Detecting Adversarial Examples via Neural Fingerprinting: ICLR Reproducibility Challenge 2019

Jonathan Besomi, Pietro Carta, Pawel Golinski
Department of Computer Science, EPFL, Switzerland

***Abstract*—Neural fingerprinting [1] has been proposed as a method to detect adversarial examples [3]. The paper has been submitted for acceptance to ICLR'19, and our work revolves around reproducing the results obtained by its authors. The key idea of *Neural fingerprinting* is to detect adversarial examples by using a a form of *consistency check of the model output around the input*. After reproducing the method and experiments independently, we confirm *Neural fingerprinting* as viable, robust defense against adversarial attacks.**

I. Introduction

Neural fingerprinting can be considered as a Neural Network regularization method to curb vulnerability to adversarial examples. We provide a brief recap of the working mechanism of *Neural fingerprinting* . For a more extensive explanation of the topic, refer to the original paper under review [1].

We define a fingerprint as a pair $(\Delta \boldsymbol{x}, \{\Delta \boldsymbol{y}_k\})$, with $k \in \{1, ..., K\}$ and K the number of classes. $\Delta \boldsymbol{x}$ is the fingerprint direction, and $\{\Delta \boldsymbol{y}_k\}$ is the set of target fingerprint responses. $\Delta \boldsymbol{x}$ is chosen by sampling $\mathcal{U}[-\epsilon, \epsilon]^k$, and consist in the direction in which each data point $\boldsymbol{x}$ is to be perturbed. $\{\Delta \boldsymbol{y}_k\}$ is a set of the K target fingerprint responses. Every class has a different target response. When evaluating the network f on $\boldsymbol{x} + \Delta \boldsymbol{x}$ and on $\boldsymbol{x}$, the *change* in output should be close to $\Delta \boldsymbol{y}_k$, if $\boldsymbol{x}$ belongs to class k.

Several definitions of the *change* $F(f(\boldsymbol{x}), f(\boldsymbol{x} + \Delta \boldsymbol{x}))$ can be used. The original authors accept as the change the difference between the normalized logits for $\boldsymbol{x}$ and for $\boldsymbol{x}+\Delta \boldsymbol{x}$.

Therefore, during training, when the class of an example $\boldsymbol{x}$ is known, the network parameters are to be trained not only so that $f(\boldsymbol{x})$ indicates the correct class k, but also so that requirement that the distance between $F(f(\boldsymbol{x}), f(\boldsymbol{x} + \Delta \boldsymbol{x}))$ and $\Delta \boldsymbol{y}_k$ is minimized.

In this setting, $\Delta \boldsymbol{y}_k$ is a K dimensional vector, and the original authors suggest that the value of its coordinates be -0.25 if the coordinate index is different than the class index, or 0.75 if they coincide.

The training phase therefore simply requires an addition of the fingerprint loss L_{fp} the standard loss function L_{data}. The minimization problem with respect to the parameters θ of the model becomes:

$$\min_{\theta} L_{\text{data}} + \lambda L_{fp} \tag{1}$$

When evaluating the network on a data point $\boldsymbol{x}$ for which the class is unknown, every class is tried for a valid fingerprint match. The validity of a fingerprint response is given by how far the output change F is from its target $\Delta \boldsymbol{y}$. If the distance for every class is bigger than some τ, we discard $\boldsymbol{x}$ as an adversarial example. If the distance is smaller than τ for any class, we accept $\boldsymbol{x}$ as real.

The setting can be generalized without efforts to several fingerprint directions. During training, the contribute of each fingerprint is considered equally and the losses for different directions are summed. During evaluation, the distances of the fingerprint responses from the target responses are averaged.

