Adversarial Robustness via Runtime Masking and Cleansing

Yi-Hsuan Wu [1] **Chia-Hung Yuan** [1] **Shan-Hung Wu** [1]

Abstract

Deep neural networks are shown to be vulnerable to adversarial attacks. This motivates robust learning techniques, such as the adversarial training, whose goal is to learn a network that is robust against adversarial attacks. However, the sample complexity of robust learning can be significantly larger than that of "standard" learning. In this paper, we propose improving the adversarial robustness of a network by leveraging the potentially large test data seen at runtime. We devise a new defense method, called runtime masking and cleansing (RMC), that adapts the network at runtime before making a prediction to dynamically mask network gradients and cleanse the model of the non-robust features inevitably learned during the training process due to the size limit of the training set. We conduct experiments on real-world datasets and the results demonstrate the effectiveness of RMC empirically.

1. Introduction

Deep neural networks (DNNs), despite achieving remarkable performance in many applications, have shown to be vulnerable to *adversarial examples* (Szegedy et al., 2014; Goodfellow et al., 2014), i.e., examples that are intentionally designed to be misclassified by the models but nearly indistinguishable from regular examples in human eyes or some distance measures in the input space. Typically, an adversarial example is generated by slightly perturbing the input of a regular example in directions where the output of the model gives the highest loss (Szegedy et al., 2014; Goodfellow et al., 2014; Kurakin et al., 2016; Papernot et al., 2016a; Moosavi-Dezfooli et al., 2016; Carlini & Wagner, 2017b). The existence of such adversarial examples raises concerns about the security-critical machine learning systems, such as autonomous cars and speech recognition authorization.

[1]Department of Computer Science, National Tsing Hua University, Taiwan. Correspondence to: Shan-Hung Wu <shwu@cs.nthu.edu.tw>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

A plethora of defenses has been proposed, aiming to increase the robustness of a network to adversarial perturbations. Based on different hypotheses about the cause of adversarial examples, some works change the model architecture (Krotov & Hopfield, 2018), distill a large network into a small network (Papernot et al., 2016b), use regularization (Gu & Rigazio, 2014; Hein & Andriushchenko, 2017; Cisse et al., 2017; Jakubovitz & Giryes, 2018; Ross & Doshi-Velez, 2018), or statistically detect adversarial examples (Hendrycks & Gimpel, 2016; Grosse et al., 2017; Gong et al., 2017; Metzen et al., 2017; Feinman et al., 2017). Many of these methods, however, have been shown to fail (Carlini & Wagner, 2017a;b; Athalye et al., 2018).

Recent studies (Fawzi et al., 2018; Gilmer et al., 2018; Shafahi et al., 2018; Mahloujifar et al., 2019) show that adversarial examples can be an unavoidable consequence of learning high-dimensional geometry of data manifold in a statistical setting. In this vein, the study (Schmidt et al., 2018) proved that, for any training algorithm or model class, the sample complexity of robust learning can be significantly larger than that of "standard" learning. In other words, one theoretically grounded way to increase adversarial robustness is to acquire more data. This partially explains why the adversarial training (Goodfellow et al., 2014; Kurakin et al., 2016; Madry et al., 2017), a data augmentation technique, is empirically strong and also motivates a new defense approach (Dubey et al., 2019) for convolutional neural networks (CNNs) that relies on a web-scale amount of external images.

In this paper, we study whether it is possible to improve the adversarial robustness of a DNN by leveraging the potentially large test data seen at runtime. Currently, most existing defenses use only data known at training time. A key challenge of learning from test data is that they are unlabeled, so one cannot simply apply the adversarial training in an online manner. We devise a new defense method, called *runtime masking and cleansing* (RMC), that fine-tunes the network weights for several gradient-descent steps before making a prediction using precomputed adversarial examples residing near the test instance in the input space. This dynamically masks the network gradients and cleanses the model of the non-robust patterns inevitably learned during the training process due to the size limit of the training set. The following summarizes our contributions:

- We propose the runtime masking and cleansing (RMC), which uses test data to improve the adversarial robustness of a model after deployment. The RMC is compatible with any existing defense technique running at training time. To the best of our knowledge, this is the first work on robust learning using unlabeled test data.
- We conduct extensive experiments to evaluate the performance of RMC. Empirical results on real-world datasets show that it significantly improves the adversarial robustness of a network at the cost of delays in making predictions, which may be acceptable to non-realtime applications.
- We also propose two new attack methods against RMC and demonstrate that the RMC is hard to break due to some practical limitations faced by adversaries on a deployed system.

The RMC has implications for existing machine learning systems in production. In particular, it does not require a model to be retrained to defend a new attack discovered after deployment, thereby shortening the "vulnerable windows" of a system. We will discuss this in more details in the last section of this paper.

2. Related Work

There is a large body of work on adversarial robustness, attack, and defense. Please refer to (Yuan et al., 2019; Xu et al., 2019) for comprehensive surveys. Here, we discuss the works most closely related to ours.

Getting More Data. A theoretically grounded way to increase the adversarial robustness of a model is to acquire more data to offset the increased sample complexity of robust learning (Schmidt et al., 2018). (Dubey et al., 2019) proposed using a web-scale image database containing tens of billions of images as a manifold and projecting a test image (a potential adversarial example) onto the manifold. By taking only the projected images as input, the model can make more robust predictions. However, web-scale data may not be easily collectable in other domains, and the performance of this method drops significantly when the size of the database decreases (Dubey et al., 2019).

Adversarial Training. The adversarial training (Goodfellow et al., 2014; Kurakin et al., 2016; Madry et al., 2017) is another way to acquire data to increase robustness. Given a training dataset $\mathbb{D} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_i$ and a loss function L for a classification task, an adversarial training algorithm aims to find a network parametrized by $\boldsymbol{\theta}^*$ that classifies adversarial examples correctly, i.e.,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathbb{D}} \max_{\boldsymbol{\delta}\in\mathbb{A}(\boldsymbol{x})} L(\boldsymbol{\theta}, \boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}), \tag{1}$$

where $\mathbb{A}(\boldsymbol{x})$ is the set of allowed perturbations for a data point $\boldsymbol{x} \in \mathbb{D}$ and is defined by an *attack model*.[1] The adversarial examples provide extra information about the underlying data manifold and make the decision boundary of the network smooth along the directions of adversarial perturbations. Adversarial Training is one of the few defense techniques that withstand strong attacks such as the projected gradient descent (PGD) (Madry et al., 2017).

