Bringing Engineering Rigor to Deep Learning

Kexin Pei
kpei@cs.columbia.edu
Columbia University

Shiqi Wang
tcwangshiqi@cs.columbia.edu
Columbia University

Yuchi Tian
yuchi.tian@columbia.edu
Columbia University

Justin Whitehouse
jaw2228@cs.columbia.edu
Columbia University

Carl Vondrick
vondrick@cs.columbia.edu
Columbia University

Yinzhi Cao
yinzhi.cao@jhu.edu
Johns Hopkins University

Baishakhi Ray
rayb@cs.columbia.edu
Columbia University

Suman Jana
suman@cs.columbia.edu
Columbia University

Junfeng Yang
junfeng@cs.columbia.edu
Columbia University

ABSTRACT

Deep learning (DL) systems are increasingly deployed in safety- and security-critical domains including autonomous driving, robotics, and malware detection, where the correctness and predictability of a system on corner-case inputs are of great importance. Unfortunately, the common practice to validating a deep neural network (DNN) – measuring overall accuracy on a randomly selected test set – is not designed to surface corner-case errors. As recent work shows, even DNNs with state-of-the-art accuracy are easily fooled by human-imperceptible, adversarial perturbations to the inputs. Questions such as how to test corner-case behaviors more thoroughly and whether all adversarial samples have been found remain unanswered.

In the last few years, we have been working on bringing more engineering rigor into deep learning. Towards this goal, we have built five systems to test DNNs more thoroughly and verify the absence of adversarial samples for given datasets. These systems check a broad spectrum of properties (e.g., rotating an image should never change its classification) and find thousands of error-inducing samples for popular DNNs in critical domains (e.g., ImageNet, autonomous driving, and malware detection). Our DNN verifiers are also orders of magnitude (e.g., 5,000×) faster than similar tools. This article overviews our systems and discusses three open research challenges to hopefully inspire more future research towards testing and verifying DNNs.

1 INTRODUCTION

Deep Learning (DL) has made tremendous progress over the past few years, achieving or surpassing human-level performance for a diverse set of tasks including visual recognition [37, 46, 69], speech recognition [39, 90], and playing games [59, 68]. These advances have led to widespread adoption and deployment of DL in security- and safety-critical systems such as self-driving cars [6, 13, 15], malware detection [65, 92], and aircraft collision avoidance systems [43].

This wide adoption of DL presents new challenges as the predictability and correctness of such systems are of crucial importance. Unfortunately, DL systems, despite their impressive capabilities, often demonstrate unexpected or incorrect behaviors for several reasons such as biased training data, overfitting, and underfitting of the models. In safety- and security-critical settings, such incorrect behaviors can lead to disastrous consequences such as a fatal collision of a self-driving car. For example, a Google self-driving car recently crashed into a bus partly because it expected the bus to yield under a set of rare conditions, but the bus did not [33]. In 2016, a Tesla car in autopilot crashed into a trailer partly because the autopilot system failed to recognize the trailer as an obstacle due to its “white color against a brightly lit sky” and the “high ride height” [78]. A similar incident happened in 2019 [79]. These disasters call for more thorough validation of DL systems against corner cases.

Unfortunately, prior approaches to validating DL systems are not designed to thoroughly surface corner-case errors. A common practice is to measure a DL system’s prediction accuracy on a randomly selected test set, which often covers few or none corner cases. One can gather and label as much real-world data as possible [1, 5], but given the enormous input space, such blind gathering risks at once wasting much manual effort and missing many corner cases. Unsurprisingly, recent work on adversarial DL [32, 56, 77] showed that human-imperceptible perturbations easily fooled today’s most accurate DL systems to the inputs. Adversarial DL itself, however, is designed to find only the most effortless adversarial samples quickly. It limits the perturbations to minimal noise, not realistic transformations such as light condition change [62]. Nor does it try to find as many adversarial samples as possible. Questions such as how to find more corner cases under diverse transformations and whether all corner cases for a given dataset have been found remain open.

This challenge of thoroughly checking DL systems sounds extremely familiar as researchers have been working towards the same goal for traditional software. Unfortunately, the plethora of testing and verification tools created for traditional software cannot directly apply to DL because the two programming paradigms are drastically different. In traditional software engineering, developers translate the decision logic in their brains into program statements, each of which gradually progresses toward a final goal. In DL engineering, the decision logic is automatically extracted from a vast dataset and embedded in millions of opaque weight parameters. Consider statement coverage, the predominant empirical metric to quantify testing thoroughness of traditional software. Such traditional software testing metric is meaningless in a DL system as any single input can exercise all statements executed by DL inference.

