that of the validation loss per epoch. The training loss was expected to have a smaller value than the validation loss, and the validation subset was much smaller than the training subset of data as described in Section 4.7.3.

NetHVF Training Results

The *NetHVF* model took many more epochs to finish training (via early stopping). Figure 5.1 (b) shows that the model converged at epoch 196. As expected the validation loss was larger than the training loss by the end of the training sequence.

End-to-End Training Results

The *End-to-End* model architecture had a noisy training error and a smoother validation error, as seen in Figure 5.2 (a). The *End-to-End* model required many more epochs than



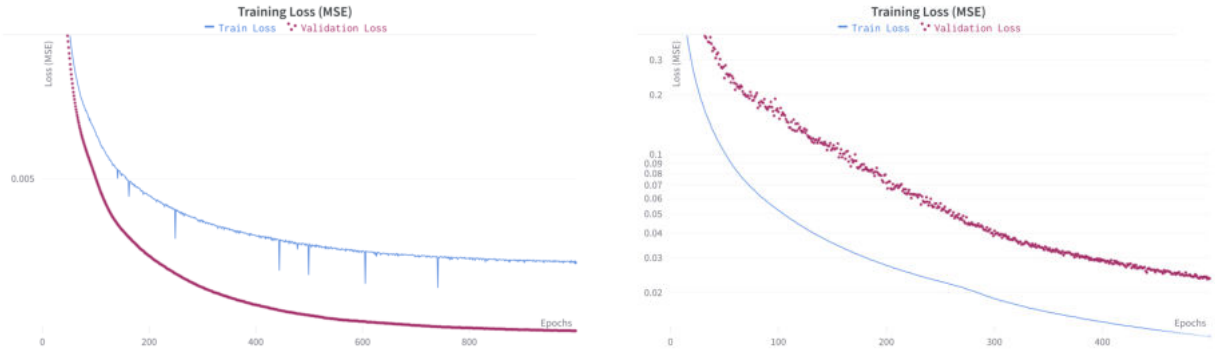
(a) *End-to-End* (b) *Autonomous Cookbook*

Figure 5.2: Comparison of training and validation loss per model (over each epoch) for the *End-to-End* and *Autonomous Cookbook* Architectures

the *NetSVF* and *NetHVF* models to train to convergence. The best model (based on validation loss) was at epoch 573.

Autonomous Cookbook Training Results

The *Autonomous Cookbook* model required even more epochs to train than the previous and subsequent models. The best model was found at epoch 1877. The training loss had many points which appeared to be local minima but continued training showed that convergence happened later on.



(a) TestModel1 (b) TestModel2

Figure 5.3: Comparison of training and validation loss per model (over each epoch) for TestModel1 and TestModel2 Architectures

TestModel1 Training Results

TestModel1 was the only model with the training loss having a larger value than the validation loss, as seen in Figure 5.3 (a). The early stopping conditions triggered at epoch 995 which is the second largest epoch after the *Autonomous Cookbook* model. Early stopping helped reduce over-fitting in this case as the improvement of validation loss slowed to a rate below the early stopping condition.

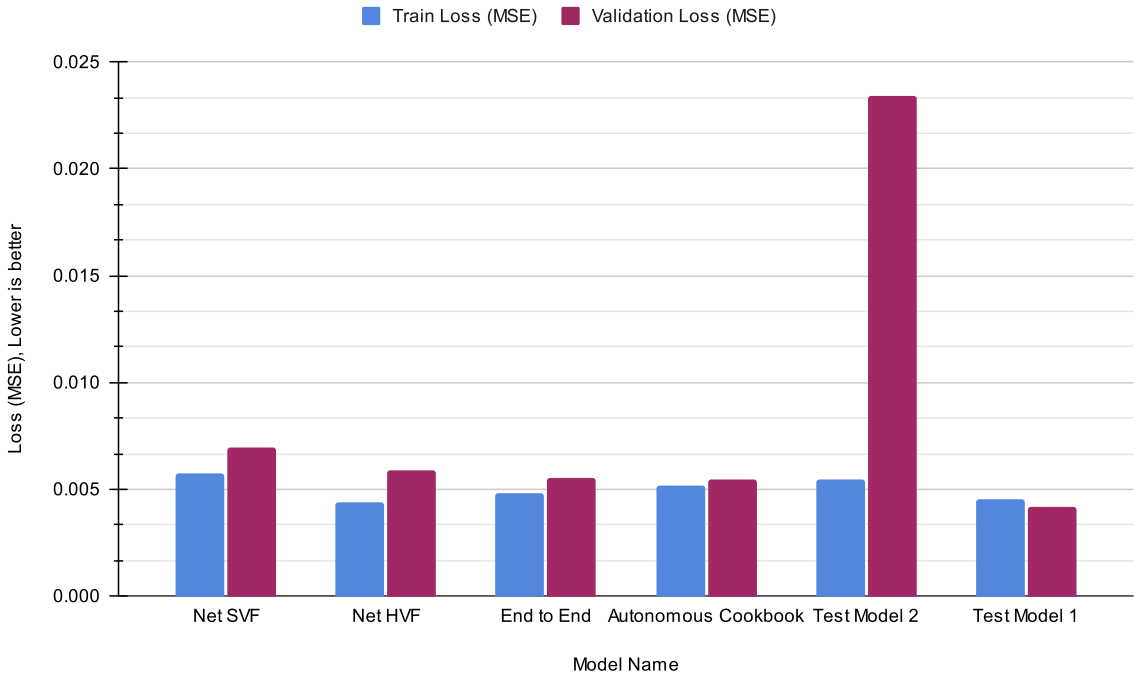


Figure 5.4: Comparison of Model Train Loss and Validation Loss

Table 5.1: Results of the Best Model Epochs During Training

Model Name	Train Loss (MSE)	Validation Loss (MSE)	Best Epoch
Net SVF	0.005742651447566322	0.006963569619613937	44
Net HVF	0.004415138194091477	0.005913741746917367	196
End-to-End	0.004822185874894955	0.005494360290146594	573
Autonomous Cookbook	0.005174464495624152	0.005440801723549763	1877
Test Model 2	0.005440801723549763	0.023412920887871034	497
Test Model 1	0.004514714197950883	0.004143353202380240	995

TestModel2 Training Results

Figure 5.3 (b) shows that *TestModel2* had the largest range between the validation and training losses, with the validation loss being much larger. The model stopped training at epoch 497.

Comparing Training Results Between All Model Types

Table 5.1 and Figure 5.4 show that for *NetSVF*, *NetHVF*, and *End-to-End* model architectures, more epochs were needed to reach convergence as the architecture complexity decreased from the largest model, *NetSVF*, to the smallest of the three being the *End-to-End* model architecture. *Autonomous Cookbook* model needed the most epochs to train to convergence, and *TestModel2* required half the number of epochs than *TestModel1*.

Comparison between *TestModel1* and *TestModel2*

Figure 5.4 shows that *TestModel2* had the worst validation loss and that *TestModel1* had the smallest validation loss. Both model architectures were designed for their simplicity rather than their ability to predict steering angles when driving a vehicle. *TestModel1* was also the only model with a larger training loss than validation loss at the end of training.

The difference between the training loss and the validation loss during a training cycle can provide some insights into a model's ability to generalise and show if it has over-fitted. In Figure 5.4 *TestModel1* has a smaller difference between training and validation loss than *TestModel2*. Figure 5.3 shows that the training and validations losses (over epochs) decrease for *TestModel1* and *TestModel2*. This difference in the training and validation losses between the two models may indicate that the trained *TestModel1* model is more generalisable than the trained *TestModel2* model.

An interesting observation is that the validation loss between *NetSVF*, *NetHVF*, and *End-to-End* was improved as the complexity of the model architecture decreased. This could indicate that the less complex model architectures are able to more easily train to the features which correlate more closely to the valid prediction of steering angle for the samples used for validation. More complex architectures may require more data to reach the same level of loss as the simpler architectures or may learn different features that are themselves more complex.

These observations are made in isolation by only looking at the training and validation losses of each model which does not necessarily indicate the real world performance of these trained models. Other factors of the models such as their architectures have to also be considered. Further analysis will be done using GradSUM, the MoA defined in Chapter 3, and other techniques to help compare the model architectures.

Analysis of *TestModel2* having a larger validation loss than training loss

The validation loss for *TestModel2* is larger than its training loss as shown in Figure 5.3 (b). A possible reason for this could be that the initial weight randomisation employed may not have generated any valid initial models since only 10 randomly initialised starting models were selected.

To further investigate if this was caused by model initialisation 10 complete variants of *TestModel2* were trained until early stopping was triggered. Each model started out by training 10 randomly initialised models, selecting the best one, and then completing the training regime on that model.

Figure 5.5 shows that for some of the model runs validation loss is larger than training loss but in other cases the opposite is true. This leads to the conclusion that the model initialisation is most likely the cause of this result. However the first *TestModel2* will be kept and used for further evaluation since its error metrics are interesting as its an under-performing model variant structure (due to drop-out being employed). The intention of both *TestModel* architectures is to be used as a juxtaposition of better architectures in understanding good and bad predictions made by all these model architectures.

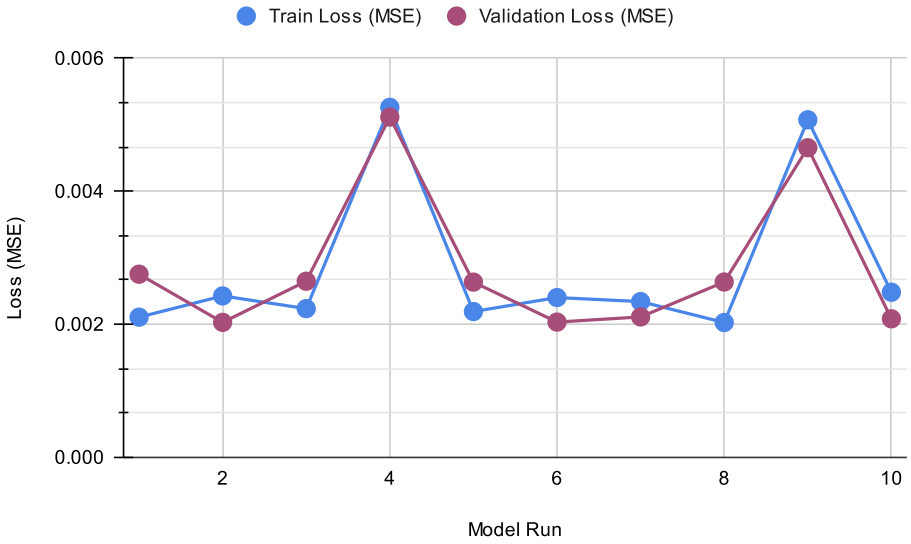


Figure 5.5: Comparison of Model Train Loss and Validation Loss

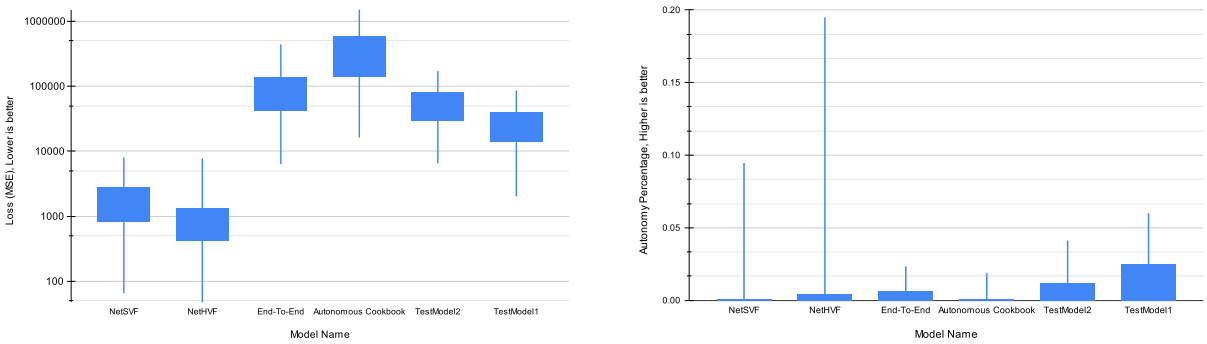
5.3 Testing Results

5.3.1 Random Test Results

In Section 2.11.5, we discuss how Cao and Wu [2022] found that even randomly initialised CNN models can identify objects from input images. These tests aim to determine if randomly initialised models can predict steering angle. This helps determine if models that have ǵoodŕesults in predicting steering angles from input images is due solely to the model architecture or if the training strategy plays an important role in producing this behaviour. It also helps to show that the GradSUM results are able to delineate the difference between an untrained model and a trained model. This aims to show that a model requires training on relevant data in-order to pick up semantic and useful information for driving a vehicle which aligns to the work by Adebayo *et al.* [2018] in determining if a method like GradSUM is dependant on the relationship between the input data and the output predictions.

Figure 5.6 (a) shows the range of test loss that 100 randomly initialised models produced (using Xavier Normalisation as described in Section 2.11.4). *Autonomous Cookbook* had the largest maximum loss. *NetSVF* and *NetHVF* had losses for random initialised weights that were two orders of magnitude smaller than the rest of the model architectures. This shows that all the architectures with randomly initialised weights perform poorly (in terms of MSE loss) on average when predicting steering angle. For acceptable results these model architectures have to be trained. This implies that training these architectures does produce better results and that the model architectures by themselves (without training) did not intrinsically make any evaluated model perform better before being trained.

However *NetSVF* and *NetHVF* which are the most complex of the models evaluated have a performance (MSE loss), when randomly initialised, that is two orders of magnitude



(a) Test Loss Results

(b) Autonomy Results

Figure 5.6: Spread of results from running the Udacity Self-Driving Car Dataset on Models with small Random Weights

better than the simpler models. This means that these models should theoretically train to convergence or early stopping in less epochs than the simpler models.

Figure 5.6 (b) shows that the autonomy of all the randomly initialised models was below 0.20%. This means that none of the evaluated model architectures with random weights had a good MoA value and that training is required to get an acceptable value which would be over 50%.

