https://doi.org/10.1007/s00165-021-00539-2
BCS © 2021
Formal Aspects of Computing (2021) 33: 325–341

Formal Aspects of Computing

SDLV: verification of steering angle safety for self-driving cars

Huihui Wu[1,2], Deyun Lv[1,2], Tengxiang Cui[1,2], Gang Hou[1,2], Masahiko Watanabe[3], and Weiqiang Kong[1,2]

[1] School of Software Technology, Dalian University of Technology, Dalian, China
[2] Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, Dalian, China
[3] NTT DATA Automobiligence Research Center, Yokohama, Japan

Abstract. Self-driving cars over the last decade have achieved significant progress like driving millions of miles without any human intervention. However, behavioral safety in applying deep-neural-network-based (DNN-based) systems for self-driving cars could not be guaranteed. Several real-world accidents involving self-driving cars have already happened, some of which have led to fatal collisions. In this paper, we present a novel and automated technique for verifying steering angle safety for self-driving cars. The technique is based on deep learning verification (DLV), which is an automated verification framework for safety of image classification neural networks. We extend DLV by leveraging neuron coverage and slack relationship to solve the judgement problem of predicted behaviors, and thus, to achieve verification of steering angle safety for self-driving cars. We evaluate our technique on the NVIDIA's end-to-end self-driving architecture, which is a crucial ingredient in many modern self-driving cars. Experimental results show that our technique can successfully find adversarial misclassifications (i.e., incorrect steering decisions) within given regions if they exist. Therefore, we can achieve safety verification (if no misclassification is found for all DNN layers, in which case the network can be said to be stable or reliable w.r.t. steering decisions) or falsification (in which case the adversarial examples can be used to fine-tune the network).

Keywords: Self-driving cars; Safety verification; Steering angle; Neuron coverage; Slack relationship

1. Introduction

Recent advances in Machine Learning (ML) techniques [GBC16] have led to the development of self-driving cars. Previous experimental results show that a well-trained system of self-driving cars, using sensors like cameras, radars, and LiDARs, etc., has already driven millions of miles without any human intervention [dee17]. Several major car manufacturers including Tesla, GM, Ford, BMW, and Waymo/Google are building and actively testing self-driving cars [TPJR18]. Self-driving cars are becoming widespread, and this trend is likely to continue and intensify.

Correspondence to: Weiqiang Kong, e-mail: wqkong@dlut.edu.cn

The system for self-driving cars is trained over large training data, which is collected under different conditions such as a wide variety of road types and a diverse set of lighting and weather conditions. Every system for self-driving cars is expected to make good choices (to behave correctly for previously-unseen inputs) at any time like a real driver.

The key component of a self-driving car is the perception module controlled by the underlying Deep Neural Network (DNN) [TPJR18]. However, it has been observed that DNN-based software, including the system for self-driving cars, can react in unexpected and incorrect ways to even slight perturbations of their inputs [SZS+14]. Those previously-unseen erroneous behaviors in DNN-driven self-driving cars can lead to dangerous consequences like a fatal collision. Several such real-world cases have already been reported [Goo16, Tes18, Ube18]. This unexpected behavior is likely to result in the unsafe system, or restrict the usage of self-driving cars in safety-critical applications [KBD+17a]. Hence, there is a pressing need for a well-studied method that can provide formal guarantees for the behavior of DNN-driven self-driving cars. Unfortunately, input-output space (i.e., all possible combinations of inputs and outputs) is too large to explore exhaustively, as these cars adjust their behaviors based on the environment measured by different sensors (e.g., camera, infrared obstacle detector, etc.).

One approach to enhancing the safety of self-driving cars is case-based test. For instance, Tian et al. [TPJR18] proposed a systematic testing tool called DeepTest for automatically detecting erroneous behaviors of DNN-driven self-driving cars that can potentially lead to fatal crashes. They leveraged nine different image transformations to automatically generate test cases, and then found thousands of erroneous steering behaviors. The image transformations include changing brightness, changing contrast, translation, scaling, horizontal shearing, rotation, blurring, fog effect, and rain effect. However, despite their remarkable progress, real-world driving conditions involve more than just the nine transformations mentioned above. Another potential approach is simulation-based, but recent research [KP16] has shown that fully autonomous vehicles in simulation would have to be driven hundreds of millions of miles and sometimes hundreds of billions of miles to demonstrate their reliability in terms of fatalities and injuries, which would have to take decades to accomplish. Therefore, although case-based test and simulation are often used to check the performance of autonomous systems, they do not provide sufficient guarantees. This is especially true for safety-critical areas such as autonomous driving, where unsafe incidents are rare and difficult to characterize.

In fact, verifying DNN-driven self-driving cars is a difficult problem. In general, DNNs are large, non-linear, and nonconvex, and verifying even some simple properties about them is an NP-complete problem [KBD+17b]. DNN verification is experimentally beyond the reach of general-purpose tools such as linear programming (LP) solvers or existing satisfiability modulo theories (SMT) solvers [BIL+16, HKWW17, KBD+17a, PT12]. The difficulty in proving properties about self-driving cars is caused by the judgement of predicted behaviors. The DNN-driven self-driving cars take inputs from different sensors such as cameras, light detection and ranging sensors (LiDAR), and IR (infrared) sensors, and output predicted behaviors such as the steering angle, braking, speed, etc., which are needed to control the car safely under current conditions [TPJR18]. Outputs are only predicted by well-trained weights and biases and given activation functions, which makes it difficult to judge predicted behaviors. Moreover, it is challenging to make manual reasoning for such a system as it essentially involves recreating the logic of a human driver.

Past efforts at verifying properties of self-driving cars have got into trouble. Shalev-Shwartz et al. [SSS17] suggested the notion of who is responsible for an accident in a non-deterministic setting. However, there is no automatic formal verification tool that can be used to prove these properties. Roohi et al. [RKW+18] specified the most fundamental policies defined in [SSS17] and presented a simple formal model for self-driving cars. After some simplifications, the safety of this system has already been proven manually, but no automatic formal verification tool supports its dynamics. What's more, they are not aware of any automatic formal verification tool that can be used to specify these properties.

In this paper, we propose a novel and automated technique for the formal verification of steering angle safety for self-driving cars. The technique is based on Deep Learning Verification (DLV) [HKWW17], which is an automated verification tool for safety of image classification neural networks. DLV is based on search for adversarial misclassifications within a given region. That is, if an adversarial example is found, the network is said to be unsafe for the image classification task; and otherwise, if no misclassifications are found for all DNN layers, the network is proved to be safe. However, DLV cannot be used directly to the verification problem of steering angle safety for self-driving cars, since correct steering angles are within a certain range of values rather than a concrete single value like in the case of image classification. To solve this problem, we use neuron coverage [PCYJ17] and slack relationships (introduced in this paper) to turn the steering angle judgement problem into a classification problem.

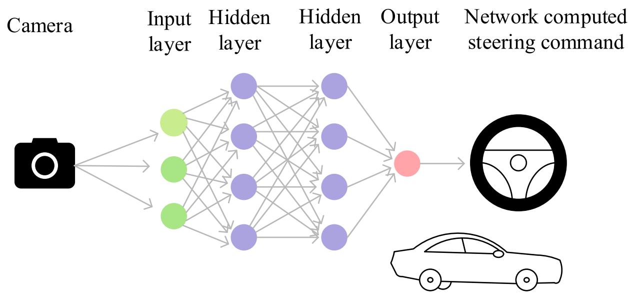


Fig. 1. The DNN for self-driving cars that takes input from camera, and outputs the steering angle

Neuron coverage is the ratio of unique neurons that get activated for a given input image to the total number of neurons in a DNN. Correlation with positive statistical significance suggests that the steering angle increases with increasing neuron coverage and vice versa [TPJR18]. We use the statistical significance between the neuron coverage and the steering angle, and then introduce the neuron coverage as a reference for the same steering angles. The slack relationships are based on the relationship we define for a safe DNN-driven self-driving cars, i.e., the steering angle should not change significantly for minimal changes of the input image. If the neuron coverage of the perturbed image is equal to that of the original image, and the predicted steering angle of the perturbed image satisfies two slack relationships, then the perturbed image and the original image are considered to have the same class. As long as any one of these three conditions is not satisfied, the class of the perturbed image is considered to be different from that of the original image. After transforming the verification problem into a classification problem, DLV is used to tackle the verification of steering angle safety for self-driving cars. We name this technique as SDLV (namely "Steering with DLV").