In the last three years, we have been building new testing and verification tools to bring more rigor to DL engineering. Given the challenges in specifying a full functional spec of a deep neural network (DNN) as it would amount to specifying that for human brains, we design our tools to check transformation-invariant properties such as "slight light condition change must not change the image class." They explore design tradeoffs between scalability (whether the tool can scale to large DNNs), completeness (whether the tool can find all property violations for a given dataset), and domain knowledge needed (whether the tool requires access to the DNN internals). We briefly describe each system below.

- DeepXplore is a whitebox testing tool that defines the first test coverage metric for DNNs we call *neuron coverage* – the percentage of activated neurons by a test set, and uses this metric to guide the generation of new inputs to increase coverage [61].
- DeepTest leverages neuron coverage to test autonomous-driving systems by adding fogs or rains to road scenes [80].
- VeriVis is a blackbox verification tool that exhaustively checks a computer vision system correctly handles certain transformations of an image (e.g., all rotations within five degrees have the same correct image label) [62].
- ReluVal is a whitebox verification tool that, given an input range, leverages interval arithmetic and symbolic analysis to compute rigorous DNN output bounds for property verification [86]. It can prove the absence of adversarial examples or find all input sub-intervals that may contain adversarial examples.
- Neurify improves upon ReluVal and leverages what we call linear relaxation to tighten the DNN output bounds further and reduce false positives [85].

A key additional benefit of exhaustive testing and verification is that our tools can serve as an objective, rigorous benchmark for many DNN techniques. For instance, any technique purporting to make DNNs robust against adversarial attacks should not be evaluated only on the attacks designed to find adversarial samples quickly. Instead, they should be evaluated on the exhaustive set of attack samples that our tools, especially the verifiers, generate. Experiments using tools did reveal such issues in prior robustness training techniques.

Our tools have found thousands of corner-case errors over a broad spectrum of DL systems including ImageNet-scale image classifiers, object detectors, malware detectors, self-driving car systems, and cloud computer vision systems built by Google, Amazon, IBM, and Microsoft. They also verified some of these DL systems on popular datasets.

Our verification tools outperform other tools of the same kind by orders of magnitude (5,000× on average). We are encouraged to see that the concepts and algorithms in our tools start to gain adoption by other research groups and the industry [14, 27, 35, 49–52, 58, 74, 89, 93, 94].

This article overviews our tools because we are most familiar with them; we are by no means the only group in this emerging area of testing and verifying DL. In fact, multiple groups have also begun working in this area [28, 31, 44, 55, 70, 71, 81, 88]. We hope our article will help inspire more researchers to join us in tackling this important and exciting challenge of robust DL.

2 THE PROPERTIES TO CHECK

We design our tools to check *transformation-invariant* properties: given input x and transformation $\mathcal{T}$ such as lighting condition change, a DNN's prediction on the transformed sample $\mathcal{T}(x)$ should be similar to that on the original sample x, in most cases the same label. (It is conceivable to relax the property and include a set of related labels, though our systems did not use this relaxed form.) If the DNN outputs a real number such as the driving angle in autonomous driving, the difference between the predictions on the two samples should be smaller than a given threshold.

Our rationale behind this design choice is that the decision logic contained in a DNN is often opaque even to its designers. For instance, creating a complete specification for the correct behavior of a self-driving car under different driving conditions essentially equals recreating the logic of a human driver, computationally infeasible and not practical. In addition, the nature of optimization means that there are multiple acceptable ways through different internal states for satisfying the final goal. For instance, a car can be safely driven on the road with many slightly different but similar steering

Table 1: Transformations supported by our tools to check safety properties. The ✓mark indicates that the tool supports the transformation, and ✗otherwise.

Transformation $\mathcal{T}$	DeepXplore	DeepTest	VeriVis	ReluVal	Neurify
L-norm	✗	✗	✗	✓	✓
Smoothing	✗	✓	✓	✗	✗
Contrast	✗	✓	✓	✗	✓
Brightening	✓	✓	✓	✗	✓
Occlusion	✓	✓	✓	✗	✗
Affine	✓	✓	✓	✗	✓
Weather	✗	✓	✓	✗	✗

angles. Therefore, we conjecture that DL testing and verification should focus on partial, input-output correctness rather than complete functional correctness.

Although simple, transformation-invariant properties can express crucial safety requirements of a wide range of DL systems. For example, they can ensure that the recognized phrases/sentences of a speech recognition system will not change under different background noises. Malware detection systems should not change their classifications from malware to benign due to varying types of code obfuscation/transformation techniques that do not affect malicious functionality [91].

One caveat is that, like all other DL testing and verification tools, our tools check properties on the individual, not all possible, inputs. A fundamental assumption in DL (and machine learning in general) is that the decision logic learned from a representative dataset will generalize to all data produced by the same underlying distribution as the dataset. Therefore, the guarantees achieved on individual inputs should hopefully also generalize.

