Styx: A Data-Oriented Mutation Framework to Improve the Robustness of DNN

Weijiang Hong[1,2], Zhenbang Chen[1], Chendong Feng[1], Weiyu Pan[1], Ji Wang[1,2]
[1]College of Computer, National University of Defense Technology, Changsha, China
[2]State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha, China
{zbchen,wj}@nudt.edu.cn

ABSTRACT

The robustness of deep neural network (DNN) is critical and challenging to ensure. In this paper, we propose a *general* data-oriented mutation framework, called Styx, to improve the robustness of DNN. Styx generates new training data by slightly mutating the training data. In this way, Styx ensures the DNN's accuracy on the test dataset while improving the adaptability to small perturbations, *i.e.*, improving the robustness. We have instantiated Styx for image classification and proposed pixel-level mutation rules that are applicable to any image classification DNNs. We have applied Styx on several commonly used benchmarks and compared Styx with the representative adversarial training methods. The experimental results indicate: Styx can improve the robustness while ensuring the test accuracy; compared with adversarial training methods, Styx gets an order of magnitude improvement in the training efficiency. The website of Styx is https://dnn-styx.github.io, and the demonstration video can be accessed at https://youtu.be/PcUCo_WwiG4.

KEYWORDS

DNN, Robustness, Mutation, Adversarial examples

1 INTRODUCTION

Nowadays, deep learning (DL) techniques (*e.g.*, deep neural network (DNN) [21]) are widely adopted in more and more applications and make great success, such as image classification [10] and audio recognition [14]. When DNN is applied in safety-critical areas, such as autonomous driving [9] and flight control systems [8], it is important to guarantee the system's safety and security. However, it is challenging to ensure the safety and security of DNN-based applications due to DNN's nature of non-interpretation. One representative threat is the existence of adversarial examples [24], which are produced by adding imperceptible perturbation to the original example but cause the DNN to produce wrong outputs. Almost all DNN models struggle with the threat of adversarial examples [?], and adversarial examples have already caused several disasters in some safety-critical areas [4, 25].

Robustness is an important factor in measuring the safety and reliability of DNN models. *Adversarial training* [5, 13] is an effective method for improving DNN's robustness. The basic idea of *adversarial training* is to retrain the DNN with the adversarial examples to improve the DNN's robustness. However, the improved robustness sacrifices the DNN's test accuracy. For example, when we use BIM [11] to train a CNN model for *CIFAR-10* [26], the test accuracy drops from 75.62% (using *traditional training*) to 53.84%.

According to DNN's back-propagation training mechanism [20], we observe that there may be a balance between robustness and test accuracy. If we only slightly mutate the training dataset, the model trained on the mutated dataset will have a similar test accuracy with the one trained by the original dataset. On the other hand, the model trained by the mutated training dataset will be more robust to the adversarial examples generated by small perturbations. Based on this observation, we propose a general mutation framework, called Styx, to improve the robustness of DNN while maintaining the test accuracy. Styx generates the new training dataset by slightly mutating the training dataset[1]. Styx improves the model's adaptability to small perturbations (*i.e.*, improving the robustness) while ensuring the test accuracy.

The idea of Styx is general. However, the mutation operations of dataset are application-dependent. In this paper, we instantiate Styx in the area of image classification and propose several *pixel-level* mutation rules. Compared with the existing mutation-based methods for DNN [19, 28], our mutation rules are more general and applicable to the DNNs of any image classification tasks. We have implemented Styx and evaluated it on several representative benchmarks. The experimental results indicate the effectiveness and efficiency of Styx.

2 FRAMEWORK AND ALGORITHMS

This section provides an introduction to Styx's framework. Then, we introduce the algorithms that instantiate Styx for image classification DNNs.

2.1 Basic Procedure

Figure 1 shows the basic procedure of Styx. We have a two-stage procedure. The first stage is to use Styx to generate a new training dataset. To prepare for the mutation, we select certain samples from the training dataset according to *Data Ratio*. Then we mutate the samples by the mutator and replace the original data with the mutated ones to get a new training dataset. The second stage contains the training and evaluation. We train different DNN models by the original training dataset and the new training dataset. After that, we use different adversarial attacking methods to evaluate the robustness of the model. We evaluate the robustness as follows: for the set (denoted by $dataset_c$) of correctly classified samples in the test dataset, we apply an adversarial attack to each sample in $dataset_c$; if the new sample is misclassified, it is an adversarial example, and we called the original sample is successfully *attacked*. We record the number of samples that can be successfully attacked (represented by $\#attacked$). We define the robustness of the model as follows.

$$Robustness = 1 - \frac{\#attacked}{\#dataset_c} \tag{1}$$

[1] This is the reason why we call the framework Styx, which is a river offering invulnerability powers. Here we strengthen the training data by mutation.

