Verifying the Resilience of Neural Network Watermarking

Ben Goldberger[1], Yossi Adi[2], Joseph Keshet[2], and Guy Katz[1]

[1] The Hebrew University of Jerusalem, Israel
{jjgold, guykatz}@cs.huji.ac.il
[2] Bar Ilan University, Israel
{a, b}@biu.ac.il

1 Introduction

Deep Neural Networks (DNN) are a nowadays phenomena in research and in Industry; They're used for large verity of applications, and achieving state of the art result in many fields (Computer vision, speech recognition, AI, and many more). DNNs flexibility and diversity are pushing the limits on what is possible for a computer to solve efficiently. As a result from their empiric success DNNs are now changing the way software is being designed, and broaden the role of Machine Learning trained functions in applications.

Because of the success of DNNs and because designing and training a DNN require certain expertise, time and processing power, the demand for a specifically designed DNN is growing, And Machine Learning as a Service (MLaaS) is now a thing. There are many machine learning services appearing in many forms, from data visualization and cloud computing to frameworks and semi-trained DNNs. This create some issues regarding the rights and ownership of some part of a trained network or design. Because of the relatively simple components of a DNN (Matrices, vectors and simple functions) it's quite easy to copy or use without permission.

In order to deal with such issues we need a way to authenticate a DNN. This may sound simple but the authentication needs to be robust, such that it's hard to remove. Here comes the concept of Digital Watermarking, a way of signing some Digital property such that it's hard to remove the signature, and said signature is unique. There are some proposed method of watermarking a DNN, but it's unclear how effective they are, meaning how difficult it is to remove the watermark from the DNN and what is the effect of removing the watermark.

DNN verification is a new and promising field. We propose a novel methodology to use verification to measure and verify the robustness of watermarking techniques. Main uses of our approach: verify watermarked networks, assess efficiency of watermarking schemes.

The rest of this paper is organized as follows. In Section 2 we provide the necessary background on DNNs, watermarking, and DNN verification. Next, in Section 3 we introduce our technique for casting the watermark resilience problem into a verification problem. Section 4 describes our implementation and evaluation of the approach on several watermarked DNNs for image recognition. We discuss related work in Section 5, and conclude in Section 6.

2 Background

[2,3]

3 Finding the minimal change that will get rid of the network's WaterMark

Guy: make this a proper citation

Ben: ok

Given a watermarked trained neural network as described here [1]. We tested what is the minimal change to the network last layer in order to "remove" some watermarks from the network.

3.1 Defining the problem for one input

Given a watermarked network N with a set of K watermarks (A set of inputs to the network) $\{x_1, \cdots, x_K\}$ we'll mark the network last layer matrix L such that L is a $m \times n$ matrix were n is the layer's number of neurons and m is the network output size. The change to the last layer will be a matrix with the same dimension as L we'll mark as ε, such that $\varepsilon_{i,j}$ is the change to the last layer matrix entry $L_{i,j}$. We can describe the network output as For a certain input x we're only interested in the input to the last layer we'll mark the input to the last layer v. v is a $n \times 1$ vector. So the original network output $y = Lv$ and the changed network output is $y' = (L + \varepsilon)v$.

For a single input x we denote the original network prediction:

$$d_x := argmax_{i\in[m]} \{y_i\}$$

And the changed network prediction:

$$d'_x := argmax_{i\in[m]} \{y'_i\}$$

We're interested to find the minimal change to the last layer ε so that the prediction will change i.e. $d_x \neq d'_x$

Well measure the overall change to the layer in two ways

$$\|\varepsilon\|_\infty = max_{i,j} \{|\varepsilon_{i,j}|\}. \tag{1}$$

And

$$\|\varepsilon\|_1 = \sum_{i,j} |\varepsilon_{i,j}|. \tag{2}$$

For the $\infty - norm$ 1 and a chosen d'_x that is different from d_x the minimization problem looks like that:

$$\begin{aligned}
&Minimize: && M \\
&Subject\ to: && \forall i,j \quad -M \le \varepsilon_{i,j} \le M \\
& && y' = (L+\varepsilon)v \\
& && y'_{d_x} \le y'_{d'_x}
\end{aligned}$$