Gradient Masking. The RMC can be regarded as a gradient masking method (Yuan et al., 2019; Xu et al., 2019), where a defender deliberately hides the gradient information of the model in order to confuse the adversaries. Studies (Carlini & Wagner, 2017b; Athalye et al., 2018) have shown that many gradient masking methods can be defeated by attacking algorithms that simulate a masking mechanism using a differentiable component. The RMC differs from most existing methods in that 1) it hides the gradients of the model *at runtime* where new data keep coming in. It is hard to simulate the masking because the masking itself is changing. We will discuss more on this in Section 5. 2) RMC not only hides the gradients but helps the model unlearn non-robust features through more data.

Semi-Supervised Learning. Recent studies (Carmon et al., 2019; Stanforth et al., 2019; Najafi et al., 2019) show that learning from unlabeled data at training time can improve model robustness because the model has a better understanding of the data distribution in the input space that is useful for differentiating adversarial examples. RMC uses unlabeled data coming *at runtime*. Unlike the train-time semi-supervised learning methods whose goal is to use unlabeled data to better learn the underlying data distribution or the relationship between data points and labels, our goal is to use unlabeled data to *unlearn* patterns that are harmful to adversarial robustness (i.e., to cleanse the model).

3. Runtime Masking and Cleansing

In this section, we present the runtime masking and cleansing (RMC). Although the RMC can be readily applied to different supervised learning tasks, we study a robust classification problem defined below for ease of presentation.

Problem 1. Given an attack model, a training set $\mathbb{D} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^N$ containing benign examples, and a deployed network f parametrized by $\boldsymbol{\theta}^*$ trained by minimizing a loss function L over $\mathbb{D}$, our goal is to make robust predictions for a sequence of data points $\mathbb{U} = \{\hat{\boldsymbol{x}}^{(i)}\}_{i=1}^M$ seen at runtime such that even if a point $\hat{\boldsymbol{x}} \in \mathbb{U}$ has been adversarially perturbed by an adversary using the attack model, we can still correctly predict its label.

[1] An attack model (Madry et al., 2017) specifies known attack methods and some constraints such as the maximum allowable norm of the perturbations.

Algorithm 1 Runtime Masking and Cleansing (RMC).

1: **procedure** PREPARE(L, $\boldsymbol{\theta}^*$, $\mathbb{D}$, attack model, N')
2: Initiate $\mathbb{D}' = \mathbb{D}$;
3: **repeat**
4: Randomly sample $(\boldsymbol{x}, \boldsymbol{y})$ from $\mathbb{D}$;
5: $\boldsymbol{x}' \leftarrow \boldsymbol{x} + \arg\max_{\boldsymbol{\delta} \in \mathbb{A}(\boldsymbol{x})} L(\boldsymbol{\theta}^*, \boldsymbol{x} + \boldsymbol{\delta}, \boldsymbol{y})$;
6: $\mathbb{D}' = \mathbb{D}' \cup \{(\boldsymbol{x}', \boldsymbol{y})\}$;
7: **until** $|\mathbb{D}'| = N'$
8: **return** $\mathbb{D}'$;
9: **end procedure**
10:
11: **procedure** PREDICT($\hat{\boldsymbol{x}}$, $f(\cdot\,; \boldsymbol{\theta}^*)$, L, $\mathbb{D}$, $\mathbb{D}'$, K, λ)
12: Search $\mathbb{D}'$ to get $\mathbb{N}'(\hat{\boldsymbol{x}})$ of size K;
13: **repeat** ▷ local adaptation
14: Sample a batch $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{B}$ from $\mathbb{N}'(\hat{\boldsymbol{x}})$;
15: $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}^* - \lambda \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}^*} L(\boldsymbol{\theta}^*, \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$;
16: **until** early-stop condition holds
17: $\boldsymbol{\theta}' = \boldsymbol{\theta}^*$;
18: **repeat** ▷ calibration
19: Sample a batch $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{B}$ from $\mathbb{D}$;
20: $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}^* - \lambda \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}^*} L(\boldsymbol{\theta}^*, \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$;
21: **until** early-stop condition holds
22: **return** $f(\hat{\boldsymbol{x}}; \boldsymbol{\theta}')$ and $\boldsymbol{\theta}^*$;
23: **end procedure**

Note that the deployed model can be regularly or adversarially trained, and the size of $\mathbb{U}$ can be unbounded. In practice, one may also expect the network to have a certain level of robustness to attacks not defined in the attack model.

The basic idea of RMC is that, for each new coming point $\hat{\boldsymbol{x}} \in \mathbb{U}$, we adapt the network weights $\boldsymbol{\theta}^*$ to $\hat{\boldsymbol{x}}$ at runtime before making a prediction $f(\hat{\boldsymbol{x}}; \boldsymbol{\theta}^*)$. So, the network can dynamically hide its gradients from $\hat{\boldsymbol{x}}$ (a potential attack) and in the meanwhile learn from the potentially large $\mathbb{U}$ over time to become more robust statistically. Now, the challenge is to design the adaptation process.

Since the data in $\mathbb{U}$ is unlabeled, performing the adversarial training using $\mathbb{U}$ at runtime is infeasible because the attack model cannot compute adversarial examples without $\boldsymbol{y}$ when solving $\arg\max_{\boldsymbol{\delta} \in \mathbb{A}(\hat{\boldsymbol{x}})} L(\boldsymbol{\theta}, \hat{\boldsymbol{x}} + \boldsymbol{\delta}, \boldsymbol{y})$ in Eq. (1). One native way to work around this is to 1) find the top-K nearest neighbors of $\hat{\boldsymbol{x}}$ in $\mathbb{D}$ in a deep, latent space of the network (Papernot & McDaniel, 2018; Xie et al., 2019) and 2) use these labeled neighbors, denoted by $\mathbb{N}(\hat{\boldsymbol{x}})$, to adapt the network following the adversarial training.

However, the above naive online extension of the adversarial training does *not* work because $\hat{\boldsymbol{x}}$ could have been adversarially perturbed. Figures 1(a)(b) give an example showing the distributions of data points and labels in $\mathbb{N}(\hat{\boldsymbol{x}})$ provided that $\hat{\boldsymbol{x}}$ is an adversarially perturbed "automobile" image from the CIFAR-10 dataset (Krizhevsky et al., 2009). The labels of benign examples (blue points) concentrate at the wrong class "truck." This is because $\hat{\boldsymbol{x}}$ fools the network by looking similar to other "truck" images in the latent space. The wrong benign examples also lead to incorrectly labeled adversarial examples (cyan points). After local adaptation (lines 8-11), the network will be misled by $\mathbb{N}(\hat{\boldsymbol{x}})$ to wrongly assign the "truck" label to an input region around $\hat{\boldsymbol{x}}$, worsening robustness.