5.3.2 Test Results per Model

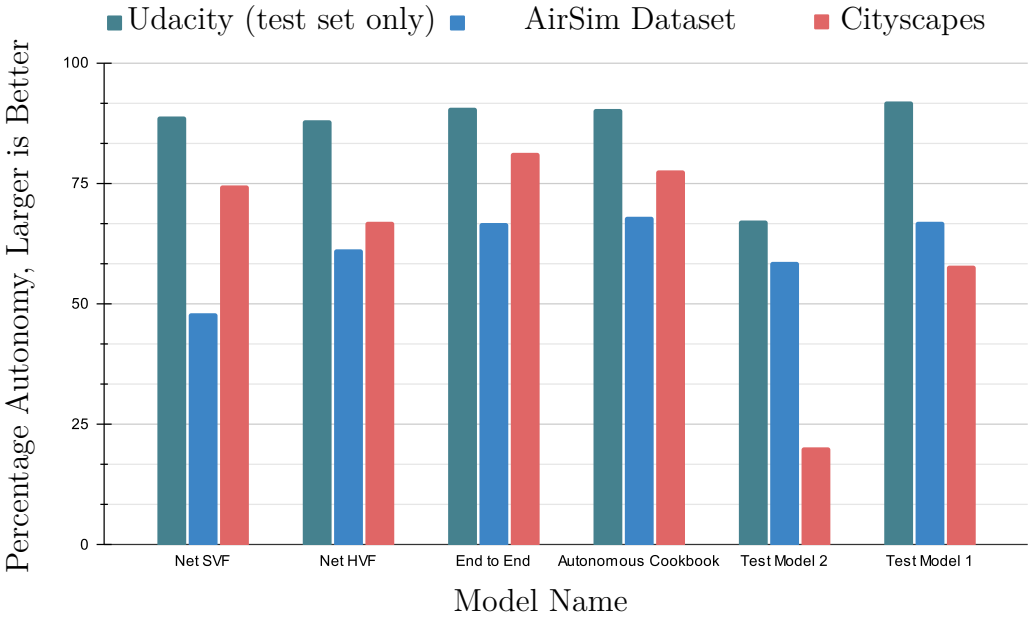


Figure 5.7: Comparison of Model Autonomy over Three Datasets

Figure 5.7 shows *TestModel2* performed the worst of any model architecture for the Udacity test set and The Cityscapes Dataset when considering autonomy percentage. *NetSVF* performed poorly with Microsoft's AirSim Tutorial Dataset, which does not have the same road-side objects as the real-world datasets.

Another interesting observation was that the three related architectures, *NetSVF*, *NetHVF*, and *End-to-End* have comparable trends in performance between each architecture. Of the three model architectures, *NetSVF* performed the worst, with *End-to-End* performing the best.

Test Autonomy Comparison Between Trained Models

The best model architectures were dependent on the dataset used in testing. For the Udacity test subset, *TestModel1* had the highest model autonomy percentage for predicting steering angle in Table 5.2. For Microsoft's AirSim Tutorial Dataset, the *Autonomous Cookbook* model architecture had the highest model autonomy. The *Autonomous Cookbook* model was the model designed for the synthetic dataset, Microsoft's AirSim Tutorial Dataset. For The Cityscapes Dataset, the *End-to-End* model architecture had the highest autonomy percentage.

The worst autonomy values per dataset tested were *TestModel2* for the Udacity test subset, *NetSVF* for Microsoft's AirSim Tutorial Dataset, and *TestModel2* for the Cityscapes Dataset.

Validation Loss Comparison Between Trained Models

The Udacity test set's results in Figure 5.7 and Table 5.2 indicated that *TestModel1* had the highest autonomy with 91.97%, followed by *End-to-End* with 90.73%, and then, *Autonomous Cookbook* with 90.35% autonomy.

When considering the Autonomous Cookbook Dataset, a different picture emerged. The *Autonomous Cookbook* model had the highest autonomy at 68.02%, while *TestModel1*'s performance dropped significantly to 67.06%. This suggested that all the models, and specifically *TestModel1* (which performed well on the test subset of the Udacity dataset), struggled to generalise against a synthetic dataset they had not been trained against.

For the Cityscapes Dataset, *End-to-End* had the highest MoA value at 81.22%. *TestModel2* performed poorly with an autonomy of 20.21%. The Cityscapes autonomy metrics were interesting because some models, like the *End-to-End* model, appeared to generalise well against real-world data, and other models performed considerably worse than on the test subset of the dataset they were trained on, namely the Udacity dataset.

Examining the validation loss presented in Table 5.3, shows that *TestModel1* and *TestModel2* have the largest MSE loss of any of the models when computed against the Udacity Dataset test subset.

The *Autonomous Cookbook* model had the lowest MSE for the same dataset. This somewhat contradicts the autonomy results in Figure 5.7 and Table 5.2. There *TestModel1* was the best performing model on the same dataset, however the *Autonomous Cookbook* model also performed well with a 90.35% autonomy.

Table 5.2: Comparison of Autonomy Results Per Trained Model

Model Name	Udacity Autonomy % (test set only)	AirSim Dataset Autonomy %	Cityscapes Autonomy %
Net SVF	88.88	48.09	74.44
Net HVF	88.02	61.19	66.89
End-to-End	90.73	66.67	81.22
Autonomous Cookbook	90.35	68.02	77.57
Test Model 2	67.19	58.63	20.21
Test Model 1	91.97	67.06	57.89

Table 5.3: Comparison of Loss (MSE) Results Per Trained Model for Best Epoch

Model Name	Udacity Loss (MSE)	AirSim Dataset Loss (MSE)	Cityscapes Loss (MSE)	Best Epoch
Net SVF	0.008742361666	0.83379107909	0.5197570956	44
Net HVF	0.005556635145	0.03896982281	0.0913531640	196
End-to-End	0.005638524889	0.06024779889	0.1627474733	573
Autonomous Cookbook	0.004747479719	0.02865050559	0.2155943607	1877
Test Model 2	0.036119960930	0.06620504407	2.5869187270	497
Test Model 1	0.022790845320	0.02279084532	0.1133918576	995

The validation loss results in Table 5.3 reflect the same overall result for the worst performing models, namely *TestModel2* performing the worst for the Udacity dataset, *NetSVF* performing the worst for the Autonomous Cookbook Dataset, and *TestModel2* again performing the worst for the Cityscapes Dataset. The best performing models when looking at the MSE in Table 5.3 were different to the autonomy results in Table 5.2. *TestModel1* and *TestModel2* had the largest MSE loss of any of the models when computed against the Udacity Dataset test subset. The best models according to the MSE test results was *TestModel1* for the Udacity dataset, *Autonomous Cookbook* for the Autonomous Cookbook Dataset, and *End-to-End* for the Cityscapes dataset.

This discrepancy in autonomy and MSE results imply that these metrics do not align, and possibly show that some models such as *TestModel1*, may have been over-trained leading to the best performance on the training dataset but poor performance on the other datasets.

5.4 GradSUM Analysis Results

5.4.1 Random GradSUM Results

NetSVF Random GradSUM Results

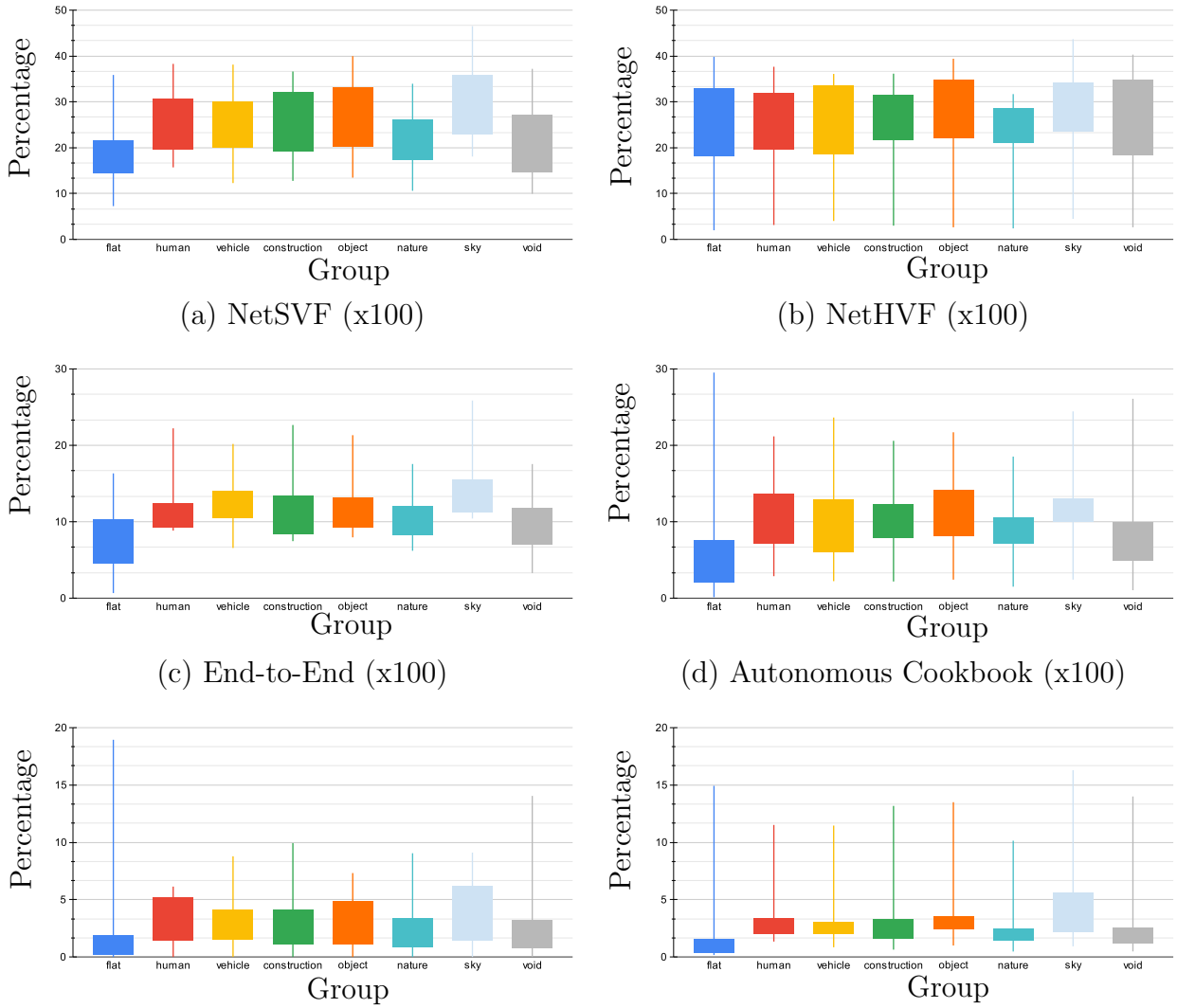
Figure 5.8 (a) shows that for 100 randomly initialised and untrained *NetSVF* models that the *flat*, *nature*, and *void* groups had a lower effect on the model's decisions than the rest of the groups. The *sky* group had a slightly higher effect. All the other groups appeared to be in the 10 to 40 percent range with similarly shaped box and whisker plots.

NetHVF Random GradSUM Results

Figure 5.8 (b) shows that *NetHVF* had a more uniform set of values when comparing the Cityscapes groups. There was a larger range of values, between 0 to 40 percent, and a similar shaped profile for each group.

End-To-End Random GradSUM Results

The random behaviour of the *End-to-End* models appeared to have a much lower set of percentages for each group in Figure 5.8 (c). The *sky* group appeared to have a larger impact than other groups. The *flat* group appeared to have a lower effect on the model's weight activation than the other groups.



(a) NetSVF (x100)

(b) NetHVF (x100)

(c) End-to-End (x100)

(d) Autonomous Cookbook (x100)

(e) TestModel1 (x100)

(f) TestModel2 (x100)

Figure 5.8: Comparison of randomly initialised versions of each model architecture

Autonomous Cookbook Random GradSUM Results

The *Autonomous Cookbook* models in Figure 5.8 (d) appeared to have a similar range of values as the *End-to-End* models, but the groups *human*, *vehicle*, and *object* had a slightly larger spread of possible percentage values. Interestingly, the *flat* group had a similar spread of data as the *End-to-End* models but with a much larger possible maximum outlier value.

TestModel1 Random GradSUM Results

TestModel1 in Figure 5.8 (e) appeared very similar to *TestModel2*. However, the variance of each group was much more significant than *TestModel2*.

TestModel2 Random GradSUM Results

The scale of percentage values was much smaller in Figure 5.8 (f) than in previous graphs, as it ranged from 0 to 20 percent. Most groups (averaged over each model) had more minor variances with larger maximum outlier values. The *sky* group again had a slightly more significant impact on the model decisions. The *flat* group was also the group which appeared to have the most negligible impact on the random models.

5.4.2 GradSUM Results across trained epochs per model

NetSVF GradSUM Results

Figure 5.9 (a) shows the generated GradSUM model profiles for *NetSVF* from its initial randomly initialised epoch 0 until the converged and trained model epoch 44. The model profiles changed over the training loop. Initially, the *flat* group had the most dominance on the model's weight activation. Over time as the model trained, it began to fixate on the *sky* group. As the training continued towards stopping, the *flat* group once again had the most effect on the activation of the model weights. The *human* group appeared to have the most negligible effect on the activation of the model's weight by the end of the training loop. Other notable groups such as *vehicle*, *construction*, and *object* appeared to have less effect than *flat* and *sky* groups but more than the *human* group of the Cityscapes Dataset labels.

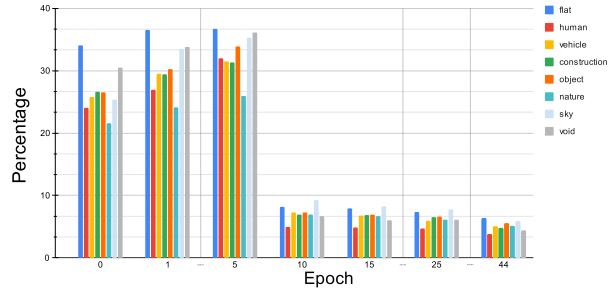
NetHVF GradSUM Results

The pattern of the trained *NetHVF*'s model profiles in Figure 5.9 (b) was substantially different to *NetSVF*. The first profile at epoch 0 shows that the *flat* group had the most impact on the activation of the model weights. The *sky* and *void* groups had the second and third most effect, respectively. The overall model profile pattern did not decrease in the range of the percentage values throughout the training loop, unlike the *NetSVF* model. By epoch 5 the model pattern for *NetHVF* was very different, with the *vehicle* and *sky* groups being the most prominent groups. The pattern stabilised by epoch 10 and became more refined by epoch 196. At epoch 196 the model appeared to have focused more on the *sky* group than the other groups. The *vehicle* and *object* groups were the

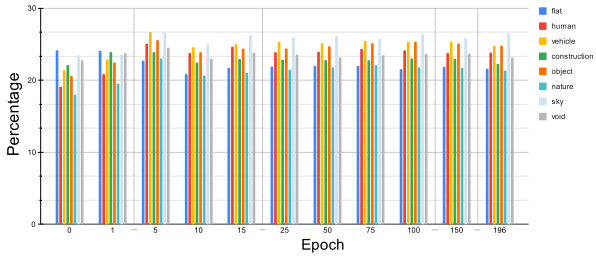
following two most prominent groups. The *flat* group was one of the groups with the lowest percentages of pixels.