The original authors call the set of fingerprints χ, and the distance of the responses from the target fingerprints $D(\boldsymbol{x}, k, f, F, \chi)$. If the number of fingerprints is N, D is defined as follows:

$$D(\boldsymbol{x}, f, \chi, k) = \frac{1}{N} \sum_{i=1}^{N} \|F(\boldsymbol{x}, \Delta \boldsymbol{x}_i) - \Delta \boldsymbol{y}_{i,k}\|_2 \tag{2}$$

input : example $\boldsymbol{x}$, comparison function D, fingerprinting $\mathcal{X}$, model f
output: accept/reject
if $\exists$ k *such that D(*$\boldsymbol{x}$*,* f*,* χ*,* k*)* $\leq \tau$ **then**
  Return accept;
else
  Return reject;
end

Algorithm 1: Neural Fingerprint

II. Implementation

The authors of [1] include their code for reproducibility purposes. The original code from the paper is mostly implemented in Pytorch, but uses the Tensorflow library Cleverhans [5] for adversarial examples generation. It also requires a number of other dependencies. However, the library versions were not explicitly specified, and we were unable to run it without errors. We therefore resolved to produce an independent reimplementation of *Neural fingerprinting* within an up-to-date version of Tensorflow.

The reimplementation pointed out a couple of minor differences relative to what was written in the original paper. In

section 2.3 of [1], the $\Delta \boldsymbol{x}$ are said to be chosen by uniform sampling of $[-\epsilon, \epsilon]^l$, where l is the input dimension. However the reference implementation for MNIST fingerprints shared by the paper authors uses a uniform sampling from $[0, \epsilon]^l$.

Another point was specially confusing. *Neural fingerprinting* is said to be robust with respect to the choice of $\Delta \boldsymbol{y}$. In practice, we found that targeting with $\delta \boldsymbol{y}_j = -0.25$ if $j = i$ and $\Delta \boldsymbol{y}_j = 0.75$ if $j \ = i$ when the class of the example is i proved computationally unstable, often taking several training epochs before sudden convergence. On the other hand, using $\Delta \hat{\boldsymbol{y}} = -\Delta \boldsymbol{y}$ as target for the fingerprints showed much stabler convergence.

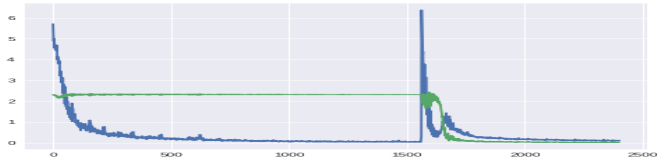

Fig. 1. data (blue) and fingerprint (green) losses when training with proposed $\Delta \boldsymbol{y}$.

III. Evaluation

We were able to reproduce key results from the paper after rewriting the code in Tensorflow. Experiments are available in the repository accompanying this report, in form of a single Jupyter Notebook. The following sections describes our findings from each reproduced experiment.

A. Gaussian balls

The original paper features a section that investigates the effects of *Neural fingerprinting* on a simple classifier to distinguish the points of two two-dimensional Gaussian balls. In the original paper, this example is supposed to provide qualitative insight on the mechanism by which *Neural fingerprinting* works. This example provided us a simple testing framework for our implementation. Since the size of the perturbation $\Delta \boldsymbol{x}$ was not provided in the original paper, we derived a viable one through trial and error, and came to the understanding that the perturbation size must be in the order of magnitude of the size of the separation between classes.

Our Gaussian balls are centered around $(-3, -3)$ and $(3, 3)$, with identity covariance. When tunining the ϵ parameter responsible for the perturbation size, we noticed that for small epsilons in the order of 0.1, the expected fingerprint responses couldn't be trained, while for larger epsilon in the order of 6 and more the response could be easily trained. This is intuitive when reasoning that small epsilons translate the input $\boldsymbol{x}$ to a nearby $\boldsymbol{x} + \Delta \boldsymbol{x}$. It follows that $f(\boldsymbol{x})$ and $f(\boldsymbol{x} + \Delta \boldsymbol{x})$ will tend to be close together, and learning a specific difference might require a huge optimization effort. When $\Delta \boldsymbol{x}$ is big, on the other hand, the fingerprints can be learned without making the classification function too unstable. We therefore suggest that the ϵ parameter should be in the order of magnitude of the distance of each feature between a class and the others.