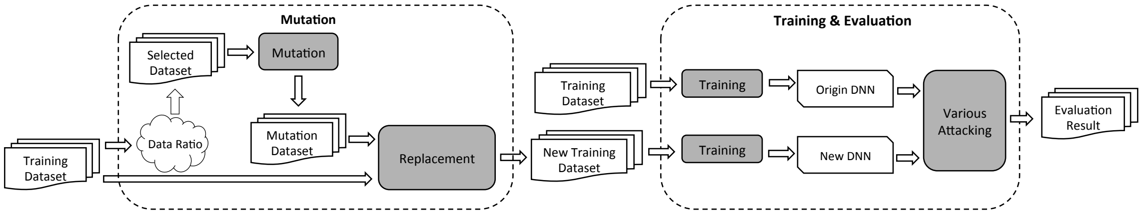

Figure 1: The basic procedure of Styx.

2.2 Algorithms

We instantiate Styx to the applications employing image classification DNNs. Algorithm 1 gives the top-level procedure of Styx. To improve the generality, we use a pixel-level mutation. First, we train a network N_1 (Line 1) based on the original training dataset. Then, we use Random_Inputs and Random_Pixels to randomly select the data and the pixels that we want to mutate (Line 2-3). The selected data is then mutated one by one to get a new dataset (Line 4-6). Finally, we train a new network N_2 (Line 7) based on the mutated training dataset.

Algorithm 1: Styx(D, $data\_ratio$, $pixel\_ratio$)

Input: The training dataset $D, data\_ratio, pixel\_ratio$.
Output: The comparison result $Result$.
1: $N_1 = \text{Training}(D)$
2: $D_s \leftarrow \text{Random\_Inputs}(D, data\_ratio)$
3: $P_s \leftarrow \text{Random\_Pixels}(pixel\_ratio)$
4: **for** $Idx \in D_s$ **do**
5: $\quad D \leftarrow (D \setminus \{D[Idx]\}) \cup \text{Mutate}(D[Idx], P_s)$
6: **end for**
7: $N_2 = \text{Training}(D)$
8: $Result = \text{Evaluation}(N_1, N_2, attack\_method)$
9: **return** $Result$

Mutation. Algorithm 2 gives the details of the mutation. Given an input x and the set P_s of the pixels to be mutated, we mutate the pixels with respect to mutation rules, where mutator ∈ {Zero Mutation, Average Mutation, Random Mutation, Gaussian Noise Mutation}. Each sample can be mutated to get *Count* (set to 1 by default) variation samples. Noted that we normalize the pixel values of each pixel to the interval [0,1] before mutation. In order to be applicable to any image classification DNNs, Styx provides the following four mutators:

- `Zero Mutation`: Since pixels are critical in the prediction of DNN, this rule tries to eliminate the influence of these pixels to the prediction. Hence, the intuitive idea is to reset the value of the pixel to be zero, *i.e.*, using a *black pixel* to replace the original one.
- `Average Mutation`: Contract to the first rule that may be too radical, the second rule replaces the value of the pixel with the average pixel value around it.
- `Random Mutation`: Another common idea is to use random value. Hence, this rule replaces the pixel's value with a random value from 0 to 1.

Algorithm 2: Mutate(x, P_s)

Input: An input x, the set of selected pixels P_s.
Output: A set of mutated input M of size Count.
$M \leftarrow \emptyset$
for $i \in \{1 \ldots \text{Count}\}$ **do**
$\quad$ **for** $Idx \in P_s$ **do**
$\quad\quad x \leftarrow mutator(x, Idx)$
$\quad$ **end for**
$\quad M \leftarrow M \cup \{x\}$
end for
return M

- `Gaussian Noise Mutation`: As Gaussian noise [1] is one of the most popular and natural noises, we have this rule to mutate the value of a pixel by adding Gaussian noise to the original value.

