Analysis of Robustness in Lane Detection using Machine Learning Models

A thesis presented to
the faculty of
the Russ College of Engineering and Technology of Ohio University

In partial fulfillment
of the requirements for the degree
Master of Science

William A. Adams

December 2015

© 2015 William A. Adams. All Rights Reserved.
This thesis titled
Analysis of Robustness in Lane Detection using Machine Learning Models

by
WILLIAM A. ADAMS

has been approved for
the School of Electrical Engineering and Computer Science
and the Russ College of Engineering and Technology by

Mehmet Celenk
Professor of Electrical Engineering and Computer Science

Dennis Irwin
Dean, Russ College of Engineering and Technology
ABSTRACT

ADAMS, WILLIAM A., M.S., December 2015, Electrical Engineering

Analysis of Robustness in Lane Detection using Machine Learning Models

Director of Thesis: Mehmet Celenk

An appropriate approach to incorporating robustness into lane detection algorithms is beneficial to autonomous vehicle applications and other problems relying on fusion methods. While traditionally rigorous empirical methods were developed for mitigating lane detection error, an evidence-based model-driven approach yields robust results using multispectral video as input to various machine learning models.

Branching beyond the few network structures considered for image understanding applications, deep networks with unique optimization functions are demonstrably more robust while making fewer assumptions. This work adopts a simple framework for data collection; retrieving image patches for comparison via regression through a learning model. Along a horizontal scanline, the most probable sample is selected to retrain the network. Models include simple regressors, various autoencoders, and a few specialized deep networks. Samples are compared by robustness and the results favor deep and highly specialized network structures.
A special thanks to Rebeka Bolmida and all my family and friends for providing the love and support necessary to accomplish my dreams.
ACKNOWLEDGMENTS

Fundamental in this work over the last 5 years and responsible for first aiding in my endeavor and passion for signal processing, Dr. Mehmet Celenk has helped me through every step from the first course in signal processing to the thesis.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>3</td>
</tr>
<tr>
<td>Dedication</td>
<td>4</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>5</td>
</tr>
<tr>
<td>List of Tables</td>
<td>8</td>
</tr>
<tr>
<td>List of Figures</td>
<td>9</td>
</tr>
<tr>
<td>Chapter 1: Objectives</td>
<td>11</td>
</tr>
<tr>
<td>Chapter 2: Introduction</td>
<td>12</td>
</tr>
<tr>
<td>Chapter 3: Background</td>
<td>14</td>
</tr>
<tr>
<td>Scene Understanding Schemes</td>
<td>14</td>
</tr>
<tr>
<td>Data-Driven Schemes</td>
<td>15</td>
</tr>
<tr>
<td>Model-Driven Schemes</td>
<td>15</td>
</tr>
<tr>
<td>Lane Analysis as a Subtask</td>
<td>16</td>
</tr>
<tr>
<td>Region Labeling Approaches</td>
<td>18</td>
</tr>
<tr>
<td>The Lane Detection Problem</td>
<td>19</td>
</tr>
<tr>
<td>Modern Lane Detection</td>
<td>19</td>
</tr>
<tr>
<td>Classical Lane Detection</td>
<td>20</td>
</tr>
<tr>
<td>Conclusion</td>
<td>21</td>
</tr>
<tr>
<td>Chapter 4: Methodology</td>
<td>23</td>
</tr>
<tr>
<td>General Assumptions</td>
<td>23</td>
</tr>
<tr>
<td>Physical Orientation</td>
<td>23</td>
</tr>
<tr>
<td>Data Domain Analysis</td>
<td>24</td>
</tr>
<tr>
<td>The Data</td>
<td>25</td>
</tr>
<tr>
<td>Machine Learning Models</td>
<td>28</td>
</tr>
<tr>
<td>Learning Models Overview</td>
<td>28</td>
</tr>
<tr>
<td>How Data is Input</td>
<td>45</td>
</tr>
<tr>
<td>Regressors</td>
<td>52</td>
</tr>
<tr>
<td>AutoEncoders</td>
<td>57</td>
</tr>
<tr>
<td>Deep Networks</td>
<td>62</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Condensed Confusion Matrix for Regressors</td>
<td>83</td>
</tr>
<tr>
<td>Table 2</td>
<td>Hard Metrics for Regressors</td>
<td>86</td>
</tr>
<tr>
<td>Table 3</td>
<td>Condensed Confusion Matrix for Autoencoders</td>
<td>91</td>
</tr>
<tr>
<td>Table 4</td>
<td>Hard Metrics for Autoencoders</td>
<td>92</td>
</tr>
<tr>
<td>Table 5</td>
<td>Condensed Confusion Matrix for Deep Autoencoders</td>
<td>97</td>
</tr>
<tr>
<td>Table 6</td>
<td>Hard Metrics for Deep Autoencoders</td>
<td>97</td>
</tr>
<tr>
<td>Table 7</td>
<td>Condensed Confusion Matrix for Deep Autoencoders</td>
<td>99</td>
</tr>
<tr>
<td>Table 8</td>
<td>Hard Metrics for Deep Autoencoders</td>
<td>99</td>
</tr>
<tr>
<td>Table 9</td>
<td>Condensed Confusion Matrix for All Autoencoders</td>
<td>102</td>
</tr>
<tr>
<td>Table 10</td>
<td>Hard Metrics for Deep Autoencoders</td>
<td>103</td>
</tr>
<tr>
<td>Table 11</td>
<td>Condensed Confusion Matrix for All Regressors</td>
<td>104</td>
</tr>
<tr>
<td>Table 12</td>
<td>Hard Metrics for All Regressors</td>
<td>104</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Overall Design of Research</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Video Examples</td>
<td>26</td>
</tr>
<tr>
<td>Figure 3</td>
<td>BGR Image Data Representation</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Hypothesized ADAS lane detector original model</td>
<td>30</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Streamlined hypothesized ADAS lane detector model</td>
<td>30</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Two-Stage hypothesized ADAS lane detector model</td>
<td>31</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Neural Network ADAS lane detector model</td>
<td>32</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Optimization Through Minimization Overview</td>
<td>38</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Image Data Extraction</td>
<td>45</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Flow Diagram of Normalization</td>
<td>46</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Data Normalization</td>
<td>47</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Window Selection</td>
<td>48</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Model Training and Testing Comparison</td>
<td>51</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Regressor Structure</td>
<td>52</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Relation between Convolution and Regression</td>
<td>53</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Normal Use of Autoencoders</td>
<td>58</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Dynamic Autoencoder Process</td>
<td>58</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Autoencoder Structure</td>
<td>59</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Deep Autoencoder Structure</td>
<td>64</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Deep Regressor Structure</td>
<td>64</td>
</tr>
</tbody>
</table>
Figure 21: Confusion Matrix Determination .................................................................73
Figure 22: Area Under the Curve Determination ..........................................................75
Figure 23: Missed Track Determination .....................................................................76
Figure 24: Worst Performing Regressors .................................................................81
Figure 25: Best Performing Regressors ....................................................................81
Figure 26: Regressor Model Error for Plain Set .......................................................84
Figure 27: Regressor Model Error for Glare Set .......................................................84
Figure 28: Regressor Model Error for Shadow Set ..................................................85
Figure 29: Regressor Model Error for Snow Set .......................................................85
Figure 30: Convolutional Filters from Regressors.....................................................87
Figure 31: Error of Autoencoders with Linear Activation .......................................89
Figure 32: Error of Autoencoders with Rectified Linear Activation .......................89
Figure 33: Error of Autoencoders with Hyperbolic Tangential Activation .............90
Figure 34: Autoencoder 500-tanh Performance ......................................................93
Figure 35: Deep Autoencoder Error Characteristics ..............................................96
Figure 36: Deep Regressor Error Characteristics ...............................................98
Figure 37: Comparison between Color and Grayscale Regressors .......................100
Figure 38: Comparison of Autoencoders with Linear Activation .........................101
Figure 39: Comparison of Autoencoders with Rectified Linear Activation ..........101
CHAPTER 1: OBJECTIVES

In this document, I will attempt to convey the radical way in which machine
learning has changed the approach taken in scene understanding tasks through the
evaluation of its performance in the popular lane detection problem. The origins of image
processing lie in signal processing. As the field progressed, the process of data
acquisition, processing, and evaluation became a standard for researchers to undergo by
hand. Later, filters which maintained some memory, called adaptive, were shown to have
an advantage over filters without such memory. As computational power became more
readily available, automation of the analysis process became more forthcoming. A recent
addition to the field, pattern recognition, proved its dominance over even highly adaptive
filters in terms of performance but at a detriment to efficiency. In the new age of big data
and the gpu, the complete automation of on-the-fly pattern recognition coupled with new
representations for model assumptions to produce machine learning.

In this work, I will show how machine learning became the dominant force
behind image processing and computer vision, I will provide a solid example in the lane
detection problem of how this trend has progressed, I will show the performance of a
variety of machine learning models for this task, and I will provide an analysis of the
results in terms of model structure.
CHAPTER 2: INTRODUCTION

As machines are slowly but surely taking over the roadways, their sensor systems are becoming increasingly sophisticated. Recently Google, an internet search company, has announced their remarkable fully autonomous ground based vehicle; revolutionizing the car industry [1]. Their ability to sweep aside long-standing established car dynasties is rooted in what their company was founded on: machine learning. Everything from the natural language processing their search algorithm employs to the optimized performance of their server farms has artificial intelligence built in. Applying the same approach to a cornucopia of sensors on a moving platform produces an autonomous vehicle.

Google, however, is a well-established company with more capital and resources than any one person can afford. Autonomous vehicles cannot make the roadway any safer in a golden garage. While the cost of lidar (an active sensor emitting a known optical pattern to detect distance) is decreasing, multispectral cameras offer a more cost effective alternative. At lower costs, these vehicles should be better able to flood the market and prevent an estimated thirty five thousand motor vehicle deaths per year in the United States alone [2].

While advancements in the use of computer vision have better enabled electro-optical (EO) sensor control of autonomous and semi-autonomous vehicles, there still remains a great deal of uncertainty. Classical problems like obtaining depth information, tracking moving objects, and incorporating new information into object models require increasingly creative solutions. These solutions are prone to being highly specialized on the specific dataset available and are often gravely limited by numerous assumption. This
work addresses a seemingly more appropriate approach to analyzing the highly aggregate data that is multispectral video. The overall format of this research is shown in the following figure.

A highly varied and challenging dataset is used to develop thirty-eight machine learning models with unique characteristics. Observations from the model development of each model and among classes of models are compiled to provide an analysis of the lane detection problem using visual spectrum cameras in terms of robustness. As the spark that began with the scientific approach and culminated as the engine behind the big data movement, machine learning has more than enough power to address computer vision’s lane detection problem.
CHAPTER 3: BACKGROUND

Computer vision takes root in signal processing; wherein the effect of a system on a signal is studied and frameworks for exploring this effect are examined. The traditional black box is an excellent illustration of the basic premise of signal processing. In general, a signal is input into the black box, propagates through the unknown system therein, and another signal is output. The usual question we tend to ask is “what is in the black box?” Traditionally we may look at the input and output in terms of certain characteristics called metrics to infer some quality of the process undergone inside the black box. A research engineer may even explore different metrics that seem intuitive based on other information about the process and describe the unknown system in a novel way. A data scientist, however, is not confined by this or that set of metrics but explores across all possible sets of metrics (and even some impossible sets). The problem with the black box mentality is that no metric or set of metrics can indicate with certainty the contents of the black box because many unique functions exist with the same solution. While managing complexity using assumptions and approximations may suffice for many other fields of engineering, the data in computer vision is much too complex and voluminous for this approach.

Scene Understanding Schemes

Scene Understanding is at the core of most computer vision tasks and road scene understanding continues to provide a testing ground for many of the theoretical and practical problems faced by computer vision scientists. The umbrella task of scene understanding is the general method of deriving useful information from sensed data
given a model constructed either by the physical relationships and metrics inherent in the system or by a heuristic characterizing some meaningful relationship between features.

**Data-Driven Schemes**

Modeling a scene using the metrics provided by the physical composition of the system is referred to as the data-driven scheme and usually involves a comparison of the sensed data and expected input. This approach typically starts with the natural representation of the data and moves toward a synthetically meaningful representation. Color space transformation is a good example as the data collected is typically the literal measurement from a sensor reacting to the intensity of the light over a given spectrum of wavelengths, usually exhibiting a normal distribution. These sensor readings are often subject to a nonlinear transformation, as the natural color space is non-Euclidean, and made to represent human concepts like hue, saturation, and intensity. Forest et al. in 1994 referred to data-driven schemes as the “bottom-up … strategy” [3] and more recently Valentino in 2010 called it a “sensor based” approach [4]. Though this is traditionally how research is done, a more scientifically sound approach is the model-driven scheme.

**Model-Driven Schemes**

As we will see, the model in a data-driven scheme tends to be driven by knowledge of the system that may be considered a priori. This is a major assumption whereby traditional approaches have found benefit. Without making this assumption, a model-driven scheme learns an incrementally better representation of the data which eventually embodies both a priori and posteriori knowledge. Machine learning is the powerful tool of model-driven schemes. A statistical model, rather than a physics-
grounded one, is favored for its flexibility, ease of use, and power of nondeterministic approximation. Features pertaining to the videos may be broken down into spectral, spatial, temporal, and semantic. More simply, the features may be considered as data range, data domain, and semantic. Instead of treating the problem as a segmented multi-algorithm combination, all feature types are considered together and the true representation is often never shown. Assumptions are made in the form of an optimization objective and network structure and the model is made to reward or punish its features based on what it observes within the context of this framework. Examples in [5]–[7] capture this idea by not explicitly determining a representation of the data as features but rather by noting a relationship between particular feature types such as shape, texture, and color. [8] provides an excellent example wherein the specific object detection task is left to be solved by the learning model while the structure of the model is built. In [8], deep networks are made to capture not only the fact that a detection occurred but also the bounding box where it happened. While these approaches vary tactically, mixtures of the two called hybrids are often employed to complete a specific task. Lane analysis is one such task which has become very useful with the advent of advanced driver assistance systems.

Lane Analysis as a Subtask

Lane analysis is fundamental in the design of lane departure warning systems as well as lane keeping techniques for control systems in autonomous vehicles. Analyzing the lane’s appearance may be accomplished in a number of ways including both methods which build a scene based on detected edges, synthetically meaningful spatio-spectral
features, and methods which rely on distinguishing contiguous regions in the image. While there are practically countless examples of the prior, the work presented here address the later referred to as region labeling approaches.

\textit{Edge Detection Approaches}

As a vast majority of thought on the lane detection problem has leveraged edge detection, it deserves some recognition. An edge is a two-dimensional discontinuity in the spectral data along the image plane. Resulting from an overlap in the depth information as the three dimensional world is projected onto only two, an edge can provide a valuable distinction between objects. It’s just as probable that objects simply consist of multiple colors on the same surface, like a chess board, and thus an edge is produced. Edges also carry statistical significance as they represent a dramatic change in the spectral data along the domain and thus provide information. In the case of lane detection, it is intuitive that edges should indicate either the marking itself or the point on a ground-plane which bounds the driving surface. The assumption of a contiguous image domain, as required by Fourier analysis, is most confused by the presence of edges.