Table 1 summarizes the transformations supported by our tools to check safety properties. We consider seven general categories of transformation functions, much more complete and realistic than adding slight noise as in prior adversarial DL. Many of these transformations are widely used by computer vision researchers to emulate the naturally occurring distortions and deformations and motivate the new design of model architectures to be invariant against such transformations [17, 23, 87]. The first category of transformation is more general that computer vision. It covers domains such as malware detection or aircraft collision avoidance. For instance, changing the number of authors of a malicious PDF file should not change its classification. We describe each transformation category in the following.

L-norm bounded perturbation. Testing and verifying L-norm based properties includes perturbing the input x into $x' = \mathcal{T}(x)$ so that $L_p(x' - x)$, the distance between x and x', is bounded a user-defined value. Our tools support L_1-norm or L_∞-norm. This property category is widely used in adversarial testing of image classifiers [32], malware detectors [61], self-driving cars [61, 62, 80, 85] and aircraft collision avoidance systems [44, 85, 86].

Smoothing. This transformation emulates the blurring effect that may be encountered in different scenarios, such as autonomous driving or face authentication. It is part of the convolution-based transformations which apply a convolution kernel on the input image and produce the output images such that each pixel values are determined by its local neighbors and the corresponding kernel weights. In particular, the smoothing transformations we consider include the average blurring, median blurring, erosion, and dilation, which compute the average, median, minimum, and maximum of the pixel values within the kernel, respectively, and replace the center pixel value of the kernel with the result of the computation.

Contrast, Brightening, and Occlusion. These transformations emulate the lighting effect, variations of camera configurations (in rendering visual inputs), and occlusions by unexpected objects in front of the camera. Specifically, changing the contrast or brightness of a visual input involves multiplying or adding the same constant c for each pixel values. Adding occlusions is straightforward: defining a patch using another image with a smaller size and overlaying it in the original image.

Affine transformation. Affine transformations emulate the potential distortions of the image that may happen in the real world. They operate on the coordinates of pixels. In particular, this category includes five operations: rotation, shear, scale, translation, and reflection. All the operations include multiplying with the matrix of the pixel coordinates with a 3-by-3 affine transformation matrix. We refer interested readers to [62, 80] for a detailed description of such transformations.

Weather. Finally, it is important to check whether a model, especially those working with visual inputs in the wild (e.g., autonomous driving and collision avoidance), is robust against different weather conditions. However, it is challenging to mathematically represent weather conditions without making strong assumptions. To emulate such transformations, we employ a simple patch image such as rain or fog effect and directly overlay it on the original image. There exist more advanced techniques using the generative adversarial network (GAN) [48]. We discuss this in our future research problem in Section 5.

3 THE TOOLS

This section overviews the five tools we have built, discussing each's key techniques and related work. All are open sourced.

3.1 DeepXplore

DeepXplore is the first (to the best of our knowledge) systematic white-box testing system for DNNs. It has two key techniques. First, it defines *neuron coverage* as a new metric to measure the test coverage of DNNs. The intuition is that the neurons in a DNN are for recognizing features in input, with earlier layers recognizing lower-level features and later layers higher-level features. Thus, the neurons activated by a test input serves as a good indicator of the decision logic exercised by the test. Neuron coverage is analogous to statement coverage, the empirical test coverage metric for traditional software.

Second, given an input, DeepXplore uses neuron coverage to guide the generation of new inputs that (1) increase neuron coverage and (2) are realistic transformations of the given input. The basic idea works as follows. To increase neuron coverage, DeepXplore computes a change to the input that maximizes a particular neuron's activation. Since DNNs are differentiable, DeepXplore leverages gradient descent to compute this input change. However, this greedily computed change is not necessarily a realistic transformation of the input. For instance, it may alter the pixel values by different amounts when the transformation we wanted is brightening, which requires that all pixel values change by the same amount. DeepXplore thus revises the greedy change based on these constraints, computing a new input that can indeed be a realistic transformation of the given input.

Many follow-up projects improved different aspects of DeepXplore, such as the coverage metric [14, 49, 58, 89], the application domains and properties [27, 52, 80, 93, 94], the test generation algorithm [35, 50, 51, 74].

3.2 DeepTest

DeepTest extends the coverage guided training developed in DeepXplore in testing autonomous driving DNNs. It supports a much broader set of transformations, including different weather conditions. Some of them are not differentiable, so gradient descent is not directly applicable. DeepTest proposes neuron coverage guided greedy search to generate error-inducing inputs and maximize neuron coverage. In this greedy algorithm, different transformations are combined, and those transformations that can successfully increase the neuron coverage will be recorded and prioritized while more images are synthesized.