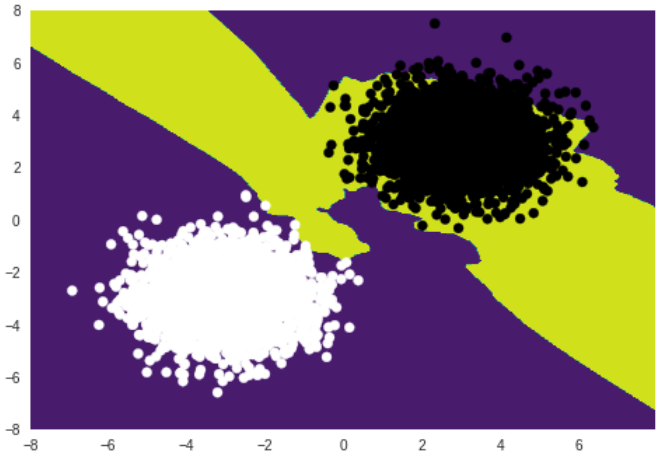

Fig. 2. Including the fingerprint loss in the training generates highly non-linear decision regions

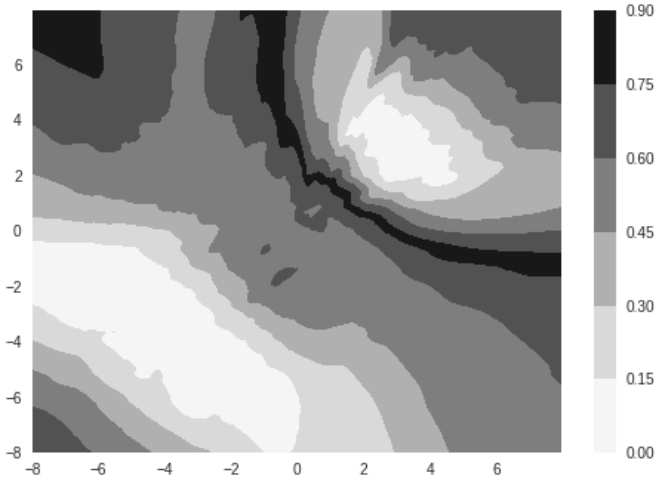

Fig. 3. The fingerprint loss for the gaussian case. Darker regions are likely to be classified as fake.

B. MNIST

As in the original paper, we trained a simple 4-layer network to classify handwritten digits using MNIST as dataset. We implemented the same neural network architecture as specified in table 8 of the original paper.

While training the network, we used five fingerprints, selecting $(\Delta \boldsymbol{x})$ perturbations from uniform distribution on $[0, 0.1]$. $\Delta \boldsymbol{y}$ has been chosen for each fingerprint i and class j had the form of$\Delta \boldsymbol{y}^{i,k}_{n \neq j} = 0.25$, $\Delta \boldsymbol{y}^{i,k}_{j=n} = -0.75$. Our network was trained on weighted sum of fingerprints and cross entropy loss, were both losses contributed equally to the sum. We used the Adam optimizer, with a learning rate of 0.001 in order to achieve convergence. After 10 epochs, the network yields 99.2% accuracy on test data.

Once our model was working properly we started attacking it. Our first experiment has been to compare the distribution of fingerprints loss between original and adversarial examples. Table 2 and Figure 4 from the original paper. Due to

computational constraints, we were able to reproduce only FGSM[3] and BIM[4] attack results, although original paper reports that Neural Fingerprints behave similarly even in case of more sophisticated JSMA [6] and CW[2] attack methods. The results of this experiment are described on Figures 4 and 5.

Next, we examined how different settings of threshold parameters correspond to true and false positive rate. The standard metric to examine that is ROC curve. The result is described on Figure 6 and corresponds to Table 4 and Figure 7c from the original paper.