End-To-End GradSUM Results

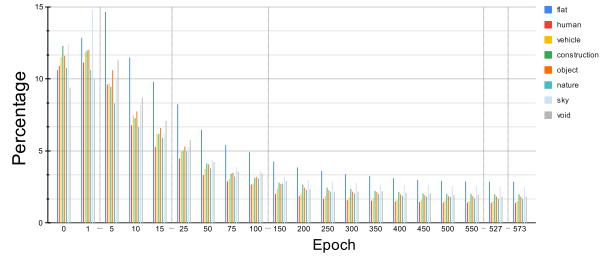
The *End-to-End* trained model began with a profile, as shown in Figure 5.9 (c), that shows that the model was mainly activated by the *sky* group, followed by the *construction* group. The *void* group had the least effect on the activation of the model, and the *flat* group had the second greatest effect on activation. The effect of the *flat* group considerably shifted from the first epoch to the last epoch and became the most prominent group by percentage of activated pixels (relative to other groups for the epoch). The *sky* group maintained a high relative effect on the model throughout training, however the *void* group did become more prominent, and its effect on the model was trained out as the training reached the end of the training loop. The *void* group still had a more significant impact on the model activations than the *human*, *vehicle*, and *nature* groups by the end of the training loop. The *construction*, and *object* groups had slightly more impact on model activations than the previously mentioned *human*, *vehicle*, and *nature* groups.



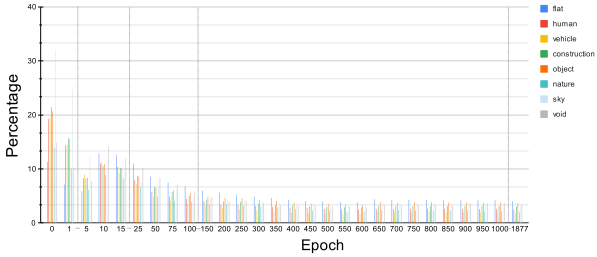
(a) NetSVF Profile



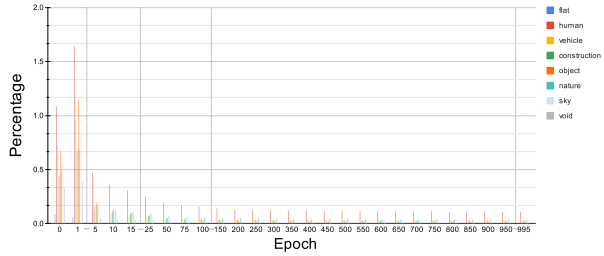
(b) NetHVF Profile



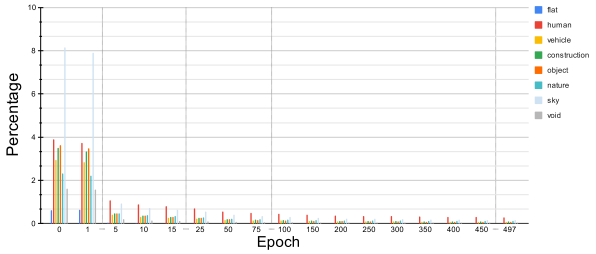
(c) End-to-End Profile



(d) Autonomous Cookbook Profile



(e) TestModel1 Profile



(f) TestModel2 Profile

Figure 5.9: Comparison of model profiles where the models are trained to their best epoch

Autonomous Cookbook GradSUM Results

Figure 5.9 (d) shows that the model profiles for the trained *Autonomous Cookbook* model remained very similar after the first 50 training epochs with only slight changes until the last epoch of 1877. The model started with a profile where the effect of the *sky* group of pixels in the ground truth dataset had a much greater effect on the model's activation than other groups. The *construction* and *object* groups were second and third. The *flat* group had the lowest impact on model activation. By the last epoch, where the model's loss stopped changing, the *flat* group had the most significant relative impact on the model activations relative to other groups for that epoch. The *object* group had the second highest impact, with the *void* group coming in third.

TestModel1 GradSUM Results

TestModel1's GradSUM results appear similar to *TestModel2*'s results below in Section 5.4.2. In Figure 5.9 (e), the first epoch began differently to *TestModel2*, with the *human* group impacting the model's activations most. However, by epoch 5 the *human* group had the most impact, with the rest being close to zero. Throughout the evaluation, the *flat* group had limited to no contribution to the model's activation of weights.

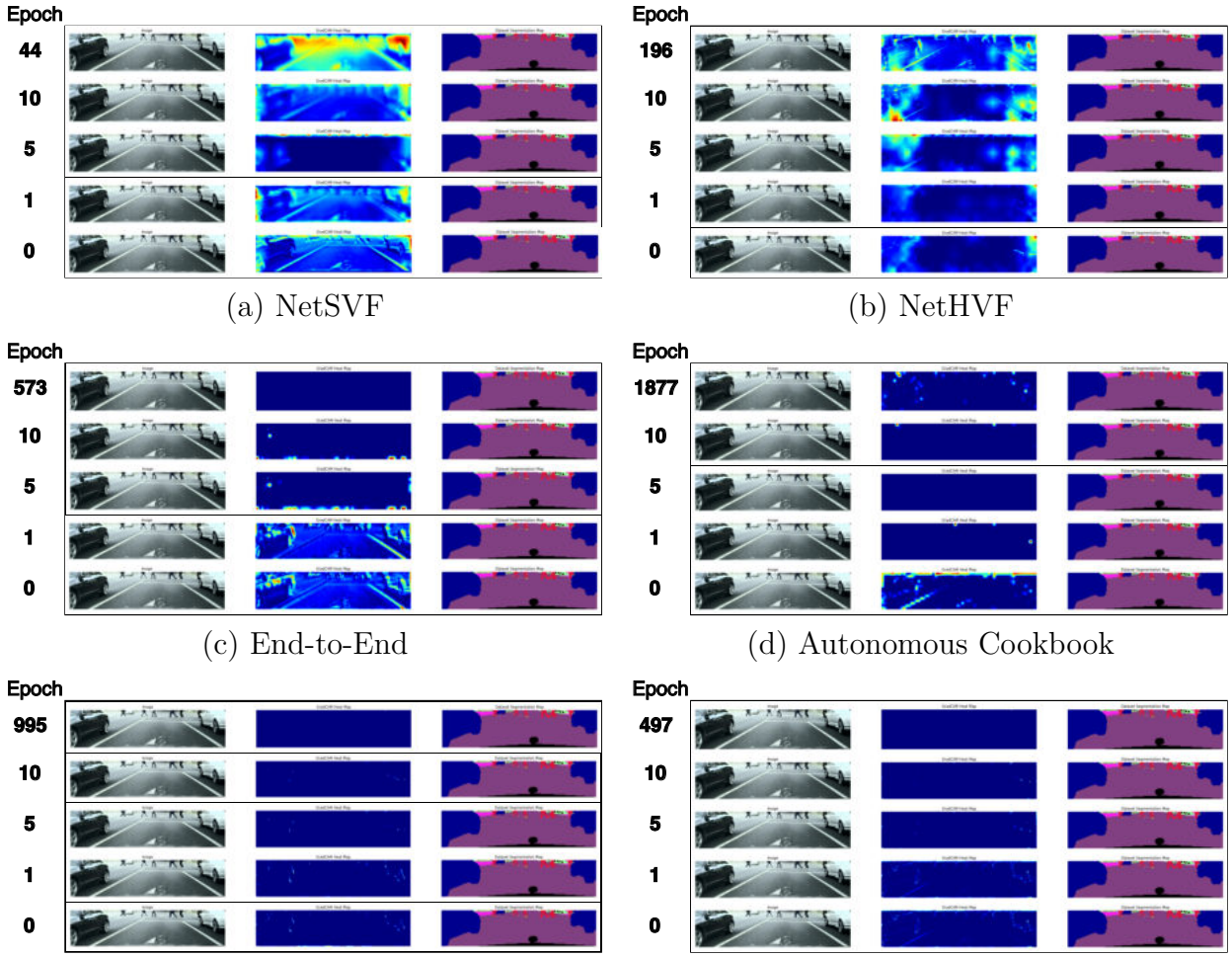
TestModel2 GradSUM Results

TestModel2's GradSUM results in Figure 5.9 (f) showed that the *sky* group had the most effect on the model's activation. The *flat* group had the most negligible effect. By epoch 5, the general shape of the model's profile per epoch appeared to have been found. By the last epoch (497), the *human* group of pixels appeared to have the most effect on model activation, with the *flat* group being near zero percent.

Drop in GradSUM percentage as the models train

When evaluating all the model training graphs there are two noticeable results related to the magnitude of each pixel group's model epoch percentages. For each model the percentage magnitude appears to drop as the models train. This could show that the models begin to focus and converge on the extracted image features found during the training loops. By focusing in on the core feature structure the amount of pixels that activate the model weights would also narrow. Another observation is that this reduction in GradSUM percentages does not occur for *NetHVF*, and occurs with different drops in magnitude for different model architectures. This potentially shows that this behaviour is model architecture dependant and possibly influenced by the drop-out layers turning on and off internal network structures in the model architectures.

Also the magnitudes of changes for the pixel percentages appear to be related to the complexities of the model architectures with the models with the largest complexities having the largest overall GradSUM percentages and the smallest models having the smallest magnitudes in their pixel percentages.



(a) NetSVF (b) NetHVF

(c) End-to-End (d) Autonomous Cookbook

(e) TestModel1 (f) TestModel2

Figure 5.10: Examples of a single GradCAM export over each model over a series of epochs, where the models have been trained on the Udacity dataset

5.4.3 Example GradCAM outputs for each model

NetSVF GradCAM Sample Comparison

The *NetSVF* sample output per epoch in Figure 5.10 (a) shows that the model generated a reasonably sharp and observable gradient map throughout its epochs, with the last epoch having multiple places with a higher gradient value. The scene shows that these areas were ahead of the vehicle where people were crossing the street.

NetHVF GradCAM Sample Comparison

NetHVF's sample, as shown in Figure 5.10 (b), shows a gradient map which goes from being blurry and hard to discern to showing more of the outline of the road and cars in the scene.

End-to-End GradCAM Sample Comparison

Figure 5.10 (c) shows that the *End-to-End* model's sample GradCAM output went from being reasonably sharp and outlining the road and vehicles in early epochs to being a nearly solid colour gradient map as there were not many gradients and their values were insignificant. This implies that for this sample no parts of the image had a large impact on the model's activation. However Section 5.4.2 and Figure 5.9 (c) gives a more realistic look at which components of the *fine* segmentation groups in the Cityscapes dataset had an impact on the model activation of the *End-to-End* model. The analysis of a single or very few sample GradCAM outputs does not necessarily imply a general behaviour of a model, but the behaviour of a model for a specific sample, which could be different to the model's average behaviour.

Autonomous Cookbook GradCAM Sample Comparison

The sample figures of the *Autonomous Cookbook* model, as shown in Figure 5.10 (d), went from a low resolution and pixelated gradient map, which appeared to show the road edges on the side of the frame to a gradient map with specific hot spots. This could show that the *Autonomous Cookbook* model does focus on specific parts of the sample frame but does not have a very high resolution feature extraction of the scene.

TestModel1 GradCAM Sample Comparison

TestModel1's GradCAM sample appeared to be an even more sparse version of *TestModel2*'s GradCAM sample. There was no helpful information from the sample map in Figure 5.10 (e).

TestModel2 GradCAM Sample Comparison

TestModel2's GradCAM sample in Figure 5.10 (f) had a fine outline of the edges of the scene in the early epochs, and in the later epochs, the gradient map was similar to the *End-to-End* model in Figure 5.10 (c), where the gradient map had no visible gradients.

5.4.4 Time taken to Generate Results

Figure 5.11 and Table 5.4 shows that the most costly model architecture to train and evaluate was *NetHVF*. The second most costly architecture was *NetSVF*, and the third was *End-to-End*. These were the three most complex model architectures evaluated. *NetSVF* had a larger architecture with a greater field of view than *NetHVF*. *Autonomous Cookbook* had the most trained model epochs but took less time to train, evaluate and analyse than the *End-to-End* model.

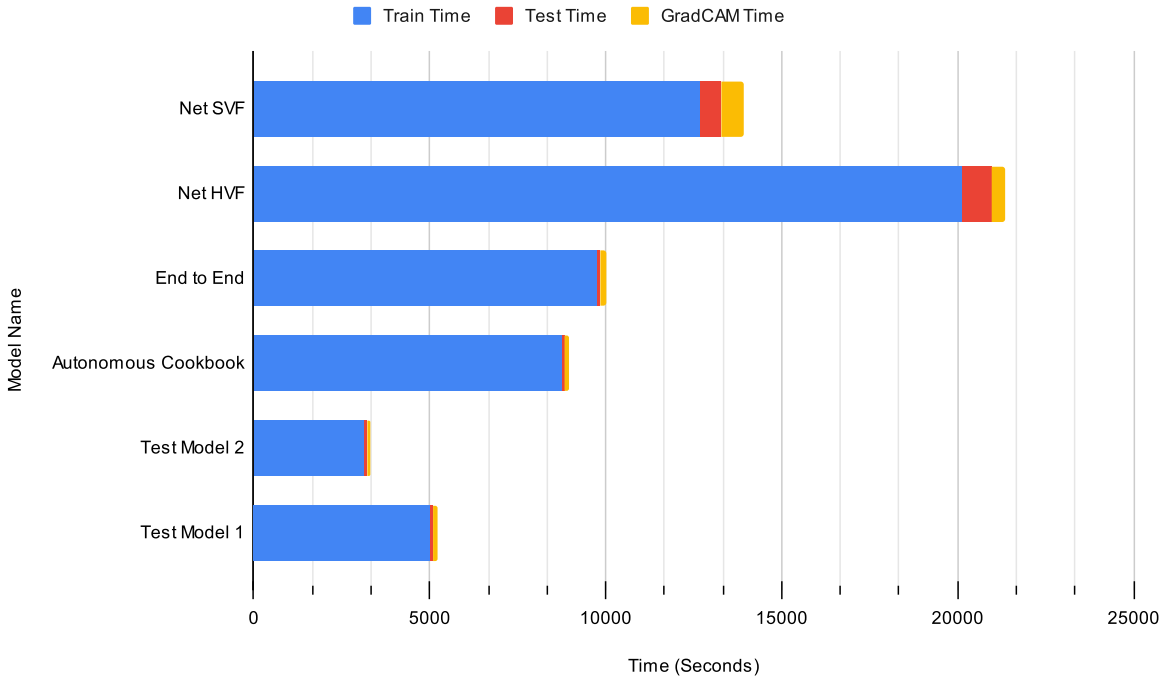


Figure 5.11: Comparison of Time Taken by Each Model Architecture

Table 5.4: Time that was taken to train, evaluate (test) and analyse model architectures