DeepTest also leverages metamorphic relations to identify erroneous behaviours. Intuitively, the steering angle of a self-driving cars on synthesized inputs should not differ much from the steering angle on original inputs. However, there is no one true steering angle given each input for an autonomous car. For examples, small variations of steering angles can still be tolerated by cars. Therefore, a tighter metamorphic relation will result in more false positives. To strike a balance between false positives and false negatives, the following metamorphic relations are defined. The predicted labels of original images are $\{\theta_{o1}, \theta_{o2}, ..., \theta_{on}\}$. The predicted labels of synthesized images are $\{\theta_{t1}, \theta_{t2}, ..., \theta_{tn}\}$. The respective manual labels are $(\{\hat{\theta}_1, \hat{\theta}_2, ..., \hat{\theta}_n\})$. The metamorphic relation are defined as $(\hat{\theta}_i - \theta_{ti})^2 \leq \lambda \, MSE_{orig}$. while $MSE_{orig} = \frac{1}{n}\sum_{i=1}^{n}(\hat{\theta}_i - \theta_{oi})^2$. λ is a parameter used for making trade-off between false positives and false negatives.

3.3 VeriVis

Despite finding thousands of corner-case errors, our testing tools cannot guarantee that all errors are found or there are no errors for a given input and transformation. This limitation is analogous to the testing of traditional software.

To provider stronger guarantees, we build VeriVis, a black-box verification tool for computer vision systems that can fully verify a DNN against a given image and a supported transformation (Section 2). For instance, it can verify however one rotates an image within five degrees, the resultant images all have the same label as the given image.

A difficult challenge is that the parameter of a transformation is often continuous and can be an arbitrary real number, such as the rotation degree which can be 1, 0.1, 0.01, etc. How to exhaustively verify a model against the infinitely many possible transformed inputs? Fortunately, our key insight is that the image space itself is discrete because the pixel values are integers from 0–255 and coordinates are integers bounded by the image size. Leveraging this insight, VeriVis reduces the continuous, infinite transformation parameter space into a finite number – polynomial to the image size – of parameter values. For instance, up to n^3 ($n = w \cdot h$ is the image size) number of rotation degrees can provably cover all possible rotated images. This technique is analogous to the state-space reduction in model checking [12, 20, 30] and helps DeepXplore avoid redundantly checking many equivalent inputs.

3.4 ReluVal

There are two challenges of whichVeriVis falls short. First, many safety-critical DNNs work in continuous input and output space. For instance, the unmanned aircraft collision avoidance system X (ACAS Xu), which uses DNNs to predict the best actions such as "90-degree left" according to the distance, speed, and approaching angle of an intruder plane in the vicinity. NASA and FAA [2, 53] successfully tested it and is on schedule to install it in over 30,000 passengers and cargo aircraft worldwide [57] and US Navy's fleets [7]. It is thus paramount to guarantee that ACAS Xu predicts robust actions for given input ranges. Second, a discrete space can still be too large to check exhaustively. For instance, a 28-by-28 MNIST hand-written digit data [47] with $L_\infty = 1$ bounded perturbation can have up to 2^{784} number of concrete images to check.

We build ReluVal to address these difficult challenges. It focuses on verifying properties of the following form: a DNN never violates any safety property (e.g., , no collisions) for any (maliciously fed) values in an input range (e.g., , between 0 and 500 mph for the intruder speed). Mechanically, given an input range X, ReluVal propagates it through the layers of a DNN and computes a sound overapproximation of the output bound Y leveraging a classic static analysis technique called abstraction interpretation [25]. It does so by executing every operator of the DNN abstractly in the interval domain leveraging interval arithmetic [63]. If Y contains no property violations, ReluVal has soundly verified that the DNN has no violations on X.

A key challenge in ReluVal is the inherent overestimation caused by input dependencies [26, 63, 86] when interval arithmetic is applied to complex functions. Specifically, the operands of each hidden neuron depend on the same input to the DNN, but interval arithmetic assumes that they are independent and may thus compute an output range much larger than the true range. For instance, consider a simplified neural network in which input x is fed to two neurons that compute $2x$ and $-x$ respectively, and the intermediate outputs are summed to generate the final output $f(x) = 2x - x$. If the input range of x is $[0, 1]$, the true output range of $f(x)$ is $[0, 1]$. However, naive interval arithmetic will compute the range of $f(x)$ as $[0, 2] - [0, 1] = [-1, 2]$, introducing a huge overestimation error.

Much of our research effort in ReluVal focuses on mitigating this challenge; here we describe two effective techniques to tighten output bounds. The first is *symbolic interval*, similar to symbolic execution for traditional software [16, 45, 66, 67]. In particular, ReluVal tracks the intermediate computations using not only the intervals but also the symbolic values whenever possible. In the preceding example, ReluVal tracks the intermediate outputs symbolically ($[2x, 2x]$ and $[-x, -x]$ respectively) to compute the range of the final output as $[x, x]$. When propagating symbolic bound constraints across a DNN, ReluVal correctly handles non-linear functions such as ReLU ($max(x, 0)$), one of the most common neuron activation functions, and calculates proper symbolic upper and lower bounds.

