Special Initiative "Autonomous Systems Design" Paper

Continuous Safety Verification of Neural Networks

Chih-Hong Cheng✉*, Rongjie Yan✉†‡
*DENSO AUTOMOTIVE Deutschland GmbH, Eching, Germany
† State Key Laboratory of Computer Science, ISCAS, Beijing, China
‡ University of Chinese Academy of Sciences, Beijing, China
Email: c.cheng@eu.denso.com, yrj@ios.ac.cn

2021 Design, Automation & Test in Europe Conference & Exhibition (DATE) | 978-3-9819263-5-4/21/$31.00 ©2021110.23919/DATE51398.2021.9473994

***Abstract*—Deploying deep neural networks (DNNs) as core functions in autonomous driving creates unique verification and validation challenges. In particular, the continuous engineering paradigm of gradually perfecting a DNN-based perception can make the previously established result of safety verification no longer valid. This can occur either due to the newly encountered examples (i.e., input domain enlargement) inside the Operational Design Domain or due to the subsequent parameter fine-tuning activities of a DNN. This paper considers approaches to transfer results established in the previous DNN safety verification problem to the modified problem setting. By considering the reuse of state abstractions, network abstractions, and Lipschitz constants, we develop several sufficient conditions that only require formally analyzing a small part of the DNN in the new problem. The overall concept is evaluated in a $1/10$-scaled vehicle that equips a DNN controller to determine the visual waypoint from the perceived image.**

***Index Terms*—DNN, safety, formal verification, continuous engineering**

I. Introduction

Deep neural networks (DNNs) have been widely adopted in automated driving, with proven-in-use applications from perception to assisting complex decision making. Deploying autonomous driving functionalities in the open environment creates substantial challenges in the underlying AI (artificial intelligence)/ML (machine learning) component, partly due to the safety-critical nature and partly due to the encountering of "black swans", i.e., scenarios that were not considered in the design time. Recently developed safety guidelines in automated driving such as ISO TR4804 or UL4600 explicitly suggest the monitoring of abnormal cases in operation, either in an online or an offline fashion. When encountering these cases, the involved DNN component should be improved accordingly. The underlying rationale is to admit the imperfection of the initially engineered system while targeting a *continuous improvement*. One natural question that arises is regarding the huge computational efforts in *formally verifying* a DNN: to what extent can results in formal verification of previous models be reused?

In this paper, we consider the problem of *formal DNN safety verification under continuous engineering (fine-tuning)*. Motivated by concrete issues in automated driving, the problem is related to migrating the result of one verification problem to another, with two problems differing in two aspects.

- Due to the discovery of *black swans* (more precisely speaking, out-of-distribution data points) in run time, the *input domain* for verification will be enlarged. Here we take the approach of abstraction-based monitoring [1], [2] where the abstraction D_{in} capturing the in-distribution data will be enlarged to $D_{in} \cup \Delta_{in}$, due to the newly discovered data $d \in \Delta_{in}$ falling outside D_{in}.
- The trained parameters (weights, bias) between two neural networks may only differ slightly, due to their fine-tunings (i.e., the new model is further tuned from the old model with a very small learning rate such as 10^{-3}).

To enable continuous verification, one premise is to have reasonable assumptions on proof artifacts reused from the old verification problem to the new one. Concretely, we consider reusing *state abstractions*, *network abstractions*, and *Lipschitz constants*. State abstractions are sound over-approximations of all possible reachable states created via layer-wise reasoning methods [3]–[6]. Network abstraction [7] refers to property-directed abstraction on the structure of a given neural network. Finally, Lipschitz constants are conservative bounds on the degree of output change subject to input change.

The key contribution of this paper is the establishment of several sufficient conditions utilizing state abstractions, Lipschitz constants, and network abstractions, in order to avoid complete safety verification on the new problem. The satisfaction of sufficient conditions requires checking one or more substantially simpler problems involving local property reasoning over part of the new network. To reuse state abstraction $S_1, S_2, \ldots, S_n$ over intermediate layers, one can take advantage of the precision increase in exact methods or abstraction methods with refinement capabilities to *locally check* if the abstract state S_{i-1} before layer i, after passing the computation of the layer, can be contained in the abstract state set S_i. The locality of subproblems makes the verification pipeline implementable in a parallelized fashion. Reusing Lipschitz constants requires that the worst-case estimate of output change subject to domain enlargement does not influence safety. Lastly, reusing network abstraction requires that the new network can also be transformed into the same abstraction generated from the previous network.