We work around the above problem by *precomputing* adversarial examples. The RMC first augments $\mathbb{D}$ with adversarial examples (by solving $\arg\max_{\boldsymbol{\delta} \in \mathbb{A}(\boldsymbol{x})} L(\boldsymbol{\theta}^*, \boldsymbol{x} + \boldsymbol{\delta}, \boldsymbol{y})$ for each example $(\boldsymbol{x}, \boldsymbol{y})$ in $\mathbb{D}$) to obtain an augmented dataset $\mathbb{D}'$, and then, for each new coming point $\hat{\boldsymbol{x}} \in \mathbb{U}$, adapts the network by solving

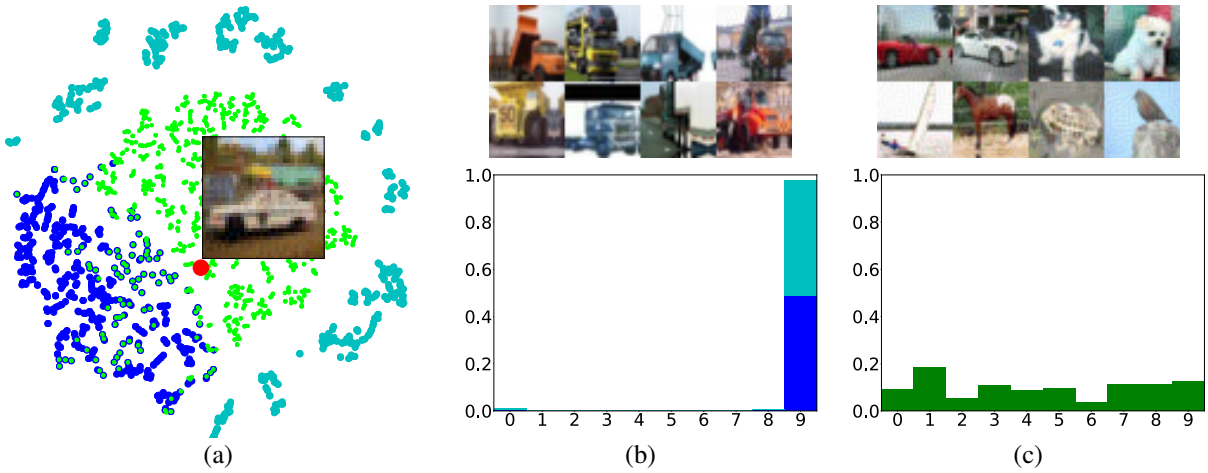
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{N}'(\hat{\boldsymbol{x}})} L(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}), \qquad (2)$$

where $\mathbb{N}'(\hat{\boldsymbol{x}})$ is the set of examples in $\mathbb{D}'$ that are top-K nearest to $\hat{\boldsymbol{x}}$ in a distance measure. Algorithm 1 outlines the key steps of RMC.

The $\mathbb{N}'(\hat{\boldsymbol{x}})$ may contain both benign and adversarial examples, and we use the Euclidean distance in the feature space of a deep (convolution) layer of the network to find the top-K nearest neighbors of $\hat{\boldsymbol{x}}$. A feature vector at a deep convolution layer usually encodes high-level concepts. Furthermore, studies (Papernot & McDaniel, 2018; Xie et al., 2019) have shown that the feature vector of an adversarially perturbed input can be drastically changed at a deep layer. This allows $\mathbb{N}'(\hat{\boldsymbol{x}})$ to have semantically similar data points, from the network point of view, but diverse labels.

Following the previous example, Figure 1(c) shows the distribution of labels in $\mathbb{N}'(\hat{\boldsymbol{x}})$ in Algorithm 1, which no longer concentrate at a single class. Line 5 in Algorithm 1 creates a non-targeted attack, that is, the hidden representation of $\boldsymbol{x}'$ can look similar to those of images in *any* class except $\boldsymbol{y}$. Therefore, the adversarial examples of the same class in $\mathbb{D}'$ may scatter over input regions of other classes. Equivalently for any given position in the input space, there may be nearby adversarial examples coming from different classes, as Figure 1(c) shows. When the network is adapted to $\mathbb{N}'(\hat{\boldsymbol{x}})$ (lines 13-16), such diversely labeled adversarial examples will cleanse the network by letting it *unlearn* the non-robust patterns encoded in the adversarial perturbations because these patterns are now associated with noisy labels and useless for reducing L. The network will end up relying more on benign patterns to make predictions.

We call lines 13-16 in Algorithm 1 the *local adaptation* procedure because the network is fine-tuned using examples local to the test point $\hat{\boldsymbol{x}}$. To prevent overfitting, we separate a validation set from $\mathbb{N}'(\hat{\boldsymbol{x}})$ and early stop the local adaptation when the validation loss does not decrease over time. Although making a prediction using the model weights $\boldsymbol{\theta}'$ obtained right after the local adaptation (line 22), the algo-



(a) (b) (c)

Figure 1. Statistics of the nearest neighbors of $\hat{\boldsymbol{x}}$ where $K = 1024$ and $\hat{\boldsymbol{x}}$ is an adversarially perturbed "automobile" (class 1) image from the CIFAR-10 dataset (Krizhevsky et al., 2009). The perturbation was computed using the PGD attack (Madry et al., 2017). (a) Distributions of the nearest neighbors in a 2D t-SNE (Maaten & Hinton, 2008) projection of the feature space where the distance measure is defined. The red dot denotes $\hat{\boldsymbol{x}}$, blue and cyan dots denote benign and adversarial examples in $\mathbb{N}(\hat{\boldsymbol{x}})$ in the naive extension of the adversarial training, respectively, and green dots denote examples in $\mathbb{N}'(\hat{\boldsymbol{x}})$ in Algorithm 1. (b) Distribution of class labels of the examples in $\mathbb{N}(\hat{\boldsymbol{x}})$ and sample images. The labels concentrate at a wrong class "truck" (class 9). (c) Distribution of class labels of the examples in $\mathbb{N}'(\hat{\boldsymbol{x}})$ in Algorithm 1. The labels scatter over different classes.

rithm performs a *calibration* procedure (lines 18-21) before the end. This is because the local examples in $\mathbb{N}'(\hat{\boldsymbol{x}})$ may violate the i.i.d. assumption. As compared to an alternative approach that discards the $\boldsymbol{\theta}'$ and starts over from a static $\boldsymbol{\theta}^*$ for the next test instance, the calibration procedure prevents the side-effects of the violation but still allows the network to learn from $\mathbb{U}$ in the long term.

Although being cosmetically similar to the adversarial training, the RMC works fundamentally differently. While the adversarial training makes the decision boundary of a model smooth along the directions of adversarial perturbations, the RMC cleanses the model by letting the model unlearn non-robust patterns, which does not necessarily result in a smooth decision boundary. The adversarial training and RMC are orthogonal yet complementary defenses working at training time and runtime, respectively.