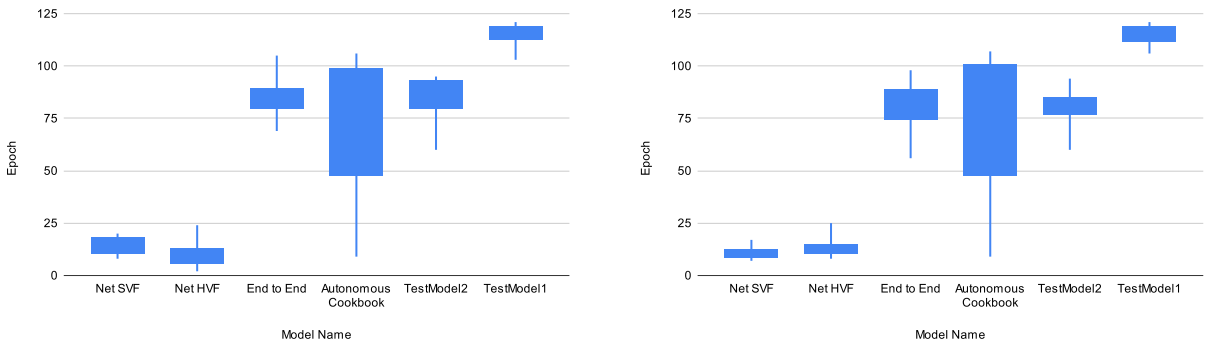
Model Name	Train Time (S)	Test Time (S)	GradSUM Time (S)	Total Time (S)
NetSVF	12677.88	622.85	613.67	13914.40
NetHVF	20130.22	845.25	388.48	21363.95
End-to-End	9761.73	71.67	174.94	10008.34
Autonomous Cookbook	8733.41	85.11	128.30	8946.82
TestModel2	3125.18	87.38	110.91	3323.47
TestModel1	5010.65	102.93	127.86	5241.44

5.5 Additional Average Experimental Runs

Further analysis of model architecture is needed. Single model variants may behave in a specific way that could be different to behaviour in other variants with the same model architecture.

In order to determine a more statistical view of the behaviours of these model architectures, each one was run through the same experimental framework 10 times. This includes pre-training 10 randomly initialised models for 10 epochs and selecting the best one.

Figure 5.12 shows that the average spread of the best models for MSE and MoA are very closely aligned with some edge case differences. Also there appears to be a correlation between the size and complexity of a model architecture and how many epochs are needed to reach an early stopping condition. This shows that the more complex model



(a) Best Autonomy Epoch Per Model Type (b) Best MSE Epoch Per Model Type

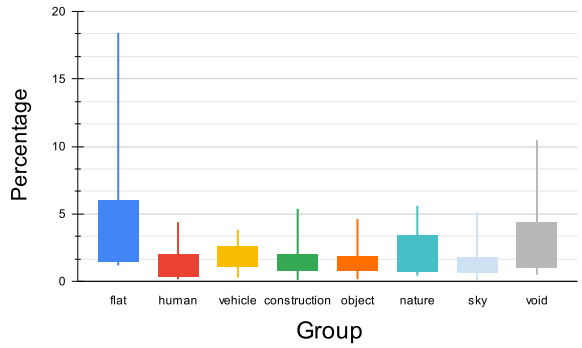
Figure 5.12: Best Selected Models Over All 10 Runs and Each Architecture

architectures tend to converge faster than the smaller and simpler ones. The *Autonomous Cookbook* model architecture appears to have the greatest spread of values.

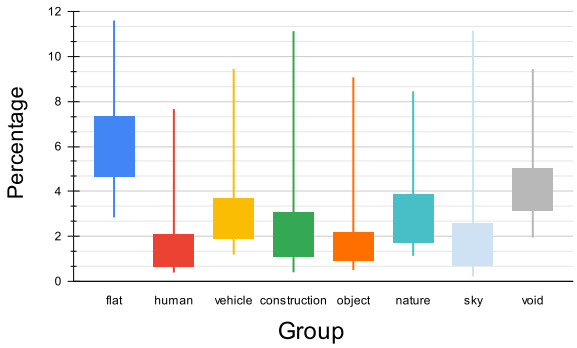
Figure 5.13 shows the combined model profiles for all 10 runs over each model architecture evaluated.

For the *NetSVF*, *End-to-End*, and *NetHVF* architectures as shown in Figure 5.13 (a), (b), and (c), the *flat* group is shown to have the largest average relevance versus the other groups. This shows that these two architectures (which are expected to be the best for predicting steering angle in the given setup) is predisposed to focusing on one of the most important groups of pixels, the *flat* group. This group is the one which includes the road surface pixels. The *End-to-End* architecture also appears to favour the *construction* and *object* groups after the *flat* group. These are also semantically relevant pixel groups in the Cityscapes dataset for self-driving vehicles.

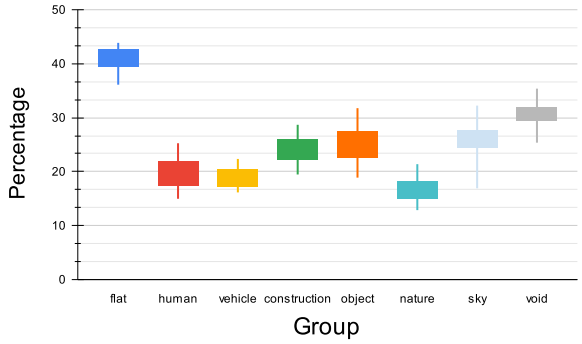
The last three model architectures, *Autonomous Cookbook*, *TestModel2*, and *TestModel1* all show a fairly large range of values as indicated in Figure 5.13 (d), (e), and (f), with outliers close to zero for all groups. This is a departure from the previous model architectures where at least the *flat* group of pixels has GradSUM values that were not close to zero. *Autonomous Cookbook* also has a much smaller range of GradSUM values close to zero compared to the other two architectures. *TestModel2*, and *TestModel1* appear to have the bulk of their *flat* group GradSUM pixel values as the most relevant pixel group, however it is possible to randomly initialise these architectures in such a way as to have near zero pixel values for the *flat* group.



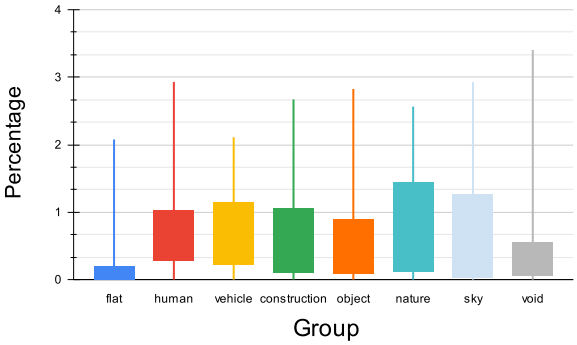
(a) NetSVF



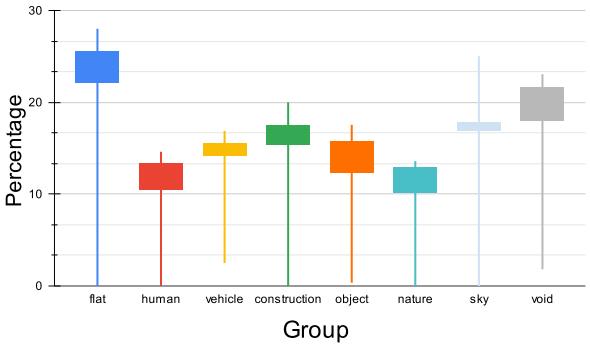
(b) NetHVF



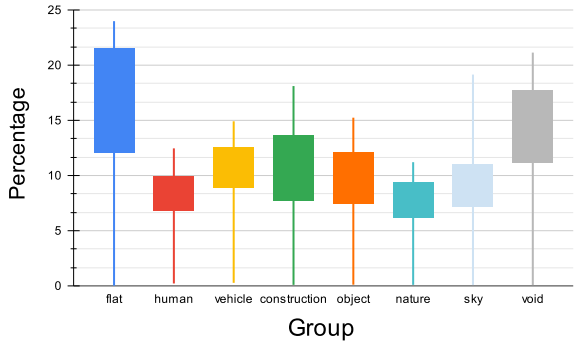
(c) End-to-End



(d) Autonomous Cookbook



(e) TestModel1



(f) TestModel2

Figure 5.13: GradSUM box-and-whisker profile plots for each model architecture

5.6 Conclusion

Larger model architectures (such as *NetSVF*, *NetHVF*, and *Autonomous Cookbook*) appeared to trigger early stopping in their respective training loops at a faster rate than simpler model architectures. The more complex model architectures also needed more resources to train and evaluate. The best-performing model based on its autonomy performance evaluated on the test dataset was *TestModel1* for the Udacity testing subset, *Autonomous Cookbook* for Microsoft's AirSim Tutorial Dataset and the *End-to-End* model architecture for the Cityscapes Dataset. The two most complex model architectures did not attain the best autonomy percentages for any test datasets. Autonomy did not appear to be affected by the randomness of how the models were initialised as all architectures before training and in the random experiments scored autonomy very close to zero percent.

Training had a direct effect on the model profiles generated by the GradSUM scheme. Different architectures appeared to have different possible GradSUM group percentages when initialised over a large range of values. However, the larger model architectures tended to have larger initial GradSUM values, whilst the smaller models appeared to have had much smaller values. As training proceeded, most models (except *NetHVF*) dramatically decreased the total cumulative GradSUM values per epoch.

Random initialisation also appears to directly influence the outcome of training model architectures with the best and worst outcomes being tied to a good or bad initialisation. This is most evident with the *TestModel1* and *TestModel2* architectures.

For specific model variants the *NetSVF*, *End-to-End*, and *Autonomous Cookbook* models all appeared to fixate on the *flat* group of pixels once trained. *NetHVF* seemed to fixate on the *sky* group, whilst *TestModel1*, and *TestModel2* fixated on the *human* group. For *NetSVF*, *End-to-End*, *Autonomous Cookbook*, and *TestModel2* the *sky* group greatly impacted model decisions in early trained epochs but that observed behaviour shifted as those models trained per epoch.

MSE and MoA metrics did not align for the best performing models for each metric and test dataset. They did align for the worst performing model per dataset. This implies that autonomy performance (based on the task of steering angle prediction) at the upper end of performance is not necessarily indicated by the error loss differences of the best performing models. This means that there are factors in the input data beyond the direct numerical accuracy of the predictions made, when comparing well performing trained model architectures. However when evaluating the worst performing model architectures the two metrics did align - which shows that models with bad training, or which have been over-trained (such as *TestModel1*) will have a much larger MSE and therefore a lower (worse) autonomy percentage and will therefore perform predictably worse in the task.

Different architectures also showed different abilities to generalise over different datasets both simulated and from real-world data capture. For a model to be used in the real world, it needs to be able to continue to perform at a high level of performance even on real world data it has never seen before. Testing across multiple datasets (two of which

the model has never seen before), showed that model architecture does play an important role in being able to generalise task performance over different datasets.

Chapter 6

Discussion

6.1 Introduction

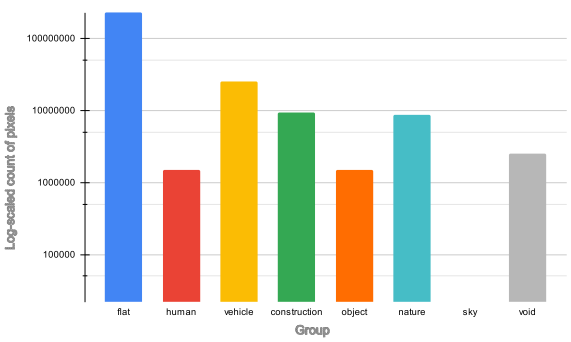
It is evident that using a single metric to evaluate model behaviours is limited in terms of what conclusions can be made for that model. This study aimed to explore the potential for a general *End-to-End* AI model architecture to learn meaningful road features from sparse training signals for the task of steering angle prediction. In contrast to the approach taken by Bojarski *et al.* [2016b] who solely evaluated the specific model architecture called *End-to-End*, we evaluated multiple model architectures using multiple analysis techniques.

Our findings revealed that different architectures displayed varying behaviours during the training process and even once trained. Specific model architectures demonstrated the ability to learn to bias more to semantically meaningful road features that are relevant to the task, whilst other architectures tended to fixate on less useful components of the input data (using a ground truth dataset for evaluation). This would indicate that these models were learning spurious correlations which produced worse model performance. The semantic information used is from *The Cityscapes Dataset* and has data relevant for self-driving vehicles.

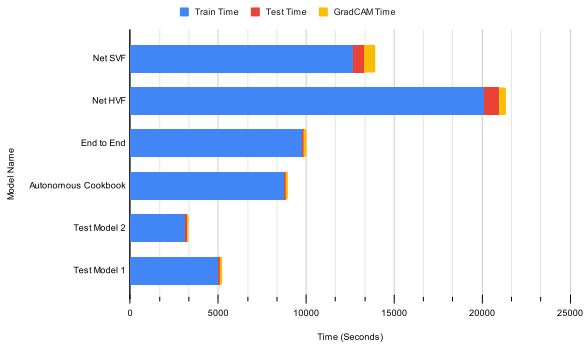
It is important to study the behaviour of AI models using multiple metrics and taking a holistic approach to that analysis. This helps in identifying useful and possibly harmful emergent behaviours in trained or training models. By identifying the bounds of a model's behaviour for a given task can in turn help understand these models better, in order to produce safer, and more effective AI systems.

6.2 Overall Performance of the End-To-End Systems Based on Results

Initial analysis of the ground truth dataset, when using the Canny Edge study shown in Section 4.13 indicated that the edges of the pixels of each group in the Cityscapes Dataset have similar profiles, shapes and values relative to each Cityscape group. When



(a) Cityscapes Pixel Count (Section 4.4)



(b) Time Taken (Section (5.4.4)

Figure 6.1: Figures Related to the Overall Performance of End-To-End Systems

analysing each group of Cityscapes pixel counts in Figure 6.1 (a), it is evident that some groups, such as the *flat* group, have far more pixels than other groups. This is a distinct result from each group's edge map percentages which were very similar. This helps to eliminate edges as being a bias towards a specific segmentation group in the ground truth dataset.