We evaluate SDLV on the well-trained NVIDIA's end-to-end system for self-driving cars [BTD$^+$16], which has been widely used as the perception module and end-to-end controller for self-driving cars such as Rambo model [Ram17]. The system's network contains 9 layers: a normalization layer, 5 convolutional layers, and 3 fully connected layers. This system takes inputs from a single front-facing center camera and outputs steering commands. Recent results have demonstrated that a well-trained end-to-end system can predict the steering angle with an accuracy close to that of a human driver [BTD$^+$16]. However, the adversarial examples reported by our experiments have shown that such systems (networks) can be instable and result in dangerous behaviors.

Our primary contributions can be summarized as follows. First, we present SDLV, which, to the best of our knowledge, is the first automated verification tool for steering angle safety for self-driving cars; second, we have found adversarial examples of the NVIDIA's self-driving car network by using SDLV, proving the instability of this network.

The rest of the paper is organized as follows. We begin with some background on DNNs and the abstract DLV algorithm in Sect. 2. We then describe the verification method of the steering angle by SDLV in Sect. 3, followed by experiments in Sect. 4. We then introduce the end-to-end system background in Sect. 4.1. Experimental results and the comparison are described in Sects. 4.2 and 4.3, respectively. In Sect. 5, some discussions on SDLV are proposed. Related work is discussed in Sect. 6, and we conclude the paper in Sect. 7.

2. Background

2.1. Deep neural networks for self-driving cars

The key component of a self-driving car is the perception module controlled by the underlying DNN [TPJR18]. The DNN for self-driving cars takes input from different sensors including cameras, light detection and ranging sensor (LiDAR), and IR (infrared) sensors that measure the environment and outputs the steering angle, braking, speed, etc., which are needed to maneuver the car safely under current conditions. In this paper, we focus on the camera input and the steering angle output as shown in Fig. 1.

A typical feed-forward DNN consists of multiple processing layers, which are stacked together to extract different representations of the input [BLPL06]. Each layer of the DNN increasingly abstracts the input, e.g., from raw pixels to semantic concepts. For example, the first few layers of a self-driving car DNN extract low-level features such as edges and directions, the deeper layers identify objects like stop signs and other cars, and the final layer outputs the steering decision (e.g., turning left or right).

Each layer of the DNN is composed of a sequence of individual computing units called neurons. The neurons in adjacent layers are connected through edges. Each edge has a corresponding weight, which is determined by performing a training phase of the DNN based on the labeled training data. Most existing DNNs are trained with gradient descent based on backpropagation [DERW98]. After being trained, a DNN can be used for prediction without changing the weights. For example, a DNN for self-driving car can be used to predict the steering angle based on input images without any further changes to the weights. The value of each neuron is determined by calculating a linear combination of neurons values from the previous layer, and then by applying a non-linear activation function [GBC16]. After that, each neuron sends the value to the subsequent neurons as shown in Fig. 1. In this paper, we use Rectified Linear Unit (ReLU) [NH10] as the main activation function. When a ReLU activation function acts on a neuron, the neuron's value is calculated as the maximum of the linear combination of neurons from the previous layer and 0.

2.2. Deep learning verification (DLV)

Several research projects attempted to build custom tools for formally verifying safety properties of DNNs, such as Reluplex and DLV. We refer interested reader to [Sur18] for a detailed survey. Unfortunately, the current state-of-the-art DNNs, such as DNN-driven self-driving cars, typically contain at least multi-million hidden neurons, while most verification tools work only with small networks (up to a few thousands hidden neurons) [Sur18]. Huang et al. [HKWW17] proposed a novel framework for automated verification of safety of classification decisions made by DNNs, and implemented a software tool called DLV. DLV is based on search for an adversarial misclassification within a given region, and is applicable to large-scale networks. In this paper, we extend DLV to verify steering angle safety for self-driving cars. Thus, we review, in this subsection, the main notions and techniques implemented in DLV for safety verification of a neural network w.r.t. image classification (i.e., classification decision). We refer interested reader to [HKWW17] for the details.

In [HKWW17], safety is defined for an individual classification decision and is parameterized by the class of manipulations and a neighbouring region around a given image. A vector space of images (points) is denoted $\mathbb{R}^{n_k}$. Each layer L_k of a network is associated with an n_k-dimensional vector space $D_{L_k} \subseteq \mathbb{R}^{n_k}$, where each dimension corresponds to a neuron. The mapping $\phi_k : D_{L_{k-1}} \rightarrow D_{L_k}$ denotes an activation function, and $\varphi_k : D_{L_k} \rightarrow D_{L_{k-1}}$ in the opposite direction is used to represent how a manipulated activation of layer L_k affects the activations of layer L_{k-1}. The network is fed an input x (point in D_{L_0}) from its input layer, which is then propagated through the layers by successive application of the activation functions. An activation for point x in layer k is the value of the corresponding function, denoted as $\alpha_{x,k} = \phi_k(\phi_{k-1}(\ldots\phi_1(x))) \in D_{L_k}$ for $k \in \{1, \ldots, n\}$, where $\alpha_{x,0} = x$.

The definition of safety in [HKWW17] for a classification decision (abbreviated safety at a point) is similar to [FFF15]. The same point is that a network $\hat{f}$ approximating human capability f is said to be not robust at an input x if there exists a point y in the region $\eta = \{z \in D_{L_0} \mid \|z - x\| \leq d\}$ such that $\hat{f}(x) \neq \hat{f}(y)$. The point y closest to x is known as an adversarial example. The difference is that [HKWW17] works layer by layer, and therefore will identify such a region η_k, a subspace of D_{L_k}, at each layer L_k, for $k \in \{0, \ldots, n\}$, and successively refine the regions through the deeper layers.

Assumption 1 *For each activation $\alpha_{x,k}$ of point x in layer L_k, the distance between $\alpha_{x,k}$ and activations contained in the region $\eta_k(\alpha_{x,k})$ is very small so that the human observer classifies them into the same class.*

Definition 1 (*General Safety*) Let $\eta_k(\alpha_{x,k})$ be a region in layer L_k of a neural network N such that $\alpha_{x,k} \in \eta_k(\alpha_{x,k})$. We say that N is safe for input x and region $\eta_k(\alpha_{x,k})$, written as $N, \eta_k \models x$, if for all activations $\alpha_{y,k}$ in $\eta_k(\alpha_{x,k})$ we have $\alpha_{y,n} = \alpha_{x,n}$.

A manipulation is an operator that simulates image perturbations such as bad angles, scratches or weather conditions, and its type can be determined by the application environment or the user. But under such manipulations, the classification decisions in a region of images close to it should be invariant.

$$\begin{tikzcd}
N,\eta_0 \models x & N,\eta_1 \models x \arrow[l] \arrow[d] & N,\eta_2 \models x \arrow[l] \arrow[d] & \ldots \arrow[l] & N,\eta_k \models x \arrow[l] \arrow[d] \\
& N,\eta_1,\Delta_1 \models x & N,\eta_2,\Delta_2 \models x \arrow[l] & \ldots \arrow[l] & N,\eta_k,\Delta_k \models x \arrow[l] \arrow[u, bend right, "\Delta_k \text{ is minimal}"']
\end{tikzcd}$$

Fig. 2. DLV refinement framework

For simplicity, DLV works with operators $\delta_k : D_{L_k} \to D_{L_k}$ over the activations in the vector space of layer k, and consider the Euclidean (L^2) and Manhattan (L^1) norms to measure the distance between an input and its perturbation through δ_k. More specifically, applying a manipulation $\delta_k(\alpha_{x,k})$ to an activation $\alpha_{x,k}$ will result in another activation such that the values of some or all dimensions are changed. DLV therefore represent a manipulation as a hyper-rectangle, defined for two activations $\alpha_{x,k}$ and $\alpha_{y,k}$ of layer L_k by $rec(\alpha_{x,k}, \alpha_{y,k}) = \times_{p \in P_k}[min\{\alpha_{x,k}(p), \alpha_{y,k}(p)\}, max\{\alpha_{x,k}(p), \alpha_{y,k}(p)\}]$ [HKWW17]. For an activation $\alpha_{x,k}$ and a set Δ of manipulations, we denote by the polyhedron $rec(\Delta, \alpha_{x,k}) = \cup_{\delta \in \Delta} rec(\alpha_{x,k}, \delta_k(\alpha_{x,k}))$ after applying some manipulation in Δ on $\alpha_{x,k}$. Let Δ_k be the set of all possible manipulations for layer L_k.