In the literature, a great number of ways to detect edges are explored. Gradient-based methods are most intuitive because they provide a direction of largest change in the image plane. Researchers tackling the lane detection problem who use the gradient as an edge detector include [6], [9]–[16]. Less intuitively, the two-dimensional Gaussian used by [5], [17]–[21] can provide an orientation as well. Extending the Gaussian with a sinusoid, banks of Gabor filters are often thought to provide scale as well as orientation. Gabor filter banks (GFB) are found in the work of [22], [23]. John Canny’s filter from
1986 is a very fast and efficient method for finding edges and is incorporated in the works of [24]–[27]. The Sobel operator or a variation of such is employed by [28]–[30]. Lastly, an even more basic method for describing the texture of a region, contract, is used by [31], [32].

While it seems intuitive to follow this trail and continue with yet another variation of edge detector, we must not forget the shortcoming of the black box mentality. The performance of methods relying on edge features does not necessarily reflect on the usefulness of edges as features. There may be many features that happen to share characteristics with edges that are not being examined. In other words, edges may tell part of the story, but we cannot assume they tell the whole story. If there is an advantage to using edge detection, the machine learning algorithm will naturally reward features leveraging this advantage while venturing beyond to explore features well outside of our intuition and even our imagination.

Region Labeling Approaches

Segmentation via classification seems to be the most powerful and well accepted approach to the analysis of road lanes. Sun et al. in [22] present a hybrid model which uses textural features for segmentation and also to determine the dominant orientation of road lanes as well as the horizon. [17], [33], [34] provide further examples using model-based rather than hybrid approaches. A personally recommended review of lane analysis methods may be found in [35] for the interested reader. While many problems may be solved by lane analysis, the current roadblock in this area has to do with the problem of lane detection.
The Lane Detection Problem

The classical problem of using visual spectrum input to detect road lanes from an egomotion perspective is hindered by poor weather conditions. In [35] robustness is defined by “high accuracy across a defined range of scenarios.” This roughly translates to a model’s ability to maintain a good working definition of the road lane in scenarios where other models fail. The definition in this case is a set of noise-tolerant features and learned feature relationships. Among these methods, a paradigm shift has presented itself as a preprocessing step which separates modern from classical lane detection approaches.

Modern Lane Detection

Inverse perspective mapping (IPM) is the mark of a modern lane detection algorithm. This preprocessing step relies on the camera parameters including focal length, position, and orientation to transform the image plane directly into the top-view of the physical ground plane. Many optimizations exist which involve interpolating the farther regions for feature consistency and computation efficiency. Work done typically after the year 2000 like that of [6], [11], [16]–[18], [31] are likely to make use of IPM as an initial step. These works also benefited from having a consistent scale for which to employ edge features. The work of Tsukada et al. in [6] specifically addresses the scaling issue, “[b]y using a pyramid of gradient and trilinear mimimapping interpolation as an approximation … we can compute efficiently the adaptive gradient image using GPU capabilities.” The pyramid here refers to a scaling of the image plane by interpolation which acts as a digital focus. While IPM may provide some advantages in the use of intuitive features, robustness experiences a detriment as model assumptions are
strengthened. Robustness is obtainable in these models, however, during the model fitting or the temporal update steps.

The data-driven approach for modern lane detection updates in the model fitting step wherein a priori assumptions are made about the spatial features representing shape. [31] and [20] show good examples of robustness being achieved in the feature set by selecting features which are more tolerant whereas [16] uses a temporal features which disregards outliers for robustness. While this approach has shown good results, the assumptions limit how well the algorithm can be generalized to other applications.

The model-driven approach yields a more generalizable algorithm because it makes fewer assumptions and incorporates robustness in the learning process by revising the features to accommodate the diversity of the input. While [5], [13], [14], [21], [22], [25], [30], [36]–[40] use a model-driven approach to accomplish lane detection, [6], [11], [16] also use IPM as a preprocessing step in a modern approach. Despite the power of modern lane detection solutions, the Achilles heel of the modern approach is its dependence on IPM; a weakness from which classical lane detection algorithms do not suffer.

Classical Lane Detection

Without requiring contextual knowledge of the camera parameters, classical lane detection starts with a more robust platform. Fewer assumptions means an algorithm can generalize easier and be applied to other scenarios by then making assumptions pertaining to that scenario. In the absence of IPM, the structural model must bear the burden of compensating by temporal filtering in spatial and semantic features.
representing shape. Similar to the modern approach, a lane may be detected robustly using a model-driven or data-driven scheme. Model-driven classical detection algorithms often use a deformable definition of the lane as a template [13], [14] or a hypothesis [19]. Hybrid and data-driven models of this sort tend to employ a form of filtering for matching the data to their model as shown in [37] by filtering morphological spatial features and in [12], [19], [20], [26], [41], [42] via a Kalman filter. As per usual, a hybrid model may be best to fulfil the requirements of lane detection while continuing the classical approach in making the fewest assumptions.

The research presented extends the classical framework for solving the lane detection problem and employs a hybrid model for a synergistic effect benefiting from the efficiency and speed of a data-driven approach and the robustness of a model-driven approach. A region containing the road lane marking is inferred and a scanline is laid out for tracking the marking. Analytical methods are employed for determining feature sets most representative of the lane marking and background. Methods are discussed and compared by which these clusters may be tracked in time to achieve maximum robustness. Lastly, results are presented showing the effectiveness of each method under various road conditions including lane markings which are clearly visible (called plain), subject to changes in illumination, changes in spectral reflectance, and occlusion by snow.

Conclusion

The theory behind scene understanding is the core of all computer vision tasks and can be applied even to the practical application of lane detection. While lane models
may rely on assumptions of previous knowledge concerning shape or on a
nondeterministic framework of undescribed feature relationships, hybrid models boast a
synergy between the two. In this thesis, the notable and recent machine learning
algorithms are described in chapter 4. The classical approach is extended to incorporate
state-of-the-art machine learning algorithms. Analysis of the resulting performance when
tested on real-world samples is explored in chapter 5 and a discussion of future work in
this area is given in chapter 6.
CHAPTER 4: METHODOLOGY

A plethora of machine learning algorithms are tested here, spanning different aspects of network structure. With a general objective of determining which aspects of network structure best reflect the data, it is important to adequately compare and contrast between them. As the approach to feature-finding in machine learning is heavily tied to the structure of the network, no stable comparison may be made sufficiently regarding those parameters intrinsic to the models. However, the performance given a variety of tasks may be explored in postprocessing analysis. In an attempt to address these extrinsic parameters, the methodology is focused around obtaining performance metrics.

General Assumptions

In tackling the problem of lane detection, a few general assumptions must be made regarding the approach itself. These assumptions will aid in the development of a framework and are broken down into those concerning physical orientation and those concerning the domain of the data.

Physical Orientation

The position and orientation of the camera is perhaps the most obvious assumption that must be made. In this work, the multispectral sensor is forward-facing and fixed just behind the windshield of the vehicle. It is assumed the horizontal of the image frame is loosely aligned with the true horizon of the real ground plane. While typically the vehicle is moving on a forward trajectory, no assumption of this is implied. The existence of a horizontal axis (scanline) along which one lane marking appears is assumed. This implies both the existence of a lane marking and the existence of a
scanline which intersects it. Together, the assumptions of physical orientation imply that
there is a camera behind the windshield facing forward which is aligned as the driver
would see and has a downward angle such that a scanline exists which intersects a visible
lane marking. More simply, the camera should see what the driver would see.

Data Domain Analysis

Assumptions regarding the data’s domain are significantly more involved and less
straight-forward. Starting simply, the assumption is made that the road lane marking has
a different spectral response than the road itself. Obviously, the camera must not only be
physically oriented to capture the lane marking but also spectrally capable of
distinguishing it. Because lane markings are designed to be easily distinguished by
human eyes and visual spectrum cameras operate on the same wavelengths as human
eyes, it seems a justified assumption that visual spectrum cameras would be similarly
capable of distinguishing the lane marking.

Secondly, it is assumed that the characteristics defining the lane marking are
independent of the scale of the sample. In other words, the features by which a lane
marking can be distinguished should be present in a longer or shorter sampling of the
marking and also present in a near or distance sample. Several examples in the literature
discuss this notion and opt for features like edges or a parametric model describing the
lane’s curvature.

The last general assumption is how the model should interpret the sample. The
overall objective is to detect the lane marking at different points in time. The sequence of
frames matches with a contiguous time domain. However, it is assumed that the domain
of time does not influence the appearance of the lane marking but rather that the lane marking appears to change in time due to the sampling of a different portion of it. More basically, as time progresses, each sample is not of the same object but only of the same type of object being influenced by a common noise per video. This carries two implications: that different sections of the marking should exhibit the same characteristics (which is an assumption we’ve already made) and that the influence of noise is the only factor affected by time. The latter is often referred to as domain adaptation in the machine learning literature. Domain adaptation can be thought of as training in one scenario but testing and learning in another. The temporal aspect of noise is idealized here to be influential on at most frequently a per-day basis. This is vastly longer than the sampling rate of a standard visual spectrum camera and even the update rate of a deep neural network so the influence is assumed to be on a per-video basis. Thus the model is trained on a range of scenarios but only learns on sequential temporal samples: because any order of sample learning in the video is sufficient and the natural one lends itself most readily. In all, the model should learn a general sense of the appearance of the markings while adapting to new conditions encountered along the way. The performance of a learning model is highly dependent on these assumptions.

**The Data**

Another assumption regarding the data’s domain is implied in the sample set’s representativeness of the whole. It is traditionally assumed that from any data set, a random sampling should be sufficiently representative of the characteristics exhibited by the entirety of the data. The data used here, however, breaks this traditional assumption
by using a challengingly diverse subsample set exhibiting different characteristics and representing them disproportionately but maintaining those proportions among the common training set. In all, five videos were used as representative examples of the variety of situations an autonomous vehicle may face. Below are frames from the videos used which demonstrate the difference in both scenarios and the appearance of the lane marking itself.

![Figure 2](image)

*Figure 2. Demonstrating the five videos ranging plain conditions (a), glare (b), shadow (c), and snow (d,e).*

The number of frames in each video varies from 8 to 245. The actual proportions of data used from these videos is elaborated on in the beginning of chapter 5. For now it is sufficient to note that the data set was collected using a number of cameras with unknown parameters on unknown roads. The diversity among the data set itself provides an admirable challenge for any advanced driver assistance system. Thus it is a key assumption that a system which trains and tests on this data must be robust by nature.
The information provided by these videos is naturally represented in the form of image data; two dimensional discrete arrays on each channel illustrated in Figure 3 below.

![Image Plane](image.png)

*Figure 3. BGR image data representation*

An unsigned 8-bit integer represents the quantized measure relating to photonic energy at a given point in the sensor array’s domain. For each channel, the sensor measures photonic energy at a band of wavelengths centered on the named representation; a Gaussian centered at around 700 nm is the distribution captured by the typical “red” sensor. Larger values correspond to brighter light at that wavelength and in image representation, values closer to 255. The image plane is analogous to the configuration of the physical sensor array capturing the photonic energy and the different color sensors are physically situated in such a way to correspond accordingly per pixel. The vertical and horizontal size of an image is determined by the dimensions of the
sensor array. Thus, the natural representation as a data object is a set of two-dimensional arrays corresponding to the wavelength of the measured photonic signals blue, green, and red.

**Machine Learning Models**

The basic idea behind machine learning is to automatically tune a model to incrementally achieve better performance. Performance is a highly subjective term and is specified on a per-task basis. Achieving better performance may appear as minimizing erroneous behavior or as maximizing desired outcomes. A model is tuned incrementally by the process of optimization, which both measures the performance of the model and determines how to improve it. The object of much debate is how to go about determining what exactly a model is.

**Learning Models Overview**

A model may be thought of as a set of properties or behavioral characteristics describing some phenomenon or object. There is no formal definition of a model because it is such a general concept. Information, in these terms, is an unexpected variance which contributes a change to the model. In general, a model can be used to describe anything from thought patterns to sports statistics to reality itself. Learning models are a special class of models that don’t only describe a body of data but can also adapt to new information. Automating this change can be very difficult. Luckily, computer scientists, mathematicians, and engineers all have a vested interest in finding ways to accomplish it.
Introduction to Machine Learning Models

Machine learning (ML) models span every field of engineering and are finding uses in every niche. The ability of a machine learning model to capture features by analyzing a memory of past instances on-the-fly and producing new features makes them ideal for many research-oriented tasks from processor optimization to computer vision. ML models don’t have to keep the entire history of an input but instead build a function which optimally describes new inputs in terms of the desired aspects.

To understand artificial neural networks (ANN), it’s most intuitive to begin with the data-driven scheme. Imagine a lane detecting advanced driver assistance system (ADAS) which contains multiple models depending on different noise characteristics observed at different times. Perhaps one model is robust to snow, another to shadow, another to glare, and one is generally for normal roadway conditions. Each model has its own parameters and features. In order to switch from one model to another model, a metric must be computed which assesses how well each model performs on the data. Another metric must then be computed which compares those metrics in order to select the best model. Figure 4 provides an illustration of such a system.
Streamlining the process may first look like combining the computation of each model’s performance with the computation selecting the best model. Now the performance of all models are considered simultaneously and used directly to determine the system’s output. This streamlined system is shown in Figure 5 below.

Figure 4. Hypothesized ADAS lane detector original model

Figure 5. Streamlined hypothesized ADAS lane detector model
Breaking down the models into their fundamental elements, the generation of features and decision making may be separated. Because the decision making framework is also a mathematical computation based on the features generated from the input, it may be combined with the model selection step. Now there are only two stages in the ADAS: feature generation and the simultaneous model selection / decision making computation. Generalizing the decision-making process allows us to combine the individual decision making from each process. The generalization of the decision-making process can be seen as support vector machines (SVM), perceptrons, and other simple networks. It is an extremely common research approach to experiment with a multitude of features by plugging them into a simple network and optimizing toward a desired performance. Because the features are assumed beforehand and are unable to be enhanced, I consider the ADAS up to and including this stage to be a data-driven approach. The system is again illustrated in Figure 6 below.

*Figure 6. Two-Stage hypothesized ADAS lane detector model*
In order to further generalize the ADAS we must find a way to combine the feature generation and the feature subset selection stages. To be justified later in the next few sections, I will simply state that a common architecture is used for both of these stages and that extending the decision-making framework allows us to automatically derive combinations of the input which approximate the features we would have otherwise used. Many variations exist for finding these approximated features which enhance performance or make assumptions for specific tasks but in general, the structure of the decision-making framework is a network. More formally, a network is a graph which receives input at input nodes, communicates the signal across weighted paths to other nodes, and propagates toward an output. Treating the network as a directed graph which inputs the observations and outputs the detection makes our ADAS a neural network. Shown below is the last step in the transformation from a data-driven scheme to a model-driven scheme, ending in a neural network.