4. Experiments

In this section, we evaluate the performance of RMC using the robust classification task defined in Problem 1 on the MNIST (LeCun & Cortes, 2010), CIFAR-10 (Krizhevsky et al., 2009), and ImageNet (Deng et al., 2009) datasets. Our code is available at `https://github.com/nthu-datalab/Runtime-Masking-and-Cleansing`.

Trained Networks. We build the classification (defense) network $f(\cdot;\boldsymbol{\theta}^*)$ by following the settings in (Madry et al., 2017) and (Xie et al., 2019), where ResNet-32 and ResNet-152 are used for CIFAR-10 and ImageNet, respectively. The RMC works with either a regularly or adversarially trained model. Therefore, for the CIFAR-10 dataset, we train the above networks using 1) regular training, 2) adversarial training with FGSM (Goodfellow et al., 2014), 3) adversarial training with PGD (Madry et al., 2017), 4) regular training with Jacobian regularization (Jakubovitz & Giryes, 2018), and 5) regular training with Cross-Lipschitz regularization (Hein & Andriushchenko, 2017). For ImageNet, we use a regularly trained ResNet-152.[2]

Baselines. We compare the RMC with two baseline defenses that can run at test time. The first baseline, which we call WebNN, is a down-scaled version of the work (Dubey et al., 2019) where the web-scale image database is replaced with the augmented training dataset $\mathbb{D}'$ used by RMC. The second baseline, which we call DeepNN (Papernot & McDaniel, 2018), makes a prediction $f(\hat{\boldsymbol{x}};\boldsymbol{\theta}^*)$ for a test point $\hat{\boldsymbol{x}}$ by first searching the top-K nearest neighbors of $\hat{\boldsymbol{x}}$ from the original dataset $\mathbb{D}$ and then averaging the output vectors of these neighbors. Note that the Deep-NN uses the Euclidian distance in the feature space of the deepest CNN layer to find the nearest neighbors, which is the same as the RMC but different from the original paper (Papernot & McDaniel, 2018). For the ImageNet dataset, we also compare RMC with a train-time defense called the denoising block (DB) (Xie et al., 2019).

Settings. We set K= 1024 and 2048 for CIFAR-10 and

[2]See `https://www.tensorflow.org/api_docs/python/tf/keras/applications/ResNet152`.

Table 1. The performance of different defenses under different train-time white-box attacks ($\epsilon = 8/255$) on CIFAR-10.

	Acc.	Robustness				
		FGSM	BIM	PGD	CW-L2	JSMA
Regularly Trained						
None	83.3	25.3	8.5	6.7	9.4	8
DeepNN	84.3	26.5	9.2	8	55.2	23
WebNN	81.8	40.9	47.8	48.6	64.6	38.3
RMC	**89.3**	**85.3**	**86.7**	**87.5**	**89.7**	**88.6**
Adversarially Trained w. FGSM						
None	83.2	78.9	9.3	8.3	8.8	17.3
DeepNN	85	81	9.9	9.1	56.2	23.1
WebNN	80	81.9	42.5	43.3	64.2	34.4
RMC	**89.3**	**87.3**	**87.1**	**88.7**	**89.7**	**89.1**
Adversarially Trained w. PGD						
None	78.7	50.6	43.6	44.3	11.5	7.8
DeepNN	75.6	52.5	45.6	45.8	48.7	38.5
WebNN	73.5	54	48.1	48.4	53.4	47
RMC	**88.3**	**81.2**	**81.1**	**80.7**	**88.7**	**87.7**
Regularly Trained w. Jacobbian Reg.						
None	86.3	37.9	20.6	20.2	8	10.2
DeepNN	**87.8**	39.8	21	21.4	63.1	41.1
WebNN	76.2	49.9	55.5	55.5	68.9	49
RMC	87.1	**82.4**	**83.6**	**83.5**	**86.6**	**88.4**
Regularly Trained w. Cross-Lipschitz Reg.						
None	85.3	31	18.6	18.4	8.4	13
DeepNN	**86.9**	32.6	19	19	61.9	36.8
WebNN	74.5	46.5	51	50.5	67.1	48.6
RMC	85	**79.8**	**80.8**	**81.1**	**84.9**	**86.9**

ImageNet respectively. To prevent the local adaptation in RMC from overfitting, we separate the top-20% nearest neighboring examples from $\mathbb{N}'(\hat{\boldsymbol{x}})$ as the validation set and early stop the local adaptation when the validation loss does not decrease over time. We use the Adam optimizer (Kingma & Ba, 2014) to train and adapt the models. The initial learning rate for local adaptation is set to 25% of that used by a model during the training time.

4.1. Train-Time White-Box Settings

First, we evaluate the robustness of different runtime defense techniques by using the train-time white-box settings where an adversary has access to the model, including its architecture and weights, after training. In the next section, we will use more aggressive settings where the input of RMC, such as $\mathbb{D}'$ and $\mathbb{U}$, is leaked to the adversaries.

We take into account some well-known attack models, including the FGSM(Goodfellow et al., 2014), BIM (Kurakin et al., 2016), PGD (Madry et al., 2017), CW-L2 attack (Carlini & Wagner, 2017b), and JSMA (Papernot et al., 2016a).

Table 2. The performance of different defenses under different train-time white-box attacks on ImageNet.

	Acc.	Robustness	
		$\epsilon = 8/255$	$\epsilon = 16/255$
None	72.9	8.5	5.2
Adv. Trained	62.3	N/A	52.5
DB	65.3	N/A	55.7
DeepNN	26.6	12.9	8.7
WebNN	27.8	18.8	15.2
RMC	**73.6**	**62.4**	**55.9**

We use FGSM, BIM, and PGD to create non-targeted attacks, and CW-L2 and JSMA to create targeted attacks. We use the open-source CleverHans library (Papernot et al., 2018) to implement these attacks. For each attack model, we augment $\mathbb{D}$ using the adversarial examples generated by the attack to get $\mathbb{D}'$, and we generate a sequence $\mathbb{U}$ of adversarial test examples by perturbing the benign test examples against the trained model weights. We report the robustness using the prediction accuracy over $\mathbb{U}$.

CIFAR-10. We set $N' = 5|\mathbb{D}|$ so $\mathbb{D}'$ contains $|\mathbb{D}|$ clean and $4|\mathbb{D}|$ adversarial examples. We set the maximum allowable perturbation, ϵ, to $8/255$ for gradient-based attacks (FGSM, BIM and PGD). The learning rate of CW-L2 attack is 0.002. Table 1 shows the clean accuracy and robustness of different models paired up with different runtime defenses. For WebNN and RMC which uses $\mathbb{D}'$, we report the clean accuracy by averaging the results under different attacks. As we can see, the RMC achieves state-of-the-art performance and performs significantly better than all other baseline defenses in all combinations of the trained networks and attacks. The networks without any runtime defense have the worst robustness. This shows that runtime defenses can indeed improve robustness. The DeepNN and WebNN, which dynamically mask the gradients of a network at runtime, give improved performance but are not on par with the RMC. This justifies that the local adaptation process of the RMC works not only by gradient masking but also by model cleansing. Note that the clean accuracy of RMC is high in all cases. This shows that the cleansing, which does not necessarily result in a smooth decision boundary in a model as described in Section 3, is beneficial in practice.