When analysing the randomly initialised weight GradSUM results for the model architectures in Section 5.4.1, most models exhibited similar activation patterns for each group for that particular model. Some groups deviated in many of the models, such as *NetSVF* and *End-to-End*, where the *sky* group appeared to have a slightly higher percentage value and the *flat* group had a slightly lower percentage value. This behaviour appeared to change as the models were trained. This validates that training of these model architectures had a direct impact on which segmentation groups had an effect on a specific model's activation during and after the training loop.

More general observations around the training of the different model architectures could be made. It appeared that the computational and time cost to train models depended on two factors. The first was the size of the model architecture, which related to how extensive the matrix calculations were due to the size and depth of the models. A surprising result shown in Figure 6.1 (b), was that *NetSVF* was faster to train and evaluate than *NetHVF*, a simpler model architecture. The early stopping scheme made the training and validation process end faster. The larger architecture appeared to have its validation loss stop changing in fewer epochs and produced excellent MSE loss results and a relatively high (but not the best) autonomy percentage. This implies that predicting the steering angle from image inputs, had a very strong correlation between the input data (input frame) and the steering angle. This could mean that other tasks where the sparse input data is not as strongly correlated may not perform as well with trained models of any architecture, as this task did.

6.3 Evaluation of Performance Per Model

The random tests performed on 100 randomly initialised versions of each model architecture showed that both the test loss and autonomy results were dependent on the training of the models. This gives confidence that the test metrics for each model were directly related to training the models and not due to the architecture alone.

The models were all trained on the Udacity Dataset. However, the GradSUM analysis was done with the Cityscapes Dataset since it contains a good ground truth set of *fine* segmentation maps. The ability for the models to transfer learning between datasets was not considered, but their performance between datasets was evaluated.

When looking at the random test GradSUM results in Figure 5.8, the group *flat* usually appeared to have the lowest percentage values compared to the other groups per model. Only *NetHVF* in Figure 5.8 (b) showed the *flat* group having a closer value to the other groups in the GradSUM results. This is interesting because Figure 4.4 in the analysis of the Cityscapes Dataset shows that the *flat* group contains the most pixels of any of the groups. Another observation was that the resultant profile from the Canny edge analysis for the *flat* group in Figure 4.19 indicates that this group had the highest (if only marginally higher) percentage of any group. The group contains all the road information, such as the road surface and side-walk as shown in Table 4.5. This could be considered a meaningful group for predicting steering angle because the angle of the road would be encoded into this component of the input image being fed into a model.

The *sky* group in the same GradSUM results from Figure 5.8 had the highest percentage of any of the groups. The *Autonomous Cookbook* results in Figure 5.8 (d) showed that the *sky* group had the third largest percentage value and was skewed slightly higher than most groups. The *sky* group only contained pixels of the sky in each frame. Arguably this data is not meaningful to predicting steering angle when driving a vehicle.

When using GradSUM to analyse models over their training epochs, when their metrics (such as MSE) improve, the model profile percentages within each profile, also shifts. This implied that training the models caused the models being trained to observe different areas of the inputted data. This behaviour change needed to be evaluated per model.

6.3.1 NetSVF Behaviour

The *NetSVF* model trained until epoch 44, as seen in Figure 5.1. Table 5.1 shows that it had the highest training loss of all the models, but its validation loss was less than *TestModel2*. Its autonomy was not the lowest for the Udacity and Cityscapes datasets but was for the Microsoft AirSim Tutorial Dataset. It scored an autonomy of 88.88% on the Udacity dataset and 74.44% on the Cityscapes dataset. This would mean that the AI performed better than random model variants on these datasets and must have learned to extract information useful for predicting the steering angle from the input data. However this model shows to be unable to generalise to synthetically generated data but can perform with some degree of performance against real world datasets.

The model profile shown in Figure 5.9 (a), shows that the model focused more on the *flat* group of pixels than the other groups. The second highest group was the *sky* group.

The lowest group was the *human* group. This means that the model focused on a group of pixels that can be considered meaningful. However, it also seems to have activations on groups that are not meaningful i.e. the *sky* group of pixels.

Another note in Figure 5.9 (a) is how the range of percentages for each group per epoch reduced as the model trained. This could be explained by the model focusing on parts extracted from the input data as the model trained. Model weights were adjusted to ignore some parts of the input and had higher activations on other parts.

Section 5.5 shows that the *NetSVF* architecture on average maintains its focus on the *flat* group of pixels when looking at its GradSUM results in Figure 5.13 (a). Figure 5.12 also shows that the model architecture on average converges to an epoch with either the best MSE or best MoA, earlier than most other architectures evaluated.

These results imply that the model learned meaningful road features from the input data.

6.3.2 NetHVF Behaviour

Figure 5.1 shows that the *NetHVF* model trained for 196 epochs. Its loss and MoA were not the lowest or highest of the trained models. It scored 88.02% autonomy for the Udacity dataset and 66.89% autonomy for The Cityscapes Dataset. This implies that the model learned to extract enough data to predict steering angle better than a random model can but was more error-prone than the better-performing models. This implies that its ability to generalise on real world data is not as great as other models evaluated such as the *End-To-End* model architecture.

Figure 5.9 (b) shows that by the end of training the group with the most significant percentage was the *sky* group. This is not a meaningful feature in the dataset for predicting steering angle. The following two groups by percentage were the *vehicle* group and the *object* group, which are more meaningful features from the input data. Table 5.1 shows that the *vehicle* group contained the pixels for any kind of vehicle in the ground truth dataset. This includes the *car*, *truck*, *bus* and many more labels. This information is potentially useful when driving behind or near these other vehicles and predicting what the steering angle should be. The *object* group contained *pole*, *traffic sign*, and *traffic light* labels. This information is meaningful for driving decisions but may not directly correlate to a specific vehicle steering angle. Another observation is that the *flat* group had the second smallest value in the last epoch model profile in Figure 5.9 (b).

Considering that the model variant focused more on groups with less relevance to predicting steering angle, we can say that this model variant did not focus on meaningful segmentation groups in the input data (from the ground truth dataset) over other information included in the input. This implies that the model did not learn meaningful road features from the input data.

However section 5.5 shows that the *NetHVF* architecture on average maintains it focus on the *flat* group of pixels when looking at its GradSUM results in Figure 5.13 (b). Figure 5.12 also shows that the model architecture on average converges to an epoch with either the best MSE or best MoA, earlier than most other architectures evaluated. This is a similar result to *NetSVF* meaning that this variant was probably influence by a bad set of

initial weight variables. GradSUM is able to help identify this edge case where a specific model initialisation produces worse results once trained than on average.

6.3.3 End-to-End Behaviour

The *End-to-End* model was trained in 573 epochs. It had a test autonomy of 90.73% on the Udacity test set and 81.22% on The Cityscapes Dataset. It had the highest result for The Cityscapes Dataset. This synthetic result was expected to translate into a good model profile of its behaviour using GradSUM.

Figure 5.9 (c) shows that by the end of training, the *End-to-End* model's largest group of pixels to effect activation relative to the other groups was the *flat* group. The *sky* group was second, and the *construction* group was third. Similar to the *NetSVF* model, the *End-to-End* model also had its range of percentages for each group per epoch reduced over each training epoch.

Evaluating the *End-to-End* architecture's average results shown in Figure 5.13 (c) reinforces the fact that this architecture is able to learn semantically meaningful information for a self-driving vehicle. It is shown to mainly focus on the *flat* pixel group but also the *object* pixel group, which are both semantically important groups for driving. The spread of convergence for this architecture is also fairly narrow and not skewed as seen in Figure 5.12.

The same conclusion made for the *NetSVF* model's behaviour can also be made about this model learning meaningful features from the input data.

6.3.4 Autonomous Cookbook Behaviour

The *Autonomous Cookbook* model finished training by 1877 epochs. The model produced an autonomy of 90.35% for the Udacity test set and 77.57% for The Cityscapes Dataset. This result implies that the model could predict steering angle from a sparse input signal from a forward-facing image better than a random result.

In Figure 5.9 (d), the model profile for the last epoch shows that the group with the highest percentage was the *flat* group. The second highest group was the *object* group. The *flat* group is a more meaningful group than the *object* group. However, the *object* group contains meaningful information for higher-level tasks needed for driving, such as knowing when to slow down or stop.

This implies that the model variant was able to learn meaningful road features from the input data.

However when analysing the spread of data produced when looking at the average behaviour of this model architecture in Section 5.5 a different story can be seen. Here this architecture has the most spread and is skewed when looking at the epochs it will converge at as shown in Figure 5.12. This shows that it is very sensitive to model weight initialisation. This implies that the model is sensitive to noise which may make it more unreliable versus the largest the architectures evaluated.

Figure 5.13 (d) shows that the *Autonomous Cookbook* model architecture has *gradsum* values per pixel group close to zero and with a much smaller magnitude (on average) than the other evaluated model architectures. Also the *flat* group of pixels appear skewed to have an even smaller value on average than the other pixel groups for this architecture.

These findings in the average test performed in section 5.5 shows that this model architecture is probably not as desirable to use to predict steering angle from input images than the other evaluated model architectures.

6.3.5 TestModel1 Behaviour

The *TestModel1* model variant in Figure 5.9 (e) had a near zero value for its model profiles. It also had a slightly higher value for the *human* group of pixels than the others. It did produce a 91.97% autonomy value for the Udacity test set and 57.89% value for The Cityscapes Dataset. Even though the synthetic metrics appeared suitable for the model (it was the top-performing model on the Udacity test set in autonomy), the low activation values imply that this model variant is unlikely to have learned meaningful features from the input data.

When considering the average results of the *TestModel1* model architecture as shown in Section 5.5, it can be seen that this model is extremely sensitive to model weight initialisation as it has extreme edge case GradSUM pixel group values close to zero, but also on average each group appears to have a significant value as seen in Figure 5.13 (e). The *flat* pixel group is also on average of greater relevance to model weight activation than other pixel groups. Figure 5.12 shows that this architecture appears to take the longest of any architecture to converge but that convergence remains predictable and consistent between 100 and 125 epochs.

This model architecture has the potential to learn meaningful semantic road information, however it is sensitive to how it is initialised and to noise in the system. This makes it less ideal than other evaluated models.

6.3.6 TestModel2 Behaviour

The *TestModel2* variant had a much higher validation loss than the other model variants, as seen in Figure 5.4. It took 497 epochs for the model to finish training. As seen in Figure 5.9 (f), by epoch 497 the range of percentage values was very small - less than 1%. The highest percentage group was the *human* group. The *human* group contains pixels of *person* and *rider* labels. This group and the fact that there was a minimal value for the percentages means that it is unlikely that this model variant learned meaningful features from the input dataset.

The *TestModel2* architecture has similar findings to *TestModel1*, where in Section 5.5, and Figure 5.13 the architecture shows that it is sensitive to model weight initialisation and noise. In Figure 5.12 the skewness of best epochs is greater than *TestModel1* meaning that convergence is not as predictable. The *flat* pixel group is on average the most relevant pixel group as well. However this architecture also suffers from extreme outliers which would make some variants have inappropriate learned biases.

6.3.7 *TestModel1* and *TestModel2* Fixation on *human* Pixel Groups

Table A.1 shows that the *human* group of pixels is the second smallest group found in the Cityscapes dataset whilst table A.2 shows that the *human* group of pixels appears in most of the image samples evaluated. Both *TestModel1* and *TestModel2* variants have very small values for their model percentages. It is possible that these models being less complex than the other model architectures evaluated, may lead to the models fixating on features which appear in most samples during training.

When looking at the average results for both model architectures in Figure 5.13, there does not appear to be any added bias on the *human* pixel group which implies these model variants have had this bias induced either during initial selection of random weights or during training.

6.3.8 The Difficulty of Predicting Steering Angle from Sparse Data

The results above imply that whilst some of the models (3 out of 6) could learn meaningful road features, the others could not. This result implies that the architecture of a model is an major component in the ability of a model to learn meaningful road features from the input data.

Another implication is that steering angle prediction is highly correlated to the images from a forward-facing camera of the road from a vehicle being driven. Other tasks may not have this correlation between the input images and the task. Without such a correlation any GradCAM analysis may produce low quality results. Data with a lot of noise or of low quality may also produce spurious correlations and lead to heat-maps which do not reflect reality.

Other possible tasks where this kind of analysis would be helpful include multi-task problems such as steering-angle and speed prediction or higher abstracted tasks such as parking the vehicle in a parking lot. Other problems and tasks may require more complex testing datasets, and possibly modifications to the analysis scheme to take into account larger, and more dimensionally complex input data. This complexity would translate into increased expense in training models to parse this data and also a more complex analysis to explain these models.

6.4 Cost-Benefit of using the GradSUM system

The amount of time GradCAM took to compute depended on which model architecture was analysed. The GradSUM scheme took the added time cost of every GradCAM generated for each sample in the ground truth dataset. Counting up the pixels of each group also added some computational cost. Table 5.4 shows that the cost of the GradSUM results was also architecture dependant.

For *NetSVF* and *NetHVF*, the training and evaluation (test) stages took longer than the GradSUM stage. The GradSUM scheme took more time to generate for all the other models than the evaluation step. There is a trade-off between the size of the ground truth dataset and how long the analysis will take to compute for a given model architecture.

This might be a consideration if the analysis is required in real-time or for a more advanced training loop where added cost could be undesirable.

The GradSUM scheme can be computationally expensive due to the available ground truth data samples available. If too few samples are used, the results of the analysis could be skewed. This means there is a balance between selecting an adequate and large enough dataset for the analysis and how much it will cost to analyse a model.

The scheme could also be used to initially select randomised models and other decisions in the training pipeline. This could help speed up the overall experimental setup by starting with a preferential initial selection and excluding bad initialisations.

Other attribution methods which also generate activation maps could be used with GradSUM. More analysis would need to be performed, but the primary cost of the scheme is generating the activation maps against a model over the whole ground truth dataset.

6.5 Future Work

6.5.1 Application of other attribution methods

Many other attribution methods produce activation maps that could be used with GradSUM. The sanity checks defined by Adebayo *et al.* [2018] should be used on any attribution method being considered to ensure that the result is an attribution map correlated to the input data and shows the activations of the model weights.

The GradCAM has a modified GradCAM method called *FullGrad* which computes the GradCAM map for every layer in the model where the gradient can be computed and then sums the resultant maps together. This can produce higher-resolution maps but sanity checks also need to be run on this method.