*Figure 7.* Neural Network realization of a mode-switching ADAS lane detector.
The input is fed to the first layer of representative nodes which capture multiple aspects of the data. As information propagates through the network, the class definitions are more distinguished in deeper nodes until a result is produced. The truly astounding aspect of this idealization of a neural network is that it may be reversed: features derived in the network can be approximated by ones we have already optimized computationally and the system can be fragmented for efficiency at a detriment to robustness.

Discussion on Features

A feature is any function of the data. Consider a feature as a way to examine some aspect of the data. Naturally, the realm of applicable features widens with each conceivable task. As we’ve seen in many fields, the combinations of features to affect in even the simplest signals has practically no upper bound. It also mathematically has no upper bound as the number of features of a non-zero dataset is infinite by definition.

Often, features fall into three distinct types: data range features, data domain features, and semantic features. Data domain features are functions of both the range and domain. Semantic features can be thought of as any feature that is more complex than a domain feature. As functions of domain features (and thus functions of the range and the data itself), semantic features capture the higher-level ideas behind analysis.

In computer vision, spectral transformations are the data range features of choice. Color space transforms from the inherent red, green, and blue (RGB) range to Euclidean spaces, like the La*b* space, frequent the literature. A very relatable color space is the hue, saturation, and value (HSV) space because each feature individually represents
something most people can identify. An important characteristic of these types of transforms is that they are reversible, meaning a reverse transformation also exists.

Textural is the go-to domain feature of computer vision. Textural features span from co-occurrence matrices to adaptive filters. Edge detectors are textural features because they denote a divergence from the usual texture. Semantic features in computer vision span the many human ideas we can’t quite write an equation to describe. Any metric you put to things like “snowy” or “shiny” or “matte” can be considered semantic features. When constructing a model representing the appearance of a car, a semantic feature may require that two or more wheels be visible, a windshield, and headlights. While equations denoting the “two-or-more-wheeledness” of an object may not be forthcoming, it is obvious that we ourselves can make that judgement intuitively. Nonetheless, if we are able to deduce from the data that two wheels are visible, and further if we can make an algorithm that determines if it is seeing a wheel, then there must be some function of the data that provides us with this information specifically. Thus semantic features must also be functions of the data, and likewise by definition functions of domain and range features as well. These feature types are not strictly defined but instead describe general levels of feature relationships in terms of the data’s structure.

*Universal Approximation Theorem*

Perhaps the strongest theoretical justification for the use of neural networks is the universal approximation theorem (UAT). Found in the master’s thesis of [43], the theorem states that one hidden layer of a feed-forward neural network with a continuous,
bounded, non-constant, monotonically increasing activation function can approximate any continuous function; including those compact in finite-dimensional space. The activation of a hidden layer unit is calculated as a function of the weighted sum of the activations feeding into it.

\[ a^{(l+1)}_i = f \left( \sum_{j=1}^{S_l} w_{i,j}^{(l)} \cdot a_j^{(l)} \right) \]  

The number identifying each layer is \( l \), the number of elements in \( l \) is denoted as \( S_l \), and the weight matrix at layer \( l \) is \( W^{(l)} \). The function \( f \) is referred to as the activation function and is typically a continuous, monotonically increasing function which limits the value of the output to between 0 and 1 for compact computational representation. Sigmoids, hyperbolic tangents, and piecewise linear functions are often chosen for this task. While many authors include the addition of a bias term, it may be supposed here that a bias is arranged by using a value of one as the last element in \( \tilde{a}^{(l)} \). When \( l = 1 \), the layer is denoted as being the input. The UAT states that a linear combination of activations of a finite set of second-layer nodes can approximate any continuous function. In other words, any activation after the second layer can be an approximation of any possible continuous feature of the data. It is important to note that not all things that can happen do happen; there is absolutely no guarantee of approximating the best features in a global sense. Because the feature space is nonlinear, the optimization algorithm may converge onto a local maximum or minimum.
Optimization

Optimization is the iterative process of innovating a feature set toward incrementally better performance of a task. Optimizing involves three steps: evaluation, selection, and advancement. An objective function is a mathematical equation that assesses the performance of a model. Naturally, in the evaluation stage the objective function is utilized to provide a hard analysis of the model given the data. Typically the objective function’s gradient is known on all features, so the performance of the model given a shift this way or that on each feature can be determined. During the selection stage, the various changes of each feature are examined and a set is selected which will most beneficially change the model. In the last stage, that change is applied and the model is advanced in such a way that it performs objectively better at the task. While this seems like quite a process, much of it can be automated rather easily.

Because the objective performance of a feature set can be highly heuristic and convergence can be approached in many ways, both the original selection of an objective function and the computation of the objective function benefit from great flexibility. Consider the objective function for autoencoders which includes an error term.

\[
J(W, X) = h_{error} = \left[ \sum_{i=1}^{S_1} (\hat{x}^{(i)} - \tilde{x}^{(i)})^2 \right]^{\frac{1}{2}}
\]  

(2)

where \(\tilde{x}\) is a reconstructed version of \(\overline{x}\) using parameter set \(W\). The autoencoder reconstructs the input and the \(L_2\) norm is used here to originally capture the error between the actual input and the reconstructed input for minimization. Because a monotonically increasing function applied to an original function converges as the original function converges, we may apply any monotonically increasing function to the
error term. Here we can use the square because the argument of the square root is guaranteed to be non-negative and the non-negative domain of the squaring function (and only that portion) is monotonically increasing.

\[ h_{\text{error}} = \sum_{i=1}^{S_1} \left( \hat{x}^{(i)} - \tilde{x}^{(i)} \right)^2 \]  \hspace{1cm} (3)

Multiplying by a positive constant is also allowed because the linear function with a positive slope is most certainly monotonically increasing.

\[ h_{\text{error}} = \frac{1}{S_1} \sum_{i=1}^{S_1} \left( \hat{x}^{(i)} - \tilde{x}^{(i)} \right)^2 \]  \hspace{1cm} (4)

Now, given a batch of examples on which to train, the error term may be changed to average results over the batch.

\[ h_{\text{error}} = \frac{1}{m} \sum_{j=1}^{m} \left[ \frac{1}{S_1} \sum_{i=1}^{S_1} \left( \hat{x}^{(i)} - \tilde{x}^{(i)} \right)^2 \right] \]  \hspace{1cm} (5)

The number of samples in the batch is denoted as \( m \). Rearranging the terms in this equation and multiplying by \( m \) gives us the following equation.

\[ h_{\text{error}} = \frac{1}{S_1} \sum_{i=1}^{S_1} \sum_{j=1}^{m} \left( \hat{x}^{(i)} - \tilde{x}^{(i)} \right)^2 \]  \hspace{1cm} (6)

Multiplying again by a positive constant, we can obtain an equation that is easily differentiable but still converges with the original formulation.

\[ h_{\text{error}} = \frac{1}{2S_1} \sum_{i=1}^{S_1} \sum_{j=1}^{m} \left( \hat{x}^{(i)} - \tilde{x}^{(i)} \right)^2 \]  \hspace{1cm} (7)

Likewise, other functions can be similarly manipulated into functions that are more efficient to compute while maintaining the original purpose. With an objective function that efficiently evaluates the model’s performance in terms of error, we must select a minimizing framework with which to optimize the model.
Algorithms that train the model by minimization are referred to as trainers in many software packages. While popular trainers include the stochastic gradient decent method [44], Nesterov’s accelerated gradient [45], and ADADELTA [46], the RMSProp [47] algorithm is used in this work. What’s necessary to understand about gradient descent algorithms is that each estimates the gradient with respect to the parameters of the model and gradually changes the value of those parameters in the direction which minimizes the model error. Illustrated below is the general idea behind optimization; specifically minimization.

The idea is that for a given set of data and associated target value(s), the objective function is computed with one realization of the weight values. These parameters of the system comprise the state at epoch 1. The objective function is then evaluated at different combinations of weight values using the same data and target value(s). The weight values

Figure 8. Optimization through minimization overview
producing a nominally better objective are selected for assessment at the next epoch. The figure above shows how one weight is manipulated from epoch 1 to epoch 5 such that the objective function is minimized each step. In general, the objective function is assumed to be a smooth manifold on the weight matrix (or matrices when there are more than two layers) and so the gradient of the objective function with respect to each weight is assessed each epoch to discern how exactly the system parameters should be tuned to better fit the output given the data to the target value(s).

A coefficient called the learning rate is applied to scale the gradient and acts as a measure of how far the parameters are influenced at each iteration. More literally, the new value of a neural network’s weights are computed based on both the current gradient of the objective function with respect to each weight and also a history of those gradients on each weight. The learning rate is the degree to which the gradient influences the new weight values. In this way, the optimization of an autoencoder evaluates its performance, selects an optimal adjustment, and advances the model toward an “objectively better” goal.

Statistical Background

Statistics is the language describing the logic of dealing with uncertainty. In order to analyze trends in the randomness of collected information, one must suppose the influence of unknown variables. The information sought in the data is relative to a hypothesis concerning the interaction of hidden variables. In other words, one must start with a hypothesis, review aspects of the data that may provide information about addressing the hypothesis, and then interpret what that information means. The process of
regression was developed to fit data to an equation. In the realm of modeling and
simulation, both regression and information theory are employed to provide a framework
for defining hypotheses based on what information can be obtained and interpreting data
toward addressing those hypotheses. When considering the probability of the data
associating with a given distribution, information is a measure of how informative that
data is in addressing the hypothesis. Typically measured in entropy, information provides
a way of deducing what part of the data is likely to be most useful and what parts can be
disregarded based on a less resource-intensive calculation. In modeling and simulation,
the entropy provides a sort of priority metric for what among the data to compute first in
case time or other resources are limited. When considering the probability of how
informative a feature is to the model, entropy provides a metric for how well the feature
distinguishes trends in the data. In reference to using non-negative sparse autoencoders
for community detection (a clustering problem), [48] says “[t]he non-zero entries of [the
derived non-orthogonal, positive-valued basis vectors] are then taken as good candidates
for network communities.” This roughly translates to the idea that because the
autoencoder’s hidden layer features found information, some grouping of the data should
be present which produced that information. The power of information theory in this
setting can help to address even the deepest questions in computer vision like how to
derive depth from a two-dimensional image. [49] uses deep autoencoders and Restricted
Boltzmann Machines to reconstruct a three-dimensional image from only a two-
dimensional projection. While this seems a figment of science fiction, it calls into
question the types of associations being captured in these trends. The age-old adage
“correlation does not imply causation” comes to mind as there, by definition, cannot be depth information provided in only a two-dimensional projection. However, even if the autoencoder provides only a set of assumptions for which to guide analysis, the set it provides is still orders of magnitude more refined and informed than a human can consider in years of research; the sheer volume of data a computer can work through rivals the focused capacity and precision of the human brain.

Robustness

Robustness is typically thought of as good performance over a wide variety of inputs. A diverse input is a challenging landscape when characterizing a model. The pursuit of robustness may seem to conflict with the interest of finding descriptive features. However, the terrain becomes more traversable when characteristics of the noise are better identified. The advantage of adaptive and learning models is in collecting information not only of the signal but also of the noise. When the noise is able to be characterized, features defining the model are more forthcoming and the diversity of input the model is able to handle increases.

In the data-driven scheme, robustness can be accomplished in feature selection, outlier filtering, and template deformation. As the feature selection stage is so highly supervised, it become most important for the researcher to have an intuitive understanding of the behavior of the noise given a particular signal. Often the influence of the noise itself is treated as a signal and characterized by a set of features on an input which represents the true signal. A model is made robust by selecting features that behave counter to the noise features; exhibiting opposing responses or having very small
covariance between the two sets. Ideally, if the feature set of the noise is an orthonormal vector, the feature set chosen for the signal would be another orthonormal vector which lies orthogonal to that of the noise. A more statistical approach to robustness is seen in outlier removal. Given that a class may be defined by a set of features, observations of the class can be thought of in terms of a cluster in feature space. When given a noisy set of observations, robustness may mean refining the definition of the class by specifically rejecting the characteristics of outliers on the common set of features. While the features may not change, the combinations of features that lead to misclassifications are exiled from the definition used by the model. Extending this idea, a template is a means of defining the desired feature values and may incorporate robustness by allowing for some flexibility. A deformable template often keeps a base-line set of expected values while incorporating changes based on data gathered from new observations. Frequently a Kalman update is used to incorporate new information into a deformable template. As a wider context (more information) is considered in refining a model’s definition, robustness seems to take on different forms.

A model-driven scheme typically approaches robustness via noise injection, in the optimization structure, or by only partially defining the class. Noise injection is by far the most popular method as it is the most intuitive. Injecting noise into the input during preprocessing, in optimization during processing, or at the target value while postprocessing all contribute to a model’s robustness. [50]–[54] each use a type of noise injection. Denoising autoencoders are famous for injecting noise into the input while expecting a reconstruction of the original signal. [50] described them as “robust to the
input.” [52] found that “various levels of noise yield different features, which are more global for larger [amounts of noise corruption].” [54] experimented longitudinally with injecting several different types of noise into the input. Noise injecting has become so popular that the process has been optimized in several research-oriented software libraries. While noise injection is all the rage, [50], [51], [55]–[59] report utilizing the optimization structure itself as a means to achieve robustness. The optimization structure, consisting of an objective function, a graph structure, an activation function, and even hyperparameters offers ample opportunity for robustness. Contractive autoencoders include a term in the objective function which enforces the rule that learned features should be invariant to what isn’t observed in the training set [50]. Also utilizing the objective function, Boltzman machines may include the assumption of a Gaussian variance in their so called energy function [55]. [51] leverages the objective function to enforce a coherence of denoising autoencoder features between the original system and the system when adapted to a different domain. Similarly, [58] found that “the degree of invariance of the representations grows towards the higher levels in all tested models but much faster with modulated lateral connections” when using a term linking the weights of encoding layers with those of decoding layers. In the objective function of a decision tree, [56] employs soft decision-making criteria to remove outliers for robustness. [57] explores rectified linear activation functions which lose information for robust effect. Lastly, [59] uses feature pruning for robustness as a directed, supervised learning step in postprocessing. While the optimization structure has a lot more to offer, partial definitions seem to stand on their own. Partially defining a class ignores some aspects of
the class which may contribute characteristics less robust to noise. While originally intended to provide the capability to use incomplete data, [60]–[64] use partial definition to achieve robustness. Typically either the input is lossy and the model is made to reconstruct it, or data is pooled in some manner during the processing and details are lost. [60] and [63] reconstruct the missing input itself and then proceed to learn. In particular, [63] noted about the feature space that “[the autoencoder] produces a point over that manifold that is the closest to the known values of the incomplete input vector.” While the model is not better informed from the missing input, it is still partially informed by the present input. [61] and [62] prefer to reject information in the input wholesale by using a windowing or binning method respectively. [64], more directly, labels only part of the samples and allows the model to label and learn from the rest on its own. It’s worth noting that this approach may be favored in larger datasets when providing thousands of labels may not be cost effective. Robustness in any form has tradeoffs but in general, having more information is better; even if it means having more to reject.

