2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)

Symbolic Execution for Attribution and Attack Synthesis in Neural Networks

Divya Gopinath, Corina S. Păsăreanu
Carnegie Mellon University Silicon Valley
Moffett Field, CA 94035, USA
{divyag1@andrew.,pcorina@}cmu.edu

Kaiyuan Wang, Mengshi Zhang, Sarfraz Khurshid
University of Texas at Austin
Austin, TX 78712, USA
{kaiyuanw,mengshi.zhang,khurshid}@utexas.edu

***Abstract*—This paper introduces *DeepCheck*, a new approach for validating Deep Neural Networks (DNNs) based on core ideas from program analysis, specifically from *symbolic execution*. DeepCheck implements techniques for lightweight symbolic analysis of DNNs and applies them in the context of image classification to address two challenging problems: 1) identification of *important* pixels (for attribution and adversarial generation); and 2) creation of adversarial attacks. Experimental results using the MNIST data-set show that DeepCheck's lightweight symbolic analysis provides a valuable tool for DNN validation.**

I. Overview

As deep neural networks become more widely used in tasks of high importance, developing techniques that validate them becomes increasingly urgent. This paper introduces a new approach for validating neural networks based on the classic program analysis of symbolic execution. The key insight is to transform the network into an imperative program that is amenable to analysis using symbolic execution. Two analyses are presented: 1) to identify important pixels that can explain the classification decisions made by a neural network; and 2) to create 1-pixel and 2-pixel attacks by identifying pixels or pixel-pairs and computing their values so the neural network misclassifies the modified images. The two analyses apply in synergy and provide a more scalable approach to finding attacks. An experimental evaluation using the widely studied MNIST dataset demonstrates the usefulness of symbolic execution in analyzing neural networks.

Although 1-pixel and 2-pixel attacks have been studied before [5], we make the surprising new observation that neural networks can be vulnerable to such attacks even along the paths that follow the *same activations* pattern as the original input. Such attacks went unnoticed with previous testing techniques [4], [6], [7], which focused on generating tests that increase the coverage of activated neurons, and hence did not check for attacks along the same path. Furthermore, if such attacks are shown to not exist, our tool then is able to provide *guarantees* that the network is behaving as expected. A full version of this paper can be found in [2].

We illustrate our approach on a subject neural network $\mathcal{N}$; a fully connected $784\times10\times10\times10\times10$ network, which has been trained on all 60,000 images in the training data of the MNIST dataset [3], and has an accuracy of 92%. Given the trained network $\mathcal{N}$, we apply our technique DeepCheck$^{\tau}$ to translate it to an imperative program $\mathcal{P}$ that has the same behavior as the original network but is amenable to program analysis. Figure 1(a) shows an example image $\mathcal{I}$ from the standard MNIST training data, which has the predicted label of 3 (which is the same as its true label).

A. Identifying important pixels

Given $\mathcal{I}$ as an input, our important pixel identification technique DeepCheck^{Imp} executes the program $\mathcal{P}$, and for the execution path taken by $\mathcal{I}$, computes for every output label, a linear expression in terms of the input variables (784 image pixels). The algorithm then uses the coefficients (coeff) of the input pixels in the expression corresponding to the label assigned by the network (3 in the case of the example), to assign an *importance score* for every pixel. A pixel p_1 is considered more important than another p_2, if the classification decision is impacted more by p_1 than p_2. DeepCheck^{Imp} employs three metrics; *abs*, *co*, *coi*, to calculate the importance of each pixel.[1]. The pixels are then sorted in the descending order of their scores. The pixels which are higher on this list (top threshold %) are identified as being *important*. The insight is that the short-listed important pixels can be held responsible for the classification decision. A small change to the image with respect to the important pixels, such as changing the value of just one important pixel can have a high impact on the classification decision, and may lead to the discovery of *adversarial* examples – the new image differs from the original image by the value of just one pixel but this makes the network incorrectly assign a different label to this image.

Fig. 1(b) illustrates the top-5%, i.e., 39, important pixels highlighted in green. Note, how the important pixels trace the shape of the digit 3 and do not point to areas of the image irrelevant to the classification, such as the background or the edges. The top-10%, i.e., 78 important pixels (Fig. 1(c)) form a denser pattern that traces the shape of the digit 3. This highlights that identifying important pixels based on the coefficients of their respective expressions for the expected label, can help *explain* the classification decision.

[1] *abs*: absolute value of coeff, *co*: actual signed value of coeff, *coi*: actual value of coeff $\times$ input value.

2574-1934/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-Companion.2019.00115

(a) (b) (c) (d) (e)

Fig. 1: (a) Example image with predicted label 3. (b) Top-5% important pixels (highlighted in green) identified by DeepCheck^{Imp}. (c) Top-10% important pixels (green) identified by DeepCheck^{Imp}. (d) 1-pixel attack (highlighted in red) identified by DeepCheck^{tPA}; changing the red pixel to black changes the predicted label to 8. (e) 2-pixel attack (red) that does include an attackable pixel for 1-pixel attack.