An analysis of potential replacement methods for GradCAM should be done. This study will look at the improvement to the resolution of the resultant maps and the performance cost of each method. The method selected needs to produce valid activation maps and have a balance between computational cost and output resolution.

6.5.2 Application of attention based models

There are newer kinds of model architectures which use attention maps to help encode representations of features from the input data in-order to make predictions from those representations. Devlin *et al.* [2018] introduces a model called *BERT* (Bidirectional Encoder Representations from Transformers) which uses attention maps in a transformer architecture to perform many different NLP tasks. Dosovitskiy *et al.* [2020] further expands these kinds of transformer architecture to produce a ViT (Vision Transformer) model which is designed to replace CNN model architectures in order to perform computer vision tasks. These models are shown to attain excellent results when compared to the more traditional CNN model architectures. They are considered to be the state-of-the-art in terms of model architectures used for solving computer vision complex tasks.

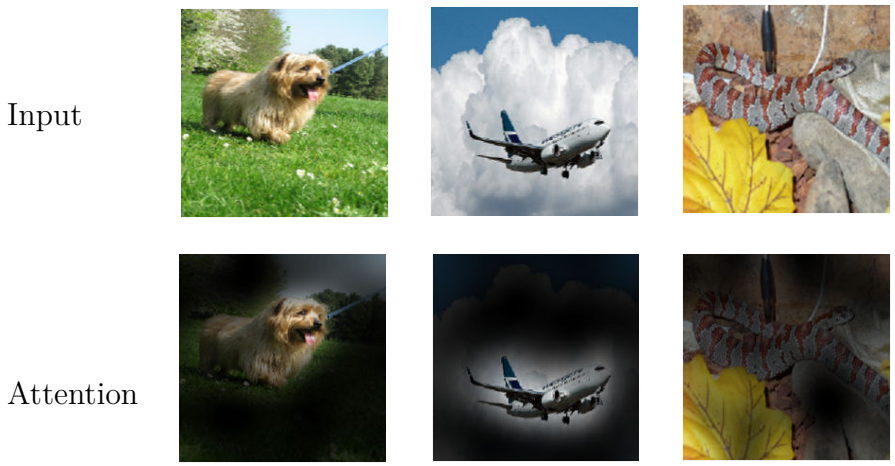


Figure 6.2: Example attention results of input images using a vision transformer [Dosovitskiy *et al.* 2020]

These ViT models make use of attention layers which are an internal representation of the most important parts of the given input. These models convert a 2-dimensional image into a 1-dimensional sequence of patches which can be considered to be a token of data similar to how NLP data would be processed [Li *et al.* 2023].

The computed attention weights inside a ViT model can be considered analogous to a GradCAM heat-map in the sense that it is an ordered representation of patches with corresponding magnitudes taken from the input. The importance or weight of each part of the image is tracked. Dosovitskiy *et al.* [2020] generates visualisations of the input data with the attention map applied as a filter. These visualisations are strikingly similar to a GradCAM heat-map applied to given input data as seen in Figure 6.2.

Since these ViT model architectures produce attention weights internally as part of their computation, they are available for analysis. Future work in this area includes using these attention maps instead of GradCAM maps on trained ViT architectures against a ground truth dataset. This would also allow GradSUM to be applied using the attention maps and not GradCAM heat-maps. The same models can also have GradCAM applied to them [Gildenblat 2022]. A comparison between the results of using attention with GradSUM and GradCAM with GradSUM can then be made. The ViT model architectures can then be compared in a regression task (such as steering angle prediction using forward facing images) against more traditional CNN model architectures using the same methodology outlined in this research.

A possible benefit of using attention weights over GradCAM is that the computational cost would be considerably less. The attention weights have to be computed when evaluating the model with a given input dataset. This means that the added step of the GradCAM algorithm is omitted. However an analysis of the viability of this technique is required.

6.5.3 Improving the ground truth dataset

The ground truth dataset from The Cityscapes Dataset only had 5000 *fine* samples. An improved ground truth dataset should be evaluated. This could be generated from the test division of the training dataset using AI tools so that the models being evaluated do not have to transfer over to an unseen dataset.

Another approach would be to combine existing datasets and create a ground truth with a fair spread of information both in the labels present and the kind of scene (turning vs going straight or other relevant information for the task being learned).

A third approach to be evaluated is to use an available simulator and create benchmarks for more straightforward tasks. This would allow repeated evaluation of models with metrics that can be compared. It could also allow for the ability to generate a dataset with all the relevant information around segmentation, vehicle information and forward-facing images.

6.5.4 Improve the selection of models and their architectures

More model architectures should be evaluated. Their architectures should be constructed using GradSUM analysis to help identify which architectures focus on meaningful data over other architectures, which can then be selected and trained.

GradSUM analysis can also be used in the random initialisation of models to select the model which prioritises on the best features automatically. A study of how this changes the overall performance of the models should be conducted.

6.5.5 Evaluate different regression tasks and the performance of GradSUM

Other single tasks, such as controlling the vehicle's speed should be evaluated in the future, and the modified models should be evaluated in the same way as the current models. Predicting speed may require temporal data to be included in the input, such as processing multiple sequential frames. Another approach is also improving the model architectures to process temporal data better.

Composite tasks can also be evaluated. An example of such a task would be predicting the steering angle and controlling the speed of a vehicle. These models may show the progression of the attribution and performance as the tasks become more complex.

Higher-level tasks such as parking, which Karpathy [2019] mentioned is performed by Tesla, Inc's end-to-end AI model, should also be evaluated in a later study to determine if the same analysis scales to real-world complex scenarios.

6.6 Conclusion

The use of the GradSUM analysis scheme has shown that some model variants, such as *End-to-End*, could learn meaningful road features from a sparse input signal whilst other model variants, such as *NetHVF* were unable to learn meaningful road features. This shows that the chosen model architecture and weight initialisation significantly impacts

what data is used from the input for any model trying to predict steering angle. The trade-off on increasing computational complexity when performing GradSUM is comparable to a conventional machine learning pipeline's evaluation (testing) phase.

The three biggest model architectures, *NetSVF*, *NetHVF*, and *End-to-End* were on average able to focus on semantically meaningful pixel groups such as the *flat* group in the GradSUM evaluation dataset.

The other three model architectures, *Autonomous Cookbook*, *TestModel1*, and *TestModel2* showed that they were less likely to always learn to focus on semantically meaningful pixel groups. They also showed that they were sensitive to weight initialisation and noise which could produce badly trained models which would introduce more error in their outputs. These architectures would need added analysis to produce the best potential results. They were also the smallest of the model architectures evaluated.

More work is needed to expand the experimental setup to more complex and different problems. This will help determine the usefulness of the GradSUM analysis scheme and will help answer the question: if the input signal size increases, can the same conclusions be made? A customised benchmark can also be used to evaluate more straightforward tasks for self-driving vehicles to collect more valuable metrics. Improving the ground truth dataset to have more data and be more expressive would also improve the fidelity of the results from the GradSUM analysis scheme. Evaluation of newer kinds of model architectures which make use of attention maps is another interesting area of further study.

Chapter 7

Conclusion

End-to-End networks show promise for automating the processing of the large datasets that are now available. The very nature of these models makes them more challenging to understand. Attribution methods help explain what parts of the input data activates the model weights when making predictions. GradCAM is one of these methods. Explaining these models is vital from a legal and moral standpoint as these deep learning systems are actively being used in the real world, where lives and property could be affected when adverse emergent behaviour occurs.

This research aimed to reproduce the experimental setup which Bojarski *et al.* [2016b] defined with an added quantitative analysis of multiple model architectures to explain the features the models would learn from the sparse input data. The experimental setup was defined with a sparse input signal of RGB images from a forward-facing camera inside a vehicle (being driven) which is used to predict the vehicle's steering angle. This setup included a CNN model known as *End-to-End*. Bojarski *et al.* [2016b] found that *"The CNN is able to learn meaningful road features from a very sparse training signal (steering alone)."*

An analysis scheme called GradSUM (Gradient Summation) was devised, which uses a ground truth dataset that contains high-quality segmentation maps and the GradCAM algorithm to quantitatively show what a specific trained model is focusing on in the ground truth dataset. This is done by computing the gradient activation maps for the entire ground truth dataset and then working out the number of activated pixels per category (or group) from the maps using the segmentations. Those counted results are then divided by the total number of those pixels in the ground truth dataset per category (or group) and multiplied by 100 to get a percentage. This scheme allows for a way to compare models against the same ground truth dataset and observe which categories (or groups) relative to other categories (or groups) within the same model epoch have a more significant impact on model activation. This comparison can then be compared with other model's epochs to evaluate the performance and behaviour between two different models.

This body of work showed that the *End-to-End* model variant did indeed learn meaningful road features that are directly related to the task - steering angle prediction. Other model

variants, such as *NetSVF* and *Autonomous Cookbook*, could also learn meaningful road features. The other three analysed model variants, *NetHVF*, *TestModel1*, and *TestModel2*, did not learn meaningful road features.

When investigating model architecture behaviour on average, the *NetSVF*, *NetHVF*, and *End-to-End* model architectures showed that they were able to be biased against semantically relevant pixel groups for predicting steering angle. The other three architectures evaluated showed that they were very sensitive to noise and weight initialisation which would lead to badly performing model variants which would focus on pixel groups that were not semantically relevant to predicting steering angle.

These results points to the conclusion that the model architecture plays a vital role in a model making use of meaningful road features when predicting steering angle from forward-facing images of the road surface.

This work confirmed the result of Bojarski *et al.* [2016b]'s work and expanded on it across more model architectures. However, other questions have been raised around the effect of increasing the amount of input data (such as adding temporal information), changing the task from steering angle prediction to other tasks needed for self-driving cars and modifying the model architectures to focus on meaningful information from the input data or applying newer model architectures and different kinds of heat-maps to the GradSUM analysis. These are potential avenues for future investigation.

Appendix A

Cityscapes Segmentation Labels and Groups

A.1 Counts of Cityscapes Dataset Pixels by Label

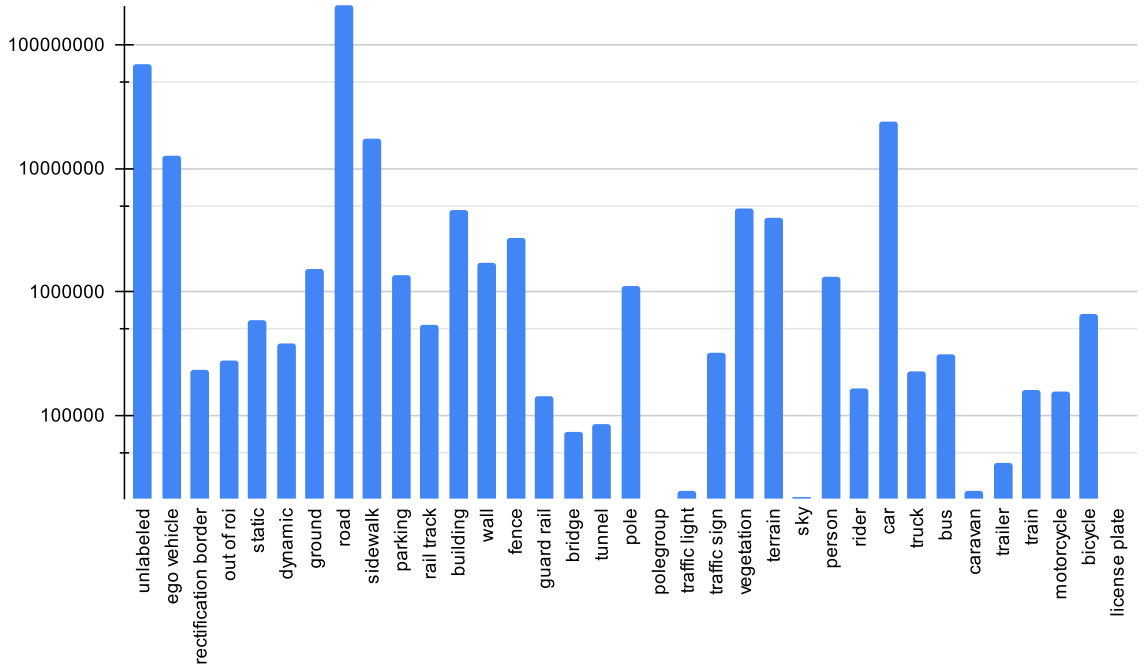
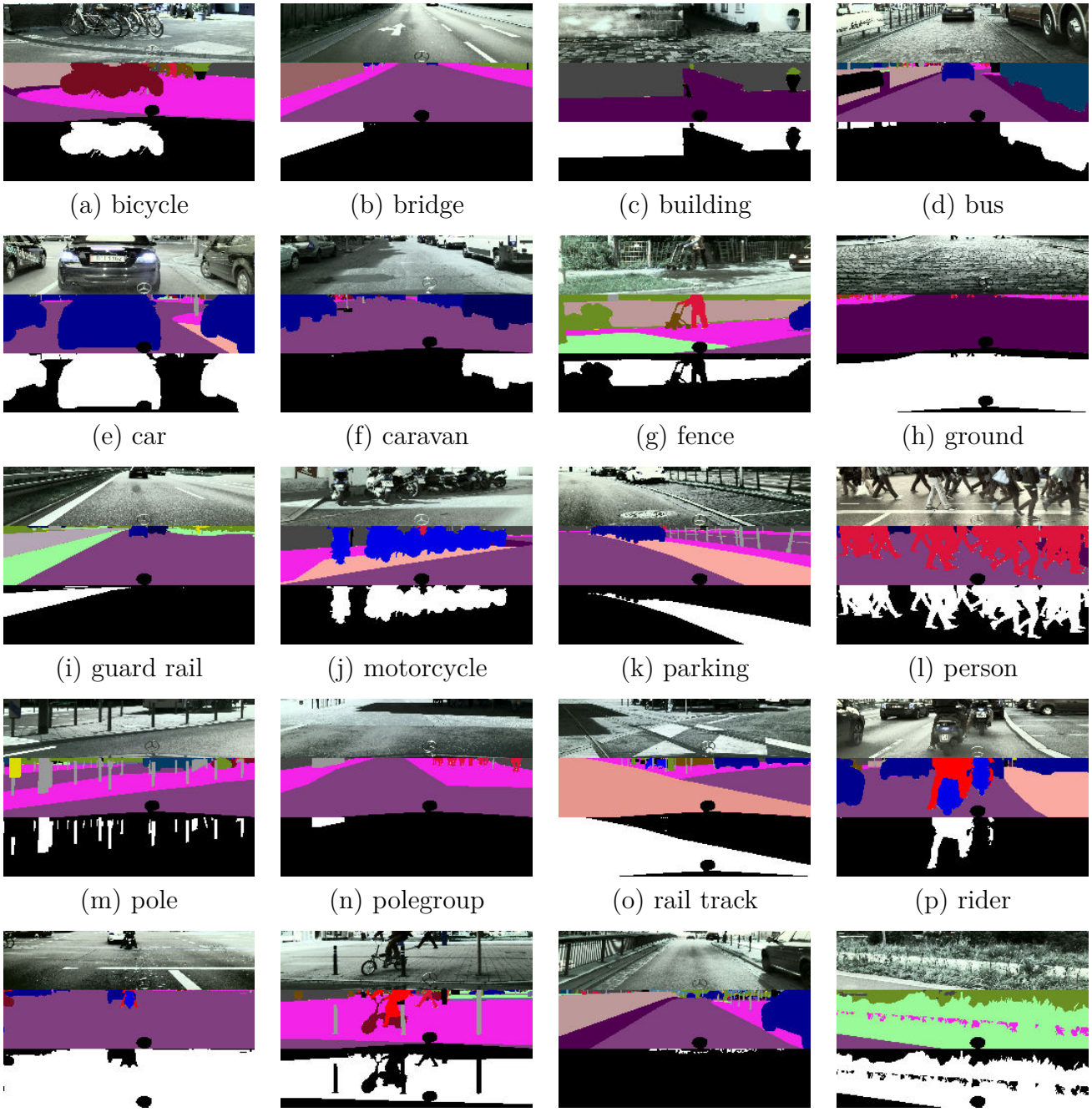