Definition 2 (*Safety w.r.t. Manipulations*) Given a neural network N, an input x and a set Δ_k of manipulations, we say that N is safe for input x with respect to the region η_k and manipulations Δ_k, written as $N, \eta_k, \Delta_k \models x$, if a complete tree is generated starting from $\alpha_{x,k}$, each node of the tree has the same classification as $\alpha_{x,k}$ and the limited number of hyper-rectangles generated by adjacent nodes can cover the region $\eta_k(\alpha_{x,k})$.

Definition 3 The functions $\{\eta_0, \eta_1, \ldots, \eta_n\}$ and $\{\psi_1, \ldots, \psi_n\}$ mapping activations to regions are such that

1. $\eta_k(\alpha_{x,k}) \subseteq D_{L_k}$, for $k = 0, \ldots, n$,
2. $\alpha_{x,k} \in \eta_k(\alpha_{x,k})$, for $k = 0, \ldots, n$, and
3. $\eta_{k-1}(\alpha_{x,k-1}) \in \psi_{\eta_k(\alpha_{x,k})}$ for all $k = 1, \ldots, n$.

The main verification idea of DLV is that for a given neural network N and an input x, a series of continuous manipulations Δ_k are performed on $\alpha_{x,k}$ from some layer L_k, and the region $\eta_k(\alpha_{x,k})$ containing $\alpha_{x,k}$ is divided into a limited number of small regions that could cover the region $\eta_k(\alpha_{x,k})$. For each small region, if there is no adversarial example found in all small regions, then refinement operations in layer L_{k+1} are performed. Under the condition that Δ_k is minimal, it can be ensured that the region $\eta_k(\alpha_{x,k})$ satisfies $N, \eta_k \models x$ in layer L_k. Minimal operator here means that no point is different from the classification of $\alpha_{x,k}$ and a manipulation $\delta_k(\alpha_{x,k})$ can be found in a small region. The refinement framework is shown in Fig. 2. The arrows represent the implication relations between the safety notions and are labelled with conditions if needed. The goal of the refinements is to find a chain of implications to justify $N, \eta_0 \models x$.

Definition 4 A manipulation $\delta_{k-1}(\alpha_{y,k-1})$ is refinable in layer L_k if there exist activations and valid manipulations in layer L_k can express $\delta_{k-1}(\alpha_{y,k-1})$. Here, a valid manipulation δ_k means that a corresponding activation $\alpha_{y,k}$ is an interior point of the polyhedron which includes all hyper-rectangles of $\alpha_{y,k}$. Given a neural network N and an input x, the manipulations Δ_k are a refinement by layer of η_{k-1}, Δ_{k-1} and η_k if, for all $\alpha_{y,k-1} \in \eta_{k-1}(\alpha_{z,k-1})$, all its valid manipulations $\delta_{k-1}(\alpha_{y,k-1})$ are refinable in layer L_k.

After developing theoretical analyses (only a part of which was reviewed above), Huang et al. [HKWW17] summarized the following search-based recursive verification procedure. The method is parameterized by the region η_k around a given point and a family of manipulations Δ_k. DLV can start from any layer and propagate analysis into deeper layers. The vector norm to identify the region can be specified by the user and varies by layer. When DLV finds an adversarial example, it will terminate and map it back to the input layer.

Algorithm 1 Given a neural network N and an input x, recursively perform the following steps, starting from some layer $l \geq 0$. Let $k \geq l$ be the current layer under consideration.

1. determine a region η_k such that if $k > l$ then η_k and η_{k-1} satisfy Definition 3;
2. determine a manipulation set Δ_k such that if $k > l$ then Δ_k is a refinement by layer of η_{k-1}, Δ_{k-1} and Δ_k according to Definition 4;

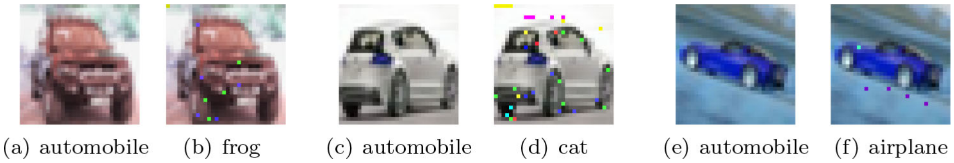

(a) automobile (b) frog (c) automobile (d) cat (e) automobile (f) airplane

Fig. 3. Automobile images (classified correctly) and their perturbed images (classified wrongly)

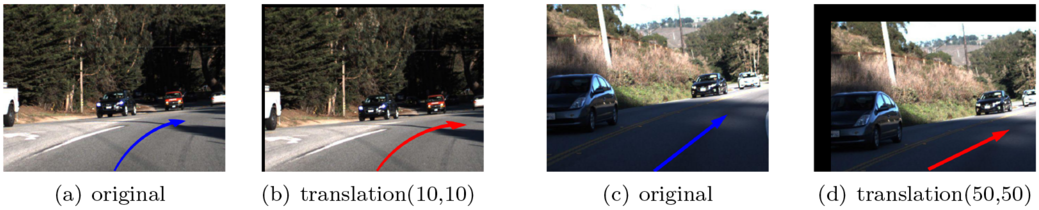

(a) original (b) translation(10,10) (c) original (d) translation(50,50)

Fig. 4. False positives

3. verify whether $N, \eta_k, \Delta_k \models x$,
   (a) if $N, \eta_k, \Delta_k \models x$ then
      i. report that N is safe at x with respect to $\eta_k(\alpha_{x,k})$ and Δ_k, and
      ii. continue to layer $k + 1$;
   (b) if $N, \eta_k, \Delta_k \not\models x$, then report an adversarial example.

Algorithm 1 was implemented by using satisfiability modulo theory (SMT) solvers. The SMT problem is a decision problem for logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality.

3. The verification method

The DLV described in Sect. 2.2 is an efficient method for automated verification of safety of image classification decisions. However, this method cannot be used to verify DNNs for self-driving cars.

When an image classification network maps an image to a class label, each classification decision (i.e., true or false) is unique. For example, Fig. 3 gives adversarial perturbations of automobile images that are misclassified as a frog, cat or airplane by a highly trained state-of-the-art network. Moreover, each image classification decision is easy to judge by comparing with the predetermined manual label. Mapping an image to a steering command in a self-driving car network is similar to the mapping of the image classification network, but the correct steering command for self-driving car is not unique. For example, DeepTest [dee17] reports two false positives in Fig. 4. In each group of images, the steering angle (blue arrow) in the left original image is a manual label, and the steering angle (red arrow) in the right synthetic image is erroneous behavior incorrectly reported by DeepTest [dee17], but the self-driving model's steering command output is indeed safe.

The theory of DLV requires that the original output and the perturbed output must have the same class. However, the perturbed steering angle output can be different from the original output because the safe steering angle command is not unique. Therefore, the DNNs for self-driving cars cannot be verified directly by DLV.

If we want to verify the safety of the steering angle for self-driving cars, we must face and solve the problem of judging the steering angle command predicted by the self-driving model. However, the judgment of the steering angle is complicated and difficult. This is because, under the same road condition, different human drivers take different steering angle commands. Fortunately, we can find that these different and safe steering angles fluctuate in a continuous range. Moreover, the steering angle is an important issue related to life. DNN-based software used for self-driving cars often exhibits incorrect/unexpected turning behaviors, which may lead to dangerous consequences such as fatal collisions [dee17]. Several such real-world cases have already been reported [Goo16, Tes18, Ube18].

A naïve approach is then to express steering label using some mathematical expressions, which is possible because a correct steering decision is within a certain range. However, it is challenging to specify the correct behaviors for a self-driving car system as it essentially involves recreating the logic of a human driver. To solve this problem, we leverage neuron coverage [PCYJ17] and introduce slack relationships between car behaviors. In this paper, we compare whether the neuron coverage of the original image and the perturbed image are equal, and check whether the predicted steering angles of the perturbed image satisfies two slack conditions. If the neuron coverage of the perturbed image is equal to that of the original image, and the predicted steering angle of the perturbed image satisfies two slack conditions, then the perturbed image is considered to be classified the same as the original image. As long as any one of these three conditions is not satisfied, the perturbed image and the original image have different classes.