Model Types

While a variety of network structure and optimization configurations exist, three major groupings lend themselves: regressors, autoencoders, and deep networks. The distinction between them does not depend on the choice of training algorithm but rather on the objective itself. Regressors compute the objective function based on reconstructing a desired real number. The target of the regressor is given as an input to use when training the model. Because training a regressor requires a target value, the training is said to be supervised. When using a regressor in a learning model, a target must be
derived from the data before the model can be retrained. Autoencoders are said to be unsupervised because the target they use is the original input; which does not require any prior labeling of the data. Autoencoders are thought to define a feature set which expresses the natural trends in the data because the objective does not include synthetic target values. The last and most interesting type of model here is the deep network. Deep networks can have any target value but are characterized by having many layers which are thought to build a more abstract, semantic representation of the data. A general trick with neural networks is to boost efficiency by using fewer nodes but stacking them more deeply. As a signal propagates through the network, information is typically captured in the first few layers and reused in the depth. It’s worth noting that these three model types often overlap and no exclusivity is implied.

**How Data is Input**

Perhaps the least trivial aspect of the model is how data input is handled. Data is collected as image patches along a scanline in each frame. Figure 9 below illustrates this.

*Figure 9. Image patches are extracted and the data is flattened into arrays.*
Having been flattened, the arrays are then normalized individually by first dividing by 3 standard deviations, shifting the minimum value to zero and shrinking the variance to 1. The process is outlined in flow diagram below in Figure 10.

\[
\begin{align*}
\{ (900,1) \} &= \bar{x} \\
\bar{x}_1 &= \frac{\bar{x}}{3 \cdot E[(\bar{x} - E[\bar{x}])^2]} \\
\bar{x}_2 &= \bar{x}_1 - \min(x_1) \\
\bar{x}' &= \frac{\bar{x}_2}{\max(x_2)} \\
\end{align*}
\]

Starts off on the range [0,255] as unsigned 8-bit integer. (uint8)

Ends up on the range [0,1] as a signed 32-bit floating point number. (float32)

*Figure 10.* Flow diagram demonstrating normalization of each flattened window.

In the flow diagram above, the general concept is to divide by 3 standard deviations and then fit to the range [0,1]. By necessity there are checks for dividing by zero as uniform data has zero variance and zero data has a maximum of zero. In Figure 11, an illustration is provided which demonstrates what happens after the data is normalized. The normalized data is concatenated into one array and elements of this array are used as input directly to the neural network.
Figure 11. Processing of data through normalization and into the neural network.

While this process demonstrates how individual data samples are fed into the neural network structure, it is also important to understand the larger scheme. Before a model can be used, it must be pretrained. A dataset which includes patches randomly taken at various horizontal scanlines in each video was aggregated as the training set. At this point it’s worth noting that 5 videos were used: one representing a sunny day, one with glare, one with shadow, and two with varying amounts of snow. The videos were captured using a variety of unknown cameras with varying intrinsic and extrinsic parameters. The videos are relatively stable but meet the requirements of the general assumptions. In order to obtain samples, I iterated through frames in the videos and selected windows randomly which contained the lane marking. The vertical position of the selected window was used to draw a horizontal scanline and every window along the scanline was labeled as a sample set. Only the window which was selected was labeled with a target value of 1 while all other samples had target values of zero. Figure 12 illustrates this.
Figure 12. The selected window is labeled a 1 while others are labeled a 0.

While it may not be apparent in the illustration, a sliding windowing method was used and there is much overlap in the data set. Each window was 30 pixels high and 30 pixels wide for a total of 900 pixels per channel. Models labeled “Color” were trained on a fully concatenated array of 2700 pixels while “Grayscale” models were trained on only the intensity for a total of 900 inputs. The OpenCV library as used to perform this color transformation in the Python programming language via the function cvtColor with an argument CV_BGR2GRAY. The transformation was linear as depicted with the following equation.

\[ Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \]  (8)
Y represents the grayscale value and the transformation is performed at every point in the image domain. The same training set was used for both the color and grayscale models so that results would be comparable.

This selection method was employed for many reasons, namely because samples varied in size, did not assume any order, and exhibited robustness. As discussed earlier, the area along the image plane decreases as objects become more distant from the camera. While modern methods use IPM, this work extends the classical approach and expects robust results without it. Maintaining invariance to the scale in a training set causes the model to learn features that don’t depend on assumptions of scale. This is doubly important because as features become more distant, the scale along the window itself becomes more nonlinear. In other words, if your window is nearly at the horizon, the scale at the bottom of your window is dramatically different than the scale at the top. Another important justification for this type of selection process is statistical in nature. Because the model will be asked to regress along windows on a scanline, it makes sense to train it using a similar probability of encountering the lane marking. Support for this idea can be seen in [65], where a modified Overfeat convolutional neural network is used to detect both vehicles and road lane marking using data that includes depth information as entire context windows in the image frames are considered at a time but only the center portion is chosen to represent the detection. Furthermore, by compiling the samples in no particular order to train the model, no assumption of order is made. Neural networks make no assumption of order in training, which is part of the reason why batch computation can be used; increasing the efficiency and speed of the training and testing
processes. In order to ensure that the batches are consistent, data from 100 scanlines are aggregated into one set and models are trained in batches. Robustness is the main driving factor for aggregating the testing samples among so many diverse weather conditions and scales. Incorporating robustness into the training set yields a better chance of learning features which are robust. Incorporating these characteristics into the model allows for a preliminary notion that these types of noise may become important. This leads to the methodology of testing.

Domain adaptation is the idea that a learning model changes to confront the challenges of a new domain, having been trained and learned features pertaining to another context. In this work, the models are trained on a variety of scenarios ranging from snow and glare to shadow and clear weather but are tested and learn in only one of these domains. For each video, scanlines were selected as a basis for sufficient comparison between models. Only the windows along each scanline were considered by the model. Before each scanline was used in testing, the model was reset with the original weights obtained from training. In this way, the model was trained weakly on a variety of domains but was only able to learn in one. After testing on a scanline, the model parameters were saved before being reset. In this way, domain adaptation was leveraged to first build a feature set catering robustly to many scenarios and then adapt to one specifically.

Having discussed the training of each model, a comparison with the testing is shown in Figure 13. In an almost opposite manner, the training begins with the selection of a window, the capturing of scanline windows, and the labeling whereas the testing
begins with the capturing of scanline windows, the selection of a positive sample and finally the labeling.

Figure 13. Model training and testing comparison

In testing, the windows of an entire scanline were aggregated, the model regressed a value for each window, the dataset was labeled, and the model was retrained by the new values. To clarify, the result of regression at points along the scanline were used to label the dataset with target values. The target values were assumed zero everywhere and the point at the maximum of regressed values along the scanline was set to a one. This produced a similar dataset as during the training stage except that instead of being selected by hand, the model regression values select the lane marking. Even though no assumption of order was made about which samples to test at what time, the general idea of this type of system is to run in real-time. Samples from one frame were used to retrain the model in preparation for regression of data in the subsequent frame. The effect this had was not to enforce some temporal order but rather to learn the new domain of each
type of noise more specifically. Because the software was written to run automatically
with each model, I refer to the process of training and testing in this manner hereafter as
model deployment. The models themselves are discussed next.

Regressors

The objective of a regressor is to match a target real number. The regressors used
in this work consist of only two layers: an input layer and an output layer. This type of
regressor is intended to reconstruct the target via a linear combination of the inputs
directly. The implication expressed is that the data is linearly separable. When used for
classification, the model is called a perceptron. A perceptron builds a single hyperplane
as the decision boundary between two classes. Without sufficient separation, the
perceptron exhibits remarkably bad performance.

Network Organization

The structure of the class of regressors used is shown in Figure 14 below as a two-
layer model consisting of only an input and an output layer. The entire model is defined
by only the aspects of the optimization.

Figure 14. The structure of a perceptron with input nodes x and output value a.
No real features were able to be trained in this structure because the only activation value was taken as the output itself. Only the influence of the weights on the inputs was affected during training. The variety among regressor models is in the input and activation function. Color models used an input of 2700 nodes while grayscale models used only 900 input nodes. In both, the output was only a single node.

This network structure has a very strong link to the highly esteemed convolutional filter. The idea behind a convolutional filter is to multiply pointwise between a window in the image and a filter of the same dimensions. When designing a filter, a desired filter response is known and this value is analogous to the target value. The purpose for selecting target values of either 0 or 1 is for this very reason: a filter with a sharp response only when given the right input is considered ideal. By enforcing that the “filter” should exhibit an impulse only when given the lane marking and reject all other inputs, I am tailoring my model toward exhibiting the best possible response that a filter can achieve. Figure 15 illustrates the similarities between my choice of regressors and a convolutional filter.

*Figure 15. Relation between convolution and specific class of Regressors*
Two things that are worth noting here are the activation of the output layer and the order of coefficients. In convolution, the filter is flipped about the x and y axes before being multiplied pointwise and summed. For this reason, the data at (0,1) is multiplied by the coefficient at (3,2) instead of being multiplied by the one at (0,1) in the filter. Were this a correlation filter, the later would be the case. This is no problem as we can show proper comparison by equating the weight at (1,1) with the filter coefficient at (3,3), the weight at (1,2) with the filter coefficient at (3,2) and so on. The weights are not treated any differently from each other during the learning process so the only time order would matter is if you wished to reconstruct the filter found via regressor into two dimensions. To do this, it would seem straightforward to first index the weights as the input is ordered and then flip them about the x and y axes. The other aspect to consider is the activation function. While many activation functions exist, only the linear activation function produces the result shown above.

There is more to be said of the choice in activation function as well. Convolutional filters are often designed such that the sum of all values in the filter is 1. The idea is that by summing to 1, the filter provides nearly a weighted sum and does not affect the maximum or minimum of the original data. An activation function in my regressors receives the weighted sum of the pixel values and can have the effect of normalizing that value to a given range depending on which function is used. So while the structure of the regressors I use doesn’t exactly produce a convolutional filter, it can be said that it produces a filter that acts analogously but includes the ability to learn new
values and optimally fits the data. In fact, the reason I first explored neural networks was by looking for a way to design an adaptive filter using as much of the data as possible.

**Optimization Types**

Because the RMSProp algorithm was used to optimize in all models, the other cause of variety among the regressors was the choice of activation function. Linear, rectified linear, thresholded rectified linear, sigmoid, hyperbolic tangential, and softmax activation functions were compared. In the following, all of the activation functions are represented as $g(x)$ to signify that they are interchangeable in the models but not that they are equal to each other. Linear activation means that the input was directly taken as the output. As an equation, the activation function is the linear function with slope 1 and intercept 0.

$$g(x) = x$$

(9)

Rectifying a linear activation (Relu – Rectified Linear Unit) sets any non-positive input to zero while maintaining the linear activation for positive inputs.

$$g(x) = \begin{cases} x & \text{for } x > 0 \\ 0 & \text{for } x \leq 0 \end{cases}$$

(10)

A noisy Relu injects noise into the output when the input is positive and a leaky Relu outputs a nonzero constant when $x$ is non-positive. A thresholded Relu only linearly passes through values that are above 1.

$$g(x) = \begin{cases} x & \text{for } x > 1 \\ 0 & \text{for } x \leq 1 \end{cases}$$

(11)

The thresholded Relu was among those tested in the regressors. The next activation function tested was the sigmoid (also called logistic), for which
\[ g(x) = \frac{1}{(1 + e^{-x})} \]  

(12)

The sigmoid is perhaps the most famous of activation functions and the equation itself is useful in normalizing probabilities as a partition function. The sigmoid has the desirable quality of mapping any input to an output on the range \((0,1)\). Similarly, the hyperbolic tangent (tanh) maps inputs to the range \((-1,1)\).

\[ g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \]  

(13)

While more computationally intensive, maintaining negative values may aid in defining a wider variety of features. Lastly, softmax activation considers all of the activations in the layer and normalizes accordingly so that the entire layer’s activations sum to 1 and each individual activation is on the range \([0,1]\).

\[ g(x) = \frac{e^{x_j}}{\sum_{i=1}^{S_1} e^{x_i}} \]  

(14)

In the softmax activation equation, \(j\) denotes the number of the node within layer \(l\). In the case of a two-layer regressor, the softmax output always yields a value of 1. The other aspect of optimization with a regressor is the objective function. The objective function is fairly straight-forward. The \(L_2\) norm of the difference between the target value and the reconstructed value is a good measure of error. Progressing in a similar manner to the example earlier, the objective function is arrived at below.

\[ J_R(\tilde{x}, y, \bar{w}) = \left[ g\left( \sum_{i=1}^{S_1} w_i \cdot x_i \right) - y \right]^2 \]  

(15)

As before, the function \(g\) is the activation function, the vector \(\bar{w}\) is the vector of weights, \(\tilde{x}\) is the input data, and \(y\) is the target value. The number of input units, \(S_1\), is 2700 for color patches and only 900 for grayscale patches.
Model Assumptions

The only assumption specific to this model is that the weights are likely to define a hyperplane which does its best to approximate the target value. This roughly translates to the idea that the pixels with larger weights have greater influence in defining the class. In grayscale models, these pixels can be thought of as points in the domain which make the largest contribution to the class definition. In color models, they can be idealized to represent the aspects of color that are most indicative of a lane marking. While this does not provide much indication for semantic features, the weights computed this way can help to indicate some range and domain aspects of the class. Beyond this, nothing else is assumed regarding only the regressors.

Another important aspect of training the networks is the time constraint. A constraint was put onto the iterative learning such that during training the networks were allowed to iterate 500 times but during testing only 100 times. This was meant to capture the idea that considerably more time would be afforded when initializing the lane detector than when actively using it. The same scheme is used throughout all networks.

AutoEncoders

Originally intended to aid regressors in finding a more global optimal fit, autoencoders have changed the machine learning landscape. Features of autoencoders are thought to portray the natural trends exhibited in the data without having to provide a single label. Applications using autoencoders range from clustering and data mining to domain adaptation and data restoration. This work contributes one more use: learning.
The autoencoders presented are trained on-the-fly to keep up with the new domain. The graphic below explains the usual use of autoencoders.

**Figure 16. Normal use of Autoencoders**

During the testing phase, the network which incorporates the autoencoder layer is treated as a deep regressor and the features in the hidden autoencoder layer are overwritten in the learning process. In an attempt to preserve the benefit found in the autoencoder’s ability to capture features that are naturally found in the data, I’ve extended the training process into the testing phase. I call this a learning autoencoder and Figure 17 provides a walkthrough of what is meant by this.

**Figure 17. The dynamic, learning autoencoder process.**
The learning process is a very fluid optimization that incrementally tunes model parameters in a given direction toward an objective. As long as both models are updated at the same time, the objective is still met but the features learned using the autoencoder remain separate from the influence of the target. This allows them the ability to adapt to a new domain and aid in convergence toward a new global optimum. In the work of [66], autoencoders are trained on different domains and the features are bottlenecked together to form a layer of features common to both domains. While not exactly the same, my method adjusts one autoencoder instead of combining separate ones.

**Network Organization**

The structure of the autoencoder is fairly straightforward. A layer of input nodes is followed by a layer of hidden nodes and a layer of output nodes. This is illustrated in Figure 18 below.