The second is *iterative interval refinement*. When the output range of the DNN is too large to be conclusive, ReluVal iteratively bisects the input range and repeats the range propagation on the smaller input ranges. This technique is in a spirit similar to abstraction refinement [11, 38]. Mathematically, we prove that interval refinement on practical DNNs always converges in finite steps.

Compared to the state-of-art DNN verifier Reluplex which leverages SMT and linear programming solvers, ReluVal is on average over 200× faster. In addition, ReluVal is amenable to massive parallelization so that the speedup could be much bigger compared hard-to-parallelize SMT solvers. Concurrent to ReluVal, other DNN verifiers [28, 81, 88] have also been developed. To the best of our knowledge, ReluVal is the only one that scales to DNNs with tens of thousands of neurons.

3.5 Neurify

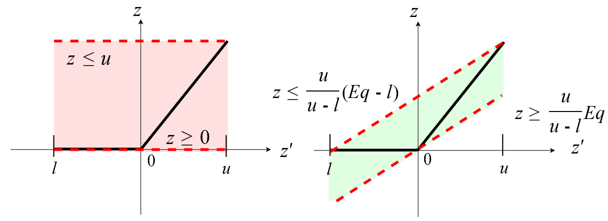
We build Neurify to address two challenges in ReluVal. Consider $ReLU(x) = max(x, 0)$. When the symbolic input interval may span 0, ReluVal concertizes the output interval to $[0, u]$ where u is the concrete upper bound of x, shown in Figure 1(a). It does so to avoid symbolic reasoning of both linear pieces of the ReLU, which may easily explode considering the large number of ReLU nodes in a DNN. However, this concretization method eliminates any symbolic dependencies tracked. Second, ReluVal bisects the DNN input interval to refine the DNN output bound, but doing so at the DNN input is not efficient or direct because overestimation happens actually at internal ReLU nodes.

Neurify solves these challenges using two techniques. First, instead of concretizing the output interval of an overestimated node to $[0, u)$, Neurify uses *symbolic interval relaxation* to bound the output as shown in Figure 1(b), simultaneously simplifying the symbolic constraints to avoid explosion while retaining more accurate dependencies. This technique is similar to linear relaxation in [88], but Neurify adapts it to represent the bounds symbolically. Second, Neurify splits directly at an overestimated node and produces two sets of linear equations, one covering the case when the node input is smaller than or equal to 0, the other the node input is greater than 0. Each set of linear equations is then solved efficiently using off-the-shell LP solvers.

These techniques cut down overestimation errors by up to 59.64% compared to ReluVal. As a result, Neurify is over 20× faster than ReluVal and 5,000× faster than Reluplex.

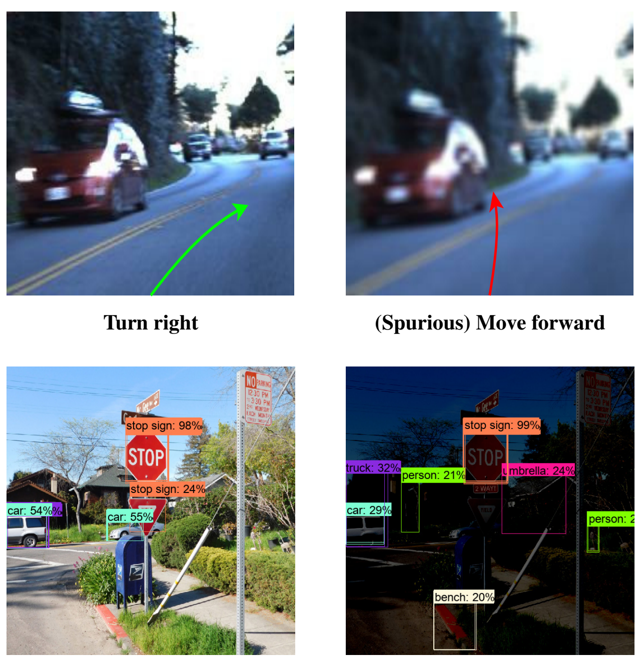
4 A TASTE OF THE RESULTS

We evaluate our comprehensive toolset on a wide range of applications including the safety-critical domains as well as common benchmarks. These include (1) 12 ImageNet classification models [18, 37, 40, 41, 69, 75, 76, 95]; (2) 6 self-driving systems such as Nvidia's DAVE2 [3, 4, 10, 15, 21, 22, 83]; (3) 5 online commercial vision APIs [8, 19, 34, 42, 54] built by the largest companies including Google, Microsoft, Amazon, and IBM; (4) 4 handwritten digit classification models [47];



(a) Naive concretizaion (b) Symbolic linear relaxation

Figure 1: Subfigure (a) shows how ReluVal concretizes the output interval of a ReLU when its input spans 0, and (b) how Neurify uses symbolic linear relaxation to simplify constraints while retaining dependencies.