ImageNet. We set $N' = 4|\mathbb{D}|$ and experiment on both $\epsilon = 8/255$ and $\epsilon = 16/255$. Following the settings in (Athalye et al., 2018; Kannan et al., 2018; Xie et al., 2019), we consider only the targeted attacks here.[3] We run the PGD attack for 10 descent iterations. In addition to the runtime defenses, we compare RMC with the train-time defense called denoising block (DB) because it is currently

[3]See the supplementary file for the results under non-targeted attacks.

Table 3. The performance of different defenses under different gray- and black-box attacks ($\epsilon = 8/255$) on CIFAR-10.

	Acc.	Robustness			
		Reg. ResNet		Reg. wResNet	
		FGSM	MIM	FGSM	MIM
None	83.3	39.5	21	68.8	67.2
A.T. (FGSM)	83.2	73.4	69.9	69.4	67.9
A.T. (PGD)	85.3	70.5	68.7	72.7	70.4
DeepNN	84.3	41.8	20.6	72.2	**70.5**
WebNN	80.2	45.4	23.4	66.7	65.6
RMC	**85.7**	**77.7**	**74.4**	**78.2**	68.4

the state-of-the-art on ImageNet. As in (Dubey et al., 2019), we find the top-K nearest neighbors using a similar measure defined in the feature space of the `conv_5_1` layer of the ResNet-152. Table 2 shows the results. The numbers of the adversarial training and DB are extracted from the original paper (Xie et al., 2019). We can see that the RMC achieves comparable (if not better) robustness than DB. This, again, justifies that the local adaptation process gives a strong gradient masking and cleansing effects. In addition, the RMC yields significantly higher clean accuracy than other baselines, which demonstrates the benefit of allowing a model to have a non-smooth decision boundary during cleansing. Note that the WebNN, a runtime gradient masking method, does not perform well on ImageNet because the $\mathbb{D}'$ we used (of size 10^6) is much smaller than that (of size 10^9) used in the original paper (Dubey et al., 2019). On the other hand, the RMC does not require a large $\mathbb{D}'$ to work thanks to the cleansing effect in addition to the gradient masking.

4.2. Gray- and Black-Box Settings

In black-box settings, an adversary does not have access to the information about the target model, such as its architecture and weights. Following (Madry et al., 2017), we create a gray- and black-box attacks by regularly training a ResNet-32 (denoted by "Reg. ResNet") of different weights than our target model and a wide ResNet (denoted by "Reg. wResNet") with 10 times of filters at some layers, respectively, on CIFAR-10. The results, which are shown in Table 3, empirically justifies that the RMC can still perform well under gray- or black-box attacks.

5. Defense-Aware Attacks

We develop two types of defense-aware attacks aiming to break RMC. The first type of attacks targets the selection of top-K nearest neighboring examples in $\mathbb{N}'(\hat{\boldsymbol{x}})$ (line 12 in Algorithm 1), and the second type tries to bypass the local adaptation (lines 13-16). In addition to the information revealed by the train-time white-box settings, we allow an attacker to partially or totally access the information used by the RMC at runtime. Unless mentioned specifically, we follow the settings for CIFAR-10 in Section 4.1. For ease of presentation, we consider the regularly trained model here.

Table 4. The robustness of runtime defenses under the PGD-NN attack. Although the PGD-NN affects the selection of nearest neighboring examples, the RMC can still defend PGD-NN.

	None	DeepNN	WebNN	RMC
PGD	6.7	8	48.6	**86.6**
PGD-NN	11.3	11.4	1.5	**75.4**

5.1. PGD-NN

Assuming that the augmented training dataset $\mathbb{D}'$ is exposed, we modify the PGD attack (Madry et al., 2017), which we call PGD-NN, to generate an adversarial test instance $\hat{\boldsymbol{x}}$ whose goal is to let $\mathbb{N}'(\hat{\boldsymbol{x}})$ contain "wrong" examples such that the local adaptation of RMC will mislead the network into a wrong prediction.[4] In PGD-NN, we compute the adversarial perturbation for a benign test example $(\hat{\boldsymbol{x}}, \boldsymbol{y})$ by solving

$$\arg\max_{\boldsymbol{\delta} \in \mathbb{A}(\hat{\boldsymbol{x}})} H\Big(\frac{1}{K} \sum_{\boldsymbol{x} \in \mathbb{N}'(\hat{\boldsymbol{x}}+\delta)} f(\boldsymbol{x}; \boldsymbol{\theta}^*), \boldsymbol{y}\Big),$$

where $H(\cdot\,,\,\cdot)$ is the cross entropy. We then add the perturbation back to $\hat{\boldsymbol{x}}$. In effect, the perturbed $\hat{\boldsymbol{x}}$ will let $\mathbb{N}(\hat{\boldsymbol{x}})$ contain as few examples of class $\boldsymbol{y}$ as possible. Note that we use SGD to solve the above problem. When computing the gradient at each iteration, we let the error signal back-propagated *though* the operation of finding the K nearest neighbors from $\mathbb{D}'$.

The results are shown in Table 4. We can see that the DeepNN and WebNN fail to defend the PGD-NN attack as the latter can successfully mislead the selection of top-K nearest neighboring examples. The RMC also gives degraded robustness under this attack. However, the performance drop is much less than the other baselines. We find that, although containing less examples of the correct labels, the $\mathbb{N}'(\hat{\boldsymbol{x}})$ still includes adversarial examples that can help the model cleansing. This allows the RMC to withstand the PGD-NN attack.

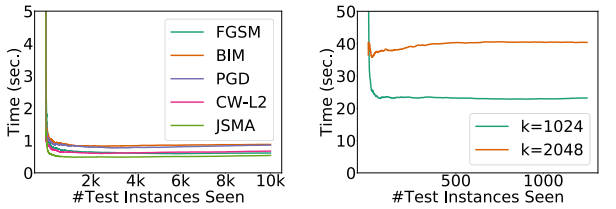
5.2. PGD-Skip

Next, we consider an even stronger attack by assuming a complete white-box setting where all information, including the test sequence $\mathbb{U}$, is known by the adversary. We propose the PGD-Skip attack that uses PGD to compute an adversarial point $\hat{\boldsymbol{x}}^{(p+1)}$ (the $(p+1)$-th test input seen

[4] We also take into account a variant of the PGD-NN attack and study the performance of RMC. Please see the supplementary for more details.