*Figure 18. Structure of an autoencoder*
The number of input and output nodes are the same because the input is being reconstructed and compared one-to-one to the output. Conceptually each output node is reconstructing a real number which happens to consistently be one of the input values. An autoencoder is a multi-output regressor with three layers which considers the inputs to be the target values. The number of nodes in the hidden layer is a selection the engineer must make. More nodes in the hidden layer than the input can mean that the hidden layer doesn’t learn features but simply passes the values directly through. This is called overcompleteness and occurs because the autoencoder has learned the identity function. Having the same number of nodes in the hidden layer as the input is called complete and can function in the same way. When the weights of an autoencoder are tied, it means the encoding weights (weights between the input and hidden layers) are the same as the decoding weights (weights between the hidden and output layers). Mathematically, the weights are transformation matrices from one layer to the next so the decoding weights would be the transpose of the encoding weights. Nonetheless, when the hidden layer has the same number of nodes as the input layer and the weights are tied, the autoencoder can perform principal component analysis. Of course, the activation function must be linear and the cost function must be the $L_2$ norm and enforce orthogonality among the weight vectors, but it is possible.

The choice of how many units were in the hidden layer can be considered another degree of freedom along which to experiment. A term in the objective function called sparsity was considered but was decided against given some recently published research. Sparsity is the concept that information should be corralled into as few nodes as desired.
Typically a sparsity term is considered in the objective function which employs the Kullback-Leibler divergence. The function punishes weights that are both too large and too small; punishing nodes that are too large more severely. [67] suggests that when using this sparsity term, the bias values (weights of the input which is set to a value of 1) become large and can impede learning with high-dimensional data. All things considered, treating every pixel as an input, images can be considered high-dimensional data.

Furthermore, [68] recommends using a smaller hidden layer instead because the benefit is nearly the same and the training is much faster. So in this work, three sizes are tested: 100 nodes, 500 nodes, and 1000 nodes in the hidden layer. It’s worth noting that the color models used an input of size 2700 while the grayscale models used only 900 inputs and that for grayscale models with 1000 nodes, there is a risk of overcompleteness.

**Optimization Types**

The objective function of the autoencoder is kept separate from the objective function of the regressor. Because the regressors used for the set of autoencoders are exactly the same as the ones presented already, I will only provide the objective function used by the autoencoder.

\[
J_{AE}(X, W^{(1)}, W^{(2)}) = \frac{1}{ms_1} \sum_{i=1}^{s_1} \sum_{j=1}^{m} \left( \hat{x}_j^{(i)} - x_j^{(i)} \right)^2
\]  

(16)

where

\[
\hat{x}_j^{(i)} = g \left( \sum_{r=1}^{s_2} w_{j,r}^{(2)} \cdot g \left( \sum_{q=1}^{s_1} w_{r,q}^{(1)} \cdot x_r^{(i)} \right) \right)
\]  

(17)

Note that the reconstructed \( \hat{x}_j^{(i)} \) is the result of propagating through the network.

Also note that the activation function is the same for both layers. While not strictly
necessary, it makes computation more straight-forward by utilizing the same code. The activation functions used here were selected as the best the regressors could offer: the linear, rectified linear, and hyperbolic tangential activation functions. The idea is that, all else held constant, the only degree of freedom with the regressors would be the activation function. So those activations that performed better may be better suited for the task at hand. More explanation for the selection of activation functions is in the analysis section.

Model Assumptions

The variety of autoencoders lies in both the choice of activation function and the number of hidden units. While the activation functions were limited to linear, rectified linear, and hyperbolic tangential, the number of hidden units varied from 100 to 500 and 1000. Together with both color and grayscale data, there were a total of 18 autoencoder combinations tested. Every experiment experienced the same structure of training and testing. No other assumptions were made regarding the autoencoders.

Deep Networks

The last class of machine learning models explored here are the deep networks. Depth is the quality of a network which describes the number of layers. Deep features are found in deeper layers of the network and are thought to represent more abstract concepts as the equations they could approximate become more complex. There is also great flexibility in the design of deep networks as each layer may be trained in different ways, on different data, using different activation functions, and furthermore different objective functions altogether. Google’s work has recently focused on network depth and with the introduction of Deep Dream has granted free, public access to the imagination of one of
its models. In the literature, a method called Adaboost (Adaptive Boosting) has been popularized for its ability to generate strong classifiers using weak classifiers [69]. The secret to Adaboost is in the structure: poorly classifying features are stacked deeply until a synergy is found and the combination of these comprise a well classifying feature. The deep networks in this work, however, include only deep regressors and deep autoencoders. Both deep networks are given the adjective ‘stacked’ because the same method for training their two and three layer counterparts respectively is the same in deeper layers. Deep regressors propagate the input signal through the network and compare the result to the target, changing the weights accordingly. Deep autoencoders, on the other hand, train each layer using the autoencoder optimization function by imagining a new set of output nodes with accompanying weights until the last layer which is granted a regressor. The learning autoencoder method is used for the deep networks presented as well.

**Network Organization**

More flexibility is found in deep networks because the dimension of depth is added to the other degrees of freedom. As each layer can have its own unique set of properties and even the data can be changed between layers, no firm structure can truly be presented to represent all deep networks. The deep regressor and the deep autoencoder in this thesis, however, are shown in Figures 19 and 20 below.
Figure 19. Deep autoencoder with a regressor to be used like the regular Autoencoders.

Figure 20. Deep regressor with orange hidden units.
As shown in the figures, each network has 3 hidden layers for a total of 5 layers. Each hidden layer has 50 nodes in it. These numbers were chosen arbitrarily based on the idea that fewer nodes overall in a deeper structure should reduce the amount of processing time while maintaining a higher level of fidelity.

**Optimization Types**

The same objective functions were used in the deeper networks as in their shallower counterparts.

\[
J_R(\tilde{x}, y, W) = \left[a_1^{(5)} - y\right]^2
\]  \hspace{1cm} \text{ (18)}

where

\[
a_k^{(l+1)} = g \left(\sum_{i=1}^{s_l} w_{k,i}^{(l)} \cdot a_i^{(l)}\right) \quad \text{and} \quad \tilde{a}_1 = \tilde{x}
\]  \hspace{1cm} \text{ (19)}

\[
J_{\text{AE}}(\tilde{x}, W) = \frac{1}{s_1} \sum_{i=1}^{s_1} \left[a_i^{(5)} - x_i\right]^2
\]  \hspace{1cm} \text{ (20)}

for the same definition of \(a_i^{(l)}\).

The activation at a given unit is left open-ended because the realization of those equations would require too much room with 5 layers. The activation functions used in the deeper networks were the linear and the rectified linear as much of the literature favors these in particular. [57] discusses at length the merits and ideas associated with the rectified linear unit.

**Model Assumptions**

In general it is assumed that deeper networks obtain a more abstract representation of the data by using more complex features; literally because the complexity of representation in the deeper features is higher. It is generally understood
that fewer abstract features are required to represent the same data. So reducing the total number of nodes in the network but stacking the layers more deeply can reduce the time it takes to train a well-functioning network while providing better performance. With this understanding, the networks are put to the test and analyzed in post process which is presented in the results chapter.

Tools

The two most powerful tools available to an engineer are his brain and other computing machines. When dealing with image data, nothing beats a graphics processing unit (GPU). The GPU is optimized to perform 32-bit floating point arithmetic. Originally designed for tearing through matrix operations in rendering video game graphics, the GPU has become a staple in the academic and scientific world. Specifically, the company Nvidia has produced a set of GPUs that come with a free software library, allowing anyone to utilize them [70]. The line of processors is referred to as CUDA and the specialized cores are called CUDA cores. The hardware used in this thesis includes a CUDA-enabled graphics card containing 5 GPUs with 192 cores each for a total of 960 CUDA cores. The graphics card also includes 2 gigabytes of GDDR5 random access memory (RAM). This type of RAM is optimized for graphics operations. The high bandwidth comes from a so called ‘double data rate’ which means it utilizes both the uptick and downtick of the clock to transfer data. When using this graphics card, the only problem I encountered was feeding it data fast enough.

The Cuda Toolkit is the software compliment to this piece of hardware. While I was unable to leverage massive parallelism, the Cuda Toolkit version 5.5 provided
enough functionality for matrix operations. The toolkit is a set of functions which is responsible for transferring data to and from the graphics memory, issuing commands, managing memory on the card, and a number of other essential tasks. As a software library, the toolkit has a front-end written in the C programming language. This enables software developers to directly call the functions or home-brew libraries that link to them. Because the CUDA line is affordable and so vastly popular, it is often incorporated into open-source software development.

Open-source projects combine the work of volunteers and produce products which are freely available without the threat of copywrite fraud. Not only may the intellectual property (IP) developed though open-source projects be changed and used in any manner, the IP itself is free of charge. As a result of collaboration between research and the computer science industry, fields of research progress and concepts in computer science advance as well.

Python, a product of open-source development, is a scripting language which means it has an interface that compiles and runs code in real-time at the command of a user. Originally intended to extend the shell, Python became a powerful interpretive language. Scripts in Python are able to be run line-by-line and the results are stored by the prompt exactly as they would be when running the script on its own. This interface allows a user to rapidly develop code, check syntax, and analyze results in a fraction of the time that it would take to develop a project in C or C++. Python has proven itself among the best research tools available, especially among those who research without funding.
Software libraries developed for Python are called modules. When a set of objects and functions are designed for a related task, they are often bundled into one library. Numpy (Numerical Python) is a module which performs array operations and includes efficiently implemented functions tailored for the task [71]. Scipy (Scientific python) includes a range of functionality from linear algebra operations to data analysis [71]. The plotting library used to display aspects of the analysis is called Matplotlib [72]. Many more modules are prerequisite for using Theano [73], which interprets user input to automatically develop and build C code optimized for the user’s native machine. A structure called the computational graph is responsible for determining dependencies between bodies of code in order to compile, run, and interpret the result. When a function in written using Theano, it is interpreted into a body of highly optimized C code and placed in the computational graph. When the function is called, Theano runs the body of code and interprets the results back into Python. Of course, Theano is among the various open-source projects that leverage the Cuda toolkit in order to increase performance. This affords researchers in many fields the ability to develop very high-level code very quickly that runs as fast as native C and automatically uses the GPU if available. In other words, there’s no longer a need to be an expert in computer science to leverage the power of computers because basic programming skills are sufficient.

Extending Theano, Theanets is a module that implements neural and probabilistic networks efficiently. As an even more abstract layer of software development, Theanets provides a remarkable and flexible framework for designing and implementing neural networks. A network consists of a list of layers and attributes, with each layer keeping its
own attributes. A great deal of flexibility is found not only in specifying attributes of the network but also in creating custom functionality and layers. Processing the data is defined in such a way that data management is the focus instead of the processing itself. All of the aspects that usually dominate research in this area, from debugging code to optimizing it, are handled automatically and the researcher is left to focus on the actual research. Theanets is not only a research tool but also a learning tool. Tutorials online which walk through the functionality often have the effect of clarifying and substantiating the concepts while presenting the theory and even citing research. By learning to use the coding package, one can also pick up a great deal of understanding.

Another common tool used in this research was the OpenCV (open-source computer vision) libraries. OpenCV includes an entire package of computer vision-specific tasks. Its primary use in this research is in the development of the human interface for data selection and viewing. Using powerful, standardized methods, OpenCV is able to read a variety of formats for videos and images in short order. The common image object is a pointer while the data itself is loaded into another location selected by the user whether in the memory of the GPU or the system’s main memory. The functions which perform operations over the image are then designed to receive the pointer and follow it to the data rather than instantiating a new copy with each call. An image processing researcher might feel perfectly at home among functions familiarly found in the literature like segmentation, pyramidal interpolation, and filtering. OpenCV is designed for computer vision experts to quickly assemble systems implementing any specific scene understanding task. While the data management aspect is very straight-
forward, the library actually does a lot of work to preserve the ordering of the data. Comparatively, ordering is far more important when dealing with a computer vision task than a machine learning task. This has the effect of limiting feature generation to those which can be extracted readily given an expected ordering of the domain. In the same vein, image features which rely on order in the domain are considered much less stable when assumptions of the domain’s continuity are tested. A good example is the fast two-dimensional Fourier transform, which assumes a contiguous image plane. While many optimizations and approximations exist for finding the Fourier coefficients, the features become significantly less reliable when presented with an edge because the edge may either represent a discontinuity in the image plane, which invalidates Fourier analysis, or only a discontinuity in the spectral values, which are desirable in Fourier analysis.

Nonetheless, Fourier coefficients are among a much wider range of features that can be generated rapidly by OpenCV functions. Convolutional filters can easily be defined in OpenCV and the application of these filters is implemented efficiently. Because of the sliding-window application of the filter, every window in the image may be convolved in parallel. Processors employ pipelining to perform operations on the data and data dependency is often the limiting factor in processor throughput. Computing values in parallel allows non-dependent operations to run on the same processor at the same time in different stages and thus boosts throughput. It is now a common trend in research to find a solution in postprocessing analysis and, through optimizing feature generation, follow it through to a real-time system. This is a contributing factor in the mentality behind the data-driven scheme that leads researchers to use libraries like
OpenCV to try out combinations of features and see what works in an attempt to arrive at a real-time system without having done the leg work. To aid in performing the initial analysis, OpenCV includes a set of rudimentary pattern recognition functions. While these include neural networks, they are not specialized for the task and often require the user to work in the C++ programming language. C++ does not make a good research tool as it is highly complex and requires years of study to master for the level of task that research demands.

This high-level research demands a high-level language like Python with even more abstract libraries like Theanets employing the wealth of knowledge that open-source developers have to offer. Using OpenCV for the image-facing applications like data extraction, labeling, and image formatting seems the proper level task suited for the library. The data is then transferred into a Numpy object for compact storage, fast normalization, and the use of other array operations. Theanets is used to define the neural networks, process the data, and provide a result for analysis. Theano is responsible for building and running the optimized C code and utilizes the GPU. Numpy is again responsible for producing analysis and another module generates plots for displaying the results. Together, the specialized hardware and open-source software provide an elegant toolchain for rapid research.

Metrics

Metrics provide quantitative assessment through some sort of analysis. A metric is a way of measuring the data in order to discern some information about it. The difference between a feature and a metric in this work is that features are functions of the data that
comprise the model while metrics are functions of the model which evaluate it. It is often the case in image processing that researchers define different metrics for their specific tasks. The metrics often come from either an intuitive sense of how to describe some aspect of the model’s behavior or are borrowed from a neighboring field. Because computer vision neighbors both image processing and machine learning, it is common to see a mixture of metrics pertaining to either.

The metrics from image processing used in this work pertain to the filtering subtask whereas the metrics from machine learning pertain more to statistical aspects of the analysis. In order to derive metrics, the true performance must be compared against ideal performance. This required the labeling of every lane marking on every selected scanline on every video. Labeling every testing sample by hand injected some error into the truth data because graduate students are imperfect by nature and typically sleep-deprived. Another source of error is inherent in the model, which originally trained on data collected by the same exhausted graduate student in the same manner. For these reasons, the labeling of the data is not straight-forward but must be approached with some tolerance. More exactly, the tolerance shown in Figure 21 must be determined empirically based on how well the model performs.
Figure 21. Demonstrating scanline tolerance concept given actual regressed data.