Turn right (Spurious) Move forward

No pedestrian (Spurious) Pedestrian detected

Figure 2: The right shows error-inducing inputs generated from the original inputs (left). The properties and models under test are (from top to bottom): blurring on Rambo, brightening on Faster-RCNN Inception-V2.

(5) 42 models in aircraft collision avoidance systems (ACAX XU) [44]; and (6) 6 malware detectors for Android and PDF files [9, 24, 36, 60, 72, 73, 82, 84].

As our extensive evaluation on these datasets and systems shows, our tools effectively found thousands of error-inducing inputs even for models with state-of-the-art accuracy. They also verified the absence of errors for up to 32% of the images ImageNet dataset are robust against transformations described in Section 2. Figure 2 presents some interesting errors found. The upper row shows that Rambo [22], Top-ranked self-driving models in Udacity challenge [3, 4, 22] is not robust against blurring effect. The lower row shows that state-of-the-art object detection model – faster RCNN [64] detects spurious pedestrian when the lighting condition changes. Besides images, our tools also found errors in non-visual DNNs such as malware detectors [60, 72]. For instance, changing three attributes of a malicious PDF file – size from 1 to 34, number of actions from 0 to 21, and number of font objects from 1 to 20 – causes the malware detector to classify it as benign.

5 CONCLUSIONS AND OPEN CHALLENGES

In this article, we reviewed the tools and core techniques we have developed in the last three years towards rigorous testing and verification of DL systems. Although the initial results are auspicious, several difficult open challenges remain to be addressed.

First, as discussed in Section 2, all current DNN verification tools focus on verifying properties on a limited set of samples with the hope that the guarantees achieved on individual samples generalize to unseen samples. We believe this fundamental question requires making certain probabilistic assumptions on the distributions of the dataset (i.e., the distribution of available data is the same with those unseen) – the same underlying assumption for why machine learning generalizes. The next concrete research question is how we can adapt existing specific testing and verification techniques (e.g., interval analysis, mixed-integer programming) on reasoning distributions of inputs.

Second, the properties and specifications considered so far are largely “syntactic" such as changing the lighting effect should not change an image’s semantics. How can we support verification of richer properties and semantic transformations such as "changing the season from summer to winter?"

Third, although our tools support many operations (e.g., convolutions and ReLUs) in DNNs, they cannot handle batch normalization or other activation functions such as Sigmoid [29]. How to define reasonable coverage metric or compute intervals or sound overapproximation for these operations?

These open research questions are by no means a complete list in this exciting new research area. For instance, robust training is another exciting direction that leverages the errors found to train more robust DNNs. We hope our initial research findings and insights introduced in this article helps shed light on these important topics and inspire more research effort towards reliable DL.

6 ACKNOWLEDGEMENTS

We thank everyone who helped with the work [61, 62, 80, 85, 86] summarized here. It was sponsored in part by NSF grants CNS-18-54000, CNS-16-18771, CNS-16-17670, CNS-15-63843, and CNS-15-64055; ONR grants N00014-17-1-2010, N00014-16-1-2263, and N00014-17-1-2788; faculty fellowships from Google, DiDi, and J.P. Morgan; and Amazon Cloud Credits for Research. Opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Governments.

REFERENCES

[1] 2010. ImageNet crowdsourcing, benchmarking & other cool things. http://www.image-net.org/papers/ImageNet_2010.pdf.

[2] 2015. NASA, FAA, Industry Conduct Initial Sense-and-Avoid Test. https://www.nasa.gov/centers/armstrong/Features/acas_xu_paves_the_way.html.

[3] 2016. Chauffeur model. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/chauffeur.

[4] 2016. Epoch model. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/cg23.

[5] 2016. Report on autonomous mode disengagements for waymo self-driving vehicles in california. https://www.dmv.ca.gov/portal/wcm/connect/946b3502-c959-4e3b-b119-91319c27788f/GoogleAutoWaymo_disengage_report_2016.pdf?MOD=AJPERES.

[6] 2017. Baidu Apollo Autonomous Driving Platform. https://github.com/ApolloAuto/apollo.

[7] 2018. NAVAIR Plans to Install ACAS Xu on MQ-4C Fleet. https://www.flightglobal.com/news/articles/navair-plans-to-install-acas-xu-on-mq-4c-fleet-444989/.