Algorithm 2 The PREDICT procedure of RMC$^+$.

procedure PREDICT($\hat{\boldsymbol{x}}$, $f(\cdot;\boldsymbol{\theta}^*)$, L, $\mathbb{D}$, $\mathbb{D}'$, K, λ)
  Search $\mathbb{D}'$ to get $\mathbb{N}(\hat{\boldsymbol{x}})$ of size K;
  $\cdots$ ▷ local adaptation
  $\boldsymbol{\theta}' = \boldsymbol{\theta}^*$;
  $\cdots$ ▷ calibration
  Update $\mathbb{D}'$ such that the adversarial examples in $\mathbb{N}'(\hat{\boldsymbol{x}})$ are recomputed using $\boldsymbol{\theta}'$;
  return $f(\hat{\boldsymbol{x}};\boldsymbol{\theta}')$ and $\boldsymbol{\theta}^*$;
end procedure



(a) CIFAR-10 (b) ImageNet

Figure 2. The inference delay given different numbers of test instances seen by the network at runtime on the (a) CIFAR-10 and (b) ImageNet datasets.

at runtime) against the network that has been adapted to $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p)} \in \mathbb{U}$. So, the PGD-Skip will bypass all RMC effects due to $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p)}$.

The PGD-Skip can successfully defeat RMC. Under such a strong attack, the RMC gives only 14.9% robustness. Nevertheless, the number is still better than the 6.7% given by the regularly trained network without any runtime defense. See the supplementary file for more details.

The PGD-Skip, however, may be infeasible in practice because it uses two strong assumptions: 1) the adversary can have access to all data points seen at runtime, and 2) there is no other data point coming in between $\hat{\boldsymbol{x}}^{(p)}$ and $\hat{\boldsymbol{x}}^{(p+1)}$. When a model is publicly deployed on, for example, the World Wilde Web, it is unlikely that an adversary can eavesdrop every routing packets coming from the worldwide. Furthermore, the adversary can hardly mute all requests from other users so that the attack point can be placed at the position $p+1$.[5] In another example, if the packets may be encrypted so the adversary may take a while to decrypt the packets to get plaintext data points. Therefore, we consider two weakened but more practical attacks, which we call the PGD-Skip-Partial and PGD-Skip-Delayed. In PGD-Skip-Partial, only partial data points in $\mathbb{U}$ are known, and the known points are out of order. The adversary runs local adaptation over the known points to get an approximate network based on which the adversarial perturbations are computed. In PGD-Skip-Delayed, the adversary knows $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p)}$ but can only place the generated adversarial point at $\hat{\boldsymbol{x}}^{(p+q+1)}$, where q reflects the delay of knowing $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p)}$ and/or computing adversarial perturbations.

To successfully defend PGD-Skip attacks, the model needs to have a strong masking and/or cleansing effect even when only seeing one extra testing instance. This can be achieved by either 1) using a larger learning rate during the local adaptation or 2) continuously adding new information in $\mathbb{D}'$ via either new adversarial examples or new benign examples. Here we show the results of the latter approach. Algorithm 2 outlines the extended version of RMC, called the RMC$^+$, that updates $\mathbb{D}'$ by recomputing the adversarial examples in $\mathbb{N}'(\hat{\boldsymbol{x}})$ based on the adapted weights $\boldsymbol{\theta}'$. We introduce a hyperparameter δ that controls the portion of the adversarial examples in $\mathbb{N}'(\hat{\boldsymbol{x}})$ to replace.

Table 5 shows the performance of RMC$^+$. As we can see (Table 5(a)), the RMC$^+$ can successfully defend the PGD-Skip-Delayed attack whenever the delay (q) is longer than the arrival interval of 50 test instances. Furthermore, the RMC$^+$ can defend the PGD-Skip-Partial whenever the attacker knows less than 70% of the test instances in $\mathbb{U}$ (Table 5(b)). In particular, in most of the success cases, the RMC$^+$ gives comparable or better robustness than the adversarial training.

6. More Experiments

Next, we investigate the behavior of RMC in more dimensions. We follow the settings in Section 4.1 and use the non-targeted PGD attack with $\epsilon = 8/255$ and $\epsilon = 16/255$ for the CIFAR-10 and ImageNet datasets, respectively.

How long is the delay incurred by RMC at runtime? We test the delay incurred by the RMC at runtime on a machine with an NVIDIA V100 GPU. We set the batch size for adaptation to 128 and 1024 on CIFAR-10 and ImageNet, respectively. Figure 2 shows the results. Generally, the RMC gives a delay of about 1 second on CIFAR-10 and a delay of 20-40 seconds on ImageNet, which may be acceptable for non-realtime applications. We believe that the RMC can greatly benefit from existing acceleration techniques. This is left as our future work.

How does the performance of RMC change when a model sees more test data points? Figure 3 shows the average robustness and steps of local adaptation given different numbers of test instances seen by the network at runtime on the CIFAR-10 dataset. We found that before giving a relatively stable performance, the RMC takes about tens of instances to warm up. This is because there are

[5] Many other practical concerns invalidate the assumptions used by PGD-Skip. For example, if the packets sent by the users have been encrypted, it will take time for the adversary to decrypt the packets to eavesdrop $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p)}$ and then place an attack point.

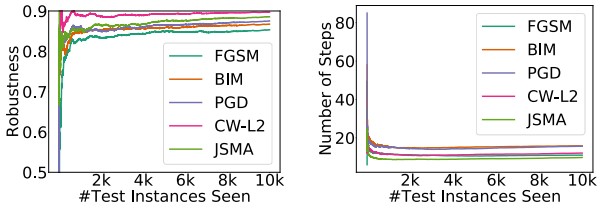
Table 5. The robustness of RMC$^+$ under the (a) PGD-Skip-Delayed and (b) PGD-Skip-Partial attacks.