Figure 2 shows the effect of each mutation operation. Here the pixel ratio is 0.1. The first column represents the samples taken from the MNIST dataset [27]. Each row shows different mutator's results.

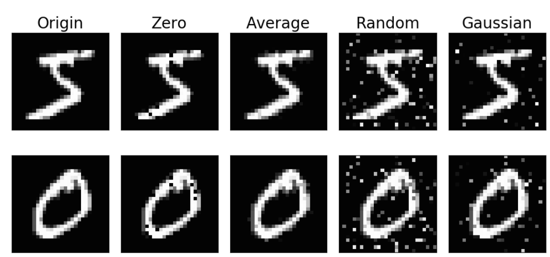

Figure 2: Demonstration of mutator's effect.

Evaluation. After generating the new training dataset, we can train a new DNN model. To evaluate whether the model's robustness is improved, we use different attacking methods to attack the model and calculate the robustness by the Formula 1.

Actually, there are many existing attacking methods for generating adversarial examples, including FGSM [5], BIM [11], DeepFool [15], *etc.* For each attacking method, there are several parameters, such as maximum perturbation, the order of the norm, and the maximum number of iterations. In principle, a larger perturbation and more iterations bring a stronger attack; however, the generated adversarial examples may also be very different from the original sample. As suggested in [3], we have carefully investigated the hyper-parameters of different attacking methods and selected 10 parameter configurations for each attacking method.

3 EVALUATION

To evaluate STYX, we have carried out extensive experiments. This section starts by describing the setup of the experiment in Section 3.1. The results and discussion are given in Section 3.2.

3.1 Experimental Setup

We have implemented STYX in *Python3*. Our evaluation uses three benchmarks: *MNIST*, *Fashion-MNIST* and *CIFAR-10*. We use the standard model structures (*i.e.*, the multilayer perceptron "MLP" and the convolutional neural network "CNN") provided in Keras[2] for the benchmarks. The $data\_ratio$ and $pixel\_ratio$ are 0.5 and 0.1, respectively. The test accuracies of the well-trained models are 98.53% (mnist_MLP), 99.03% (mnist_CNN), 88.69% (fmnist_MLP), 92.50% (fmnist_CNN) and 75.62% (cifar10_CNN), respectively. During evaluation, we use FGSM [5], BIM [11] and DeepFool [15] as the attacking methods. IBM's adversarial-robustness-toolbox[3] is the implementation of these attacking methods. The experiments were carried out on a server with 8 cores and 32G memory. The GPU is RTX 2080 and the OS is Ubuntu Linux 16.04.

3.2 Experimental Results

We evaluate the effectiveness and efficiency of STYX on each benchmark.

- **Effectiveness**: Compared with existing defensive methods, can STYX improve robustness while preserving test accuracy?
- **Efficiency**: Is STYX more efficient than the defensive methods on the training time?

Effectiveness Figure 3 shows the test accuracy result of different training methods. The test accuracy under *adversarial training* decreases compared with the other two training methods. STYX has a similar test accuracy with that of the *traditional training*. Figure 4 shows the average robustness of these models under different attacking methods *w.r.t.* 10 different parameter configurations. For 15 comparisons (*i.e.*, 3 attacks $\times$ 5 models), STYX performs best in 7/15 comparisons and improves the robustness by 8.8% (FGSM), 9.8% (BIM) and 1.9% (DeepFool) on average, respectively. These results indicate that STYX's effectiveness.

Efficiency Table 1 shows the time-costs of different training methods. Compared with *traditional training* (second column), *adversarial training* (third column) often takes 10-18x time to train a model. In contrast, the time-cost of STYX is much cheaper and close to that of *traditional training*. These experimental results demonstrate that STYX is more efficient for training models than *adversarial training*.

4 USAGE

The usage of STYX is a two-step procedure. The first step is to generate the mutated training dataset and train a model. Given the data information of the benchmark (*e.g.*, "mnist"), the model's structure (*e.g.*, "MLP") and the epoch of training(*e.g.*, 20), we can use the following command to train a model.

```
python styx.py <data> <model> <epoch>
```

Then, we use the following command to evaluate the model trained by the method (*e.g.*, "STYX") with respect to an attacking method (*e.g.*, "BIM"). The evaluation generates an excel file containing the statistics of the model's robustness and outputs a result summary.

```
python evaluation.py <data> <model> <train> <attack>
```

More details can be found on the website of STYX.