[8] amazon [n. d.]. Amazon Rekognition, deep learning-based image recognition search, verify, and organize millions of images. https://aws.amazon.com/rekognition/.

[9] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium*.

[10] autopilot:dave 2016. Nvidia-Autopilot-Keras. https://github.com/0bserver07/Nvidia-Autopilot-Keras.

[11] Thomas Ball and Sriram K Rajamani. 2002. The S LAM project: debugging system software via static analysis. In *ACM SIGPLAN Notices*, Vol. 37. ACM, 1–3.

[12] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. 1999. Symbolic model checking without BDDs. In *International conference on tools and algorithms for the construction and analysis of systems*. Springer, 193–207.

[13] Cara Bloom, Joshua Tan, Javed Ramjohn, and Lujo Bauer. 2017. Self-driving cars and data collection: Privacy perceptions of networked autonomous vehicles. In *Symposium on Usable Privacy and Security (SOUPS)*.

[14] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. 2017. Coverage-based greybox fuzzing as markov chain. *IEEE Transactions on Software Engineering* (2017).

[15] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[16] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs.. In *OSDI*, Vol. 8. 209–224.

[17] Gong Cheng, Peicheng Zhou, and Junwei Han. 2016. RIFD-CNN: Rotation-Invariant and Fisher Discriminative Convolutional Neural Networks for Object Detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[18] François Chollet. 2016. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv preprint arXiv:1610.02357* (2016).

[19] clarifai 2013. Clarifai API: Large Scale Visual Recognition. https://developer.clarifai.com/models/general-image-recognition-model/aaa03c23b3724a16a56b629203edc62c.

[20] Edmund M Clarke, Orna Grumberg, Marius Minea, and Doron Peled. 1999. State space reduction using partial order techniques. *International Journal on Software Tools for Technology Transfer* 2, 3 (1999), 279–287.

[21] clone:dave 2016. Behavioral cloning: End-to-end learning for self-driving cars. https://github.com/navoshta/behavioral-cloning.

[22] clone:dave 2017. Rambo model for Udacity self-driving car challenge 2. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/rambo.

[23] Taco Cohen and Max Welling. 2016. Group equivariant convolutional networks. In *International conference on machine learning*. 2990–2999.

[24] contagio 2010. Contagio, PDF malware dump. http://contagiodump.blogspot.de/2010/08/malicious-documents-archive-for.html.

[25] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 238–252.

[26] Luiz Henrique De Figueiredo and Jorge Stolfi. 2004. Affine arithmetic: concepts and applications. *Numerical Algorithms* 37, 1 (2004), 147–158.

[27] Andrea Drmic, Marin Silic, Goran Delac, Klemo Vladimir, and Adrian S Kurdija. 2017. Evaluating robustness of perceptual image hashing algorithms. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 995–1000.

[28] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*. Springer, 121–138.

[29] Mahyar Fazlyab, Manfred Morari, and George J Pappas. 2019. Safety Verification and Robustness Analysis of Neural Networks via Quadratic Constraints and Semidefinite Programming. *arXiv preprint arXiv:1903.01287* (2019).

[30] Cormac Flanagan and Patrice Godefroid. 2005. Dynamic partial-order reduction for model checking software. In *ACM Sigplan Notices*, Vol. 40. ACM, 110–121.

[31] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

[32] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *Proceedings of the 3rd International Conference on Learning Representations*. http://arxiv.org/abs/1412.6572

[33] google-accident 2016. A Google self-driving car caused a crash for the first time. http://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report.

[34] google-vision-api 2011. Cloud Vision API - Derive insight from images with our powerful Cloud Vision API. https://cloud.google.com/vision/.

[35] Divya Gopinath, Kaiyuan Wang, Mengshi Zhang, Corina S Pasareanu, and Sarfraz Khurshid. 2018. Symbolic execution for deep neural networks. *arXiv preprint arXiv:1807.10439* (2018).

[36] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2016. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435* (2016).

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[38] Thomas A Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. 2002. Lazy abstraction. *ACM SIGPLAN Notices* 37, 1 (2002), 58–70.

[39] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[40] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. (2017).

[41] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, Vol. 1. 3.

[42] ibm [n. d.]. IBM Watson Visual Recognition Service. https://www.ibm.com/watson/developercloud/doc/visual-recognition/index.html.

[43] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. 2016. Policy compression for aircraft collision avoidance systems. In *Proceedings of the 35th IEEE/AIAA Digital Avionics Systems Conference*.

[44] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the 29th International Conference On Computer Aided Verification*.

[45] James C King. 1976. Symbolic execution and program testing. *Commun. ACM* 19, 7 (1976), 385–394.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*.