3.1. Neuron coverage

Neuron coverage was initially proposed by Pei et al. [PCYJ17] for guided differential testing of multiple similar DNNs. It is defined as the ratio of unique neurons that get activated for given an input image to the total number of neurons in a DNN:

$$Neuron\ Coverage = \frac{|Activated\ Neurons|}{|Total\ Neurons|} \tag{1}$$

An individual neuron is considered activated if the neuron's output is larger than a predetermined neuron activation threshold. To better compare with the method in [TPJR18], we also use 0.2 as the neuron activation threshold for our experiments presented in Sects. 4.2 and 4.3.

Neurons may produce different types of output values depending on the type of the corresponding layer, i.e., single and multiple values organized in a multi-dimensional array. Since neurons in the fully connected layer output a single scalar value, their output can be directly compared to the neuron activation threshold. By contrast, neurons in convolutional layers output multidimensional feature maps as each neuron outputs the result of applying a convolutional kernel across the input space [SHR15].

In order to obtain neuron coverage formulas of fully connected layers and convolutional layers, we first define a function h as:

$$h(x, y) = \begin{cases} 1 & \text{x>y} \\ 0 & \text{x≤y} \end{cases} \tag{2}$$

For a DNN N, we use P_k to denote the set of neurons in layer L_k, and $n_k = |P_k|$ is the number of neurons (dimensions) in layer L_k. For neuron $p \in P_k$, the value of its activation on input x is denoted $\alpha_{x,k}(p)$. We use d to denote the neuron activation threshold (i.e., d = 0.2) and NC to denote the neuron coverage. We write A for the set of the layer index k of all fully connected layers. For an input x, the neuron coverage formula for fully connected layers is defined as:

$$NC(x) = \frac{\sum\limits_{k\in A} \sum\limits_{p\in P_k} h(\alpha_{x,k}(p), d)}{\sum\limits_{k\in A} n_k} \tag{3}$$

We write B for the set of the layer index k of all convolutional layers. Assuming that the convolutional layer L_k produces M_k feature maps from the previous layer L_{k-1}, the set of neurons for a any feature map $m \in M_k$ is denoted $P_{k,m}$, and the number of neurons is denoted $n_{k,m} = \left|P_{k,m}\right|$. For an input x, the neuron coverage formula for convolutional layers is defined as:

$$NC(x) = \frac{\sum\limits_{k\in B} \sum\limits_{m\in M_k} h\left(\frac{\sum\limits_{p\in P_{k,m}} \alpha_{x,k}(p)}{n_{k,m}}, d\right)}{\sum\limits_{k\in B} M_k} \tag{4}$$

Tian et al. [TPJR18] analyzed the neuron coverage for a steering direction (left/right) and a steering angle separately. As steering angle is a continuous variable, they checked spearman rank correlation [Spe04] between neuron coverage and steering angle.

Table 1. Relation between neuron coverage and the steering angle

Model	Steering angle Spearman correlation	Steering direction Wilcoxon test	Steering direction Effect size (Cohen's d))
End to end	$-0.25^{(***)1}$	left(+ve)<right(-ve)$^{(***)}$	Large

[1] *** indicates statistical significance with p value $< 2.2 \times 10^{-16}$.

This is a non-parametric measure to compute monotonic association between the two variables [HK11]. Correlation with positive statistical significance suggests that the steering angle increases with increasing neuron coverage and vice versa [TPJR18]. Their experimental results indicate that Spearman correlations for all three models [Cha16, Ram16, Epo16] are statistically significant. We want to use the statistical characteristic between the neuron coverage and the DNN for self-driving cars to incorporate the neuron coverage into the reference of the same steering angle. In this paper, we evaluate SDLV on the NVIDIA's end-to-end system for self-driving cars, so we also conduct statistical investigations on this end-to-end model. We randomly intercept 3000 input images from the test dataset and study the correlation between the neuron coverage and the predicted output steering angle, and obtain table 1. Table 1 shows that the Spearman correlation about the end-to-end model is statistically significant, and the model shows a negative correlation. This result shows that the neuron coverage changes with the changes in output steering angles. At the same time, we use the Wilcoxon nonparametric test to measure the association between neuron coverage and steering direction. The results of the two rightmost columns in table 1 confirm that the neuron coverage changes with the steering direction is statistically significant. These results show that the neuron coverage changes significantly for different input-output pairs. Therefore, we use the same neuron coverage as a reference for the same steering angle.

3.2. Slack relationships

Slack relationship 1. The key insight is that even though it is hard to specify the correct behavior of a self-driving car for every operated image, one can define metamorphic relations [TYCY98] between the car's behaviors across the input image and operated image. For instance, the steering angle of autonomous vehicles should not change significantly for input images with minor changes (i.e. adversarial disturbances). Formally, we use $\hat{\alpha}_{x,n}$ to denote the steering angle of manual label and $\alpha_{x,n}$ to denote the predicted steering angle for the input image x. The operated image y is generated by applying some manipulations Δ to some activations of the input image x, and its corresponding predicted steering angle is denoted by $\alpha_{y,n}$. Then, one may define a simple metamorphic relation $\alpha_{x,n} = \alpha_{y,n}$. If we have $\hat{\alpha}_{x,n} = \alpha_{x,n}$, then ideal metamorphic relation is $\hat{\alpha}_{x,n} = \alpha_{y,n}$.

However, there is usually no single correct steering angle for a given image, that is, a car can safely tolerate small variations. Therefore, there is a trade-off between defining the metamorphic relations very tightly such as the one described above (may result in a large number of false positives) and making the relations more permissive (may lead to many false negatives). In this paper, we strike a balance between these two extremes using the metamorphic relations defined below.

To minimize false positives, we relax our metamorphic relations and allow variations within the error ranges of the original input images. We observe that the set of outputs predicted by a DNN model for the original images, say $\{\alpha_{x_1,n}, \alpha_{x_2,n}, \cdots, \alpha_{x_m,n}\}$, in practice, results in a small but non-trivial number of errors w.r.t. their respective manual labels $(\{\hat{\alpha}_{x_1,n}, \hat{\alpha}_{x_2,n}, \cdots, \hat{\alpha}_{x_m,n}\})$. Such errors are usually measured using Root Mean Squared Error (RMSE), where

$$RMSE_0 = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{\alpha}_{x_i,n} - \alpha_{x_i,n})^2} \tag{5}$$

Based on this property, we redefine a new metamorphic relation as:

$$\left|\hat{\alpha}_{x,n} - \alpha_{y,n}\right| \leq \mu\, RMSE_0 \tag{6}$$

The above equation assumes that the errors produced by a model for the perturbed images as input should be within a range of μ times the $RMSE_0$ produced by the original image set. Here, μ is a configurable parameter that allows us to strike a balance between the false positives and false negatives. $RMSE_0$ is a parameter calculated from a set of predicted outputs obtained by a trained neural network (our analysis target), which does not affect our verification results.

However, not all behaviors that satisfy the first slack relationship Eq. (6) can be correct, as the correct steering angle can safely tolerate small variations. Therefore, we take another different route and introduce the following slack relationship 2.

Slack relationship 2. We only consider minor perturbations where the correct operated output $\alpha_{y,n}$ should not deviate too much from the corresponding input image manual label $\hat{\alpha}_{x,n}$. In general, the deflection amplitude of the left and right sides of the steering angle can be set to 25 degree, which is scaled by 1/25(0.04) and mapped to $[-1, 1]$. Therefore, we further use a filter criteria, defined as follows:

$$\left|\hat{\alpha}_{x,n} - \alpha_{y,n}\right| \leq 0.04\xi \tag{7}$$

where ξ is also a configurable parameter which allows us to strike a balance between the false positives and false negatives.

3.3. Verification

Since there is a positive (or negative) statistical correlation between neuron coverage and steering angle, we regard the equal neuron coverage as a necessary condition for the same type of steering angle. However, it is too rough to use only the equal neuron coverage to deal with the steering angle classification problem, so we also analyze the steering angle based on a set of manual labels and a single manual label. According to the defined relations, the steering angle for a self-driving car should not change significantly for the same input image with minimal changes. On the one hand, the RMSE computed from a set of manual labels constrains the steering angle after disturbance (that is, introduces the first relaxation condition) to balance false positives and false negatives. On the other hand, by constraining a single manual label, it is intended to compensate the possible deviation produced by comparing with a set of manual labels. Therefore, we summarize the above theoretical analyses as a search-based recursive verification procedure given below. Before presenting the verification method, we give the following definition.

Definition 5 Let $NC(x)$ and $NC(y)$ be neuron coverage of an input x and the operated image y, respectively. We say that x and y have the same class, i.e. $\alpha_{x,n} = \alpha_{y,n}$, if $NC(x) = NC(y)$, Eqs. (6), and (7) are satisfied.