For initial validation, we use a $1/10$ scale vehicle platform that performs autonomous lane following using DNN-based

This research is part of FOCETA project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956123. This work has been partly funded by Key Research Program of Frontier Sciences, CAS, under Grant No. QYZDJ-SSW-JSC036, the CAS-INRIA major project under No. 171311KYSB20170027.

visual perception. The incremental verification, due to only checking conditions of substantially simpler problems, leads to a significant performance gain. The required execution time to prove the safety property in the new problem can be as few as 0.16% of the original problem-solving time, thanks to reusing proof artifacts. Despite encouraging outcomes in our initial evaluation, the research also reveals many interesting yet challenging questions to be further explored.

The rest of the paper is structured as follows. After highlighting related works in Section II, Section III provides basic definitions of DNN safety verification and defines the variation under continuous engineering. Section IV presents the main technical results on several sufficient conditions for the safety property to hold in the new problem. We explain our experimental setup in Section V and conclude with future work in Section VI. Due to space limits, propositions with intuitively simple proofs are omitted.

II. Related work

The recent adoption of DNNs in safety-critical applications such as autonomous driving, flight control, and medical diagnostics leads to fruitful results in developing specialized verification and validation techniques for DNNs [8]. For formal verification of safety properties, DNNs using piecewise linear activation functions such as ReLU or Leaky ReLU are essentially 0-1 integer programs and can be solved with satisfiability modulo theories (SMT) [9]–[11] or mixed-integer linear programming (MILP) [12]–[14] techniques. Such methods could provide sound and complete answers but fail to verify large scale DNNs. To improve the scalability of the solver, various sound approximation methods such as symbolic interval [3], zonotope [4], polyhedron [5], and star set [6] have been adopted to check various safety properties. The approximations provide layered state abstraction and are sound with respect to the given safety properties. To improve the precision of abstraction-based methods, further heuristics are considered to refine specific abstractions (e.g. [15]). Lipschitz continuity is used in specialized DNN training schemes to intentionally increase the robustness [16], while also being adopted for safety verification of DNNs. For example, the set of possible output values can be computed with Lipschitz constant and the given set of inputs, thereby allowing the examination of safety properties [17]. Due to the importance of Lipschitz constants in understanding the characteristics of DNNs, several recent results [18], [19] focus on the accurate estimation of Lipschitz constants. Additional to the aforementioned methods, the structure of DNNs can also be abstracted [7] such that both exact and approximation methods could be applied. When the false positive happens, refinement over the structure is required. Lastly, an interesting problem related to ours is a recent work to check the difference of two DNNs [20], which provides formal guarantees for the relationship between two networks with forward interval analysis and backward refinement.

Despite tremendous efforts targeting to formally verify DNNs, to the best of our knowledge, all of the existing results do not consider a continuous verification setup similar to ours. Our key focus is on reusing the proof artifacts in previous verification tasks to avoid re-proving everything from scratch. This is because, from our practical experience, DNNs used in autonomous driving are extremely complex; the required time for completing the verification task can be enormous, and it is a realistic expectation to encounter multiple domain enlargement and fine-tuning activities. To this end, our work well complements all existing works, and results in formal safety verification of DNNs can be reconsidered under a continuous verification setting.

III. Formulation

A. DNN and Safety Verification

A feed-forward DNN model consists of an input layer, multiple hidden layers and an output layer. We consider only DNNs after training, i.e., all associated parameters related to the model (weights, bias) are fixed. With fixed parameters, a DNN model is a function $f : X \to Y$, where X is the input domain, and Y is the output domain. The model f is built out of a sequence of functions $f := g_n \otimes g_{n-1} \otimes \ldots \otimes g_2 \otimes g_1$, where n is the number of layers in the DNN, and g_k $(k = 1, \ldots, n)$ is the function transformation in the k-th layer. Given an input $x \in X$, the computation of f is $f(x) = g_n(g_{n-1}(\ldots(g_2(g_1(x)))\ldots))$, i.e., to perform functional composition over $g_1, \ldots, g_n$. The internal transformation g_k can be viewed as another function composition where the input of g_k is first linearly transformed, followed by a nonlinear computation using functions such as ReLU (Rectified Linear Unit), Leaky ReLU or sigmoid.

Given a DNN f, ***safety verification*** considered in this paper is formulated as follows: Let $D_{in} \subseteq X$ be the set of input values to be verified, and $D_{out} \subseteq Y$ be the set of safe output values. The safety verification problem checks that given f, D_{in} and D_{out}, whether the property $\phi^f_{D_{in},D_{out}}$ holds, where

$$\phi^f_{D_{in},D_{out}} := \forall x \in D_{in} : f(x) \in D_{out}$$

B. DNN Safety Verification under Continuous Engineering

We consider the following changes in the process of continuous improvement, allowing us to define the problem of continuous safety verification.