Variables are the entries in ε, y' and M

(3)

Similarly for the $1 - norm$ 2 the minimization problem looks like that:

$$\begin{aligned}
&Minimize: && M \\
&Subject\ to: && \forall i,j \quad -M \le \sum_{i,j} |\varepsilon_{i,j}| \le M \\
& && y' = (L+\varepsilon)v \\
& && y'_{d_x} \le y'_{d'_x}
\end{aligned}$$

Variables are the entries in ε, y' and M

(4)

3.2 Defining the problem for many inputs

Our definition to a single input minimal change ε to the network last layer L can be extended to more then one input very easily by adding more constraint to the problem.
Given inputs $\{x_1, \cdots, x_k\}$ and their respective values:

$\{v_1, \cdots, v_k\}$	:Inputs to the last layer
$\{y_1, \cdots, y_k\}$	:Output vectors
$\{d_1, \cdots, d_k\}$	:Decisions

Such that

$$\forall 1 \le j \le k \quad d_j \ne argmax_{i\in[m]} \left\{ \left((L+\varepsilon)\, v_j \right)_i \right\}$$

Assuming we choose our new desired decisions $\{d'_1, \cdots, d'_k\}$ our minimization problem for the $\infty - norm$ looks like this:

$$\begin{aligned}
&Minimize: && M \\
&Subject\ to: && \forall i,j \quad -M \le \varepsilon_{i,j} \le M \\
& && \forall j \quad y'_j = (L+\varepsilon)v_j \\
& && \forall j \quad \left(y'_j\right)_{d_j} \le \left(y'_j\right)_{d'_j}
\end{aligned}$$

Variables are the entries in $\varepsilon, y'_1, \cdots, y'_k$ and M

(5)

Similarly for the $\ell_1 - norm$ 2 the minimization problem looks like that:

$$\begin{aligned}
&Minimize: && M \\
&Subject\ to: && \forall i,j \quad -M \le \sum_{i,j} |\varepsilon_{i,j}| \le M \\
& && \forall j \quad y'_j = (L+\varepsilon)v_j \\
& && \forall j \quad \left(y'_j\right)_{d_j} \le \left(y'_j\right)_{d'_j}
\end{aligned}$$

Variables are the entries in $\varepsilon, y'_1, \cdots, y'_k$ and M

(6)

Guy: Overall, this looks good. Some stuff will need to be moved to other sections according to the paper layout.

4 Evaluation

For the $\ell_\infty - norm$ 1 when choosing a prediction different from the original the problem can be expressed as a linear programming query. This is possible since $y' = (L+\varepsilon)v$ is a linear set of constraint, and each entry of ε can be bounded independently.

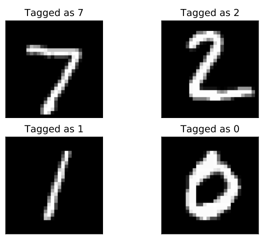
We tested this approach on a Neural network trained on the MNIST data set, the network was watermarked with 100 images of Gaussian noise. The original network had 96.8% accuracy on the MNIST test set and 100% accuracy on the watermark images.

4.1 Removing a single watermark

The first test we did was to find the minimal change according to $\|\cdot\|_\infty$ norm and $\|\cdot\|_1$ norm for every water mark image as describe **??**. We choose to change the original tagging of the image to the second highest score that image got. For example an image w with an output y

$$j = \underset{i \in \{0,\cdots,9\}}{argmax} \{y_i\}$$

(a) Water marks images.



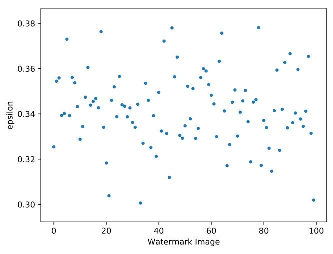
(b) MNIST images.

j is the original tagging of w.