[47] Yann LeCun, Corinna Cortes, and Christopher JC Burges. 2010. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[48] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. 2017. Unsupervised image-to-image translation networks. In *Advances in Neural Information Processing Systems*. 700–708.

[49] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 120–131.

[50] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. 2018. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 100–111.

[51] Lei Ma, Fuyuan Zhang, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. Combinatorial testing for deep learning systems. *arXiv preprint arXiv:1806.07723* (2018).

[52] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 175–186.

[53] Mike Marston and Gabe Baca. 2015. ACAS-Xu initial self-separation flight tests. *NASA Technical Reports Server* (2015).

[54] microsoft [n. d.]. Microsoft Computer Vision API. https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/.

[55] Matthew Mirman, Timon Gehr, and Martin Vechev. 2018. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*. 3575–3583.

[56] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition*.

[57] MIT Tech Notes. 2015. Airborne Collision Avoidance System X. *MIT Lincoln Laboratory* (2015).

[58] Augustus Odena and Ian Goodfellow. 2018. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. *arXiv preprint arXiv:1807.10875* (2018).

[59] OpenAI. 2018. OpenAI Five. https://blog.openai.com/openai-five/.

[60] pdfrate 2012. PDFRate, A machine learning based classifier operating on document metadata and structure. http://pdfrate.com/.

[61] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles*.

[62] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Towards practical verification of machine learning: The case of computer vision systems. *arXiv preprint arXiv:1712.01785* (2017).

[63] Michael J. Cloud Ramon E. Moore, R. Baker Kearfott. 2009. *Introduction to Interval Analysis*. SIAM.

[64] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'15)*. MIT Press, Cambridge, MA, USA, 91–99. http://dl.acm.org/citation.cfm?id=2969239.2969250

[65] Joshua Saxe and Konstantin Berlin. 2015. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 11–20.

[66] Edward J Schwartz, Thanassis Avgerinos, and David Brumley. 2010. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *2010 IEEE Symposium on Security and Privacy*. IEEE, 317–331.

[67] Koushik Sen, Darko Marinov, and Gul Agha. 2005. CUTE: a concolic unit testing engine for C. In *ACM SIGSOFT Software Engineering Notes*, Vol. 30. ACM, 263–272.

[68] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354.

[69] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[70] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*. 10802–10813.

[71] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings*

of the ACM on Programming Languages 3, POPL (2019), 41.

[72] Charles Smutz and Angelos Stavrou. 2012. Malicious PDF detection using metadata and structural features. In *Proceedings of the 28th Annual Computer Security Applications Conference*.

[73] Michael Spreitzenbarth, Felix Freiling, Florian Echtler, Thomas Schreck, and Johannes Hoffmann. 2013. Mobile-sandbox: having a deeper look into android applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*.

[74] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic Testing for Deep Neural Networks. In *Automated Software Engineering (ASE)*. ACM, 109–119.

[75] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning.. In *AAAI*, Vol. 4. 12.

[76] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition*.

[77] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations*.

[78] tesla-accident 2016. Understanding the fatal Tesla accident on Autopilot and the NHTSA probe. https://electrek.co/2016/07/01/understanding-fatal-tesla-accident-autopilot-nhtsa-probe/.

[79] tesla-accident-2019 2019. Understanding the fatal Tesla accident on Autopilot and the NHTSA probe. https://abcnews.go.com/Politics/teslas-autopilot-engaged-fatal-florida-crash-ntsb/story?id=63107290.

[80] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. ACM, 303–314.

[81] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HyGIdiRqtm

[82] virustotal 2004. VirusTotal, a free service that analyzes suspicious files and URLs and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware. https://www.virustotal.com/.

[83] visualize:dave 2016. Visualizations for understanding the regressed wheel steering angle for self driving cars. https://github.com/jacobgil/keras-steering-angle-visualizations.

[84] Nedim Šrndic and Pavel Laskov. 2014. Practical evasion of a learning-based classifier: a case study. In *Proceedings of the 35th IEEE Symposium on Security and Privacy*.

[85] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*. 6367–6377.

[86] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1599–1614.

[87] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. 2017. A-Fast-RCNN: Hard Positive Generation via Adversary for Object Detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[88] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. 2018. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems*. 8400–8409.

[89] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Hongxu Chen, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, Jianxiong Yin, and Simon See. 2018. Coverage-Guided Fuzzing for Deep Neural Networks. *arXiv preprint arXiv:1809.01266* (2018).

[90] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. 2016. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256* (2016).

[91] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium*.

[92] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*.

[93] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. Deeproad: Gan-based metamorphic autonomous driving system testing. *arXiv preprint arXiv:1802.02295* (2018).

[94] Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2018. Generating Natural Adversarial Examples. In *International Conference on Learning Representations (ICLR)*.

[95] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2017. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012* (2017).