- **(Domain enlargement)** The domain of input may be enlarged (due to the need to reconsider additional input values). Given $D_{in} \subseteq X$, we use $D_{in} \cup \Delta_{in} \subseteq X$ to represent the new set of input states to be verified against the safety property.
- **(Network parameter adjustment)** The parameters of a DNN can be changed via further training. We use the notation $f' := g'_n \otimes g'_{n-1} \otimes g'_2 \otimes g'_1$ to indicate the model being further tuned from f.

Problem Statement 1. *Given a neural network model f, a modified model f', as well as D_{in}, Δ_{in}, and D_{out}, the problem of* ***Safety Verification between Two Versions (SVbTV)***

asks the following: Provided that $\phi^{f}_{D_{in},D_{out}}$ *holds, check if* $\phi^{f'}_{D_{in}\cup\Delta_{in},D_{out}}$ *holds as well.*

One sub-case appears when $\Delta_{in} = \emptyset$, i.e., we consider only the change of network parameters. This sub-case is handled in our general framework. Yet the other sub-case appears when f' and f are the same, i.e., we consider the same network under an enlarged input set for safety verification.

Problem Statement 2. *Given a neural network model* f *as well as* D_{in}, Δ_{in}, *and* D_{out}, *the problem of* ***Safety Verification under Domain Change (SVuDC)*** *asks the following: Provided that* $\phi^{f}_{D_{in},D_{out}}$ *holds, check if* $\phi^{f}_{D_{in}\cup\Delta_{in},D_{out}}$ *holds as well.*

IV. Continuous verification

To address SVbTV or SVuDC problems, it is important to have realistic assumptions in terms of how the proof over the verified property $\phi^{f}_{D_{in},D_{out}}$ is stored for reuse. Without any proof reusing, checking the property $\phi^{f'}_{D_{in}\cup\Delta_{in},D_{out}}$ needs to completely restart from scratch. In this paper, we assume that the original DNN $f := g_n \otimes \ldots \otimes g_1$ has been verified to satisfy property $\phi^{f}_{D_{in},D_{out}}$, with the proof artifacts available in one or more of the following categories:

- **Lipschitz constant** of the original DNN f. Precisely, a Lipschitz constant ℓ is a positive real number such that
$$|f(x_1) - f(x_2)| \leq \ell\,|x_1 - x_2| \quad \forall x_1, x_2 \in X \qquad (1)$$
- **State abstractions** $S_1, \ldots, S_n$ over layers to establish the safety proof, where
  - $\forall x \in D_{in}$, $g_1(x) \in S_1$,
  - $\forall i \in \{1, \ldots, n-1\}, \forall x_i \in S_i : g_{i+1}(x_i) \in S_{i+1}$, and
  - $S_n \subseteq D_{out}$.
- **Network abstraction** $\hat{f}$ of the original DNN f where $\forall x \in D_{in}$, $f(x) \in \{\hat{f}(x) \mid x \in D_{in}\}$, and we use $f \xrightarrow{D_{in}} \hat{f}$ to represent the the relation between f and $\hat{f}$. Safety verification utilizing network abstraction techniques is based on establishing the safety proof that $\{\hat{f}(x) \mid x \in D_{in}\} \subseteq D_{out}$, where $\hat{f}$ is a structurally simpler network to be verified against.

As stated in previous sections, there are many verification methods to derive Lipschitz constants or to derive various forms of state or network abstractions, and it is beyond the scope of this paper. Recall that our goal is to enable *proof reuse* subject to the verification setting.

A. Solving SVuDC

The following results demonstrate that the SVuDC problem can sometimes be solved with a local problem that involves constraint solving of two layers.

Proposition 1. *[Proof reuse at layers 1 and 2] Given state abstractions* $S_1, \ldots, S_n$ *for establishing the proof of* $\phi^{f}_{D_{in},D_{out}}$. *If* $\forall x \in D_{in} \cup \Delta_{in}$, $g_2(g_1(x)) \in S_2$, *then property* $\phi^{f}_{D_{in}\cup\Delta_{in},D_{out}}$ *also holds.*