	δ = 0.5			δ = 0.75			δ = 1		
q	**0**	**50**	**100**	**0**	**50**	**100**	**0**	**50**	**100**
$p = 50$	19.3	51	63.7	20.4	48.9	62.8	20.9	44.1	48.6
$p = 100$	25.3	50.8	55.1	25.5	51.5	56.1	39.5	41	30.6

(a) PGD-Skip-Delayed with $\mathbb{D}'$ replacement

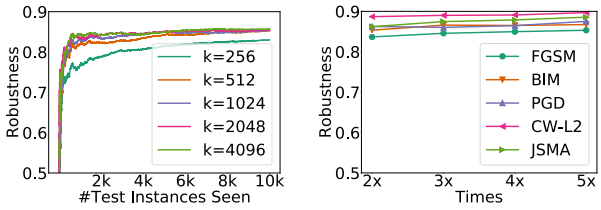
	δ = 0.5			δ = 0.75			δ = 1		
known	**30%**	**50%**	**70%**	**30%**	**50%**	**70%**	**30%**	**50%**	**70%**
$p = 50$	48.4	48.1	45.2	47.5	49	43.3	50.4	52.4	49.5
$p = 100$	64.1	63.1	63.5	64.3	61.1	59.4	63.3	61.7	61.8
$p = 150$	69.2	69.2	68.5	68.9	68.7	68.3	59.6	61.1	64.8

(b) PGD-Skip-Partial with $\mathbb{D}'$ replacement



(a) Robustness (b) Steps

Figure 3. Higher robustness can be achieved at a lower cost (in terms of adaptation steps and time) when the model sees more test points on CIFAR-10. (a) Robustness under different attacks. (b) The number of gradient descent steps of local adaptation.



(a) (b)

Figure 4. Sensitivity check of two hyperparameter K and $|\mathbb{D}'|$ on the CIFAR-10 dataset. (a) K-sensitivity check, which shows robustness with different size of K. (b) $|\mathbb{D}'|$-sensitivity check, indicating robustness with various $|\mathbb{D}'|$.

many non-robust patterns to unlearn in the first few rounds of local adaptation, as evidenced by the number of gradient descent steps of local adaptation for the very first instances. After the warm-up stage, the RMC gives a relatively stable, slightly increasing performance at a stable cost when the mode sees more test instances. This trend is consistent under different attacks.

How do the K and N' affect performance? Next, we study the effects of the hyperparameters K and N' in RMC. Figure 4(a) shows the robustness given by the RMC with different settings for K. As we can see, the RMC with a larger K requires less time to warm up and converges to higher robustness, although this benefit saturates at larger K's. Nevertheless, according to Figure 2, one should not use a too large K because it incurs a high delay without significantly improving robustness. We give similar results on ImageNet in the supplementary file.

Table 6. The white-box robustness under different attacking algorithms pairing with different $\mathbb{D}'$ on CIFAR-10. With Single-Attack $\mathbb{D}'$, RMC performs better than other types of $\mathbb{D}'$ under gradient-based attacking algorithms.

	FGSM	**BIM**	**PGD**	**CW-L2**	**JSMA**
None	11.6	0.6	0.5	0.7	14.1
Single-Attack $\mathbb{D}'$	85.3	86.7	87.5	89.7	88.6
All-In $\mathbb{D}'$	82.6	85.7	86.1	88.8	85.8
Leave-One-Out $\mathbb{D}'$	63.1	85.3	85.8	87.5	66.7

Figure 4(b) shows the robustness given by RMC with different $|\mathbb{D}'|$. In line with earlier work (Dubey et al., 2019), there is a positive correlation between the robustness and the number of precomputed examples in $\mathbb{D}'$. This result is consistent across different types of attacks, which supports the very first assumption of our work that requiring more data at runtime can improve robustness.

What if an adversarial test example is generated by an attack not considered in $\mathbb{D}'$? We evaluate the performance of RMC where the augmented dataset $\mathbb{D}'$ is configure differently on CIFAR-10. Specifically, we consider the variants of $\mathbb{D}'$ that contains 1) a single attack used by the attacker (denoted by "Single-Attack"), 2) all types of attacks shown in Figure 4(b), including the one used by the attacker (denoted by "All-In"), and 3) all types of attacks except the one used by the attacker (denoted by "Leave-One-Out"). As Table 6 shows, the effect of the specific types of adversarial examples included in $\mathbb{D}'$ is not significant except in the case of Leave-One-Out $\mathbb{D}'$ against the FGSM and JSMA attacks.

Table 7. Robustness of the non-masking RMC (i.e., the RMC without the local adaptation for the current test instance $\hat{\boldsymbol{x}}^{(p+q+1)}$) and the hindsight adversarial training with labeled $\mathbb{U}$ under the PGD-Skip attack (where $q = 0$ and 100% of the previous test instances $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p)}$ are known) on CIFAR-10.

p	**0**	**100**	**1000**	**5000**
Hindsight Adv. Training on Labeled $\mathbb{U}$	6.7	8.4	16.4	13.9
Non-masking RMC		11.7	14.3	26.2

The FGSM is known to be an effective attack against the defense based on gradient masking. And the JSMA, which is pixel-based, is very different from other gradient-based attacks.

How much robustness can be improved solely by cleansing? The RMC improves robustness via both the gradient masking and cleansing effects. To understand how much the cleansing contributes to the overall performance, we conduct an experiment that evaluates the performance of a *weakened* version of RMC where there's no gradient masking for the current test instance. Following the settings in Section 5.2, we consider the completely white-box PGD-Skip attack which assumes 1) the network to attack has been adapted to $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p)} \in \mathbb{U}$, 2) the attacker uses PGD to compute an adversarial point $\hat{\boldsymbol{x}}^{(p+q+1)}$, where q is the delay of obtaining/placing the point from/into $\mathbb{U}$, and 3) $q = 0$ and 100% of the previous test instances $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p)}$ are known by the attacker. We remove the gradient masking effect of RMC by *not* allowing it to perform the local adaptation for the test instance $\hat{\boldsymbol{x}}^{(p+q+1)}$ (but the adaptations for previous instances $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p)}$ are still allowed to accumulate the cleansing effect). We call this weakened version of the *non-masking RMC*. We also consider a baseline where the adversarial training is performed in hindsight, that is, we treat $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p+q)} \in \mathbb{U}$ as a new training set by revealing their correct labels and then, before predicting the label of $\hat{\boldsymbol{x}}^{(p+q+1)}$, fine-tune the network by running the adversarial training algorithm on the training set. To ensure that this baseline sees the same amount of adversarial examples as the no-masking RMC, we stop the adversarial training when $N' = 4|\mathbb{D}|$ adversarial examples have been generated.

Table 7 shows the results. As we can see, the non-masking RMC improves the robustness of a regularly trained model when more test instances are seen, which justifies the effectiveness of the standalone cleansing. Without gradient masking, the RMC can still function by teaching the model to unlearn non-robust patterns that were inevitably learned from the training set $\mathbb{D}$ under statistical settings. As compared with the hindsight adversarial training, the non-masking RMC has a smaller sample complexity because it improves the robustness faster as p increases. Furthermore, it does *not* utilize the labels of $\hat{\boldsymbol{x}}^{(1)}, \cdots, \hat{\boldsymbol{x}}^{(p)}$ in the ground truth and thus is more practical at runtime.

More Experiments. We also investigate the behavior of RMC in other aspects, such as targeted vs. non-targeted attacks, more defense-aware attacks, and particular datasets, etc. Please refer to the supplementary file for more details.