We say the neural network N is safe for the input x, if the input x and all the operated image y on Δ have the same classification during the search process. Here, we extend the DLV theory to a theory of verification of the steering angle safety for self-driving cars.

Algorithm 2 *Given a neural network N for self-driving cars and an input x, perform the following steps recursively. Let $k \geq l \geq 0$ be the current layer under consideration.*

1. *determine the configurable parameters μ and ξ in Eq.* (6) *and* (7);
2. *determine a region η_k such that if $k > l$ then η_k and η_{k-1} satisfy Definition* 3;
3. *determine a manipulation set Δ_k such that if $k > l$ then Δ_k is a refinement by layer of η_{k-1}, Δ_{k-1} and Δ_k according to Definition* 4;
4. *determine that the input x and the operated image $\delta(x)$ have the same class for all $\delta \in \Delta_k$ if x and $\delta(x)$ satisfy Definition* 5*. This step converts the steering angle prediction problem for self-driving cars into a problem that can be classified like an image classification network;*
5. *verify whether $N, \eta_k, \Delta_k \models x$,*
   - (a) *if $N, \eta_k, \Delta_k \models x$ then*
     - i. *report that N is safe at x with respect to $\eta_k(\alpha_{x,k})$ and Δ_k, and*
     - ii. *continue to layer $k + 1$;*
   - (b) *if $N, \eta_k, \Delta_k \not\models x$, then report an adversarial example.*

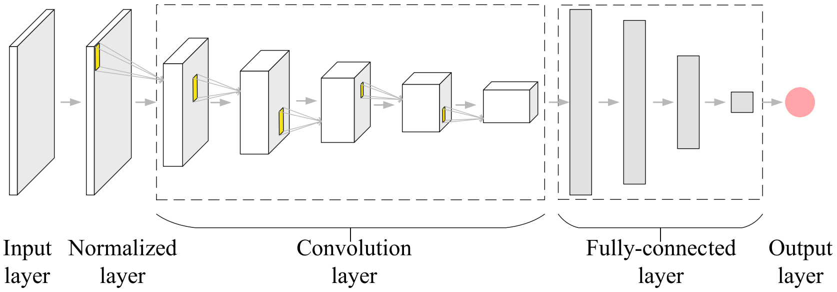


Fig. 5. The network architecture of the end-to-end system for self-driving cars

Our work focuses on verifying the local robustness of the predicted steering angle output in neural networks for self-driving cars. In the verification process, for any given input, the trained neural network can get a steering angle output. Then SDLV starts the operation from a certain layer of the neural network to determine whether the predicted steering angle after the operation is consistent with the original predicted steering angle. If no adversarial examples are found for all layers, the neural network can achieve safety verification of the steering angle. If an adversarial example is found, the neural network is unsafe about this input.

4. Experiments

4.1. Background

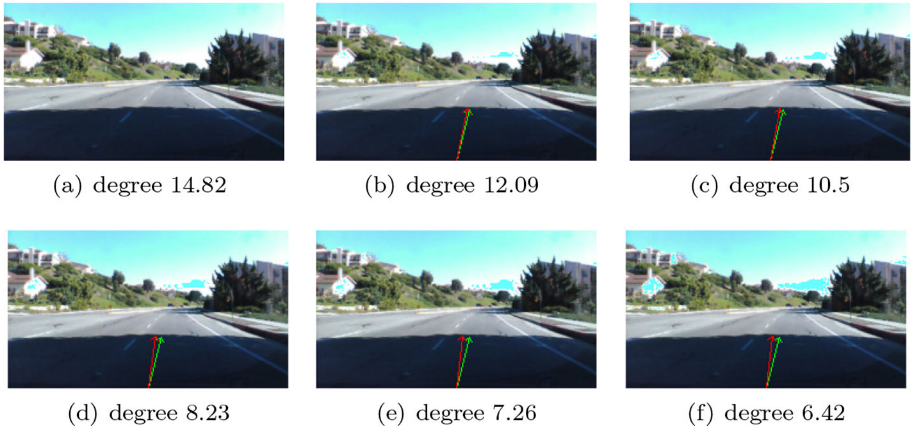
In this paper, we evaluate our technique on the NVIDIA's end-to-end system for self-driving cars [BTD+16]. An end-to-end system for self-driving cars means that the inputs of the system (such as sensors) directly determine the behaviors of the cars (such as throttle, brake, and direction, etc). In this paper, we focus on the camera input and the steering angle output.

The network architecture of the end-to-end system for self-driving cars is shown in Fig. 5. The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. Unlike a fully connected layer where the neurons are connected to all of the neurons in the next layer and multiple connections among different neurons share the different weight, the neurons in a convolution layer are connected only to some of the neurons in the next layer and multiple connections among different neurons share the same weight. Here, we use strided convolutions in the first three convolutional layers with a 2 × 2 stride and a 5 × 5 kernel size and a non-strided convolution with a 3 × 3 kernel size in the last two convolutional layers. With five convolutional layers and three fully connected layers, the output control value can be obtained, which is the steering angle command.

4.2. Experimental results

We implement in Python our proposed approach on the top of DLV [Hua], and name the approach as SDLV (Steering with DLV). The SMT solver we used is Z3 [Z319], which has Python APIs. The neural network of the end-to-end system for self-driving cars is built from a widely used neural networks library Keras [Ker19] with a deep learning package Theano [BTD+16] as its backend. Our experiments are conducted on a desktop computer with 3.2 GHz Intel Core i7 CPU and 32 GB memory.

In [HKWW17], there are two search methods for DLV to verify the safety of image classification networks: DLV on single-path search for the Euclidean norm (L^2) and on multi-path search for the L^1 and L^2 norms. If the checking of points in the region partitioned according to the feature is conducted by following a pre-specified sequential order, this way is called the single-path search. If the checking of points in the region partitioned according to the feature is conducted by exhaustively searching all possible orders, this way is called the multi-path search. In this paper, we also use SDLV on single-path search for the Euclidean norm (L^2) and on multi-path search for the L^1 and L^2 norms to verify the steering angle safety of the end-to-end system for self-driving cars. The network of the end-to-end system uses two representative types of layers: fully connected layers and convolutional layers.



(a) degree 14.82

(b) degree 12.09

(c) degree 10.5

(d) degree 8.23

(e) degree 7.26

(f) degree 6.42

Fig. 6. An illustrative example of modified steering predictions

Given a neural network for self-driving cars, for an input image x, we assume that the output is the correct steering angle. In the following figures, we use the green arrow to denote the correct steering angle. If the predicted steering angle of the operated image does not satisfy the condition in Definition 5, we determine that the steering angle is wrong, which is denoted by a red arrow in the following figures.

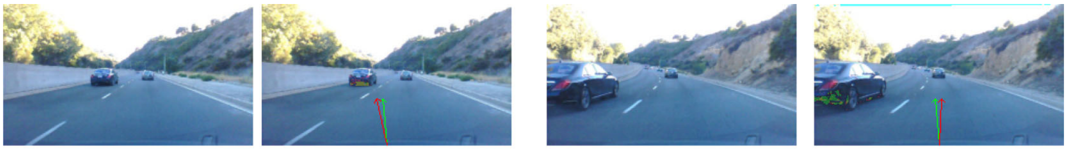
The setting values of the parameters μ and ξ do affect the verification results. The two values we chose is the same as the optimal values reported by DeepTest experiments. In addition, even if the setting of these two values is not really reasonable, in our method, neuron coverage can correct unreasonable results to ensure the correctness of results.

The single-path search. We work with a large size neural network of the end-to-end system for self-driving cars, which we train for more than 6 hours on the driving dataset [Sul]. The inputs to the network are images of size 135 × 240 with three channels. The trained network has more than a million real-valued parameters and includes convolutional layers, flatten layers, dropout layers, fully-connected layers, and a arctan layer. The images are preprocessed to make the value of each pixel within the interval [0, 1].

Given an input image x, we start with layer $k = 1$ and the parameter is set to be at most 150 dimensions (there are 1824 dimensions in layer L_1). All η_k and Δ_k for $k \geq 2$ are computed according to the simple heuristic mentioned in Algorithm 1 and satisfy Definitions 3 and 4. For the region $\eta_0(\alpha_{x,0})$, the activation value for each selected dimension (within $[-1, 1]$) is allowed to change. The set Δ_0 includes manipulations that can change the activation value for a subset of the 150 dimensions, by incrementing or decrementing the value for each dimension by 1. The experimental results show that for most of the examples we can find a class change within 100 dimensional changes in layer L_1, by comparing the number of pixels that have changed (some of them can have less than 20 dimensional changes).