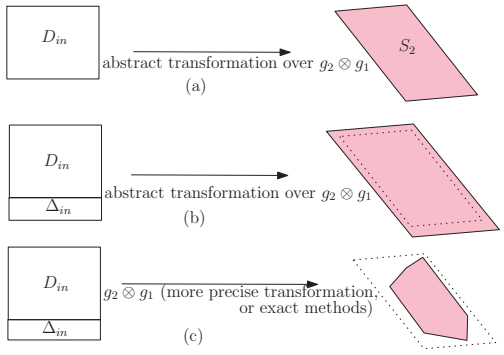

Fig. 1. Insight of Proposition 1

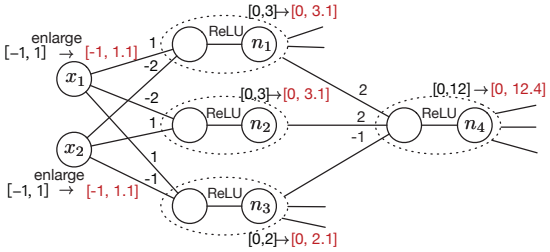

Fig. 2. A simple DNN for applying Proposition 1

Proof. Recall that for the state abstractions $S_1, \ldots, S_n$, they by definition satisfy $\forall i \in \{2, \ldots, n-1\}, \forall x_i \in S_i : g_i(x_i) \in S_{i+1}$ and $S_n \subseteq D_{out}$. This means that any state in S_2, after passing the rest of DNN, leads to an output that is contained in D_{out}. Therefore, so long as $\forall x \in D_{in} \cup \Delta_{in}$, $g_2(g_1(x)) \in S_2$, then $\forall x \in D_{in} \cup \Delta_{in}$, one derives $f(x) \in D_{out}$. □

To utilize the above proposition, the key is to prove that $\forall x \in D_{in} \cup \Delta_{in} : g_2(g_1(x)) \in S_2$, i.e., all inputs in the enlarged set, after passing g_2, can be captured using S_2. Illustrated in Figure 1, using the same abstract transformation over $D_{in} \cup \Delta_{in}$ shall generate an abstract state set larger than S_2 (Figure 1-b), making it impossible to reuse the proof. However, the set of actual reachable values after transformation can be smaller, similar to the one illustrated in Figure 1-c. This creates the potential to use methods with higher precision. For example, one can use *exact verification methods* that encode $\forall x \in D_{in} \cup \Delta_{in} : g_2(g_1(x)) \in S_2$ as constraints. Note that here the verification method only solves a substantially smaller problem that encodes non-linearity in the first two layers.[1]

(Example) Consider a DNN shown in Figure 2 with two inputs and multiple hidden layers. Here we only visualize part of the network to explain the concept. The intervals in black show the intervals of inputs as well as the result of abstract interpretation[2]; for neuron n_4 its value is bounded by $[0, 12]$. When the input domain is increased from $[-1, 1] \times [-1, 1]$

[1]Proposition 1 involves the computation of two hidden layers rather than one hidden layer; this is based on an observation that existing sound methods using abstract interpretation can lose precision after passing two nonlinear layers, thereby creating space for local solving using exact methods or abstraction-refinement techniques.

[2]Here we use boxed abstraction to ease the discussion, but in our evaluation, other types abstract transformers with better precision are used.

to $[-1, 1.1] \times [-1, 1.1]$, the result of abstract interpretation (shown in Figure 2 with red texts) indicates that n_4 is conservatively bounded by $[0, 12.4]$. If one wishes to reuse the proof, the condition in Proposition 1 requires that n_4 is bounded by $[0, 12]$. The condition can be encoded (Equation 2) into a mixed integer programming problem, where the nonlinearity of ReLU can be encoded using big-M approaches [12]–[14]. In this example, exact approaches indicate that the maximum possible value for n_4 equals 6.2. As $6.2 < 12$, the safety property also holds in the enlarged domain.

$$\begin{aligned} -1 \leq x_1 \leq 1.1 \\ -1 \leq x_2 \leq 1.1 \\ n_1 = \mathsf{ReLU}(x_1 - 2x_2) \\ n_2 = \mathsf{ReLU}(-2x_1 + x_2) \\ n_3 = \mathsf{ReLU}(x_1 - x_2) \\ n_4 = \mathsf{ReLU}(2n_1 + 2n_2 - 1n_3) \\ n_4 \geq 12 \end{aligned} \tag{2}$$

When Proposition 1 is not applicable, one may proceed with building new sets of abstractions in a layer-wise fashion. During the construction, one can check in parallel if the new state abstraction S'_j, after passing to the next layer using exact methods encoding g_{j+1}, falls inside the originally created abstraction S_{j+1}. When such a situation occurs, safety is immediately guaranteed in the enlarged domain, as for every $x_{j+1} \in S_{j+1}$, the previous proof artifact ensures that $g_n(g_{n-1} \dots (g_{j+2}(x_{j+1}))) \in D_{out}$.