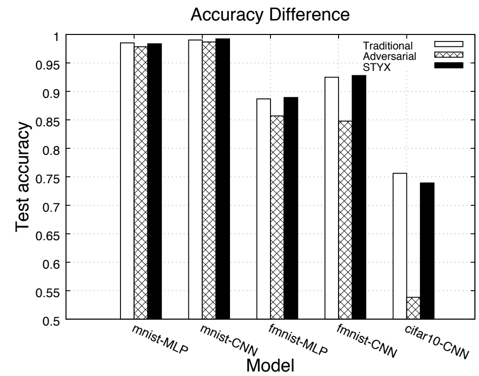

Figure 3: The Accuracy Evaluation.

Table 1: Training time (s).

Model	Traditional	Adversarial	STYX
mnist_MLP	31.56	579.62 (18x)	43.16
mnist_CNN	490.71	5622.76 (11x)	503.42
fmnist_MLP	32.11	568.09 (18x)	43.12
fmnist_CNN	491.00	5626.63 (11x)	503.55
cifar10_CNN	1037.59	10437.13(10x)	1078.61

5 RELATED WORK

STYX is closely related to the existing methods that defend against adversarial attacks, measure the robustness of DNN, or fuzz DNNs.

Existing methods for defending against adversarial attacks and improving the robustness of DNN can be divided into three categories: adversarial retraining [5, 15, 24], network modification [16, 18], and pre-detection [7, 22]. These methods are challenged by the problems including specific attacking defense, scalability, feasibility, *etc.* STYX is close to *adversarial training*. STYX uses mutated training dataset for network training and prevents the over-fitting problem of the specific attacking method.

Measuring the robustness of DNN is also an active topic. In [15], the authors quantify the robustness of DNN by measuring the minimal perturbation that results in adversarial examples. In [2], the authors propose two different metrics: adversarial frequency and adversarial severity. Furthermore, the test adequacy criterion is also regarded as a criterion for evaluating the robustness of DNN. Many coverage criteria have been proposed, such as neuron coverage [19], k-multisection neuron coverage [12], the coverage criteria inspired by MC/DC [23], to name a few. Different from them, we measure

[2]https://github.com/keras-team/keras/tree/master/examples
[3]https://github.com/IBM/adversarial-robustness-toolbox

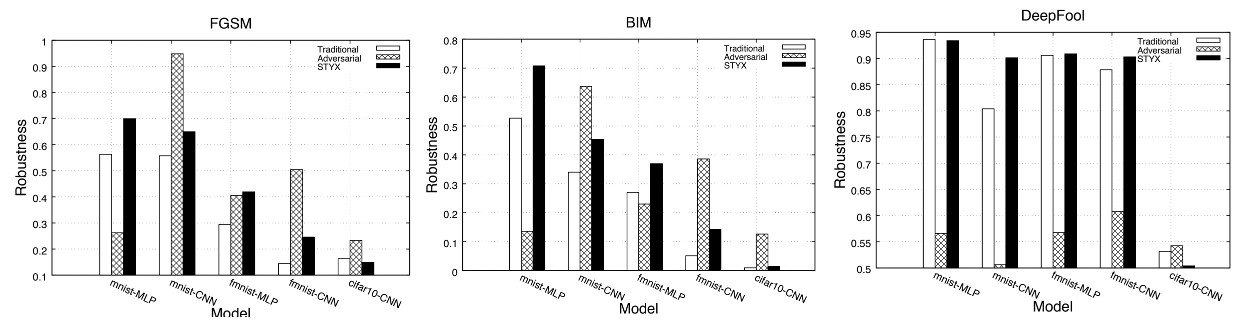

Figure 4: The Robustness Results.

the DNN's robustness from the perspective of attacking methods, and the measurement is more intuitive and realistic.

The application of fuzzing on DNNs has also attracted much interest. TensorFuzz [17] is the first work that introduces the concept of coverage-guided fuzzing for DNN. In [6], the authors propose a differential fuzzing testing framework to check the safety of DL systems. DeepHunter [29] uses metamorphic mutation to generate new inputs based on fuzzing seeds. While these existing fuzzing methods are all coverage-oriented, Styx focuses on the data augmentation with respect to the robustness.