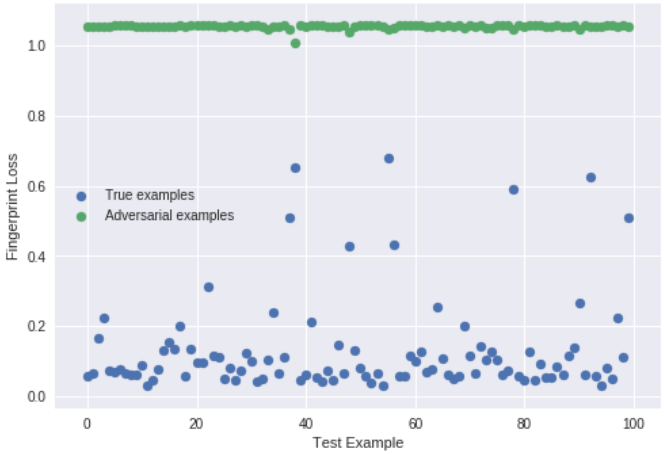

Fig. 4. In case of FGSM attack, there is a clear distinction between true test examples (blue) and adversarial examples (green), when comparing fingerprint loss function value. The accuracy of network on adversarial examples generated by FGSM is 5%.

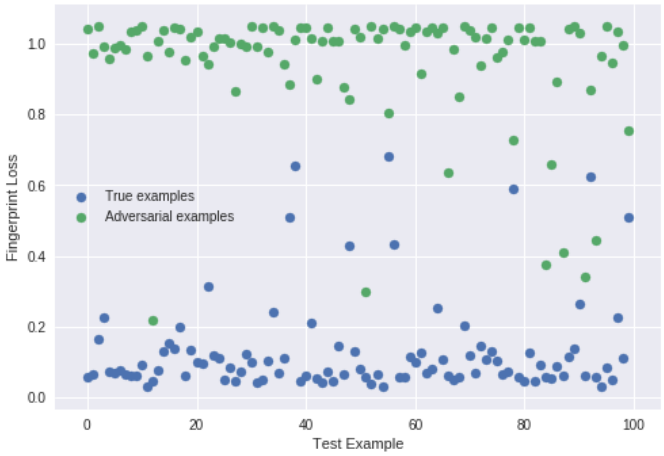

Fig. 5. In case of BIM attack, the distinction is blurrier and there are some outliers. The accuracy of network on adversarial examples generated by BIM is 2%.

C. CIFAR-10

For the CIFAR-10 problem, we obtained similar results as to MNIST. We used the architecture described in table 9 of the paper. More details are available in Jupyter nootebook accompanying this report.

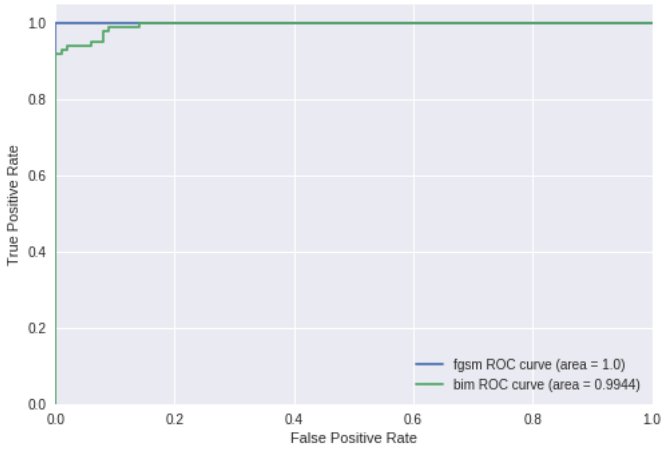

Fig. 6. High AUC-ROC indicates great robustness of Neural Fingerprints against both FGSM and BIM attacks.

IV. Varying number of fingerprints

We examined how varying number of fingerprints affects final network accuracy, execution time and robustness. All figures are available in Jupyter notebook in its final section, and correspond to figure 7b from original paper. Our conclusion from these experiments is similar to the ones that were presented in the paper - increasing number of fingerprints doesn't affect accuracy, but increases robustness. Moreover, we examined how adding fingerprints affects execution time. As expected, increasing fingerprint number scales training and evaluation time linearly - every additional fingerprint requires an additional network inference.

V. Cleverhans attack

In order to test the efficiency and the statement of the paper, with the support of Tensorflow library for adversarial attack Cleverhans we prepared some attacks. Figure 7 show three example of attack generated by Cleverhans.