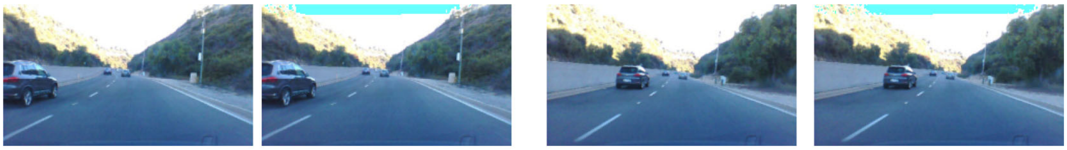
As an illustration of the degree of perturbations that we are investigating, we consider the images in Fig. 6 that correspond to the parameter (μ^2, ξ) set to (2, 2.5), (5, 4), (8, 6), (10, 7), (12, 8) respectively, for layer k = 1. The image (a) is the original image. The images (b)–(d) are obtained by mapping back from the first hidden layer and represent a corresponding point close to the boundary of the corresponding region. In this paper, we set the parameters $\mu^2 > 4$ and $\xi > 6$ to satisfy the condition that the amplitude change between the steering angle after disturbance and the original steering angle about a given input does not exceed 6 degree. This is because when the change of the steering angle amplitude exceeds 6 degree, there is a high possibility that a traffic accident will occur. According to the parameters set above, the relation $N, \eta_1, \Delta_1 \models x$ holds for the images $(b) - (d)$ in Fig. 6, but fails for the last two implying that the images are false steering decisions. Larger number of selected dimensions implies larger region to which we apply manipulations, and, more importantly, suggests a more dramatic change to the knowledge represented by the activations when moving to the boundary of the region.

In Fig. 7 we also give two pairs of original (predicted correctly) and perturbed (predicted wrongly) images. The image on the left is reported as being unsafe for the first layer with 27 dimensional changes (0.0148% of the 1824 dimensions of layer L_1). The one on the right is reported as being unsafe for the first layer with 111 dimensional changes (0.06% of the 1824 dimensions of layer L_1).



(a) degree -4.47 (b) degree -12.02 (c) degree -5 (d) degree 2.67

Fig. 7. Adversarial examples for the end-to-end network for self-driving cars trained on the driving dataset by single-path search



(a) degree -13.11 (b) degree -17.38 (c) degree -15.47 (d) degree -18.68

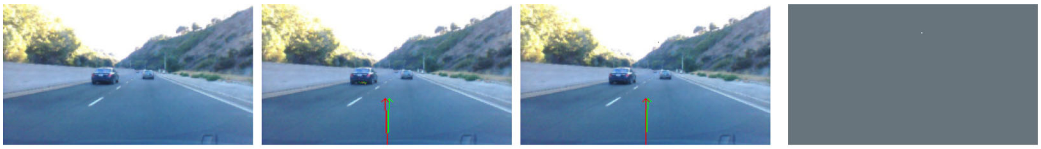
Fig. 8. Cannot find an adversarial example for the right image for 300 dimensions, by single-path search

However, images in Fig. 8 are reported as being safe for 300 dimensional changes of layer L_1. It appears that more complex manipulations, involving more dimensions, are needed in this case to cause a steering decision change.

The multi-path search. The image (c) in Fig. 9 presents the result of a multi-path search on the original input image (a). The image (a)'s steering prediction (approx. 27 min to manipulate) is 2.57 degree (confidence 0.00078) changed into 0.23 degree (see the image (c)), with an L^1 distance of 420 and L^2 distance of 11.225. The image (b) is the result of the single-path search obtained from the image (a). Comparing the image (b) and the image (c), it is easy to see that the steering prediction's change of image (b) is larger than that of the image (c). Therefore, in this case, multi-path search is not as effective as single-path search. Since the multi-path search searches over many different paths, the time it takes to find an adversarial example is longer than the single-path search. This point is also mentioned in [HKWW17]. However, if a more optimized method is proposed in path selection on the multi-path search, its efficiency may be improved, which is also our future work. The fourth image (d) is the difference between the original image (a) and perturbed image (c), and it is not hard to see that the degree of disturbance in the image is very small.

4.3. Comparison

In this section, we compare SDLV with DeepTest [TPJR18], the only (to the best of our knowledge) existing approach to automatically detecting erroneous steering angles of DNN-driven self-driving cars. DeepTest applies image transformations to automatically generate test cases for detecting erroneous steering behaviors, whereas SDLV explores, in a formal manner, a proportion of dimensions in the feature space in the input or hidden layers.



(a) degree 2.57 (b) degree -1.52 (c) degree 0.23 (d) difference

Fig. 9. An adversarial example for the end-to-end network for self-driving cars trained on the driving dataset by multi-path search

Table 2. DeepTest versus SDLV (on a single path search)

	DeepTest for guided search	SDLV		
		$(dim = 50)$	$(dim = 150)$	$(dim = 300)$
L^1	25731	10246.25	16531	38359.6
L^2	159.6	97.08	112.55	199.66
%	19%	37%	52%	73%

Table 3. SDLV based on different parameters (on a single path search)

	SDLV$(dim = 150)$		
	$\mu^2 = 4, \xi = 4$	$\mu^2 = 6, \xi = 6$	$\mu^2 = 4, \xi = 5$
L^1	12162.47	17609.93	14964.22
L^2	97.14	116	104.07
%	63%	42%	57%

Although SDLV on multi-path search can search for an adversarial example that has a smaller distance from the original input than SDLV on single-path search and DeepTest, in terms of the time taken to find an adversarial example, SDLV on multi-path search may take longer than SDLV on single-path and DeepTest, since it searches over many different paths. It means that the multi-path search method puts a burden on the computer and makes batch processing difficult. Therefore, when comparing batches, we choose the single-path method and Deeptest for comparison. However, SDLV's single-path and multi-path methods are just different in the starting point of the search. From a theoretical point of view, it will not bring any impact.

Table 2 gives a comparison of robustness evaluation of DeepTest and SDLV on the driving dataset. From the experimental results in [TPJR18], we observed that setting the input parameter λ to 5 is appropriate because it can balance false positives and false negatives, and can test more adversarial examples. Hence, we use 5 as the input parameter λ for DeepTest, and 5 as the configurable parameters μ^2 (Here, μ^2 is equivalent to λ in [TPJR18].) and ξ for SDLV, and then select a single path search for the first hidden layer. We change the maximum dimension (denoted by dim) according to the values $\{50, 150, 300\}$. SDLV manipulates fewer than the maximum dimension on each input image, and then returns an adversarial example if found. When the search reaches the maximal number of dimension, SDLV reports failure and returns the last perturbed example.

The test set size is 100 images selected randomly. Three statistics are collected as shown in table 2. Statistics include the average L^1 distance and the average L^2 distance between an input image and the returned perturbed image, and the success rate of finding adversary examples. For the case when the success rates are very high, i.e., 19% for DeepTest with λ = 5, and 73% for SDLV with $dim = 300$, DeepTest has smaller average distances than SDLV on both L^1 and L^2 distances. When $dim = 50$ or 150, SDLV has smaller average distances than DeepTest on both L^1 and L^2 distances.

A smaller distance leading to a misclassification may result in a lower rate of transferability [PMG+16], meaning that a misclassification can be harder to witness on another model trained on the same (or a small subset of) dataset [HKWW17]. In fact, DeepTest performs image transformation on the entire input image, while SDLV only operates on some dimensions that may cause misclassification according to the network topology. Therefore, the average L^1 and L^2 distances of DeepTest are larger than that of SDLV with $dim = 50$ or 150. However, when the dimension increases, such as $dim = 300$, SDLV has an advantage over DeepTest.

In order to explore more comprehensive effect of parameter selections on the experimental results, we randomly select 100 images (different from selected images in Table 2) again and add experiments with different parameters on SDLV. We select the intermediate value 150 in table 2 as the dim, and obtain table 3. When we use 4 as the configurable parameters μ^2 and ξ for SDLV, the success rate of finding adversary examples is 63% higher than that of the other two groups (42% and 57%) and the same dim situation in table 2 (52%), but the average L^1 and L^2 distances are the smallest among the four groups. Since the parameter values decrease, the criteria for judging the steering angle classification are stricter, and it is easier and faster to find adversarial examples. Conversely, when the parameter values increases, the criteria for judging the steering angle classification are more relaxed, and it is necessary to search for higher dimensions to find adversarial examples, so the average L^1 and L^2 distances become larger and the success rate decreases. Although more stringent judgement conditions will increase the success rate, it should be noted that the probability of false positives in actual operations will also increase.