6 CONCLUSION

We propose Styx in this paper, a general data mutation framework, to improve DNN's robustness. Styx can improve the DNN's robustness with respect to different adversarial attacks while preserving test accuracy. We instantiate Styx to image classification DNNs and propose a set of general pixel-level mutation rules. We have evaluated Styx on representative benchmarks. The experimental results indicate that Styx is effective and efficient. The next step lies in several aspects: 1) investigate more general mutation rules; 2) recommend the mutation strategy that results in the best robustness result; 3) apply Styx to more representative benchmarks with respect to more attacking methods.

REFERENCES

[1] Tudor Barbu. [n.d.]. Variational Image Denoising Approach with Diffusion Porous Media Flow. *Abstract and Applied Analysis, pp. 1-8, 2013* ([n. d.]).

[2] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. [n.d.]. Measuring Neural Net Robustness with Constraints. In *Advances in NeurIPS 29, pp.2613–2621, 2016.*

[3] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. 2019. On Evaluating Adversarial Robustness. *arXiv:1902.06705* (2019).

[4] George E. Dahl, Jack W. Stokes, Li Deng, and Dong Yu. 2013. Large-scale malware classification using random projections and neural networks. In *IEEE International Conference on Acoustics*.

[5] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. *CoRR* abs/1412.6572 (2014).

[6] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. [n.d.]. DLFuzz: differential fuzzing testing of deep learning systems. In *Proceedings of ACM FSE, pp. 739–743, 2018.*

[7] Andrew Ilyas, Ajil Jalal, Eirini Asteri, Constantinos Daskalakis, and Alexandros G. Dimakis. 2017. The Robust Manifold Defense: Adversarial Training using Generative Models. *CoRR* abs/1712.09196 (2017).

[8] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I.* 97–117.

[9] Jiman Kim and Chanjong Park. [n.d.]. End-To-End Ego Lane Estimation Based on Sequential Transfer Learning for Self-Driving Cars. In *2017 IEEE Conference on CVPR Workshops, pp. 1194–1202, 2017.*

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[11] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *CoRR* abs/1607.02533 (2016).

[12] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. [n.d.]. DeepGauge: multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE, ASE 2018, pp. 120–131, 2018.*

[13] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards Deep Learning Models Resistant to Adversarial Attacks. (2017).

[14] Tomas Mikolov, Anoop Deoras, Daniel Povey, Lukás Burget, and Jan Cernocký. [n.d.]. Strategies for training large scale neural network language models. In *IEEE Workshop on ASRU, pp. 196–201, 2011.*

[15] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. [n.d.]. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *IEEE Conference on CVPR, pp. 2574–2582, 2016.*

[16] Aran Nayebi and Surya Ganguli. 2017. Biologically inspired protection of deep networks from adversarial attacks. *CoRR* abs/1703.09202 (2017).

[17] Augustus Odena and Ian J. Goodfellow. 2018. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. *CoRR* abs/1807.10875 (2018).

[18] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. [n.d.]. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *IEEE Symposium on S&P, pp. 582–597, 2016.*

[19] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. [n.d.]. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th SOSP, pp. 1–18, 2017.*

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Leaning internal representations by back-propagating errors. *Nature* 323, 6088 (1986), 318–362.

[21] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85–117.

[22] Shiwei Shen, Guoqing Jin, Ke Gao, and Yongdong Zhang. 2017. AE-GAN: adversarial eliminating with GAN. *CoRR* abs/1707.05474 (2017).

[23] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. 2018. Testing Deep Neural Networks. *CoRR* abs/1803.04792 (2018).

[24] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *CoRR* abs/1312.6199 (2013).

[25] 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/.

[26] The CIFAR-10 database Home Page. [n.d.]. https://www.cs.toronto.edu/ kriz/cifar.html.

[27] The MNIST database of handwritten digits Home Page. [n.d.]. http://yann.lecun.com/exdb/mnist/.

[28] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. [n.d.]. DeepTest: automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th ICSE, pp. 303-314, 2018.*

[29] 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. *CoRR* abs/1809.01266 (2018).