The purple arrow indicates the peak of the regressed values which was taken as an indication that the lane marking had been found. The green line represents where the truth data indicates the lane marking. Detected peaks occurring within the area denoted by the dashed green lines were taken as correctly labeled whereas if a peak were to fall beyond, in the red zone, it would be taken as incorrectly labeled. Based on this, a confusion matrix can be calculated. The confusion matrix indicates the number of true positives, false positives, false negatives, and true negatives produced by the model. True positives are correctly labeled lane markings and true negatives are correctly labeled non-lane markings. False positives are incorrectly labeled non-lane markings while false negatives are incorrectly labeled lane markings. The tolerance for defining whether the model was correct or not is explored in the analysis. However, because of the assumption that every sample set had one detection, the confusion matrix true detections contribute.
the same ratio of positives and negatives with every sample and the same applied for the false detections. Thus to avoid redundancy, the confusion matrix is condensed to only the ratio of true detections as the ratio of false detections is the complement of this.

A metric from signal processing used in this work is known as “area under the curve” (AUC). In object detection, it is most desirable to design a system that exhibits an impulse only at the detected object and ignores all other inputs. Of course, this is unreasonable so it was decided that the sharper peak the better. Because the height of the detection peak is independent of the height of the detection at any other point, the area of detection over the entire curve can be indicative of how well the model ignores non-detections. Because there is no guarantee that the detection value in all of the models will be positive, the AUC is calculated as the accumulated absolute regression value.

Similarly, to limit the AUC values themselves, only a tolerance is considered on either side of the detection. It’s worth noting that the AUC can only be calculated over true positive examples. Figure 22 below illustrates what is meant using actual training data.
In this image, the peak is aligned with the true lane marking and the area within the dashed lines is considered for this metric. As can be seen, some of the values are negative so the absolute value of this curve would reflect the contribution of that data. Again, that tolerance is chosen empirically in the analysis.

The last metric to discuss in this work comes from the actual lane detection task, though I’m entirely sure it was used elsewhere before. A missed track occurs when the detection in one frame is displaced by a large amount in the next. In tracking, it is often the case that initially finding a target requires more processing than tracking it. When a missed track occurs, the system must search again for the target. It is desirable to have as few missed tracks as possible but there is no hard statistic describing this metric. In the
scope of this work, it is assumed that a missed track is in relation to the entire video. As a general assumption, the scanlines are expected to contain one lane marking per frame with the idea that different parts of the same line were being observed in each frame. In Figure 23 an example of a missed track is given.

*Figure 23.* Illustration of a missed track with one frame on top and the next below.

In each frame, the detection was made but the large gap between detections indicates that the track was missed. It is assumed that some instability in the videos occurs but jumping by 100 pixels in less than a second seems unreasonable. The offset amount here is also determined empirically during analysis. It’s important to note that
because the number of missed tracks does not incorporate whether the track was true or false, the metric should be considered as a measure of model consistency and not validity.
CHAPTER 5: RESULTS

A few key elements to understanding the results should be covered first. Error here is the difference in pixels between the actual and anticipated position of the lane marking. The probability density functions shown are empirically derived using a histogram binning method with the number of bins set to be a thenth of the total number of samples used in the plot. With binning methods there is always a risk of misrepresentation but for the task of overall observation and determining acceptable tolerances for the harder metrics, this convention for bin width was sufficient. The analysis included 5 videos with 4 types of noise: plain, glare, shadow, and snow. The plain video had 4 scanlines in it on 8 frames for a total of 32 sample sets tested (4.27% of the total). The glare video contained 3 scanlines on 95 frames comprising 285 sample sets (38.05%). The video with shadow had 4 scanlines on 9 frames, making 36 sample sets (4.81%). There were two snow videos with one scanline each having 245 and 151 frame respectively for an aggregate of 396 snow sample sets (52.87%). Together these comprised 749 testing sample sets. Each sample set comprised the same number of samples as the width of every video happened to be the same when a scanline was taken. Each scanline contained 322 samples with only 1 containing the lane marking itself. So the total number of inputs to the models was 241,178 and the probability of seeing a lane marking was 0.31% consistently during training and testing. There were 1193 training sample sets more or less randomly selected from these videos along random scanlines, making a testing sample size of 384,146. Combined the training and testing sets included 625,324 samples. Each color sample included 2700 inputs while each grayscale sample
included only 900. All models included both variations of grayscale and color. Around 2.25 million inputs were processed as 32 bit floating point numbers by the GPU for a grand total of 9 GB per model variation: 2.25 GB for grayscale models and 6.75 GB for color. In total, regressors accounted for the processing of 108 GB, autoencoders accounted for 162 GB, and deep networks only 72 GB. A grand total of 342 GB were processed by the tool chain in this analysis.

Individual Model Performance Breakdown

The performance of individual models can be a valuable assessment of the qualities that make those models unique. Among the regressors, the only degree of freedom was the choice in activation function. Assessing model performance in terms of activation provides a notion of what contribution that degree of freedom makes to the model’s overall aptitude. Likewise among the autoencoders the size of the hidden layers helps to determine how densely the information in the samples is packed. Lastly the deep networks provide a notion of the depth of the features that are required for this particular task. Analysis of the performance of individual models provides a sense of what influence exactly these aspects of network structure and optimization have in the task of robustly performing lane analysis.

Regressors

The choice of activation function is examined among the regressors as the only degree of freedom. While often overlooked, the decision of which function to use for activation plays a crucial role in how the value is reconstructed.
Challenges and Design Options

Though it’s been covered so far in this thesis, it’s worth reiterating here that the activation functions tested by the regressors include: linear, rectified linear, thresholded rectified linear, hyperbolic tangential, sigmoid, and softmax. Color and grayscale models were tested for each of these activation functions.

Regressor Model Comparison

Having already reviewed the data, I will attempt to display it here in a naturally flowing way. While results among the regressors may have come in a different order, no assumptions are made that affect later analysis so a walkthrough maintains the flexibility to present in any order. It’s also important to note that each model type is compared at a time and the assumptions made in the autoencoders were made having reviewed all results from the regressors. Deep networks are similarly related to the regressors as well but not similarly related to the autoencoders.

At first, performance was measured in terms of error characteristics in order to define the tolerances for the harder performance metrics. The error characteristics in Figure 24 show the 3 worst-performing activation functions overall. The activation functions are aligned vertically with color model performance along the top row and grayscale along the bottom row. The distributions are quite wide and common characteristics in the error can be seen in all columns.
These models can’t provide a very good indication of what acceptable tolerances might look like. However, the best performing regressors are shown in Figure 25.

As can be seen, there appear to be much larger peaks on either side of the lane marking in these models. For this reason, the linear, rectified linear, and hyperbolic
tangential activation functions were chosen for the autoencoders and furthermore the linear and rectified linear units were selected for the deep networks.

From the better performing regressors, a selection can be made regarding a tolerance for the hard metrics. While the detection peaks seem to range from -50 pixels to 50 pixels, the width of the window was only 30. It is taken as a matter of good judgement to limit the tolerance to a reasonable value. With the understanding that these regressors are meant to exhibit poor performance, having the equivalent to only one feature on which to rely, the tolerance is set at \( \pm 15 \) pixels offset from the true value. This means that the detector may vary to the point which a minimally small portion of the lane marking resides within the window.

Tabulated below are the resulting reduced confusion matrix values which provide only a percent of correct detection. The models are considered row-wise with noise types along the columns. Aggregated in the last column are the overall results from the model. While the figures are not very good, this type of performance is to be expected. The shallow nature of the network allows for only one feature to be computed and that alone is used to define the performance of the model.
Table 1

*Tabulated condensed confusion matrix results showing the probability (%) of selecting the correct lane marking while rejecting the incorrect data.*

<table>
<thead>
<tr>
<th>Confusion Matrix</th>
<th>Plain (%)</th>
<th>Glare (%)</th>
<th>Shadow (%)</th>
<th>Snow (%)</th>
<th>Overall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear</td>
<td>0</td>
<td>96.14</td>
<td>0</td>
<td>49.5</td>
<td>62.75</td>
</tr>
<tr>
<td>Color – Relu</td>
<td>3.13</td>
<td>7.02</td>
<td>0</td>
<td>28.54</td>
<td>17.89</td>
</tr>
<tr>
<td>Color – Sigmoid</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.8</td>
<td>2.54</td>
</tr>
<tr>
<td>Color – Softmax</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Color – Tanh</td>
<td>3.13</td>
<td>0</td>
<td>0</td>
<td>16.41</td>
<td>8.81</td>
</tr>
<tr>
<td>Color – Trec</td>
<td>0</td>
<td>3.51</td>
<td>0</td>
<td>2.53</td>
<td>2.67</td>
</tr>
<tr>
<td>Gray – Linear</td>
<td>9.38</td>
<td>33.33</td>
<td>0</td>
<td>37.63</td>
<td>32.98</td>
</tr>
<tr>
<td>Gray – Relu</td>
<td>3.13</td>
<td>31.23</td>
<td>0</td>
<td>58.59</td>
<td>42.99</td>
</tr>
<tr>
<td>Gray – Sigmoid</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.29</td>
<td>2.27</td>
</tr>
<tr>
<td>Gray – Softmax</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Gray – Tanh</td>
<td>3.13</td>
<td>0</td>
<td>8.33</td>
<td>34.6</td>
<td>18.83</td>
</tr>
<tr>
<td>Gray – Trec</td>
<td>6.25</td>
<td>9.82</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

It seems common that almost all models vehemently rejected functioning in the presence of shadow. It is likely that because so little data was provided for the plain and shadow videos, models could not be adequately trained on them. Below are the error plots broken up by noise type. In each plot, color is along the top row and grayscale are along the bottom with an aggregated pdf on the left and an overlaid pdf on the right. Only the models using linear, rectified linear, and hyperbolic tangential activation functions are included.
Figure 26. Aspects of the model error on the plain set

Figure 27. Aspects of the model error on the glare set
Figure 28. Aspects of the model error on the shadow set

Figure 29. Aspects of the model error on the snow set
These figures indicate the best performance the regressors had to offer per a given type of noise. Clearly these simple regressors favor the noise types for which more data was used in training and testing. Next the area under the curve and the missed track rate are tabulated.

Table 2

*AUC and missed track rate (%)*

<table>
<thead>
<tr>
<th>Model</th>
<th>Area Under the Curve</th>
<th>Missed Track Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear</td>
<td>168.41</td>
<td>5.57</td>
</tr>
<tr>
<td>Color – Relu</td>
<td>217.4</td>
<td>5.71</td>
</tr>
<tr>
<td>Color – Sigmoid</td>
<td>8.69e-11</td>
<td>19.27</td>
</tr>
<tr>
<td>Color – Softmax</td>
<td>NaN</td>
<td>0</td>
</tr>
<tr>
<td>Color – Tanh</td>
<td>212.35</td>
<td>7.74</td>
</tr>
<tr>
<td>Color – Trec</td>
<td>2.3e-4</td>
<td>39.27</td>
</tr>
<tr>
<td>Gray – Linear</td>
<td>110.3</td>
<td>8.15</td>
</tr>
<tr>
<td>Gray – Relu</td>
<td>111.34</td>
<td>6.39</td>
</tr>
<tr>
<td>Gray – Sigmoid</td>
<td>1.12e-4</td>
<td>20.92</td>
</tr>
<tr>
<td>Gray – Softmax</td>
<td>NaN</td>
<td>0</td>
</tr>
<tr>
<td>Gray – Tanh</td>
<td>95.3</td>
<td>6.66</td>
</tr>
<tr>
<td>Gray – Trec</td>
<td>3.18e-5</td>
<td>38.04</td>
</tr>
</tbody>
</table>

The first column, area under the curve, was measured only for correct tracks and only within ±30 pixels of the detected peak. The numbers are unitless and averaged across all successful detections of the lane marking per model. A smaller value is desired, though each output was measured differently based on the activation. The possible range for linear units is unrestricted whereas the units for Relu are non-negative, for sigmoid are on (0,1), and so on as previously discussed. The missed track rate measures the number of missed tracks out of the total number of opportunities to miss a track. The
remarkably low missed track rates indicate that the regressors, while not good at detecting the lane marking, were very consistent in their bad performance. In other words, they maintained strong ideas about what to look for even if their ideas were wrong. From a sensor perspective, this is good news because the simple feature found in these models can provide a stable platform for building more adept systems.

*Model-Wise Observations*

Numerically, the color regressor employing the linear activation function was able to make correct detections 96.14% of the time for the glare video and almost half the time for the ones with snow while only missing an average of 5.57% of tracks. To put it simply, the color regressor performed very well. Overall the grayscale model with a rectified linear activation was a close second and showed more robust performance even in the presence of snow. It is interesting to note that these specific regressors represent convolutional filters. The linear activation represents a convolutional filter which can attain negative values while the rectified linear contains only non-negative coefficients. The representative convolutional filters are shown below.

*Figure 30. Convolutional filter representation of color and relu regressors*
The values are displayed as colors because in each 30 by 30 window there are 2700 weights. These represent the best possible performance of convolutional filters on the data set. A humbling realization is that the chances of coming across these filters using the data-driven scheme are almost nonexistent.

**AutoEncoders**

The degree of freedom provided by the autoencoders was in the number of hidden units used to build an alternate representation of the data. This representation was then used as input to a regressor. Three activation functions were used on in the autoencoder models: linear, rectified linear, and hyperbolic tangential. Three choices of hidden layer size were also selected: 100, 500, and 1000 nodes.

**Challenges and Design Options**

Because the typical use for autoencoders does not account for their updating in a new domain, a different method was used. This adaptation feature slowed down deployment of the model and the end result was to limit the number of deeper networks that could be run due to time constraints. It is also difficult to interpret results from the autoencoder because of the unique method whereby they were deployed.