Proposition 2. *[Proof reuse at layer $j+1$] Given state abstractions $S_1, \dots, S_n$ for establishing the proof of $\phi^f_{D_{in}, D_{out}}$. The property $\phi^f_{D_{in} \cup \Delta_{in}, D_{out}}$ also holds if there exists $j \in \{2, \dots n-1\}$ and $S'_1, \dots, S'_{j-1}$ where*

- $\forall x \in D_{in} \cup \Delta_{in}$, $g_1(x) \in S'_1$,
- $\forall i \in \{1, \dots, j-1\}, \forall x_i \in S'_i : g_{i+1}(x_i) \in S'_{i+1}$, *and*
- $\forall x_j \in S'_j : g_{j+1}(x_j) \in S_{j+1}$.

Our second line of attacks utilizes the fact that the Lipschitz constant provides a conservative estimation on the new set of output values upon input domain enlargement. This enables another sufficient method to only examine the relation between S_n and D_{out}.

Proposition 3. *[Lipschitz-based proof reuse] Given state abstractions $S_1, \dots, S_n$ for establishing the proof of $\phi^f_{D_{in}, D_{out}}$, and given the Lipschitz constant ℓ of DNN f over X. Let κ be a constant where for any input $x_1 \in \Delta_{in}$, its distance to the nearest point $x_2 \in D_{in}$ is bounded by κ. Then $\phi^f_{D_{in} \cup \Delta_{in}, D_{out}}$ holds when the following conditions hold:*

$$\forall \hat{s} \in Y, s \in S_n \subseteq Y : |\hat{s} - s| \leq \ell\kappa \rightarrow \hat{s} \in D_{out}$$

Proof. Given $x_1 \in \Delta_{in}$, following the definition of κ, there exists $x_2 \in D_{in}$ such that $|x_1 - x_2| \leq \kappa$. Then consider the difference between $f(x_1)$ and $f(x_2)$. The Lipschitz constant ℓ over the complete input domain X ensures the following:

$$|f(x_1) - f(x_2)| \leq \ell \, |x_1 - x_2| \leq \ell\kappa$$

As S_n contains all $f(x_2)$ where $x_2 \in D_{in}$, the set $\hat{S}_n$ by enlarging S_n with all points closer than $\ell\kappa$, i.e., $\hat{S}_n := \{\hat{s} \mid \forall s \in S_n : |\hat{s} - s| \leq \ell\kappa\}$, contains the set $\{f(x_1) \mid x_1 \in \Delta_{in}\}$. So long if any point of $\hat{S}_n$ is also in D_{out} (i.e., the condition in the proposition), the property $\phi^f_{D_{in} \cup \Delta_{in}, D_{out}}$ also holds due to the following:

$$\{f(x_1) \mid x_1 \in \Delta_{in}\} \subseteq \hat{S}_n \subseteq D_{out} \qquad \square$$

(Example) Consider the input of a DNN being two dimensional. Let $D_{in} = [1, 2] \times [1, 2]$, and $\Delta_{in} = [0.99, 2.01] \times [0.99, 2.01] \setminus D_{in}$. Then the smallest value of κ can be $\sqrt{0.01^2 + 0.01^2}$, where we set κ to be 0.02 for simplicity purposes. κ quantifies the amount of domain enlargement. Let output of the DNN f be one dimensional, and let $D_{out} = [-10, 10]$ and $S_n = [1, 8]$. Assume that Lipschitz constant ℓ equals 100, then $\ell\kappa = 2$. Creating by expanding S_n with amount $\ell\kappa$ on both sides, one gets $\hat{S}_n = [1 - 2, 8 + 2] = [-1, 10]$. As $[-1, 10] \subseteq [-10, 10]$, the safety property holds not only in D_{in} but also for $D_{in} \cup \Delta_{in}$.

B. Solving SVbTV

For the general case where a DNN f is improved to f' with a slight change in the underlying parameters, our first strategy is to reuse the previously created state abstraction for f and check if they can also be used as the state abstraction for f'.

Proposition 4. *[Reusing state abstraction - single layer] Given state abstractions $S_1, \dots, S_n$ for establishing the proof of $\phi^f_{D_{in}, D_{out}}$. When the following conditions hold, the property $\phi^{f'}_{D_{in} \cup \Delta_{in}, D_{out}}$ also holds.*

- $\forall x \in D_{in} \cup \Delta_{in}$, $g'_1(x) \in S_1$,
- $\forall i \in \{1, \dots, n-2\}, \forall x_i \in S_i : g'_{i+1}(x_i) \in S_{i+1}$.
- $\forall x_{n-1} \in S_{n-1} : g'_n(x_{n-1}) \in D_{out}$.