Figure A.1: Scaled Counts of Segmentation Labels from Cityscapes Segmentation Data

A.2 Examples of Each Cityscapes Segmentation Label



(a) bicycle (b) bridge (c) building (d) bus

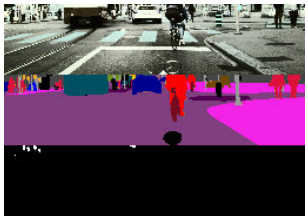
(e) car (f) caravan (g) fence (h) ground

(i) guard rail (j) motorcycle (k) parking (l) person

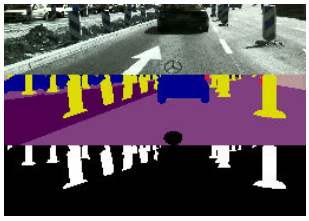
(m) pole (n) polegroup (o) rail track (p) rider

(q) road (r) sidewalk (s) sky (t) terrain

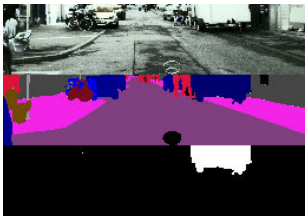
Figure A.2: Examples of Labels from Cityscapes. White represents the label. The original cropped frame, the segmentation map and the label map are shown



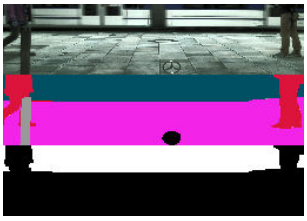
(u) traffic light



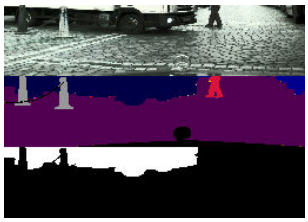
(v) traffic sign



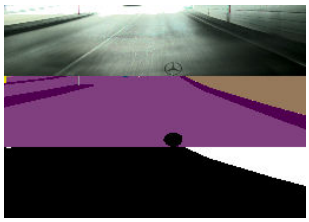
(w) trailer



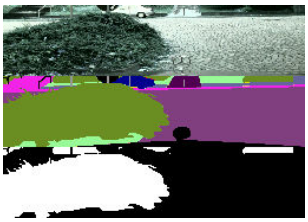
(x) train



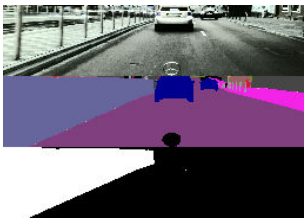
(y) truck



(z) tunnel



(aa) vegetation



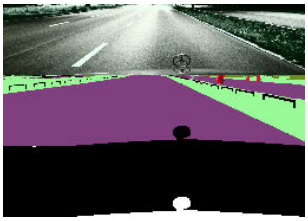
(ab) wall



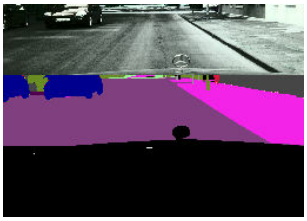
(ac) dynamic



(ad) static



(ae) ego vehicle



(af) rectification border

Figure A.3: Examples of Labels from Cityscapes. White represents the label. The original cropped frame, the segmentation map and the label map are shown

A.3 Pixel Counts of Each Cityscapes Segmentation Group

Table A.1: Pixel Counts per Group from Cityscapes

Group	Number of Pixels (After pre-processing)
flat	228192572
vehicle	25451957
construction	9311061
nature	8700712
object	1490951
human	1483559
sky	22068

A.4 Binary Split for Each Cityscapes Label

Table A.2: Comparison of each Cityscapes segmentation label sample counts for the simple binary classification problem

Dataset Split		Train			Validation		
Group Name	Label Name	Yes	No	Total	Yes	No	Total
void	ground	2973	2	2975	500	0	500
	dynamic	1631	1344		301	199	
	static	2728	247		470	30	
sky	sky	1366	1609		222	278	
nature	vegetation	2768	207		471	29	
	terrain	2483	492		400	100	
object	pole	2866	109		483	17	
	polegroup	2352	623		400	100	
	traffic sign	2206	769		376	124	
	traffic light	1774	1201		271	229	
construction	building	2882	93		478	22	
	wall	2612	363		446	54	
	fence	2375	600		397	103	
	guard rail	1975	1000		353	147	
	bridge	2231	744		391	109	
	tunnel	2060	915		343	157	
vehicle	car	2810	165		477	23	
	truck	838	2137		188	312	
	bus	754	2221		180	320	
	motorcycle	733	2242		166	334	
	bicycle	1546	1429		333	167	
	caravan	327	2648		68	432	
	trailer	513	2462		114	386	
human	person	2730	245		445	55	
	rider	2033	942		368	132	
flat	road	2974	1		499	1	
	sidewalk	2961	14		496	4	
	parking	2706	269		439	61	
	rail track	2330	645		380	120	

Appendix B

From Games Segmentation Labels and Groups

B.1 Counts of From Games Dataset Pixels by Label

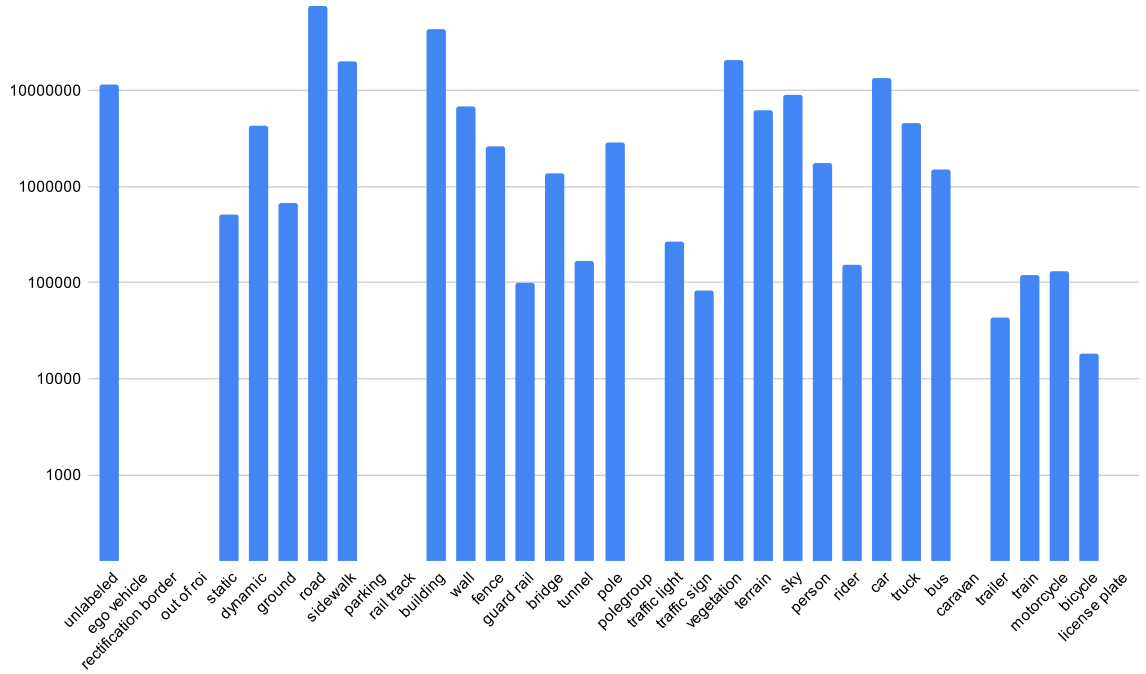
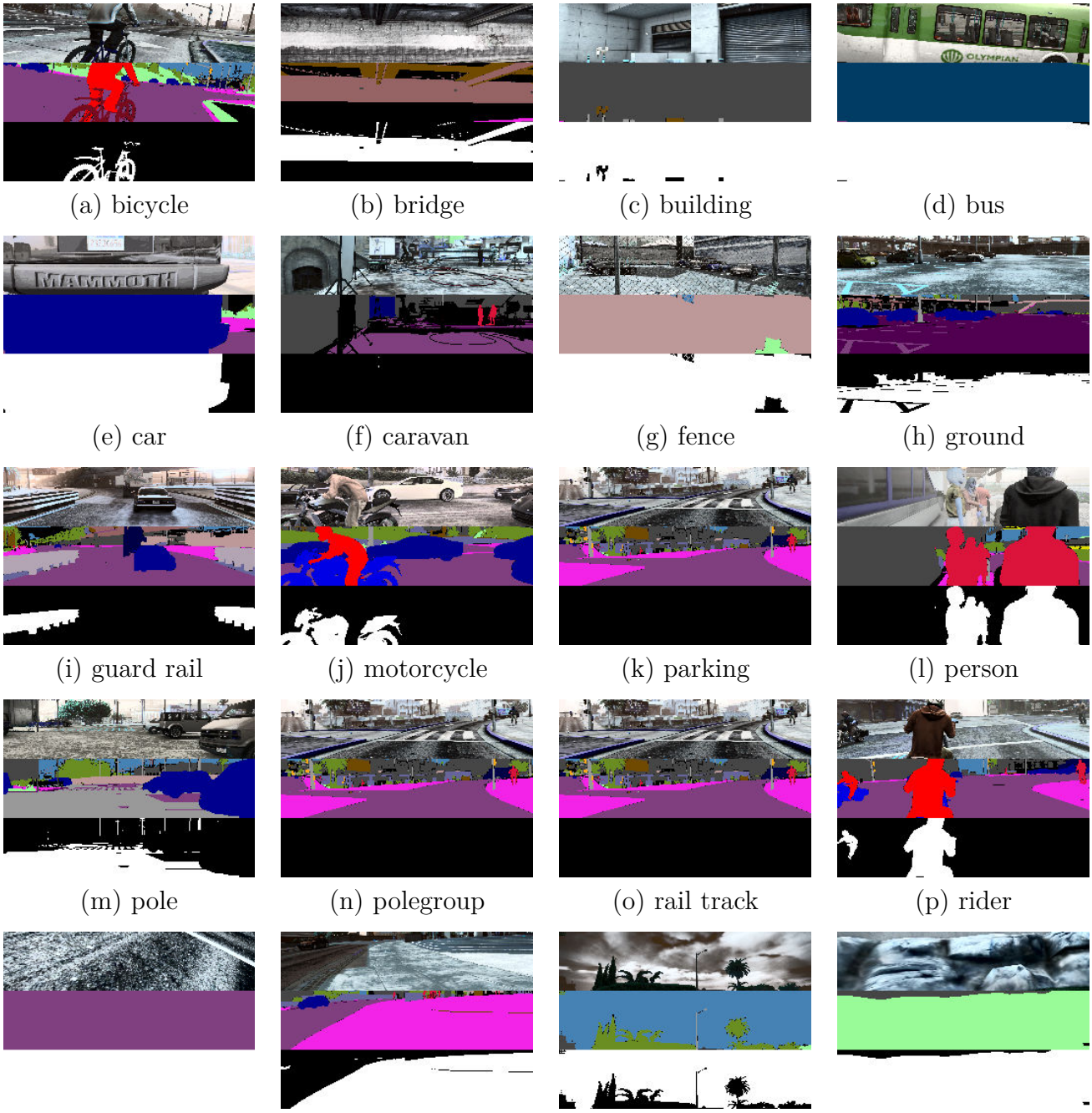


Figure B.1: Scaled Counts of Segmentation Labels from From Games Segmentation Data