7. Implications and Conclusion

We propose the runtime masking and cleansing (RMC) that uses test data to improve the adversarial robustness of a model after deployment. To the best of our knowledge, RMC is the first adaptive defense at runtime. It is compatible with any existing defense technique running at training time. We conduct extensive experiments to evaluate the performance of RMC. Empirical results on real-world datasets show that it significantly improves the adversarial robustness of a network at the cost of delays in making predictions, which may be acceptable to non-realtime applications. We also propose two new attack methods against RMC and demonstrate that the RMC is hard to break due to practical limitations faced by adversaries on a production system.

The RMC has implications for the lifecycle of a model on existing machine learning systems. Since the discovery of adversarial examples, the attack and defense techniques have been chasing each other. A robustly trained and deployed model may become vulnerable after a new attack is proposed. Currently, there is a "vulnerable window" before the model incorporates further defenses and is redeployed. The RMC can greatly reduce a vulnerable window because it does not require the model to be retrained to defend the new attack. Instead, only the $\mathbb{D}'$ needs to be updated, which can be done asynchronously and much faster than the adversarial training.

As our future work, we plan to investigate approaches that can accelerate the local adaptation procedure and reduce the delay incurred by RMC. Another direction to explore is the attack model. Is it possible to make an attack *adaptive* (at runtime) to break an adaptive defense like RMC? In particular, can an adversary perturb *a sequence of* test points instead of just one to create stronger attacks? This is our future inquiry. We also plan to study the performance of RMC on real-world applications, such as face recognition authorization, where security is in high demand.

References

Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018. 1, 2, 4.1

Carlini, N. and Wagner, D. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proc. of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017a. 1

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *Proc. of S&P*, 2017b. 1, 2, 4.1

Carmon, Y., Raghunathan, A., Schmidt, L., Duchi, J. C., and Liang, P. S. Unlabeled data improves adversarial robustness. In *Proc. of NeurIPS*, 2019. 2

Cisse, M., Bojanowski, P., Grave, E., Dauphin, Y., and Usunier, N. Parseval networks: Improving robustness to adversarial examples. In *Proc. of ICML*. JMLR. org, 2017. 1

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Prof. of CVPR*, 2009. 4

Dubey, A., Maaten, L. v. d., Yalniz, Z., Li, Y., and Mahajan, D. Defense against adversarial images using web-scale nearest-neighbor search. In *Proc. of CVPR*, 2019. 1, 2, 4, 4.1, 6

Fawzi, A., Fawzi, H., and Fawzi, O. Adversarial vulnerability for any classifier. In *Prof. of NeurIPS*, 2018. 1

Feinman, R., Curtin, R. R., Shintre, S., and Gardner, A. B. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017. 1

Gilmer, J., Metz, L., Faghri, F., Schoenholz, S. S., Raghu, M., Wattenberg, M., and Goodfellow, I. Adversarial spheres. *arXiv preprint arXiv:1801.02774*, 2018. 1

Gong, Z., Wang, W., and Ku, W.-S. Adversarial and clean data are not twins. *arXiv preprint arXiv:1704.04960*, 2017. 1

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1, 2, 4, 4.1

Grosse, K., Manoharan, P., Papernot, N., Backes, M., and McDaniel, P. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017. 1

Gu, S. and Rigazio, L. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014. 1

Hein, M. and Andriushchenko, M. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Proc. of NIPS*, pp. 2266–2276, 2017. 1, 4

Hendrycks, D. and Gimpel, K. Early methods for detecting adversarial images. *arXiv preprint arXiv:1608.00530*, 2016. 1

Jakubovitz, D. and Giryes, R. Improving dnn robustness to adversarial attacks using jacobian regularization. In *Proc. of ECCV*, 2018. 1, 4

Kannan, H., Kurakin, A., and Goodfellow, I. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018. 4.1

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 3, 1, 4

Krotov, D. and Hopfield, J. Dense associative memory is robust to adversarial inputs. *Neural computation*, 30(12): 3151–3167, 2018. 1

Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016. 1, 2, 4.1

LeCun, Y. and Cortes, C. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/. 4

Maaten, L. v. d. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov): 2579–2605, 2008. 1

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017. 1, 2, 2, 1, 1, 4, 4.1, 4.2, 5.1

Mahloujifar, S., Diochnos, D. I., and Mahmoody, M. The curse of concentration in robust learning: Evasion and poisoning attacks from concentration of measure. In *Proc. of AAAI*, 2019. 1

Metzen, J. H., Genewein, T., Fischer, V., and Bischoff, B. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017. 1

Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. Deepfool: a simple and accurate method to fool deep neural networks. In *Proc. of CVPR*, 2016. 1

Najafi, A., Maeda, S.-i., Koyama, M., and Miyato, T. Robustness to adversarial perturbations in learning from incomplete data. In *Proc. of NeurIPS*, pp. 5542–5552, 2019. 2

Papernot, N. and McDaniel, P. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018. 3, 3, 4

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. In *Proc. of EuroS&P*, 2016a. 1, 4.1

Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proc. of S&P*, 2016b. 1

Papernot, N., Faghri, F., Carlini, N., Goodfellow, I., Feinman, R., Kurakin, A., Xie, C., Sharma, Y., Brown, T., Roy, A., Matyasko, A., Behzadan, V., Hambardzumyan, K., Zhang, Z., Juang, Y.-L., Li, Z., Sheatsley, R., Garg, A., Uesato, J., Gierke, W., Dong, Y., Berthelot, D., Hendricks, P., Rauber, J., and Long, R. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018. 4.1

Ross, A. S. and Doshi-Velez, F. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *Proc. of AAAI*, 2018. 1

Schmidt, L., Santurkar, S., Tsipras, D., Talwar, K., and Madry, A. Adversarially robust generalization requires more data. In *Prof. of NeurIPS*, 2018. 1, 2

Shafahi, A., Huang, W. R., Studer, C., Feizi, S., and Goldstein, T. Are adversarial examples inevitable? *arXiv preprint arXiv:1809.02104*, 2018. 1

Stanforth, R., Fawzi, A., Kohli, P., et al. Are labels required for improving adversarial robustness? *arXiv preprint arXiv:1905.13725*, 2019. 2

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *Proc. of ICLR*, 2014. 1

Xie, C., Wu, Y., Maaten, L. v. d., Yuille, A. L., and He, K. Feature denoising for improving adversarial robustness. In *Proc. of CVPR*, 2019. 3, 3, 4, 4.1

Xu, H., Ma, Y., Liu, H., Deb, D., Liu, H., Tang, J., and Jain, A. Adversarial attacks and defenses in images, graphs and text: A review. *arXiv preprint arXiv:1909.08072*, 2019. 2, 2

Yuan, X., He, P., Zhu, Q., and Li, X. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9), 2019. 2, 2