Again, checking each condition is a substantially simpler problem that only involves encoding neurons in one layer, and overall there are n such problems to be checked independently. This makes the checking highly parallelizable and the worst case (under parallelization) is bounded by the maximum number of neurons in one layer. The generalization of Proposition 4 tries to only select a subset of $S_1, \dots, S_n$ to be reused, with a price of each subproblem involving multiple layers. Still, each subproblem can be checked independently, thereby utilizing the power of parallelization.

Proposition 5. *[Reusing state abstraction - multiple layers] Given state abstractions $S_1, \dots, S_n$ for establishing the proof of $\phi^f_{D_{in}, D_{out}}$. The property $\phi^{f'}_{D_{in} \cup \Delta_{in}, D_{out}}$ also holds when the following condition is met: There exists positive integers $\langle\alpha_1\rangle, \langle\alpha_2\rangle, \dots, \langle\alpha_l\rangle$, where $1 < \langle\alpha_1\rangle < \langle\alpha_2\rangle < \dots < \langle\alpha_l\rangle < n - 1$, such that*

- $\forall x \in D_{in} \cup \Delta_{in}$, $g'_{\langle\alpha_1\rangle}(\dots(g'_1(x))) \in S_{\langle\alpha_1\rangle}$,
- $\forall j \in \{1, \dots, l-1\}, \forall x_{\langle\alpha_j\rangle} \in S_{\langle\alpha_j\rangle}$: $g'_{\langle\alpha_{j+1}\rangle}(\dots(g'_{\langle\alpha_j\rangle+1}(x_{\langle\alpha_j\rangle}))) \in S_{\langle\alpha_{j+1}\rangle}$.
- $\forall x_{\langle\alpha_l\rangle} \in S_{\langle\alpha_l\rangle} : g'_n(\dots(g'_{\langle\alpha_l\rangle+1}(x_{\langle\alpha_l\rangle}))) \in D_{out}$.

As an example, consider a 6-layer network, i.e., $n = 6$. Let $l = 2$ and $\langle \alpha_1 \rangle = 2$ and $\langle \alpha_2 \rangle = 4$. Then the verification is split into three subproblems by reusing S_2 and S_4, namely

- $\forall x \in D_{in} \cup \Delta_{in}, g'_2(g'_1(x)) \in S_2$,
- $\forall x_2 \in S_2 : g'_4(g'_3(x_2))) \in S_4$.
- $\forall x_4 \in S_4 : g'_6(g'_5(x_4)) \in D_{out}$.

Our final strategy is to reuse the previously created network abstraction for DNN f.

Proposition 6. *[Reusing network abstraction] Given network abstraction $\hat{f}$ of the original DNN f where $f \xrightarrow{D_{in}} \hat{f}$. Provided that $\phi^{f}_{D_{in},D_{out}}$ is based on the proof where $\forall x \in D_{in} : \hat{f}(x) \in D_{out}$. Then for the new DNN f', if $f' \xrightarrow{D_{in}} \hat{f}$, then $\phi^{f'}_{D_{in},D_{out}}$ also holds.*

Proof. If $f' \xrightarrow{D_{in}} \hat{f}$, then based on the definition of network abstraction we have $\forall x \in D_{in}$, $f'(x) \in \{\hat{f}(x) \mid x \in D_{in}\}$. As in the original proof, we have $\{\hat{f}(x) \mid x \in D_{in}\} \subseteq D_{out}$, so $\forall x \in D_{in}$, $f'(x) \in D_{out}$, thereby implying that $\phi^{f'}_{D_{in},D_{out}}$ also holds. □

While Proposition 6 handles the case with parameter adjustment, encountering both domain enlargement and parameter adjustment, we can first assume that the domain is not enlarged, and perform network transformation to reach the same structure of $\hat{f}$. Then, we can apply Propositions 1 or 3 by reusing the intermediate results in verifying the abstraction of the original DNN f to check whether the property $\phi^{f'}_{D_{in} \cup \Delta_{in},D_{out}}$ holds also in domain enlargement.

C. Incremental abstraction fixing for SVbTV

In this section, we provide a summary on how to proceed when previously mentioned sufficient conditions do not hold. We consider the case over Proposition 4, where extensions to Proposition 5 is straightforward. If Proposition 4 does not hold for DNN f', then one or more of the followings may occur:

- $\exists x \in D_{in} \cup \Delta_{in}, g'_1(x) \notin S_1$,
- $\exists\, i \in \{1, \ldots, n-2\}$ s.t. $\exists x_i \in S_i : g'_{i+1}(x_i) \notin S_{i+1}$, or
- $\exists x_{n-1} \in S_{n-1} : g'_n(x_{n-1}) \notin D_{out}$.