B.2 Examples of Each From Games Segmentation Label



(a) bicycle (b) bridge (c) building (d) bus

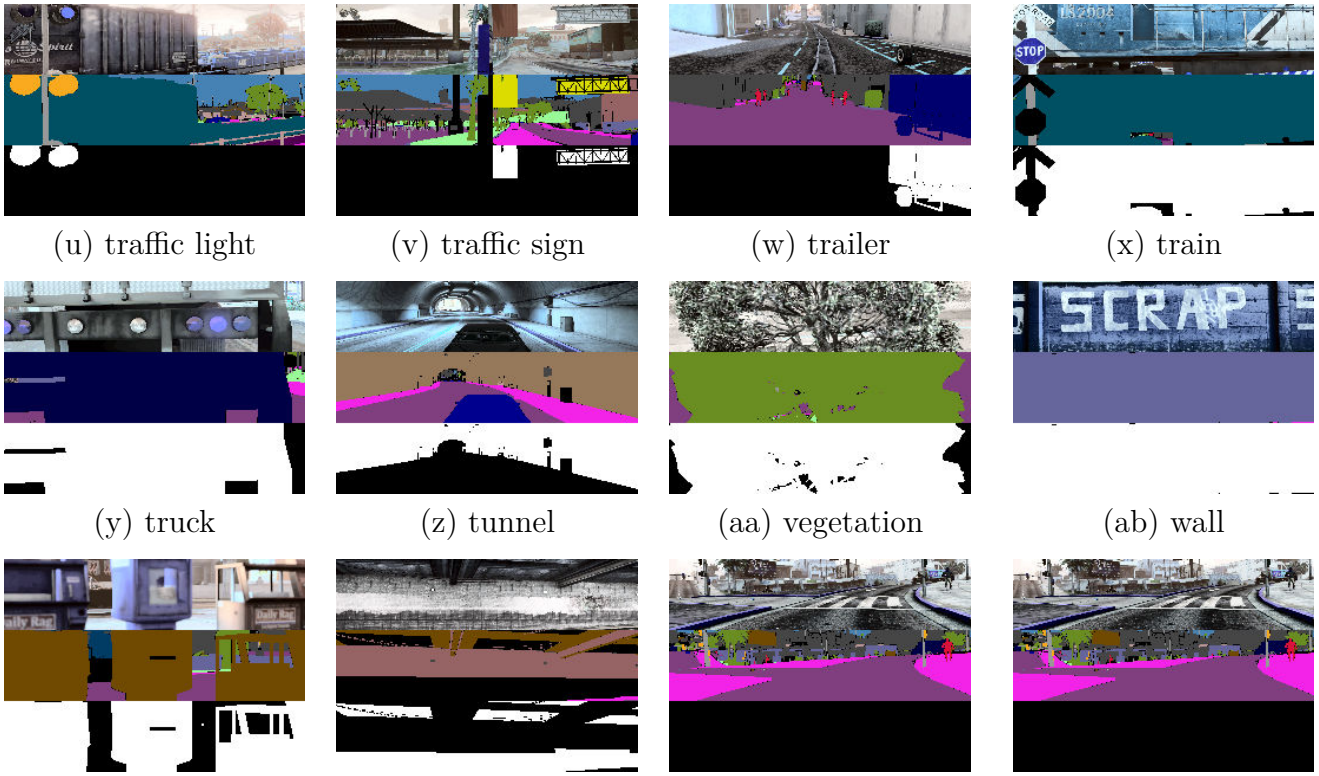
(e) car (f) caravan (g) fence (h) ground

(i) guard rail (j) motorcycle (k) parking (l) person

(m) pole (n) polegroup (o) rail track (p) rider

(q) road (r) sidewalk (s) sky (t) terrain

Figure B.2: Examples of Labels from From games. White represents the label. The original cropped frame, the segmentation map and the label map are shown



(u) traffic light (v) traffic sign (w) trailer (x) train

(y) truck (z) tunnel (aa) vegetation (ab) wall

(ac) dynamic (ad) static (ae) ego vehicle (af) rectification border

Figure B.3: Examples of Labels from From Games. White represents the label. The original cropped frame, the segmentation map and the label map are shown

B.3 Pixel Counts of Each From Games Segmentation Group

Table B.1: Pixel Counts per Group from From Games

Group	Number of Pixels (After pre-processing)
flat	96311959
vehicle	19972604
construction	55029767
nature	27076788
object	3271969
human	1932494
sky	9139693

References

[Adebayo *et al.* 2018] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian J. Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Conference on Neural Information Processing Systems*, 2018.

[Administration 2023] National Highway Traffic Safety Administration. *RCLRPT-23V085-3451.PDF*. https://static.nhtsa.gov/odi/rcl/2023/RCLRPT-23V085-3451.PDF, 02 2023. (Accessed on 04/02/2023).

[Amini *et al.* 2022] Alexander Amini, Tsun-Hsuan Wang, Igor Gilitschenski, Wilko Schwarting, Zhijian Liu, Song Han, Sertac Karaman, and Daniela Rus. Vista 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2419–2426, 2022.

[Ancona *et al.* 2017] Marco B Ancona, Enea Ceolini, A. Cengiz Öztireli, and Markus H. Gross. A unified view of gradient-based attribution methods for deep neural networks. In *NIPS Workshop on Interpreting, Explaining and Visualizing Deep Learning*, 2017.

[Antipov *et al.* 2015] Grigory Antipov, Sid-Ahmed Berrani, Natacha Ruchaud, and Jean-Luc Dugelay. Learned vs. hand-crafted features for pedestrian gender recognition. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, pages 1263–1266, New York, NY, USA, 2015. ACM. (Accessed on 03/12/2023).

[Baeldung 2023] Baeldung. *How ReLU and Dropout Layers Work in CNNs — Baeldung on Computer Science*. Technical report, 04 2023. (Accessed on 05/24/2023).

[Barkan *et al.* 2021] Oren Barkan, Omri Armstrong, Amir Hertz, Avi Caciularu, Ori Katz, Itzik Malkiel, and Noam Koenigstein. Gam: Explainable visual similarity and classification via gradient activation maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 68–77, New York, NY, USA, 2021. Association for Computing Machinery.

[Bojarski *et al.* 2016a] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry D. Jackel, Urs Muller, and Karol Zieba. Visualbackprop: visualizing cnns for autonomous driving. *CoRR*, abs/1611.05418, 2016.

[Bojarski *et al.* 2016b] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs

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

[Bojarski *et al.* 2018] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry D. Jackel, Urs Muller, and Karol Zieba. Visualbackprop: efficient visualization of cnns. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4701–4708, May 2018.

[Cambridge Spark 2018] Cambridge Spark. *50 Free Machine Learning Datasets: Self-Driving Cars*. https://blog.cambridgespark.com/50-free-machine-learning-datasets-self-driving-cars-d37be5a96b28, December 2018. (Accessed on 09/27/2019).

[Cameron 2016] Oliver Cameron. *Open Sourcing 223GB of Driving Data - Udacity Inc - Medium*. https://medium.com/udacity/open-sourcing-223gb-of-mountain-view-driving-data-f6b5593fbfa5, October 2016. (Accessed on 09/27/2019).

[Cao and Wu 2022] Yun-Hao Cao and Jianxin Wu. A random cnn sees objects: One inductive bias of cnn and its applications. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(1):194–202, Jun. 2022.

[CARLA Autonomous Driving Leaderboard 2023] CARLA Autonomous Driving Leaderboard. *CARLA Autonomous Driving Leaderboard*. https://leaderboard.carla.org/, 2023. (Accessed on 06/25/2023).

[CARLA Simulator 2023] CARLA Simulator. *CARLA Simulator*. https://carla.org/, 2023. (Accessed on 06/25/2023).

[Cordts *et al.* 2016] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[Corso *et al.* 2022] Anthony Corso, Robert Moss, Mark Koren, Ritchie Lee, and Mykel Kochenderfer. A survey of algorithms for black-box safety validation of cyber-physical systems. *J. Artif. Int. Res.*, 72:377–428, Jan 2022.

[Devlin *et al.* 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[Dosovitskiy *et al.* 2017] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[Dosovitskiy *et al.* 2020] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[Du *et al.* 2017] Shuyang Du, Haoli Guo, and Andrew Simpson. *Self-Driving Car Steering Angle Prediction Based on Image Recognition*. Technical report, 2017.

[Forbes 2023] Forbes. *Tesla Recall Hits Nearly 363,000 Cars With "Full Self-Driving" Software*. https://www.forbes.com/sites/qai/2023/02/20/tesla-recall-hits-nearly-363000-cars-with-full-self-driving-software/, 02 2023. (Accessed on 04/02/2023).

[Fridman *et al.* 2017] Alex Fridman, Daniel E. Brown, Michael Glazer, William Angell, Spencer Dodd, Benedikt Jenik, Jack Terwilliger, Aleksandr Patsekin, Julia Kindelsberger, Li Ding, Sean Seaman, Alea Mehler, Andrew Sipperley, Anthony Pettinato, Bobbie D. Seppelt, Linda Angell, Bruce Mehler, and Bryan Reimer. Mit advanced vehicle technology study: Large-scale naturalistic driving study of driver behavior and interaction with automation. *IEEE Access*, 7:102021–102038, 2017.

[Gildenblat and contributors 2021] Jacob Gildenblat and contributors. *PyTorch library for CAM methods*. https://github.com/jacobgil/pytorch-grad-cam, 2021.

[Gildenblat 2022] Jacob Gildenblat. *How does it work with Vision Transformers — Advanced AI explainability with pytorch-gradcam*. https://jacobgil.github.io/pytorch-gradcam-book/vision_transformers.html, 2022. (Accessed on 03/03/2024).

[GitHub 2023] GitHub. *GitHub Docs*. Technical report, 2023. (Accessed on 06/25/2023).

[Glorot and Bengio 2010] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[Gómez *et al.* 2010] David Gerónimo Gómez, Antonio M. López, Angel Domingo Sappa, and Thorsten Graf. Survey of pedestrian detection for advanced driver assistance systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:1239–1258, 2010.

[Gonzalez *et al.* 2017] Eric Gonzalez, MacCallister Higgins, and Oliver Cameron. *The Udacity open source self-driving car project*. https://github.com/udacity/self-driving-car, 2017. (Accessed on 09/27/2019).

[Goodfellow *et al.* 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[Gordon 2022] Rachel Gordon. *Researchers release open-source photorealistic simulator for autonomous driving — MIT News — Massachusetts Institute of Technology*. https://news.mit.edu/2022/researchers-release-open-source-photorealistic-simulator-autonomous-driving-0621 June 2022. (Accessed on 06/25/2023).

[Gringer 2019] Bonnie Gringer. *History of the Autonomous Car*. https://www.titlemax.com/resources/history-of-the-autonomous-car/, 2019. (Accessed on 09/29/2019).

[Guidotti *et al.* 2018] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5), aug 2018.

[Karpathy 2019] Andrej Karpathy. *PyTorch at Tesla - Andrej Karpathy, Tesla - YouTube*. https://www.youtube.com/watch?v=oBklltKXtDE, November 2019. (Accessed on 06/25/2023).

[Li *et al.* 2023] Yiran Li, Junpeng Wang, Xin Dai, Liang Wang, Chin-Chia Michael Yeh, Yan Zheng, Wei Zhang, and Kwan-Liu Ma. How does attention work in vision transformers? a visual analytics attempt. *IEEE Transactions on Visualization and Computer Graphics*, 29:2888–2900, 2023.

[Li 2017] Yuxi Li. Deep reinforcement learning: An overview. *ArXiv*, abs/1701.07274, 2017.

[Liu *et al.* 2016] Peng Liu, Arda Kurt, Keith Redmill, and Umit Ozguner. Classification of highway lane change behavior to detect dangerous cut-in maneuvers. In *The Transportation Research Board (TRB) 95th Annual Meeting, vol. 2*, 2016.

[Maddern *et al.* 2017] Will Maddern, Geoff Pascoe, Chris Linegar, and Paul Newman. 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*, 36(1):3–15, 2017. (Accessed on 03/12/2023).

[Microsoft Research 2018] Microsoft Research. *AirSim Documentation*. https://microsoft.github.io/AirSim/, 2018. (Accessed on 06/25/2023).

[NHTSA 2017] NHTSA. *Automated Driving Systems: A Vision for Safety — US Department of Transportation*. https://www.transportation.gov/policy-initiatives/automated-vehicles/automated-driving-systems-vision-safety, September 2017. (Accessed on 06/20/2023).

[Nie *et al.* 2018] Weili Nie, Yang Zhang, and Ankit Patel. A theoretical explanation for perplexing behaviors of backpropagation-based visualizations. In *International Conference on Machine Learning*, 2018.

[NumPy 2023] NumPy. *NumPy Scientific Computing Library for Python*. https://numpy.org/, 2023. (Accessed on 06/25/2023).

[OpenCV: Geometric Image Transformations Documentation] OpenCV: Geometric Image Transformations Documentation. *OpenCV: Geometric Image Transformations*. https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html. (Accessed on 07/05/2023).

[OpenCV 2023] OpenCV. *OpenCV - Open Computer Vision Library*. https://opencv.org/, 2023. (Accessed on 06/25/2023).

[Paden *et al.* 2016] Brian Paden, Michal Cáp, Sze Zheng Yong, Dmitry S. Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1:33–55, 2016.

[Papers With Code 2023] Papers With Code. *Cityscapes Dataset — Papers With Code*. https://paperswithcode.com/dataset/cityscapes, 2023. (Accessed on 06/11/2023).

[Python Programming Language 2023] Python Programming Language. *Python Programming Language*. https://www.python.org/, 2023. (Accessed on 06/25/2023).

[PyTorch 2.0 documentation 2023] PyTorch 2.0 documentation. *PyTorch 2.0 documentation*. https://pytorch.org/docs/stable/nn.init.html, 2023. (Accessed on 06/25/2023).

[PyTorch Machine Learning Framework 2023] PyTorch Machine Learning Framework. *PyTorch Machine Learning Framework*. https://pytorch.org/, 2023. (Accessed on 06/25/2023).

[Richter *et al.* 2016a] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016.

[Richter *et al.* 2016b] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. *Playing for Data: Ground Truth from Computer Games — Source Website*. https://download.visinf.tu-darmstadt.de/data/from_games/, 2016. (Accessed on 09/09/2024).

[Rosebrock 2021] Adrian Rosebrock. *OpenCV Edge Detection (cv2.Canny) - PyImageSearch*. Technical report, May 2021. (Accessed on 06/25/2023).

[Ruby Programming Language 2023] Ruby Programming Language. *Ruby Programming Language*. https://www.ruby-lang.org/en/, 2023. (Accessed on 06/25/2023).

[Schafer *et al.* 2018] Harald Schafer, Eder Santana, Andrew Haden, and Riccardo Biasini. *A Commute in Data: The comma2k19 Dataset*, 2018.

[Selvaraju *et al.* 2016] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2016.

[Shah *et al.* 2018] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Marco Hutter and Roland Siegwart, editors, *Field and Service Robotics*, pages 621–635, Cham, 2018. Springer International Publishing.

[Shepardson 2023] David Shepardson. *Tesla recalls 362,000 U.S. vehicles over Full Self-Driving software — Reuters*. https://www.reuters.com/business/autos-transportation/

tesla-recalls-362000-us-vehicles-over-full-self-driving-software-2023-02-16/, 02 2023. (Accessed on 04/02/2023).

[Simonyan *et al.* 2013] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[Spryn and Sharma 2018] Mitchell Spryn and Aditya Sharma. *The Autonomous Driving Cookbook: Scenarios, tutorials and demos for Autonomous Driving.* https://github.com/Microsoft/AutonomousDrivingCookbook, 2018. (Accessed on 09/27/2019).

[Wang *et al.* 2018] Peng Wang, Xinyu Huang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[Weights & Biases 2023] Weights & Biases. *Weights & Biases.* https://wandb.ai/, 2023. (Accessed on 06/25/2023).

[Weng 2019] Lilian Weng. *Domain Randomization for Sim2Real Transfer.* https://lilianweng.github.io/posts/2019-05-05-domain-randomization/, 2019. (Accessed on 03/12/2023).

[Yu *et al.* 2018] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *ArXiv*, abs/1805.04687, 2018.

[Yu *et al.* 2020] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.