Fig. 2: Attackable pixels for 1-pixel attack highlighted in red.

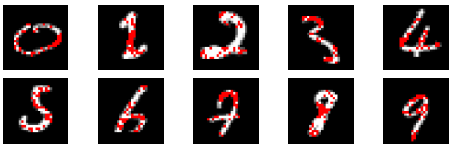

Fig. 3: Attackable pixels for 2-pixel attack highlighted in red.

B. Identifying attack pixels

Conceptually, our t-pixel attack technique DeepCheck^{tPA} aims to create a new image that differs from the original image at t pixels, and has (1) the same *activation pattern* as the original image but (2) a different label from the original image. Specifically, for a 1-pixel attack, DeepCheck^{tPA} selects a pixel p, makes its value symbolic p_s, retains the original concrete values for all other pixels, and constructs a constraint-solving problem, which requires (1) execution of the same path up to the output layer as the original image $\mathcal{I}$ and (2) change in the output label from the original predicted label of $\mathcal{I}$. The constraint-solving problem consists of a simplified path condition for image $\mathcal{I}$'s execution path such that the path condition contains only one symbolic value, i.e., p_s. If the constraint is satisfiable, a solution provides the value for pixel p to create the 1-pixel attack image. For solving constraints, we use the SMT solver Z3 [1].

DeepCheck^{tPA} checks whether any pixel of $\mathcal{I}$ can be attacked by making one pixel symbolic at a time and checking the resulting path condition. Figure 1(d) shows a 1-pixel attack identified by our approach for image $\mathcal{I}$; changing the red pixel to black changes the predicted label of the image to 8. This attackable pixel actually lies in the top-5% (top 39) important pixels for $\mathcal{I}$ identified by DeepCheck^{Imp}. The rank order of this attackable pixel in descending order of importance is 21. For this case, focusing the 1-pixel attack on important pixels can allow finding an attack much quicker than checking every pixel for attackability. In fact, this image only has one 1-pixel attack. A linear search that starts at the first image pixel (top-left corner) and scans left-to-right takes 346 attempts to find this attack pixel, which is over 16X the attackable pixel's rank-order (21). We believe *important pixels* can provide a practical heuristic for a *more scalable approach to create attacks*.

To create 2-pixel attacks, we focus DeepCheck^{tPA} on the important pixels identified by DeepCheck^{Imp}, specifically on the top-5% important pixels. We make $\binom{39}{2}$ = 741 unordered pairs of the selected important pixels, and for each pair, we make the two corresponding variables symbolic, so each path condition created by symbolic execution contains exactly two symbolic variables. Applying DeepCheck^{tPA} to the 741 pairs results in 93 unique 2-pixel attacks. 38 of the 2-pixel attack pairs contain as an element the pixel that was earlier identified for the 1-pixel attack, whereas 55 of the pairs contain only pixels that are not 1-pixel attackable; Figure 1(e) shows one such pair in red. The important pixels identified by DeepCheck^{Imp} play a key role in focusing DeepCheck^{tPA} to find a 2-pixel attack. The first attack found by DeepCheck^{tPA} includes 2 of the 3 top-most important pixels. Thus, the search for a 2-pixel attack for this example requires checking no more than just $\binom{3}{2}$ = 3 pairs.

II. Evaluation and Future Work

The results of applying our tool to ten images from the MNIST data set are displayed in Figures 2 and 3. Our experimental results indicate that it is feasible to use symbolic execution to identify important pixels and to create 1-pixel and 2-pixel attacks, and that important pixels enable a more scalable approach for generating 1-pixel and 2-pixel attacks. For example, for images of 9 out of 10 digits, a 2-pixel attack is found by checking the 2-pixel combinations of just top-4 important pixels, i.e., a pair of pixels to attack the network is found by checking no more than 6 pixel pairs. For details about the experiments see [2]. For the future, we plan to evaluate our technique on larger networks and extend it to other kinds of networks such as convolutional neural nets.

Acknowledgement: This work was supported in part by NSF #1549161 and CCF #1704790.

References

[1] L. de Moura and N. Bjorner, "Z3: An efficient SMT solver," in *TACAS*, 2008.

[2] D. Gopinath, K. Wang, M. Zhang, C. S. Pasareanu, and S. Khurshid, "Symbolic execution for deep neural networks," *CoRR*, vol. abs/1807.10439, 2018. [Online]. Available: http://arxiv.org/abs/1807.10439

[3] "The MNIST database of handwritten digits Home Page," http://yann.lecun.com/exdb/mnist/.

[4] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *SOSP*, 2017.

[5] J. Su, D. V. Vargas, and S. Kouichi, "One pixel attack for fooling deep neural networks," *CoRR*, vol. abs/1710.08864, 2017.

[6] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," *arXiv preprint arXiv:1805.00089*, 2018.

[7] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*. ACM, 2018, pp. 303–314.