We further restrict the case where only *one state abstraction* S_{i+1} is problematic, i.e., $\exists x \in S_i : g'_{i+1}(x) \notin S_{i+1}$; for other layers where $j \neq i$, $\forall x \in S_j : g'_{j+1}(x_j) \in S_{j+1}$. We may find a new S'_{i+1} as a replacement of S_{i+1} to ensure that $\forall x \in S_i : g'_{i+1}(x) \in S'_{i+1}$. However, we need a forward propagation from S'_{i+1} to later layers, and check whether there exists $k \in \{i+2, \ldots, n-2\}$, such that $\forall x \in S'_k : g'_{k+1}(x) \in S_{k+1}$. The existence of k shows that the propagation from enlarged approximation in earlier layers is again covered by the approximation of later layers in the previous proof. Therefore, the property is preserved.

When we fail to find a k such that the forward propagation can stop before reaching the last layer, we have to resort to traditional methods to check whether the additional approximation in $S'_{i+1} \setminus S_{i+1}$ is reachable. The problem is reduced to check the safety property of the sub-network from the new DNN, where the property is encoded with the additional state abstraction. However, in the worst case, the problem can appear in the first state abstraction, and nothing can be reused; this implies that we may need to re-verify the whole network.

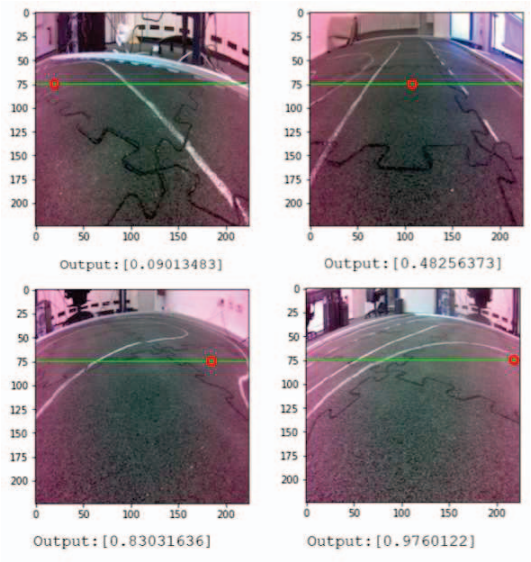

Fig. 3. Visualizing the output of the DNN (red circle) on the race track

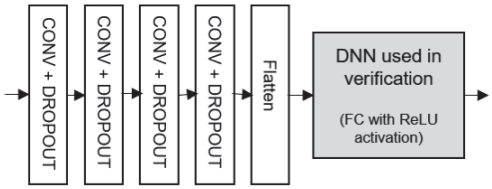

Fig. 4. Visualizing the DNN to be formally verified in the experiment

V. Experiment

To understand the technology in realistic settings, in our experiment, we use a $1/10$ scale vehicle platform that equips with a GPU and a camera module to enable autonomous driving using visual perception. The DNN deployed in the GPU takes an RGB image of size 224×224 and produces a single output v_{out} within the range $[0, 1]$. Output v_{out} can be used to reconstruct the visual waypoint via the formula $(x, y) := (\mathsf{int}(224 * v_{out}), 75)$, which suggests the vehicle the next destination on the image plane to follow (examples shown in Figure 3). We first train a DNN on CIFAR10 dataset with an enlarged image matching the dimension, remove the last layer (i.e., perform transfer learning) and add a new linear layer in order to produce the v_{out} value. Then the modified single-output network is trained using a manually labeled data set collected on the race track.

The network to be verified is truncated from the original one for visual perception by taking layers after convolution, as illustrated in Figure 4. The decision is largely due to the limitation of state-of-the-art DNN formal verification tools. The input bound D_{in} on the verified network is created by recording the minimum and maximum visited neuron value (output of layer "Flatten" in Figure 4) for the complete data

TABLE I
TIME SAVINGS FROM INCREMENTAL VERIFICATION

case ID	SVuDC time / original time	SVbTV time / original time
1	5.27%	37.52%
2	0.72%	4.19%
3	0.16%	4.68%
4	1.34%	8.52%

set, together with additional buffers. Subsequently, we deploy the system, perform continuous execution on the race track, and monitor the DNN. Whenever an image, when being fed into the DNN, creates neuron values (for the output of layer "Flatten" in Figure 4) that exceed the bound, the enlarged bound is recorded to form $D_{in} \cup \Delta_{in}$ for the second verification task. To generate model variations, we fix the weights on the convolution layer and perform fine-tuning such that multiple DNNs to be verified share the same input domain.