Table 4. The impact of each condition on the search for adversarial examples

Condition	The success rate of finding adversarial examples (%)
Neuron coverage	29
Slack relationship 1	43
Slack relationship 2	47
Slack relationship 1 and 2	51

5. Discussions

1. What is the relationship between the variation of neuron coverage and slack relationships? Why neuron coverage is important, and if dropped, how would it affect the results? Why are not Eq. (6) (the first slack relationship defined based on the metamorphic relation) and Eq. (7) (the second slack relationship) sufficient for verification?

In order to illustrate the relationship between neuron coverage and slack relationships, we performed a single conditional constraint experiment on the 100 images involved in table 2 to obtain table 4. We use the intermediate value 150 in table 2 as dim, and 5 as the configurable parameters μ^2 and ξ for SDLV. From the success rate in table 4, it can be seen that among the three conditions, neuron coverage is the most relaxed condition, and slack relationship 2 is the most stringent condition. Under the same conditions, the success rate for slack relationship 1&2 in table 4 (51%) is lower than that (52%) in table 2, which indicates that the neuron coverage plays a certain role in the judgement of the steering angle. In the three conditions, the neuron coverage is not sensitive to small disturbances. However, when the parameters of slack relationship 1 and 2 are selected too loose (for example, the two parameters in table 3 are both set to 6), the success rate can still be kept it at 42%, which indicates that the neuron coverage rate can play a supplementary role. If neuron coverage is dropped, some adversarial examples may be missed and false negative examples will be reported. Although Eqs. (6) and (7) balance the adversarial examples of false positives and false negatives, in experiments we found that permissive slack relationships occasionally miss real adversarial examples in the search process, and false negative examples are reported. In order to make up for the lack of slack relationships, we supplement them with neuron coverage as a reference for a safe steering angle, the sensitive changes in neuron coverage will lead us to discover real adversarial examples in time.

2. Is the reported adversarial example always a true positive guaranteed by the algorithm? In other words, how often the reported example turned out to be not an adversary?

When improper selection of parameters makes slack relationships too strict or too relaxed, some cases of false positives or false negatives will appear in the experiment. However, when calculating the success rate of finding adversarial examples, we found three domain experts to artificially identify the returned adversarial examples in the experimental data. Only when opinions of three domain experts agree, we identify it as an adversarial example.

3. What is the performance number for SDLV vis-a-vis DeepTest. How long does it take to generate an adversary example? How much hit we observed when the dimension is increased? (Ref. table 2)

It takes about 5–25 min for SDLV to generate an adversarial example for an input. Regarding the memory, the computer we use is 32G memory, and the memory usage at runtime is 20.2% (6.3G)–20.4% (6.4G). In the comparative experiment in Sect. 4.3, we randomly selected 100 images as inputs. When the dimensions were 50, 150, and 300, we obtained 37, 52, and 73 adversarial examples respectively. However, only 19 adversarial examples among them were found by DeepTest. Therefore, the success rate of SDLV for finding adversarial examples is greater than that of DeepTest. On the other hand, a smaller L^1 or L^2 distance (leading to an adversarial example) may result in a lower rate of transferability, meaning that an adversarial example can be harder to witness on the neural network model trained on the same (or a small subset of) dataset. Although DeepTest performs image transformation on the entire input images, and SDLV operates only in certain dimensions, it is clear from the data in table 2 that when the dimension increases to 300, the average L^1 distance and the average L^2 distance obtained by SDLV is larger than DeepTest, that is, there are higher transferability (SDLV performs better than DeepTest).

4. What is the scalability and applicability of SDLV? In reality, there will be billions of images. Would we be applying perturbation on all images?

SDLV does not operate on all images. Because SDLV focuses on the robustness of the neural network interior to perturbations rather than the large number of inputs. For any input, the operation is started from a certain layer inside the neural network. We use neuron coverage and slack relationships to analyze the predicted output after the operation. Since different inputs activate different neurons in the neural network, the robustness of the neural network can be verified by looking for multiple inputs that activate different parts of neurons.

6. Related work

An early unconventional test approach for neural networks was proposed in [PCYJ17], which used the new concept of neuron coverage to systematically test DL systems and automatically identify erroneous behaviors without manual labels. In [TPJR18], the authors proposed a systematic testing tool called DeepTest for automatically detecting erroneous behaviors of DNN-driven self-driving cars that can potentially lead to fatal crashes. They leveraged image transformations to simulate different real-world phenomena like camera lens distortions, object movements, different weather conditions, etc., and then checked how robust the self-driving DNNs are to those changes. Image transformations include changing brightness, changing contrast, translation, scaling, horizontal shearing, rotation, blurring, fog effect, and rain effect. The combination of transformations, which is guided by neuron coverage, generate test inputs that maximize the numbers of activated neurons. The authors found thousands of erroneous behaviors under test cases, many of which lead to potentially fatal crashes. In [ZZZ$^+$18], the authors proposed DeepRoad, an unsupervised learning framework to synthesize realistic driving scenes to test inconsistent behaviors of DNN-based autonomous driving systems, and validate online input images to improve the system robustness. Unlike DeepTest, DeepRoad could automatically synthesize a large number of diverse driving scenes without using image transformation rules (such as zooming, cutting, and rotating). They used DeepRoad to detect thousands of inconsistent behaviors on three real-world Udacity autonomous driving models, and effectively verify input images to potentially enhance the robustness of the system.

The authors of [PZA17] proposed a novel approach for verifying the safety of lane change maneuvers, using formalized traffic rules according to the Vienna Convention on Road Traffic. A safe free space is derived by making direct use of traffic rules, and they showed that the ego vehicle is located within this space at any time during the lane change. When a collision occurs, if the self-driving vehicle has respected the traffic rules at all times, they believed that another traffic participant must have violated the rules and should be responsible. They assumed that following vehicles in the target lane only have to maintain a safe distance during the lane change of the ego vehicle once the latter merged into the target lane. They further assumed that following vehicles potentially fully accelerate up to some threshold over the maximum allowed velocity, so it is not possible to change lanes in dense traffic with their approach [NKS19].

There are methods relied on formal and deterministic fundamentals can provide guarantees based on imposed requirements. These methods include reachable sets [PKA19, SHE$^+$17, AD14], runtime verification [KCDK15] and metric-based methods [FSA17], including the Responsibility-Sensitive Safety (RSS) model [SSS17]. In [PKA19], the authors proposed a safety framework that can use formal methods to deal with uncertainty measurements and future behaviors of traffic participants as well as disturbances acting on the ego vehicle, thereby dynamically verifying the safety of each planned trajectory. In [SHE$^+$17], the authors proposed a framework that guides the development process of such a holistic online verification method for an autonomous driving software stack. However, this method had limitations in implementation, especially when facing dynamic scenes. An approach for formally verifying the safety of automated vehicles was proposed in [AD14]. The verification is performed by predicting the set of all possible occupancies of the automated vehicle and other traffic participants on the road. However, specific emergency maneuvers for all other situations cannot be completely stored. In [KCDK15], the authors developed an efficient runtime monitoring algorithm, EgMon, that eagerly checks for violations of desired properties written in future-bounded propositional metric temporal logic. In [FSA17], the authors presented a conceptual framework by means of a metamodel to define a component for safety monitoring - a Safety Supervisor - as an instantiation of the metamodel. In [SSS17], the authors proposed a formal model that covers all the important ingredients of self-driving car: sense, plan and act. The formal model shows how safety and scalability are pieced together into an autonomous vehicles program that society can accept and is scalable in the sense of supporting millions of cars driving anywhere in the developed countries. On the safety front, they introduced a model called RSS, which formalizes an interpretation of Duty of Care from tort law. RSS is a rigorous mathematical model that formalizes the laws that apply to self-driving cars. Moreover, they described a design of system that adheres to our safety assurance requirements and is scalable to millions of cars. However, some of the approaches are tailored to a specific software (e.g., a certain trajectory planner) and cannot be bundled with other software components or approaches [SEBD20].

To the best of our knowledge, SDLV is the first formal tool that can automatically verify steering angle safety of self-driving cars.