**AutoEncoder Model Comparison**

The error plots below show the characteristics of all 18 models with color model error on top and grayscale model error below. The hidden layer size is denoted in larger text column-wise.
Figure 31. Error of autoencoders with linear activation functions

Figure 32. Error of autoencoders with rectified linear activation functions
Figure 33. Error of autoencoders with hyperbolic tangential activation functions

Two observations are immediately apparent from examining these error plots: that the hyperbolic tangential activation function seemed to demonstrate better results on the robust data set in this organization and that no reasonable tolerances could mitigate the amount of error seen here. The table below puts a metric to the performance.
Table 3

*Tabulated condensed confusion matrix for autoencoders*

<table>
<thead>
<tr>
<th>Confusion Matrix</th>
<th>Plain (%)</th>
<th>Glare (%)</th>
<th>Shadow (%)</th>
<th>Snow (%)</th>
<th>Overall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear 100</td>
<td>15.63</td>
<td>10.53</td>
<td>0</td>
<td>7.32</td>
<td>8.54</td>
</tr>
<tr>
<td>Color – Linear 500</td>
<td>0</td>
<td>19.3</td>
<td>8.33</td>
<td>21.21</td>
<td>18.96</td>
</tr>
<tr>
<td>Color – Linear 1000</td>
<td>0</td>
<td>11.23</td>
<td>0</td>
<td>11.11</td>
<td>10.15</td>
</tr>
<tr>
<td>Color – Relu 100</td>
<td>12.5</td>
<td>4.56</td>
<td>2.78</td>
<td>0.76</td>
<td>2.8</td>
</tr>
<tr>
<td>Color – Relu 500</td>
<td>3.13</td>
<td>1.05</td>
<td>0</td>
<td>5.05</td>
<td>3.2</td>
</tr>
<tr>
<td>Color – Relu 1000</td>
<td>6.25</td>
<td>21.05</td>
<td>5.56</td>
<td>11.11</td>
<td>14.42</td>
</tr>
<tr>
<td>Color – Tanh 100</td>
<td>0</td>
<td>22.11</td>
<td>0</td>
<td>5.81</td>
<td>11.45</td>
</tr>
<tr>
<td>Color – Tanh 500</td>
<td>3.13</td>
<td>20.35</td>
<td>11.11</td>
<td>27.02</td>
<td>22.7</td>
</tr>
<tr>
<td>Color – Tanh 1000</td>
<td>9.38</td>
<td>16.49</td>
<td>0</td>
<td>27.27</td>
<td>21.09</td>
</tr>
<tr>
<td>Gray – Linear 100</td>
<td>9.38</td>
<td>3.16</td>
<td>11.11</td>
<td>5.81</td>
<td>5.21</td>
</tr>
<tr>
<td>Gray – Linear 500</td>
<td>9.38</td>
<td>7.37</td>
<td>8.33</td>
<td>15.15</td>
<td>11.62</td>
</tr>
<tr>
<td>Gray – Linear 1000</td>
<td>6.25</td>
<td>12.63</td>
<td>0</td>
<td>9.85</td>
<td>10.28</td>
</tr>
<tr>
<td>Gray – Relu 100</td>
<td>9.38</td>
<td>2.46</td>
<td>0</td>
<td>2.27</td>
<td>2.54</td>
</tr>
<tr>
<td>Gray – Relu 500</td>
<td>6.25</td>
<td>9.47</td>
<td>0</td>
<td>26.26</td>
<td>17.76</td>
</tr>
<tr>
<td>Gray – Relu 1000</td>
<td>0</td>
<td>6.67</td>
<td>2.78</td>
<td>13.38</td>
<td>9.75</td>
</tr>
<tr>
<td>Gray – Tanh 100</td>
<td>0</td>
<td>44.21</td>
<td>11.11</td>
<td>10.61</td>
<td>22.96</td>
</tr>
<tr>
<td>Gray – Tanh 1000</td>
<td>0</td>
<td>0</td>
<td>8.33</td>
<td>15.66</td>
<td>8.68</td>
</tr>
</tbody>
</table>

This is a dramatic difference from the performance of the regressors. Results here indicate that while overall performance suffered greatly, the autoencoders were better able to handle the data from domains that contributed fewer samples (plain and shadow). It seems reasonable to speculate that the transformation through the hidden layer captured some aspect of the lane marking which was robust to effects seldom observed during training and was able to capitalize upon this aspect during testing. Tabulated next are the other hard metrics.
Table 4.

*Area under the curve and missed track rates for autoencoders tabulated.*

<table>
<thead>
<tr>
<th>Model</th>
<th>Area Under the Curve</th>
<th>Missed Track Rate (%)</th>
<th>Overall Detection Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear 100</td>
<td>5885.28</td>
<td>46.6</td>
<td>8.54</td>
</tr>
<tr>
<td>Color – Linear 500</td>
<td>4468.86</td>
<td>48.67</td>
<td>18.96</td>
</tr>
<tr>
<td>Color – Linear 1000</td>
<td>2897.36</td>
<td>49.73</td>
<td>10.15</td>
</tr>
<tr>
<td>Color – Relu 100</td>
<td>6482</td>
<td>50</td>
<td>2.8</td>
</tr>
<tr>
<td>Color – Relu 500</td>
<td>111.42</td>
<td>44.43</td>
<td>3.2</td>
</tr>
<tr>
<td>Color – Relu 1000</td>
<td>655.88</td>
<td>43.61</td>
<td>14.42</td>
</tr>
<tr>
<td>Color – Tanh 100</td>
<td>2634.42</td>
<td>34.92</td>
<td>11.45</td>
</tr>
<tr>
<td>Color – Tanh 500</td>
<td>3050.25</td>
<td>37.5</td>
<td>22.7</td>
</tr>
<tr>
<td>Color – Tanh 1000</td>
<td>2634.3</td>
<td>48.67</td>
<td>21.09</td>
</tr>
<tr>
<td>Gray – Linear 100</td>
<td>5059.3</td>
<td>23.23</td>
<td>5.21</td>
</tr>
<tr>
<td>Gray – Linear 500</td>
<td>1628.5</td>
<td>52.04</td>
<td>11.62</td>
</tr>
<tr>
<td>Gray – Linear 1000</td>
<td>1949.37</td>
<td>40.76</td>
<td>10.28</td>
</tr>
<tr>
<td>Gray – Relu 100</td>
<td>66.35</td>
<td>34.1</td>
<td>2.54</td>
</tr>
<tr>
<td>Gray – Relu 500</td>
<td>689.89</td>
<td>43.75</td>
<td>17.76</td>
</tr>
<tr>
<td>Gray – Relu 1000</td>
<td>832.02</td>
<td>37.5</td>
<td>9.75</td>
</tr>
<tr>
<td>Gray – Tanh 100</td>
<td>1825</td>
<td>38.86</td>
<td>22.96</td>
</tr>
<tr>
<td>Gray – Tanh 500</td>
<td>1130.21</td>
<td>43.07</td>
<td>9.35</td>
</tr>
<tr>
<td>Gray – Tanh 1000</td>
<td>1876.56</td>
<td>40.35</td>
<td>8.68</td>
</tr>
</tbody>
</table>

The values for area under the curve can be evaluated having 6 measures with which to compare in like terms: 3 color models and 3 grayscale models all with the same activation function. For clarity, the overall detection rate was included and models with comparable AUC measures were color coated. It seems to be a trend that having around 500 nodes in the hidden layer gives a good measure for area under the curve. For comparison, that’s around a fifth of the number of inputs in color models and half the number of inputs in grayscale models. While the autoencoders showed robustness to a wider variety of noise, they also seemed to get distracted more often, with missed track rates nearly 900 times their two-layer counterparts.
**Model-Wise Observations**

The color autoencoder using a hyperbolic tangential activation function showed the best overall performance with a peak of 22.7% correct detection on 500 hidden layer nodes. The same model was also arguably the most robust, providing correct detection in 3.13% of plain scans, 20.35% of those with glare, 11.11% of those with shadow, and 27.02% of those with snow. This model was the star of the autoencoders and demonstrated the key trends in the rest of the autoencoder analysis. Figure 34 provides the error plots for further consideration.

*Figure 34.* Individual performance of the color AE with 500 hidden nodes using the hyperbolic tangential activation function
As expected, the reason for the increased performance in situations requiring robustness from glare are attributed to a general flattening out of that pdf. The probability distribution becoming more uniform indicates the observation of more or less random guessing instead of exhibiting a bias associated with any quality in particular. In other words, the reason for the increase is the disassociation with a particular quality of the noise. The indication of strong peaks in the other noise types means that the disassociation with shadow features did not indicate a general disassociation with the qualities of the lane marking. More simply, the autoencoder is exhibiting robustness by capturing the qualities of the noise so the regressor can specifically reject those qualities.

*Deep Networks*

The depth of a network can help to determine the complexity of features it requires. If a deeper network shows better performance, it can be indicative of the task itself. It is common practice to examine a complex problem by imposing many assumptions and restrictions and producing a method for that scenario. The reduced problem can then be solved and other sets of assumptions can yield a different solution. The two solutions are usually compared to provide a better understanding of the main task. Deeper layers can reproduce more complex functions of the data as per the UAT so it follows that if features at a given depth are good indicators for solving a problem, the problem itself requires the same complexity from a solution. [68], [74] in the pursuit of robustness through sparsity refine their feature selection by keeping only the most active nodes in each layer because they provide better indicators than inactive nodes. The distinction between deep regressors and deep autoencoders is an important one as
autoencoders train without the influence of the target value while regressors train specifically to that target. While [68], [74] use autoencoders to provide a representation of the data and thus active nodes representing complexity found in the data, it seems reasonable to assume deep regressors would exhibit similar characteristics.

Challenges and Design Options

The separation of the deep autoencoders from the deep regressors, as opposed to trying mixtures of the two, was supported by what was seen in the autoencoders. Being trained independently of the regressor provided an advantage in the form of learning and maintaining an understanding of the characteristics inherent in the data free of the influence of the target value. Those characteristics inherent in the data included both features contributing to the signal and features contributing to the noise, often overlapping. While a regressor may have rewarded only those features contributing to the signal and punished those contributing to noise, the autoencoders maintained both class definitions and were able to exhibit robustness. Similarly, the deep autoencoders built each layer free of knowledge of the task and learned only the complexity of the data while the deep regressors built each layer tailored specifically to the goal of addressing lane detection. It’s important to denote the duality between the complexity inherent in the data and the complexity inherent in the task. The depth of regressors can indicate the complexity required to address the task given the data. The depth of autoencoders can indicate the complexity of the data itself. Using mixtures of these two systems can provide another degree of freedom along which to compromise a tradeoff between what works best for the data and what works best for the task. While this demonstration ends
with these aspects, it’s important to note that the room for expansion in this direction is virtually limitless.

**Deep Network Model Comparison**

Among the autoencoders, only 4 were tested. Based on observations from the regressors, the deep autoencoders employed the linear and rectified linear activation functions. Analysis of the deep regressors speaks to the complexity of the data itself. While only 5 layers are used here, the effect of depth is still somewhat visible. Below are the error plots from the deep autoencoders in Figure 35.

![Figure 35. Deep autoencoder error characteristics](image)

Along the top row are results from the color networks and along the bottom are the grayscale, with linear activation in the first column and rectified linear in the second. The plots don’t indicate a huge amount but do speak to the nature of how evenly
distributed the features are. Because the errors aren’t simply uniform it stands to reason that
the deep autoencoders are picking up some real influence. The next section provides a comparative performance between all model types and we will see that features captured by these deeper autoencoders exhibit different performance than features captured by their shallower counterparts. The hard metrics are tabulated next.

Table 5

*Condensed confusion matrix for deep autoencoders*

<table>
<thead>
<tr>
<th>Confusion Matrix</th>
<th>Plain (%)</th>
<th>Glare (%)</th>
<th>Shadow (%)</th>
<th>Snow (%)</th>
<th>Overall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear</td>
<td>1.13</td>
<td>19.65</td>
<td>2.78</td>
<td>10.61</td>
<td>13.35</td>
</tr>
<tr>
<td>Color – Relu</td>
<td>0</td>
<td>3.51</td>
<td>5.56</td>
<td>9.1</td>
<td>6.41</td>
</tr>
<tr>
<td>Gray – Linear</td>
<td>3.13</td>
<td>2.81</td>
<td>5.56</td>
<td>42.42</td>
<td>23.9</td>
</tr>
<tr>
<td>Gray – Relu</td>
<td>6.25</td>
<td>25.61</td>
<td>16.67</td>
<td>28.28</td>
<td>25.77</td>
</tr>
</tbody>
</table>

Table 6

*AUC and missed track rate for deep autoencoders*

<table>
<thead>
<tr>
<th></th>
<th>Area Under the Curve</th>
<th>Missed Track Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear</td>
<td>8756.38</td>
<td>38.59</td>
</tr>
<tr>
<td>Color – Relu</td>
<td>110.02</td>
<td>31.39</td>
</tr>
<tr>
<td>Gray – Linear</td>
<td>4402.62</td>
<td>33.02</td>
</tr>
<tr>
<td>Gray – Relu</td>
<td>792.16</td>
<td>33.02</td>
</tr>
</tbody>
</table>

The most robust performance observed so far can be seen in the grayscale deep autoencoder employing the rectified linear activation function. While exhibiting a
substantial missed track rate, the response (denoted by AUC) was very sharp given the proper stimulus. Lastly, results from the deep regressors are shown in Figure 36 next.

![Figure 36. Deep regressor error characteristics](image)

The top left plot indicates the deep color regressor error characteristics with a linear activation function. This model exhibited the sharpest peak and the smallest cluster overall but did not achieve the best performance. The error characteristics for the deep regressors employing a rectified linear activation were exactly the same; which indicated that the models were likely completely useless. Tabulated below is confirmation of this.
Table 7

*Hard performance metrics for the deep regressors*

<table>
<thead>
<tr>
<th>Confusion Matrix</th>
<th>Plain (%)</th>
<th>Glare (%)</th>
<th>Shadow (%)</th>
<th>Snow (%)</th>
<th>Overall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear</td>
<td>0</td>
<td>30.18</td>
<td>0</td>
<td>39.9</td>
<td>32.58</td>
</tr>
<tr>
<td>Color – Relu</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Gray – Linear</td>
<td>0</td>
<td>33.33</td>
<td>0</td>
<td>32.32</td>
<td>29.77</td>
</tr>
<tr>
<td>Gray – Relu</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

As the deep regressors with rectified linear activation did not correctly detect even a single lane marking, it is likely that the activation function was responsible for pushing values beyond reasonable limits for the data’s representation. The linear deep regressors, however, did not suffer this problem. As expected, they showed good classification on only a few of the scenarios with good overall performance, favoring the scenarios which were better represented in the dataset and overlooking others. The hard metric results are tabulated below.

Table 8

*Metrics for the performance of the deep regressors*

<table>
<thead>
<tr>
<th></th>
<th>Area Under the Curve</th>
<th>Missed Track Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear</td>
<td>100.95</td>
<td>1.63</td>
</tr>
<tr>
<td>Color – Relu</td>
<td>NaN</td>
<td>0</td>
</tr>
<tr>
<td>Gray – Linear</td>
<td>102.58</td>
<td>2.31</td>
</tr>
<tr>
<td>Gray – Relu</td>
<td>NaN</td>
<td>0</td>
</tr>
</tbody>
</table>

With the higher correct classification rate, the regressors also exhibited smaller area under the curve and very few missed tracks. These regressors showed very similar
characteristics as the simple regressors; having sharp detection and good consistency but not exhibiting robustness or preserving unused characteristics.

Comparative Performance

Many axes exist along which comparison can be made. Among the regressors, an obvious comparison is between the error characteristics of the color models versus those of the grayscale models. Shown below with color on the left and grayscale on the right, results from each video are overlaid in different colors. It is clear that some of the distributions are quite wide and others are quite sharply centered near zero. The linear and rectified linear units respectively show the best performance with the hyperbolic tangent a close third.

![Overlaid PDF Among Color Models](image1)

![Overlaid PDF Among Grayscale Models](image2)

*Figure 37. Comparison between color and grayscale error qualities*

Color data’s larger amount of information benefiting model performance is also observed in the autoencoders. While the detection performance overall was better in the
regressors, the autoencoders were able to detect the lane in some scenarios where regressors failed entirely. In terms of robustness, the autoencoders exhibited better performance in those areas while the regressors more strongly associated their features with the target. Deeper autoencoders continued this trend of exhibiting more robust performance. The figures compare the impact of depth on error characteristics with the deep network on the left and those with varying sizes of hidden layers on the right.