We evaluate the performance of incremental verification in both SVuDC and SVbTV with various models during incremental tuning. To generate state abstractions of layers, we adopt tool ReluVal [3], which formally checks properties on a given neural network with symbolic interval analysis. That is, the state abstraction of a neuron is bounded by its lower and upper valuations. In incremental verification, we set the bounds of state abstractions and check whether there is any violation. To accelerate the verification, we decompose a network into two parts such that

- if the first part preserves the state abstraction, the verification stops in SVuDC case.
- verification can be parallelized for the SVbTV case.

Totally we generate four networks from the first in the incremental tuning process. Every network marked with $i+1$ is obtained by tuning i-th network, for $1 \leq i \leq 4$. The results are listed in Table I. As the approximation is usually larger than the reachable states, the verification shows that the properties are preserved in these networks. From Table I, we observe that incremental verification can always take less than ten percent cost of the original.[3] The reasoning is that the scale of the verified network is smaller, and the computation is saved by reusing the established state abstraction. The exception for the first row of SVbTV is that the verification for the first network is fast, making the time saving via decomposition less obvious.

VI. CONCLUDING REMARKS

In this paper, we formulated the problem of formal verification of DNN under the continuous improvement setting. We laid the theoretical foundation by proposing multiple sufficient conditions to avoid complete re-verification efforts, and conducted an initial experiment in the lab setting.

By combining formal verification of DNNs with the concept of continuous engineering, our initial work opens many interesting directions to be explored. Solver-wise, it is worth investigating how exact solvers based on MILP or SMT can be engineered to enable proof reuse, as we observe that techniques such as cuts (conditions to remove real-valued solutions while maintaining all feasible integer-valued solutions) lose their validity upon domain enlargement. Another direction is to consider the continuous evolution of the quantitative specification of DNN [21] and the corresponding reuse in the formal verification setting. Yet another direction is to consider the symbolic reasoning using both forward and backward propagation in a continuous verification setup.

[3]In Table I, the value for SVbTV is taken by the maximum execution time among all subproblems, as executing of each subproblem checking is independent of others.

REFERENCES

[1] C.-H. Cheng, G. Nührenberg, and H. Yasuoka, "Runtime monitoring neuron activation patterns," in *DATE*. IEEE, 2019, pp. 300–303.

[2] T. A. Henzinger, A. Lukina, and C. Schilling, "Outside the box: Abstraction-based monitoring of neural networks," in *ECAI*. IOS Press, 2020, pp. 2433–2440.

[3] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *USENIX Security*, 2018, pp. 1599–1614.

[4] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *S&P (Oakland)*. IEEE, 2018, pp. 3–18.

[5] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[6] H.-D. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis of deep neural networks," in *FM*. Springer, 2019, pp. 670–686.

[7] Y. Y. Elboher, J. Gottschlich, and G. Katz, "An abstraction-based framework for neural network verification." in *CAV*. Springer, 2020, pp. 43–65.

[8] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[9] G. Katz, C. Barrett, D. L. Dill, K. D. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *CAV*. Springer, 2017, pp. 97–117.

[10] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *ATVA*. Springer, 2017, pp. 269–286.

[11] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić *et al.*, "The marabou framework for verification and analysis of deep neural networks," in *CAV*. Springer, 2019, pp. 443–452.

[12] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *ATVA*. Springer, 2017, pp. 251–268.

[13] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward relu neural networks," *arXiv preprint 1706.07351*, 2017.

[14] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NFM*. Springer, 2018, pp. 121–138.

[15] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting robustness certification of neural networks," in *ICLR*, 2018.

[16] Y. Tsuzuku, I. Sato, and M. Sugiyama, "Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks," in *NeurIPS*, 2018, pp. 6541–6550.

[17] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," in *IJCAI*, 2018, pp. 2651–2659.

[18] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas, "Efficient and accurate estimation of lipschitz constants for deep neural networks," in *NeurIPS*, 2019, pp. 11 427–11 438.

[19] D. Zou, R. Balan, and M. Singh, "On lipschitz bounds of general convolutional neural networks," *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1738–1759, 2019.

[20] B. Paulsen, J. Wang, and C. Wang, "Reludiff: Differential verification of deep neural networks." in *ICSE*. IEEE, 2020.

[21] S. A. Seshia, A. Desai, T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, and X. Yue, "Formal specification for deep neural networks," in *ATVA*. Springer, 2018, pp. 20–34.