7. Conclusion and future work

In this paper, we presented SDLV, an automated verification tool for steering angle safety for self-driving cars. We leveraged neuron coverage and slack relationships to transform a judgement problem of predicted behaviors into an image classification problem, which can then be handled by DLV. Our case study on NVIDIA's architecture for self-driving cars shows the possible instability of its trained neural network.

Regarding future work, we would like to enhance the scalability of SDLV. Specifically, we will explore better predicted behaviors analysis for the safety verification of steering angle by taking other realistic factors such as braking, accelerated speed, and possibly traffic regulations etc., into account. In SDLV on the multi-path search, we will explore more optimized method in path selection to improve its effect. In addition, we would like to provide better interpretability by exploring the relationship between disturbance and steering angle, which we believe is an interesting and challenging work.

Acknowledgements

The authors really appreciate the comments and suggestions of the anonymous referees, which were highly insightful and enabled us to greatly improve the quality of our manuscript. This work is supported by the Fundamental Research Funds for the Central Universities (DUT20TD107) and NTT DATA Automobiligence Research Center.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

References

[AD14] Althoff M, Dolan John M (2014) Online verification of automated road vehicles using reachability analysis. IEEE Trans Robot 30(4):903–918

[BIL+16] Bastani O, Ioannou Y, Lampropoulos L, Vytiniotis D, Nori AV, Criminisi A (2016) Measuring neural net robustness with constraints. In: Advances in neural information processing systems 29: annual conference on neural information processing systems 2016, December 5–10, 2016, Barcelona, Spain, pp 2613–2621

[BLPL06] Bengio Y, Lamblin P, Popovici D, Larochelle H (2006) Greedy layer-wise training of deep networks. In: Advances in neural information processing systems 19, proceedings of the twentieth annual conference on neural information processing systems, Vancouver, British Columbia, Canada, December 4–7, 2006, pp 153–160

[BTD+16] Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp B, Goyal P, Jackel LD, Monfort M, Muller U, Zhang J, Zhang X, Zhao J, Zieba K (2016) End to end learning for self-driving cars. CoRR, arXiv:1604.07316

[Cha16] Chauffeur model (2016)

[dee17] The numbers dont lie: self-driving cars are getting good (2017)

[DERW98] Hinton GE, Rumelhart DE, Williams RJ (1998) Learning representations by back-propagating errors. In: Cognitive modeling, 1998

[Epo16] Epoch model (2016)

[FFF15] Fawzi A, Fawzi O, Frossard P (2015) Analysis of classifiers' robustness to adversarial perturbations. CoRR, arXiv:1502.02590

[FSA17] Feth P, Schneider D, Adler R (2017) A conceptual safety supervisor definition and evaluation framework for autonomous systems. In: Computer safety, reliability, and security—36th international conference, SAFECOMP 2017, Trento, Italy, September 13–15, 2017, proceedings, pp 135–148

[GBC16] Goodfellow IJ, Bengio Y, Courville AC (2016) Deep learning. Adaptive computation and machine learning. MIT Press, Berlin

[Goo16] Google's self-driving car caused its first crash (2016)

[HK11] Hauke J, Kossowski T (2011) Comparison of values of Pearsons and Spearmans correlation coefficients on the same sets of data. In: Quaestiones geographicae, p 87

[HKWW17] Huang X, Kwiatkowska M, Wang S, Wu M (2017) Safety verification of deep neural networks. In: Computer aided verification—29th international conference, CAV 2017, Heidelberg, Germany, July 24–28, 2017, proceedings, part I, pp 3–29

[Hua] Huang X

[KBD+17a] Katz G, Barrett CW, Dill DL, Julian K, Kochenderfer MJ (2017) Reluplex: an efficient SMT solver for verifying deep neural networks. In: Computer aided verification—29th international conference, CAV 2017, Heidelberg, Germany, July 24–28, 2017, proceedings, part I, pp 97–117

[KBD+17b] Katz G, Barrett CW, Dill DL, Julian K, Kochenderfer MJ (2017) Reluplex: an efficient SMT solver for verifying deep neural networks. CoRR, arXiv:1702.01135

[KCDK15] Kane A, Chowdhury O, Datta A, Koopman P (2015) A case study on runtime monitoring of an autonomous research vehicle (ARV) system. In: Runtime verification–6th international conference, RV 2015 Vienna, Austria, September 22–25, 2015. Proceedings, pp 102–117

[Ker19] Keras (2019). https://keras.io

[KP16] Kalra N, Paddock SM (2016) Driving to safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability?. In: Transportation research part A: policy and practice, vol 94, p 182C193

[NH10] Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10), June 21–24, 2010, Haifa, Israel, pp 807–814

[NKS19] Naumann M, Königshof H, Stiller C (2019) Provably safe and smooth lane changes in mixed trafic. In: 2019 IEEE intelligent transportation systems conference, ITSC 2019, Auckland, New Zealand, October 27–30, 2019, pp 1832–1837

[PCYJ17] Pei K, Cao Y, Yang J, Jana S (2017) Deepxplore: automated whitebox testing of deep learning systems. In: Proceedings of the 26th symposium on operating systems principles, Shanghai, China, October 28–31, 2017, pp 1–18

[PKA19] Pek C, Koschi M, Althoff M (2019) An online verification framework for motion planning of self-driving vehicles with safety guarantees. In: AAET

[PMG+16] Papernot N, McDaniel PD, Goodfellow IJ, Jha S, Celik ZB, Swami A (2016) Practical black-box attacks against deep learning systems using adversarial examples. CoRR, arXiv:1602.02697

[PT12] Pulina L, Tacchella A (2012) Challenging SMT solvers to verify neural networks. AI Commun 25(2):117–135

[PZA17] Pek C, Zahn P, Althoff M (2017) Verifying the safety of lane change maneuvers of self-driving vehicles based on formalized traffic rules. In: IEEE intelligent vehicles symposium, IV 2017, Los Angeles, CA, USA, June 11–14, 2017, pp 1477–1483

[Ram16] Rambo model (2016)

[Ram17] Rambo model (2017). https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/rambo

[RKW+18] Roohi N, Kaur R, Weimer J, Sokolsky O, Lee I (2018) Self-driving vehicle verification towards a benchmark. CoRR, arXiv:1806.08810

[SEBD20] Stahl T, Eicher M, Betz J, Diermeyer F (2020) Online verification concept for autonomous vehicles—illustrative study for a trajectory planning module. CoRR, arXiv:2005.07740

[SHE+17] Schürmann B, Hess D, Eilbrecht J, Stursberg O, Köster F, Althoff M (2017) Ensuring drivability of planned motions using formal methods. In: 20th IEEE international conference on intelligent transportation systems, ITSC 2017, Yokohama, Japan, October 16–19, 2017, pp 1–8

[SHR15] Rishi K, Samer H, Chris R (2015) Using convolutional neural networks for image recognition. Technical report, 2015

[Spe04] Spearman C (1904) The proof and measurement of association between two things. Am J Psychol 72C101

[SSS17] Shalev-Shwartz S, Shammah S, Shashua A (2017) On a formal model of safe and scalable self-driving cars. CoRR, arXiv:1708.06374

[Sul] SullyChen. Driving dataset

[Sur18] Safety and trustworthiness of deep neural networks: a survey. CoRR, arXiv:1812.08342, 2018. Withdrawn

[SZS+14] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow IJ, Fergus R (2014) Intriguing properties of neural networks. In: 2nd international conference on learning representations, ICLR 2014, Banff, AB, Canada, April 14–16, 2014, conference track proceedings

[Tes18] Tesla says vehicle in deadly crash was on autopilot (2018)

[TPJR18] Tian Y, Pei K, Jana S, Ray B (2018) Deeptest: automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th international conference on software engineering, ICSE 2018, Gothenburg, Sweden, May 27–June 03, 2018, pp 303–314

[TYCY98] Cheung SC, Chen TY, Yiu SM (1998) Metamorphic testing: a new approach for generating next test cases. Technical report, technical report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong

[Ube18] Fatal car crash involving a self-driving uber shows there are major flaws in the software, hardware, and testing procedures involving autonomous vehicles (2018)

[Z319] Z3 (2019). http://rise4fun.com/z3

[ZZZ+18] Zhang M, Zhang Y, Zhang L, Liu C, Khurshid S (2018) Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In: Proceedings of the 33rd ACM/IEEE international conference on automated software engineering, ASE 2018, Montpellier, France, September 3–7, 2018, pp 132–142

Received 10 September 2020

Accepted in revised form 30 January 2021 by Zhiming Liu, Xiaoping Chen, Ji Wang and Jim Woodcock

Published online 4 March 2021