VI. Machine Learning stack

During the whole project we tried different platforms and solution to work with the data. Because the code is available on Github, we attempted to execute it online on Google Cloud Platform. Due to misstaded dependencies and the use of obsolete library versions, we opted for a reimplementation, We started experimenting locally within Jupyter notebooks. The problem with such approach is that the execution is slow, since our machines don't feature GPUs. Moreover it was complicated to share resources and code among the team member. A good compromise turned out to be Google Colab Notebook. The advantages of this new services is that it offers for free (as of December 2018) collaborative notebooks accelerated by Tesla K80 GPUs.

VII. Difficulties encountered

The original paper effectively explains the concept of *Neural fingerprinting* and provides all the tools for reimplementation. We still incurred some issues while implementing

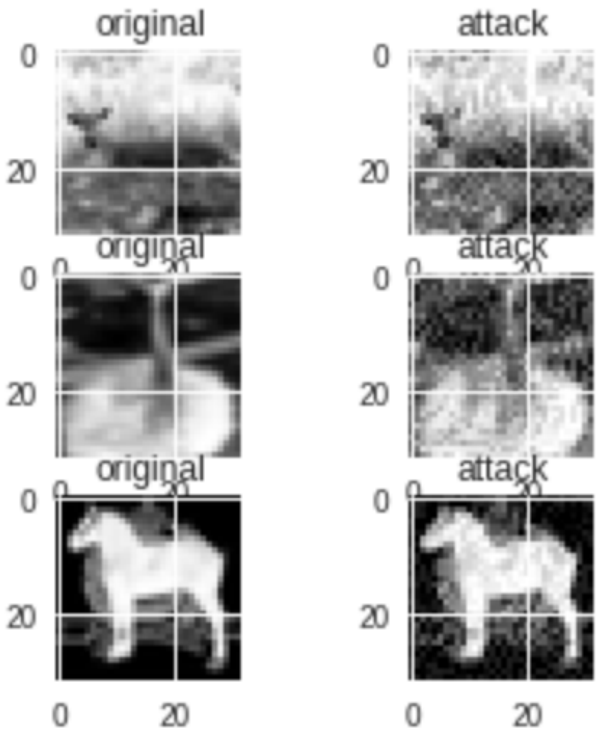

Fig. 7. Cleverhans attack on CIFAR-10

the Gaussian example, since the hyperparameters were not provided. We suspect that due to the low dimensionality of the problem, the optimization was specially sensible to the choice of parameters. The biggest issue remained the trouble when running the provided code. Libraries in use are out of date and since the code is distributed without any form of containerization, sourcing every piece of the puzzle can be very time consuming. We opted to reimplement it ourselves both for the gained flexibility and to learn more specific details about the method. However, our schedule was impacted by this choice, and we couldn't reach our most ambitious goals (replication of adaptive whitebox attacks, implementation of attacks specific for *Neural fingerprinting*). Reproducibility in Machine Learning research remains a moving target, due to the fast moving ecosystem and the necessity for flexibility which is fundamental for researchers. A satisfying middle point could however be achieved by use of up to date libraries and more cautious development practices

VIII. Conclusion

Neural fingerprinting as described in the original paper effectively detects all the classes of adversarial attacks that we have tested. Unfortunately, we were far from exhaustive, and in computer security absence of evidence of vulnerabilities doesn't constitute evidence of their absence. We are however confident that this method deserves the attention of the research community, and we impatiently wait to see it faced to more and better attacks.

References

[1] Anonymous. Detecting adversarial examples via neural fingerprinting. In *Submitted to International Conference on Learning Representations*, 2019. under review.

[2] A. Athalye, N. Carlini, and D. Wagner. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. *arXiv e-prints*, page arXiv:1802.00420, Feb. 2018.

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv e-prints*, page arXiv:1412.6572, Dec. 2014.

[4] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv e-prints*, page arXiv:1607.02533, July 2016.

[5] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, R. Long, and P. McDaniel. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv e-prints*, page arXiv:1610.00768, Oct. 2016.

[6] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. Berkay Celik, and A. Swami. The Limitations of Deep Learning in Adversarial Settings. *arXiv e-prints*, page arXiv:1511.07528, Nov. 2015.