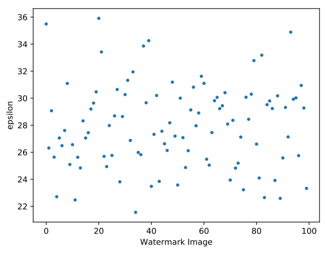
$$j' = \underset{i \in \{0,\cdots,9\}\setminus\{j\}}{argmax} \{y_i\}$$

j' is the new tagging of w.

meaning that after we change the last layer of the network the new output y' largest coordinate is j'. Each watermark image have different minimal ε. This can give us some measure of how difficult it is to remove a single watermark image.



(a) The Minimal $\|\cdot\|_\infty$ for every watermark image



(b) The Minimal $\|\cdot\|_1$ for every watermark image

Fig. 2: Notice the scale of the graphs, the $\|\cdot\|_\infty$ values are much smaller then the $\|\cdot\|_1$ values. As expected

After removing a single watermark with minimal change to the last layer we're interested in the effect this had on our original network accuracy. To test the accuracy after we found the changes all we had to is apply the changes to the original network and run an evaluation on the original network dataset. As seen in this table ?? the $\|\cdot\|_1$ gives better accuracy result on average by can have a bad accuracy result, and the $\|\cdot\|_\infty$ after removing a single watermark have a consistent result (there minimal accuracy and the maximal accuracy are quite close)

Table 1: Epsilons and Accuracy

Number	Avrg eps	Min eps	Max eps	Average acc	Min acc	Max acc	Norm
0	0	0	0	0.97	0.97	0.97	
1	0.34	0.3	0.38	0.87	0.86	0.88	Infinity
1	27.95	21.56	35.91	0.94	0.56	0.97	One

4.2 Removing Multiple watermark

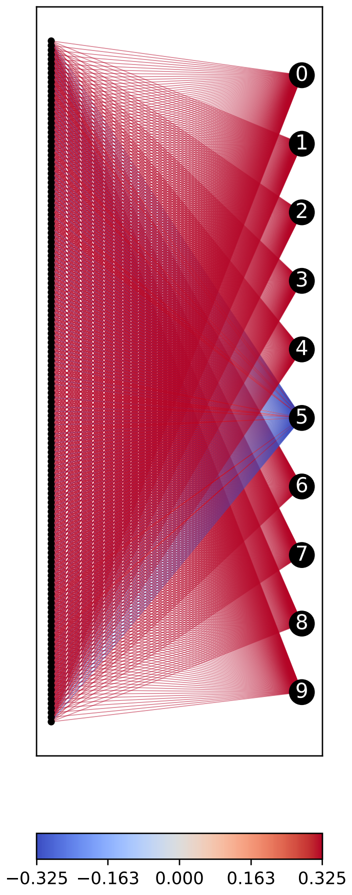
Removing a single

5 Related Work

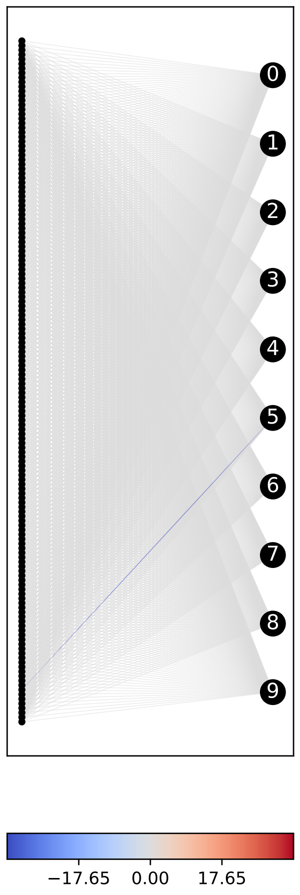
6 Conclusion and Future Work

References

1. Y. Adi, C. Baum, B. Pinkas, and J. Keshet. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *Proc. 27st USENIX Security Symposium*, 2018.
2. G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 97–117, 2017.
3. G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. Dill, M. Kochenderfer, and C. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, 2019.



(a) The Minimal $\|\cdot\|_\infty$ for every watermark image



(b) The Minimal $\|\cdot\|_1$ for every watermark image