Figure 38. Comparison of autoencoders with linear activation

Figure 39. Comparison of autoencoders with rectified linear activation
In general, the use of more complex features widened the peaks among autoencoders using the same data type. This means that the description of the data through features in the hidden layer provided for flexibility in the class definitions found by their respective regressors. A comparison of their performance indicates that deeper features did bring some benefit by providing better characterization of both the signal and the noise, causing sharper indication and better consistency. The table below shows the numerical comparison between the deep autoencoders and those with only one hidden layer.

Table 9  
*Performance of the comparable autoencoders*

<table>
<thead>
<tr>
<th>Confusion Matrix</th>
<th>Plain (%)</th>
<th>Glare (%)</th>
<th>Shadow (%)</th>
<th>Snow (%)</th>
<th>Overall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear 100</td>
<td>15.63</td>
<td>10.53</td>
<td>0</td>
<td>7.32</td>
<td>8.54</td>
</tr>
<tr>
<td>Color – Linear 500</td>
<td>0</td>
<td>19.3</td>
<td>8.33</td>
<td>21.21</td>
<td>18.96</td>
</tr>
<tr>
<td>Color – Linear 1000</td>
<td>0</td>
<td>11.23</td>
<td>0</td>
<td>11.11</td>
<td>10.15</td>
</tr>
<tr>
<td>Deep Color – Linear</td>
<td>1.13</td>
<td>19.65</td>
<td>2.78</td>
<td>10.61</td>
<td>13.35</td>
</tr>
<tr>
<td>Color – Relu 100</td>
<td>12.5</td>
<td>4.56</td>
<td>2.78</td>
<td>0.76</td>
<td>2.8</td>
</tr>
<tr>
<td>Color – Relu 500</td>
<td>3.13</td>
<td>1.05</td>
<td>0</td>
<td>5.05</td>
<td>3.2</td>
</tr>
<tr>
<td>Color – Relu 1000</td>
<td>6.25</td>
<td>21.05</td>
<td>5.56</td>
<td>11.11</td>
<td>14.42</td>
</tr>
<tr>
<td>Deep Color – Relu</td>
<td>0</td>
<td>3.51</td>
<td>5.56</td>
<td>9.1</td>
<td>6.41</td>
</tr>
<tr>
<td>Gray – Linear 100</td>
<td>9.38</td>
<td>3.16</td>
<td>11.11</td>
<td>5.81</td>
<td>5.21</td>
</tr>
<tr>
<td>Gray – Linear 500</td>
<td>9.38</td>
<td>7.37</td>
<td>8.33</td>
<td>15.15</td>
<td>11.62</td>
</tr>
<tr>
<td>Gray – Linear 1000</td>
<td>6.25</td>
<td>12.63</td>
<td>0</td>
<td>9.85</td>
<td>10.28</td>
</tr>
<tr>
<td>Deep Gray – Linear</td>
<td>3.13</td>
<td>2.81</td>
<td>5.56</td>
<td>42.42</td>
<td>23.9</td>
</tr>
<tr>
<td>Gray – Relu 100</td>
<td>9.38</td>
<td>2.46</td>
<td>0</td>
<td>2.27</td>
<td>2.54</td>
</tr>
<tr>
<td>Gray – Relu 500</td>
<td>6.25</td>
<td>9.47</td>
<td>0</td>
<td>26.26</td>
<td>17.76</td>
</tr>
<tr>
<td>Gray – Relu 1000</td>
<td>0</td>
<td>6.67</td>
<td>2.78</td>
<td>13.38</td>
<td>9.75</td>
</tr>
</tbody>
</table>
These show the significant improvement in robustness in almost all deep autoencoders (highlighted in green) with the exception of the color deep rectified linear autoencoder. The network structure overall seemed to point to autoencoders as having more robust performance and furthermore deep autoencoders exhibiting more robustness than shallower ones. The hard metrics comprise the table below.

**Table 10**

*Hard metrics for the autoencoders overall*

<table>
<thead>
<tr>
<th></th>
<th>Area Under the Curve</th>
<th>Missed Track Rate (%)</th>
<th>Overall Detection Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear 100</td>
<td>5885.28</td>
<td>46.6</td>
<td>8.54</td>
</tr>
<tr>
<td>Color – Linear 500</td>
<td>4468.86</td>
<td>48.67</td>
<td>18.96</td>
</tr>
<tr>
<td>Color – Linear 1000</td>
<td>2897.36</td>
<td>49.73</td>
<td>10.15</td>
</tr>
<tr>
<td>Deep Color – Linear</td>
<td>8756.38</td>
<td>38.59</td>
<td>13.35</td>
</tr>
<tr>
<td>Color – Relu 100</td>
<td>6482</td>
<td>50</td>
<td>2.8</td>
</tr>
<tr>
<td>Color – Relu 500</td>
<td>111.42</td>
<td>44.43</td>
<td>3.2</td>
</tr>
<tr>
<td>Color – Relu 1000</td>
<td>655.88</td>
<td>43.61</td>
<td>14.42</td>
</tr>
<tr>
<td>Deep Color – Relu</td>
<td>110.02</td>
<td>31.39</td>
<td>6.41</td>
</tr>
<tr>
<td>Gray – Linear 100</td>
<td>5059.3</td>
<td>23.23</td>
<td>5.21</td>
</tr>
<tr>
<td>Gray – Linear 500</td>
<td>1628.5</td>
<td>52.04</td>
<td>11.62</td>
</tr>
<tr>
<td>Gray – Linear 1000</td>
<td>1949.37</td>
<td>40.76</td>
<td>10.28</td>
</tr>
<tr>
<td>Deep Gray – Linear</td>
<td>4402.62</td>
<td>33.02</td>
<td>23.9</td>
</tr>
<tr>
<td>Gray – Relu 100</td>
<td>66.35</td>
<td>34.1</td>
<td>2.54</td>
</tr>
<tr>
<td>Gray – Relu 500</td>
<td>689.89</td>
<td>43.75</td>
<td>17.76</td>
</tr>
<tr>
<td>Gray – Relu 1000</td>
<td>832.02</td>
<td>37.5</td>
<td>9.75</td>
</tr>
<tr>
<td>Deep Gray – Relu</td>
<td>792.16</td>
<td>33.02</td>
<td>25.77</td>
</tr>
</tbody>
</table>

With comparable values shown by color and depth indicated by effect, the table indicates that the deep networks classified more generally with larger AUC. The consistency was fairly similar between all autoencoders as indicated by the missed track rate. The overall detection rate is shown with the hard metrics to provide an overall idea of what context the other metrics should be interpreted in. Again, on the whole, the deep regressors showed better performance. Meanwhile, the deep regressors showed worse
performance than their simpler counterparts. Tabulated are the results to compare directly.

Table 11

**Regressor table of model performance**

<table>
<thead>
<tr>
<th>Confusion Matrix</th>
<th>Plain (%)</th>
<th>Glare (%)</th>
<th>Shadow (%)</th>
<th>Snow (%)</th>
<th>Overall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear</td>
<td>0</td>
<td>96.14</td>
<td>0</td>
<td>49.5</td>
<td>62.75</td>
</tr>
<tr>
<td>Gray – Linear</td>
<td>9.38</td>
<td>33.33</td>
<td>0</td>
<td>37.63</td>
<td>32.98</td>
</tr>
<tr>
<td>Deep Color – Linear</td>
<td>0</td>
<td>30.18</td>
<td>0</td>
<td>39.9</td>
<td>32.58</td>
</tr>
<tr>
<td>Deep Gray – Linear</td>
<td>0</td>
<td>33.33</td>
<td>0</td>
<td>32.32</td>
<td>29.77</td>
</tr>
</tbody>
</table>

Table 12

**Regressor metrics**

<table>
<thead>
<tr>
<th></th>
<th>Area Under the Curve</th>
<th>Missed Track Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color – Linear</td>
<td>168.41</td>
<td>5.57</td>
</tr>
<tr>
<td>Gray – Linear</td>
<td>110.3</td>
<td>8.15</td>
</tr>
<tr>
<td>Deep Color – Linear</td>
<td>100.95</td>
<td>1.63</td>
</tr>
<tr>
<td>Deep Gray – Linear</td>
<td>102.58</td>
<td>2.31</td>
</tr>
</tbody>
</table>

It is apparent from the tables above that while deep regressors showed sharper detection and better consistency, they found a detriment in overall performance compared to the simpler regressors. The simpler regressors also seemed to exhibit better robustness. Having less complex features, the simpler regressors seemed better able to disregard erroneous information at a detriment to robustness.
Collective Observations

The regressors indicated that the choice of activation function was important for this diverse dataset, showing better performance among the linear, rectified linear, and hyperbolic tangential activation functions. Using these activations for the autoencoders revealed robustness that wasn’t achievable with the regressors. The autoencoders revealed that around 500 inputs worked best for both robust and correct performance. The consistency of autoencoders, however, was far less than the consistency of the regressors. With the added dimension of depth, the regressors maintained even better consistency at a detriment to overall performance and robustness. The deep autoencoders exhibited even greater robustness and better performance while remaining fairly consistent. In the next chapter, conclusions are drawn about these findings.
CHAPTER 6: CONCLUSION

As engineers, we are tasked with wrestling against the immense complexity of both reality and the ideas we seek to implement. Often we skip in-depth analysis in favor of using what seems to work at the time but it is easy to fall victim to the never ending quest of trying countless combinations of features in hopes of landing on the right one. To remedy this mentality, a model-driven scheme was favored in this work to demonstrate the approach to analysis that leverages statistically sound, efficient, and easy-to-use tools from machine learning.

Discussion of Observations

The goal of any research is to discover the domain specific knowledge that underlies the data and governs model performance. Even though this work did not produce a specific model that outperformed some standard, taking this approach provided a wealth of domain specific understanding from which other research can actually benefit. Any engineer who provides a well-functioning model without providing an explanation for its advantage should be questioned and second-guessed. Replicating research is seldom published and performing rigorous analysis can take too long for those wanting to publish rapidly and so both have become less frequent but no less vital. This work is meant to prove the value of the model-driven mentality of using powerful tools to aid in analysis for the discovery of domain specific knowledge. Thus conclusions here do not regard the models themselves but instead what the models have demonstrated about the domain of video data in the task of lane detection.
While not exhibiting the best performance, the models seem to have an affinity for both noise types which contained more data to begin with; indicating that even large amounts of noisy data does a better job of training them than a small amount of perfect data (plain). Furthermore the consistency between color data and grayscale data model performance with a slight improvement in color data indicates that having more information on the same exact scene from the same exact angle using the same exact acquisition system provides better results. It follows that models using more of the spectrum in video data would have an advantage over those using less of it.

The autoencoders bring a new understanding of how that network structure influences model behavior. By translating the raw data through a hidden layer which builds features to represent it free of the influence of a target value, features robust to different types of noise are stored together. These features can later be recalled and capitalized on to provide more robust performance in scenarios that are less frequently observed. Furthermore, the example of the color autoencoder with 500 hidden units demonstrates that the autoencoder can build a profile of the noise and allow those features to be actively rejected for more robust results. This plays into the age-old adage that more information is always better. Information about the noise itself can help a model understand what to reject and actively learning a profile of the noise can be very beneficial.

Robustness being the key in this work requires some summary as well. Robustness was approached first in the data itself. The diversity of inputs from those containing shadow, glare, and snow to the plain group demanded a robust model to begin
with. Robustness was also found in the mixture of these noise types as the training had to accommodate for having both very few and very many samples of different domains and being able to adapt to both scenarios. The reconstructed convolutional filters from the regressors show visually how robustness was achieved in those models: by sampling more or less randomly the background while distinguishing the lane marking itself. The autoencoders showed a most interesting link to robustness by generating features of the signal and of the noise separately and joint. An accompanying regressor was able to make a distinction between them for a more robust result than simply using the raw input. As the depth of the network increased, the autoencoders became better able to handle the robustness using more complex features to match the complexity of the data. In stark contrast, the deep regressors showed even stricter criteria for rejecting noise and, in doing so, also rejected valuable parts of the data. This lead to worse performance but better consistency throughout the video.

Through this analysis, the application of machine learning models in terms of robustness can be better understood and the powerful tools of machine learning can be better utilized for model design. In general, regressors show behavior that indicates their sharp rejection of outliers even at a detriment to robustness. Regressors also kept very little “memory” of features provided from those outlier instances. Autoencoders, on the other hand, seemed to better reflect the data itself by maintaining those features which applied even to outliers. As autoencoders grew deeper their structure was better able to capture and separate both the signal and the noise, which lead to better robustness and better performance. Even more generally, it seems that while regressors capture the
complexity of solving a task given the data, autoencoders capture the complexity of only
the data itself and inherent separations therein.

The Next Step

As neural networks become more heavily used, other aspects of model design can
be explored which have yet to be invented. Recurrent neural networks keep a memory
and provide a sense of feature longevity. Sequential networks provide a sense of order.
The realm of machine learning is only limited by our imagination. As we progress to
different tasks, model development becomes more relevant and leveraging powerful
analysis tools becomes more important.

The model-driven scheme provides a method for streamlining the analysis of
video data but no system currently exists which approximates those features as real-time
features for streamlined performance. It is often the case that in research, postprocessing
analysis is used for finding which aspects of the data contribute to a decision, those
aspects are derived using feature generation, and that generation is streamlined for real-
time application. By streamlining the analysis, a third of the process has been automated.
It stands to reason that future work includes automating the rest of the process as well. A
good first step in this direction would be automating the generation of a profile for model
characteristics and performance using different aspects like target value (regressor vs
autoencoder), layer structure (number of units per layer), data and task complexity
(network depth), and other aspects of model design. From there another system might
rank the predefined features by the tradeoffs in the model profile. The engineer would
then use this ranking to construct a model that weighs the performance against other factors beyond the scope of the specific sensor analysis.

Another beneficial contribution is based in the information input to the system. As observed in the analysis, regressors benefit from having more information while autoencoders deem it more complex. It follows that the inclusion of alternate sensor types as similar input would benefit a lane detection system. Inclusion of hyperspectral, radar, and sonic sensors may provide advantages that visual-spectrum input alone cannot. The addition of new sensors is straight-forward in a machine learning model as an augmentation of the input data vector. Furthermore, the dynamic introduction and excision of sensors is relatively simple; corresponding to the contribution of the weights. Denoising can account for robustness to the input itself. Lastly, the flexibility of the system may indeed provide another axis along which to analyze a model.

Conclusion

This work extends the model-driven scheme in the classical approach to lane detection by performing analysis along different aspects of model development. Robustness, being the main focus, was pursued in both the data set and the choice of models and analysis of model performance in terms of how robustness could be incorporated was explored. Conclusions were made regarding both how the results could be interpreted and also how analysis in other problems could be streamlined. Future progress in this direction was outlined and a straightforward path was laid out for achieving it. In all, the value of data analysis was shown to aid in the problem of lane detection through the exploration of robustness in machine learning models.
REFERENCES


[47] A. Graves, “Generating Sequences With Recurrent Neural Networks,”
ArXiv13080850 Cs, Aug. 2013.


